Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής
Εργαστήριο Δικτύων Υπολογιστών

Προσωπικός καταγραφέας δραστηριοτήτων σε Android

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΝΙΚΟΛΑΟΥ ΒΑΛΜΑ

Επιβλέπων: Ευστάθιος Συκάς
Καθηγητής Ε.Μ.Π

Αθήνα, Σεπτέμβριος 2017

Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής
Εργαστήριο Δικτύων Υπολογιστών

Προσωπικός καταγραφέας δραστηριοτήτων σε Android

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΝΙΚΟΛΑΟΥ ΒΑΛΜΑ

Επιβλέπων: Ευστάθιος Συκάς
Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 26η Σεπτεμβρίου 2017

(Υπογραφή)	(Υπογραφή)	(Υπογραφή)
……..	……..	……..
Ευστάθιος Συκάς	Γεώργιος Στασινόπουλος	Ιωάννα Ρουσσάκη
Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π	Επίκουρη Καθηγήτρια Ε.Μ.Π

Αθήνα, Σεπτέμβριος 2017

.....

Νικόλαος Μ. Βαλμάς

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Σκοπός της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη δύο εφαρμογών για την καταγραφή και επεξεργασία δεδομένων κλήσεων και SMS. Συγκεκριμένα περιλαμβάνεται η ανάπτυξη μιας εφαρμογής για smartphones με λειτουργικό Android, η οποία θα τρέχει στο παρασκήνιο και θα αποθηκεύει (τοπικά) πληροφορίες που αφορούν στις «επικοινωνίες» του συνδρομητή (π.χ., εισερχόμενες/εξερχόμενες κλήσεις φωνής, SMSs), πληροφορίες δικτύου (π.χ., Cell-id, LAC, RAT, RSSI) και πληροφορίες θέσης (latitude, longitude). Οι εν λόγω πληροφορίες θα αποστέλλονται, με ασφαλή τρόπο, σε (dedicated) εξυπηρετητή για την αποθήκευση (σε βάση δεδομένων) και επεξεργασίας τους. Τέλος, ο εξυπηρετητής θα περιλαμβάνει γραφικό περιβάλλον μέσω του οποίου ο χρήστης θα έχει την δυνατότητα να ανακτήσει πληροφορίες σχετικά με τις κλήσεις/SMS, να δει σε ποιες τοποθεσίες έκανε/έλαβε κλήσεις/SMSs και να απεικονίζει «επικοινωνίες» από/προς συγκεκριμένους αριθμούς κλήσης.

Λέξεις Κλειδιά: προσωπικός καταγραφέας, εφαρμογή Android, εφαρμογή εξυπηρετητή, βάση δεδομένων, καταγραφή κλήσεων, καταγραφή SMSs, καταγραφή θέσης, καταγραφή κατάστασης δικτύου, αναζήτηση καταγραφών, αναζήτηση στον χάρτη, γράφος επικοινωνιών.

Abstract

The purpose of this thesis is to develop two applications for recording and processing data from smartphones such as calls and SMSs. In particular, it includes the development of an Android-based app that runs in the background and stores (locally) information about the subscriber's "communications" (e.g., incoming / outgoing voice calls, SMSs), network information (E.g., Cell-id, LAC, RAT, RSSI) and user/terminal location information (latitude, longitude). This information is being sent securely to a (dedicated) server for storage and processing. Finally, the server provides a graphical environment through which the user is able to retrieve information about calls / SMSs, see the location that he / she sent / received calls / SMSs and display 'communications' from / to specific phone numbers.

Keywords: personal logger, Android application, server application, database, log calls, log SMSs, log location, log network state, search logs, search on map, communication graph.

Ευχαριστίες

Θα ήθελα σε αυτό το σημείο να ευχαριστήσω τους γονείς και τα αδέρφια μου, χωρίς την υποστήριξη των οποίων δεν θα ήταν δυνατό να φτάσω μέχρι το σημείο αυτό. Επίσης θα ήθελα να ευχαριστήσω όλους τους συμφοιτητές και καθηγητές μου που με βοήθησαν να φτάσω μέχρι εδώ. Τέλος θα ήθελα να ευχαριστήσω τον κ. Γιώργο Λυμπερόπουλο για την βοήθειά του κατά την εκπόνηση της παρούσας εργασίας.

Πίνακας Περιεχομένων

Πίνακας Εικόνων

Πίνακας Δειγμάτων Κώδικα

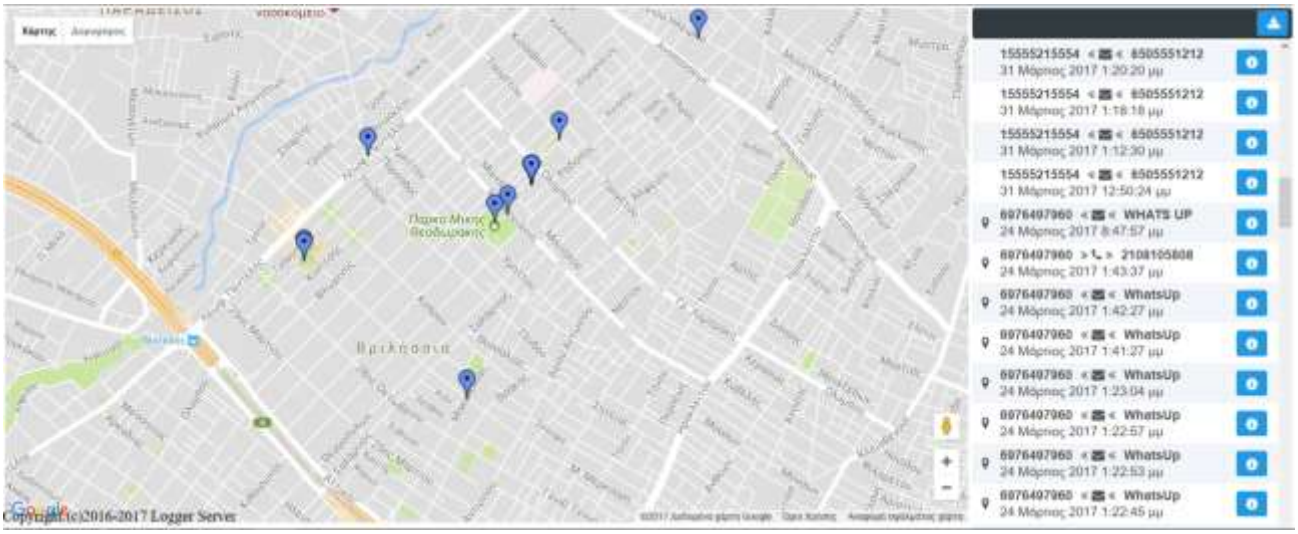
1 Εισαγωγή

1.1 Στόχος της Διπλωματικής εργασίας

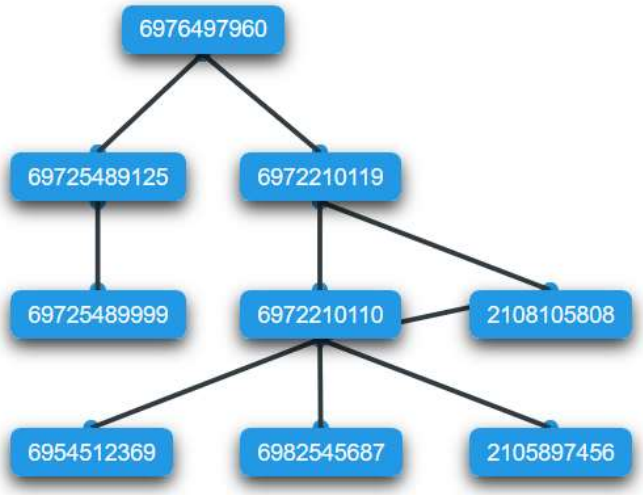
Ο στόχος της διπλωματικής εργασίας είναι η ανάπτυξη ενός προσωπικού καταγραφέα δραστηριοτήτων (κλήσεις/SMSs) που γίνονται από ένα smartphone. Η εφαρμογή θα απευθύνεται σε άτομα ή μικρές ομάδες ατόμων που επιθυμούν να διατηρούν τις δραστηριότητες τους.

Η εφαρμογή θα δίνει την δυνατότητα διατήρησης των δραστηριοτήτων ενός χρήστη για μεγαλύτερο χρονικό διάστημα από αυτό που μπορεί να την διατηρήσει μια συσκευή smartphone καθώς και την συσχέτιση τις κάθε επικοινωνίας που είχε με την θέση που είχε καθώς και την κατάσταση του δικτύου εκείνη την χρονική στιγμή. Επιπλέον η εφαρμογή δεν περιορίζεται στην ατομική χρήση αλλά έχει την δυνατότητα να χρησιμοποιηθεί από περισσότερα άτομα που θέλουν να διατηρούν τις δραστηριότητες σε ένα κοινό σημείο (οικογένεια, φίλοι).

Αναλυτικότερα, θα αποτελείτε από μια εφαρμογή για smartphones με λειτουργικό Android, η οποία θα τρέχει στο background και θα αποθηκεύει (τοπικά) πληροφορίες που αφορούν στις «επικοινωνίες» του συνδρομητή (π.χ., εισερχόμενες/εξερχόμενες κλήσεις φωνής, SMSs), πληροφορίες δικτύου (π.χ., cell-id, LAC, RAT, RSSI) και πληροφορίες θέσης (latitude, longitude). Επιπλέον θα περιλαμβάνει και την ανάπτυξή ενός εξυπηρετητή στον οποίο οι εν λόγω πληροφορίες θα αποστέλλονται για αποθήκευση και επεξεργασία. Τέλος, θα παρέχεται γραφικό περιβάλλον μέσω του οποίου ο χρήστης θα μπορεί να ανακτήσει πληροφορίες σχετικά με τις κλήσεις/SMS, να δει σε ποιες τοποθεσίες έκανε/έλαβε π.χ. κλήσεις (Εικόνα 1) και να απεικονίζει «επικοινωνίες» από/προς συγκεκριμένους αριθμούς κλήσης (Εικόνα 2).



Εικόνα 1: Απεικόνιση σε λίστα και στον χάρτη καταγραφών



Εικόνα 2: Γράφος που απεικονίζει τις άμεσες ή έμμεσες διασυνδέσεις ενός MSISDN

1.2 Δομή της Διπλωματικής εργασίας

Στο κεφάλαιο 2 αναφέρονται οι λειτουργικές απαιτήσεις που θα πρέπει να πληροί η εφαρμογή πελάτη Logger Client και η εφαρμογή Logger Server.

Στο κεφάλαιο 3 αναφερόμαστε στην αρχιτεκτονική του συστήματος που θα υλοποιήσουμε, δηλαδή τα μέρη από τα οποία αυτό αποτελείται. Στο ίδιο κεφάλαιο αναφέρεται και το υλικό / λογισμικό που χρησιμοποιήθηκε για την ανάπτυξη της λύσης.

Στο κεφάλαιο 4 υπάρχει μια σύντομη περιγραφή της τεχνολογίας android. Επιπλέον, στο ίδιο κεφάλαιο περιγράφεται η διαδικασία απόκτησής του κώδικα και εγκατάστασης της εφαρμογής πελάτη Logger Client.

Στο κεφάλαιο 5 παρουσιάζονται οι βασικότερες κλάσεις της εφαρμογής πελάτη Logger Client.

Στο κεφάλαιο 6 περιγράφεται η διαδικασία για την εγκατάσταση του JBoss Application Server που αποτελεί τον Web Server στον οποίο θα τρέχει η εφαρμογή Logger Server.

Στο κεφάλαιο 7 δίνεται μια σύντομη περιγραφή των τεχνολογιών που χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής Logger Server.

Στο κεφάλαιο 8 περιγράφεται η διαδικασία απόκτησής του κώδικα και εγκατάστασης της εφαρμογής Logger Server.

Στο κεφάλαιο 9 παρουσιάζονται οι βασικότερες κλάσεις της εφαρμογής Logger Server.

Στο κεφάλαιο 10 αναφέρονται οι δομές των πινάκων των βάσεων δεδομένων που χρησιμοποιήθηκαν από τις εφαρμογές Logger Client και Logger Server. Στο ίδιο κεφάλαιο περιγράφεται η διαδικασία της εγκατάστασης του MySQL Server που χρησιμοποιείται από την εφαρμογή Logger Server.

Στο κεφάλαιο 11 γίνεται αναφορά σε μέτρα ασφάλειας και το πώς αυτά υλοποιούνται από τις εφαρμογές Logger Client και Logger Server.

Στο κεφάλαιο 12 περιγράφονται σενάρια χρήσης που απεικονίζουν το σύνολο της λειτουργικότητας των εφαρμογών Logger Client και Logger Server.

2 Λειτουργικές απαιτήσεις Client - Server

2.1 Λειτουργικές απαιτήσεις Logger Client

Η συγκεκριμένη εφαρμογή θα καταγράφει δεδομένα από την συσκευή στην οποία εκτελείται και θα τα αποστέλλει στον server

2.1.1 Καταγραφή δεδομένων

Η εφαρμογή θα πρέπει να καταγράφει δεδομένα για κάθε κλήση (εισερχόμενη - εξερχόμενη) και για κάθε μήνυμα για όση διάρκεια η εφαρμογή είναι ενεργοποιημένη. Η εφαρμογή θα καταγράφει τόσο τα στοιχεία της κλήσης/μηνύματος αλλά και τα στοιχεία του δικτύου την συγκεκριμένη χρονική στιγμή.
Παρακάτω απαριθμείται το σύνολο των δεδομένων που πρέπει να καταγράφονται:

- Ο αριθμός τηλεφώνου (MSISDN) της συσκευής
- Ο εξωτερικός αριθμός τηλεφώνου (MSISDN) που κάνει / δέχεται την κλήση / μήνυμα από την συσκευή στην οποία τρέχει η εφαρμογή
- Χρονική στιγμή στην οποία έγινε η κλήση / SMS
- Κατεύθυνση (εισερχόμενη - εξερχόμενη)
- Διάρκεια κλήσης (στην περίπτωση κλήσης)
- Περιεχόμενο μηνύματος (στην περίπτωση SMS)
- Μάρκα / Μοντέλο της συσκευής
- Έκδοση του λογισμικού της συσκευής
- IMEI (International Mobile Equipment Identity) ο κωδικός που περιγράφει μοναδικά την συσκευή
- IMSI (International Mobile Subscriber Identity) ο κωδικός που περιγράφει μοναδικά την κάρτα SIM της συσκευής
- Cell ID ο αριθμός του σταθμού βάσης/κυψέλης (BTS) στον οποίο είναι «συνδεδεμένη» η συσκευή
- LAC (Location area code) ο κωδικός μιας περιοχής που περιλαμβάνει ένα σύνολο από σταθμούς στον οποίο ανήκει ο σταθμός στον οποίο είναι συνδεδεμένη η συσκευή
- RAT (Radio access technology) η τεχνολογία που υποστηρίζει ο συνδεδεμένος σταθμός/τερματικό (EDGE, GSM, UMTS, HSPA, HSPA+, LTE)
- MNC (Mobile network code) – MCC (Mobile country code) για τον εντοπισμό του φορέα του συνδεδεμένου δικτύου
- RSSI (Received signal strength indication) η μέτρηση που αντιπροσωπεύει την ισχύ του λαμβανόμενου σήματος

Επιπλέον, καταγράφονται οι παρακάτω πληροφορίες στην περίπτωση που η συσκευή είναι συνδεδεμένη σε 4G/LTE δίκτυο:

- RSRP (Reference Signal Received Power)
- RSRQ (Reference Signal Received Quality)
- RS SNR (Reference Signal Signal to Noise Ratio)
- CQI (Channel Quality Indicator)

Τέλος, αν η συσκευή έχει ενεργοποιημένη την υπηρεσία εντοπισμού θέσης (μέσω GPS, WLAN, mobile network) τότε καταγράφεται και πληροφορία θέσης:

- Longitude το γεωγραφικό μήκος
- Latitude το γεωγραφικό πλάτος

Η καταγραφή γίνεται από μια «υπηρεσία» (service) που εκτελείται στο παρασκήνιο και καταγράφει τις παραπάνω πληροφορίες ακόμα και όταν η εφαρμογή έχει κλείσει/τερματίσει. Η εκκίνηση και τερματισμός τις συγκεκριμένης υπηρεσίας γίνεται μέσα από την εφαρμογή.

2.1.2 Εμφάνιση της καταγεγραμμένης πληροφορίας στον χρήστη

Η εφαρμογή εμφανίζει στην αρχική οθόνη την πληροφορία που σχετίζεται με την συσκευή:

- Ο αριθμός τηλεφώνου (MSISDN) της συσκευής
- Μάρκα / Μοντέλο της συσκευής
- Έκδοση του λογισμικού της συσκευής
- IMEI (International Mobile Equipment Identity) ο κωδικός που περιγράφει μοναδικά την συσκευή
- IMSI (International Mobile Subscriber Identity) ο κωδικός που περιγράφει μοναδικά την κάρτα SIM της συσκευής

Επιπλέον εμφανίζει το κουμπί για την εκκίνηση η τερματισμό της υπηρεσίας καταγραφής και τέλος των αριθμό των κλήσεων και των SMS που έχουν καταγραφεί. Ο χρήστης έχει την δυνατότητα να δει την καταγεγραμμένη πληροφορία για οποιαδήποτε από τις κλήσεις/SMS που έχουν καταγραφεί. Τέλος, αν υπάρχει διαθέσιμη πληροφορία θέσης για κάποια καταγραφή τότε αυτή εμφανίζεται μέσω του χάρτη στον χρήστη.

2.1.3 Αποστολή δεδομένων

Η εφαρμογή αποστέλλει την καταγεγραμμένη πληροφορία στον Server όταν υπάρχει δυνατότητα χρήσης του internet από την συσκευή. Η πληροφορία που στέλνεται περιλαμβάνει όλη την παραπάνω πληροφορία που καταγράφηκε.

Στην περίπτωση κλήσης: την στιγμή που ξεκινάει η κλήση στέλνεται όλη η πληροφορία εκτός από την τοποθεσία και την διάρκεια κλήσης. Η τοποθεσία στέλνεται την στιγμή που γίνεται διαθέσιμη. Όταν η κλήση τερματίσει στέλνεται ο συνολικός χρόνος κλήσης.
Σε περίπτωση μηνύματος: την στιγμή που ξεκινάει η κλήση στέλνεται όλη η πληροφορία εκτός από την τοποθεσία. Η τοποθεσία στέλνεται τη στιγμή που γίνεται διαθέσιμη.
Με την παραπάνω μέθοδο δεν καθυστερείται να σταλεί η καταγεγραμμένη πληροφορία από την προσπάθεια απόκτησης της τοποθεσίας από την συσκευή που μπορεί να καθυστερήσει αισθητά όταν για παράδειγμα η συσκευή βρίσκεται σε σημείο που δεν μπορεί να λάβει σήμα από το GPS (εσωτερικό κτηρίου, μετρό). Επιπλέον, στην περίπτωση κλήσης η καταγεγραμμένη πληροφορία δεν θα πρέπει να αναμένει την λήξη της κλήσης για να σταλεί στον server καθώς μια κλήση είναι δυνατόν να διαρκέσει από λίγα λεπτά έως μερικές ώρες.
Στην περίπτωση που δεν είναι δυνατή η χρήση του internet από την εφαρμογή η καταγεγραμμένη πληροφορία που δεν έχει σταλεί στον server διατηρείται από την εφαρμογή μέχρι να υπάρξει δυνατότητα χρήσης του internet. Η εφαρμογή ενημερώνεται για το πότε υπάρχει δυνατότητα χρήσης του internet.
Στην περίπτωση που δεν είναι δυνατή η επικοινωνία με τον server (για παράδειγμα λόγω συντήρησης) η καταγεγραμμένη πληροφορία που απέτυχε να σταλεί στον server διατηρείται από την εφαρμογή.
Στην περίπτωση που η υπηρεσία καταγραφής τερματιστεί και υπάρχει καταγεγραμμένη πληροφορία που δεν έχει σταλεί στον server αποθηκεύεται τοπικά έτσι ώστε να μπορεί να ανακτηθεί και να αποσταλεί την επομένη φορά που θα ξεκινήσει η υπηρεσία.

2.1.4 Ρυθμίσεις εφαρμογής

Η εφαρμογή δίνει στον χρήστη τις παρακάτω επιλογές:

- Να επιλέγει αν η υπηρεσία καταγραφής των δεδομένων θα εκκινείτε κατά την ενεργοποίηση του κινητού.
- Να επιλέγει αν η οθόνη του κινητού θα παραμένει αναμμένη όταν είναι ανοιχτή η εφαρμογή.
- Να επιλέγει με ποίον τρόπο θα αποστέλλεται η καταγεγραμμένη πληροφορία στον server. Συγκεκριμένα ο χρήστης πρέπει να έχει την δυνατότητα να ενεργοποιήσει μια από τις παρακάτω τρεις επιλογές:
    1. Χρήση WIFI ή δεδομένων δικτύου κινητής τηλεφωνίας
    2. Χρήση μόνο WIFI
    3. Χρήση μόνο δεδομένων δικτύου κινητής τηλεφωνίας
- Να ενημερώνει τον χρήστη ότι μια καινούργια έκδοση της εφαρμογής είναι διαθέσιμη καθώς και για τις αλλαγές που υλοποιεί η νέα έκδοση. Η εφαρμογή θα πρέπει να κατεβάζει και να εγκαθιστά την καινούργια έκδοση κατ’ επιλογή του χρήστη.

2.2 Λειτουργικές απαιτήσεις Logger Server

Ο server θα λαμβάνει και θα αποθηκεύει τα δεδομένα από όλες τις συσκευές στις οποίες εκτελείται η εφαρμογή πελάτη Logger. Επιπλέον θα δίνει την δυνατότητα αναζήτησης και απεικόνισης της συλλεγμένης πληροφορίας.

2.2.1 Λήψη και αποθήκευση της πληροφορίας

Ο server θα λαμβάνει και θα αποθηκεύει την παρακάτω πληροφορία στην βάση δεδομένων του:

- Ο αριθμός τηλεφώνου (MSISDN) της συσκευής που κατέγραψε την πληροφορία
- Ο εξωτερικός αριθμός τηλεφώνου (MSISDN) που κάνει / δέχεται την κλήση / μήνυμα από την συσκευή στην οποία τρέχει η εφαρμογή
- Χρονική στιγμή στην οποία έγινε η κλήση / SMS
- Κατεύθυνση (εισερχόμενη - εξερχόμενη)
- Διάρκεια κλήσης (στην περίπτωση κλήσης)
- Περιεχόμενο μηνύματος (στην περίπτωση SMS)
- Μάρκα / Μοντέλο της συσκευής
- Έκδοση του λογισμικού της συσκευής
- IMEI (International Mobile Equipment Identity) ο κωδικός που περιγράφει μοναδικά την συσκευή
- IMSI (International Mobile Subscriber Identity) ο κωδικός που περιγράφει μοναδικά την κάρτα SIM της συσκευής
- Cell ID ο αριθμός του σταθμού βάσης/κυψέλης (BTS) στον οποίο είναι «συνδεδεμένη» η συσκευή
- LAC (Location area code) ο κωδικός μιας περιοχής που περιλαμβάνει ένα σύνολο από σταθμούς στον οποίο ανήκει ο σταθμός στον οποίο είναι συνδεδεμένη η συσκευή
- RAT (Radio access technology) η τεχνολογία που υποστηρίζει ο συνδεδεμένος σταθμός/τερματικό (EDGE, GSM, UMTS, HSPA, HSPA+, LTE)
- MNC (Mobile network code) – MCC (Mobile country code) για τον εντοπισμό του φορέα του συνδεδεμένου δικτύου
- RSSI (Received signal strength indication) ή μέτρηση που αντιπροσωπεύει την ισχύ του λαμβανόμενου σήματος
- RSRP (Reference Signal Received Power)
- RSRQ (Reference Signal Received Quality)
- RS SNR (Reference Signal Signal to Noise Ratio)
- CQI (Channel Quality Indicator)
- Longitude το γεωγραφικό μήκος
- Latitude το γεωγραφικό πλάτος
- Ο τύπος της καταγραφής (κλήση - SMS)

- Η ακτίνα (σε μέτρα) της ακρίβειας της προσέγγισης υπολογισμού της τοποθεσίας που έγινε η καταγραφή. Αν η τοποθεσία έχει καταγραφεί από την εφαρμογή πελάτη τότε η ακτίνα παίρνει την τιμή 0.

2.2.2 Υπολογισμός τοποθεσίας με χρήση του Mozilla Location Service

Στην περίπτωση που κατά την καταγραφή της κλήσης / SMS από την εφαρμογή πελάτη δεν είναι διαθέσιμη η τοποθεσία, κατά την λήψη της πληροφορίας ο server θα προσπαθήσει να υπολογίσει μια προσέγγιση της τοποθεσίας κάνοντας χρήση του Mozilla Location Service API. Συγκεκριμένα θα στείλει την παρακάτω πληροφορία:

- LAC (Location area code) ο κωδικός μιας περιοχής που περιλαμβάνει ένα σύνολο από σταθμούς στον οποίο ανήκει ο σταθμός στον οποίο είναι συνδεδεμένη η συσκευή
- MNC (Mobile network code) – MCC (Mobile country code) για τον εντοπισμό του φορέα του συνδεδεμένου δικτύου
- RAT (Radio access technology) η τεχνολογία που υποστηρίζει ο συνδεδεμένος σταθμός
- RSSI (Received signal strength indication) ή μέτρηση που αντιπροσωπεύει την ισχύ του λαμβανόμενου σήματος
- Cell ID ο αριθμός ένα σταθμού (BTS) στον οποίο είναι η συσκευή συνδεδεμένη

Το MLS API θα επιστρέψει το γεωγραφικό μήκος και πλάτος καθώς και μια ακτίνα (σε μέτρα) που αντιπροσωπεύει την ακρίβεια της μέτρησης.

2.2.3 Παροχή υπηρεσίας για την ενημέρωση της εφαρμογής πελάτη

Ο server θα πρέπει να ενημερώνει την εφαρμογή πελάτη για ενδεχόμενη νέα έκδοση της καθώς και να της την στέλνει. Συγκεκριμένα χρειάζεται μια υπηρεσία που θα στέλνει το όνομα της νέας έκδοσης καθώς και μια σύντομη περιγραφή των αλλαγών που υλοποιεί η νέα έκδοση στην εφαρμογή πελάτη. Η εφαρμογή πελάτη θα την συγκρίνει με την υπάρχουσα έκδοση και στην περίπτωση που δεν είναι η ίδια θα έχει την δυνατότητα να την κατεβάσει από τον server μέσω μιας δεύτερης υπηρεσίας

2.2.4 Είσοδος στην εφαρμογή

Για την είσοδο του χρήστη στην εφαρμογή θα απαιτείται η εισαγωγή ονόματος χρηστή και κωδικού. Η εισαγωγή νέων χρηστών ή η αλλαγή των στοιχείων θα γίνεται από την βάση δεδομένων.

2.2.5 Μενού εφαρμογής

Μέσα από το μενού ο χρήστης θα έχει την δυνατότητα να διαλέξει μια από τις παρακάτω λειτουργικότητες

- Αρχική σελίδα
- Αναζήτηση στον χάρτη
- Αναζήτηση μονοπατιού
- Εύρεση σχέσης

2.2.6 Αρχική σελίδα

Στην Αρχική σελίδα της εφαρμογής θα εμφανίζονται οι 15 πιο πρόσφατες καταγραφές ταξινομημένες κατά ημερομηνία σε φθίνουσα σειρά. Ο χρήστης θα έχει την δυνατότητα να δει όλη την καταγεγραμμένη πληροφορία για κάθε εγγραφή καθώς και την τοποθεσίας της στον χάρτη εφόσον υπάρχει για την συγκεκριμένη εγγραφή.

2.2.7 Αναζήτηση στον χάρτη

Στην Αναζήτηση στον χάρτη ο χρήστης θα έχει την δυνατότητα να αναζητήσει εγγραφές καταχωρημένες στην βάση δεδομένων χρησιμοποιώντας τα παρακάτω κριτήρια:

- Αριθμός τηλεφώνου
- Εξωτερικός αριθμός τηλεφώνου
- Κατεύθυνση (Εισερχόμενη, Εξερχόμενη)
- Ημερομηνία από
- Ημερομηνία μέχρι
- Τύπος εγγραφής (Κλήση, Μήνυμα)
- Κέντρο γεωγραφικού μήκους
- Κέντρο γεωγραφικού πλάτους
- Ακτίνα
- Περιεχόμενο μηνύματος

Στα αποτελέσματα της αναζήτησης ο χρήστης θα μπορεί να δει όλη την καταγεγραμμένη πληροφορία για κάθε εγγραφή καθώς και την τοποθεσίας της στον χάρτη εφόσον υπάρχει για την συγκεκριμένη εγγραφή. Στην περίπτωση που η τοποθεσία υπολογίστηκε κατά προσέγγιση θα απεικονίζεται ως κύκλος με κέντρο το γεωγραφικό μήκος και το γεωγραφικό πλάτος που υπολογίστηκαν και ακτίνα την ακρίβεια της προσέγγισης.

2.2.8 Εξαγωγή αποτελεσμάτων σε αρχείο Excel

Τα αποτελέσματα της αναζήτησης στον χάρτη ο χρήστης θα έχει την δυνατότητα να τα εξάγει σε ένα αρχείο (τύπου Excel .xls) που θα αποθηκευτεί στον τοπικό του δίσκο. Το αρχείο θα περιέχει ένα πίνακα στον οποίο κάθε γραμμή θα περιλαμβάνει μια εγγραφή με όλη την καταγεγραμμένη πληροφορία.

2.2.9 Αναζήτηση μονοπατιού

Η αναζήτηση μονοπατιού θα δίνει στον χρήστη την δυνατότητα να δει την διαδρομή μιας συσκευής με ένα συγκεκριμένο αριθμό τηλεφώνου. Ειδικότερα η συγκεκριμένη λειτουργικότητα απεικονίζει στον χάρτη εγγραφές (των οποίων έχει καταγραφεί η τοποθεσία) που ελήφθησαν από ένα συγκεκριμένο αριθμό τηλεφώνου μέσα σε ένα χρονικό διάστημα και τις συνδέει με χρονική σειρά. Η αναζήτηση γίνεται βάσει των παρακάτω κριτηρίων:

- Αριθμός τηλεφώνου (ο χρήστης μπορεί να βάλει ως και 5 αριθμούς τηλεφώνου και να δει τις διαδρομές αυτών ταυτόχρονα στον χάρτη. Για την αναζήτηση απαιτείται να βάλει τουλάχιστον ένα)
- Ημερομηνία από (υποχρεωτικό πεδίο)
- Ημερομηνία μέχρι

Στα αποτελέσματα της αναζήτησης ο χρήστης θα μπορεί να δει όλη την καταγεγραμμένη πληροφορία για κάθε εγγραφή

2.2.10 Εύρεση σχέσης

Η εύρεση σχέσης δίνει στον χρήστη την δυνατότητα να δει το σύνολο των αριθμών τηλεφώνων με το οποίο είχε επικοινωνία ένας συγκεκριμένος αριθμός τηλεφώνου σε ένα χρονικό διάστημα βάση των καταγεγραμμένων εγγραφών. Το σύνολο αυτό περιλαμβάνει και αριθμούς που υπήρχε επικοινωνία μέσω τρίτων αριθμών τηλεφώνου. Συγκεκριμένα τα αποτελέσματα απεικονίζονται με ένα γράφο με ρίζα τον αριθμό τηλεφώνου που αναζητείται και κορυφές τους αριθμούς τηλεφώνου με τους οποίους υπήρχε άμεσα ή έμμεσα επικοινωνία. Οι ακμές στον γράφο απεικονίζουν την άμεση επικοινωνία μεταξύ δύο κορυφών. Η αναζήτηση γίνεται βάση των παρακάτω κριτηρίων:

- Αριθμός τηλεφώνου (υποχρεωτικό πεδίο)
- Αριθμός τηλεφώνου στόχου
- Ημερομηνία από
- Ημερομηνία μέχρι

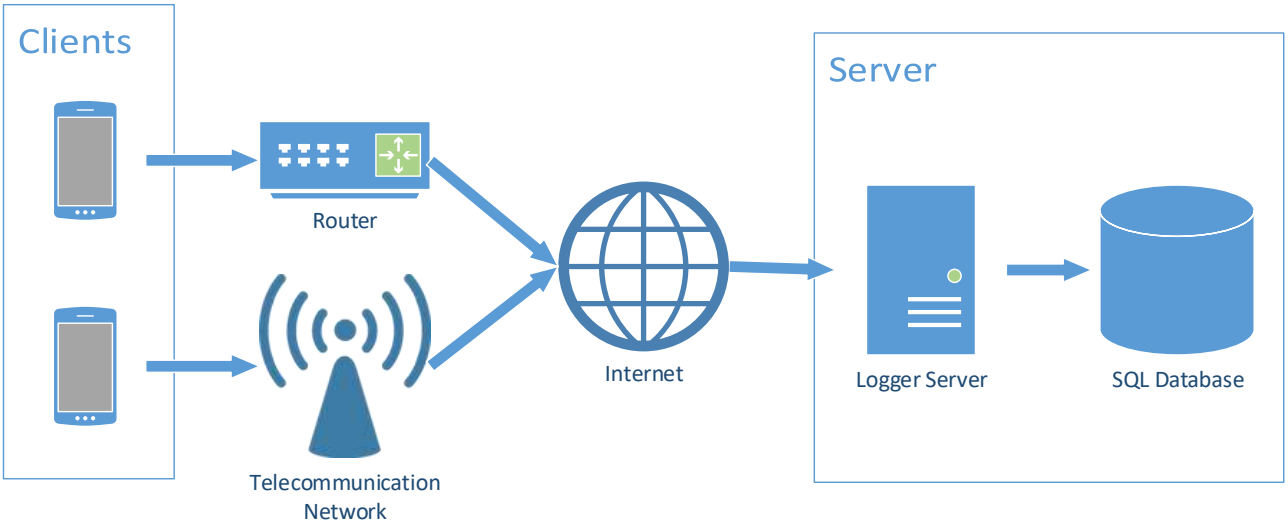
Στην περίπτωση που ο χρήστης εισάγει επιπλέον τον «Αριθμό τηλεφώνου στόχου» ο γράφος θα σχεδιαστεί αν υπάρχει σχέση (άμεση ή έμμεση) μεταξύ των δύο αριθμών διαφορετικά ο χρήστης θα ειδοποιηθεί ότι δεν υπάρχει κάποια σχέση μεταξύ των δύο αριθμών.

3 Σχεδιασμός

3.1 Γενικό αρχιτεκτονικό μοντέλο

Η λύση που αναπτύξαμε, βασίζεται στην αρχιτεκτονική τύπου client-server. Αυτό είναι απαραίτητο από την στιγμή που η εφαρμογή πελάτη θα είναι εγκατεστημένη σε μια πληθώρα συσκευών οι οποίες θα συγκεντρώνουν και θα αποστέλλουν δεδομένα σε έναν εξυπηρετητή (server) για την επεξεργασία και θα τα αποθήκευσή τους.
Για την υλοποίηση του server θα χρειαστεί ένα υπολογιστικό σύστημα με πρόσβαση στο internet. Στο συγκεκριμένο υπολογιστικό σύστημα θα πρέπει να είναι εγκατεστημένο κάποιο λειτουργικό σύστημα τύπου Windows ή GNU/Linux. Επιπλέον, απαιτείται η ύπαρξη ενός HTTP server στον οποίο θα τρέχει η εφαρμογή μας καθώς και ενός SQL server ο οποίος θα αποθηκεύει τα δεδομένα.
Η εφαρμογή πελάτη θα μπορεί τρέχει σε μια οποιαδήποτε συσκευή android (smartphones, tablets). με SDK 19 (Android 4.4) και πάνω, με πρόσβαση στο δίκτυο κινητής τηλεφωνίας καθώς και στο internet.

Στην Εικόνα 3 βλέπουμε μια σχηματική απεικόνιση της αρχιτεκτονικής μας λύσης:



Εικόνα 3: Αρχιτεκτονικό μοντέλο της εφαρμογής Logger Client - Server

3.2 Το λογισμικό και το υλικό που χρησιμοποιήθηκε

3.2.1 HTTP Server και εξωτερικές βιβλιοθήκες

Χρησιμοποιήθηκε ο **JBoss AS 7.1.1.Final,** ο όποιος βασίζεται σε μια εξαιρετικά ελαφριά αρχιτεκτονική και αποτελεί μια πιστοποιημένη υλοποίηση της Java Enterprise Edition 6, στον οποίο θα τρέχει η εφαρμογή Logger Server που υλοποιήσαμε. Επιπλέον χρησιμοποιήθηκαν οι παρακάτω βιβλιοθήκες για την κάλυψη των λειτουργικών απαιτήσεων της εφαρμογής:

- spring-core (org.springframework) 4.1.4.RELEASE
- spring-beans (org.springframework) 4.1.4.RELEASE
- com.sun.faces.jsf 2.1.25
- servlet.api 2.5
- servlet.api.jstl 1.2
- mysql-connector-java 5.1.39
- javaee-api 6.0
- apache-poi 3.15
- cdi-api (javax.enterprise) 1.2
- log4j 1.2.16
- primefaces 6.0
- gson (com.google.code.gson) 2.7
- hibernate-entitymanager 4.0.1.Final
- httpclient (org.apache.httpcomponents) 4.5.3
- junit 4.12

3.2.2 Βάσεις δεδομένων

Για την αποθήκευση και αποδοτική ανάκτηση της καταγεγραμμένης πληροφορίας χρησιμοποιήθηκε ο MySQL Server Version 14.14 Distribution 5.5.53 για την εφαρμογή Logger Server, ενώ χρησιμοποιήθηκε η SQLite για την εφαρμογή Logger Client.

3.2.3 Ολοκληρωμένα Περιβάλλοντα ανάπτυξης

Για την παραγωγή του κώδικα της εφαρμογής Logger Server χρησιμοποιήθηκε το Red Hat JBoss Developer Studio Version: 10.3.0. Για την παραγωγή του κώδικα της εφαρμογής πελάτη Logger χρησιμοποιήθηκε το Android Studio Version: 2.3.1.

3.2.4 Λογισμικό / υλικό για την δοκιμή της εφαρμογής πελάτη Logger

Χρησιμοποιήθηκαν δύο android emulators (Nexus 5X API 25 , Galaxy Nexus API 23) καθώς και μια εμπορική συσκευή android smartphone (Lenovo a-7000 API 23) για την δοκιμή και επιβεβαίωση της σωστής λειτουργίας της εφαρμογής πελάτη.

3.2.5 Λογισμικό / υλικό για την δοκιμή της εφαρμογής Logger Server

Χρησιμοποιήθηκε ένα πραγματικό μηχάνημα με λειτουργικό Windows 10 64 bit καθώς και ένα εικονικό μηχάνημα στον ωκεανό (1) με λειτουργικό Debian GNU/Linux 8 (jessie) 64 bit για την δοκιμή και επιβεβαίωση της σωστής λειτουργίας της εφαρμογής Logger Server.

4 Εφαρμογές Android και εγκατάσταση της εφαρμογής πελάτη

4.1 Βασικές παρατηρήσεις για τον προγραμματισμό σε Android

Για μια εφαρμογή Android, ιδιαίτερο ρόλο έχουν οι δραστηριότητες (activities) και οι υπηρεσίες (services). Οι δραστηριότητες χρησιμοποιούνται για την υλοποίηση λειτουργικότητας που απαιτούν την αλληλεπίδραση του χρήστη με την συσκευή, ενώ οι υπηρεσίες για εργασίες που πρέπει να εκτελούνται στο παρασκήνιο, ακόμα και όταν ο χρήστης δεν αλληλεπιδρά με την εφαρμογή.

Τόσο οι δραστηριότητες όσο και οι υπηρεσίες αποτελούνται από ειδικές κλάσεις που υλοποιούν συγκεκριμένες διεπαφές (interfaces) από την υλοποίηση των οποίων αποκτούν συγκεκριμένες συμπεριφορές. Για παράδειγμα, η συμπεριφορά μιας δραστηριότητας όταν εκκινεί, σταματάει προσωρινά, επανέρχεται ή τερματίζει μπορεί να προγραμματιστεί χρησιμοποιώντας τις μεθόδους που παρέχει η διεπαφή της. Κατά παρόμοιο τρόπο λειτουργούν και οι υπηρεσίες. Μια υπηρεσία μπορεί να προγραμματιστεί ώστε να εκτελέσει συγκεκριμένο κώδικα κατά την δημιουργία, εκκίνηση και τερματισμό της.

Η επικοινωνία μεταξύ υπηρεσιών και δραστηριοτήτων επιτυγχάνεται με μια ειδική διεπαφή, τον BroadcastReceiver, ο οποίος λαμβάνει προθέσεις (intents) που περιέχουν πληροφορίες από την δραστηριότητα/υπηρεσία που τα έστειλε και τα προωθεί στην δραστηριότητα/υπηρεσία στην οποία είναι συνδεδεμένος.

Όσον αφορά την αλληλεπίδραση του χρήστη με την εφαρμογή υπάρχουν οι στατικές ή οι δυναμικές διατάξεις (layouts). Κάθε τέτοια διάταξη περιγράφεται από ένα σύνολο αρχείων xml τα οποία ρυθμίσουν τα στατικά στοιχεία της διάταξης καθώς και από μια δραστηριότητα η οποία ρυθμίζει τα δυναμικά της στοιχεία. Επιπλέον η δραστηριότητα είναι υπεύθυνη για την αλληλεπίδραση του χρήστη με την διάταξη, συνεπώς, κάθε click, scroll, pinch του χρήστη λαμβάνεται από την δραστηριότητα η οποία μπορεί να το αγνοήσει ή να εκτελέσει συγκεκριμένο κώδικα. Μια ακόμη χρήσιμη δυνατότητα αλληλεπίδρασης παρέχει το γραφικό στοιχείο Toast το οποίο ενημερώνει τον χρήστη για κάποιο γεγονός. Ένα πιο περίπλοκο γραφικό στοιχείο ενημέρωσης είναι οι υλοποιήσεις της κλάσης Dialog οι οποίες χρησιμοποιούνται όταν ένα γεγονός απαιτεί κάποια αλληλεπίδραση από τον χρήστη. Για παράδειγμα το πάτημα ενός κουμπιού από τον χρήστη εκκινεί μια υπηρεσία η οποία απαιτεί την παροχή συγκεκριμένων αδειών από τον χρήστη για να ξεκινήσει. Στην περίπτωση αυτήν ένα παράθυρο Dialog εμφανίζεται στον χρήστη και ζητώντας του να παραχωρήσει (αν επιθυμεί) συγκεκριμένες άδειες (permissions) στην εφαρμογή.

Οι υπηρεσίες εκκινούνται είτε από αλληλεπίδραση του χρήστη είτε μετά από ένα γεγονός (π.χ. εκκίνηση της συσκευής), εκτελούνται στο παρασκήνιο ακόμα και όταν ο χρήστης έχει κλείσει την εφαρμογή και τερματίζουν από αλληλεπίδραση του χρήστη είτε μετά από ένα γεγονός. Η εφαρμογή που υλοποιήσαμε

βασίζεται σε μια υπηρεσία η οποία καταγράφει πληροφορίες όταν η συσκευή στέλνει / λαμβάνει μια κλήση / μήνυμα.

Μια android εφαρμογή είναι πιθανό να χρειαστεί πρόσβαση σε πληροφορία ή υπηρεσία που διαθέτει η συσκευή (π.χ. πρόσβαση σε συγκεκριμένους φακέλους, πρόσβαση στο GPS, πρόσβαση στην κατάσταση του δικτύου). Αρκετές από αυτές μπορεί να δώσουν την δυνατότητα σε μια κακόβουλη εφαρμογή να βλάψει την συσκευή ή/και τον χρήστη. Για αυτό τον λόγο πρέπει να χορηγούνται άδειες από τον χρήστη. Από το SDK 23 (Android 6.0) οι εφαρμογές που χρειάζονται συγκεκριμένες «επικίνδυνες» άδειες ζητούν από τον χρήστη να τους τις χορηγήσει κατά την ώρα λειτουργίας της εφαρμογής. Αν ο χρήστης δεν θέλει να χορηγήσει κάποια από αυτές τότε η εφαρμογή πρέπει να απενεργοποιήσει την λειτουργικότητα που απαιτεί την συγκεκριμένη άδεια.

Στο αρχείο AndroidManifest.xml βρίσκονται όλες οι δραστηριότητες και οι υπηρεσίες της εφαρμογής καθώς και το σύνολο των αδειών που απαιτεί η εφαρμογή. Στο Δείγμα Κώδικα 1 φαίνεται το AndroidManifest.xml της εφαρμογής μας.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.ntua.ote.logger">

    <!-- Permissions -->
    <uses-permission android:name="android.permission.READ_CALL_LOG" />
    <uses-permission android:name="android.permission.READ_SMS" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />

    <!-- Application configurations, Activities and Services -->
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:theme="@style/AppTheme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".SettingsActivity"
            android:label="@string/action_settings"
            android:parentActivityName=".MainActivity"
            android:theme="@style/AppTheme.NoActionBar">
```

```
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value=".MainActivity" />
        </activity>
        <activity
            android:name=".CallsViewActivity"
            android:label="@string/calls_view_label"
            android:parentActivityName=".MainActivity"
            android:theme="@style/AppTheme.NoActionBar"
            android:launchMode="singleTop">
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value=".MainActivity" />
        </activity>
        <activity
            android:name=".ViewEntryActivity"
            android:label="@string/view_entry"
            android:parentActivityName=".CallsViewActivity"
            android:theme="@style/AppTheme.NoActionBar"
            android:launchMode="singleTop">
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value=".CallsViewActivity" />
        </activity>

        <service
            android:name=".CallLogService"
            android:exported="false" />

        <receiver android:name=".BootEventReceiver">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
            </intent-filter>
        </receiver>

        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="@string/google_maps_key" />

        <activity
            android:name=".MapActivity"
            android:label="@string/title_activity_map"
            android:theme="@style/AppTheme.NoActionBar">
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="com.ntua.ote.logger.ViewEntryActivity" />
        </activity>
    </application>

</manifest>
```

Δείγμα Κώδικα 1: AndroidManifest.xml της εφαρμογής Logger Client

4.2 Χρήση του Google Maps Android API

Το Google Maps Android API δίνει την δυνατότητα χρήσης των Google Maps και απεικόνισης της θέσης για εφαρμογές Android. Συγκεκριμένα, ο προγραμματιστής μπορεί να απεικονίσει θέσεις στον χάρτη (Markers,

κύκλους), να δημιουργήσει διαδρομές κλπ. Περισσότερες πληροφορίες σχετικά με την χρήση του API είναι διαθέσιμες στην ιστοσελίδα (2).
Για την χρήση του API απαιτείται ένα κλειδί που παρέχει η Google έχοντας δημιουργήσει ένα λογαριασμό στο (3). Το συγκεκριμένο κλειδί το δηλώνουμε στο AndroidManifest.xml όπως φαίνεται στο Δείγμα Κώδικα 1.
Στην εφαρμογή μας χρησιμοποιήσαμε το API για να απεικονίσουμε την θέση των καταγραφών στον χάρτη. Παρακάτω φαίνεται ο κώδικας της εφαρμογής μας για την συγκεκριμένη λειτουργικότητα. Στο Δείγμα Κώδικα 2 φαίνεται η κλάση MapActivity. Στην μέθοδο onCreate (που καλείται όταν δημιουργείται η δραστηριότητα) δηλώνεται με πιο layout σχετίζεται η συγκεκριμένη δραστηριότητα, ξεκινάει να φορτώνεται ο χάρτης ο οποίος θα τοποθετηθεί στην θέση του fragment το layout που φαίνεται στο Δείγμα Κώδικα 3 και τέλος αποκτάται η θέση της καταγραφής. Η MapActivity υλοποιεί την διεπαφή OnMapReadyCallback και την μέθοδο της onMapReady η οποία θα εκτελεστεί την στιγμή που θα φορτώσει ο χάρτης. Η συγκεκριμένη μέθοδος δημιουργεί έναν Marker και τον τοποθετεί στην θέση της καταγραφής.

```
package com.ntua.ote.logger;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;
import com.ntua.ote.logger.utils.Constants;

public class MapActivity extends AppCompatActivity implements OnMapReadyCallback {

    private double latitude;
    private double longitude;
    private String title;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_map);
        // Obtain the SupportMapFragment and get notified when the map is ready to be used.
        SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
                .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);

        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);

        Bundle b = getIntent().getExtras();
        latitude = b == null ? -1 : b.getDouble(Constants.LATITUDE_KEY);
        longitude = b == null ? -1 : b.getDouble(Constants.LONGITUDE_KEY);
        title = b == null ? "" : b.getString(Constants.EXTERNAL_NUMBER_KEY);
    }

    @Override
```

```
    public void onMapReady(GoogleMap googleMap) {
        if(latitude != -1 && longitude != -1) {
            LatLng location = new LatLng(latitude, longitude);
            googleMap.addMarker(new MarkerOptions().position(location).title(title));
            googleMap.moveCamera(CameraUpdateFactory.newLatLng(location));
            googleMap.animateCamera(CameraUpdateFactory.zoomTo(15));
        }
    }
}
```

Δείγμα Κώδικα 2: Η κλάση com.ntua.ote.logger.MapActivity

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:context="com.ntua.ote.logger.MapActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >

        <android.support.design.widget.AppBarLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:theme="@style/AppTheme.AppBarOverlay">

            <android.support.v7.widget.Toolbar
                android:id="@+id/toolbar"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
                android:background="?attr/colorPrimary"
                app:popupTheme="@style/AppTheme.PopupOverlay" />

        </android.support.design.widget.AppBarLayout>

        <fragment
            android:id="@+id/map"
            android:name="com.google.android.gms.maps.SupportMapFragment"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            tools:context="com.ntua.ote.logger.MapActivity" />

    </LinearLayout>
</android.support.design.widget.CoordinatorLayout>
```

Δείγμα Κώδικα 3: Το layout activity_map.xml

4.3 Εγκατάσταση και εκτέλεση της εφαρμογής πελάτη Logger Client

Ο κώδικας της εφαρμογής βρίσκεται στο GitHub. Για να κατεβάσουμε την εφαρμογή απαιτείται ένας git client που μπορεί να αποκτηθεί από την διεύθυνση (4) ή στην περίπτωση GNU/Linux και μέσω της εντολής **apt-get install git**. Έχοντας κατεβάσει έναν git client κατεβάζουμε τον κώδικα από το (5). Αφού κατέβει ο κώδικας θα πρέπει να κατεβάσουμε το Android Studio από την ιστοσελίδα της Google (6). Η εφαρμογή μας έχει αναπτυχθεί με το SDK 25 συνεπώς κατά την εγκατάσταση του Android Studio πρέπει να το επιλέξουμε για να εγκατασταθεί. Στην ιστοσελίδα (6) υπάρχει ένα βίντεο που δείχνει τα βήματα που χρειάζονται για την εγκατάσταση του Android Studio

Μετά την εγκατάσταση, μπορούμε να εισάγουμε τον κώδικα που κατεβάσαμε από το μενού import existing project. Για να το εκτελέσουμε πατάμε το κουμπί Run και επιλέγουμε είτε μια εικονική συσκευή είτε μια συσκευή που έχουμε συνδέσει. Ο κώδικας θα μεταγλωττιστεί και θα εγκατασταθεί στην επιλεγμένη συσκευή. Στο κάτω μέρος του Android Studio στην καρτέλα Android Monitor εμφανίζεται η καταγραφή πληροφοριών για γεγονότα και προβλήματα που προκύπτουν στην εφαρμογή καθώς αυτή εκτελείται.

Ο κώδικας της εφαρμογής μεταγλωττίζεται και πακετάρεται σε ένα αρχείο APK με την βοήθεια του εργαλείου gradle (7). Στον αρχείο build.gradle υπάρχουν οι ρυθμίσεις που απαιτούνται για να χτιστεί η εφαρμογή. Συγκεκριμένα η έκδοση του SDK που θα χρησιμοποιηθεί για να μεταγλωττιστεί ο κώδικας, οι εκδόσεις Android για τις οποίες είναι συμβατή η εφαρμογή μας και οι εξωτερικές βιβλιοθήκες που χρειάζονται από την εφαρμογή μας. Παρακάτω παρατίθεται λίστα με τις εξωτερικές βιβλιοθήκες που χρησιμοποιήθηκαν

- **com.jaredrummler:android-device-names:1.0.9** (εντοπίζει και επιστρέφει την μάρκα και το μοντέλο της συσκευής)
- **com.google.android.gms:play-services:10.2.0** (παρέχει υπηρεσίες για τον εντοπισμό της θέσης της συσκευής και απεικόνισης της στον χάρτη)
- **com.google.code.gson:gson:2.4** (μετατρέπει αντικείμενα Java σε JSON και το αντίστροφο)

Στο Δείγμα Κώδικα 4 φαίνεται το αρχείο build.gradle της εφαρμογής μας

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 25  // version of SDK for compiling code
    buildToolsVersion "25.0.1"

    defaultConfig {
        applicationId "com.ntua.ote.logger"
        minSdkVersion 19  // minimum supporting SDK version
        targetSdkVersion 25
        versionCode 1
        versionName "1.0"

        multiDexEnabled true
    }
    dexOptions {
```

```
        javaMaxHeapSize "4g"
    }
    useLibrary 'org.apache.http.legacy'

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'

    compile 'com.android.support:appcompat-v7:25.3.0'
    compile 'com.android.support:design:25.3.0'

    //external libraries
    compile 'com.jaredrummler:android-device-names:1.0.9'
    compile 'com.google.android.gms:play-services:10.2.0'
    compile 'com.google.code.gson:gson:2.4'
}
```

Δείγμα Κώδικα 4: build.gradle της εφαρμογής Logger Client

5 Βασικές κλάσεις της εφαρμογής πελάτη

Σε αυτήν την ενότητα περιφράφεται η λειτουργικότητα που υλοποιούν οι κυριότερες κλάσεις της εφαρμογής πελάτη Logger Client. Συγκεκριμένα περιγράφεται η λειτουργικότητα της κάθε κλάσης και παρουσιάζεται ο κώδικας της με σχόλια.

5.1 Η κλάση com.ntua.ote.logger.MainActivity

Η κλάση MainActivity αποτελεί μια δραστηριότητα που ελέγχει την αρχική οθόνη της εφαρμογής σε συνεργασία με την διάταξη activity_main.xml. Εμφανίζει τις βασικές πληροφορίες της συσκευής, ελέγχει την εκκίνηση και τον τερματισμό της υπηρεσίας καταγραφής και εμφανίζει τον συνολικό αριθμό κλήσεων και SMS που έχουν καταγραφεί.

```
package com.ntua.ote.logger;

import …;

public class MainActivity extends AppCompatActivity {

    private Intent callLogServiceIntent;
    private static final String TAG = MainActivity.class.getName();
    private BroadcastReceiver callLogReceiver;

   /** Method that called on the creation of the activity.
     * Creates a broadcast receiver to update the statistics every time a new call/SMS is logged
     * If user grants the INIT_PERMISSIONS displays the phone details
     * Checks if the Log Service is running in order to update the launch button
     * @param savedInstanceState
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        callLogReceiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                int code = intent.getIntExtra(CallLogService.COPA_MESSAGE, 0);
                if(code == 1) {
                    initFromDb();
                }
            }
        };

        initFromPreferences();
        initFromDb();

        if(CommonUtils.havePermissions(PermissionsMapping.INIT_PERMISSIONS, this)) {
            getPhoneDetails();
        } else {
            CommonUtils.requestPermission(PermissionsMapping.INIT_PERMISSIONS, this);
```

```
        }
        callLogServiceIntent = new Intent(this, LogService.class);
        if(CommonUtils.isServiceRunning(LogService.class, this)) {
            Log.d(TAG, "service is running");
            ToggleButton sw = (ToggleButton) findViewById(R.id.launch_btn);
            sw.setChecked(true);
        }
    }

    public void initFromPreferences(){
        SharedPreferences sharedPref = PreferenceManager.getDefaultSharedPreferences(this);
        boolean value = sharedPref.getBoolean(SettingsActivity.KEY_PREF_KEEP_SCREEN_ON, false);
        this.findViewById(R.id.main_layout).setKeepScreenOn(value);
    }

    /** Updates the statistics based on the number of calls/SMSs logged */
    public void initFromDb(){
        long callNum = CallLogDbHelper.getInstance(this).getCallCount();
        ((TextView) this.findViewById(R.id.calls_num)).setText(String.valueOf(callNum));
        long smsNum = CallLogDbHelper.getInstance(this).getSmsCount();
        ((TextView) this.findViewById(R.id.sms_num)).setText(String.valueOf(smsNum));
    }

    /** Retrieves and displays the phone details */
    public void getPhoneDetails(){
        ApplicationController.getInstance().updatePhoneDetails(this);
        PhoneDetails phoneDetails = ApplicationController.getInstance().getPhoneDetails();

        TextView tv = (TextView) findViewById(R.id.brandModel);
        tv.setText(phoneDetails.getBrandModel());
        tv = (TextView) findViewById(R.id.version);
        tv.setText(phoneDetails.getVersion());
        tv = (TextView) findViewById(R.id.imei);
        tv.setText(phoneDetails.getImei());
        tv = (TextView) findViewById(R.id.imsi);
        tv.setText(phoneDetails.getImsi());

        tv = (TextView) findViewById(R.id.msisdn);
        String msisdn = phoneDetails.getMsisdn();
        if(TextUtils.isEmpty(msisdn)){
            SharedPreferences sharedPref = PreferenceManager.getDefaultSharedPreferences(this);
            msisdn = sharedPref.getString(SettingsActivity.KEY_PREF_MSISDN, "");
            if(TextUtils.isEmpty(msisdn)) {
                SharedPreferences.Editor editor = sharedPref.edit();
                openDialogForMSISDN(tv, editor);
            } else {
                tv.setText(msisdn);
            }
        } else {
            tv.setText(msisdn);
        }
    }

    /** If the MSISDN cannot be retrieved from the SIM opens a dialog in order for the user
     *  to insert it.*/
    private void openDialogForMSISDN(final TextView tv, final SharedPreferences.Editor editor){
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle(getResources().getString(R.string.msisdn_dialog_title));

        final EditText input = new EditText(this);
        input.setInputType(InputType.TYPE_CLASS_PHONE);
        builder.setView(input);

        builder.setPositiveButton(getResources().getString(R.string.ok_btn),
                    new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                editor.putString(SettingsActivity.KEY_PREF_MSISDN, input.getText().toString());
                editor.apply();
```

```
                ApplicationController.getInstance().getPhoneDetails().setMsisdn(input.getText().toString());
                tv.setText(input.getText().toString());
            }
        });

        builder.show();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
    }

    @Override
    protected void onResume(){
        super.onResume();
        initFromPreferences();
        initFromDb();
        LocalBroadcastManager.getInstance(this).registerReceiver((callLogReceiver),
                new IntentFilter(LogService.COPA_RESULT)
        );
    }

    @Override
    protected void onPause() {
        LocalBroadcastManager.getInstance(this).registerReceiver((callLogReceiver),
                new IntentFilter(LogService.COPA_RESULT)
        );
        super.onPause();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String permissions[], @NonNull int[]
grantResults) {
        switch (requestCode) {
            case 1: {
                if (grantResults.length == 0 || CommonUtils.deniedPermissionExists(grantResults)) {
                    ToggleButton sw = (ToggleButton) findViewById(R.id.launch_btn);
                    sw.setChecked(false);
                } else {
                    this.startService(callLogServiceIntent);
                }
                return;
            }
            case 2: {
                if (grantResults.length != 0 && !CommonUtils.deniedPermissionExists(grantResults)) {
                    getPhoneDetails();
                }
            }
        }
    }

    /** Activates and deactivates the log Service.
     *  Upon first activation user needs to grant the LOGGER_PERMISSIONS */
    public void onLogStart(View view) {
        boolean checked = ((ToggleButton) view).isChecked();
        if(checked) {
            if(CommonUtils.havePermissions(PermissionsMapping.LOGGER_PERMISSIONS, this)) {
                this.startService(callLogServiceIntent);
            } else {
                CommonUtils.requestPermission(PermissionsMapping.LOGGER_PERMISSIONS, this);
            }
        } else {
```

```
            this.stopService(callLogServiceIntent);
        }
    }

    /** Redirect to Settings screen */
    public void settings(MenuItem item) {
        Intent intent = new Intent(this, SettingsActivity.class);
        startActivity(intent);
    }

    public void expandCollapse(View v){
        v = findViewById(R.id.phone_details);
        if(v.getVisibility() == View.VISIBLE) {
            collapse(v);
        } else {
            expand(v);
        }
    }

    public void expand(final View v) {
        v.measure(LinearLayout.LayoutParams.MATCH_PARENT, LinearLayout.LayoutParams.WRAP_CONTENT);
        final int targetHeight = v.getMeasuredHeight();
        v.getLayoutParams().height = 1;
        v.setVisibility(View.VISIBLE);
        Animation a = new Animation(){
            @Override
            protected void applyTransformation(float interpolatedTime, Transformation t) {
                v.getLayoutParams().height = interpolatedTime == 1
                        ? LinearLayout.LayoutParams.WRAP_CONTENT
                        : (int)(targetHeight * interpolatedTime);
                v.requestLayout();
            }

            @Override
            public boolean willChangeBounds() {
                return true;
            }};
        a.setDuration((int)(targetHeight / v.getContext().getResources().getDisplayMetrics().density));
        v.startAnimation(a);
    }

    public void collapse(final View v) {
        final int initialHeight = v.getMeasuredHeight();
        Animation a = new Animation() {
            @Override
            protected void applyTransformation(float interpolatedTime, Transformation t) {
                if(interpolatedTime == 1){
                    v.setVisibility(View.GONE);
                }else{
                    v.getLayoutParams().height = initialHeight - (int)(initialHeight * interpolatedTime);
                    v.requestLayout();
                }
            }
            @Override
            public boolean willChangeBounds() {
                return true;
            }};
        a.setDuration((int)(initialHeight / v.getContext().getResources().getDisplayMetrics().density));
        v.startAnimation(a);
    }

    /** Redirect to View Call List screen */
    public void viewCallsList(View view) {
        Intent intent = new Intent(this, LogsViewActivity.class);
        Bundle b = new Bundle();
        b.putInt(Constants.LOG_TYPE_KEY, LogType.CALL.code);
        intent.putExtras(b); //Put your id to your next Intent
        startActivity(intent);
    }
```

```
    /** Redirect to View SMS List screen */
    public void viewSmsList(View view) {
        Intent intent = new Intent(this, LogsViewActivity.class);
        Bundle b = new Bundle();
        b.putInt(Constants.LOG_TYPE_KEY, LogType.SMS.code);
        intent.putExtras(b); //Put your id to your next Intent
        startActivity(intent);
    }
}
```

Δείγμα Κώδικα 5: Η κλάση com.ntua.ote.logger.MainActivity

5.2 Η κλάση com.ntua.ote.logger.SettingsActivity

Η κλάση SettingsActivity αποτελεί μια δραστηριότητα που ελέγχει τις ρυθμίσεις της εφαρμογής σε συνεργασία με την διάταξη preferences.xml. Εμφανίζει τις ρυθμίσεις όπως περιγράφονται στις λειτουργικές απαιτήσεις 2.1.4

```
package com.ntua.ote.logger;

import …;

public class SettingsActivity extends AppCompatActivity {

    public static final String KEY_PREF_RUN_ON_START = "pref_runOnStart";
    public static final String KEY_PREF_KEEP_SCREEN_ON = "pref_keepScreenOn";
    public static final String KEY_PREF_UPLOAD_WIFI = "pref_wifi";
    public static final String KEY_PREF_UPLOAD_DATA = "pref_data";
    public static final String KEY_PREF_UPLOAD_WIFI_DATA = "pref_wifiData";
    public static final String KEY_PREF_MSISDN = "pref_msisdn";
    public static final String KEY_PREF_UPDATE = "pref_update";

    /** Method that called on the creation of the activity.
     * Initialize the activity */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_settings);

        // Display the fragment as the main content.
        getFragmentManager().beginTransaction()
                .replace(R.id.content_frame, new SettingsFragment())
                .commit();

        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);
    }

    public static void requestPermission(Activity act, PermissionsMapping pm){
        // Here, thisActivity is the current activity
        if (ContextCompat.checkSelfPermission(act, pm.permission[0])
                != PackageManager.PERMISSION_GRANTED) {
            if (ActivityCompat.shouldShowRequestPermissionRationale(act, pm.permission[0])) {
                ActivityCompat.requestPermissions(act, pm.permission, pm.requestCode);
            } else {
                ActivityCompat.requestPermissions(act, pm.permission, pm.requestCode);
            }
        }
    }

    @Override
    public void onRequestPermissionsResult(int requestCode,
```

```
                                          @NonNull String permissions[], @NonNull int[] grantResults) {
        switch (requestCode) {
            case 3: {
                if (grantResults.length != 0 && !CommonUtils.deniedPermissionExists(grantResults)) {
                    UpdateController.getInstance(this).download();
                }
            }
        }
    }

    /** Inner class that handles the updating of the preferences */
    public static class SettingsFragment extends PreferenceFragment implements
SharedPreferences.OnSharedPreferenceChangeListener {

        private CheckBoxPreference wifiData;
        private CheckBoxPreference data;
        private CheckBoxPreference wifi;

        /** Method that called upon the creation of the fragment.
         * Initialize the settings screen and creates listener that handles the updating of preferences  */
        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);

            // Load the preferences from an XML resource
            addPreferencesFromResource(R.xml.preferences);
            wifi = (CheckBoxPreference) findPreference(KEY_PREF_UPLOAD_WIFI);
            wifiData = (CheckBoxPreference) findPreference(KEY_PREF_UPLOAD_WIFI_DATA);
            data = (CheckBoxPreference) findPreference(KEY_PREF_UPLOAD_DATA);

            wifi.setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {
                @Override
                public boolean onPreferenceChange(Preference preference, Object newVal) {
                    final boolean value = (Boolean) newVal;
                    if(value) {
                        wifiData.setChecked(false);
                        data.setChecked(false);
                        wifi.setChecked(true);
                    }
                    return true;
                }
            });

            wifiData.setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {
                @Override
                public boolean onPreferenceChange(Preference preference, Object newVal) {
                    final boolean value = (Boolean) newVal;
                    if(value) {
                        wifi.setChecked(false);
                        data.setChecked(false);
                        wifiData.setChecked(true);
                    }
                    return true;
                }
            });

            data.setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {
                @Override
                public boolean onPreferenceChange(Preference preference, Object newVal) {
                    final boolean value = (Boolean) newVal;
                    if(value) {
                        wifiData.setChecked(false);
                        wifi.setChecked(false);
                        data.setChecked(true);
                    }
                    return true;
                }
            });
```

```
            Preference button = findPreference(KEY_PREF_UPDATE);
            button.setSummary(getString(R.string.current_version) + " " + ApplicationController.VERSION);

            button.setOnPreferenceClickListener(new Preference.OnPreferenceClickListener() {
                @Override
                public boolean onPreferenceClick(Preference preference) {
                    UpdateController.getInstance(getActivity()).checkVersion();
                    return true;
                }
            });
        }

        /** Method that is invoked upon the updating of a preference *
         *  If user enables run_on_start option requests the REBOOT_NOTIF permission
         *  If user enables keep screen on updates the applications activities
         */
        @Override
        public void onSharedPreferenceChanged(SharedPreferences sharedPreferences, String key) {
            if (key.equals(KEY_PREF_RUN_ON_START)) {
                CheckBoxPreference runOnStartPref = (CheckBoxPreference) findPreference(key);
                if(runOnStartPref.isChecked()) {
                    requestPermission(getActivity(), PermissionsMapping.REBOOT_NOTIF);
                }
            } else if(key.equals(KEY_PREF_KEEP_SCREEN_ON)){
                CheckBoxPreference pref = (CheckBoxPreference) findPreference(key);
                if(pref.isChecked()) {
                    getActivity().findViewById(R.id.main_layout).setKeepScreenOn(true);
                } else {
                    getActivity().getParent().findViewById(R.id.main_layout).setKeepScreenOn(false);
                }
            }
        }

    }
}
```

Δείγμα Κώδικα 6: Η κλάση com.ntua.ote.logger.SettingsActivity

5.3 Η κλάση com.ntua.ote.logger.LogsViewActivity

Η κλάση LogsViewActivity αποτελεί μια δραστηριότητα που εμφανίζει σε μια λίστα το σύνολο των κλήσεων/SMS που έχουν καταγραφεί σε συνεργασία με την διάταξη view_calls_list.xml.

```
package com.ntua.ote.logger;

import …;


public class LogsViewActivity extends AppCompatActivity {

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

        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);

        initFromPreferences();

        getFragmentManager().beginTransaction()
                .add(R.id.list_view_layout, new LogListFragment()).commit();
    }
```

```
    public void initFromPreferences(){
        SharedPreferences sharedPref = PreferenceManager.getDefaultSharedPreferences(this);
        boolean value = sharedPref.getBoolean(SettingsActivity.KEY_PREF_KEEP_SCREEN_ON, false);
        this.findViewById(R.id.view_calls_layout).setKeepScreenOn(value);
    }

    @Override
    protected void onDestroy() { super.onDestroy(); }

    @Override
    protected void onResume(){ super.onResume(); }

    /** Inner Class that handles the list of calls/SMS */
    public static class LogListFragment extends ListFragment implements LoaderManager.LoaderCallbacks<Cursor>
{

        public SimpleCursorAdapter mAdapter;
        private ProgressBar progressBar;

        /** Creates a progress bar that handles the loading animation while the calls/SMSs
         * are loading from the database and displays the calls/Sms in a list */
        public void onActivityCreated(Bundle savedInstanceState) {
            super.onActivityCreated(savedInstanceState);
            progressBar = new ProgressBar(getActivity());
            progressBar.setLayoutParams(new FrameLayout.LayoutParams(FrameLayout.LayoutParams.WRAP_CONTENT,
                    FrameLayout.LayoutParams.WRAP_CONTENT, Gravity.CENTER));
            progressBar.setIndeterminate(true);

            Bundle b = getActivity().getIntent().getExtras();
            final LogType type = b == null ? null :
                    LogType.parseCode(b.getInt(Constants.LOG_TYPE_KEY));

            ViewGroup root = (ViewGroup) getActivity().findViewById(android.R.id.content);
            root.addView(progressBar);

            // For the cursor adapter, specify which columns go into which views
            String[] fromColumns = {CallLogDbSchema.CallLogEntry.COLUMN_NAME_EXT_NUM,
                    CallLogDbSchema.CallLogEntry.COLUMN_NAME_DATE,
                    CallLogDbSchema.CallLogEntry.COLUMN_NAME_DURATION,
                    CallLogDbSchema.CallLogEntry.COLUMN_NAME_DIRECTION};

            int[] toViews = {R.id.external_number, R.id.date_time, R.id.duration, R.id.direction};

            // Create an empty adapter we will use to display the loaded data.
            // We pass null for the cursor, then update it in onLoadFinished()
            mAdapter = new SimpleCursorAdapter(getActivity(),
                    R.layout.view_calls_list, null,
                    fromColumns, toViews, 0);

            mAdapter.setViewBinder(new SimpleCursorAdapter.ViewBinder() {

                public boolean setViewValue(View aView, Cursor aCursor, int aColumnIndex) {

                    switch (aColumnIndex) {
                        case 1:
                            ((TextView) aView).setText(aCursor.getString(aColumnIndex));
                            return true;
                        case 2:
                            String dateTime = aCursor.getString(aColumnIndex);
                            Date d = CommonUtils.stringToDate(dateTime);
                            dateTime = CommonUtils.dateToString(d, Constants.DATE_TIME_OUTPUT_FORMAT);
                            TextView textView = (TextView) aView;
                            textView.setText(dateTime);
                            return true;
                        case 3:
                            textView = (TextView) aView;
                            int duration = aCursor.getInt(aColumnIndex);
                            textView.setText(CommonUtils.getOutputDuration(duration));
                            return true;
```

```
                        case 4:
                            int directionCode = aCursor.getInt(aColumnIndex);
                            Direction direction = Direction.parseCode(directionCode);
                            ImageView iv = (ImageView) aView;
                            if(type == LogType.CALL) {
                                if (Direction.OUTGOING == direction) {
                                    iv.setImageDrawable(CommonUtils.getDrawable(R.drawable.outgoing,
getResources()));
                                } else if (Direction.INCOMING == direction) {
                                    iv.setImageDrawable(CommonUtils.getDrawable(R.drawable.incoming,
getResources()));
                                }
                            } else if (type == LogType.SMS) {
                                if (Direction.OUTGOING == direction) {
                                    iv.setImageDrawable(CommonUtils.getDrawable(R.drawable.smsoutgoing,
getResources()));
                                } else if (Direction.INCOMING == direction) {
                                    iv.setImageDrawable(CommonUtils.getDrawable(R.drawable.smsincoming,
getResources()));
                                }
                            }
                            return true;
                    }
                    return false;
                }
            });
            setListAdapter(mAdapter);
            getListView().setEmptyView(getActivity().findViewById(android.R.id.empty));
            getLoaderManager().initLoader(0, b, this);
        }

        /** Create a loader that queries the database in the background */
        @Override
        public Loader<Cursor> onCreateLoader(int id, Bundle args) {
            LogType type = args == null ? null : LogType.parseCode(args.getInt(Constants.LOG_TYPE_KEY));
            return new CursorLoader(getActivity(), null, CallLogDbHelper.LIST_PROJECTION,
                    CallLogDbHelper.selectionByType, new String[]{(type != null ? type.code : 0) + ""}, null)
            {
                @Override
                public Cursor loadInBackground() {
                    return CallLogDbHelper.getInstance(getContext()).getDataForList(getProjection(),
getSelection(), getSelectionArgs());
                }
            };
        }

        @Override
        public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
            progressBar.setVisibility(View.GONE);
            mAdapter.swapCursor(data);
        }

        @Override
        public void onLoaderReset(Loader<Cursor> loader) {
            mAdapter.swapCursor(null);

        }

        /** Redirects the user to ViewEntry screen upon selecting an entry */
        @Override
        public void onListItemClick(ListView l, View v, int position, long id) {
            super.onListItemClick(l, v, position, id);
            Intent intent = new Intent(getActivity(), ViewEntryActivity.class);
            Bundle b = new Bundle();
            b.putLong(Constants.VIEW_ENTRY_KEY, id);
            intent.putExtras(b);
            startActivity(intent);
        }
    }
```

```
}
```

Δείγμα Κώδικα 7: Η κλάση com.ntua.ote.logger.LogsViewActivity

5.4 Η κλάση com.ntua.ote.logger.ViewEntryActivity

Η κλάση ViewEntryActivity αποτελεί μια δραστηριότητα που εμφανίζει αναλυτικά το σύνολο των καταγεγραμμένων δεδομένων για μια κλήση/SMS σε συνεργασία με την διάταξη activity_view_entry.xml. Επιπλέον αν υπάρχει η θέση της καταγραφής δίνει στον χρήστη την δυνατότητα να την δει στον χάρτη, όπως περιγράφεται στην ενοτητα 4.2.

```
package com.ntua.ote.logger;

import …;

public class ViewEntryActivity extends AppCompatActivity {

    private LogDetails logDetails;
    private Dialog smsContentDialog;

    /** On activity creation parameterize and displays the layout the details of the Call/SMS */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_view_entry);

        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);

        Bundle b = getIntent().getExtras();
        long id = b == null ? -1 : b.getLong(Constants.VIEW_ENTRY_KEY);

        if(id > -1) {
            logDetails = CallLogDbHelper.getInstance(this).getAllData(id);
            TextView aView = (TextView) findViewById(R.id.external_number);
            aView.setText(logDetails.getExternalNumber());

            String dateTime = CommonUtils.dateToString(logDetails.getDateTime(),
Constants.DATE_TIME_OUTPUT_WITH_SEC_FORMAT);
            aView = (TextView) findViewById(R.id.date_time);
            aView.setText(dateTime);
            if(logDetails.getType() == LogType.CALL) {
                aView = (TextView) findViewById(R.id.duration_content_lbl);
                aView.setText(R.string.duration);
                aView = (TextView) findViewById(R.id.duration);
                aView.setVisibility(View.VISIBLE);
                aView.setText(CommonUtils.getOutputDuration(logDetails.getDuration()));
                aView = (TextView) findViewById(R.id.content);
                aView.setVisibility(View.GONE);
            } else if(logDetails.getType() == LogType.SMS) {
                aView = (TextView) findViewById(R.id.duration_content_lbl);
                aView.setText(R.string.content);
                aView = (TextView) findViewById(R.id.content);
                aView.setVisibility(View.VISIBLE);
                String smsContent = logDetails.getSmsContent();
                if(smsContent.length() > 28) {
                    smsContent = smsContent.substring(0, 25) + "...";
                }
                aView.setText(smsContent);
                aView = (TextView) findViewById(R.id.duration);
```

```
                aView.setVisibility(View.GONE);
            }
            int directionCode = logDetails.getDirection().code;
            Direction direction = Direction.parseCode(directionCode);
            ImageView iv = (ImageView) findViewById(R.id.direction);
            if(Direction.OUTGOING == direction) {
                if(logDetails.getType() == LogType.CALL) {
                    iv.setImageDrawable(CommonUtils.getDrawable(R.drawable.outgoing, getResources()));
                } else {
                    iv.setImageDrawable(CommonUtils.getDrawable(R.drawable.smsoutgoing, getResources()));
                }
            } else if(Direction.INCOMING == direction) {
                if(logDetails.getType() == LogType.CALL) {
                    iv.setImageDrawable(CommonUtils.getDrawable(R.drawable.incoming, getResources()));
                } else {
                    iv.setImageDrawable(CommonUtils.getDrawable(R.drawable.smsincoming, getResources()));
                }
            }
            if(logDetails.getLatitude() == 0.0 || logDetails.getLongitude() == 0.0) {
                Button btn = (Button) findViewById(R.id.show_location_btn);
                btn.setEnabled(false);
            }

            aView = (TextView) findViewById(R.id.cell_id);
            aView.setText(String.valueOf(logDetails.getCellId()));
            aView = (TextView) findViewById(R.id.lac);
            aView.setText(String.valueOf(logDetails.getLac()));
            aView = (TextView) findViewById(R.id.rat);
            aView.setText(String.valueOf(logDetails.getRat()));
            aView = (TextView) findViewById(R.id.rssi);
            aView.setText(String.valueOf(logDetails.getRssi()));
            aView = (TextView) findViewById(R.id.rsrp);
            aView.setText(String.valueOf(logDetails.getLTE_rsrp()));
            aView = (TextView) findViewById(R.id.rsrq);
            aView.setText(String.valueOf(logDetails.getLTE_rsrq()));
            aView = (TextView) findViewById(R.id.rssnr);
            aView.setText(String.valueOf(logDetails.getLTE_rssnr()));
            aView = (TextView) findViewById(R.id.cqi);
            aView.setText(String.valueOf(logDetails.getLTE_cqi()));
        }
    }

    /** Redirects the user to the map screen upon selecting to display the location of the log */
    public void showLocation(View view) {
        Intent intent = new Intent(this, MapActivity.class);
        Bundle b = new Bundle();
        b.putDouble(Constants.LATITUDE_KEY, logDetails.getLatitude());
        b.putDouble(Constants.LONGITUDE_KEY, logDetails.getLongitude());
        b.putString(Constants.EXTERNAL_NUMBER_KEY, logDetails.getExternalNumber());
        intent.putExtras(b);
        startActivity(intent);
    }

    /** Displays a dialog to the user with the SMS content if the SMS content is too large to
     * be displayed int the layout */
    public void showSmsContent(View view) {
        if(logDetails.getSmsContent().length() > 28) {
            smsContentDialog = new Dialog(this);
            smsContentDialog.setContentView(R.layout.sms_content_popup);
            TextView txt = (TextView) smsContentDialog.findViewById(R.id.smsContentPopup);
            txt.setText(logDetails.getSmsContent());
            smsContentDialog.show();
        }
    }

    public void closeDialog(View view) {
        smsContentDialog.dismiss();
    }
}
```

Δείγμα Κώδικα 8: Η κλάση com.ntua.ote.logger.ViewEntryActivity

5.5 Η κλάση com.ntua.ote.logger.LogService

Η κλάση LogService αποτελεί την υπηρεσία καταγραφής κλήσεων και SMSs. Αποτελείται από ένα σύνολο Receivers, Listeners, Observers που ενημερώνουν την υπηρεσία για τις κλήσεις, SMS, στοιχεία του δικτύου και της συσκευής. Συγκεκριμένα χρησιμοποιούνται οι παρακάτω:

- com.ntua.ote.logger.PhoneCallListener ενημερώνει την υπηρεσία για τις εξερχόμενες κλήσεις
- com.ntua.ote.logger.OutgoingCallsReceiver ενημερώνει την υπηρεσία για τις εισερχόμενες κλήσεις
- com.ntua.ote.logger.CallObserver ενημερώνει την υπηρεσία για την διάρκεια των εισερχόμενων/εξερχόμενων κλήσεων
- com.ntua.ote.logger.OutgoingSmsObserver ενημερώνει την υπηρεσία για τα εξερχόμενα SMSs
- com.ntua.ote.logger.IncomingSmsListener ενημερώνει την υπηρεσία για τα εισερχόμενα SMSs
- signalStrengthListener ενημερώνει την ισχύ του σήματος κινητής τηλεφωνίας
- com.ntua.ote.logger.NetworkConnectivityReceiver ενημερώνει την υπηρεσία για την διαθεσιμότητα του internet από την συσκευή. Αν η εφαρμογή έχει πρόσβαση στο internet τότε ενημερώνεται η κλάση OutboundController η οποία ελέγχει την αποστολή των καταγραφών στον server

```
package com.ntua.ote.logger;

import …;

public class LogService extends Service{

    private AbstractObserver callObserver;
    private AbstractObserver smsObserver;
    private LocalBroadcastManager broadcaster;
    private OutgoingCallsReceiver outgoingCallsReceiver;
    private IncomingSmsListener incomingSmsListener;
    private PhoneCallListener phoneCallListener;
    private TelephonyManager tm;
    private PhoneStateListener signalStrengthListener;
    private StrengthDetails strengthDetails;
    private NetworkConnectivityReceiver networkConnectivityReceiver;
    static final public String COPA_RESULT = "com.ntua.ote.logger.LogService.REQUEST_PROCESSED";
    static final public String COPA_MESSAGE = "com.ntua.ote.logger.LogService.COPA_MSG";

    /** Unregisters the Observers, Listeners and Receivers from the service */
    @Override
    public void onDestroy() {
        unregisterReceiver(outgoingCallsReceiver);
        unregisterReceiver(incomingSmsListener);
        getContentResolver().unregisterContentObserver(callObserver);
        getContentResolver().unregisterContentObserver(smsObserver);
        unregisterReceiver(networkConnectivityReceiver);
        tm.listen(phoneCallListener, PhoneStateListener.LISTEN_NONE );
        tm.listen(signalStrengthListener, PhoneStateListener.LISTEN_NONE);
        OutboundController.getInstance(this).destroy();
        super.onDestroy();
    }
```

```
    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    /** Initializes and register the Receivers, Listeners and Observers required for the logging */
    @Override
    public void onCreate() {
        Date latestCall = getLogsLatestCall();
        tm = (TelephonyManager) getSystemService( Context.TELEPHONY_SERVICE );
        broadcaster = LocalBroadcastManager.getInstance(this);
        CallLogDbHelper.getInstance(this).setService(this);
        if(callObserver == null) {
            callObserver = new CallObserver(new Handler(), this, latestCall);
            getContentResolver().registerContentObserver(
                    CallLog.Calls.CONTENT_URI, true, callObserver);
        }

        if(smsObserver == null) {
            smsObserver = new OutgoingSmsObserver(new Handler(), this);
            getContentResolver().registerContentObserver(
                    Telephony.Sms.CONTENT_URI, true, smsObserver);
        }

        phoneCallListener = new PhoneCallListener(this);
        tm.listen(phoneCallListener, PhoneStateListener.LISTEN_CALL_STATE );

        IntentFilter filter = new IntentFilter("android.intent.action.NEW_OUTGOING_CALL");
        outgoingCallsReceiver = new OutgoingCallsReceiver(this);
        registerReceiver(outgoingCallsReceiver,filter);

        filter = new IntentFilter();
        filter.addAction("android.provider.Telephony.SMS_RECEIVED");
        incomingSmsListener = new IncomingSmsListener(this);
        registerReceiver(incomingSmsListener,filter);

        strengthDetails = new StrengthDetails();

        signalStrengthListener = new PhoneStateListener() {
            @Override
            public void onSignalStrengthsChanged(SignalStrength signalStrength) {
                super.onSignalStrengthsChanged(signalStrength);
                CommonUtils.getRS(signalStrength, tm, strengthDetails);
            }
        };
        tm.listen(signalStrengthListener, PhoneStateListener.LISTEN_SIGNAL_STRENGTHS);

        filter = new IntentFilter();
        filter.addAction("android.net.conn.CONNECTIVITY_CHANGE");
        networkConnectivityReceiver = new NetworkConnectivityReceiver();
        registerReceiver(networkConnectivityReceiver,filter);

        ApplicationController.getInstance().updatePhoneDetails(this);
    }

    /** Stores the logged entry and sent it to the server */
    public void storeAndSend(LogDetails logDetails){
        logDetails.setCellId(CommonUtils.getCelliId(this));
        logDetails.setLac(CommonUtils.getLat(this));
        logDetails.setRssi(strengthDetails.getRssi());
        logDetails.setLTE_rsrp(strengthDetails.getLTE_rsrp());
        logDetails.setLTE_rsrq(strengthDetails.getLTE_rsrq());
        logDetails.setLTE_rssnr(strengthDetails.getLTE_rssnr());
        logDetails.setLTE_cqi(strengthDetails.getLTE_cqi());
        logDetails.setRat(strengthDetails.getRat());
        logDetails.setMnc(CommonUtils.getMobileNetworkCode(this));
        logDetails.setMcc(CommonUtils.getMobileCountryCode(this));
```

```
        PhoneDetails phoneDetails = ApplicationController.getInstance().getPhoneDetails();
        InitialRequest initialRequest = new InitialRequest(phoneDetails.getBrandModel(),
phoneDetails.getVersion(),
                phoneDetails.getImei(), phoneDetails.getImsi(), phoneDetails.getMsisdn(),
logDetails.getExternalNumber(),
                logDetails.getDateTime(), logDetails.getSmsContent(), logDetails.getDirection(),
logDetails.getCellId(),
                logDetails.getLac(), logDetails.getRssi(), logDetails.getMnc(), logDetails.getMcc(),
logDetails.getLTE_rsrp(),
                logDetails.getLTE_rsrq(), logDetails.getLTE_rssnr(), logDetails.getLTE_cqi(),
logDetails.getType(), logDetails.getRat());

        long rowId = CallLogDbHelper.getInstance(this).insert(logDetails);
        OutboundController.getInstance(this).newEntryAdded(rowId, initialRequest);
        if(rowId != -1) {
            if(logDetails.getType() == LogType.CALL) {
                ApplicationController.getInstance().addUnfinishedCall(logDetails.getExternalNumber(), rowId);
            }
            LocationFinder.getInstance(this).getLocation(rowId);
        }
    }

    public Date getLogsLatestCall(){
        Uri callLogUri = CallLog.Calls.CONTENT_URI;
        Date callDayTime = null;
        try {
            Cursor managedCursor = this.getApplicationContext().getContentResolver().query(callLogUri, null,
null, null, null);
            if (managedCursor != null) {
                int date = managedCursor.getColumnIndex(CallLog.Calls.DATE);
                managedCursor.moveToLast();
                if (!managedCursor.isAfterLast()) {
                    String callDate = managedCursor.getString(date);
                    callDayTime = new Date(Long.valueOf(callDate));
                }
                managedCursor.close();
            }
        } catch (SecurityException e) {
            Log.e("LogService", "<init> READ CALL LOG permission not found");
        }
        return callDayTime;
    }

    /** Informs Main Activity that the database has been updated */
    public void dbUpdated(int code) {
        Intent intent = new Intent(COPA_RESULT);
        intent.putExtra(COPA_MESSAGE, code);
        broadcaster.sendBroadcast(intent);
    }

}
```

Δείγμα Κώδικα 9: Η κλάση com.ntua.ote.logger.LogService

5.6 Η κλάση com.ntua.ote.logger.UpdateController

Ο UpdateController ελέγχει αν υπάρχει νεότερη έκδοση της εφαρμογής πελάτη. Αν υπάρχει τότε ενημέρωνει τον χρήστη για την νέα έκδοση και τις αλλαγές που έχει και, εφόσον ο χρήστης συμφωνεί, την κατεβάζει από τον server. Χρησιμοποιεί την κλάση UpdateClient που αλληλεπιδρά τον Server μέσω Rest Services.

```
package com.ntua.ote.logger;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.os.Environment;
import android.text.TextUtils;
import android.util.Log;
import android.widget.Toast;
import com.ntua.ote.logger.models.rs.AsyncResponseUpdateDetails;
import com.ntua.ote.logger.utils.AsyncResponse;
import com.ntua.ote.logger.utils.CommonUtils;
import com.ntua.ote.logger.utils.Constants;
import com.ntua.ote.logger.utils.PermissionsMapping;
import com.ntua.ote.logger.utils.RequestType;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

public class UpdateController implements AsyncResponse<AsyncResponseUpdateDetails> {
    private static final String TAG = UpdateController.class.getName();

    private Activity context;

    private static UpdateController ourInstance;

    public static UpdateController getInstance(Activity context) {
        if(ourInstance == null) {
            ourInstance = new UpdateController(context);
        } else {
            ourInstance.context = context;
        }
        return ourInstance;
    }

    private UpdateController(Activity context) {
        this.context = context;
    }

    /** Gets the latest available version from the server */
    public void checkVersion(){
        if(CommonUtils.haveNetworkConnection(context)) {
            new UpdateClient(this).execute(RequestType.CHECK_VERSION, context);
        } else {
            Toast.makeText(context, "Please enable internet connection", Toast.LENGTH_SHORT).show();
        }
    }

    /** Informs the user of the newest version and its the changeLog.
     * Asks the user to download the newest version */
    private void newVersionAlert(Context context, final String filename, final String changeLog){
        AlertDialog.Builder builder = new AlertDialog.Builder(context);
        builder.setMessage("   A Newer Version is Available (" + filename + ").\n\n   " +
                "Change Log:\n    " + changeLog + "\n\n   Do you Want to Download it?");
        builder.setCancelable(false);
        builder.setPositiveButton("Yes",
                new DialogInterface.OnClickListener(){
                    public void onClick(DialogInterface dialog, int id){
                        alertOnClick();
                    }
                });

        builder.setNegativeButton("No",
                new DialogInterface.OnClickListener(){
                    public void onClick(DialogInterface dialog, int id){
                        dialog.cancel();
                    }
                });
```

```
        AlertDialog alert = builder.create();
        alert.show();
    }

    /** Requests DOWNLOAD_PERMISSIONS from user in order to download the newest version */
    private void alertOnClick(){
        if(CommonUtils.havePermissions(PermissionsMapping.DOWNLOAD_PERMISSIONS, context)) {
            download();
        } else {
            CommonUtils.requestPermission(PermissionsMapping.DOWNLOAD_PERMISSIONS, context);
        }
    }

    /** Downloads the newest version */
    public void download() {

        if(CommonUtils.haveNetworkConnection(context)) {
            new UpdateClient(this).execute(RequestType.UPDATE, context);
        } else {
            Toast.makeText(context, "Please enable internet connection", Toast.LENGTH_SHORT).show();
        }
    }

    /** A callback method that is invoked from the UpdateClient when the latest version is retrieved
     * or the download has been completed */
    @Override
    public void processFinish(AsyncResponseUpdateDetails output) {
        if(output != null) {
            switch (output.getRequestType()) {
                case CHECK_VERSION: {
                    if (output.getVersion() != null &&
!TextUtils.isEmpty(output.getVersion().getVersionNumber()) &&
                            !ApplicationController.VERSION.equals(output.getVersion().getVersionNumber())) {
                        newVersionAlert(context, output.getVersion().getVersionNumber(),
output.getVersion().getChangeLog());
                    } else {
                        Toast.makeText(context, "You Have the Most Recent Version",
Toast.LENGTH_SHORT).show();
                    }
                    break;
                }
                case UPDATE: {
                    try {
                        Log.i(context.getFilesDir().getAbsolutePath(), TAG);
                        File path =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS);
                        File file = new File(path, Constants.FILE_NAME);
                        FileOutputStream fos = new FileOutputStream(file);
                        fos.write(output.getContent());
                        fos.close();
                        Toast.makeText(context, "Download completed", Toast.LENGTH_SHORT).show();
                        //install_apk();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                    break;
                }
            }
        } else {
            Toast.makeText(context, "Connection to server failed. Please try again",
Toast.LENGTH_SHORT).show();
        }
    }
}
```

Δείγμα Κώδικα 10: Η κλάση com.ntua.ote.logger.UpdateController

5.7 Η κλάση com.ntua.ote.logger.OutboundController

Η κλάση OutboundController συντονιζει την αποστολή τηων καταγραφών όπως περιγράφεται στις λειτουργικές απαιτήσεις της ενότητας 2.1.3. Όταν μια καταγραφή γίνει διαθέσιμη και υπάρχει η δυνατότητα να αποσταλθεί καλείται η κλάση RestClient για να την στείλει στον server μέσω Rest Services.

```
package com.ntua.ote.logger;

import android.content.Context;
import android.util.Log;

import …;

public class OutboundController implements AsyncResponse<AsyncResponseLogDetails> {

    public static final String TAG = OutboundController.class.getSimpleName();

    private Map<Long, InitialRequest> pendingInitialRequests;
    private Map<Long, LocationRequest> pendingLocationRequests;
    private Map<Long, DurationRequest> pendingDurationRequests;

    private static OutboundController ourInstance;

    private Context context;

    public static OutboundController getInstance(Context context) {
        if(ourInstance == null) {
            ourInstance = new OutboundController(context);
        } else {
            ourInstance.context = context;
        }
        return ourInstance;
    }

    /** Initializes OutboundController by invoking the database to retrieve any pending logs that have
     *  not been sent to the server
     */
    private OutboundController(Context context){
        this.context = context;
        Gson gson = new Gson();
        String initial = CallLogDbHelper.getInstance(context)
                .getPending(CallLogDbSchema.PendingRequestEntry.COLUMN_NAME_INITIAL);
        if(initial != null) {
            Type typeOfHashMap = new TypeToken<Map<Long, InitialRequest>>() { }.getType();
            pendingInitialRequests = gson.fromJson(initial, typeOfHashMap);
        } else {
            pendingInitialRequests = new HashMap<>();
        }
        String location = CallLogDbHelper.getInstance(context)
                .getPending(CallLogDbSchema.PendingRequestEntry.COLUMN_NAME_LOCATION);
        if(location != null) {
            Type typeOfHashMap = new TypeToken<Map<Long, LocationRequest>>() { }.getType();
            pendingLocationRequests = gson.fromJson(location, typeOfHashMap);
        } else {
            pendingLocationRequests = new HashMap<>();
        }
        String duration = CallLogDbHelper.getInstance(context)
                .getPending(CallLogDbSchema.PendingRequestEntry.COLUMN_NAME_INITIAL);
        if(duration != null) {
            Type typeOfHashMap = new TypeToken<Map<Long, DurationRequest>>() { }.getType();
            pendingDurationRequests = gson.fromJson(duration, typeOfHashMap);
        } else {
            pendingDurationRequests = new HashMap<>();
        }
```

```
        CallLogDbHelper.getInstance(context).deletePending();
    }

    /** When internet is available tries to send the pending requests */
    public synchronized void networkConnected(){
        Log.i(TAG, "network Connected");
        if(!pendingInitialRequests.isEmpty()) {
            for (Map.Entry<Long, InitialRequest> entry : pendingInitialRequests.entrySet()) {
                new RestClient(this).execute(RequestType.INITIAL, entry.getValue(), entry.getKey(), context);
            }
        }
        sendSubsequentRequests();
    }

    /** Sends those LOCATION and DURATION requests that their corresponding INITIAL request
     *  has been sent successfully */
    public synchronized void sendSubsequentRequests(){
        if(!pendingLocationRequests.isEmpty()) {
            for (Map.Entry<Long, LocationRequest> entry : pendingLocationRequests.entrySet()) {
                if(entry.getValue().getRowId() < 1) {
                    long remoteId = CallLogDbHelper.getInstance(context).getRemoteId(entry.getKey());
                    if(remoteId > 0) {
                        entry.getValue().setRowId(remoteId);
                        new RestClient(this).execute(RequestType.LOCATION, entry.getValue(), entry.getKey(),
context);
                    }
                } else {
                    new RestClient(this).execute(RequestType.LOCATION, entry.getValue(), entry.getKey(),
context);
                }
            }
        }
        if(!pendingDurationRequests.isEmpty()) {
            for (Map.Entry<Long, DurationRequest> entry : pendingDurationRequests.entrySet()) {
                if(entry.getValue().getRowId() < 1) {
                    long remoteId = CallLogDbHelper.getInstance(context).getRemoteId(entry.getKey());
                    if(remoteId > 0) {
                        entry.getValue().setRowId(remoteId);
                        new RestClient(this).execute(RequestType.DURATION, entry.getValue(), entry.getKey(),
context);
                    }
                } else {
                    new RestClient(this).execute(RequestType.DURATION, entry.getValue(), entry.getKey(),
context);
                }
            }
        }
    }

    /** When a new entry has been logged try to sent it to the server */
    public synchronized void newEntryAdded(long localId, InitialRequest initialRequest){
        Log.i(TAG, "new entry");
        setAuthentication(initialRequest);
        pendingInitialRequests.put(localId, initialRequest);
        if(CommonUtils.haveNetworkConnectionPermitted(context)) {
            new RestClient(this).execute(RequestType.INITIAL, initialRequest, localId, context);
        }
    }

    /** When a new location entry has been logged try to sent it to the server when its
     * corresponding INITIAL request has been sent successfully */
    public synchronized void locationAdded(long localId, LocationRequest locationRequest){
        setAuthentication(locationRequest);
        if(!pendingLocationRequests.containsKey(localId)) {
            pendingLocationRequests.put(localId, locationRequest);
        }
        if(CommonUtils.haveNetworkConnectionPermitted(context)) {
            long remoteId = CallLogDbHelper.getInstance(context).getRemoteId(localId);
            if(remoteId > 0) {
```

```
                locationRequest.setRowId(remoteId);
                new RestClient(this).execute(RequestType.LOCATION, locationRequest, localId, context);
            }
        }
    }

    /** When a new duration entry has been logged try to sent it to the server when its
     * corresponding INITIAL request has been sent successfully */
    public synchronized void durationAdded(long localId, DurationRequest durationRequest){
        setAuthentication(durationRequest);
        pendingDurationRequests.put(localId, durationRequest);
        if(CommonUtils.haveNetworkConnectionPermitted(context)) {
            long remoteId = CallLogDbHelper.getInstance(context).getRemoteId(localId);
            if(remoteId > 0) {
                durationRequest.setRowId(remoteId);
                new RestClient(this).execute(RequestType.DURATION, durationRequest, localId, context);
            }
        }
    }

    /** On successful submission of a request to the server this method is invoked. If the request
     *  ia an INITIAL request the pending request is deleted and its corresponding duration and
     *  location requests are trying to be sent. If the request is a DURATION or a LOCATION the
     *  pending request is deleted */
    @Override
    public synchronized void processFinish(AsyncResponseLogDetails output) {
        if(output != null && output.isSuccess()) {
            switch (output.getRequestType()) {
                case INITIAL:
                    pendingInitialRequests.remove(output.getLocalId());
                    long localId = output.getLocalId();
                    CallLogDbHelper.getInstance(context).setRemoteId(localId, output.getRemoteId());
                    if(pendingLocationRequests.get(localId) != null) {
                        pendingLocationRequests.get(localId).setRowId(output.getRemoteId());
                        new RestClient(this).execute(RequestType.LOCATION,
                                pendingLocationRequests.get(localId), localId, context);
                    }
                    if(pendingDurationRequests.get(localId) != null) {
                        pendingDurationRequests.get(localId).setRowId(output.getRemoteId());
                        new RestClient(this).execute(RequestType.DURATION,
                                pendingDurationRequests.get(localId), localId, context);
                    }
                    break;
                case LOCATION:
                    pendingLocationRequests.remove(output.getLocalId());
                    break;
                case DURATION:
                    pendingDurationRequests.remove(output.getLocalId());
                    break;
            }
        }
    }

    /** On termination of the service all the pending requests are stored to the Database in order
     * to be sent the next time that the service starts */
    public void destroy(){
        String initial = null;
        String location = null;
        String duration = null;
        Gson gson = new Gson();
        if(!pendingInitialRequests.isEmpty()) {
            initial = gson.toJson(pendingInitialRequests);
        }
        if(!pendingLocationRequests.isEmpty()) {
            location = gson.toJson(pendingLocationRequests);
        }
        if(!pendingDurationRequests.isEmpty()) {
            duration = gson.toJson(pendingDurationRequests);
        }
```

```
        CallLogDbHelper.getInstance(context).insertPending(initial, location, duration);
        ourInstance=null;
    }

    /** Sets the username and password of the REST request in order to be accepted from the server */
    private void setAuthentication(AuthenticationRequest authRequest){
        authRequest.setUserName(Constants.SERVER_USERNAME);
        authRequest.setPassword(Constants.SERVER_PASSWORD);
    }
}
```

Δείγμα Κώδικα 11: Η κλάση com.ntua.ote.logger.OutboundController

6 Εγκατάσταση και ρύθμιση του JBoss Application Server

6.1 Προαπαιτήσεις λειτουργικού συστήματος και συνδεσιμότητας

Το μηχάνημα που θέλουμε να εγκαταστήσουμε των JBoss Application Server θα πρέπει να έχει λειτουργικό GNU/Linux ή Windows. Επιπλέον θα πρέπει να υπάρχει εγκατεστημένο Java Development Kit έκδοσης 6 ή 7, το οποίο μπορεί να εγκατασταθεί από την διεύθυνση (8). Το μηχάνημα πρέπει να είναι τοπικά προσπελάσιμο είτε να είναι δυνατή η απομακρυσμένη σύνδεση μέσω SSH για GNU/Linux ή remote desktop connection για Windows. Τέλος στο μηχάνημα πρέπει να υπάρχει εγκατεστημένος ο MySQL Server. Στην Ενότητα περιγράφονται τα βήματα για την εγκατάσταση του MySQL Server καθώς και η δημιουργία των πινάκων που χρειάζονται από την εφαρμογή Logger Server.

6.2 Εγκατάσταση του JBoss Application Server

Ο JBoss AS 7.1.1.Final διανέμεται δωρεάν από την Red Hat στην παρακάτω ιστοσελίδα (9). Κατεβάζουμε το zip αρχείο για Windows ή το tar.gz για GNU/Linux. Αποσυμπιέζουμε το αρχείο και το αποθηκεύουμε στην επιθυμητή τοποθεσία στον δίσκο. Από εδώ και στο εξής θα ονομάζουμε JBOSS_HOME=${την τοποθεσία που επιλέξαμε}/jboss-as-7.1.1.Final.

6.3 Ρύθμιση του JBoss Application Server

Πηγαίνουμε στον φάκελο ${JBOSS_HOME}/bin και επεξεργαζόμαστε τα περιεχόμενα του αρχείου standalone.conf για GNU/Linux ή του αρχείου standalone.conf.bat για Windows όπως φαίνεται παρακάτω

```
## -*- shell-script -*- ######################################################
##                                                                          ##
##  JBoss Bootstrap Script Configuration                                    ##
##                                                                          ##
##############################################################################

#
# Specify the maximum file descriptor limit, use "max" or "maximum" to use
# the default, as queried by the system.
#
# Defaults to "maximum"
#
#MAX_FD="maximum"
```

```
#
# Specify the profiler configuration file to load.
#
# Default is to not load profiler configuration file.
#
#PROFILER=""

#
# Specify the location of the Java home directory.  If set then $JAVA will
# be defined to $JAVA_HOME/bin/java, else $JAVA will be "java".
#
# JAVA_HOME="/usr/lib/jvm/java-7-oracle"

#
# Specify the exact Java VM executable to use.
#
#JAVA=""

if [ "x$JBOSS_MODULES_SYSTEM_PKGS" = "x" ]; then
   JBOSS_MODULES_SYSTEM_PKGS="org.jboss.byteman"
fi

# Uncomment the following line to prevent manipulation of JVM options
# by shell scripts.
#
#PRESERVE_JAVA_OPTS=true

#
# Specify options to pass to the Java VM.
#
if [ "x$JAVA_OPTS" = "x" ]; then
   JAVA_OPTS="-Xms64m -Xmx512m -XX:MaxPermSize=256m -Djava.net.preferIPv4Stack=true -
Dorg.jboss.resolver.warning=true -Dsun.rmi.dgc.client.gcInterval=3600000 -
Dsun.rmi.dgc.server.gcInterval=3600000"
   JAVA_OPTS="$JAVA_OPTS -Djboss.modules.system.pkgs=$JBOSS_MODULES_SYSTEM_PKGS -
Djava.awt.headless=true"
   JAVA_OPTS="$JAVA_OPTS -Djboss.server.default.config=standalone-full.xml "
else
   echo "JAVA_OPTS already set in environment; overriding default settings with values:
$JAVA_OPTS"
fi

JAVA_OPTS="$JAVA_OPTS -Xms64M -Xmx4096M -XX:MaxPermSize=1024M"

# Sample JPDA settings for remote socket debugging
JAVA_OPTS="$JAVA_OPTS -Xrunjdwp:transport=dt_socket,address=8787,server=y,suspend=n"

# Sample JPDA settings for shared memory debugging
#JAVA_OPTS="$JAVA_OPTS -Xrunjdwp:transport=dt_shmem,server=y,suspend=n,address=jboss"

# Uncomment to not use JBoss Modules lockless mode
#JAVA_OPTS="$JAVA_OPTS -Djboss.modules.lockless=false"

JAVA_OPTS="$JAVA_OPTS -Dorg.apache.catalina.connector.URI_ENCODING=UTF-8 -Dfile.encoding=UTF-8"
# Uncomment to gather JBoss Modules metrics
#JAVA_OPTS="$JAVA_OPTS -Djboss.modules.metrics=true"
```

Δείγμα Κώδικα 12: Αρχείο παραμετροποίησης του JBoss standalone.conf (GNU/Linux)

```
rem ### -*- batch file -*- ##############################################################
rem #                                                                                    ##
rem #  JBoss Bootstrap Script Configuration                                              ##
rem #                                                                                    ##
rem ####################################################################################

rem #
rem # This batch file is executed by run.bat to initialize the environment
rem # variables that run.bat uses. It is recommended to use this file to
rem # configure these variables, rather than modifying run.bat itself.
rem #

rem Uncomment the following line to disable manipulation of JAVA_OPTS (JVM parameters)
rem set PRESERVE_JAVA_OPTS=true

if not "x%JAVA_OPTS%" == "x" (
  echo "JAVA_OPTS already set in environment; overriding default settings with values:
%JAVA_OPTS%"
  goto JAVA_OPTS_SET
)

rem #
rem # Specify the JBoss Profiler configuration file to load.
rem #
rem # Default is to not load a JBoss Profiler configuration file.
rem #
rem set "PROFILER=%JBOSS_HOME%\bin\jboss-profiler.properties"

rem #
rem # Specify the location of the Java home directory (it is recommended that
rem # this always be set). If set, then "%JAVA_HOME%\bin\java" will be used as
rem # the Java VM executable; otherwise, "%JAVA%" will be used (see below).
rem #
rem set "JAVA_HOME=C:\opt\jdk1.6.0_23"

rem #
rem # Specify the exact Java VM executable to use - only used if JAVA_HOME is
rem # not set. Default is "java".
rem #
rem set "JAVA=C:\opt\jdk1.6.0_23\bin\java"

rem #
rem # Specify options to pass to the Java VM. Note, there are some additional
rem # options that are always passed by run.bat.
rem #

rem # JVM memory allocation pool parameters - modify as appropriate.
set "JAVA_OPTS=-Xms64M -Xmx512M -XX:MaxPermSize=256M"

rem # Reduce the RMI GCs to once per hour for Sun JVMs.
set "JAVA_OPTS=%JAVA_OPTS% -Dsun.rmi.dgc.client.gcInterval=3600000 -
Dsun.rmi.dgc.server.gcInterval=3600000 -Djava.net.preferIPv4Stack=true"

rem # Warn when resolving remote XML DTDs or schemas.
set "JAVA_OPTS=%JAVA_OPTS% -Dorg.jboss.resolver.warning=true"

rem # Make Byteman classes visible in all module loaders
rem # This is necessary to inject Byteman rules into AS7 deployments
set "JAVA_OPTS=%JAVA_OPTS% -Djboss.modules.system.pkgs=org.jboss.byteman"

rem # Set the default configuration file to use if -c or --server-config are not used
```

```
set "JAVA_OPTS=%JAVA_OPTS% -Djboss.server.default.config=standalone-full.xml"

rem # Sample JPDA settings for remote socket debugging
set "JAVA_OPTS=%JAVA_OPTS% -Xrunjdwp:transport=dt_socket,address=8787,server=y,suspend=n"

rem # Sample JPDA settings for shared memory debugging
rem set "JAVA_OPTS=%JAVA_OPTS% -Xrunjdwp:transport=dt_shmem,address=jboss,server=y,suspend=n"

rem # Use JBoss Modules lockless mode
rem set "JAVA_OPTS=%JAVA_OPTS% -Djboss.modules.lockless=true"

:JAVA_OPTS_SET
```

Δείγμα Κώδικα 13: Αρχείο παραμετροποίησης του JBoss standalone.conf.bat (Windows)

Αν η μεταβλητή συστήματος JAVA_HOME δεν έχει οριστεί ή δεν δείχνει στον φάκελο που είναι εγκατεστημένο το JDK 6 ή 7 τότε πρέπει να την ορίσουμε μέσα από το αρχείο:

```
JAVA_HOME=”${path to java}"  (GNU/Linux)

set "JAVA_HOME=${path to java}" (Windows)
```

Πηγαίνουμε στον φάκελο ${JBOSS_HOME}/standalone/configuration και επεξεργαζόμαστε το περιεχόμενο του αρχείου standalone-full.xml όπως φαίνεται παρακάτω:

```
{…}
<extension module="org.jboss.as.weld"/>
    </extensions>

    <system-properties>
        <property name="SERVER_PATH" value="/home/development/jboss-as-7.1.1.Final/"/>
    </system-properties>

    <management>
        <security-realms>
{…}
```

Δείγμα Κώδικα 14: Δήλωση της τοποθεσίας που εγκαταστημένος ο JBoss AS στο αρχείο standalone-full.xml

Η παραπάνω αλλαγή απαιτείται για να ορίσουμε ως μεταβλητή συστήματος την τοποθεσία του που είναι εγκατεστημένος ο JBoss Application Server. Αυτή η μεταβλητή συστήματος χρειάζεται από την εφαρμογή για να γνωρίζει που είναι εγκατεστημένος ο JBoss Application Server.

```
{…}
<subsystem xmlns="urn:jboss:domain:datasources:1.0">
            <datasources>
                <datasource jndi-name="java:jboss/datasources/ExampleDS" pool-name="ExampleDS"
enabled="true" use-java-context="true">
                    <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-1</connection-url>
                    <driver>h2</driver>
                    <security>
                        <user-name>sa</user-name>
                        <password>sa</password>
                    </security>
                </datasource>
```

```
                <datasource jta="true" jndi-name="java:/LoggerDS" pool-name="LoggerDS"
enabled="true" use-java-context="true" use-ccm="true">
                    <connection-
url>jdbc:mysql://localhost:${port}/logger_db?useUnicode=true&amp;characterEncoding=UTF-
8&amp;character_set_server=utf8mb4</connection-url>
                    <driver>mysql</driver>
                    <security>
                        <user-name>${user name}</user-name>
                        <password>${password}</password>
                    </security>
                    <statement>
                        <prepared-statement-cache-size>100</prepared-statement-cache-size>
                        <share-prepared-statements>true</share-prepared-statements>
                    </statement>
                </datasource>
                <drivers>
                    <driver name="mysql" module="com.mysql">
                        <driver-class>com.mysql.jdbc.Driver</driver-class>
                    </driver>
                    <driver name="h2" module="com.h2database.h2">
                        <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>
                    </driver>
                </drivers>
            </datasources>
        </subsystem>
{…}
```

Δείγμα Κώδικα 15: Εισαγωγή παραμέτρων στο αρχείο standalone-full.xml για την σύνδεση του JBoss με την βάση δεδομένων

Η παραπάνω αλλαγή απαιτείται για την σύνδεση της εφαρμογής με την βάση δεδομένων
Στο ${port} μπαίνει ο αριθμός της πόρτας στην οποία τρέχει ο MySQL Server (συνήθως 3306)

Στο ${user name} μπαίνει το όνομα χρήστη της βάσης δεδομένων
Στο ${password} μπαίνει ο κωδικός για τον επιλεγμένο χρήστη

```
{…}
<subsystem xmlns="urn:jboss:domain:web:1.1" default-virtual-server="default-host" native="false">
   <connector name="https" protocol="HTTP/1.1" scheme="https" socket-binding="https"
secure="true">
      <ssl name="ext-ssl" key-alias="${ssl cert alias}" password="${certificate password}"
certificate-key-file="${path to keystore}"/>
   </connector>
   <connector name="http" protocol="HTTP/1.1" scheme="http" socket-binding="http" redirect-
port="8443"/>
    <virtual-server name="default-host" enable-welcome-root="true">
       <alias name="localhost"/>
       <alias name="example.com"/>
    </virtual-server>
</subsystem>
{…}
```

Δείγμα Κώδικα 16: Παραμετροποίηση του αρχείου standalone-full.xml για την ενεργοποίηση HTTPS σύνδεσης

Η παραπάνω αλλαγή απαιτείται για την ύπαρξη HTTPS σύνδεσης με την εφαρμογή τόσο για την ασφαλή αποστολή δεδομένων από την εφαρμογή πελάτη όσο και για την ασφαλή χρήση της εφαρμογής

Στο ${ssl cert alias} μπαίνει το όνομα του alias με το οποίο είναι αποθηκευμένο το ssl certificate στο keystore
Στο ${certificate password} μπαίνει ο κωδικός του ιδιωτικού κλειδιού του ssl certificate (πρέπει να είναι ίδιος με τον κωδικό του keystore)
Στο ${path to keystore} μπαίνει η θέση του keystore

```
{…}
<interfaces>
        <interface name="management">
            <inet-address value="${jboss.bind.address.management:0.0.0.0}"/>
        </interface>
        <interface name="public">
            <inet-address value="${jboss.bind.address:0.0.0.0}"/>
        </interface>
        <interface name="unsecure">
            <inet-address value="${jboss.bind.address.unsecure:0.0.0.0}"/>
        </interface>
    </interfaces>
{…}
```

Δείγμα Κώδικα 17: Παραμετροποίηση του αρχείου standalone-full.xml για την ενεργοποίηση απομακρυσμένης σύνδεσης

Η προκαθορισμένη (default) συμπεριφορά του JBoss επιτρέπει συνδέσεις μόνο μέσω της διεύθυνσης 127.0.0.1 (Loopback IP), δηλαδή ο χρήστης μπορεί να συνδεθεί στην εφαρμογή ή στην διαχειριστική κονσόλα μόνο αν χρησιμοποιεί τον υπολογιστή που είναι εγκατεστημένος ο JBoss. Στο Δείγμα Κώδικα 17 φαίνεται η αλλαγή που απαιτείται για να είναι δυνατή η πρόσβαση και απομακρυσμένα.

6.4 Εκκίνηση του JBoss Application Server

Από τον φάκελο ${JBOSS_HOME}/bin εκτελούμε το **standalone.bat** για Windows ή το **standalone.sh** για GNU/Linux.

6.5 Δημιουργία χρήστη και πρόσβαση στην διαχειριστική κονσόλα του JBoss

Από τον φάκελο ${JBOSS_HOME}/bin εκτελούμε το add-user.bat για Windows ή το add-user.sh για GNU/Linux και εισάγουμε τα στοιχεία του νέου χρήστη όπως φαίνεται στην Εικόνα 4 (στο συγκεκριμένο παράδειγμα έχουμε επιλέξει για όνομα χρήστη το loggerAdmin ωστόσο μπορεί να επιλεγεί οποιοδήποτε αλφαριθμητικό όνομα χρήστη).

```
What type of user do you wish to add?
 a) Management User (mgmt-users.properties)
 b) Application User (application-users.properties)
(a): a

Enter the details of the new user to add.
Realm (ManagementRealm) :
Username : loggerAdmin
Password :
Re-enter Password :
About to add user 'loggerAdmin' for realm 'ManagementRealm'
Is this correct yes/no? yes
```

Εικόνα 4: Παράδειγμα εισαγωγής χρήστη για την πρόσβαση στην διαχειριστικη κονσόλα του JBoss

Η διαχειριστική κονσόλα τρέχει στην πόρτα 9990 άρα εφόσον έχουμε ξεκινήσει τον JBoss Application Server μπορούμε να έχουμε πρόσβαση στη διαχειριστική κονσόλα χρησιμοποιώντας την IP του μηχανήματος (π.χ http://23.12.123.67:9990) ή αν είμαστε τοπικά συνδεδεμένοι στο μηχάνημα χρησιμοποιώντας την διεύθυνση http://localhost:9990. Εισάγοντας το όνομα και τον κωδικό του χρήστη ,που δημιουργήσαμε, έχουμε πρόσβαση στη διαχειριστική κονσόλα του JBoss.

7 Τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής Logger Server

7.1 Βασικές πληροφορίες της τεχνολογίας JSF

Η τεχνολογία JavaServer Faces (JSF) αποτελεί μια δομή της Java για κατασκευή εφαρμογών ιστού (Web applications). Απλοποιεί την δημιουργία κώδικα καθώς κάνει εύκολη και γρήγορη την κατασκευή εφαρμογών ενώνοντας επαναχρησιμοποιήσιμα στοιχεία UI (User interface) σε μια σελίδα και συνδέοντας γεγονότα που δημιουργούνται στην πλευρά του πελάτη με χειριστές γεγονότων στην πλευρά του server.
Ένα από τα πλεονεκτήματα του JSF είναι ότι αποτελεί ταυτόχρονα ένα πρότυπο Java Web UI καθώς και μια δομή που ακολουθεί το σχεδιαστικό μόρφημα Model-View-Controller(MVC). Αυτό κάνει τις εφαρμογές JSF πολύ πιο εύχρηστες λόγω του ότι το ο κώδικας UI (**View**) είναι απόλυτα διαχωρισμένος από τα δεδομένα και την λογική της εφαρμογής (**Model**). Για να παρέχει πρόσβαση στα δεδομένα της εφαρμογής από τις σελίδες και να αποτρέψει την μη εξουσιοδοτημένη ή λανθασμένη πρόσβαση σε αυτές, όλες οι αλληλεπιδράσεις του χρήστη με την εφαρμογή χειρίζονται από έναν ελεγκτή που παρέχει το JSF (**Controller**).
Ο JSF ελεγκτής εξυπηρετεί ως η σύνδεση μεταξύ του χρήστη και της εφαρμογής. Λειτουργεί μέσα στα όρια ενός καλά καθορισμένου κύκλου ζωής (JSF Lifecycle) ο οποίος υπαγορεύει ολόκληρη τη ροή των γεγονότων μεταξύ των αιτημάτων των χρηστών. Για παράδειγμα κατά την αποστολή ενός αρχικού αιτήματος για την πρόσβαση σε μια εφαρμογή JSF, ο ελεγκτής χειρίζεται το αίτημα προετοιμάζοντας το JSF context (το οποίο είναι ένα αντικείμενο Java που κρατάει όλα τα δεδομένα της εφαρμογής). Ο ελεγκτής τότε δρομολογεί τον χρήστη στην αιτούμενη σελίδα. Η σελίδα παίρνει δεδομένα από το JSF context χρησιμοποιώντας ένα απλό συντακτικό που ονομάζεται Expression Language (EL). Σε επόμενο αίτημα ο ελεγκτής ενημερώνει τα δεδομένα εφαρμογής εφόσον ο χρήστης έχει εισάγει νέα στοιχεία. Ο προγραμματιστής έχει πρόσβαση σε ολόκληρο τον JSF Lifecycle οποιαδήποτε στιγμή κατά την εκτέλεση παρέχοντας υψηλό βαθμό ελέγχου της συμπεριφοράς της εφαρμογής ανά πάσα στιγμή.

7.2 Βασικές πληροφορίες της τεχνολογίας CDI

Η τεχνολογία Context and Dependency Injection (CDI) διευκολύνει και απλοποιεί την δημιουργία μιας JSF εφαρμογής. Το Dependency Injection είναι μια τεχνική στην οποία ένα αντικείμενο περιέχει τις εξαρτήσεις ενός άλλου αντικειμένου. Μια εξάρτηση (dependency) είναι ένα αντικείμενο που παρέχει δεδομένα ή υπηρεσίες που μπορούν να χρησιμοποιηθούν. Η ένεση (injection) είναι το πέρασμα μιας εξάρτησης σε ένα εξαρτώμενο αντικείμενο το οποίο μπορεί να το χρησιμοποιήσει. Η μετάδοση αυτής της εξάρτησης στο

αντικείμενο αντί να επιτραπεί στο αντικείμενο να κατασκευάσει ή να βρει την συγκεκριμένη εξάρτηση αποτελεί την θεμελιώδη απαίτηση αυτού του σχεδιαστικού μορφήματος.
Για παράδειγμα έστω ότι ο χρήστης έχει συμπληρώσει το όνομα χρήστη και τον κωδικό στην σελίδα εισόδου στην εφαρμογή. Στον κώδικα έχουμε μια κλάση που ελέγχει την πρόσβαση και περιέχει μια μέθοδο η οποία αναζητεί για το όνομα χρήστη στην βάση δεδομένων και συγκρίνει τους κωδικούς. Ένας συνηθισμένος τρόπος να γίνει αυτό θα ήταν να ψάξει για όνομα χρήστη-κωδικό που πληκτρολόγησε ο χρήστης στην συγκεκριμένη συνεδρία (session) και στην συνέχεια να δημιουργήσει μια νέα σύνδεση στην βάση δεδομένων για να κάνει τον έλεγχο. Ωστόσο χρησιμοποιώντας το CDI μπορούμε να επιτύχουμε το ίδιο πιο εύκολα και πιο απλά. Τα δεδομένα ονόματος χρήστη και κωδικός που πληκτρολόγησε περιέχονται σε ένα αντικείμενο επιπλέον υπάρχει ένα αντικείμενο που παρέχει την υπηρεσία για σύνδεση στην βάση δεδομένων. Εισάγοντας στην κλάση που κάνει τον έλεγχο τις εξαρτήσεις των παραπάνω δύο αντικείμενων επιτυγχάνεται ο έλεγχος με ένα πιο καθαρό και διατηρήσιμο κώδικα.

7.3 Βασικές πληροφορίες της τεχνολογίας JPA

Το Java Persistence API (JPA) παρέχει μια αντικειμενική / σχεσιακή αντιστοίχιση για την διαχείριση σχεσιακών δεδομένων στις εφαρμογές Java. Ένας πίνακας μιας σχεσιακής βάσης δεδομένων μπορεί να αναπαρίσταται από ένα Java αντικείμενο που ονομάζεται οντότητα (entity). Η κατάσταση μιας οντότητας εκπροσωπείται από επίμονα (persistent) πεδία. Αυτά τα πεδία χρησιμοποιούν αντικειμενική / σχεσιακή αντιστοίχιση για να αντιστοιχιστούν οι οντότητες και οι σχέσεις με οντότητες με σχεσιακά δεδομένα της βάση δεδομένων. Για την διαχείριση αυτών των οντοτήτων υπάρχει ο διευθυντής οντοτήτων (entity manager). Σκοπός του είναι η παροχή μεθόδων για την δημιουργία, τροποποίηση και διαγραφή αυτών των οντοτήτων.
Στην εφαρμογή μας χρησιμοποιήσαμε την βιβλιοθήκη Hibernate η οποία παρέχει μια πιο ευέλικτη υλοποίηση του JPA που παρέχει η Java.

7.4 Βασικές πληροφορίες της τεχνολογίας RESTful Web Service

Το RESTful Web Service βασίζεται στην τεχνολογία REST (REpresentational State Transfer) και αποτελεί μια προσέγγιση στις επικοινωνίες που συχνά χρησιμοποιούνται στην δημιουργία υπηρεσιών δικτύου (Web services). Αποτελεί μια διεπαφή που χρησιμοποιείται από εφαρμογές για την δημιουργία HTTP/HTTPS αιτημάτων τύπου GET, POST, PUT, DELETE. Μια διεπαφή που ακολουθεί τις αρχές του REST δεν απαιτεί από τον πελάτη να γνωρίζει τίποτα σχετικά με την δομή της συγκεκριμένης διεπαφής.
Η τεχνολογία REST προτιμάται σε σχέση με το πιο περίπλοκο Simple Object Access Protocol (SOAP) επειδή το REST χρησιμοποιεί μικρότερο εύρος ζώνης (bandwidth), κάνοντάς το πιο κατάλληλο για χρήση του internet.

Η πληροφορία που μεταφέρεται μεταξύ πελάτη – εξυπηρετητή χρησιμοποιώντας REST μπορεί να είναι σχεδόν οποιαδήποτε τύπου, από απλό κείμενο μέχρι μια ακολουθία από bytes που περιγράφουν ένα αρχείο, συνήθως όμως χρησιμοποιείται το συντακτικό JSON για την περιγραφή του αντικειμένου που θέλουμε να στείλουμε.
Στην εφαρμογή μας χρησιμοποιούνται RESTful Web Services για να αποσταλθεί η καταγεγραμμένη πληροφορία από την εφαρμογή πελάτη στον server, για την ενημέρωση της εφαρμογής πελάτη για κάποια νεότερη έκδοση, καθώς και για την χρήση της υπηρεσίας Mozilla Location Service από τον server.

7.5 Αρχεία JAR, WAR και EAR

Στις εφαρμογές JAVA Enterprise Edition (JEE), ενότητες (modules) συναθροίζονται σε αρχεία EAR, JAR και WAR ανάλογα με την λειτουργικότητά τους.

- JAR (Java ARchive) περιέχουν συνήθως ένα σύνολο απλών JAVA κλάσεων (POJO).
- WAR (Web Application aRchive) εκτός από απλές JAVA κλάσεις περιέχουν επιπλέον και JAVA Servlets, JAVA Beans, JAVA Server Pages, αρχεία XML, αρχεία HTML, αρχεία XHTML, αρχεία JavaScript, αρχεία CSS τα οποία συνολικά συγκροτούν μια εφαρμογή web.
- EAR (Enterprise Application aRchive) συναθροίζουν ένα σύνολο αρχείων JARs και WARs σε ένα αρχείο εξασφαλίζοντας ότι η εγκατάσταση των περιεχόμενων ενοτήτων σε ένα Application Server θα γίνει ταυτόχρονα.

7.6 Βασικές πληροφορίες του εργαλείου Maven

Το maven αποτελεί ένα εργαλείο διαχείρισης project. Η λειτουργία του περιγράφεται μέσα σε ένα αρχείο xml που ονομάζεται pom.xml και βρίσκεται στον φάκελο του project. Μέσα σε αυτό το αρχείο περιγράφονται οι παρακάτω δυο βασικές λειτουργίες:
α) Κατεβάζει τοπικά στο μηχάνημα όλες τις εξωτερικές βιβλιοθήκες που χρειάζεται το project. Οι βιβλιοθήκες που χρειάζονται πρέπει να περιγράφονται κατάλληλα όπως φαίνεται στο Δείγμα Κώδικα 18

```
<dependency>
	<groupId>javax.enterprise</groupId>
	<artifactId>cdi-api</artifactId>
	<version>1.2</version>
</dependency>
```

Δείγμα Κώδικα 18: Δήλωση εξωτερικής βιβλιοθήκης στο αρχείο pom.xml

β) Μεταγλωττίζει τον κώδικα και τον συναθροίζει σε ένα αρχείο ear μαζί με όλες τις βιβλιοθήκες που χρειάζεται. Στο Δείγμα Κώδικα 19 φαίνεται η παραμετροποίηση για την δημιουργία του αρχείου ear της εφαρμογής μας,

```
<build>
	<finalName>loggerServer</finalName>
```

```
        <plugins>
                <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-ear-plugin</artifactId>
                        <version>2.6</version>
                        <configuration>
                                <version>6</version>
                                <displayName>LoggerServer</displayName>
                                <defaultLibBundleDir>lib</defaultLibBundleDir>
                                <applicationName>LoggerServer</applicationName>
                                <modules>
                                        <webModule>
                                                <groupId>com.ntua.ote</groupId>
                                                <artifactId>web</artifactId>
                                                <contextRoot>/logger/server</contextRoot>
                                        </webModule>
                                </modules>
                                <fileNameMapping>no-version</fileNameMapping>
                        </configuration>
                </plugin>
        </plugins>
</build>
```

Δείγμα Κώδικα 19: Παραμετροποίηση για την δημιουργία του αρχείου ear μέσα από το pom.xml

Μέσω του Maven μπορούν να εκτελεστούν και βοηθητικές λειτουργίες όπως η απόκτηση του πηγαίου κώδικα και των Java Docs των εξωτερικών βιβλιοθηκών.

8 Εγκατάσταση της εφαρμογής Logger Server

8.1 Απόκτηση του κώδικα και δημιουργία του αρχείου ear της εφαρμογής Logger Server

Ο κώδικας της εφαρμογής βρίσκεται στο GitHub. Για να κατεβάσουμε την εφαρμογή απαιτείται ένας git client που μπορεί να εγκατασταθεί από την διεύθυνση (4) ή στην περίπτωση GNU/Linux και μέσω της εντολής **apt-get install git**. Έχοντας κατεβάσει έναν git client κατεβάζουμε τον κώδικα από το (10). Για την εγκατάσταση της εφαρμογής χρειάζεται το apache (11).
Έχοντας κατεβάσει τον κώδικα, χρησιμοποιώντας τον git client που κατεβάσαμε πηγαίνουμε στον φάκελο logger-server, ανοίγουμε ένα cmd/terminal και εκτελούμε την εντολή **mvn clean package.** Αυτή η εντολή θα μεταγλωττίσει και θα πακετάρει τον κώδικα της εφαρμογής στο αρχείο logger-server/ear/target/loggerServer.ear.

Αν θέλουμε να δούμε ή να τροποποιήσουμε τον κώδικα της εφαρμογής τότε ο ευκολότερος τρόπος είναι να χρησιμοποιήσουμε ένα ολοκληρωμένο περιβάλλον ανάπτυξης (Integrated Development Environment - IDE), όπως NetBeans, IntelliJ, Eclipse. Παρακάτω περιγράφονται τα βήματα για την εισαγωγή και την εκτέλεση του κώδικα της εφαρμογής σε ένα περιβάλλον Eclipse και συγκεκριμένα στο Red Hat JBoss Developer Studio 10.3.0.
Μέσα από τον Eclipse πατάμε στο File>Import και επιλέγουμε από τον φάκελο Maven το “Existing Maven Projects”. Από το κουμπί Browse διαλέγουμε τον φάκελο logger-server, που κατεβάσαμε προηγουμένως με το git, και πατάμε Finish. Ο κώδικας τώρα φαίνεται στο αριστερό τμήμα του Eclipse (Project Explorer) και από εκεί μπορούμε να δούμε όλα τα αρχεία του κώδικα της εφαρμογής και επιλέγοντας οποιοδήποτε από αυτά μπορούμε να το τροποποιήσουμε.
Το project μας είναι δομημένο σε τέσσερα υπο-projects persistence, web, core και ear.

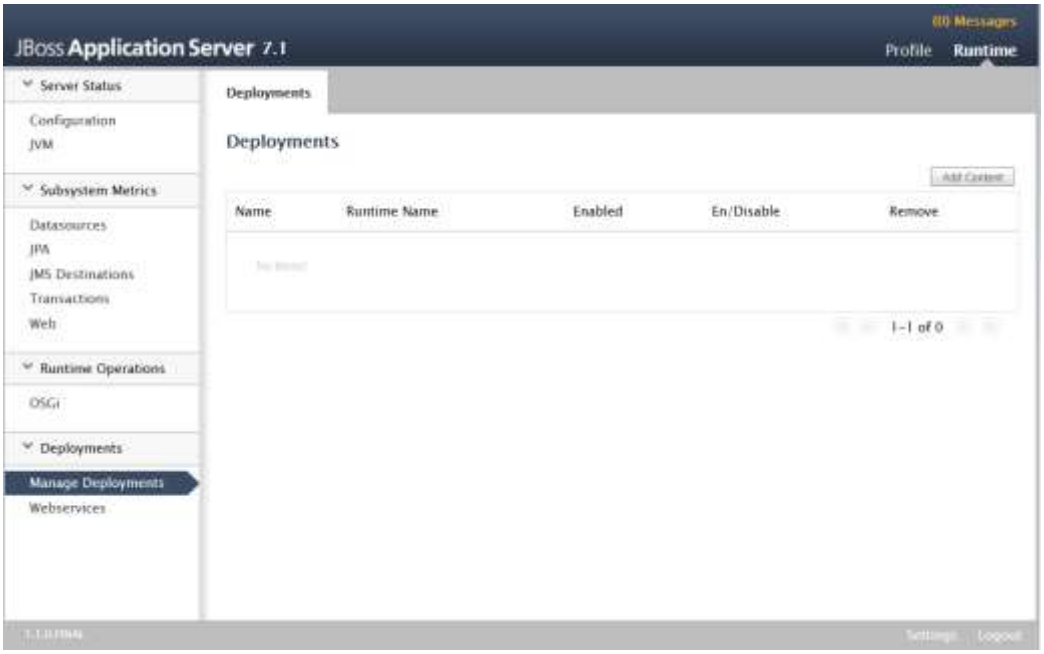
- Το persistence περιέχει τον κώδικα που σχετίζεται με την εισαγωγή, ενημέρωση και ανάκτηση πληροφορίας από την βάση δεδομένων. Το σύνολο των περιεχομένων του συναθροίζεται σε ένα αρχείο JAR.
- Το web περιέχει το κύριο κώδικα της εφαρμογής καθώς περιλαμβάνει τον κώδικα για την απεικόνιση και λογική των λειτουργιών. Επιπλέον περιέχει τον κώδικα για την αλληλεπίδραση με εξωτερικές υπηρεσίες μέσω REST. Το σύνολο των περιεχομένων του συναθροίζεται σε ένα αρχείο WAR.
- Το core περιέχει κλάσεις που χρησιμοποιούνται από κοινού τόσο από το web όσο και από το persistence. Το σύνολο των περιεχομένων του συναθροίζεται σε ένα αρχείο JAR.
- Το ear συναθροίζει τα τρία παραπάνω αρχεία σε ένα αρχείο EAR βάση των παραμέτρων που έχουν οριστεί στα αρχεία pom.xml και src/main/application/META-INF/jboss-deployment-structure.xml.

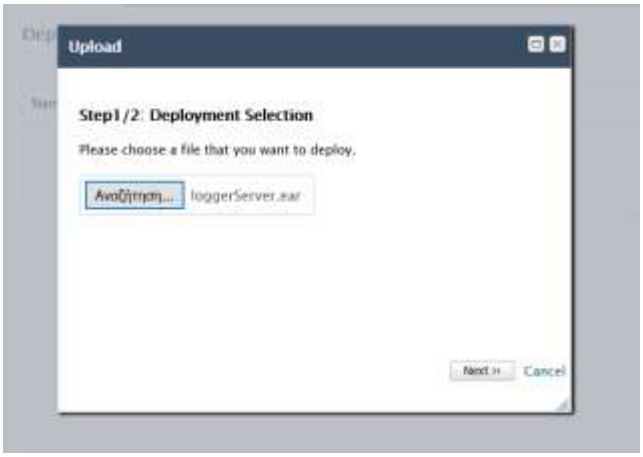
Για να δημιουργήσουμε το αρχείο ear της εφαρμογή πατάμε δεξί click στο logger-server στον Project Explorer και επιλέγουμε **Run As>Maven Build**. Στο παράθυρο που ανοίγει στο πεδίο Goals γράφουμε **clean package** και πατάμε **Run**. Ο Eclipse μεταγλωτίζει τον κώδικα χρησιμοποιώντας το εργαλείο maven και δημιουργεί το αρχείο loggerServer.ear στον φάκελο logger-server/ear/target/.

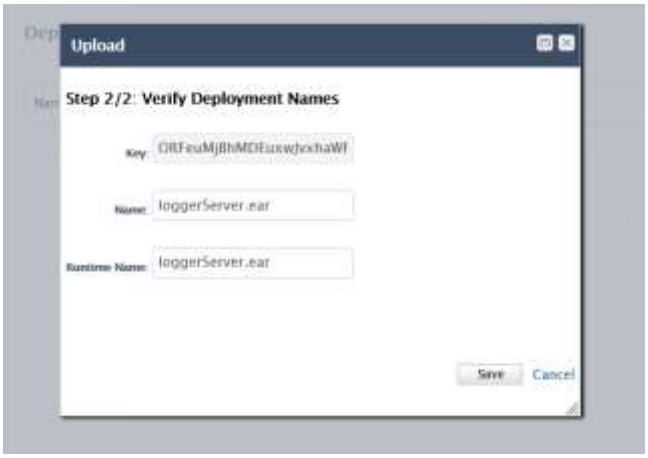
8.2 Εγκατάσταση της εφαρμογής Logger Server στον JBoss

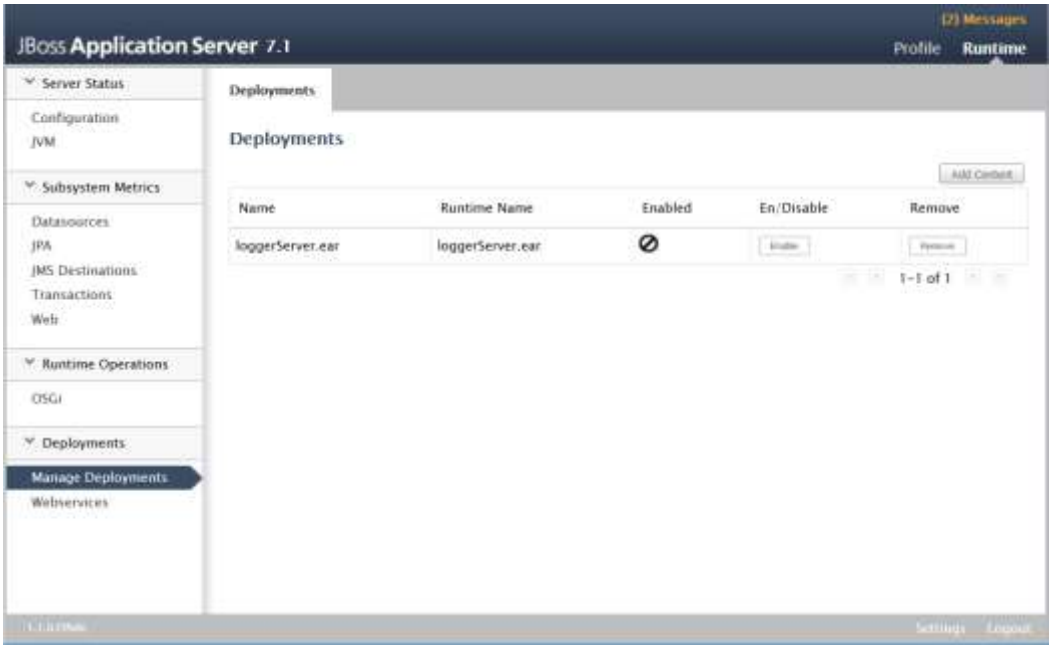
Υπάρχουν δύο τρόποι για να εγκαταστήσουμε την εφαρμογή στον JBoss.

Α) Μέσω της διαχειριστικής κονσόλας: Έχοντας συνδεθεί στην διαχειριστική κονσόλα επιλέγουμε στο αριστερό μενού το Deployments>Manage Deployments. Πατάμε το κουμπί Add Content επιλέγουμε το loggerServer.ear και πατάμε Next και Save. Περιμένουμε μέχρι να ολοκληρωθεί η μεταφορά του στον server και πατάμε το κουμπί Enable. Η Εικόνα 5 απεικονίζει τα παραπάνω βήματα.


JBoss Application Server 7.1
Profile
Runtime
Server Status
Configuration
JVM
Subsystem Metrics
Datasources
JPA
JMS Destinations
Transactions
Web
Runtime Operations
OSGi
Deployments
Manage Deployments
Webservices
Deployments
Deployments
Name
Runtime Name
Enabled
En/Disable
Remove
1-1 of 0


Upload
Step1/2: Deployment Selection
Please choose a file that you want to deploy.
loggerServer.ear
Next »
Cancel


Upload
Step 2/2: Verify Deployment Names
Key:
Name:
loggerServer.ear
Runtime Name:
loggerServer.ear
Save
Cancel



Εικόνα 5: Απεικόνιση των βημάτων για την εγκατάσταση της εφαρμογής Logger Server στον JBoss AS 7

Β) Από γραμμή εντολών: Τρέχουμε τις εντολές που φαίνονται στο

```
cd ${JBOSS_HOME}
./jboss-cli.sh (για GNU/Linux) jboss-cli.bat (για Windows)
connect
deploy ${path to loggerServer.ear}
```

Δείγμα Κώδικα 20: Σύνολο εντολών για την εγκατάσταση του αρχείου ear της εφαρμογής Logger Server στον JBoss AS 7 μέσω cmd/terminal

Η πρώτη μέθοδος είναι πιο εύκολη και φιλική στον χρήστη ωστόσο είναι δυνατόν να γίνει μόνο όταν ο χρήστης είναι τοπικά συνδεδεμένος στο μηχάνημα που τρέχει ο JBoss Application Server. Όταν ο χρήστης είναι συνδεδεμένος από απόσταση πρέπει να ακολουθήσει υποχρεωτικά την δεύτερη μέθοδο με την προϋπόθεση ότι έχει μεταφέρει πρώτα το αρχείο loggerServer.ear σε κάποιον φάκελο του απομακρυσμένου μηχανήματος.

9 Βασικές κλάσεις της εφαρμογής Logger Server

Σε αυτήν την ενότητα περιφράφεται η λειτουργικότητα που υλοποιούν οι κυριότερες κλάσεις της εφαρμογής Logger Server. Συγκεκριμένα περιγράφεται η λειτουργικότητα της κάθε κλάσης και παρουσιάζεται ο κώδικας της με σχόλια.

9.1 Οι κλάσεις com.ntua.ote.logger.web.LoginController και com.ntua.ote.logger.web.service.ApplicationFilter

Κατά την είσοδο στην εφαρμογή ο χρήστη καλείται να εισάγει όνομα χρήστη και κωδικό. Η κλάση LoginController (Δείγμα Κώδικα 21) ελέγχει μέσω της βάσης δεδομένων αν αυτά τα στοιχεία είναι έγκυρα. Στην περίπτωση που τα στοιχεία είναι έγκυρα αποθηκεύει στην συνεδρία (session) του χρήστη ότι συνδέθηκε επιτυχώς και του εμφανίζει την αρχική σελίδα.
Σε κάθε προσπάθεια του χρήστη να μεταβεί σε οποιαδήποτε σελίδα της εφαρμογής καλείται η κλάση ApplicationFilter η οποία ελέγχει, από την συνεδρία, αν ο χρήστης έχει συνδεθεί επιτυχώς. Αν έχει συνδεθεί επιτυχώς τότε εμφανίζεται η επιθυμητή σελίδα αλλιώς ο χρήστης μεταβαίνει στην σελίδα εισόδου.

```
package com.ntua.ote.logger.web;

import …;

@Named
@SessionScoped
public class LoginController implements Serializable {

	/** The Constant serialVersionUID. */
	private static final long serialVersionUID = 1L;

	/** The user name. */
	private String userName;

	/** The password. */
	private String password;

	/** The user session bean. */
	@Inject
	private UserSessionBean userSessionBean;

	/** The logger DAO impl. */
	@Inject
	private LoggerDAO loggerDAOImpl;

	/**
	 * Compare the login details username/password with
	 * the database. Redirects user to home page upon a successful login
	 *
```

```
         * @return navigation page string
         */
        public String login(){
                if(loggerDAOImpl.login(userName, password)) {
                        userSessionBean.setAuthenticated(true);
                        userSessionBean.setUserName(userName);
                        return "home";
                } else {
                        FacesUtil.addErrorMessage(FacesUtil.getMessage("error.authentication"), null, false);
                        return null;
                }
        }

        /**
         * Invalidates the session and redirect the user to login page
         *
         * @return navigation page string
         */
        public String logout(){
                FacesContext.getCurrentInstance().getExternalContext().invalidateSession();
                return "home";
        }

        public String getUserName() {
                return userName;
        }

        public void setUserName(String userName) {
                this.userName = userName;
        }

        public String getPassword() {
                return password;
        }

        public void setPassword(String password) {
                this.password = password;
        }

}
```

Δείγμα Κώδικα 21: Η κλάση com.ntua.ote.logger.web.LoginController

```
package com.ntua.ote.logger.web.service;

import …;

@WebFilter(urlPatterns = { "/views/*", "/index.xhtml"})
public class ApplicationFilter implements Filter {

        /** The Constant LOGGER. */
        private static final Logger LOGGER = Logger
                        .getLogger(ApplicationFilter.class);

        @Inject
        private UserSessionBean userSessionBean;

        /**
     * If in the user is already authenticated then the request is forwarded
     * else the request is dropped and the user is redirected in the login page
     *
         * @see javax.servlet.Filter#doFilter(javax.servlet.ServletRequest, javax.servlet.ServletResponse,
javax.servlet.FilterChain)
         */
        @Override
        public void doFilter(ServletRequest request, ServletResponse response,
                        FilterChain chain) throws ServletException, IOException {
```

```
                HttpServletRequest req = (HttpServletRequest) request;
                if (userSessionBean.isAuthenticated()) {
                        chain.doFilter(request, response);
                } else {
                        HttpServletResponse res = (HttpServletResponse) response;
                        res.sendRedirect(req.getContextPath() + "/login.xhtml");
                }
        }

        @Override
        public void destroy() {

        }

        @Override
        public void init(FilterConfig filterConfig) throws ServletException {
        }
}
```

Δείγμα Κώδικα 22: Η κλάση com.ntua.ote.logger.web.service.ApplicationFilter

9.2 Η κλάση com.ntua.ote.logger.web.MapController

Η κλάση MapController ελέγχει τις λειτουργικότητες της σελίδας «Αναζήτηση στον χάρτη». Συλλέγει τα κριτήρια που εισήγαγε ο χρήστης, καλεί την βάση δεδομένων και εμφανίζει το αποτελέσματα στον χρήστη βάση των λειτουργικών απαιτήσεων 2.2.7.

```
package com.ntua.ote.logger.web;

import …;

@Named
@SessionScoped
public class MapController implements Serializable {

        private static final long serialVersionUID = 2208902766742258519L;
        private static final Logger LOGGER = Logger.getLogger(MapController.class);

        @Inject
        private LoggerDAOImpl dao;

        private SearchCriteria searchCriteria;
        private List<LogDetails> logs;
        private Map<Long, Marker> markersMap;
        private MapModel advancedModel;
        private Marker marker;
        private String radius;
        private Marker gridMarker;
        private String mapCenter;
        private LogDetails selectedLog;
        private String latitude;
        private String longitude;
        private boolean error;
        private boolean collapsed;

        /** initialize the variables of the controller
         */
        public String init(){
                searchCriteria = new SearchCriteria();
                logs = null;
                mapCenter="0,0";
```

```
                advancedModel = new DefaultMapModel();
                collapsed = false;
                return "mapSearch";
        }

        /** Invoked on clicking search. Get the results from DB based on the criteria.
            Creates the markers on the map for the entries that have a location recorded.
            Sets the center of the map in the latest entry */
        public void search(){
                error = false;
                if(validation(searchCriteria)) {
                        logs = dao.search(searchCriteria);
                        boolean add = false;
                        markersMap = new HashMap<>();
                        advancedModel = new DefaultMapModel();
                        if(!logs.isEmpty()) {
                                String contextPath =
FacesContext.getCurrentInstance().getExternalContext().getRequestContextPath();
                                for(LogDetails log : logs) {
                                        if(Utils.hasLocation(log)) {
                                                LatLng loc = new LatLng(log.getLatitude(),
log.getLongitude());
                                                Marker marker = new Marker(loc, Utils.getMarkerTitle(log),
log, contextPath + Constants.BLUE_MARKER_ICON);
                                                advancedModel.addOverlay(marker);
                                                markersMap.put(log.getId(), marker);
                                                if(!add) {
                                                        mapCenter = "" + log.getLatitude() +"," +
log.getLongitude();
                                                        add = true;
                                                }
                                        }
                                }
                        } else {
                                FacesUtil.addInfoMessage(FacesUtil.getMessage("error.no.records"), null,
false);
                                error = true;
                        }
                }
                searchCriteria.setId(0);
        }

        /** Criteria Validation. Date from cannot be after Date to */
        private boolean validation(SearchCriteria searchCriteria){
                if(searchCriteria.getDateFrom() != null && searchCriteria.getDateTo() != null
                                && searchCriteria.getDateFrom().after(searchCriteria.getDateTo())) {
                        FacesUtil.addErrorMessage(FacesUtil.getMessage("error.date"), null, false);
                        error = true;
                        return false;
                }
                return true;
        }

        public void reset(){
                logs = null;
                searchCriteria = new SearchCriteria();
        }

        /** Creates a XLS document containing the results */
        public void export(){
                try {
                        byte[] xls = XLSBuilder.exportXLS(logs, XLSMappings.logDetailsMapping,
XLSMappings.logDetailsHeader);
                        FacesUtil.download("logDetails.xls", xls);
                } catch (IOException e) {
                        LOGGER.error(e.getMessage());
                }
        }
```

```
        /** Invoked when a marker from the map has been selected */
        public void onMarkerSelect(OverlaySelectEvent event) {
        marker = (Marker) event.getOverlay();
        selectedLog = (LogDetails) marker.getData();
        latitude = "" + selectedLog.getLatitude();
        longitude = "" + selectedLog.getLongitude();
        radius = "" + selectedLog.getRadius();
    }

        /** Invoked when an entry from the list has been selected */
        public void selectMarker() {
                if(gridMarker != null) {
                        String contextPath =
FacesContext.getCurrentInstance().getExternalContext().getRequestContextPath();
                        gridMarker.setIcon(contextPath + Constants.BLUE_MARKER_ICON);
                }
        Marker marker = markersMap.get(selectedLog.getId());
                this.marker = marker;
                marker.setIcon("");
                gridMarker = marker;
                latitude = "" + selectedLog.getLatitude();
        longitude = "" + selectedLog.getLongitude();
        radius = "" + selectedLog.getRadius();
                mapCenter = marker.getLatlng().getLat()+","+marker.getLatlng().getLng();

    }

        public LogDetails getSelectedLog() {
                return selectedLog;
        }

        public void setSelectedLog(LogDetails selectedLog) {
                this.selectedLog = selectedLog;
        }

        public SearchCriteria getSearchCriteria() {
                return searchCriteria;
        }

        public void setSearchCriteria(SearchCriteria searchCriteria) {
                this.searchCriteria = searchCriteria;
        }

        public List<LogDetails> getLogs() {
                return logs;
        }

        public void setLogs(List<LogDetails> logs) {
                this.logs = logs;
        }

        public MapModel getAdvancedModel() {
                return advancedModel;
        }

        public void setAdvancedModel(MapModel advancedModel) {
                this.advancedModel = advancedModel;
        }

        public Marker getMarker() {
                return marker;
        }

        public void setMarker(Marker marker) {
                this.marker = marker;
        }

        public String getMapCenter() {
                return mapCenter;
```

```
	}

	public void setMapCenter(String mapCenter) {
		this.mapCenter = mapCenter;
	}

	public String getLatitude() {
		return latitude;
	}

	public void setLatitude(String latitude) {
		this.latitude = latitude;
	}

	public String getLongitude() {
		return longitude;
	}

	public void setLongitude(String longitude) {
		this.longitude = longitude;
	}

	public boolean isError() {
		return error;
	}

	public String getRadius() {
		return radius;
	}

	public void setRadius(String radius) {
		this.radius = radius;
	}

	public boolean isCollapsed() {
		return collapsed;
	}

	public void setCollapsed(boolean collapsed) {
		this.collapsed = collapsed;
	}

}
```

Δείγμα Κώδικα 23: Η κλάση com.ntua.ote.logger.web.MapController

9.3 Η κλάση com.ntua.ote.logger.web.MapPathController

Η κλάση MapPathController ελέγχει τις λειτουργικότητες της σελίδας «Αναζήτηση μονοπατιού». Συλλέγει τα κριτήρια που εισήγαγε ο χρήστης, καλεί την βάση δεδομένων και σχεδιάζει ένα μονοπάτι στον χάρτη βάση των αποτελεσμάτων όπως περιγράφεται και στις λειτουργικές απαιτήσεις 2.2.9.

```
package com.ntua.ote.logger.web;

import …;

@Named
@SessionScoped
public class MapPathController implements Serializable {

	private static final long serialVersionUID = 2208902766742258519L;
```

```
        @Inject
        private LoggerDAOImpl dao;

        private SearchCriteria searchCriteria;
        private List<LogDetails> logs;
        private MapModel advancedModel;
        private Marker marker;
        private String mapCenter;
        private LogDetails selectedLog;
        private String latitude;
        private String longitude;
        private ManyInputSelect inputs;
        private boolean error;

        /** initialize the variables of the controller */
        public String init(){
                inputs = new ManyInputSelect();
                searchCriteria = new SearchCriteria();
                logs = null;
                mapCenter="0,0";
                advancedModel = new DefaultMapModel();
                return "pathSearch";
        }

        /** Invoked on clicking search. Get the results from DB based on the criteria.
    Creates the markers on the map for the entries that have a location recorded and
    connects those markers with a line. Sets the center of the map in the latest entry */
        public void search(){
                inputs.add();
                error = false;
                if(validation(searchCriteria)) {
                        logs = new ArrayList<>();
                        advancedModel = new DefaultMapModel();
                        for(ManyInputSelect.ManyInputSelectItem item : inputs.getInputs()) {
                                searchCriteria.setPhoneNumber(item.getInput());
                                List<LogDetails> partialLogs = dao.searchPath(searchCriteria);

                                Polyline polyline = new Polyline();
                                int start = 0;
                                String contextPath =
FacesContext.getCurrentInstance().getExternalContext().getRequestContextPath();
                                if(partialLogs.size() > 1) {
                                        start = 1;
                                        LogDetails startLog = partialLogs.get(0);
                                        LatLng loc = new LatLng(startLog.getLatitude(),
startLog.getLongitude());
                                        advancedModel.addOverlay(new Marker(loc,
Utils.getMarkerTitle(startLog), startLog,
                                                        contextPath + Constants.BLUE_MARKER_START_ICON));

                                        LogDetails endLog = partialLogs.get(partialLogs.size() - 1);
                                        loc = new LatLng(endLog.getLatitude(), endLog.getLongitude());

                                        advancedModel.addOverlay(new Marker(loc,
Utils.getMarkerTitle(endLog), endLog,
                                                        contextPath + Constants.BLUE_MARKER_END_ICON));

                                }
                                for(int i = start; i < partialLogs.size() - start; i++) {
                                        LogDetails log = partialLogs.get(i);
                                        LatLng loc = new LatLng(log.getLatitude(), log.getLongitude());
                                        advancedModel.addOverlay(new Marker(loc, Utils.getMarkerTitle(log),
log,
                                                        contextPath + Constants.BLUE_MARKER_ICON));
                                }

                                if(partialLogs.size() > 1) {
                                        for(LogDetails log : partialLogs) {
```

```
                                                LatLng loc = new LatLng(log.getLatitude(),
log.getLongitude());
                                        polyline.getPaths().add(loc);
                                        polyline.setStrokeWeight(10);
                                        polyline.setStrokeColor(item.getColor());
                                        polyline.setStrokeOpacity(0.7);

                                                advancedModel.addOverlay(polyline);
                                        }
                                }
                                logs.addAll(partialLogs);
                        }
                        if(!logs.isEmpty()) {
                                mapCenter = logs.get(0).getLatitude() + "," + logs.get(0).getLongitude();
                        } else {
                                FacesUtil.addInfoMessage(FacesUtil.getMessage("error.no.records"), null,
false);
                                error = true;
                        }
                }
        }

        /** Criteria Validation. Date from cannot be after Date to */
        private boolean validation(SearchCriteria searchCriteria){
                if(searchCriteria.getDateFrom() != null && searchCriteria.getDateTo() != null
                                && searchCriteria.getDateFrom().after(searchCriteria.getDateTo())) {
                        FacesUtil.addErrorMessage(FacesUtil.getMessage("error.date"), null, false);
                        error = true;
                        return false;
                }
                return true;
        }

        public void reset(){
                inputs = new ManyInputSelect();
                logs = null;
                searchCriteria = new SearchCriteria();
        }

        /** Invoked when a marker from the map has been selected */
        public void onMarkerSelect(OverlaySelectEvent event) {
                if(event.getOverlay() instanceof Marker) {
                        marker = (Marker) event.getOverlay();
                        selectedLog = (LogDetails) marker.getData();
                        latitude = "" + selectedLog.getLatitude();
                        longitude = "" + selectedLog.getLongitude();
                } else {
                        selectedLog = null;
                }
    }

        /** Invoked when an entry from the list has been selected */
        public void selectMarker() {
                if(marker != null) {
                        marker.setIcon(Constants.BLUE_MARKER_ICON);
                }
        LatLng coord = new LatLng(selectedLog.getLatitude(), selectedLog.getLongitude());
        for(Marker marker : advancedModel.getMarkers()) {
        if(marker.getLatlng().equals(coord)) {
                this.marker = marker;
                marker.setIcon("");
                break;
        }
        }
    }

        public LogDetails getSelectedLog() {
                return selectedLog;
        }
```

```
	public void setSelectedLog(LogDetails selectedLog) {
		this.selectedLog = selectedLog;
	}

	public SearchCriteria getSearchCriteria() {
		return searchCriteria;
	}

	public void setSearchCriteria(SearchCriteria searchCriteria) {
		this.searchCriteria = searchCriteria;
	}

	public List<LogDetails> getLogs() {
		return logs;
	}

	public void setLogs(List<LogDetails> logs) {
		this.logs = logs;
	}

	public MapModel getAdvancedModel() {
		return advancedModel;
	}

	public void setAdvancedModel(MapModel advancedModel) {
		this.advancedModel = advancedModel;
	}

	public Marker getMarker() {
		return marker;
	}

	public void setMarker(Marker marker) {
		this.marker = marker;
	}

	public String getMapCenter() {
		return mapCenter;
	}

	public void setMapCenter(String mapCenter) {
		this.mapCenter = mapCenter;
	}

	public String getLatitude() {
		return latitude;
	}

	public void setLatitude(String latitude) {
		this.latitude = latitude;
	}

	public String getLongitude() {
		return longitude;
	}

	public void setLongitude(String longitude) {
		this.longitude = longitude;
	}

	public ManyInputSelect getInputs() {
		return inputs;
	}

	public void setInputs(ManyInputSelect inputs) {
		this.inputs = inputs;
	}
```

```
        public boolean isError() {
                return error;
        }
}
```

Δείγμα Κώδικα 24: Η κλάση com.ntua.ote.logger.web.MapPathController

9.4 Η κλάση com.ntua.ote.logger.web.RelationFinderController

Η κλάση RelationFinderController ελέγχει τις λειτουργικότητες της σελίδας «Εύρεση σχέσης». Συλλέγει τα κριτήρια που εισήγαγε ο χρήστης, καλεί την βάση δεδομένων και σχεδιάζει έναν γράφο που απεικονίζει τις σχέσης του αριθμού τηλεφώνου που αναζητήθηκε όπως περιγράφεται και στις λειτουργικές απαιτήσεις 2.2.10.

```
package com.ntua.ote.logger.web;

import …;

@Named
@SessionScoped
public class RelationFinderController implements Serializable {

        private static final long serialVersionUID = 2208902766742258519L;

        @Inject
        private LoggerDAOImpl dao;

        private DefaultDiagramModel model;
        private SearchCriteria searchCriteria;
        private Map<String, Element> elementMap;
        private Element selectedElement;
        private boolean error;

        /** initialize the variables of the controller */
        public String init() {
                searchCriteria = new SearchCriteria();
                model = null;
                return "relationFinder";
        }

        /** Invoked on clicking search. Get the results from DB based on the criteria as
        a list of Nodes. Each Node contains the MSISDN the level in which it should be printed in
        the graph and a Set with its children. Calls method constructTree to construct the graph */
        public void search() {
                error = false;
                if(validation(searchCriteria)) {
                        elementMap = new HashMap<>();
                        SearchResults results = dao.searchRelation(searchCriteria);
                        List<Node> nodes = results.getNodes();
                        if(!nodes.isEmpty() && nodes.size() > 1) {
                                if(StringUtils.hasLength(searchCriteria.getExternalPhoneNumber()) &&
!results.isRelationFound()) {
                                        FacesUtil.addInfoMessage(FacesUtil.getMessage("error.no.relation") +
searchCriteria.getPhoneNumber() +
                                                        " - " + searchCriteria.getExternalPhoneNumber(),
null, false);
                                        error = true;
                                        model = null;
                                } else {
                                        constructTree(results, searchCriteria.getExternalPhoneNumber());
                                }
```

```
                        } else {
                                FacesUtil.addInfoMessage(FacesUtil.getMessage("error.relation.no.records"),
null, false);
                                error = true;
                                model = null;
                        }
                }
        }

        /** Criteria Validation. Date from cannot be after Date to */
        private boolean validation(SearchCriteria searchCriteria){
                if(searchCriteria.getDateFrom() != null && searchCriteria.getDateTo() != null
                                && searchCriteria.getDateFrom().after(searchCriteria.getDateTo())) {
                        FacesUtil.addErrorMessage(FacesUtil.getMessage("error.date"), null, false);
                        error = true;
                        return false;
                }
                return true;
        }

        /** Constructs the graph. Iterates from the list of nodes and prints its node to the screen.
          Then iterates the list of list of the nodes in order to print the connections */
        private void constructTree(SearchResults results, String externalPhoneNumber){
                Map<String, Integer> arrayMap = new HashMap<>();
                List<Node> nodes = results.getNodes();
                model = new DefaultDiagramModel();
                model.setMaxConnections(-1);
                model.setConnectionsDetachable(false);

                Element root = new Element(searchCriteria.getPhoneNumber(), "50em", "2em");
                root.setId(searchCriteria.getPhoneNumber());
                model.addElement(root);
                elementMap.put(searchCriteria.getPhoneNumber(), root);
                int i=0, level=0;
                for(Node node : nodes) {
                        if(level < node.getLevel()) {
                                i = 0;
                                level = node.getLevel();
                        }
                        for(Node childNode : node.getChildren()) {
                                if(!elementMap.containsKey(childNode.getPhoneNumber())) {
                                        Element el = new Element(childNode.getPhoneNumber(), (2 + i*10)
+"em", 8 + level*6 + "em");
                                        el.setId(childNode.getPhoneNumber());
                                        elementMap.put(childNode.getPhoneNumber(), el);
                                        model.addElement(el);
                                        i++;
                                }
                        }
                }
                for(int j = 0; j < nodes.size(); j++) {
                        arrayMap.put(nodes.get(j).getPhoneNumber(), j);
                }

                for(Node node : nodes) {
                        Element el1 = elementMap.get(node.getPhoneNumber());
                        if(!node.getChildren().isEmpty()) {
                                el1.addEndPoint(createEndPoint(EndPointAnchor.BOTTOM));
                        }
                        for(Node childNode : node.getChildren()) {
                                Element el2 = elementMap.get(childNode.getPhoneNumber());
                                if(el2.getEndPoints() == null || el2.getEndPoints().isEmpty()) {
                                        el2.addEndPoint(createEndPoint(EndPointAnchor.TOP));
                                }
                                EndPoint endpoint = el1.getEndPoints().size() > 1 ? el1.getEndPoints().get(1)
: el1.getEndPoints().get(0);
                                model.connect(new Connection(endpoint, el2.getEndPoints().get(0),
createConnector()));
```

```
        nodes.get(arrayMap.get(childNode.getPhoneNumber())).getChildren().remove(node);
                                if(childNode.getPhoneNumber().equals(externalPhoneNumber)) {
                                        el2.setStyleClass(Constants.ELEMENT_SELECTED_STYLE_CLASS);
                                }
                        }
                }
        }

        /** Invoked when a node of a the graph has been clisked */
        public void onElementClicked(){
                if(selectedElement != null) {
                        selectedElement.setStyleClass(null);
                        for(Connection connection : model.getConnections()) {
                                connection.getConnector().setPaintStyle(Constants.CONNECTOR_PAINT_STYLE);
                                connection.getSource().setStyle(Constants.ENDPOINT_STYLE);
                                connection.getTarget().setStyle(Constants.ENDPOINT_STYLE);
                        }
                }

                String id =
FacesContext.getCurrentInstance().getExternalContext().getRequestParameterMap().get("elementId");
                selectedElement = elementMap.get(id.split("relationDiagram-")[1]);
                selectedElement.setStyleClass(Constants.ELEMENT_SELECTED_STYLE_CLASS);
                for(Connection connection : model.getConnections()) {
                        if(selectedElement.getEndPoints().contains(connection.getSource()) ||
                                        selectedElement.getEndPoints().contains(connection.getTarget())) {

        connection.getConnector().setPaintStyle(Constants.CONNECTOR_SELECTED_PAINT_STYLE);
                                connection.getSource().setStyle(Constants.ENDPOINT_SELECTED_STYLE);
                                connection.getTarget().setStyle(Constants.ENDPOINT_SELECTED_STYLE);
                        }
                }
        }


        private EndPoint createEndPoint(EndPointAnchor anchor) {
                DotEndPoint endPoint = new DotEndPoint(anchor);
                endPoint.setRadius(5);
                endPoint.setStyle(Constants.ENDPOINT_STYLE);
                return endPoint;
        }

        private Connector createConnector() {
                StraightConnector con = new StraightConnector();
                con.setPaintStyle(Constants.CONNECTOR_PAINT_STYLE);
                return con;
        }

        public DiagramModel getModel() {
                return model;
        }

        public void reset() {
                searchCriteria = new SearchCriteria();
                model = null;
                error = false;
        }

        public SearchCriteria getSearchCriteria() {
                return searchCriteria;
        }

        public boolean isError() {
                return error;
        }

}
```

Δείγμα Κώδικα 25: Η κλάση com.ntua.ote.logger.web.RelationFinderController

9.5 Η κλάση com.ntua.ote.logger.persistence.LoggerDAOImpl

Η κλάση LoggerDAOImpl διαχειρίζεται όλα τα αιτήματα της εφαρμογής προς την βάση δεδομένων. Περιέχει έναν EntityManager οποίος είναι συνδεδεμένος με το datasource της βάσης δεδομένων της εφαρμογης (το οποίο διαχειρίζεται ο Appllication server όπως έχει οριστεί στο Δείγμα Κώδικα 15). Οι δύο πίνακες της βάσης δεδομένων ελέγχονται μέσω των οντοτήτων (entities) Users και Log. Κάθε οντοτήτα είναι συνδεδεμένη με μια γραμμή του αντίστοιχου πίνακα, συνεπώς κάθε αλλαγή που γίνεται σε αυτήν ενημερώνει και την αντίστοιχη γραμμή χωρίς να χρειάζεται η δημιοργία ενός SQL query από τον προγραμματιστή.

```
package com.ntua.ote.logger.persistence;

import …;

@ApplicationScoped
public class LoggerDAOImpl implements LoggerDAO {

	@PersistenceContext(unitName = "logger")
	private EntityManager entityManager;

	private static final Logger LOGGER = Logger.getLogger(LoggerDAOImpl.class);

	/** Adds a new Log entry in the database */
	@Override
	public long addLog(Log log) {
		try {
			entityManager.persist(log);
			entityManager.flush();
			return log.getId();
		} catch (Exception e) {
			LOGGER.error("<add Log> :", e);
			return -1L;
		}
	}

	/** Updates a Log entry in the database with the location */
	@Override
	public int updateLocation(long id, double longitude, double latitude) {
		try {
			Log log = entityManager.find(Log.class, id);
			if (log != null) {
				log.setLongitude(longitude);
				log.setLatitude(latitude);
				log.setRadius(0);
				entityManager.flush();
				return 1;
			} else {
				LOGGER.error("<updateLocation> id not found");
				return -1;
			}
		} catch (Exception e) {
			LOGGER.error("<updateLocation> :", e);
			return -1;
		}
	}

	/**
	 * Updates a Log entry in the database with the location and radius obtained
	 * from MLS
	 */
	@Override
```

```
	public int updateLocation(long id, double longitude, double latitude, double radius) {
		try {
			Log log = entityManager.find(Log.class, id);
			if (log != null) {
				log.setLongitude(longitude);
				log.setLatitude(latitude);
				log.setRadius(radius);
				entityManager.flush();
				return 1;
			} else {
				LOGGER.error("<updateLocation> id not found");
				return -1;
			}
		} catch (Exception e) {
			LOGGER.error("<updateLocation> :", e);
			return -1;
		}
	}

	/** Updates a Log entry in the database with the duration */
	@Override
	public int updateDuration(long id, int duration) {
		try {
			Log log = entityManager.find(Log.class, id);
			if (log != null) {
				log.setDuration(duration);
				entityManager.flush();
				return 1;
			} else {
				LOGGER.error("<updateLocation> id not found");
				return -1;
			}
		} catch (Exception e) {
			LOGGER.error("<updateDuration> :", e);
			return -1;
		}
	}

	/** Fetches a Log entry from the database with id */
	@Override
	public Log get(Long id) {
		try {
			Log log = entityManager.find(Log.class, id);
			return log;
		} catch (Exception e) {
			LOGGER.error("<get> :", e);
			return null;
		}
	}

	/** Gets the @limit latest logs from database */
	public List<LogDetails> getLogDetails(int limit) {
		try {
			CriteriaBuilder cb = entityManager.getCriteriaBuilder();
			CriteriaQuery<Log> query = cb.createQuery(Log.class);
			Root<Log> sm = query.from(Log.class);
			query.orderBy(cb.desc(sm.get(Log_.dateTime)));
			List<Log> logs =
entityManager.createQuery(query).setMaxResults(limit).getResultList();
			return convertLog(logs);
		} catch (Exception e) {
			LOGGER.error("<updateDuration> :", e);
			return null;
		}
	}

	/** Searches the database based on the given criteria */
	public List<LogDetails> search(SearchCriteria searchCriteria) {
		try {
```

```
                        CriteriaBuilder cb = entityManager.getCriteriaBuilder();
                        CriteriaQuery<Log> query = cb.createQuery(Log.class);
                        Root<Log> sm = query.from(Log.class);
                        List<Predicate> predicates = new ArrayList<>();
                        if (searchCriteria.getId() > 0) {
                                predicates.add(cb.equal(sm.get(Log_.id), searchCriteria.getId()));
                        }
                        if (StringUtils.hasText(searchCriteria.getPhoneNumber())) {
                                predicates.add(cb.equal(sm.get(Log_.phoneNumber),
searchCriteria.getPhoneNumber()));
                        }
                        if (searchCriteria.getDateFrom() != null) {
                                predicates.add(cb.greaterThanOrEqualTo(sm.get(Log_.dateTime),
                                                new Timestamp(searchCriteria.getDateFrom().getTime())));
                        }
                        if (searchCriteria.getDateTo() != null) {
                                predicates.add(cb.lessThanOrEqualTo(sm.get(Log_.dateTime),
                                                new Timestamp(searchCriteria.getDateTo().getTime())));
                        }
                        if (StringUtils.hasText(searchCriteria.getExternalPhoneNumber())) {
                                predicates.add(cb.equal(sm.get(Log_.extPhoneNumber),
searchCriteria.getExternalPhoneNumber()));
                        }
                        if (StringUtils.hasText(searchCriteria.getDirection())) {
                                predicates.add(cb.equal(sm.get(Log_.direction),

        Direction.valueOf(searchCriteria.getDirection().toUpperCase())));
                        }
                        if (StringUtils.hasText(searchCriteria.getLogType())) {
                                predicates.add(
                                                cb.equal(sm.get(Log_.logType),
LogType.valueOf(searchCriteria.getLogType().toUpperCase())));
                        }
                        if (StringUtils.hasText(searchCriteria.getSmsContent())) {
                                String[] tokens = searchCriteria.getSmsContent().split(" ");
                                for (String s : tokens) {
                                        predicates.add(cb.like(sm.get(Log_.smsContent), "%" + s + "%"));
                                }
                        }
                        if (searchCriteria.getLongitude() != null && searchCriteria.getLatitude() != null
                                        && searchCriteria.getRadius() > 0) {
                                double[] diffLatLng =
Utils.calculateLatLngFromRadius(searchCriteria.getRadius(),
                                                searchCriteria.getLatitude().doubleValue(),
searchCriteria.getLongitude().doubleValue());

                                predicates.add(cb.greaterThanOrEqualTo(sm.get(Log_.latitude),
                                                searchCriteria.getLatitude().doubleValue() - diffLatLng[0]));
                                predicates.add(cb.lessThanOrEqualTo(sm.get(Log_.latitude),
                                                searchCriteria.getLatitude().doubleValue() + diffLatLng[0]));

                                predicates.add(cb.greaterThanOrEqualTo(sm.get(Log_.longitude),
                                                searchCriteria.getLongitude().doubleValue() -
diffLatLng[1]));
                                predicates.add(cb.lessThanOrEqualTo(sm.get(Log_.longitude),
                                                searchCriteria.getLongitude().doubleValue() +
diffLatLng[1]));

                                predicates.add(cb.notEqual(sm.get(Log_.latitude), 0));
                                predicates.add(cb.notEqual(sm.get(Log_.longitude), 0));
                        }
                        if (!predicates.isEmpty()) {
                                Predicate andClause = cb.and(predicates.toArray(new
Predicate[predicates.size()]));
                                query.where(andClause);
                        }
                        query.orderBy(cb.desc(sm.get(Log_.dateTime)));
                        List<Log> logs = entityManager.createQuery(query).getResultList();
                        return convertLog(logs);
```

```
                } catch (Exception e) {
                        LOGGER.error("<search> :", e);
                        return null;
                }
        }

        /** Searches the database based on the given criteria */
        public List<LogDetails> searchPath(SearchCriteria searchCriteria) {
                try {
                        CriteriaBuilder cb = entityManager.getCriteriaBuilder();
                        CriteriaQuery<Log> query = cb.createQuery(Log.class);
                        Root<Log> sm = query.from(Log.class);
                        List<Predicate> predicates = new ArrayList<>();
                        if (StringUtils.hasText(searchCriteria.getPhoneNumber())) {
                                predicates.add(cb.equal(sm.get(Log_.phoneNumber),
searchCriteria.getPhoneNumber()));
                        }
                        if (searchCriteria.getDateFrom() != null) {
                                predicates.add(cb.greaterThanOrEqualTo(sm.get(Log_.dateTime),
                                                new Timestamp(searchCriteria.getDateFrom().getTime())));
                        }
                        if (searchCriteria.getDateTo() != null) {
                                predicates.add(cb.lessThanOrEqualTo(sm.get(Log_.dateTime),
                                                new Timestamp(searchCriteria.getDateTo().getTime())));
                        }
                        predicates.add(cb.notEqual(sm.get(Log_.latitude), 0));
                        predicates.add(cb.notEqual(sm.get(Log_.longitude), 0));

                        if (!predicates.isEmpty()) {
                                Predicate andClause = cb.and(predicates.toArray(new
Predicate[predicates.size()]));
                                query.where(andClause);
                        }
                        query.orderBy(cb.asc(sm.get(Log_.dateTime)));
                        List<Log> logs = entityManager.createQuery(query).getResultList();
                        return convertLog(logs);
                } catch (Exception e) {
                        LOGGER.error("<search> :", e);
                        return null;
                }
        }

        /**
         * Retrieves logs from tha database as List of nodes in order to create a
         * graph
         */
        public SearchResults searchRelation(SearchCriteria searchCriteria) {
                SearchResults results = new SearchResults();
                List<Node> nodes = new ArrayList<>();
                results.setNodes(nodes);
                if (StringUtils.hasLength(searchCriteria.getPhoneNumber())) {
                        Node root = new Node(searchCriteria.getPhoneNumber(), 0);

                        nodes.add(root);
                        searchIntRelation(searchCriteria, root);
                        searchExtRelation(searchCriteria, root);
                        Set<String> uniqueSet = new HashSet<>();
                        uniqueSet.add(searchCriteria.getPhoneNumber());
                        List<Node> list = new ArrayList<>();
                        for (Node s : root.getChildren()) {
                                list.add(s);
                                s.setLevel(1);
                        }
                        while (!list.isEmpty()) {
                                Node node = list.get(0);
                                if (node.getPhoneNumber().equals(searchCriteria.getExternalPhoneNumber())) {
                                        results.setRelationFound(true);
                                }
                                String phoneNumber = node.getPhoneNumber();
```

```
                                int level = node.getLevel();
                                if (!uniqueSet.contains(phoneNumber)) {
                                        searchIntRelation(searchCriteria, node);
                                        searchExtRelation(searchCriteria, node);
                                        for (Node s : node.getChildren()) {
                                                list.add(s);
                                                s.setLevel(level + 1);
                                        }
                                        nodes.add(node);
                                        uniqueSet.add(phoneNumber);
                                }
                                list.remove(0);
                        }
                }
                return results;
        }

        private void searchIntRelation(SearchCriteria searchCriteria, Node parent) {
                try {
                        CriteriaBuilder cb = entityManager.getCriteriaBuilder();
                        CriteriaQuery<Tuple> query = cb.createTupleQuery();
                        Root<Log> sm = query.from(Log.class);
                        query.multiselect(sm.get(Log_.extPhoneNumber));
                        List<Predicate> predicates = new ArrayList<>();
                        if (StringUtils.hasText(searchCriteria.getPhoneNumber())) {
                                predicates.add(cb.equal(sm.get(Log_.phoneNumber), parent.getPhoneNumber()));
                        }
                        if (searchCriteria.getDateFrom() != null) {
                                predicates.add(cb.greaterThanOrEqualTo(sm.get(Log_.dateTime),
                                                new Timestamp(searchCriteria.getDateFrom().getTime())));
                        }
                        if (searchCriteria.getDateTo() != null) {
                                predicates.add(cb.lessThanOrEqualTo(sm.get(Log_.dateTime),
                                                new Timestamp(searchCriteria.getDateTo().getTime())));
                        }

                        if (!predicates.isEmpty()) {
                                Predicate andClause = cb.and(predicates.toArray(new
Predicate[predicates.size()]));
                                query.where(andClause);
                        }

                        List<Tuple> tupleResult = entityManager.createQuery(query).getResultList();
                        for (Tuple t : tupleResult) {
                                parent.getChildren().add(new Node((String) t.get(0)));
                        }
                } catch (Exception e) {
                        LOGGER.error("<search> :", e);
                }
        }

        private void searchExtRelation(SearchCriteria searchCriteria, Node parent) {
                try {
                        CriteriaBuilder cb = entityManager.getCriteriaBuilder();
                        CriteriaQuery<Tuple> query = cb.createTupleQuery();
                        Root<Log> sm = query.from(Log.class);
                        query.multiselect(sm.get(Log_.phoneNumber));
                        List<Predicate> predicates = new ArrayList<>();
                        if (StringUtils.hasText(searchCriteria.getPhoneNumber())) {
                                predicates.add(cb.equal(sm.get(Log_.extPhoneNumber),
parent.getPhoneNumber()));
                        }
                        if (searchCriteria.getDateFrom() != null) {
                                predicates.add(cb.greaterThanOrEqualTo(sm.get(Log_.dateTime),
                                                new Timestamp(searchCriteria.getDateFrom().getTime())));
                        }
                        if (searchCriteria.getDateTo() != null) {
                                predicates.add(cb.lessThanOrEqualTo(sm.get(Log_.dateTime),
                                                new Timestamp(searchCriteria.getDateTo().getTime())));
```

```
                        }

                        if (!predicates.isEmpty()) {
                                Predicate andClause = cb.and(predicates.toArray(new
Predicate[predicates.size()]));
                                query.where(andClause);
                        }

                        List<Tuple> tupleResult = entityManager.createQuery(query).getResultList();
                        for (Tuple t : tupleResult) {
                                parent.getChildren().add(new Node((String) t.get(0)));
                        }
                } catch (Exception e) {
                        LOGGER.error("<search> :", e);
                }
        }

        private List<LogDetails> convertLog(List<Log> logs) {
                List<LogDetails> logDetails = new ArrayList<LogDetails>();
                for (Log log : logs) {
                        LogDetails logDetail = new LogDetails();
                        logDetail.setId(log.getId());
                        logDetail.setDateTime(new Date(log.getDateTime().getTime()));
                        logDetail.setExternalPhoneNumber(log.getExtPhoneNumber());
                        logDetail.setPhoneNumber(log.getPhoneNumber());
                        logDetail.setDuration(Utils.timeToString(log.getDuration()));
                        logDetail.setSmsContent(log.getSmsContent());
                        logDetail.setLatitude(log.getLatitude());
                        logDetail.setLongitude(log.getLongitude());
                        logDetail.setBrandModel(log.getBrandModel());
                        logDetail.setVersion(log.getVersion());
                        logDetail.setCellId(log.getCellId());
                        logDetail.setDirection(log.getDirection());
                        logDetail.setImei(log.getImei());
                        logDetail.setImsi(log.getImsi());
                        logDetail.setLac(log.getLac());
                        logDetail.setLogType(log.getLogType());
                        logDetail.setLteCQI(log.getLteCQI());
                        logDetail.setLteRSRP(log.getLteRSRP());
                        logDetail.setLteRSRQ(log.getLteRSRQ());
                        logDetail.setLteRSSNR(log.getLteRSSNR());
                        logDetail.setRssi(log.getRssi());
                        logDetail.setRat(log.getRat());
                        logDetail.setMnc(log.getMnc());
                        logDetail.setMcc(log.getMcc());
                        logDetail.setRadius(log.getRadius());
                        logDetails.add(logDetail);
                }
                return logDetails;
        }

        /** Checks if a username-password pair exists in the database */
        public boolean login(String userName, String password) {
                String query = "select count(u) from Users u where u.userName = :userName"
                                + " and u.password = PASSWORD(:userPassword)";
                Query jpqlQuery = entityManager.createQuery(query).setParameter("userName", userName)
                                .setParameter("userPassword", password);
                long count = (long) jpqlQuery.getSingleResult();
                if (count > 0) {
                        return true;
                } else {
                        return false;
                }
        }

}
```

Δείγμα Κώδικα 26: Η κλάση com.ntua.ote.logger.persistence.LoggerDAOImpl

9.6 Η κλάση com.ntua.ote.logger.web.rest.RestApplicationEndpoint

Η κλάση RestApplicationEndpoint περιλαμβάνει όλα τα REST Web Services της εφαρμογής. Αναλαμβάνει την εξυπηρέτηση των requests και προετοιμάζει/στέλνει τα responses. Αποθηκεύει στην βάση δεδομένων τις καταγραφές που του στέλνονται από τον client. Οι καταγραφές έρχονται σε τρία στάδια. Πρώτα η πληροφορία που είναι διαθέσιμη κατά την καταγραφή της κλήσης/SMS και στην συνέχεια μπορεί να έρθει είτε η θέση της καταγραφής είτε η διάρκεια της κλήσης.
Επιπλέον ενημερώνει την εφαρμογή πελάτη για νέες εκδόσεις. Αυτό το επιτυγχάνει διατηρώντας το apk με την νεότερη έκδοση της εφαρμογής πελάτη καθώς και ένα αρχείο xml που περιλαμβάνει τον αριθμό της νεότερης έκδοσης της εφαρμογής πελάτη και μια σύντομη περιγραφεί των αλλαγών που αυτή υλοποιεί στον φάκελο $SERVER_PATH /../update/. Η μεταβλητή συστήματος $SERVER_PATH έχει οριστεί στον application server όπως φαίνεται στο Δείγμα Κώδικα 14.

```
package com.ntua.ote.logger.web.rest;

import …;

@Stateless
@Path("/log")
public class RestApplicationEndpoint {

	private static final Logger LOGGER = Logger.getLogger(RestApplicationEndpoint.class);

	@Inject
	private LoggerDAO loggerDAO;

	/** REST Service for receiving a Log with the initial information */
	@POST
	@Consumes(MediaType.APPLICATION_JSON + ";charset=utf-8")
	@Produces(MediaType.APPLICATION_JSON)
	@Path("/initial/")
	public long initialLogging(InitialRequest initialRequest) {
		if (loggerDAO.login(initialRequest.getUserName(), initialRequest.getPassword())) {
			LOGGER.info("<initialLogging invoked>" + initialRequest);
			Log log = new Log();
			log.setBrandModel(initialRequest.getBrandModel());
			log.setCellId(initialRequest.getCellId());
			if (initialRequest.getDateTime() != null) {
				log.setDateTime(new Timestamp(initialRequest.getDateTime().getTime()));
			}
			log.setDirection(initialRequest.getDirection());
			log.setExtPhoneNumber(initialRequest.getExternalPhoneNumber());
			log.setImei(initialRequest.getImei());
			log.setImsi(initialRequest.getImsi());
			log.setLac(initialRequest.getLac());
			log.setLteCQI(initialRequest.getLteCqi());
			log.setLteRSRP(initialRequest.getLteRsrp());
			log.setLteRSRQ(initialRequest.getLteRsrq());
			log.setLteRSSNR(initialRequest.getLteRssnr());
			log.setPhoneNumber(initialRequest.getPhoneNumber());
			log.setRat(initialRequest.getRat());
			log.setMnc(initialRequest.getMnc());
			log.setMcc(initialRequest.getMcc());
			log.setRssi(initialRequest.getRssi());
```

```
                        log.setSmsContent(initialRequest.getSmsContent());
                        log.setVersion(initialRequest.getVersion());
                        log.setLogType(initialRequest.getLogType());
                        return loggerDAO.addLog(log);
                }
                return -1;
        }

        /**
         * REST Service for receiving the location that the Log took place. If the
         * location is unavailable a call to Mozilla Location Services is invoked in
         * order to retrieve an estimation of the location
         */
        @POST
        @Consumes(MediaType.APPLICATION_JSON)
        @Produces(MediaType.APPLICATION_JSON)
        @Path("/location/")
        public int locationLogging(LocationRequest locationRequest) {
                if (loggerDAO.login(locationRequest.getUserName(), locationRequest.getPassword())) {
                        LOGGER.info("<locationLogging> invoked" + locationRequest);
                        if (locationRequest.isLocated()) {
                                return loggerDAO.updateLocation(locationRequest.getRowId(),
locationRequest.getLongitude(),
                                                locationRequest.getLatitude());
                        } else {
                                Log log = loggerDAO.get(locationRequest.getRowId());
                                GeolocateResponse geolocateResponse = GeolocateService.geolocate(log);
                                if (geolocateResponse != null && geolocateResponse.getError() == null) {
                                        loggerDAO.updateLocation(locationRequest.getRowId(),
geolocateResponse.getLocation().getLng(),
                                                        geolocateResponse.getLocation().getLat(),
geolocateResponse.getAccuracy());
                                }
                                return 1;
                        }
                }
                return -1;
        }

        /** REST Service for receiving the call duration of a Log */
        @POST
        @Consumes(MediaType.APPLICATION_JSON)
        @Produces(MediaType.APPLICATION_JSON)
        @Path("/duration/")
        public int durationLogging(DurationRequest durationRequest) {
                if (loggerDAO.login(durationRequest.getUserName(), durationRequest.getPassword())) {
                        LOGGER.info("<durationLogging> invoked" + durationRequest);
                        return loggerDAO.updateDuration(durationRequest.getRowId(),
durationRequest.getDuration());
                }
                return -1;
        }

        /**
         * REST Service for sending the number and the change log of the latest
         * existing version
         */
        @POST
        @Produces(MediaType.APPLICATION_JSON)
        @Path("/checkVersion/")
        public String checkVersion(AuthenticationRequest authRequest) {
                if (loggerDAO.login(authRequest.getUserName(), authRequest.getPassword())) {
                        Version version = parseVersionXml();
                        if (version != null) {
                                LOGGER.info("<check Version> invoked" + version.getVersionNumber());
                                Gson builder = new GsonBuilder().create();
                                return builder.toJson(version);
                        }
                }
```

```
                return null;
        }

        /** REST Service for sending the apk of the latest existing version */
        @POST
        @Path("/update/")
        @Produces(MediaType.APPLICATION_OCTET_STREAM)
        public Response update(AuthenticationRequest authRequest) {
                if (loggerDAO.login(authRequest.getUserName(), authRequest.getPassword())) {
                        Version version = parseVersionXml();
                        if (version != null) {
                                LOGGER.info("<update> invoked" + version.getVersionNumber());
                                String fileName = "logger_v" + version.getVersionNumber();
                                String path = System.getProperty("SERVER_PATH") + "../update/";
                                File file = new File(path + fileName);
                                ResponseBuilder response = Response.ok((Object) file);
                                response.header("Content-Disposition", "attachment; filename=" + fileName);
                                return response.build();
                        }
                }
                return null;
        }

        /**
         * Parses the version.xml which contains the version number and the change
         * log of the latest version
         */
        private Version parseVersionXml() {
                String path = System.getProperty("SERVER_PATH") + "../update/";
                File file = new File(path + "version.xml");
                DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
                DocumentBuilder dBuilder;
                try {
                        dBuilder = dbFactory.newDocumentBuilder();
                        Document doc = dBuilder.parse(file);
                        doc.getDocumentElement().normalize();
                        NodeList nList = doc.getElementsByTagName("latestVersion");
                        Element latestVersion = (Element) nList.item(0);
                        String versionNumber =
latestVersion.getElementsByTagName("versionNumber").item(0).getTextContent();
                        String changeLog =
latestVersion.getElementsByTagName("changeLog").item(0).getTextContent();
                        Version version = new Version();
                        version.setVersionNumber(versionNumber);
                        version.setChangeLog(changeLog);
                        return version;
                } catch (ParserConfigurationException | SAXException | IOException e) {
                        LOGGER.error("<check Version> exception", e);
                }
                return null;
        }
}
```

Δείγμα Κώδικα 27: Η κλάση com.ntua.ote.logger.web.rest.RestApplicationEndpoint

9.7 Η κλάση com.ntua.ote.logger.web.service.GeolocateService

Όταν σε μια καταγραφή που φτάνει στον server δεν είναι διαθέσιμη η θέση τότε η κλάση RestApplicationEndpoint καλεί την κλάση GeolocateService για να πάρει μια προσέγγιση της θέσης. Η κλάση GeolocateService αναλαμβάνει την κλήση στην βάση δεδομένων του Mozilla Location Service

χρησιμοποιώντας ως παράμετρο τα στοιχεία του δικτύου της καταγραφής (όπως περιγράφεται στις λειτουργικές απαιτήσεις 2.2.2). Το Mozilla Location Service επιστρέφει μια προσέγγιση της θέσης που περιγράφεται από ένα κέντρο και μια ακτίνα (σε μέτρα).

```
package com.ntua.ote.logger.web.service;

import …;

public class GeolocateService {

        private static String API = "test";

        private static final Logger LOGGER = Logger.getLogger(GeolocateService.class);

        /** Call MLS Database with REST Services to retrieve an estimation of the
         * location based on the network details of the Log */
        public static GeolocateResponse geolocate(Log log){
                try {

                    HttpPost request = new HttpPost("https://location.services.mozilla.com/v1/geolocate?key="
+ API);
                    String json = createRequest(log);
                    StringEntity params = new StringEntity(json);
                    LOGGER.info(json);
                    request.addHeader("content-type", "application/x-www-form-urlencoded");
                    request.setEntity(params);
                    HttpClient client = HttpClientBuilder.create().build();
                    HttpResponse response = client.execute(request);
                    BufferedReader rd = new BufferedReader(
                                new InputStreamReader(response.getEntity().getContent()));

                    StringBuffer result = new StringBuffer();
                    String line = "";
                    while ((line = rd.readLine()) != null) {
                        result.append(line);
                    }
                    rd.close();
                    Gson builder = new GsonBuilder().create();
                    GeolocateResponse geolocateResponse = builder.fromJson(result.toString(),
GeolocateResponse.class);
                    LOGGER.info(result);
                    return geolocateResponse;
                }catch (Exception ex) {
                        LOGGER.error(ex.getMessage());
                        return null;
                }
        }

        /** Creates the geolocate Request */
        private static String createRequest(Log log){
                GeolocateRequest geolocateRequest = new GeolocateRequest();
                CellTowers cellTowers = new CellTowers();
                geolocateRequest.setCellTowers(new  ArrayList<CellTowers>());
                geolocateRequest.getCellTowers().add(cellTowers);
                cellTowers.setCellId(log.getCellId());
                cellTowers.setLocationAreaCode(log.getLac());
                cellTowers.setMobileCountryCode(log.getMcc());
                cellTowers.setMobileNetworkCode(log.getMnc());
                if("LTE".equalsIgnoreCase(log.getRat()) || "gsm".equalsIgnoreCase(log.getRat()) ||
"wcdma".equalsIgnoreCase(log.getRat())) {
                        cellTowers.setRadioType(log.getRat().toLowerCase());
                } else {
                        cellTowers.setRadioType("wcdma");
                }
                cellTowers.setSignalStrength(log.getRssi());
                Gson builder = new GsonBuilder().create();
```

```
                return builder.toJson(geolocateRequest);
        }
}
```

Δείγμα Κώδικα 28: Η κλάση com.ntua.ote.logger.web.service.GeolocateService

9.8 Η κλάση com.ntua.ote.logger.web.xls.XLSBuilder

Η κλάση XLSBuilder αναλαμβάνει την κατασκευή του Excel αρχείου που θα περιέχει τα αποτελέσματα μιας αναζήτησης. Αποτελεί μια γενική υλοποίηση για την κατασκευή ενός Excel αρχείο από μια λίστα από ενός τύπου αντικείμενα υπό τους παρακάτω περιορισμούς:

- Τα αντικείμενα μπορεί να είναι οποιαδήποτε κλάσης αρκεί να υλοποιούν το Interface Serializable
- Η κλάση των αντικειμένων θα πρέπει να έχει τα πεδία που θα τυπωθούν στο πρώτο επίπεδο
- Ο τύπος των πεδίων μπορεί να είναι: String, Date, int, double, Enum

Η κλάση XLSBuilder χρησιμοποιεί reflection για να εντοπίσει τα ονόματα των μεταβλητών των αντικειμένων που είναι μέσα στην λίστα. Για κάθε μεταβλητή δημιουργεί μια στήλη στο αρχείο Excel και την γεμίζει με τις τιμές τους. Για την εμφάνιση φιλικών ως προς τον χρήστη ονομάτων στις στήλες απαιτείται και ένας πίνακας από XLSColumn αντικείμενα τα οποία παρέχουν μια αντιστοίχιση των ονομάτων των μεταβλητών με φιλικές ως προς τον χρήστη ετικέτες.

```
package com.ntua.ote.logger.web.xls;

import …;

public class XLSBuilder {

        private static Workbook wb;
        private static Sheet sheet;
        private static CellStyle headercs;
        private static CellStyle labelcs;
        private static CellStyle valuecs;
        private static short columnNum;
        private static Properties resourceBundles = new Properties();

        /** Configure excel styles.*/
        private static void configureStyles() {
                headercs = wb.createCellStyle();
                headercs.setAlignment(HorizontalAlignment.CENTER);
                headercs.setDataFormat(HSSFDataFormat.getBuiltinFormat("text"));
                headercs.setFillForegroundColor(HSSFColor.GREY_80_PERCENT.index);
                headercs.setFillPattern(FillPatternType.SOLID_FOREGROUND);

                labelcs = wb.createCellStyle();
                labelcs.setDataFormat(HSSFDataFormat.getBuiltinFormat("text"));
                labelcs.setBorderBottom(BorderStyle.THIN);

                valuecs = wb.createCellStyle();
                valuecs.setDataFormat(HSSFDataFormat.getBuiltinFormat("text"));
                valuecs.setAlignment(HorizontalAlignment.RIGHT);

                // create 2 fonts objects
                Font headerFont = wb.createFont();
                headerFont.setBold(true);
                headerFont.setFontHeightInPoints((short) 12);
                headerFont.setColor(HSSFColor.WHITE.index);
```

```
                Font labelFont = wb.createFont();
                labelFont.setBold(true);

                Font valueFont = wb.createFont();

                // set cell stlye
                headercs.setFont(headerFont);
                labelcs.setFont(labelFont);
                valuecs.setFont(valueFont);

        }

        /** Draw cell. */
        private static void drawCell(Row r, String value, CellStyle cellStyle) {
                Cell c = r.createCell(columnNum);
                c.setCellStyle(cellStyle);
                c.setCellValue(value);
                columnNum++;
        }

        /** Draw results. */
        private static void drawResults(List<? extends Serializable> results, XLSColumn[] mappings, String
header) {
                // declare a row object reference
                Row r = null;
                short rownum = 0;

                // header
                r = sheet.createRow(rownum);
                drawCell(r, resourceBundles.getProperty(header), headercs);

                // column names
                columnNum = 0;
                rownum++;
                r = sheet.createRow(rownum);
                for(XLSColumn mapping : mappings){
                        drawCell(r, resourceBundles.getProperty(mapping.getLabel()), labelcs);
                }

                // data
                for (Serializable result : results) {
                        columnNum = 0;
                        rownum++;
                        r = sheet.createRow(rownum);
                        BeanWrapper beanWrapper = PropertyAccessorFactory.forBeanPropertyAccess(result);

                for (XLSColumn mapping : mappings) {
                //get the property descriptor for each field
                PropertyDescriptor pd = beanWrapper.getPropertyDescriptor(mapping.getField());
                    if (pd.getWriteMethod() != null) {
                        Class<?> propClazz = pd.getPropertyType();
                        Object propertyValue = beanWrapper.getPropertyValue(pd.getName());
                        if (String.class.isAssignableFrom(propClazz)) {
                        if (StringUtils.hasText((String) propertyValue)) {
                                drawCell(r, propertyValue.toString(), valuecs);
                        } else {
                                drawCell(r, "", valuecs);
                        }
                        } else if (Date.class.isAssignableFrom(propClazz)) {
                        if ((propertyValue != null)) {
                                Locale locale = FacesContext.getCurrentInstance().getViewRoot().getLocale();
                                String date = new SimpleDateFormat("EEE, d MMM YYY HH:mm:ss",
locale).format(((Date)propertyValue));
                                        drawCell(r, date, valuecs);
                        } else {
                                drawCell(r, "", valuecs);
                        }
                        } else if (int.class.isAssignableFrom(propClazz)) {
                        if ((propertyValue != null)) {
```

```
                                drawCell(r, propertyValue.toString(), valuecs);
                        } else {
                                drawCell(r, "", valuecs);
                        }
                        } else if (double.class.isAssignableFrom(propClazz)) {
                        if ((propertyValue != null)) {
                                drawCell(r, propertyValue.toString(), valuecs);
                        } else {
                                drawCell(r, "", valuecs);
                        }
                        } else if (Enum.class.isAssignableFrom(propClazz)) {
                        if ((propertyValue != null)) {
                                drawCell(r, ((Enum)propertyValue).name(), valuecs);
                        } else {
                                drawCell(r, "", valuecs);
                        }
                        }
                    }
                }
                }

        }

        /** Creates an Excel Document which includes the results of a search. */
        public static byte[] exportXLS(List<? extends Serializable> results, XLSColumn[] mappings, String
header, String language) throws IOException {

                InputStream input = Thread.currentThread().getContextClassLoader()
                                .getResourceAsStream("CommonMessages_"+language.toLowerCase()+".properties");
                resourceBundles.load(input);

                // create a new workbook
                wb = new HSSFWorkbook();
                // create a new sheet
                sheet = wb.createSheet();
                // create 3 cell styles
                wb.setSheetName(0, resourceBundles.getProperty(header));

                columnNum = 0;
                configureStyles();
                //create a merged region for the header
                sheet.addMergedRegion(new CellRangeAddress(0, 0, 0, mappings.length - 1));

                drawResults(results, mappings, header);

                //autosize each column
                for (int i = 0; i < mappings.length; i++) {
                        sheet.autoSizeColumn(i);
                }

                ByteArrayOutputStream bos = new ByteArrayOutputStream();
                try {
                    wb.write(bos);
                } finally {
                    bos.close();
                }
                byte[] bytes = bos.toByteArray();
                return  bytes;
        }

}
```

Δείγμα Κώδικα 29: Η κλάση com.ntua.ote.logger.web.xls.XLSBuilder

9.9 Η κλάση com.ntua.ote.logger.web.components.ManyInputSelect

Η κλάση ManyInputSelect ελέγχει το web component που αναλαμβανει την εισαγωγή πολλων μονοπατιών προς αναζήτηση (Εικόνα 31). Διαχειρίζεται την εισαγωγή και διαγραφή αριθμών τηλεφώνου χρησιμοποιώντας μια λίστα με τα τηλεφωνα που έχει εισάγει ο χρήστης και μια στοίβα με τα διαθέσιμα χρώματα.
Το web component είναι το web\src\main\webapp\components\manyInputSelect.xhtml.

```
package com.ntua.ote.logger.web.components;

import …;

public class ManyInputSelect {

	private List<ManyInputSelectItem> inputs;
	private String input;
	private Stack<String> colorStack;

	/** Initialize the stack of colors with five preset colors */
	public ManyInputSelect(){
		inputs = new ArrayList<>();
		colorStack = new Stack<>();
		colorStack.push("#a020f0");
		colorStack.push("#FF9900");
		colorStack.push("#0000FF");
		colorStack.push("#228b22");
		colorStack.push("#FF0000");
	}

	/** Checks if the input has already been added */
	private boolean exists(String input){
		for(ManyInputSelectItem item : inputs) {
			if(input.equals(item.input)) {
				return true;
			}
		}
		return false;
	}

	/** Add an entry from input. Remove the top color from the stack.
	 * Create a ManyInputSelectItem with the removed color and input and add it to the list */
	public void add(){
		if(StringUtils.hasLength(input) && !exists(input) && !colorStack.isEmpty()) {
			ManyInputSelectItem item = new ManyInputSelectItem();
			item.input = input;
			item.color = colorStack.pop();
			inputs.add(item);
			input="";
		}
	}

	/** Remove an entry from input. Adds the removed entry's color to the stack.
	 * Removes the ManyInputSelectItem from the list */
	public void remove(ManyInputSelectItem item){
		colorStack.push(item.color);
		inputs.remove(item);
	}

	/** The class ManyInputSelectItem */
	public class ManyInputSelectItem {
```

```
                private String input;
                private String color;
                public String getInput() {
                        return input;
                }
                public void setInput(String input) {
                        this.input = input;
                }
                public String getColor() {
                        return color;
                }
                public void setColor(String color) {
                        this.color = color;
                }
        }

        public List<ManyInputSelectItem> getInputs() {
                return inputs;
        }

        public String getInput() {
                return input;
        }

        public void setInput(String input) {
                this.input = input;
        }

}
```

Δείγμα Κώδικα 30: Η κλάση com.ntua.ote.logger.web.components.ManyInputSelect

10 Βάσεις δεδομένων των εφαρμογών

10.1 Βάση δεδομένων της εφαρμογής πελάτη Logger Client

Η βάση δεδομένων της εφαρμογής πελάτη (Android App) υλοποιήθηκε σε SQLite. Η βάση δεδομένων αποτελείται από δύο πίνακες. Ο πίνακας Log περιέχει τις πληροφορίες που κατέγραψε η εφαρμογή Android. Ο πίνακας pendingRequests διατηρεί το σύνολο της καταγεγραμμένης πληροφορίας που δεν έχει σταλεί ακόμα στον server κατά τον τερματισμό της εφαρμογής. Στην Εικόνα 6 φαίνεται το σχήμα των δύο αυτών πινάκων.

Η βάση της εφαρμογής καθώς και οι πίνακες δημιουργούνται κατά την εγκατάσταση της εφαρμογής Logger Client στην συσκευή. Συγκεκριμένα απαιτείται μια κλάση που να κληρονομεί την αφηρημένη κλάση (abstract class) SQLiteOpenHelper. Η συγκεκριμένη κλάση θα πρέπει να περιέχει το όνομα και τον αριθμό της τρέχουσας έκδοσης της βάσης δεδομένων και να υλοποιεί τις παρακάτω μεθόδους

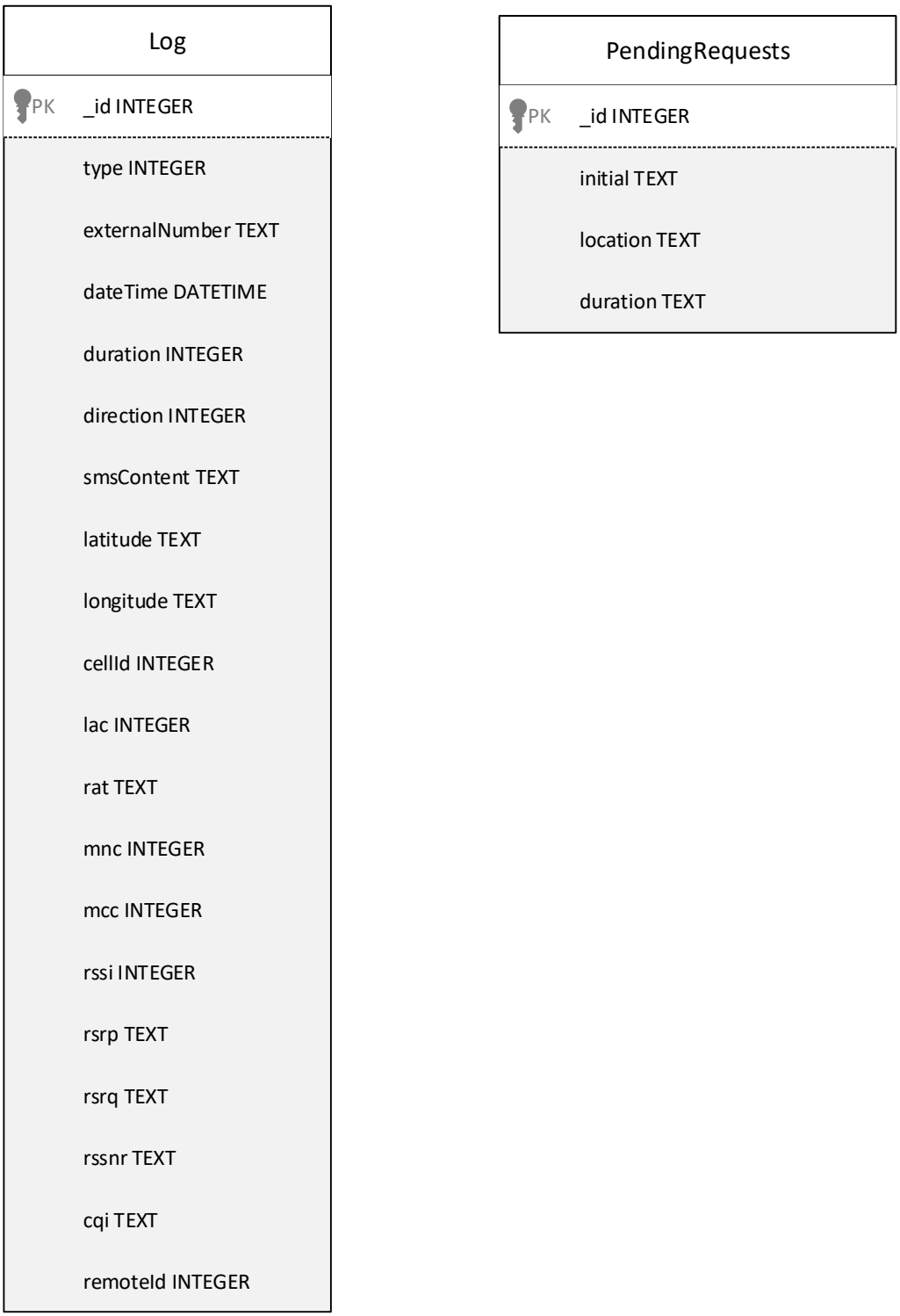
- void onCreate(SQLiteDatabase db). Περιλαμβάνει τον κώδικα που θα εκτελεστεί κατά την εγκατάσταση της εφαρμογής.
- void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion). Περιλαμβάνει τον κώδικα που θα εκτελεστεί κατά την αναβάθμιση της εφαρμογής με νεότερης έκδοσης βάση δεδομένων.
- void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion). Περιλαμβάνει τον κώδικα που θα εκτελεστεί κατά την υποβάθμιση της εφαρμογής με παλαιότερης έκδοσης βάση δεδομένων.

Στο Δείγμα Κώδικα 31 φαίνεται η υλοποίηση μας της αφηρημένη κλάση (abstract class) SQLiteOpenHelper.

```
package com.ntua.ote.logger.db;

import …

public class CallLogDbHelper extends SQLiteOpenHelper {

    public static final int DATABASE_VERSION = 14;
    public static final String DATABASE_NAME = "logger.db";

    private CallLogDbHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CallLogDbSchema.SQL_CREATE_ENTRIES);
        db.execSQL(CallLogDbSchema.SQL_CREATE_PENDING_ENTRIES);
    }

    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL(CallLogDbSchema.SQL_DELETE_ENTRIES);
        db.execSQL(CallLogDbSchema.SQL_DELETE_PENDING_ENTRIES);
        onCreate(db);
    }
    public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {
```

```
        onUpgrade(db, oldVersion, newVersion);
    }
}
```

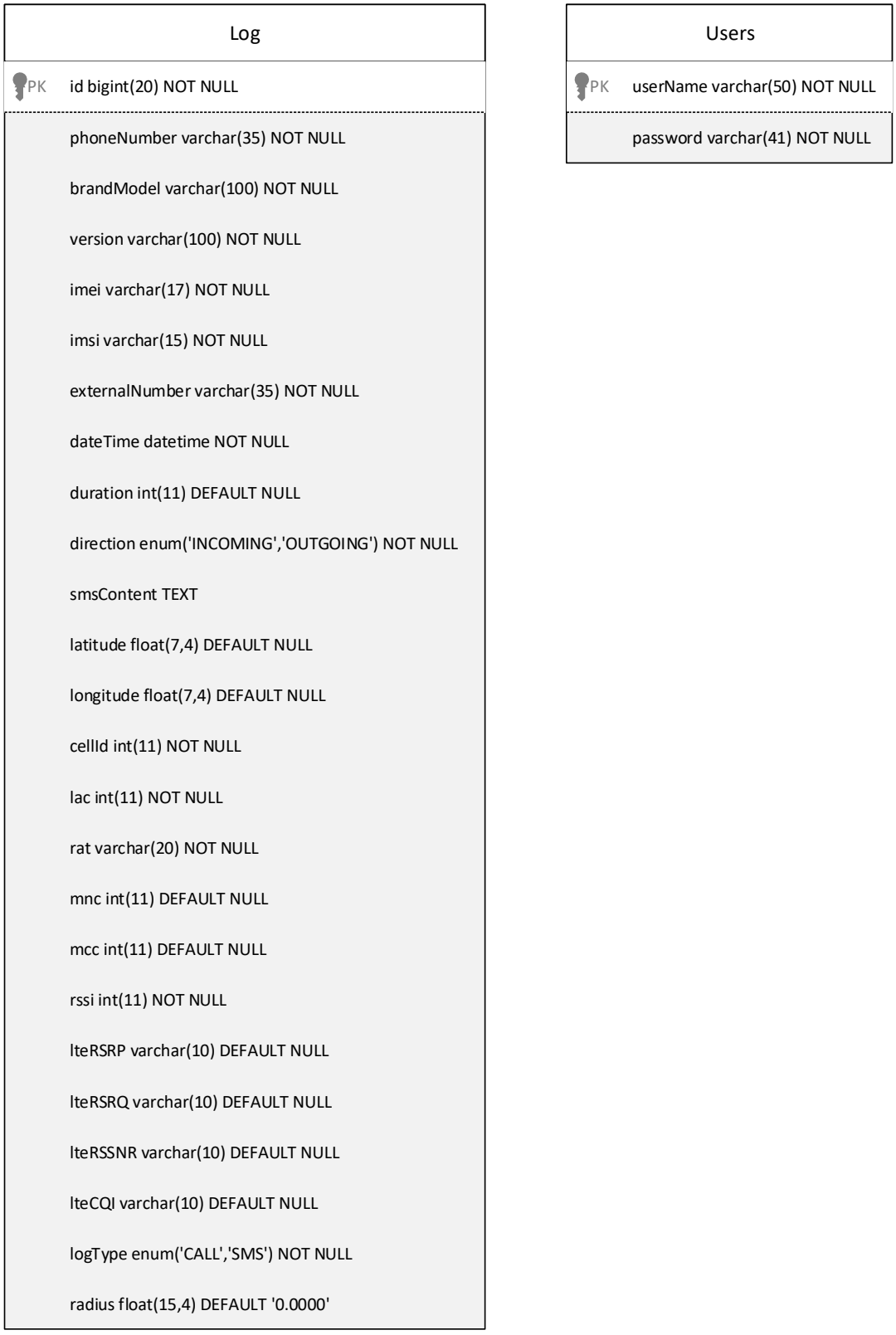
Δείγμα Κώδικα 31: Υλοποίηση της SQLiteOpenHelper



Εικόνα 6: Σχήμα της βάσης δεδομένων της εφαρμογής Logger Client

10.2 Βάση δεδομένων της εφαρμογής Logger Server

Η βάση δεδομένων της εφαρμογής Logger Server υλοποιήθηκε σε MySQL. Η βάση δεδομένων αποτελείται από δύο πίνακες. Ο πίνακας Log περιέχει τις πληροφορίες που καταγράφθηκαν και στάλθηκαν από τις συσκευές στις οποίες είναι εγκατεστημένη η εφαρμογή πελάτη (Android App). Ο πίνακας Users περιέχει τα ονόματα και τους κωδικούς των χρηστών που έχουν πρόσβαση στην εφαρμογή. Στην Εικόνα 7 φαίνεται το σχήμα των δύο αυτών πινάκων.



Εικόνα 7: Σχήμα της βάσης δεδομένων της εφαρμογής Logger Server

10.2.1 Εγκατάσταση βάσης δεδομένων της εφαρμογής Logger Server

Από την ιστοσελίδα (12) επιλέγουμε το λειτουργικό σύστημα που χρησιμοποιούμε καθώς και την έκδοσή του, το κατεβάζουμε στο μηχάνημα μας και το εγκαθιστούμε. Κατά την εγκατάσταση επιλέγουμε αριθμό πόρτας (στον οποίο θα τρέχει η βάση δεδομένων) όνομα χρήστη και κωδικό. Αυτά τα στοιχεία πρέπει να τα προσθέσουμε και στο αρχείο standalone-full.xml του JBoss Application Server (Δείγμα Κώδικα 15) έτσι ώστε να μπορέσει να συνδεθεί ο Server με την βάση δεδομένων.
Έχοντας εγκαταστήσει τον MySQL Server:

- **Για Windows**: ανοίγουμε ένα cmd και εκτελούμε την εντολή
  ${path to MySQL Server}\bin\mysql.exe –u ${username} –p
- **Για GNU/Linux:** ανοίγουμε ένα terminal και εκτελούμε την εντολή
  mysql –u ${username} –p

βάζουμε τον κωδικό και εκτελούμε τον κώδικα που φαίνεται στο Δείγμα Κώδικα 32 για την δημιουργία της βάσης δεδομένων και των πινάκων της.

```
create database logger_db;

create table Log (
	id BIGINT NOT NULL,
	phoneNumber VARCHAR(35) NOT NULL,
	brandModel VARCHAR(100) NOT NULL,
	version VARCHAR(100) NOT NULL,
	imei VARCHAR(17) NOT NULL,
	imsi VARCHAR(15) NOT NULL,
	extPhoneNumber VARCHAR(35) NOT NULL,
	dateTime DATETIME NOT NULL,
	duration INT,
	smsContent TEXT,
	direction ENUM('INCOMING', 'OUTGOING') NOT NULL,
	latitude FLOAT(7,4),
	longitude FLOAT(7,4),
	radius FLOAT(15,4),
	cellId INT NOT NULL,
	lac INT NOT NULL,
	rat VARCHAR(20) NOT NULL,
	mnc INT,
	mcc INT,
	rssi INT NOT NULL,
	lteRSRP VARCHAR(10),
	lteRSRQ VARCHAR(10),
	lteRSSNR VARCHAR(10),
	lteCQI VARCHAR(10),
	logType ENUM('CALL', 'SMS') NOT NULL,
	PRIMARY KEY (id)
);

create table Users (
	userName VARCHAR(50) NOT NULL,
	password VARCHAR(41) NOT NULL,
	PRIMARY KEY (userName)
);

ALTER DATABASE logger_db CHARACTER SET = utf8mb4 COLLATE = utf8mb4_unicode_ci;
```

```
ALTER TABLE Log CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
ALTER TABLE Log CHANGE smsContent smsContent TEXT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

Δείγμα Κώδικα 32: MySQL Script για την δημιουργία της βάσης δεδομένων και των πινάκων για την εφαρμογή Logger Server

11 Ασφάλεια

11.1 Ασφάλεια των δεδομένων κατά την αποστολή τους

Η καταγεγραμμένη πληροφορία μεταφέρεται από την εφαρμογή πελάτη στον server χρησιμοποιώντας Rest services. Ωστόσο, για να αποφευχθεί η υποκλοπή της πληροφορίας από κάποιον τρίτο που βρίσκεται στο ίδιο δίκτυο και ακούει τα πακέτα που στέλνονται, η σύνδεση είναι κρυπτογραφημένη βάσει του TLS πρωτοκόλλου που παρέχει το HTTPS. Ο server και ο client συμφωνούν σε ένα κλειδί που θα χρησιμοποιήσουν για την ασύμμετρη κρυπτογράφηση των δεδομένων που θα σταλούν εμποδίζοντας την παραποίηση ή την κλοπή δεδομένων. Για την επίτευξη HTTPS σύνδεσης ο server χρειάζεται ένα ζευγάρι κλειδιών (δημόσιο και ιδιωτικό) που θα τα χρησιμοποιήσει για την συμμετρική κρυπτογράφηση της συνομιλίας του με τον client (στέλνοντας του το δημόσιο κλειδί) μέχρι να συμφωνηθεί το ασύμμετρο κλειδί. Ιδανικά θα έπρεπε αυτό το ζευγάρι κλειδιών να είναι υπογεγραμμένο από μια έμπιστη αρχή πιστοποίησης έτσι ώστε να εγγυάται η εγκυρότητα του.
Στην δικιά μας υλοποίηση δημιουργήσαμε ένα self-signed certificate και κάναμε την εφαρμογή πελάτη να το εμπιστεύεται έτσι ώστε να μπορεί να επιτευχθεί HTTPS σύνδεση. Στο Δείγμα Κώδικα 16 φαίνεται η παραμετροποίηση που απαιτείται στο standalone-full.xml του JBoss Application Server και στο Δείγμα Κώδικα 33 φαίνεται ο κώδικας που πρέπει να προστεθεί στο αρχείο web\src\main\webapp\WEB-INF\web.xml για την ενεργοποίηση του HTTPS στην εφαρμογή μας.

```
{…}
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>SECURE</web-resource-name>
            <url-pattern>/*</url-pattern>
        </web-resource-collection>
        <user-data-constraint>
            <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
    </security-constraint>
{…}
```

Δείγμα Κώδικα 33: Κομμάτι κώδικα του αρχείου web/src/main/webapp/WEB-INF/web.xml που ενεργοποιεί το HTTPS στην εφαρμογή Logger Server

```
keytool -genkey -keyalg RSA -alias ${ssl cert alias} -keystore keystore.jks -storepass ${certificate password} -validity 360 -keysize 2048
```

Δείγμα Κώδικα 34: Εντολή για την δημιουργία ενός Self Signed Certificate χρησιμοποιώντας το εργαλείο keytool της JAVA

Επιπλέον, υπάρχει η επιλογή μέσα από την εφαρμογή πελάτη να στέλνεται η καταγεγραμμένη πληροφορία μόνο όταν είναι ενεργή η σύνδεση στο διαδίκτυο μέσω των δεδομένων του δικτύου κινητής τηλεφωνίας. Με αυτό τον τρόπο αξιοποιείται η ασφάλεια που παρέχει η υποδομή των δικτύων κινητής τηλεφωνίας για την παρεμπόδιση της υποκλοπής των πακέτων από τρίτους.

11.2 Ασφάλεια των δεδομένων στον server

Τα δεδομένα που λαμβάνονται από τον server αποθηκεύονται στην βάση δεδομένων. Μέσα από το γραφικό περιβάλλον ο χρήστης έχει την δυνατότητα να δει τα περιεχόμενα της βάσης δεδομένων. Ωστόσο ο χρήστης πρέπει να εισάγει ένα έγκυρο ζεύγος ονόματος χρήστη – κωδικού για να εισέλθει στην εφαρμογή. Έτσι αποτρέπεται η μη εξουσιοδοτημένη πρόσβαση στην εφαρμογή και στα περιεχόμενα της βάσης δεδομένων. Τα ζεύγη ονόματος χρήστη – κωδικού διατηρούνται στην βάση δεδομένων και συγκεκριμένα ο κωδικός είναι κατακερματισμένος χρησιμοποιώντας την συνάρτηση PASSWORD() της MySQL. Η συγκεκριμένη συνάρτηση μετατρέπει οποιαδήποτε ακολουθία χαρακτήρων σε μια ακολουθία με 41 χαρακτήρες προστατεύοντας έτσι τον κωδικό του χρήστη. Τέλος, τα πεδία της εισαγωγής ονόματος χρήστη – κωδικού επεξεργάζονται κατάλληλα έτσι ώστε να εμποδίζονται επιθέσεις τύπου SQL Injection.
Στο Δείγμα Κώδικα 35 φαίνεται η υλοποίηση των παραπάνω. Το όνομα χρήστη και ο κωδικός επεξεργάζονται κατάλληλα πριν μπουν στο Query από την μέθοδο javax.persistence.Query.setParameter(String name, Object value). Ο κωδικός που έβαλε ο χρήστης συγκρίνεται κατακερματισμένος με το αντίστοιχο πεδίο της βάσης.

```
public boolean login(String userName, String password){
	String query = "select count(u) from Users u where u.userName = :userName"
		+ " and u.password = PASSWORD(:userPassword)";
	Query jpqlQuery = entityManager.createQuery(query)
		.setParameter("userName", userName)
		.setParameter("userPassword", password);
	long count = (long) jpqlQuery.getSingleResult();
	if(count > 0) {
		return true;
	} else {
		return false;
	}
}
```

Δείγμα Κώδικα 35: Κομμάτι κώδικα για την επαλήθευση ονόματος χρήστη – κωδικού.
Κλάση persistence/src/main/java/com/ntua/ote/logger/persistence/LoggerDAOImpl.java

11.3 Προστασία υπηρεσιών του server

Ο Server παρέχει υπηρεσίες που μπορούν να χρησιμοποιηθούν από την εφαρμογή πελάτη έτσι ώστε να στείλει πληροφορία που κατέγραψε ή να ενημερωθεί για κάποια διαθέσιμη ενημέρωση. Ωστόσο αν αυτές οι υπηρεσίες επέτρεπαν και σε μη-χρήστες της εφαρμογής να τις χρησιμοποιήσουν αυτό θα είχε σαν αποτέλεσμα την αποστολή μη-έγκυρης πληροφορίας. Επιπλέον, επειδή αυτές οι υπηρεσίες χρησιμοποιούν αρκετούς πόρους του μηχανήματος και του δικτύου για να εκτελεστούν (κλήσεις στην βάση δεδομένων, κλήση εξωτερικών υπηρεσιών, κατέβασμα ενημερώσεων της εφαρμογής πελάτη) θα μπορούσαν να εκμεταλλευτούν από κακόβουλους χρήστες για επιθέσεις τύπου Denial of Service (DOS attacks). Συγκεκριμένα, ένας κακόβουλός χρήστης θα μπορούσε να στέλνει επαναλαμβανόμενα αιτήματα στις

υπηρεσίες της εφαρμογής οδηγώντας την εφαρμογή, στην προσπάθεια της να τα εξυπηρετήσει, να αγνοήσει άλλα αιτήματα (πιθανώς έγκυρα). Στην χειρότερη περίπτωση θα μπορούσε να οδηγηθεί σε διακοπή της λειτουργίας της λόγω του φόρτου εργασίας. Για την αποτροπή των παραπάνω κατά την χρήση των υπηρεσιών από την εφαρμογή πελάτη στέλνεται ένα ζευγάρι όνομα χρήστη-κωδικού με το οποίο επαληθεύεται η εγκυρότητα της. Με αυτών τον τρόπο τα αιτήματα ενός χρήστη που δεν έχει ένα έγκυρο ζευγάρι όνομα χρήστη-κωδικού δεν εξυπηρετούνται και συνεπώς δεν επιβαρύνουν τον server. Ο τρόπος με τον οποίο γίνεται επαλήθευση του ονόματος χρήστη-κωδικού από τον Logger Server φαίνεται στο Δείγμα Κώδικα 35.

12 Σενάρια Χρήσης

12.1 Σενάρια Χρήσης της εφαρμογής πελάτη Logger Client

Παρακάτω παρατίθενται σενάρια χρήσης που περιγράφουν και απεικονίζουν την λειτουργικότητα της εφαρμογής πελάτη

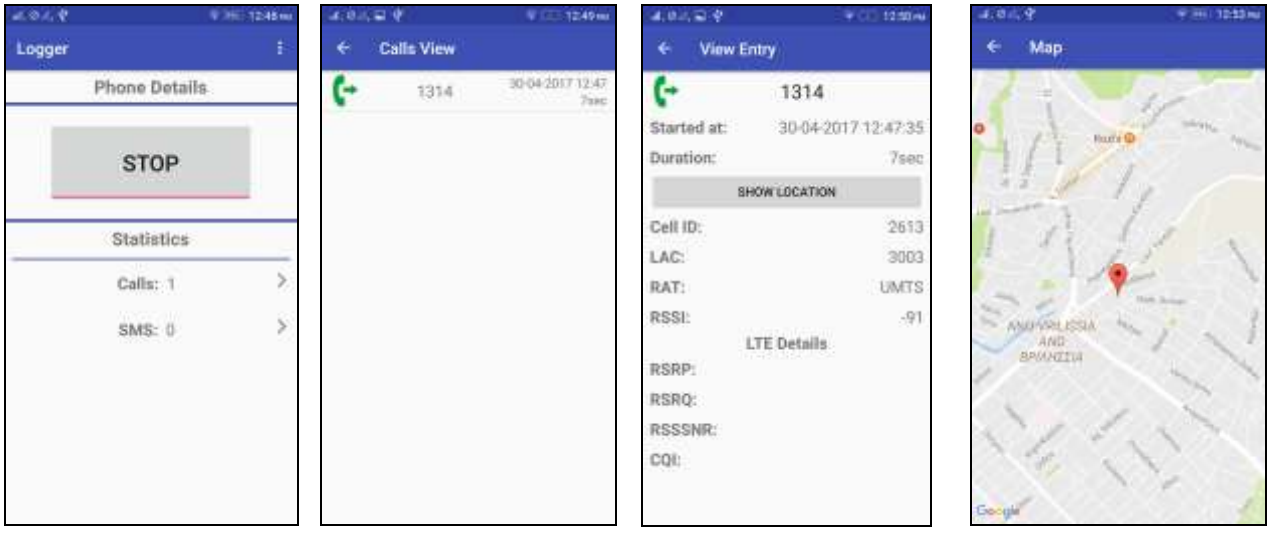
12.1.1 Αρχική οθόνη της εφαρμογής

Στην Εικόνα 8 φαίνεται η αρχική οθόνη της εφαρμογής. Στο επάνω μέρος φαίνονται πληροφορίες της συσκευής (η μάρκα / μοντέλο, η έκδοση του λογισμικού, ο αριθμός τηλεφώνου - MSISDN, IMEI, IMSI). Στην μέση βρίσκεται το κουμπί **LAUNCH** που εκκινεί την καταγραφή. Τέλος, στο κάτω μέρος της οθόνης φαίνεται πόσες κλήσεις και μηνύματα έχουν καταγραφεί.

Εικόνα 8: Αρχική οθόνη της εφαρμογής

12.1.2 Απεικόνιση των στοιχείων μιας καταγεγραμμένης κλήσης

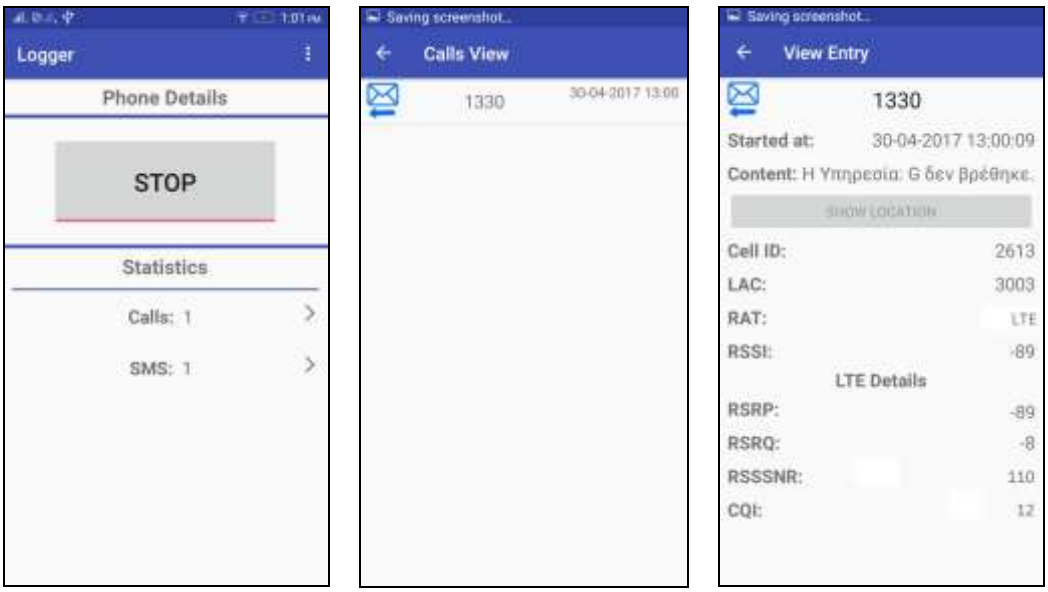
Έχοντας ξεκινήσει την καταγραφή κάνουμε μια κλήση. Στην Εικόνα 9 πατώντας στο **Calls** οδηγούμαστε σε μια οθόνη που δείχνει την λίστα με τις κλήσεις που έχουν καταγραφεί. Επιλέγοντας μια από τις κλήσεις βλέπουμε την πληροφορία που έχει καταγραφεί. Στην συγκεκριμένη περίπτωση είχαμε ενεργοποιημένη την υπηρεσία εντοπισμού θέσης και έτσι μπορούμε να δούμε την τοποθεσία από τη οποία κάναμε την κλήση.



Εικόνα 9: Απεικόνιση των στοιχείων μιας καταγεγραμμένης κλήσης

12.1.3 Απεικόνιση των στοιχείων ενός καταγεγραμμένου SMS

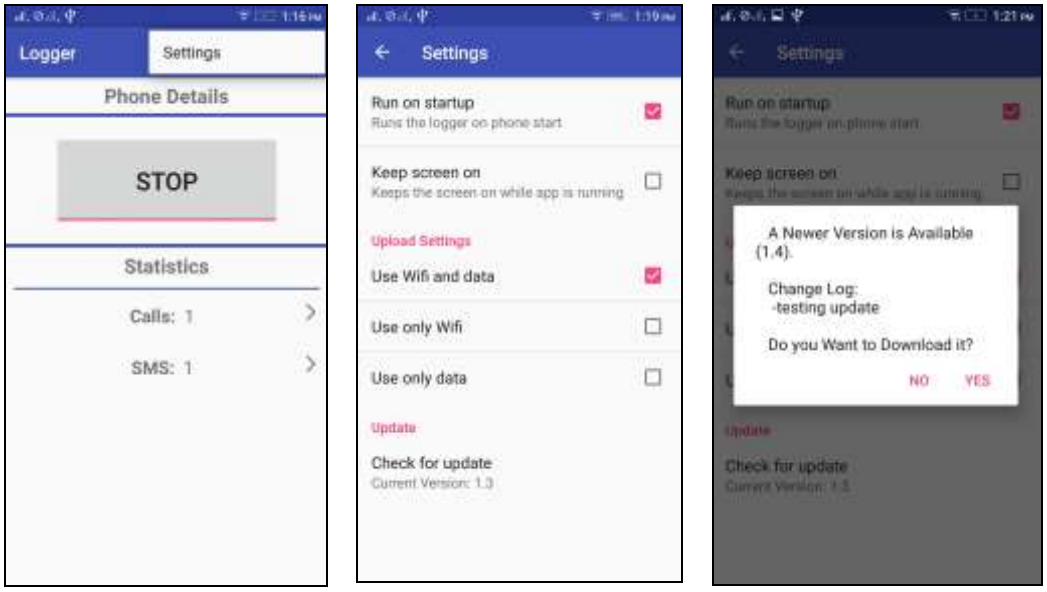
Έχοντας ξεκινήσει την καταγραφή λαμβάνουμε ένα μήνυμα. Στην Εικόνα 10 πατώντας στο **SMS** οδηγούμαστε σε μια οθόνη που δείχνει την λίστα με τα μηνύματα που έχουν καταγραφεί. Επιλέγοντας ένα από τα μηνύματα βλέπουμε την πληροφορία που έχει καταγραφεί. Στην συγκεκριμένη περίπτωση δεν είχαμε ενεργοποιημένη την υπηρεσία εντοπισμού θέσης και έτσι δεν μπορούμε να δούμε την τοποθεσία από τη οποία λάβαμε το μήνυμα



Εικόνα 10: Απεικόνιση των στοιχείων ενός καταγεγραμμένου SMS

12.1.4 Ρύθμιση και ενημέρωση της εφαρμογής

Από το μενού της αρχικής οθόνης μπορούμε να πάμε στις ρυθμίσεις της εφαρμογής. Στην Εικόνα 11 φαίνονται οι διαθέσιμες ρυθμίσεις της εφαρμογής καθώς και η επιλογή για να ενημερώσουμε την εφαρμογή μας. Πατώντας στο **Check for Update** η εφαρμογή ελέγχει την διαθεσιμότητα νέας έκδοσης και ενημερώνει τον χρήστη όπως φαίνεται στην τρίτη εικόνα.



Εικόνα 11: Ρύθμιση και ενημέρωση της εφαρμογής

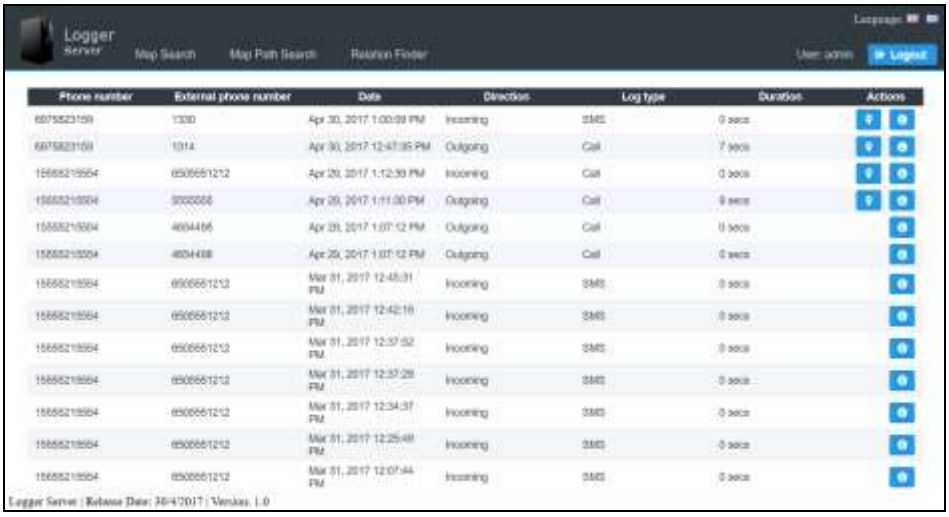
12.2 Σενάρια Χρήσης της εφαρμογής Logger Server

Παρακάτω παρατίθενται σενάρια χρήσης που περιγράφουν και απεικονίζουν την λειτουργικότητα της εφαρμογής Logger Server

12.2.1 Είσοδος του χρήστη στην εφαρμογή

Ο χρήστης εισάγει το όνομα χρήστη και τον κωδικό του Εικόνα 12. Εφόσον ο συνδυασμός ονόματος χρήστη – κωδικού είναι σωστός ο χρήστης εισέρχεται στην εφαρμογή και βλέπει την αρχική σελίδα Εικόνα 13.

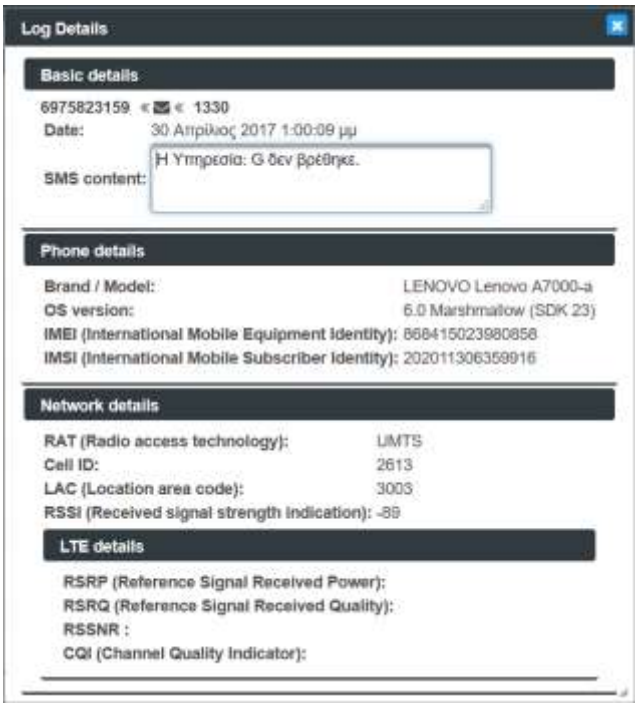
Εικόνα 12: Σελίδα για την είσοδο του χρήστη στην εφαρμογή



Εικόνα 13: Αρχική σελίδα εφαρμογής

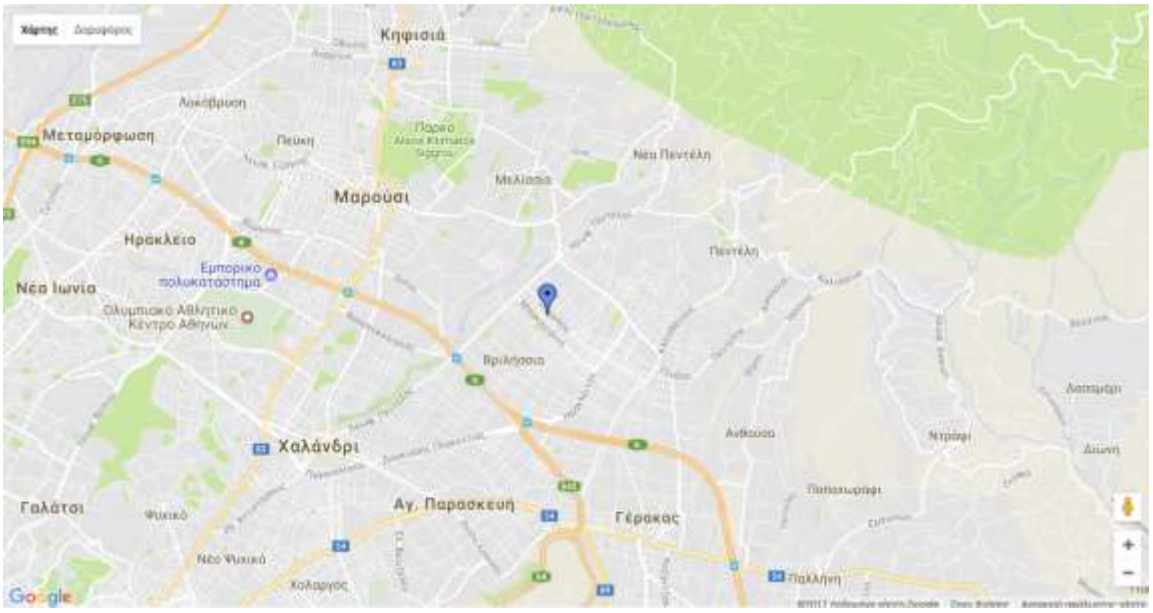
12.2.2 Εμφάνιση των πληροφοριών της πιο πρόσφατης καταγραφής και απεικόνισης της στον χάρτη

Από την Αρχική σελίδα Εικόνα 13 ο χρήστης βλέπει τις 15 πιο πρόσφατες καταγραφές. Πατώντας το δεξί κουμπί από την στήλη Actions εμφανίζεται το σύνολο της πληροφορίας για την συγκεκριμένη καταγραφή Εικόνα 14.



Εικόνα 14: Σύνολο πληροφορίας για μια συγκεκριμένη καταγραφή

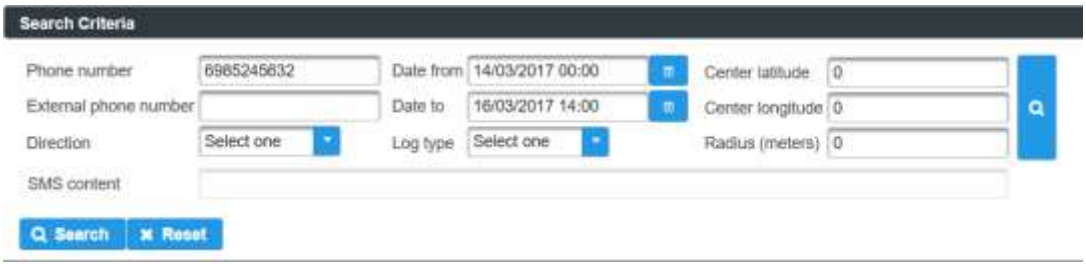
Πατώντας το αριστερό κουμπί από την στήλη Actions  (το οποίο είναι διαθέσιμο μόνο αν έχει γίνει καταγραφή της θέσης) εμφανίζεται η τοποθεσία της συγκεκριμένης καταγραφής στον χάρτη Εικόνα 15.



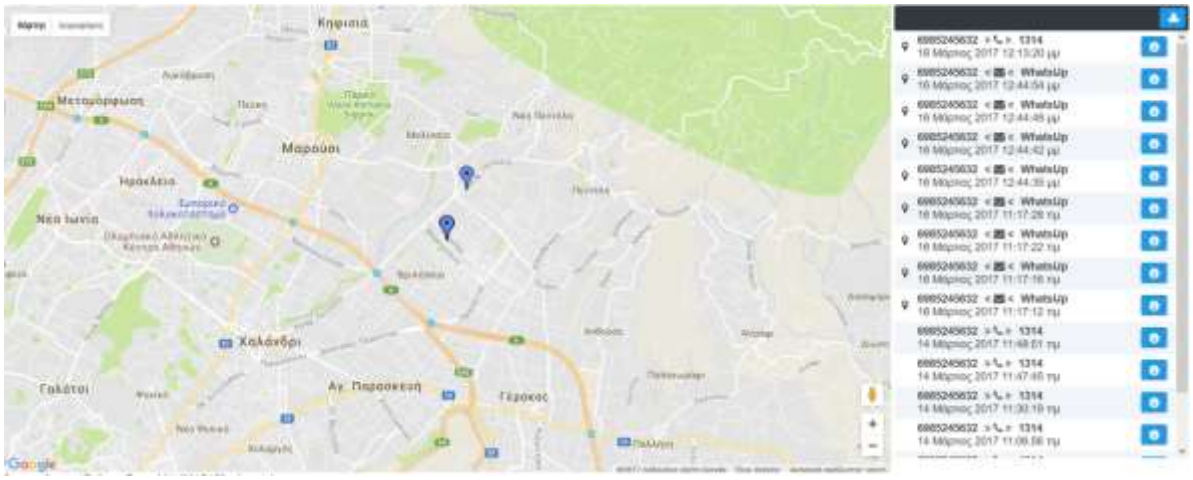
Εικόνα 15: Απεικόνιση καταγραφής στον χάρτη

12.2.3 Αναζήτηση των καταγραφών από ένα συγκεκριμένο αριθμό τηλεφώνου σε ένα συγκεκριμένο χρονικό διάστημα

Από το μενού της εφαρμογής ο χρήστης επιλέγει την αναζήτηση στον χάρτη (Map Search). Εισάγει τα κριτήρια, όπως φαίνεται στην Εικόνα 16, και πατώντας το κουμπί Search εμφανίζονται τα αποτελέσματα Εικόνα 17



Εικόνα 16: Κριτήρια για την αναζήτηση καταγραφών από ένα συγκεκριμένο αριθμό τηλεφώνου σε ένα συγκεκριμένο χρονικό διάστημα

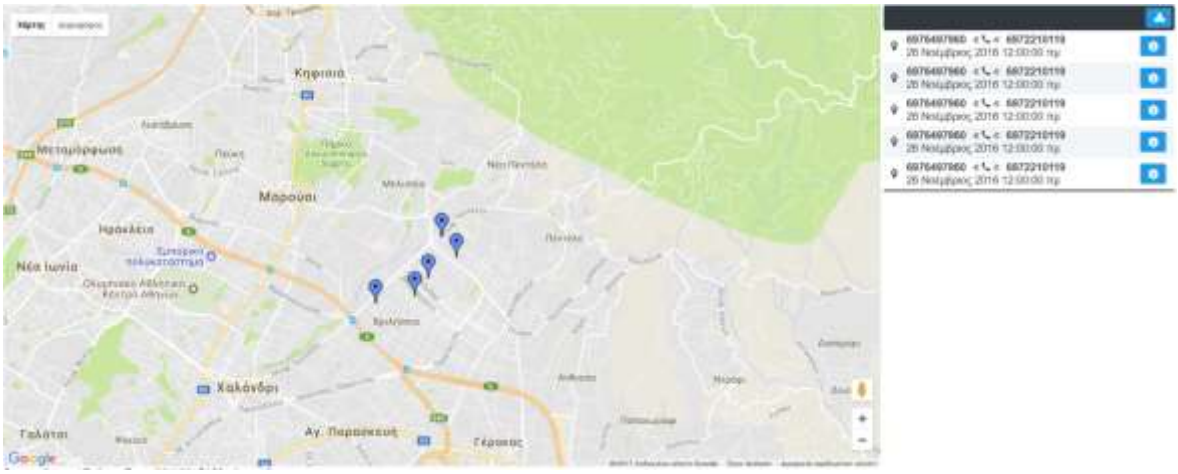


Εικόνα 17: Αποτελέσματα από την αναζήτηση καταγραφών από ένα συγκεκριμένο αριθμό τηλεφώνου σε ένα συγκεκριμένο χρονικό διάστημα

12.2.4 Αναζήτηση των καταγραφών μεταξύ δύο αριθμών τηλεφώνου

Από το μενού της εφαρμογής ο χρήστης επιλέγει την αναζήτηση στον χάρτη (Map Search). Εισάγει τα κριτήρια, όπως φαίνεται στην Εικόνα 18, και πατώντας το κουμπί Search εμφανίζονται τα αποτελέσματα Εικόνα 19

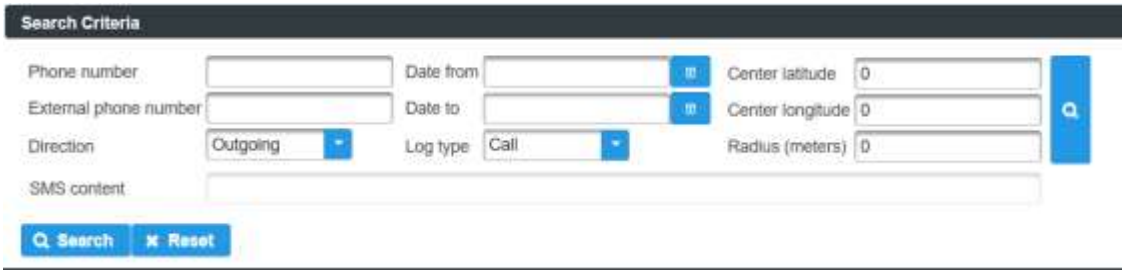
Εικόνα 18: Κριτήρια για την αναζήτηση καταγραφών μεταξύ δύο αριθμών τηλεφώνου



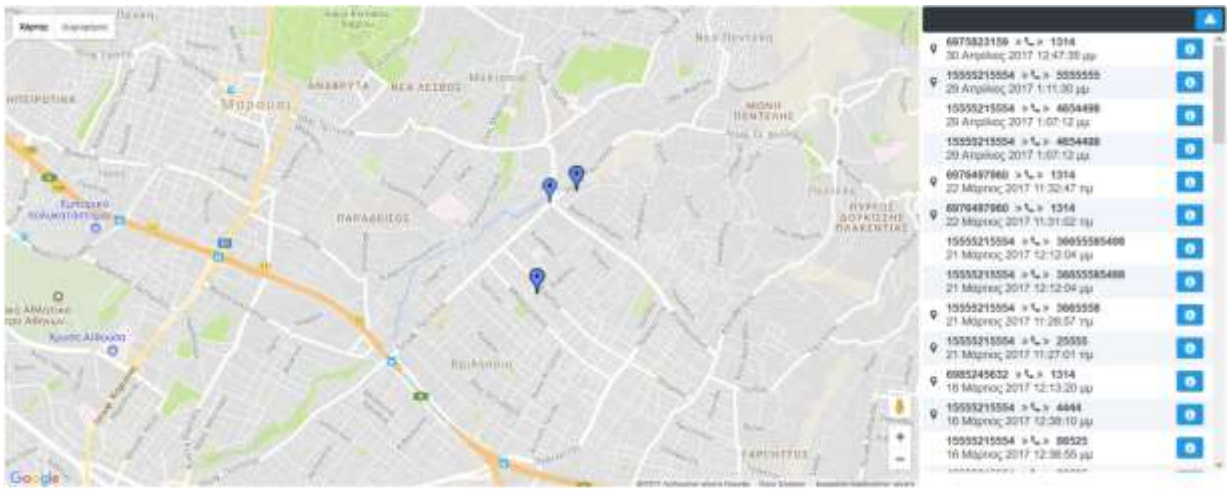
Εικόνα 19: Αποτελέσματα από την αναζήτηση καταγραφών μεταξύ δύο αριθμών τηλεφώνου

12.2.5 Αναζήτηση των καταγραφών όλων των εξερχόμενων κλήσεων

Από το μενού της εφαρμογής ο χρήστης επιλέγει την αναζήτηση στον χάρτη (Map Search). Εισάγει τα κριτήρια, όπως φαίνεται στην Εικόνα 20, και πατώντας το κουμπί Search εμφανίζονται τα αποτελέσματα Εικόνα 21



Εικόνα 20: Κριτήρια για την αναζήτηση καταγραφών όλων των εξερχόμενων κλήσεων

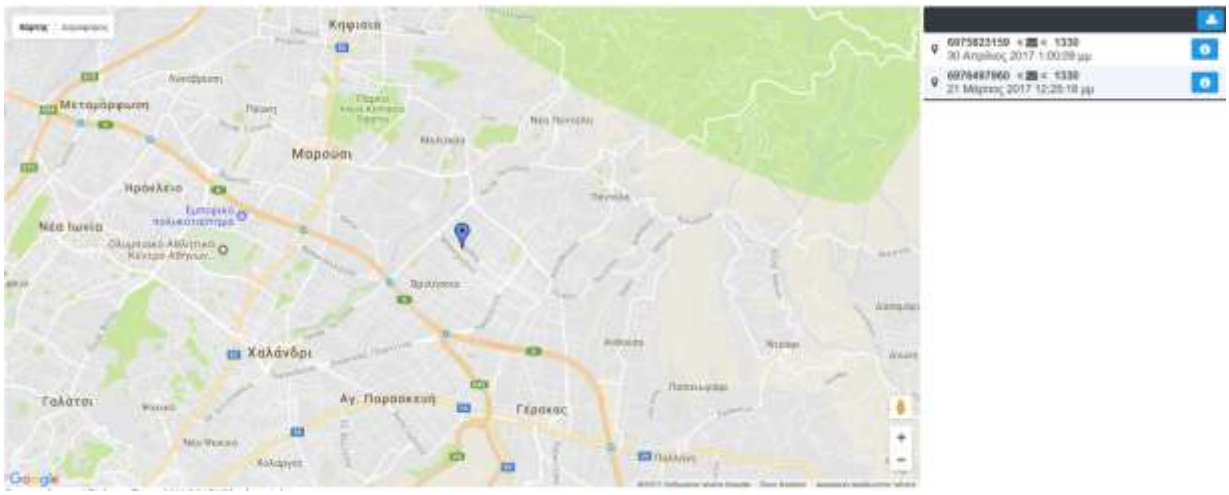


Εικόνα 21: Αποτελέσματα από την αναζήτηση καταγραφών όλων των εξερχόμενων κλήσεων

12.2.6 Αναζήτηση των καταγραφών που είναι SMS και περιέχουν την λέξη «Υπηρεσία»

Από το μενού της εφαρμογής ο χρήστης επιλέγει την αναζήτηση στον χάρτη (Map Search). Εισάγει τα κριτήρια, όπως φαίνεται στην Εικόνα 22, και πατώντας το κουμπί Search εμφανίζονται τα αποτελέσματα Εικόνα 23

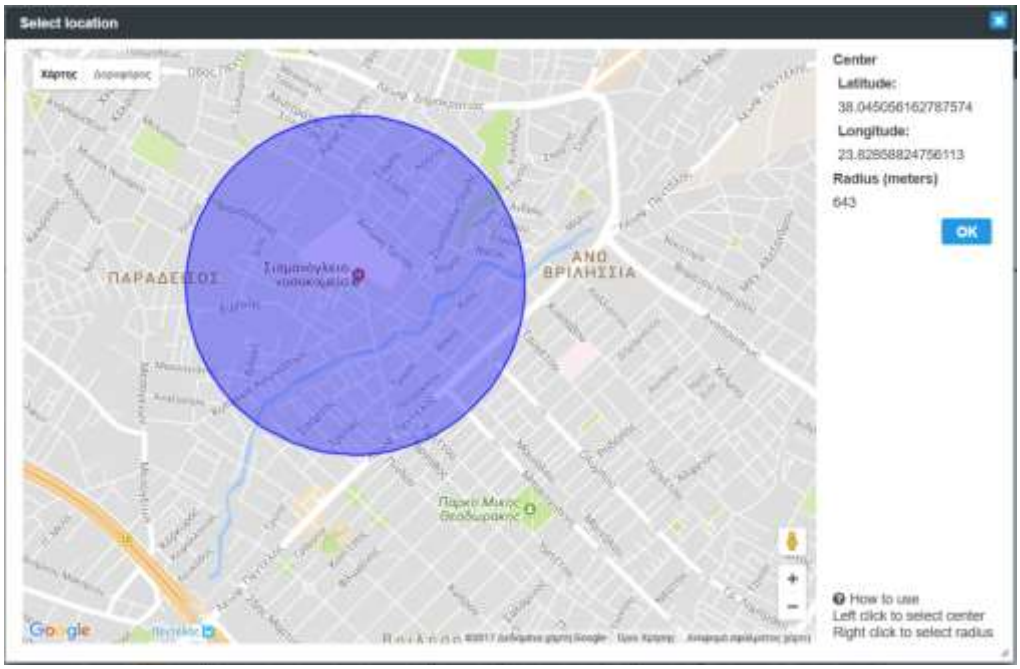
Εικόνα 22: Κριτήρια για την αναζήτηση καταγραφών που είναι SMS και περιέχουν την λέξη «Υπηρεσία»



Εικόνα 23: Αποτελέσματα από την αναζήτηση καταγραφών που είναι SMS και περιέχουν την λέξη «Υπηρεσία»

12.2.7 Αναζήτηση των καταγραφών σε μια συγκεκριμένη περιοχή

Από το μενού της εφαρμογής ο χρήστης επιλέγει την αναζήτηση στον χάρτη (Map Search). Πατάει το μεγεθυντικό φακό στο δεξί μέρος των κριτηρίων. Με το αριστερό click επιλέγει το κέντρο και με το δεξί click την ακτίνα της περιοχής στην οποία θέλει να αναζητήσει τις καταγραφές και πατάει OK Εικόνα 24. Τα κριτήρια της αναζήτησης ενημερώνονται αυτόματα Εικόνα 25 και πατώντας το κουμπί Search εμφανίζονται τα αποτελέσματα Εικόνα 26.



Εικόνα 24: Επιλογή της περιοχής στην οποία θέλουμε να αναζητήσουμε καταγραφές

Εικόνα 25: Κριτήρια για την αναζήτηση καταγραφών σε μια συγκεκριμένη περιοχή

Εικόνα 26: Αποτελέσματα από την αναζήτηση καταγραφών σε μια συγκεκριμένη περιοχή

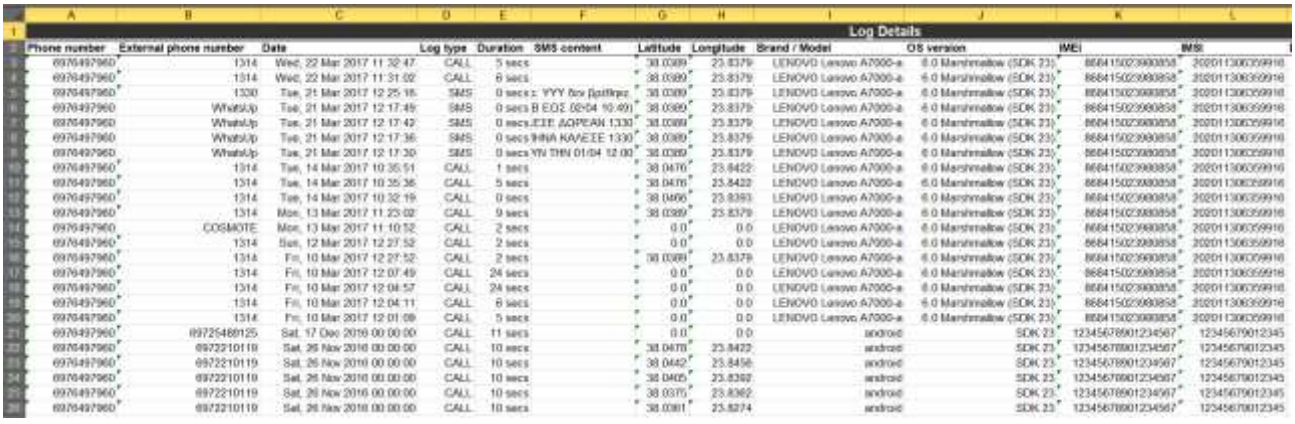
12.2.8 Απεικόνιση θέσης που έχει υπολογιστεί με χρήση του Mozilla Location Service

Από το μενού της εφαρμογής ο χρήστης επιλέγει την αναζήτηση στον χάρτη (Map Search). Ο χρήστης αναζητεί μια καταγραφή για την οποία δεν έχει γίνει καταγραφή της θέσης από την συσκευή, ωστόσο έχει η υπολογιστεί μια προσέγγιση της θέσης με χρήση του Mozilla Location Service. Ο χρήστης επιλέγει το pin που του έχει εμφανιστεί στον χάρτη και βλέπει την περιοχή μέσα στην οποία έγινε η καταγραφή Εικόνα 27.

Εικόνα 27: Απεικόνιση θέσης που έχει υπολογιστεί με χρήση του Mozilla Location Service

12.2.9 Εξαγωγή των αποτελεσμάτων σε αρχείο Excel

Από το μενού της εφαρμογής ο χρήστης επιλέγει την αναζήτηση στον χάρτη (Map Search). Κάνοντας οποιαδήποτε αναζήτηση ο χρήστης έχει την δυνατότητα να εξάγει τα αποτελέσματα σε ένα αρχείο Excel πατώντας το κουμπί  στην πάνω δεξιά πλευρά των αποτελεσμάτων. Στην Εικόνα 28 φαίνεται ένα ενδεικτικό αρχείο Excel .

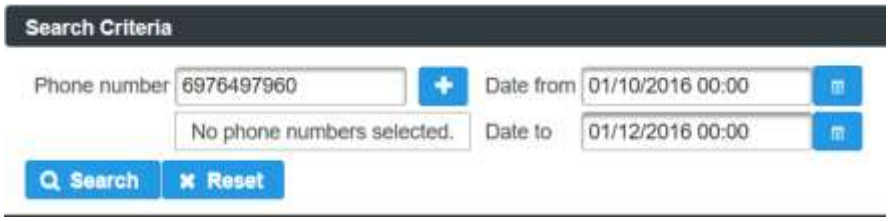


Log Details												
Phone number	External phone number	Date	Log type	Duration	SMS content	Latitude	Longitude	Brand / Model	OS version	IMEI	IMSI	
6976497960	1314	Wed, 22 Mar 2017 11:32:47	CALL	5 secs		38.0389	23.8379	LENOVO Lenovo A7000-a	6.0 Marshmallow (SDK 23)	868415023880858	202011306359916	
6976497960	1314	Wed, 22 Mar 2017 11:31:02	CALL	6 secs		38.0389	23.8379	LENOVO Lenovo A7000-a	6.0 Marshmallow (SDK 23)	868415023880858	202011306359916	
6976497960	1330	Tue, 21 Mar 2017 12:25:16	SMS	0 secs	ε. ΥΥΥ δεν βρίθηκε	38.0389	23.8379	LENOVO Lenovo A7000-a	6.0 Marshmallow (SDK 23)	868415023880858	202011306359916	
6976497960	WhatsUp	Tue, 21 Mar 2017 12:17:49	SMS	0 secs	Β ΕΟΣ 08=04 10:49)	38.0389	23.8379	LENOVO Lenovo A7000-a	6.0 Marshmallow (SDK 23)	868415023880858	202011306359916	
6976497960	WhatsUp	Tue, 21 Mar 2017 12:17:42	SMS	0 secs	ΕΣΕ ΔΩΡΕΑΝ 1330	38.0389	23.8379	LENOVO Lenovo A7000-a	6.0 Marshmallow (SDK 23)	868415023880858	202011306359916	
6976497960	WhatsUp	Tue, 21 Mar 2017 12:17:36	SMS	0 secs	ΘΑ ΚΑΛΕΣΕ 1330	38.0389	23.8379	LENOVO Lenovo A7000-a	6.0 Marshmallow (SDK 23)	868415023880858	202011306359916	
6976497960	WhatsUp	Tue, 21 Mar 2017 12:17:30	SMS	0 secs	ΥΝ ΤΗΝ 01/04 12:00	38.0389	23.8379	LENOVO Lenovo A7000-a	6.0 Marshmallow (SDK 23)	868415023880858	202011306359916	
6976497960	1314	Tue, 14 Mar 2017 10:35:51	CALL	1 secs		38.0476	23.8422	LENOVO Lenovo A7000-a	6.0 Marshmallow (SDK 23)	868415023880858	202011306359916	
6976497960	1314	Tue, 14 Mar 2017 10:35:36	CALL	5 secs		38.0476	23.8422	LENOVO Lenovo A7000-a	6.0 Marshmallow (SDK 23)	868415023880858	202011306359916	
6976497960	1314	Tue, 14 Mar 2017 10:32:19	CALL	0 secs		38.0490	23.8393	LENOVO Lenovo A7000-a	6.0 Marshmallow (SDK 23)	868415023880858	202011306359916	
6976497960	1314	Mon, 13 Mar 2017 11:23:02	CALL	9 secs		38.0389	23.8379	LENOVO Lenovo A7000-a	6.0 Marshmallow (SDK 23)	868415023880858	202011306359916	
6976497960	COSMOTE	Mon, 13 Mar 2017 11:10:52	CALL	2 secs		0.0	0.0	LENOVO Lenovo A7000-a	6.0 Marshmallow (SDK 23)	868415023880858	202011306359916	
6976497960	1314	Sun, 12 Mar 2017 12:27:52	CALL	2 secs		0.0	0.0	LENOVO Lenovo A7000-a	6.0 Marshmallow (SDK 23)	868415023880858	202011306359916	
6976497960	1314	Fri, 10 Mar 2017 12:27:52	CALL	2 secs		38.0389	23.8379	LENOVO Lenovo A7000-a	6.0 Marshmallow (SDK 23)	868415023880858	202011306359916	
6976497960	1314	Fri, 10 Mar 2017 12:07:49	CALL	24 secs		0.0	0.0	LENOVO Lenovo A7000-a	6.0 Marshmallow (SDK 23)	868415023880858	202011306359916	
6976497960	1314	Fri, 10 Mar 2017 12:04:57	CALL	24 secs		0.0	0.0	LENOVO Lenovo A7000-a	6.0 Marshmallow (SDK 23)	868415023880858	202011306359916	
6976497960	1314	Fri, 10 Mar 2017 12:04:11	CALL	6 secs		0.0	0.0	LENOVO Lenovo A7000-a	6.0 Marshmallow (SDK 23)	868415023880858	202011306359916	
6976497960	1314	Fri, 10 Mar 2017 12:01:09	CALL	5 secs		0.0	0.0	LENOVO Lenovo A7000-a	6.0 Marshmallow (SDK 23)	868415023880858	202011306359916	
6976497960	09725488125	Sat, 17 Dec 2016 00:00:00	CALL	11 secs		0.0	0.0	android	SDK 23	123456789012345	123456789012345	
6976497960	6972210119	Sat, 26 Nov 2016 00:00:00	CALL	10 secs		38.0476	23.8422	android	SDK 23	123456789012345	123456789012345	
6976497960	6972210119	Sat, 26 Nov 2016 00:00:00	CALL	10 secs		38.0442	23.8456	android	SDK 23	123456789012345	123456789012345	
6976497960	6972210119	Sat, 26 Nov 2016 00:00:00	CALL	10 secs		38.0405	23.8392	android	SDK 23	123456789012345	123456789012345	
6976497960	6972210119	Sat, 26 Nov 2016 00:00:00	CALL	10 secs		38.0375	23.8362	android	SDK 23	123456789012345	123456789012345	
6976497960	6972210119	Sat, 26 Nov 2016 00:00:00	CALL	10 secs		38.0381	23.8274	android	SDK 23	123456789012345	123456789012345	

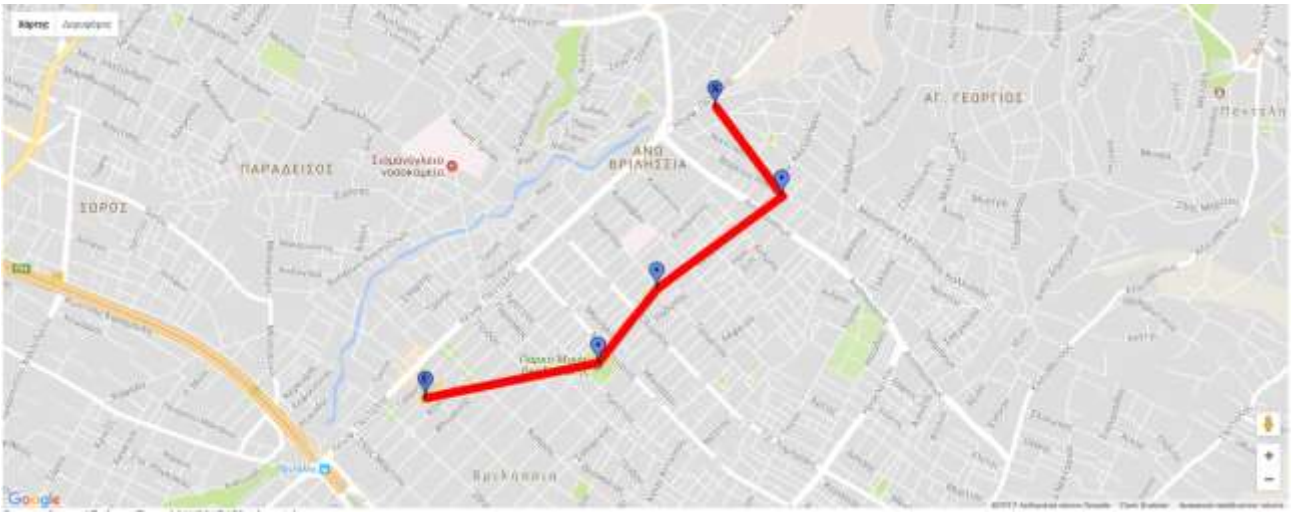
Εικόνα 28: Αρχείο Excel που περιέχει τα αποτελέσματα μιας αναζήτησης

12.2.10 Αναζήτηση μονοπατιού για έναν αριθμό τηλεφώνου σε ένα συγκεκριμένο χρονικό διάστημα

Από το μενού της εφαρμογής ο χρήστης επιλέγει την αναζήτηση μονοπατιού (Map Path Search). Εισάγει των αριθμό τηλεφώνου και τις ημερομηνίες, όπως φαίνεται στην Εικόνα 29, και πατώντας το κουμπί Search εμφανίζονται τα αποτελέσματα Εικόνα 30.



Εικόνα 29: Κριτήρια για την αναζήτηση μονοπατιού ενός αριθμού τηλεφώνου σε ένα συγκεκριμένο χρονικό διάστημα

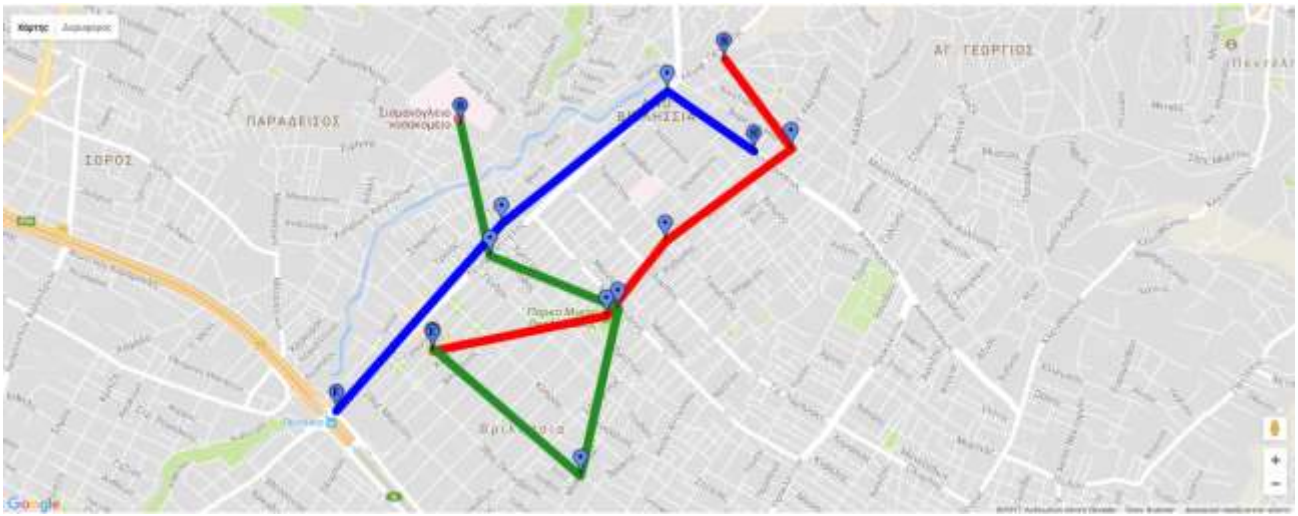


Εικόνα 30: Αποτελέσματα της αναζήτησης μονοπατιού ενός αριθμού τηλεφώνου σε ένα συγκεκριμένο χρονικό διάστημα

12.2.11 Αναζήτηση μονοπατιών για πολλαπλούς αριθμούς τηλεφώνου σε ένα συγκεκριμένο χρονικό διάστημα

Από το μενού της εφαρμογής ο χρήστης επιλέγει την αναζήτηση μονοπατιού (Map Path Search). Εισάγει έναν αριθμό τηλεφώνου και πατάει το κουμπί  . Εισάγει άλλους δύο αριθμούς τηλεφώνου και τις ημερομηνίες Εικόνα 31. Πατώντας το κουμπί Search εμφανίζονται ταυτόχρονα τα μονοπάτια για τους τρεις αριθμούς τηλεφώνου Εικόνα 32.

Εικόνα 31: Κριτήρια για την αναζήτηση πολλαπλών μονοπατιών σε ένα χρονικό διάστημα

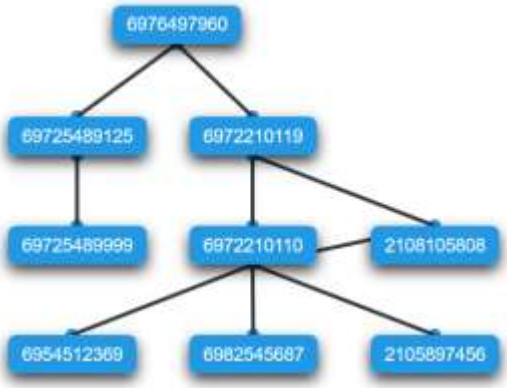


Εικόνα 32: Αποτελέσματα της αναζήτησης πολλαπλών μονοπατιών σε ένα συγκεκριμένο χρονικό διάστημα

12.2.12 Αναζήτηση σχέσης ενός αριθμού τηλεφώνου με άλλους αριθμούς τηλεφώνου σε ένα χρονικό διάστημα

Από το μενού της εφαρμογής ο χρήστης επιλέγει την εύρεση σχέσης (Relation Finder). Εισάγει τον αριθμό τηλεφώνου και τις ημερομηνίες, όπως φαίνεται στην Εικόνα 33, και πατώντας το κουμπί Search εμφανίζεται ο γράφος που απεικονίζει τις σχέσεις Εικόνα 34.

Εικόνα 33: Κριτήρια για την εύρεση σχέσης ενός αριθμού τηλεφώνου σε ένα χρονικό διάστημα

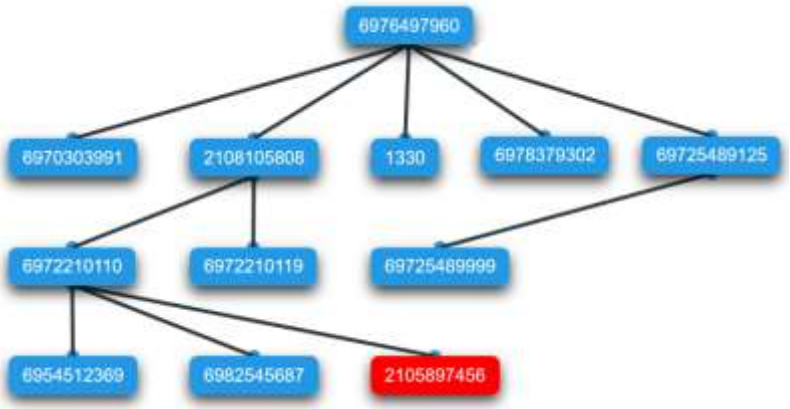


Εικόνα 34: Γράφος που απεικονίζει τις σχέσεις ενός αριθμού τηλεφώνου σε ένα χρονικό διάστημα

12.2.13 Αναζήτηση σχέσης μεταξύ δύο αριθμών τηλεφώνου

Από το μενού της εφαρμογής ο χρήστης επιλέγει την εύρεση σχέσης (Relation Finder). Εισάγει τους δύο αριθμούς τηλεφώνου, όπως φαίνεται στην Εικόνα 35, και πατώντας το κουμπί Search εμφανίζεται ο γράφος που απεικονίζει τις σχέσεις Εικόνα 36.

Εικόνα 35: Κριτήρια για την αναζήτηση σχέσης μεταξύ δύο αριθμών τηλεφώνου



Εικόνα 36: Γράφος που απεικονίζει τις σχέσεις μεταξύ δύο αριθμών τηλεφώνου

13 Συμπεράσματα και μελοντικές επεκτάσεις

Στις παραπάνω ενότητες παρουσιάσαμε την υλοποίηση ενός προσωπικού καταγραφέα. Είδαμε την διαδικασία που απαιτείτε για την εγκατάσταση του απαραίτητου λογισμικού καθώς και την εγκατάσταση των δύο εφαρμογών μας. Οι τελικές εφαρμογές ανταποκρίνονται πλήρως στις λειτουργικές τους απαιτήσεις καθώς τόσο η καταγραφή και αποστολή των δραστηριοτήτων από την εφαρμογή Android όσο και η αποθήκευση και επεξεργασία τους από τον εξυπηρετητή γίνονται επιτυχώς.
Ωστόσο κατά την υλοποίηση των δύο εφαρμογών υπήρξαν αρκετές δυσκολίες. Κυρίως κατά την ανάπτυξή της εφαρμογής Android παρουσιάστηκαν προβλήματα τα οποία περιόρισαν τις δυνατότητες της εφαρμογής και/ή δυσκόλεψαν την ανάπτυξη της. Παρακάτω παρατίθενται τα σημαντικότερα:

- **Η κάρτα SIM δεν περιέχει πάντα τον αριθμό τηλεφώνου (MSISDN)**

  Ανάλογα με τον πάροχο μπορεί μια κάρτα SIM να έχει ή να μην έχει τον αριθμό τηλεφώνου. Αυτό προκαλεί προβλήματα στην εφαρμογή μας καθώς στην περίπτωση που δεν είναι διαθέσιμος ο αριθμός τηλεφώνου στην SIM τότε ο χρήστης πρέπει να τον εισάγει μόνος του στην εφαρμογή. Λόγω του παραπάνω ο χρήστης μπορεί να βάλει έναν ψεύτικο αριθμό τηλεφώνου ο οποίος θα οδηγήσει στην αποστολή μη έγκυρων καταγραφών στον εξυπηρετητή.

- **Αδυναμία καταγραφής των ιστοσελίδων που επισκέφτηκε ο χρήστης**

  Μέχρι και την έκδοση SDK 22 υπήρχε δυνατότητα καταγραφής των επισκεπτόμενων ιστοσελίδων μέσω του ιστορικού του Google Chrome που αποτελεί τον προεγκατεστημένο και διασημότερο φυλλομετρητή των συσκευών Android. Φυσικά με αυτόν τον τρόπο δεν θα καταγράφονταν ιστοσελίδες που ο χρήστης επισκέφτηκε χρησιμοποιώντας ιδιωτική περιήγηση ή με κάποιον άλλο φυλλομετρητή. Ωστόσο από το SDK 23 και για τις νεότερες εκδόσεις Android η υποστήριξη για την πρόσβαση στο ιστορικό του Google Chrome σταμάτησε (13). Λόγω των παραπάνω δυσκολιών και της απαίτησης ότι η εφαρμογή μας θέλουμε να είναι συμβατή με τις νεότερες εκδόσεις Android, αυτή η λειτουργικότητα δεν υλοποιήθηκε.

Βεβαίως, υπάρχουν αρκετά σημεία στις εφαρμογές μας που θα μπορούσαν να βελτιωθούν. Παρακάτω παραθέτουμε προτάσεις για μελλοντικές επεκτάσεις που μπορούν να γίνουν:

- **Εισαγωγή της εφαρμογής Android στο Google Play**

  Για να μπορέσει ο χρήστης να εγκαταστήσει την εφαρμογή Android στην συσκευή του χρειάζεται να κατεβάσει τον πηγαίο κώδικα από το Github να τον μεταγλωττίσει με κάποιο εργαλείο στον

υπολογιστή του και τέλος να εγκαταστήσει το παραγόμενο apk στην συσκευή του. Επιπλέον κατά την ενημέρωση της εφαρμογής το apk με την νέα έκδοση κατεβαίνει στην συσκευή του χρήστη αυτόματα μέσα από την εφαρμογή ωστόσο, ο χρήστης πρέπει μόνος του να την εγκαταστήσει. Αυτό συμβαίνει διότι οι νεότερες εκδόσεις Android δεν επιτρέπουν την αυτόματη εγκατάσταση μη-έμπιστων εφαρμογών. Και τα δύο παραπάνω προβλήματα θα λυνόντουσαν με την προσθήκη της εφαρμογής στο Google Play καθώς μπορεί να αναλάβει τόσο την αρχική εγκατάσταση της εφαρμογής όσο και την ενημέρωση της με μια φιλική ως προς τον χρήστη διαδικασία.

- **Δικαιώματα χρήσης και εμπορική εκμετάλλευση**

  Οι εφαρμογές μας όπως έχουν δημιουργηθεί μέχρι αυτήν την στιγμή, έχουν την δυνατότητα να διατεθούν στο κοινό και να χρησιμοποιηθούν για εμπορική εκμετάλλευση. Αυτό οφείλεται στο γεγονός ότι όλο το λογισμικό που έχει χρησιμοποιηθεί δεν απαιτεί την αγορά άδειας χρήσης για εμπορική του χρήση.

- **Αντικατάσταση του JBoss AS 7 με το Wildfly 10**

  Η Red Hat έχει σταματήσει την υποστήριξη και ενημέρωση του JBoss AS 7**.** Ωστόσο υπάρχει ο Wildfly 10 (14) που αποτελεί την συνέχεια του. Για αυτό τον λόγο θα ήταν μια χρήσιμη επέκταση η μεταφορά της εφαρμογής του εξυπηρετητή από τον JBoss AS 7 στο Wildfly 10.

- **Χρήση του Docker για την αυτόματη δημιουργία του περιβάλλοντος του εξυπηρετητή**

  Όπως φαίνεται από τις ενότητες 6, 8 και 10.2.1 η εγκατάσταση του συνολικού λογισμικού καθώς και της εφαρμογής του εξυπηρετητή χρειάζεται πολλά βήματα τα οποία απαιτούν προσοχή και έναν βαθμό εμπειρίας. Αυτό είναι ικανό να αποτρέψει χρήστες από το να εγκαταστήσουν την εφαρμογή σε ένα δικό τους μηχάνημα. Για την απλοποίηση της παραπάνω διαδικασίας μπορεί να χρησιμοποιηθεί το εργαλείο Docker (15). Όλη η διαδικασία εγκατάστασης μπορεί να περιγραφεί σε ένα αρχείο Dockerfile. Ο τελικός χρήστης το μόνο που χρειάζεται είναι να εγκαταστήσει το εργαλείο Docker στο μηχάνημα που θέλει να εγκατασταθεί η εφαρμογή του εξυπηρετητή. Το Docker μέσα από τις εντολές που θα περιγράφονται στο αρχείο Dockerfile θα εγκαταστήσει το περιβάλλον (JAVA, Application Server, Database) αθτοματοποιώντας την διαδικασία.

- **Χρήση της εφαρμογής του εξυπηρετητή από έναν τηλεπικοινωνιακό πάροχο**

  Η εφαρμογή του εξυπηρετητή θα μπορούσε αντί να χρησιμοποιεί τα δεδομένα από την δικιά της βάση δεδομένων να αξιοποιήσει αντίστοιχες πληροφορίες, οι οποίες είναι διαθέσιμες μέσω των CDRs (Call Detail Records). Με αυτόν τον τρόπο ένας πάροχος κινητών τηλεπικοινωνιών –μετά από εισαγγελική παραγγελία- θα μπορούσε να παρέχει άμεσα και γρήγορα τη λίστα «επικοινωνιών» συγκεκριμένου αριθμού κλήσης (MSISDN) –παρέχοντας και πληροφορία θέσης- αλλά και να εντοπίσει πιθανή «διασύνδεση» του συγκεκριμένου συνδρομητή με «ύποπτα» MSISDNs (π.χ.

άτομα που ανήκουν σε τρομοκρατικές οργανώσεις) και με τους οποίους δεν υπάρχει άμεση «επαφή».

14 Παραπομπές

1. Okeanos. https://okeanos.grnet.gr.

2. Google Maps API. https://developers.google.com/maps/documentation/android-api/current-place-tutorial.

3. Google Developer Console. https://console.developers.google.com.

4. Git Client. https://git-scm.com/downloads/guis/.

5. Github: Logger Client. https://github.com/valmas/Logger.git.

6. Android Studio. https://developer.android.com/studio/install.html.

7. Gradle. https://gradle.org/.

8. Java 7. http://www.oracle.com/technetwork/java/javase/downloads/java-archive-downloads-javase7-521261.html.

9. JBoss AS 7. http://jbossas.jboss.org/downloads.

10. Github: Logger Server. https://github.com/valmas/logger-server.git.

11. Apache Maven. https://maven.apache.org/download.cgi.

12. MySQL Server. https://dev.mysql.com/downloads/mysql/.

13. Android dropped support for Google chrome bookmarks. https://developer.android.com/about/versions/marshmallow/android-6.0-changes.html#behavior-bookmark-browser.

14. WildFly Application Server. http://wildfly.org/.

15. Docker. https://www.docker.com/.