ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Τα Οχήματα μου – εξοδολόγια

Του φοιτητή

Τοπαλίδη Παντελή

Αρ. Μητρώου: 05/2892

Επιβλέπων καθηγητής

Κλεφτούρης Δημήτριος

Θεσσαλονίκη 2014

ΠΡΟΛΟΓΟΣ

Η εργασία αυτή παρουσιάζει την χρησιμότητα καταχώρησης των πετρελεύσεων, των σέρβις και των ασφαλειών ενός ή παραπάνω οχημάτων καθώς αναλύει και αναπτύσσει τεχνικές και αλγορίθμους με τους οποίους μπορεί να επιτευχθούν τα παραπάνω σε κινητές συσκευές και tablets. Αρχικά περιγράφονται τα εργαλεία ανάπτυξης της εφαρμογής καθώς και ο λειτουργικός και εικαστικός του χαρακτήρας. Ακολουθεί περιγραφή των τεχνικών και των αλγορίθμων που χρησιμοποιήθηκαν. Για την επίτευξη του χρησιμοποιήθηκαν επίσης σύγχρονες τεχνολογίες android, τοπική βάση δεδομένων συμβατή με android συσκευές χρήσιμη για την αποθήκευση όλων των δεδομένων που καταχωρεί ο χρήστης. Στην συνέχεια περιγράφεται η εφαρμογή που υλοποιήθηκε για τους παραπάνω σκοπούς, καθώς και αναφέρεται ο κώδικας του σε γλώσσα προγραμματισμού java.

ΠΕΡΙΛΗΨΗ

Στην παρούσα πτυχιακή εργασία περιγράφονται και αναλύονται τεχνικές ανάπτυξης εφαρμογών για smartphones καθώς με χρήση της πλατφόρμας Android SDK.

Στην εισαγωγή αναλύεται και παρουσιάζεται η ραγδαία ανάπτυξη της αγοράς των εφαρμογών για mobile πλατφόρμες και τις προοπτικές που ανοίγονται για τους νέους προγραμματιστές στην σύγχρονη εποχή. Γίνεται αναφορά στις πλέον αναπτυσσόμενες πλατφόρμες και σύγκριση αυτών καθώς και των πλεονεκτημάτων και μειονεκτημάτων που παρουσιάζουν, δίνοντας έμφαση στην πλατφόρμα ανοιχτού κώδικα Android SDK.

Στην συνέχεια περιγράφεται η χρήση και ενσωμάτωση των εργαλείων στο σύστημα ανάπτυξης εφαρμογών του Android, και γίνεται ανάλυση στις πιο σημαντικές παραμέτρους τους χρησιμοποιώντας την αντικειμενοστραφής γλώσσα προγραμματισμού JAVA. Τέλος, χρησιμοποιώντας τα παραπάνω εργαλεία και τις εμπειρίες που αποκτήθηκαν από την μελέτη αυτών, αναπτύχθηκε μια εφαρμογή διεπαφής χρήστη πάνω στην πλατφόρμα της Google, Android SDK, και τέλος περιγράφονται οι δυσκολίες που συναντήθηκαν κατά την ανάπτυξη αυτής.

Στόχος της εφαρμογής είναι να κρατάει ο χρήστης αρχείο των οχημάτων του και να αποθηκεύει πληροφορίες για αυτά με έμφαση τις πετρελαιεύσεις και το

κόστος αυτών και επιπλέον τα σέρβις και τις ασφάλειες του κάθε οχήματος. Έτσι θα έχει τη δυνατότητα να γνωρίζει το ετήσιο κόστος χρήσης κάθε οχήματος και το αντίστοιχο με διαφορετικό καύσιμο, επιλογή καίρια για εταιρίες με μεσαίους ή και μεγάλους στόλους που τους απασχολεί η μετάβαση σε άλλο καύσιμο, κατά πόσο συμφέρει και πόσος χρόνος απαιτείται για την απόσβεση μια τέτοιας επένδυσης.

ABSTRACT

In this paper described and analyzing the application development techniques for smartphones using the platform Android SDK. In the begining is analyzed and illustrated the rapid development for mobile platforms and the prospects about new developers in the modern technologies. There is a reference of most developing platforms comparisons, advantages and disadvantages of them.
Then it is described the using and the application development tools, the system of Android and the analyzing of the most important parameters, using an object-oriented programming language like JAVA. The aim of this application is the tracing information of vehicles like fueling and the cost of the services and insurance. As results, user knows the annual cost of vehicle using and the comparison by using another type of fuel.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή μου για την καθοδήγηση του καθώς και συνεργάτες και φίλους που με βοήθησαν στην βελτίωση της πτυχιακής εργασίας μου.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΙΣΑΓΩΓΗ

Περιλαμβάνει τους στόχους και σκοπούς της Π/Ε καθώς και περιγραφή των κεφαλαίων που ακολουθούν.

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή στο Λειτουργικό Σύστημα Android

1.1 Τι είναι το Android;

Το Android είναι ένα λειτουργικό σύστημα ανοιχτού κώδικα, βασισμένο στο Linux, για φορητές συσκευές όπως smartphones και tablets. Αναπτύχθηκε από την Google και αργότερα από την Open Handset Alliance η οποία είναι μια κοινοπραξία εταιριών λογισμικού, κατασκευής hardware και τηλεπικοινωνιών, οι οποίες είναι αφιερωμένες στην ανάπτυξη και εξέλιξη ανοιχτών προτύπων στις φορητές συσκευές. Η πρώτη παρουσίαση της πλατφόρμας Android έγινε στις 5 Νοεμβρίου 2007, παράλληλα με την ανακοίνωση της ίδρυσης του οργανισμού Open Handset Alliance. Η Google δημοσίευσε το μεγαλύτερο μέρος του κώδικα του Android υπό τους όρους της Apache License, μιας ελεύθερης άδειας λογισμικού. Το λογότυπο για το λειτουργικό σύστημα Android είναι ένα ρομπότ σε χρώμα πράσινου μήλου και σχεδιάστηκε από τη γραφίστρια Irina Blok. (Εικόνα 1)

Εικόνα 1, Λογότυπο Android

Τον Ιούλιο του 2005, η Google εξαγόρασε την Android Inc, μια μικρή εταιρεία με έδρα το Palo Alto στην California των ΗΠΑ. Εκείνη την εποχή ελάχιστα ήταν γνωστά για τις λειτουργίες της Android Inc, εκτός του ότι ανέπτυσσαν λογισμικό για κινητά τηλέφωνα. Αυτή ήταν η αρχή της φημολογίας περί σχεδίων της Google για να διεισδύσει στην αγορά κινητής τηλεφωνίας.

Στην Google, η ομάδα με επικεφαλής τον Andy Rubin ανέπτυξε μια κινητή πλατφόρμα που στηρίζεται στον πυρήνα του Linux, την οποία προώθησαν με την παροχή ενός ευέλικτου, αναβαθμίσιμου συστήματος. Έχει αναφερθεί ότι η Google είχε ήδη συγκεντρώσει μια σειρά από εταίρους hardware και software και επισήμανε στους παρόχους ότι ήταν ανοικτή σε διάφορους βαθμούς συνεργασίας εκ μέρους της. Έντυπα και ηλεκτρονικά μέσα ενημέρωσης σύντομα ανέφεραν φήμες ότι η Google ανέπτυσσε μια Google-branded συσκευή. Περισσότερες φήμες ακολούθησαν, αναφέροντας ότι η Google καθόριζε τις τεχνικές προδιαγραφές και έδειχνε πρωτότυπα στους κατασκευαστές κινητών τηλεφώνων και τους φορείς δικτύων. Τελικά η Google παρουσίασε το smartphone της Nexus One που χρησιμοποιεί το open source λειτουργικό σύστημα Android. Η συσκευή κατασκευάστηκε από την HTC , και έγινε διαθέσιμη στις 5 Ιανουαρίου 2010.

Εικόνα 1.1.2 Google Nexus One

1.2 Εφαρμογές Android

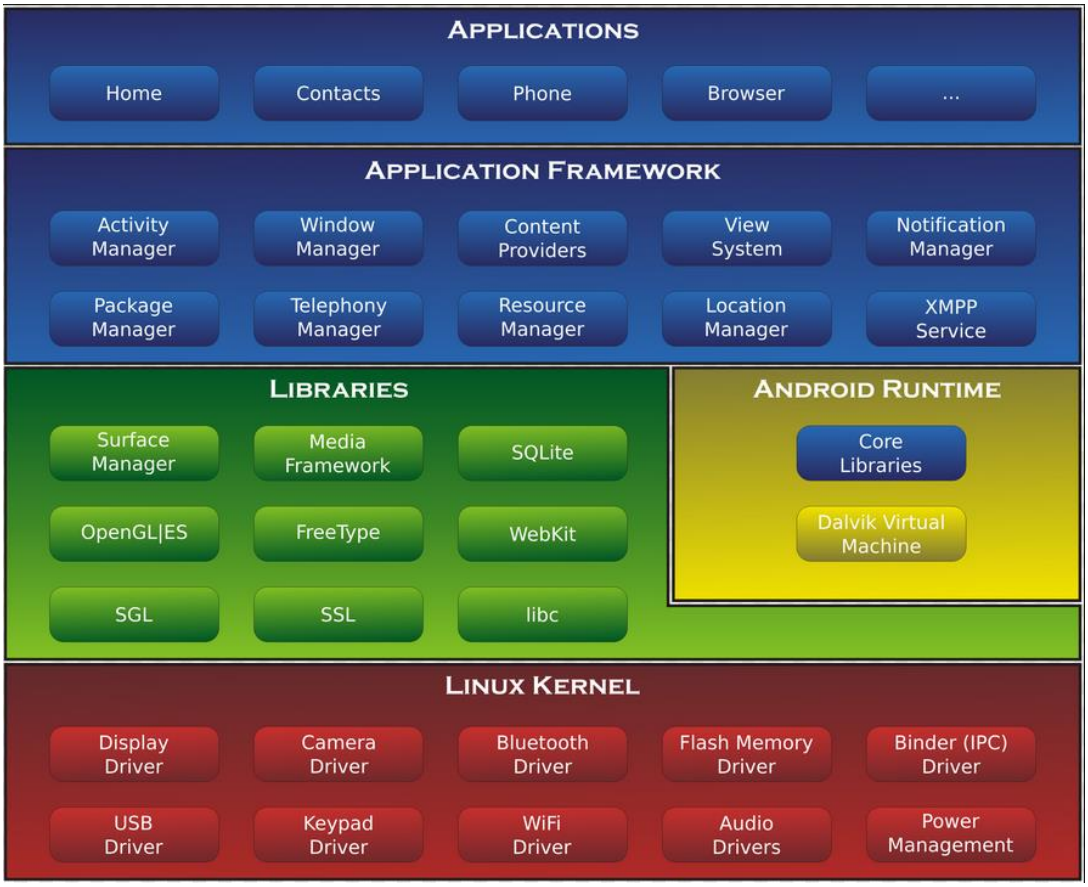
Το Android έχει πλέον αρκετά μεγάλη κοινότητα προγραμματιστών στόχος της οποίας είναι η γραφή κυρίως εφαρμογών με στόχο την επέκταση της

λειτουργικότητα των συσκευών. Οι εφαρμογές γράφονται σε μια προσαρμοσμένη έκδοση της JAVA και μπορεί κάνεις να κατεβάσει από το online κατάστημα Google Play (πρώην Android Market) της Google όπως και από άλλα sites. Μέχρι τον

Φεβρουάριο του 2012 περισσότερες από 450000 εφαρμογές ήταν διαθέσιμες για Android ενώ εκτιμάτε ότι ο αριθμός των downloads από το Android Market μέχρι το Δεκέμβριο του 2011 είχε υπερβεί τα 10 δισεκατομμύρια. Το Android είναι η πρώτη σε πωλήσεις παγκοσμίως πλατφόρμα για smartphones καθώς μέχρι το Φεβρουάριο του 2012 μετρούσε περισσότερες από 300 εκατομμύρια συσκευές σε λειτουργία.

1.3 Η Αρχιτεκτονική του Android

Το Android εκτός από λειτουργικό σύστημα, περιλαμβάνει υπηρεσίες διασύνδεσης με τις εφαρμογές (middleware) καθώς και τις κύριες εφαρμογές (core). Το σημαντικό χαρακτηριστικό του Android όπως φαίνεται και στο παρακάτω σχήμα είναι ότι διαχωρίζεται σε πέντε βασικά επίπεδα



Τα πέντε αυτά επίπεδα είναι τα εξής

- Πυρήνας λειτουργικού συστήματος Linux (Linux Kernel)
- Βασικές και εξειδικευμένες Βιβλιοθήκες (Libraries – Core Libraries)
- Την εικονική μηχανή Dalvik (Dalvik Virtual Machine)
- Το Android Runtime
- Το Application Framework

Πυρήνας λειτουργικού συστήματος

Το Android όπως αναφέρθηκε είναι βασισμένο στο λειτουργικό σύστημα linux και υποστηρίζει όλες τις κύριες λειτουργίες του όπως διαχείριση μνήμης, διεργασιών, δικτύου, ασφάλειας καθώς και οδηγοί υλικού (drivers). Οι τελευταίοι είναι υπεύθυνοι για την επικοινωνία του λογισμικού (software) με το hardware του mobile.

Ο πυρήνας του παρόλο που Android βασίζεται στον Linux Kernel περιέχει και τμήματα κώδικα τα οποία σχετίζονται μόνο με το Android καθώς και αρκετές βελτιστοποιήσεις που έχει κάνει η Google για να είναι ελαφρότερος και καταλληλότερος για κινητές συσκευές

Βασικές και εξειδικευμένες βιβλιοθήκες

Πρόκειται για τις βιβλιοθήκες του Android που ουσιαστικά αποτελούν τα APIs που μπορούν να χρησιμοποιούν οι προγραμματιστές για την ανάπτυξη εφαρμογών Android. Από μόνες τους δεν μπορούν να είναι λειτουργικές ή να είναι αυτόνομα προγράμματα αλλά χρησιμοποιούνται και ενσωματώνονται σε εφαρμογές ώστε να πραγματοποιήσουν διάφορες λειτουργίες για τις οποίες έχουν κατασκευαστεί. Η Χρήση τους γίνεται εμφανείς μέσω μέσω του Application Framework. Οι βιβλιοθήκες είναι σχεδόν όλες γραμμένες σε C και C++. Βασικές βιβλιοθήκες είναι οι εξής:

System C library

Πρόκειται για την standard βιβλιοθήκη συστήματος της C προσαρμοσμένη για κινητές συσκευές.

Surface Manager

Μέσω αυτής γίνεται η σύνθεση δισδιάστατων και τρισδιάστατων γραφικών

Βιβλιοθήκες Πολυμέσων (Media Framework)

Υποστηρίζει την αναπαραγωγή και εγγραφή ήχου και εικόνας διαφόρων format όπως MP3,MP4, MPEG3,JPEG,PNG και πολλών άλλων

SQLite

Πρόκειται για μια Σχεσιακή βάση δεδομένων η οποία παρόλο που δεν υποστηρίζει ορισμένους τύπους δεδομένων όπως Sql server Oracle ή MySql εντούτοις είναι πολύ ελαφριά και κατάλληλη για κινητά τηλέφωνα

SGL

Μηχανή Δισδιάστατων γραφικών

FreeType

Μηχανισμός ευκρίνειας των bitmap γραφικών και των γραμματοσειρών στο σύστημα

LibWebCore

Είναι μηχανή πλοήγησης στο διαδίκτυο η οποία χρησιμοποιείται από τον default browser του Android αλλά μπορεί να χρησιμοποιηθεί και από τις εφαρμογές για την αναπαράσταση και πλοήγηση διαδικτυακών τόπων

Βιβλιοθήκες 3D

Βασισμένη στο API του OpenGL ES χρησιμοποιείται για την εμφάνιση τρισδιάστατων γραφικών. Χρησιμοποιεί επιταχυντή υλικού γραφικών όταν υπάρχει, διαφορετικά θα χρησιμοποιηθεί μια βελτιωμένη έκδοση λογισμικού επιτάχυνσης γραφικών.

Dalvik Virtual Machine

Καθώς είναι γνωστό η εκτέλεση της Java γίνεται μέσω του JVM (Java Virtual Machine) ο οποίος εκτελεί το bycode της εφαρμογης (μεταφρασμένα αρχεία java) Στις συσκευές Android η εκτέλεση αυτή γίνεται μέσω της εικονικής μηχανής Dalvik. Η κάθε εφαρμογή εκτελείται μέσω της δικής εικονικής μηχανής οι οποίες με την σειρά τους εκτελούνται μέσω της Dalvik Virtual Machine. Ως συμπέρασμα σε αυτή την εικονική συσκευή μπορούν να εκτελούνται ταυτόχρονα πολλές εικονικές συσκευές γεγονός που κάνει το λειτουργικό android ευέλικτο και συμβατό με πολλές εφαρμογές μεταγλωτισμένες σε διαφορετικές εικονικές μηχανές.

Android Runtime

Περιλαμβάνει τις κύριες βιβλιοθήκες και την εικονική μηχανή Dalvik για τα οποία δόθηκε περιγραφή παραπάνω

Application Framework

Πρόκειται για σύνολο βιβλιοθηκών και αρχιτεκτονικών προγραμματισμού που προσφέρει στους developers την δυνατότητα να υλοποιήσουν εφαρμογές android. Μέσω αυτού δίνεται πρόσβαση και έλεγχος υλικού λειτουργιών και διεργασιών μέσω του πλούσιου API που διαθέτει. Το Application Framework επιτρέπει της χρήση βιβλιοθηκών και την επαναχρησιμοποίηση δομικών συστατικών και κώδικα καθώς και χρήση λειτουργιών εφαρμογών σε άλλες εφαρμογές.
Κατηγορίες δομικών μονάδων του Application Framework είναι οι εξής:

View System

Περιλαμβάνει Αντικείμενα GUI (Graphical User Interface) στα οποία μπορούν να εμφανιστούν διάφορα δεδομένα της εφαρμογής ή λήψης αποφάσεων από τον χρήστη.

Παράδειγμα τέτοιων αντικειμένων είναι οι λίστες (listview) , τα πλέγματα (gridview) , πεδία εισαγωγής κειμένου (textbox), πολλαπλής επιλογής (checkbox), μονής επιλογής (radiobutton), κουμπιά (button) και πολλά άλλα.

Content Provider

Μέσω αυτού μπορούν να επικοινωνήσουν διάφορες εφαρμογές ανταλλάσοντας δεδομένα μορφής και δομής που ορίζονται στον αντίστοιχο provider.

Resource Manager

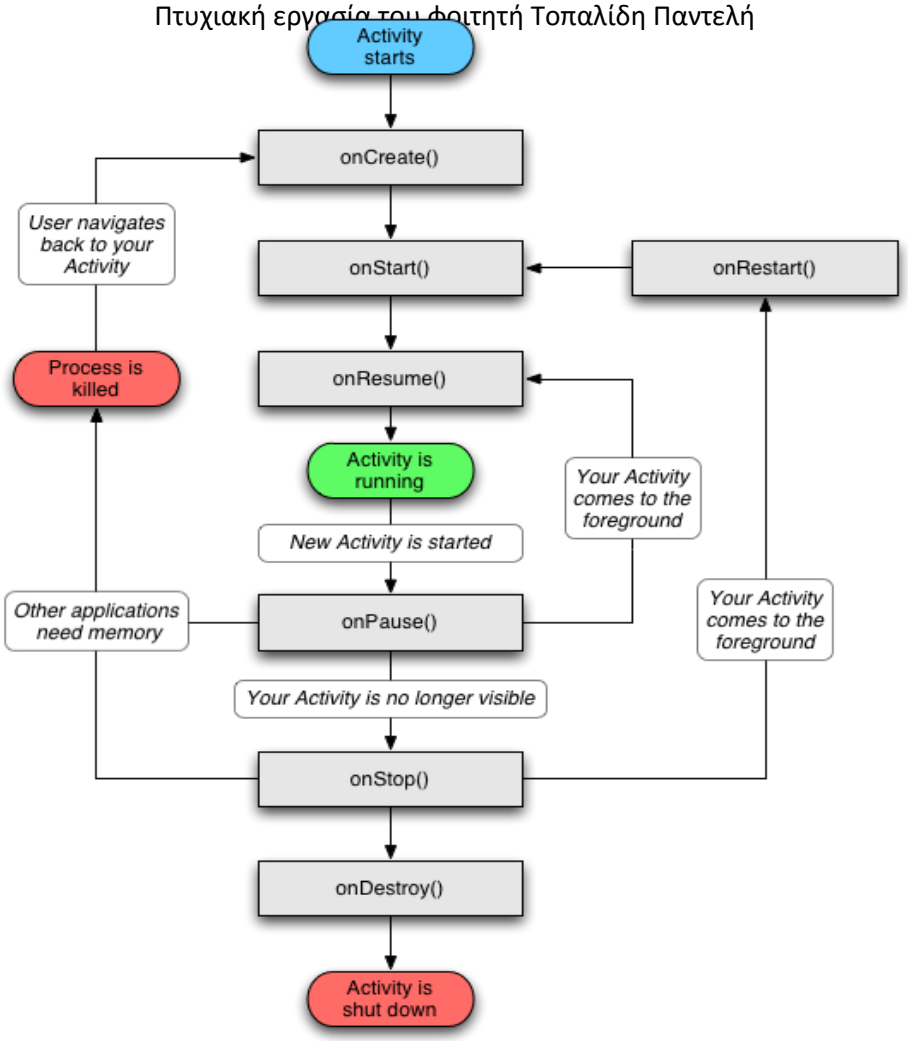
Δίνει την δυνατότητα σε πρόσβαση σε υλικό που δεν έχει σχέση με κώδικα αλλά είναι χρήσιμο για τις εφαρμογές. Παράδειγμα χρήσης εικόνων, ήχου, xml, κείμενα (string) κτλ.

Notification Manager

Διαχειριστής ειδοποιήσεων προς τον χρήστη των εφαρμογών. Οι ειδοποιήσεις μπορούν να πραγματοποιηθούν σε διάφορα σημεία ή τρόπους όπως στο notification bar, μηνύματα toast, δόνηση κινητού, φωτισμός οθόνης ή σε διάφορα άλλα τμήματα της οθόνης.

Activity Manager

Μια από τις κύριες δομικές μονάδες είναι ο Activity Manager. Μέσω αυτού μπορεί να γίνει πλοήγηση από activity σε activity κρατώντας στην μνήμη την σειρά εκτέλεσης τους. Επιπλέον διαχειρίζεται τον κύκλο ζωής ενός activity (onCreate, onStart, onResume, onPause, onStop, onRestart, onDestroy). Ο κύκλος ζωής ενός Activity περογράφεται παρακάτω



1.4 Δομή φακέλων/αρχείων Android Application

Οι εφαρμογές Android αποτελούνται από ένα σύνολο αρχείων και φακέλων οργανωμένα ανά ενότητες. Μετά την μεταγλώττιση (compile) μέσω του Android SDK προκύπτει το αρχείο apk το οποίο μπορεί να εγκατασταθεί και εκτελεστεί στην φορητή συσκευή του χρήστη. Κάθε εφαρμογή πρέπει να έχει μοναδικό όνομα πακέτου από το οποίο και αναγνωρίζει την μοναδικότητα της εφαρμογής. Η εφαρμογή ενδέχεται να αποτελείται από πολλά υποπακέτα (έχοντας ένα και μοναδικό κύριο πακέτο) κάτι που εξαρτάται από την πολυπλοκότητα, το μέγεθος και την δομή της εφαρμογής.

Αξιοσημείωτο είναι να αναφερθεί το αρχείο AndroidManifest.xml στο οποίο είναι καταχωρημένες οι κύριες πληροφορίες της εφαρμογής όπως

- Το όνομα πακέτου και το όνομα της εφαρμογής
- Η έκδοση των API Android που χρησιμοποιείται Καθώς και η version της εφαρμογής
- Τα activities που χρησιμοποιεί η εφαρμογή

Φάκελος src

Στον φάκελο αυτό περιέχονται όλα τα πηγαία αρχεία (java αρχεία)που έχει γράψει ο προγραμματιστής, οργανωμένα σε package

Φάκελος res

Περιέχονται όλα τα αρχεία εικόνας, ήχου, κειμένων,xml,layouts κτλ και τα οποία χρησιμοποιούνται από τα Activities. Τα παραπάνω αρχεία είναι ομαδοποιημένα σε φακέλους ανάλογα με το είδος τους. Ενδεικτικά αναφέρονται οι φάκελοι
drawable περιέχει εικόνες της εφαρμογής διαφόρων format όπως png,jpg,gif
layout περιέχονται τα layout της εφαρμογής. Πρόκειται για αρχεία xml που περιέχουν πληροφορίες για τα GUI (Graphical User Interface) συστατικά του κάθε layout

values περιέχει όλα τα αλφαριθμητικά και default τιμές που χρησιμοποιούνται από τα Activities της εφαρμογής
Εκτός από τους παραπάνω φακέλους υπάρχουν και άλλοι που ενδέχεται να μην χρησιμοποιούνται από τις εφαρμογές ή είναι δευτερεύουσας σημασίας.

1.5 Δομικά μέρη εφαρμογής

Τα Δομικά μέρη μια Android εφαρμογής είναι τα εξής:

Activities:

Το κυριότερο δομικό συστατικό της εφαρμογής είναι τα Activities. Αποτελείται από το αλληλεπιδραστικό τμήμα του προγράμματος με τον χρήστη δηλαδή το

Graphical User Interface καθώς και την αλλαγή και εμφάνιση δεδομένων από και προς τον χρήστη. Κάθε οθόνη συσχετίζεται και με ένα Activity. Εκτός από συγκεκριμένες περιπτώσεις όπου τμήματα οθόνης συνθέτονται δυναμικά μέσω κώδικα (δυναμική δημιουργία user interface). Τα Activities συνεργάζονται μεταξύ τους ώστε να συνθέσουν το αποτέλεσμα της εφαρμογής για την οποία κατασκευάστηκαν από τον προγραμματιστή.

Intents:

Τα Activities επικοινωνούν και ενεργοποιούνται και ανταλλάσουν δεδομένα δια μέσω των Intents. Ένα Intents Μπορεί να σταματήσει ένα Activity και να εκκινήσει ένα άλλο μεταβιβάζοντας δεδομένα που προέρχονται από το πρώτο. Συνήθως ένα Intent καλείται μέσω ενός Activity event προκειμένου να διακοπεί η κανονική λειτουργία του Activity στο οποίο κλήθηκε αυτό το event (π.χ. πάτημα του κουμπιού «επεξεργασία») και να ενεργοποιηθεί το Activity «επεξεργασία» μεταβιβάζοντας σε αυτό την πληροφορία για το τι πρέπει να επεξεργαστεί (π.χ. επεξεργασία του αυτοκινήτου με κωδικό 13)

Services:

Είναι λειτουργίες οι οποίες μπορούν και εκτελούνται στο παρασκήνιο δηλαδή μπορούν να εκτελούνται και όταν δεν είναι ενεργές στην οθόνη της κινητής συσκευής. Παράδειγμα μιας τέτοιας λειτουργίας είναι η αναπαραγωγή μουσικής ενώ ο χρήστης μπορεί πλοηγείται ταυτόχρονα στο internet.

Content Provider:

Όπως προαναφέρθηκε παραπάνω η ανταλλαγή πληροφορίας μπορεί να πραγματοποιηθεί μέσω ενός Intent. Ο Content Provider κάνει κάτι παραπλήσιο, διαχειρίζεται δηλαδή δεδομένα που έχει ορίσει by default ο προγραμματιστής της εφαρμογής.

Broadcast Receivers:

Είναι μια χρήσιμη διαδικασία ενημέρωσης ανάμεσα σε εφαρμογές ή διαδικασίες. Για παράδειγμα ένα service μπορεί να ενημερώσει ένα Activity για την ολοκλήρωση ή την αποτυχία της λειτουργία του μέσω ενός broadcast receiver. Το ίδιο μπορεί να γίνει και ανάμεσα σε μια εφαρμογή και σε μία άλλη. Παρόλο που μοιάζει με ένα Intent, ο ρόλος του είναι πιο ξεκάθαρος και έχει να κάνει με συμβάντα και ενημερώσεις συμβάντων από και προς άλλες λειτουργίες. Δεν έχουν κάποιο γραφικό περιβάλλον εντούτοις μπορούν να προβάλουν ενημερωτικά μηνύματα στον χρήστη δια μέσου της μπάρας ειδοποιήσεων.

1.6 Ανάπτυξη λογισμικού Android

Η υλοποίηση λογισμικού σε android προϋποθέτει εργαλεία ανάπτυξης και σχεδιασμού κατάλληλα για τον σκοπό αυτό καθώς και γνώσεις προγραμματισμού σε java. Το πρώτο βήμα είναι να γίνει εγκατάσταση των λογισμικών προγραμμάτων όπως IDE (όπως το Eclipse), Java, Android SDK (Software Development Kit) καθώς και τα εργαλεία ανάπτυξης Android – ADT (Android Development Tools)

Έπειτα θα πρέπει να δημιουργηθούν εικονικές συσκευές (AVD) για να μπορούν να δοκιμαστεί η εφαρμογή κατά την φάση ανάπτυξης της. Εναλλακτικά μπορεί να γίνει χρήση κανονικής συσκευής συνδέοντας την με τον υπολογιστή αλλά σε αυτή την περίπτωση θα πρέπει να διαθέτουμε πολλές συσκευές διαφορετικών οθονών και μοντέλων για να διαπιστώσει ότι η εφαρμογή θα είναι πλήρως συμβατή.

Το επόμενο βήμα αφού βέβαια προηγηθεί η ανάλυση προδιαγραφών της εφαρμογής είναι γραφή του πηγαίου κώδικα. Θα πρέπει να ληφθεί υπόψη ότι εκτός της ανάπτυξη του πηγαίου κώδικα θα πρέπει να μεριμνήσουμε και για το υλικό της εφαρμογής που θα χρησιμοποιηθεί όπως γραφικά, εικόνες, εικονίδια, πιθανόν οπτικοακουστικό υλικό κτλ. Στην ανάπτυξη της εφαρμογής θα πρέπει να συμπεριληφθεί και ο σχεδιασμός των layout στα οποία θα προστεθεί και γραφικό υλικό

Κατά την φάση ανάπτυξης των εφαρμογών όπως οφείλουμε να κάνουμε σε κάθε είδους προγραμματισμού θα πρέπει ο κώδικας να είναι οργανωμένος δομημένος και ευανάγνωστος κατά τέτοιο τρόπο ώστε η εφαρμογή να μπορεί να συντηρηθεί και επεκταθεί ευκολότερα. Επίσης όταν ο κώδικας πληροί τις παραπάνω προδιαγραφές μπορεί ευκολότερα να ανιχνεύουν λάθη ειδικά στην φάση ανάπτυξης κάνοντας χρήση του απόσφαλματοποιητή. (Debuger)

1.6.1 Android Debugging

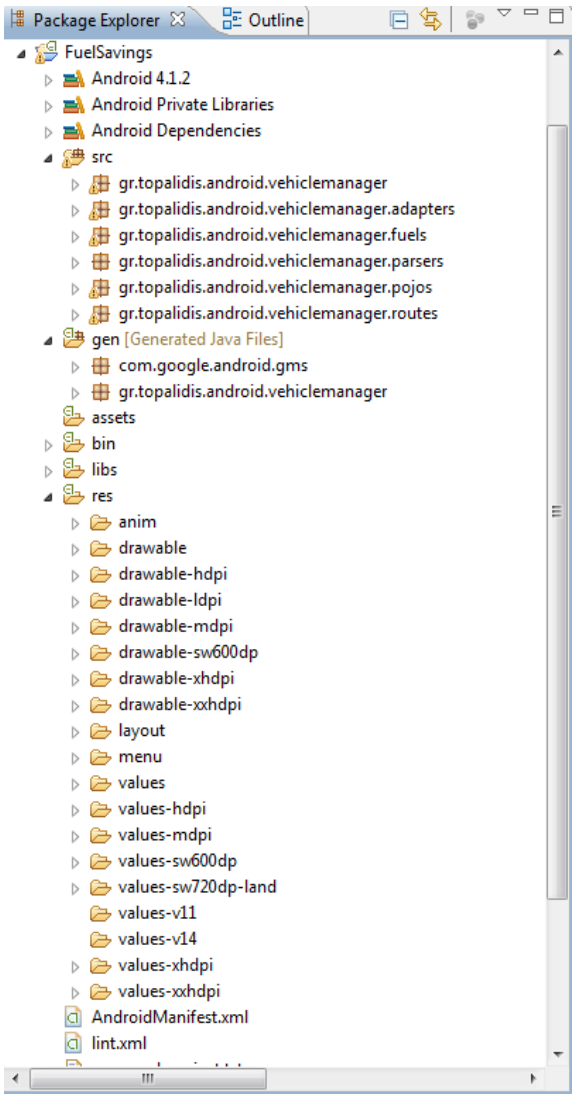
Όπως σε όλους τους τύπους προγραμματισμού έτσι και σε Android προγραμματισμό το debugging είναι μία χρήσιμη λειτουργία ώστε να ολοκληρωθεί η ανάπτυξη του κώδικα. Ακόμα όμως και μετά το τέλος της ανάπτυξης μιας Android εφαρμογής το debugging αποδεικνύεται χρήσιμο αφού πρέπει μετά το πέρας της γραφής του κώδικα να δοκιμαστούν αναλυτικά όλες οι περιπτώσεις χρήσης της εφαρμογής προτού διανεμηθεί στην αγορά.
Η πρώτη φράση της αποσφαλματοποίησης προϋποθέτει ότι ο κώδικας δεν περιέχει συνεκτικά λάθη. Έπειτα πραγματοποιείται πλοήγηση της εφαρμογής στο στάδιο που έχει υλοποιηθεί ώστε να εντοπιστούν αισθητικά λάθη, λειτουργικότητας ή και λογικά σφάλματα. Η αποσφαλματοποίηση πραγματοποιείται κυρίως μέσω του εργαλείου logcat του Eclipse IDE το οποίο μας ενημερώνει για το σημείο στο οποίο προέκυψε το σφάλμα καθώς και τα τμήματα του κώδικα από τα οποία πραγματοποιήθηκε η κλήση της προβληματικής εντολής (stack trace). Όπως αναφέρθηκε και παραπάνω το debugging πρέπει να πραγματοποιείται επαναληπτικά και κατά την διάρκεια της υλοποίησης αλλά και μετά το πέρας της ολοκλήρωσης της εφαρμογής, αφού μια αλλαγή του κώδικα ακόμα και τελευταία να είναι μπορεί να προκαλέσει σφάλμα σε κάποια άλλη περίπτωση χρήσης της εφαρμογής. Όπως γίνεται αντιληπτό το Debugging είναι μια απαραίτητη διαδικασία αλλά και αρκετά χρονοβόρα.

1.6.2 Δημοσίευση Android εφαρμογής

Μετά το τελική φάση ανάπτυξης και αποσφαλματοποίησης η εφαρμογή είναι σχεδόν έτοιμη να δημοσιευτεί. Αφού διαπιστωθεί ότι η εφαρμογή τρέχει κανονικά σε όλες τις περιπτώσεις πρέπει να ρυθμιστεί κατάλληλα σε production mode, να αφαιρεθούν (η απενεργοποιηθούν) κώδικας που χρησιμοποιήθηκε για δοκιμές και να παραμετροποιηθούν τα settings κατάλληλα (ενδέχεται να χρησιμοποιούνται διαφορετικές ρυθμίσεις σε debug και σε production mode ώστε να γίνεται πιο έυκολη η ανάπτυξη των εφαρμογών). Επίσης η μεταγλώττιση θα πρέπει να γίνει σε κανονική λειτουργία και όχι σε debug mode.
Το τελικό βήμα είναι η διανομή της εφαρμογής στο google play, στην amazon ή σε κάποιο άλλο marketplace

1.7 Δημιουργία Εφαρμογής FuelSavings

Αφού πρώτα έχουν καθοριστεί οι προδιαγραφές και οι περιπτώσεις χρήσης της εφαρμογής μπορεί να ξεκινήσει η διαδικασία υλοποίησης της. Χρησιμοποιώντας όλα τα εργαλεία που αναφέρθηκαν παραπάνω, μέσα από το περιβάλλον του Eclipse δημιουργούμε ένα νέο Android project συμπληρώνοντας τα απαραίτητα πεδία. Για λόγους επίδειξης της εφαρμογής, του κώδικα και των διαδικασιών παρουσιάζονται παρακάτω επεξηγήσεις της τελικής έκδοσης του project.



Δομή φακέλων του project

Το project android που αναπτύχθηκε χρησιμοποιεί το SKD 4.1.2. Το project επίσης περιλαμβάνει φακέλους πηγαίου κώδικα, αρχείων layout, xml, γραφικών (φάκελος res) κτλ. Όπως και η κάθε δομή που έχει ένα project android. Όπως φαίνεται στην διπλανή εικόνα ο φάκελος res περιέχει υποφάκελους drawable - hdpi - ldpi – mdpi. Ο καθένας από αυτούς περιέχει τις εικόνες της εφαρμογές σε διάφορες αναλύσεις οθονών.

Επίσης ο φάκελος res περιλαμβάνει τον υποφάκελο values που περιέχει xml αρχεία με τις default τιμές της εφαρμογής (κείμενα, ρυθμίσεις χρωμάτων κτλ)
Ένα άλλος φάκελος ο layout περιέχει τα GUI της κάθε οθόνης της εφαρμογής.

Η βασική διάρθρωση του κώδικα (φάκελος src) χωρίστηκε σε κατάλληλα package ώστε να είναι ευκολότερη η ανάπτυξη του project καθώς και να είναι ευκολότερη η χρήση και συντήρηση των λειτουργιών που αναπτύχθηκαν

ΚΕΦΑΛΑΙΟ 2

2.1 Χρησιμότητα εφαρμογής – Προδιαγραφές

Η εφαρμογή που αναπτύχθηκε στόχο έχει την εξοικονόμηση χρημάτων απο την χρήση των οχημάτων, τόσο για ιδιώτες όσο και επαγγελματίες. Ως οχήματα θεωρούνται τα αυτοκίνητα, φορτηγά, μοτοσυκλέτες, τρίκυκλα και οτιδήποτε χρησιμοποιεί καύσιμο, έχει έξοδα συντήρησης, ασφάλιση, έκτακτα έξοδα (π.χ. ατυχήματα) κτλ.

Παρόλο που η εφαρμογή έχει υλοποιηθεί για αυτοκίνητα εντούτοις μπορεί να χρησιμοποιηθεί και για άλλο τύπου οχήματος που συμπεριλαμβάνει τα παραπάνω έξοδα. Μέσω της εφαφμογής που υλοποιήθηκε ο ιδιοκτήτης του οχήματος μπορεί να καταγράφει τον ανεφοδιασμό καυσίμων την ώρα που αυτή πραγματοποιείται, δηλώνοντας την ποσότητα του καυσίμου και την τιμή του.

Επίσης ο χρήστης μπορεί να δηλώσει τον τύπο εξόδου όπως service, ασφάλεια, κτλ και το χρηματικό ποσό που έδωσε για την ενέργεια αυτή. Με την διαδικασία αυτή ο χρήστης μπορεί να έχει πλήρη καταγραφή των κινήσεων εξόδων του οχήματος του, στατιστικά καθώς και συγκρίσεις κόστους άλλων τύπου καυσίμων εκτώς της βενζίνης όπως πετρέλαιο, υγραεριο και φυσικό αέριο. Η χρησιμοτητα των παραπάνω έγκειται στο γεγονός οτι ο χρήστης μπορεί να μειώσει τα έξοδα του οχήματος του βλέποντας το κέρδος που θα είχε χρησιμοποιόντας άλλου τύπου καυσίμου.

Τα στατιστικά επίσης μπορούν να χρησιμοποιηθούν απο τον χρήστη ώστε να προβλέψει τους μήνες εκείνους που η κατανάλωση θα είναι αυξημένη σύμφωνα με δεδομένα προηγούμενων ετών. Οι παραπάνω πληροφορίες που προκύπτουν απο την καταγραφή των κινήσεων εξόδων, είναι χρήσιμη και για τους επαγγελματίες και ιδίως εταιρείες που έχουν παραπάνω του ενώς οχήματος. Ετσι η εφαρμογή υλοποιήθηκε έτσι ωστε να μπορουν να καταχωρηθούν παραπάνω του ενός οχήματος στο ίδιο κινητό. Έτσι ο χρήστης μπορεί να καταχωρεί τα παραπάνω έξοδα επιλέγοντας το όχημα που έχει ήδη καταχωρήσει στην εφαρμογή. Η χρησιμότητα σε αυτή την περίπτωση είναι οτι προκύπτουν συγκριτικά στατιστικά για δύο η παραπάνω οχήματα παρέχοντας έτσι στον επαγγελματία ή στην εταιρεία την δυνατότητα μείωσης των εξόδων των οχημάτων της.

Η εφαρμογή επίσης παρέχει την δυνατότητα υπενθύμισης πάγιων εξόδων όπως είναι η ασφάλειες και τα service των οχημάτων.

2.2 Περιπτώσεις χρήσης εφαρμογής

Μετα τον ορισμό των προδιαγραφών της εφαρμογής ακολουθούν οι περιπτώσεις χρήσης της εφαρμογής, δηλαδή τις ενέργειες του χρήστη για να πραγματοποιήσει τις καταχωρήσεις του οχήματος του. Χρειάστηκαν αρκετές δοκιμές για να πραγματοποιηθεί η διαδικασία αυτή και να καθοριστούν με την σωστή σειρά τα βήματα αυτά ωστε να είναι η εφαρμογή εύχρηστη και λειτουργική. Κάποια απο αυτά πραγματοποιούνται αυτόματα (παράδειγμα αρχικός συγχρονισμός βάσης

δεδομένων, ενημερώσεις κτλ) ενώ άλλα πραγματοποιούνται απο τον χρήστη. Παρακάτω περιγράφονται αυτές οι περιπτώσεις.

- Εγκατάσταση Εφαρμογής

Είναι το μόνο βήμα που πραγματοποιείται μια φορά όπως άλλοστε και όλες οι εφαρμογές. Ο χρήστης καταβάζει την εφαρμογή απο το appstore της google ή απο κάποιον άλλον server και την εγκαθιστά. Μετά απο αυτό το βήμα ο χρήστης μπορεί να εισάγει δεδομένα στην εφαρμογή. Ακόμα και μετά το κλείσιμο και άνοιγμα της εφαρμογής ή κάποιου απροσδόκητου κλεισίματος της εφαρμογής (crash / force close) τα δεδομένα έχουν διατηρηθεί σε τοπική βάση ώστε ο χρήστης να συνεχίσει απο εκεί που είχε σταματήσει.

- Εισαγωγή νέου Οχήματος

Ο χρήστης δημιουργεί ένα νέο όχημα στην βάση της εφαρμογής, μπορεί να ορίσει μάρκα, μοντέλο και τον τύπο του καυσίμου του οχήματος. Αυτό το βήμα μπορεί να επαναληφθεί και για την εισαγωγή και άλλων οχημάτων στην ίδια εφαρμογή. Κατα την πρώτη εισαγωγή η εφαρμογή θα επιχειρήσει να συγχρονίσει την βάση της με όλα τα τρέχον μοντέλα και μάρκες αυτοκινήτου που υπάρχουν σε ένα συγκεκριμένο WEB server. Έπειτα απο αυτή την ενημέρωση ο χρήστης επιλέγει τα παραπάνω χαρακτηριστικά του οχήματος του.

- Εισαγωγή καυσίμου

Ο χρήστης αφού πρώτα έχει επιλέξει ένα απο τα οχήματα που έχει καταχωρήσει πραγματοποιεί εισαγωγή κίνησης καυσίμου. Επιλέγει ημερομηνία κίνησης, ποσότητα, αξία και τύπου καυσίμου. Οι ενεργοί τύποι καυσίμου είναι αυτοί που έχει ορίσει στην εισαγωγή του συγκεκριμένου οχήματος. Το βήμα αυτό χρησιμοποιείται επαναληπτικά και πιο συχνά απο όλα τα άλλα αφου και στη πραγματικότητα ο ανεφοδιασμός καυσίμων γίνεται σχεδόν κάθε 2-3 ημέρες ανάλογα βέβαια τις ανάγκες του καθενός.

- Εισαγωγή Service

Στο βήμα αυτό ο χρήστης εισάγει τα service που πραγματοποιεί για το όχημα του. Οπως και στην ενέργεια καταχώρησης καυσίμου έτσι και σε αυτό το βήμα ο χρήστης ορίζει το κόστος και την ημερομηνία. Εκτός απο την εισαγωγή ώς service, ο χρήστης έχει την δυνατότητα να ορίσει άλλου τύπου υπηρεσίας (όπως μεταφορά κατόπιν ατυχήματος) και να περιγράψει το είδος αυτού το γεγονότος. Το βήμα αυτό δεν χρησιμοποιείται συχνά λόγω της φύσης των γεγονότων αυτών.

- Εμφάνιση κινήσεων - γεγονότων

  Στην πραγματικότητα αυτό το βήμα πραγματοποιείται αυτόματα απο την εφαρμογή κατόπιν εκχώρησης ενός καυσίμου ή service ή λίγο πρίν απο αυτή την εκχώρηση. Έτσι ο χρήστης έχει μια γενική εποπτία και οικονομικά στατιστικά στοιχεία των οχημάτων του.

2.3 Ανάλυση-Σχεδιασμός

Βασισμένοι στις προδιαγραφές τις εφαρμογής και στις περιπτώσεις χρήσης καθορίστικαν οι πίνακες της βάσης δεδομένων καθώς και τα πεδία τους. Έπειτα απο την σχεδίαση της βάσης πραγματοποιήθηκε και η υλοποίηση της εφαρμογής.

Κατόπιν ανάλυσης οι πίνακες της βάσης δεδομένων που σχεδιάστηκαν και υλοποιήθηκαν είναι οι εξής:

- **CarTable**

Πίνακας στον οποίο καταχωρούνται τα οχήματα του χρήστη. Στον πίνακα αυτό περιέχονται προσδιοριστικό του μοντέλου αυτοκινήτου.

- **ManufacturerTable**

  Περιέχει όλους τους κατασκευαστές (opel,ford κτλ) αυτοκινήτων, χρήσιμος πίνακας για τον ModelTable

- **ModelTable**

  Πρόκειται για όλα τα μοντέλα των αυτοκινήτων και τα οποία περιέχουν προσδιοριστικό του κατασκευαστή τους (πίνακας ManufacturerTable) Ο πίνακας αυτός χρησιμοποιείται για την συσχέτιση του αυτοκινήτου του χρήστη.

- **ServiceTable**

  Σε αυτόν τον πίνακα αποθηκεύονται τα service και τα γεγονότα των αυτοκινήτων του χρήστη.

- **FuelTable**

  Περιέχει τις κινήσεις καυσίμων των αυτοκινήτων. Αυτός ο πίνακας σε συνδυασμό με τον ServiceTable αποτελούν τα οικονομικά στατιστικά της εφαρμογής.

  Στον σχεδιασμό της βάσης δεδομένων προστέθηκαν και άλλοι δύο πίνακες. Παρόλο που δεν υλοποιήθηκαν λειτουργίες για αυτούς, συμπεριλήφθηκαν για μελλοντική έκδοση της εφαρμογής

- **RouteTable**

  Προκαθορισμένες διαδρομές που ορίζει ο χρήστης.

  (παράδειγμα ΤΕΙ Θεσσαλονίκης – Καλαμαριά)

- **RouteRecordsTable**

  Διαδρομές που πραγματοποιήθηκαν απο τον χρήστη και καύσιμα και χρόνοι που χρειάστηκαν. Ο πίνακας αυτός θα μπορούσε να χρησιμοποιηθεί για την απόδοση χρόνου/κόσους καυσίμων για συγκεκριμένες διαδρομές

ΚΕΦΑΛΑΙΟ 3

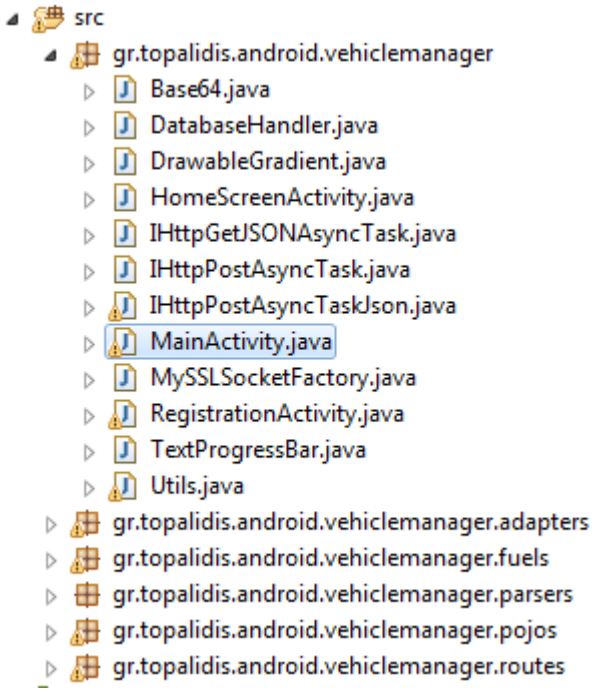
3.1 Επεξηγήση - Ανάλυση προγράμματος

Όπως κάθε java εφαρμογή έχει την main μέθοδο από την οποία θα ξεκινήσει η εφαρμογή έτσι και σε μία android εφαρμογή υπάρχει μια main Activity η οποία είναι υπεύθυνη για το τι θα εμφανιστεί ως πρώτη οθόνη κατά την εκκίνηση της εφαρμογής

Στην περίπτωση του project που αναπτύχθηκε το κύριο Activity είναι το η κλάση MainActivity (αρχείο MainActivity.java) που βρίσκεται στο package

gr.topalidis.android.vehiclemanager

Η κλάση αυτή κληρονομεί την ListActivity ώστε να μπορεί ο χρήστης να διαχειριστεί παραπάνω του ενός αυτοκινήτου με την μορφή πολλών item μιας λίστας



3.2 Αρχείο MainActivity.java Κληρονόμηση της ListActivity

```
public class MainActivity extends ListActivity implements LocationListener
```

Η κλάση MainActivity περιλαμβάνει τα περισσότερα αντικείμενα, υλοποιήσεις άλλων κλάσεων μιας και είναι υπεύθυνη για την κεντρική οθόνη της εφαρμογής.

Για παράδειγμα

```
private static ListView lv;
private static List<CarClass> carList;
```

Πρόκειται για ιδιότητες της κλάσης που τοποθετήθηκαν προκειμένου τα αυτοκίνητα να μπορούν να επεξεργαστούν, προστεθούν ή να αφαιρεθούν από την βασική activity. Η carList δηλώθηκε τύπου list ώστε να μπορούμε δυναμικά να προσθέτουμε και να αφαιρούμε στοιχεία και ο τύπος των στοιχείων δηλώθηκε ως CarClass ώστε να μπορούμε να περιέχονται δεδομένα και λειτουργίες της κλάσης κατάλληλα για ένα στοιχείο - αμάξι

Όπως σε όλα τα activities έτσι και εδώ η αρχικοποίηση γίνεται στην συνάρτηση onCreate. Λόγω του μεγέθους των μεθόδων αναφέρουμε ενδεικτικά συγκεκριμένα τμήματα του κώδικα μαζί με την επεξήγηση τους

Υπερκάλυψη της μεθόδου onCreate ώστε να αρχικοποιήσουμε εκτός από τα default και τα αντικείμενα της κλάσης.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
```

Κλήση της default onCreate (της υπερτάξης που κληρονομεί η MainActivity)

```
super.onCreate(savedInstanceState);
```

Καθορισμός του layout που θα χρησιμοποιηθεί για το συγκεκριμένο Activity. Το αρχείο που περιέχει της πληροφορίες για το GUI της οθόνης είναι στον φάκελο res και υποφάκελο layouts (αρχείο activity_main.xml). Η χρήση του όμως μέσα στην συνάρτηση setContentView γίνεται μέσω της ειδικής για αυτό τον σκοπό κλάσης R

```
setContentView(R.layout.activity_main);
```

```
Init Vars
```

Αρχικοποίηση ενός sharedPreferences με συγκεκριμένο όνομα. Δηλώνοντας ένα τέτοιου είδους αντικείμενο μπορούμε να έχουμε πρόσβαση σε δεδομένα από οποιοδήποτε Activity. Επιπλέον τα δεδομένα που υπάρχουν σε ένα sharedPreferences μπορούν να αποθηκευτούν και για επόμενες εκκινήσεις της εφαρμογής.

```
sp = getSharedPreferences(SP_NAME, 0);
```

Ανάκτηση ενός ImageView ώστε να μπορούμε να έχουμε πρόσβαση στις μεθόδους και στις ιδιότητες του. Στο αντικείμενο αυτό θα προσαρτηθεί κίνηση ενός αυτοκινήτου.

```
iv_car = (ImageView) findViewById(R.id.iv_car);
```

Φόρτωση διαφόρων οπτικών του αυτοκινήτου ώστε να παραχθεί το τελικό animation μέσω του iv_car

```
hyperspaceJumpAnimation = AnimationUtils.loadAnimation(this,
R.anim.hyperspace_jump);
```

```
hyperspaceJumpAnimationRev = AnimationUtils.loadAnimation(this,
R.anim.hyperspace_jump_rev);

slideInLeft = AnimationUtils.loadAnimation(this, R.anim.slide_in_left);
slideInRight = AnimationUtils.loadAnimation(this, R.anim.slide_in_right);
slideOutLeft = AnimationUtils.loadAnimation(this, R.anim.slide_out_left);

mainActionsAnimation = AnimationUtils.loadAnimation(this, R.anim.slide_in_up);
```

Στον παρακάτω κώδικα δηλώνονται διάφοροι listeners για τα animation. Τέτοια είναι το onAnimationStart, onAnimationEnd, onAnimationRepeat κτλ στα οποία συμβάντα λαμβάνουν δράση κάποιες ενέργειες όπως εμφάνιση ή απόκρυψη κάποιων κουμπιών. Εδώ παρουσιάζονται μόνο μερικά λόγου μεγέθους του κώδικα.

```
mainActionsAnimation.setAnimationListener(new AnimationListener() {

	@Override
	public void onAnimationStart(Animation animation) {
		ll_bottom_buttons.startAnimation(slideOutBottom);
		ll_main_actions.setVisibility(View.VISIBLE);
	}

	@Override
	public void onAnimationRepeat(Animation animation) {
	}

	@Override
	public void onAnimationEnd(Animation animation) {
		ll_bottom_buttons.setVisibility(View.GONE);
	}
});
```

Αρχικοποιήσεις άλλων αναφορών όπως LinearLayout και Drawable ώστε να τις χρησιμοποιήσουμε σε διάφορα άλλα listener

```
ll_main_header = (LinearLayout) findViewById(R.id.ll_main_header);
ll_bottom_buttons = (LinearLayout) findViewById(R.id.ll_bottom_buttons);
ll_main_actions = (LinearLayout) findViewById(R.id.ll_main_actions);
stubIcon = getResources().getDrawable(stubIconID);
```

Παρακάτω περιγράφεται ο listener του πλήκτρου edit car. Όπως φαίνεται για να εκτελεστεί η μέθοδος showEditDialog θα πρέπει η μεταβλητή selCar να είναι διάφορη του null δηλαδή να είναι επιλεγμένο κάποιο αμάξι. Παρακάτω περογράφεται η αρχικοποίηση της μεταβλητής selCar

```
btn_edit.setOnClickListener(new OnClickListener() {
	@Override
	public void onClick(View v) {
		if (selCar != null) {
		showEditCarDialog();
		}
```

```
        }
});
```

Η selCar ουσιαστικά είναι το επιλεγμένο στοιχείο από μια listView και στην συγκεκριμένη περίπτωση είναι τύπου CarClass. Αρχικοποιούμε το lv με μια κενή listView, έπειτα ορίζουμε listener για το γεγονός click ενός οποιουδήποτε στοιχείου της λίστας, ορίζουμε το selCar με το επιλεγμένο στοιχείο και τέλος ενεργοποιούμε τα κουμπιά επικύρωση επιλογής (για περεταίρω εισαγωγή καυσίμων κτλ), επεξεργασία οχήματος και το κουμπί διαγραφής οχήματος.

```
lv = getListView();
lv.setOnItemClickListener(new OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,...) {
        selCar = (CarClass) lv.getItemAtPosition(arg2);
        btn_go.setEnabled(true);
        btn_edit.setEnabled(true);
        btn_delete.setEnabled(true);
        }
});
```

Μετά τον ορισμό του παραπάνω listener αρχικοποιούμε την λίστα των αυτοκινήτων χρησιμοποιώντας την μέθοδο initializeLv

```
initializeLv(MainActivity.this);
```

Η οποία με την σειρά της αντλεί τα δεδομένα από μία τοπική βάση δεδομένων, προσθέτει ένα κενό όχημα στην αρχή της λίστας (ώστε ο χρήστης να έχει πάντα την δυνατότητα να επεξεργαστεί το πρώτο στοιχείο της λίστας ως νέο όχημα και να το αποθηκεύει στην βάση αργότερα. Με αυτό τον τρόπο μπορεί να αποθηκεύει όσα αυτοκίνητα επιθυμεί. Η carList τύπου List με δυνατότητα αποθήκευσης CarClass, χρησιμοποιείται για να δημιουργηθεί adapter ο οποίος εισάγεται τελικά στην ListView, έτοιμο για την εμφάνιση των αυτοκινήτων στην οθόνη του χρήστη.

Ένα εύλογο ερώτημα είναι γιατί να μην μπορεί να περαστεί η List με τα αυτοκίνητα απευθείας στο listView. Η απάντηση είναι ότι ο adapter περιέχει και άλλες πληροφορίες όπως το είδος της πληροφορίας που θα απεικονιστεί για κάθε στοιχείο της λίστας (αυτοκίνητο) κάτι για το οποίο θα συζητήσουμε παρακάτω

```
private static void initializeLv(Context ctx) {
        addCar = new CarClass(null, null, null, null, null, null, dAdd, -1);
        carList = getCarsFromDB(ctx);
        carList.add(0, addCar);
        listAdapter = new CarAdapter(ctx, carList);
        lv.setAdapter(listAdapter);
}
```

Επιστρέφοντας στο σημείο κλήσης της initializeLv παρατηρούμε ότι δεν είναι απαραίτητη η φόρτωση των δεδομένων της Car – ListView πριν την δήλωση των listeners της συγκεκριμένης ListView μιας και ο κώδικας των listener φορτώνεται

στην μνήμη αλλά δεν μπορεί να εκτελεστεί πρώτου τελειώσει η onCreate και προτού ο χρήστης κάνει κλικ σε κάποιο στοιχείο της λίστας.

Κατόπιν της επιλογής ενός οχήματος η selCar δεν είναι κενή οπότε κάνοντας κλικ στο κουμπί επεξεργασία πραγματοποιείται η κλήση της showEditCarDialog που με την σειρά της εκτελεί τον παρακάτω κώδικα

```
protected void showEditCarDialog() {
```

Δημιουργία διαλόγου αλληλεπίδρασης με τον χρήστη. Κατόπιν ορίζουμε τον τύπο και το περιεχόμενο του διαλόγου του οποίου τα συστατικά ορίζονται στο R.layout.dialog_edit_car δηλαδή στο αρχείο dialog_edit_car.xml του φακέλου res/layout

```
final Dialog dialog = new Dialog(MainActivity.this,
android.R.style.Theme_Translucent_NoTitleBar);
dialog.getWindow().setGravity(Gravity.CENTER);

dialog.getWindow().setLayout(LinearLayout.LayoutParams.MATCH_PAREN
T, LinearLayout.LayoutParams.MATCH_PARENT);
dialog.setContentView(R.layout.dialog_edit_car);
```

Έλεγχος των γραφικών συστατικών του gui με κώδικα. Η λήψη του κάθε χειριστηρίου γίνεται μέσω των προσδιοριστικού (π.χ. et_manufacturer) που έχουν οριστεί στα αρχεία xml στον φάκελο layouts

```
final EditText et_manufacturer = (EditText)
dialog.findViewById(R.id.et_manufacturer);

final EditText et_model = (EditText)
dialog.findViewById(R.id.et_model);

final EditText et_licencePlate = (EditText)
dialog.findViewById(R.id.et_car_licence_plate);

final EditText et_modelYear = (EditText)
dialog.findViewById(R.id.et_car_model_year);

final Button btn_insurance_date_from = (Button)
dialog.findViewById(R.id.btn_insurance_date_from);

final Button btn_insurance_date_to = (Button)
dialog.findViewById(R.id.btn_insurance_date_to);
```

Ορισμός τιμών και ετικετών για τα χειριστήρια

```
btn_insurance_date_from.setText(R.string.no_insurance_date_from);
btn_insurance_date_to.setText(R.string.no_insurance_date_to);
```

Δημιουργία listener για τα κουμπιά ημερομηνιών. Κατά το κλικ σε κάποιο από αυτά τα κουμπιά εμφανίζεται φόρμα διαλόγου για την επιλογή άλλης ημερομηνίας.

```
btn_insurance_date_from.setOnClickListener(new OnClickListener() {

	@Override
	public void onClick(View v) {
		showDateDialog(MainActivity.this,
		btn_insurance_date_from, false, true);
		}
});

btn_insurance_date_to.setOnClickListener(new OnClickListener() {

	@Override
	public void onClick(View v) {
		showDateDialog(MainActivity.this,
	btn_insurance_date_to, false, false);
	}
});
```

Ορισμός άλλων τιμών στην φόρμα διαλόγου

```
et_manufacturer.setText(selCar.getManufacturer());
et_model.setText(selCar.getModel());
et_licencePlate.setText(selCar.getLicencePlate());
et_modelYear.setText(selCar.getModelYear());
```

Διάφοροι έλεγχοι απόδοσης τιμών από τον χρήστη

```
if (selCar.getInsuranceDateFrom() == null ||
selCar.getInsuranceDateFrom().equals(""))
btn_insurance_date_from
.setText(R.string.no_insurance_date_from);
else
btn_insurance_date_from
.setText(selCar.getInsuranceDateFrom());

if (selCar.getInsuranceDateTo() == null ||
selCar.getInsuranceDateTo().equals(""))
btn_insurance_date_to
.setText(R.string.no_insurance_date_to);
else
btn_insurance_date_to.setText(selCar.getInsuranceDateTo());

Button btn_ok = (Button) dialog.findViewById(R.id.btn_ok);
Button btn_cancel = (Button) dialog.findViewById(R.id.btn_cancel);
```

Διάφοροι έλεγχοι απόδοσης τιμών από τον χρήστη

```
btn_ok.setOnClickListener(new OnClickListener() {
	@Override
	public void onClick(View v) {
		if (checkInputFields()) {
```

```
                        saveCar(et_manufacturer.getText().toString(),
                        et_model.getText().toString(),
                        et_licencePlate.getText().toString(),

                        et_modelYear.getText().toString());
                              dialog.dismiss();
                        }
                  }
```

Έλεγχος των τιμών των textbox. Αν κάποιο από από τα πεδία manufacturer, μοντέλο ή πινακίδα αυτοκινήτου είναι κενό δημιουργείται το αντίστοιχο μήνυμα και εμφανίζεται ως toast στον χρήστη

```
private boolean checkInputFields() {
      int errorCount = 0;
      String responseString = "Δεν έχετε εισάγει: ";
      if (et_manufacturer.getText().toString().equals("")) {
                  responseString += "Κατασκευαστή";
                  errorCount++;
      }
      if (et_model.getText().toString().equals("")) {
      if (errorCount == 0)
            responseString += "Μοντέλο";
      else
            responseString += ", Μοντέλο";
            errorCount++;
      }

      if (et_licencePlate.getText().toString().equals("")) {
            if (errorCount == 0)
                  responseString += "Πινακίδα";
            else
                  responseString += ", Πινακίδα";
                  errorCount++;
      }

      if (errorCount > 0)
            Utils.showToast((Activity) MainActivity.this, responseString);
                        return (errorCount == 0 ? true : false);
      }
```

Αποθήκευση των χαρακτηριστικών του αυτοκινήτου, σαν παράμετρους η μέθοδος αυτή δέχεται σε κείμενο τον κατασκευαστή, το μοντέλο και την πινακίδα του αυτοκινήτου. Έπειτα μεσω βοηθητικών συναρτήσεων της κλάσης DatabaseHandler αποθηκεύεται το αυτοκίνητο στην τοπική βάση ΄της εφαρμογής

```
private void saveCar(String manufacturer, String model, String licencePlate,
String modelYear) {

      DatabaseHandler dbh = new DatabaseHandler(MainActivity.this,
      DatabaseHandler.DB_VERSION);

      selCar.setManufacturer(manufacturer);
      selCar.setModel(model);
```

```
selCar.setLicencePlate(licencePlate);
selCar.setModelYear(modelYear);

dbh.updateCar(selCar);
dbh.close();
                    listAdapter.notifyDataSetChanged();
                }

        });

        btn_cancel.setOnClickListener(new OnClickListener() {
                @Override
                public void onClick(View v) {
                        dialog.dismiss();
                }
        });

        LinearLayout ll_dialog_content = (LinearLayout)
        dialog.findViewById(R.id.tl_dialog_car);

        dialog.show();
        ll_dialog_content.startAnimation(slideInLeftDelay);
}
```

Γραφική αναπαράσταση του dialog_edit.xml

Η παραπάνω γραφική οθόνη αλληλεπίδρασης με τον χρήστη παράγεται από xml κώδικα ο οποίος συνοψίζεται παρακάτω. Λόγω των γραμμών κώδικα έχουν αφαιρεθεί στο παρόν έγγραφο οι περισσότερες ιδιότητες των συστατικών του. Τα κύρια συστατικά του xml που παράγουν το αλληλεπιδραστικό γραφικό περιβάλλον αποτελούνται από TextView, EditText και Button στα οποία καθορίζονται τα χαρακτηριστικά τους. Κύριο χαρακτηριστικό του καθενός από αυτά είναι το android:id χρήσιμο για τον έλεγχο των χειριστηρίων από τον κώδικα java που εκτελείται στα activities.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"

    <RelativeLayout
        android:layout_width="match_parent"
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>
            android:layout_centerInParent="true"
    </RelativeLayout>
    <TableLayout
        android:id="@+id/tl_dialog_car"

        <TableRow
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            <EditText
                android:id="@+id/et_manufacturer"/>

            <EditText
                android:id="@+id/et_model"/>
        </TableRow>
        <TableRow>
            <EditText
                android:id="@+id/et_car_licence_plate"/>
            <EditText
                android:id="@+id/et_car_model_year"/>
        </TableRow>
        <TableRow
            android:layout_width="match_parent">
            <TextView/>
        </TableRow>
        <TableRow />
            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_span="2" >
                <Button
                    android:id="@+id/btn_insurance_date_from"/>
                <Button
                    android:id="@+id/btn_insurance_date_to"/>
            </LinearLayout>
        </TableRow>
        <TableRow>
            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:layout_span="2" >
                <include layout="@layout/merge_btns_ok_cancel" />
            </LinearLayout>
        </TableRow>
    </TableLayout>
</LinearLayout>
```

3.3 Αρχείο Utils.java

Γενικές συναρτήσεις και παράμετροι για σύνδεση με websservices και άλλες χρήσιμες λειτουργίες. Το αρχείο αυτό δεν αποτελεί από μόνο του αλληλεπίδραση με τον χρήστη ή λειτουργία της εφαρμογής αλλά χρησιμοποιείται από άλλες ενέργεις ως βιβλιοθήκες.

```
public class Utils {
```

Ιδιότητες και παράμετροι όπως url webservices ώστε να υπάρχει συγχρονισμός δεδομένων με το διαδίκτυο. Με αυτό τον τρόπο η εφαρμογή δεν περιλαμβάνει όλα τα δεδομένα κατά την εγκατάσταση της (ή στο downloading) αλλά κατά τη εκίνηση της την πρώτη φορά. Έτσι μπορούν να γίνουν ενημερώσεις μοντέλων, και κατασκευαστών κτλ κάθε φορά που ενημερώνεται η βάση του διαδικτύου (web application). Πλεονέκτημα αυτής της μεθόδου είναι ότι μπορούν να ενημερωθούν αυτόματα πλήθος εφαρμογών χωρίς να χρειάζεται να κατεβάσει ο χρήστης νέα έκδοση της εφαρμογής.

```
	private Context ctx;
	static final String tag = "Debug Utils";
	static final String WEB_URL_REGISTRATION =
	"http://www.allaboutmycar.eu/index.php?r=site/registration&jsontype=1";
	static final String WEB_URL_CONFIRM =
	"http://www.allaboutmycar.eu/index.php?jsontype=1&r=site/registration&con
	firm=";
	static final String WEB_URL_LOGIN =
	"http://www.allaboutmycar.eu/index.php?r=site/login&jsontype=1";
	static final String WEB_URL_GET_CITIES =
	"http://www.allaboutmycar.eu/index.php?r=services/getcities&time=";
	//give time=0 or ommit to get all

	static final String SERVICE_URL =
	"https://192.168.1.23:7775/FuelSavingsService/adds/";
	static final String SERVICE_URL_EXTERNAL =
	"http://46.198.134.106:9500/FuelSavingsService/adds/";
	static final String URL_BASE_CARQUERYAPI =
	"http://www.carqueryapi.com/api/0.3/?callback=?&cmd=";
	static final String URL_CARQUERYAPI_GET_MAKES =
	"getMakes";//&sold_in_us=0";
	static final String URL_CARQUERYAPI_GET_TRIMS =
	"getTrims";//&sold_in_us=0";
	static final String URL_CARQUERYAPI_GET_YEARS = "getYears";
	static final int MAX_TRIMS_CARQUERYAPI = 500;
	private static final int CONNECTION_TO_ms = 20000;
	private static final int SOCKET_TO_ms = 40000;
	static final int STRING_MAX_LENGTH = 20;
	public static DecimalFormat dateFormat = new DecimalFormat("00");

	public Utils(Context ctx) {
		this.ctx = ctx;
	}
```

Μέθοδοι εμφάνισης toast μηνυμάτων προς τον χρήστη, μπάρες πλήρωσης και ποσοστά ολοκλήρωσης συγχρονισμού.

```
public static void showToast(final Activity act, final String msg) {
		act.runOnUiThread(new Runnable() {

	public void run() {
		Toast toast = Toast.makeText(act, "", Toast.LENGTH_SHORT);
		toast.setGravity(Gravity.BOTTOM, 0, 0);
		LayoutInflater inflater = act.getLayoutInflater();
		LinearLayout ll = (LinearLayout)
		inflater.inflate(R.layout.custom_toast_layout,
		(ViewGroup) act.findViewById(R.id.toast_layout_root));
		TextView tv = (TextView) ll.getChildAt(1);
		tv.setText(msg);
		LinearLayout.LayoutParams llParams = new
		LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT,
		LinearLayout.LayoutParams.WRAP_CONTENT);
		llParams.setMargins(10, 0, 10, 0);
		ll.setLayoutParams(llParams);
		toast.setView(ll);
			toast.show();
			}
		});

	}

public static String getFormatedTimeFromMillis(long millis) {
	int sec = (int) (millis / 1000) % 60;
	int min = (int) ((millis / (1000 * 60)) % 60);
	int hr = (int) ((millis / (1000 * 60 * 60)) % 24);
	return String.format("%02d:%02d:%02d", hr, min, sec);
}

public static String getTodaysFormatedDate() {
	Calendar cal = Calendar.getInstance();
	return cal.get(Calendar.DAY_OF_MONTH) + "/" + (cal.get(Calendar.MONTH) +
	1) + "/" + cal.get(Calendar.YEAR);
}

public static String getTimeNowFormated() {
	Calendar cal = Calendar.getInstance();
	int hour = cal.get(GregorianCalendar.HOUR_OF_DAY);
	int min = cal.get(GregorianCalendar.MINUTE);
	return String.valueOf(dateFormat.format(hour)) + ":" +
	String.valueOf(dateFormat.format(min));
}
```

Μετατροπή Bitmap σε byte Array. Χρήσιμη μέθοδος κατά την αποστολή εικόνων του αυτοκινήτου σε ιστοσελίδα (webservice). Έτσι ο χρήστης έχει την δυνατότητα να μπορεί να έχει τα όλα τα δεδομένα και website

```
public static byte[] getByteArrayFromDrawable(Drawable d) {
	Bitmap bitmap = ((BitmapDrawable) d).getBitmap();
	ByteArrayOutputStream stream = new ByteArrayOutputStream();
```

```
        bitmap.compress(Bitmap.CompressFormat.JPEG, 100, stream);
        byte[] bitmapdata = stream.toByteArray();
        return bitmapdata;
}
```

Μετατροπή byte Array σε Bitmap. Χρήσιμο για την αντίστροφή προηγούμενη διαδικασία. Αυτή η μέθοδος χρησιμοποιείται για λήψη φωτογραφιών από το web

```
public static Bitmap getBitmapFromBytes(byte[] imageBytes) {
        if (imageBytes != null)
                return BitmapFactory.decodeByteArray(imageBytes, 0,
        imageBytes.length);
                else
        return null;
}
```

Έλεγχος για την διαθεσιμότητα διαδικτύου και των υπηρεσίων του webservice

```
public boolean isNetworkAvailable() {
        Activity act = (Activity) ctx;
        ConnectivityManager connectivityManager = (ConnectivityManager)
        act.getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo activeNetworkInfo =
        connectivityManager.getActiveNetworkInfo();
        return activeNetworkInfo != null;
}
```

Οι παρακάτω μέθοδοι έχουν γραφεί τόσο για αιτήσεις τύπου https όσο και http. Λόγω των διαφοροποιήσεων του κώδικα ανάμεσα σε http και https κατηγοροιοποιήθηκαν σε δύο κλάσεις την RestClient και την RestClientNoSSL ώστε σε περίπτωση αλλαγής προτωκόλου από την μεριά του server να γίνει ευκολότερη η προσαρμογή του κώδικα της εφαρμογής.

Έκδοση επικοινωνίας για πρωτόκολλο HTTPS

```
public final static class RestClient {
        private String URL;
        RestClient(String url) {
                this.URL = url;
}
```

Μετατροπή λίστας τιμών σε μορφή json, Αποστολή δεδομένων και λήψη απάντησης από webserver.

```
public String sendJson(ArrayList<NameValuePair> params) throws
KeyStoreException, NoSuchAlgorithmException, CertificateException,
IOException, KeyManagementException, UnrecoverableKeyException {
```

Ρυθμίσεις και επικεφαλίδες πρωτοκόλου https, cookies, πόρτας επικοινωνίας, timeout αναμονής από server κτλ.

```
        Log.i(tag, "Starting Rest client instance for URL: " + URL);
        KeyStore trustStore = KeyStore.getInstance(KeyStore.getDefaultType());
```

```
trustStore.load(null, null);
SSLSocketFactory sf = new MySSLSocketFactory(trustStore);
sf.setHostnameVerifier(SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER);

HttpParams con_params = new BasicHttpParams();
HttpProtocolParams.setVersion(con_params, HttpVersion.HTTP_1_1);
HttpProtocolParams.setContentCharset(con_params, HTTP.UTF_8);
SchemeRegistry registry = new SchemeRegistry();


registry.register(new Scheme("http",
PlainSocketFactory.getSocketFactory(), 80));
registry.register(new Scheme("https", sf, 443));
ClientConnectionManager ccm = new ThreadSafeClientConnManager(con_params,
registry);
HttpClient client = new DefaultHttpClient(ccm, con_params);
HttpConnectionParams.setConnectionTimeout(client.getParams(),CONNECTION_T
O_ms);
HttpConnectionParams.setSoTimeout(client.getParams(), SOCKET_TO_ms);
HttpResponse response;
String responseString = null;
JSONObject json = new JSONObject();
      try {
            HttpPost post = new HttpPost(URL);
```

Μετατροπή λίστας κλειδιών και τιμών σε τύπου json object και έπειτα προετοιμασία για αποστολή με την μέθοδο POST, Αποστολή δεδομένων και λήψης απάντησης από τον server

```
            for (NameValuePair p : params) {
                  json.put(p.getName(), p.getValue());
            }

            StringEntity se = new StringEntity(json.toString(),
            HTTP.UTF_8);
            post.setEntity(se);
            post.setHeader("Accept", "application/json");
            post.setHeader("Content-type", "application/json");
            response = client.execute(post);

            /* Checking response */
            if (response != null) {
                  InputStream is = response.getEntity().getContent();
                  BufferedReader bReader = new BufferedReader(new
                  InputStreamReader(is));
                  StringBuilder sb = new StringBuilder();
                  String line = null;

                  while ((line = bReader.readLine()) != null) {
                              sb.append(line);
                  }
                  responseString = sb.toString();
      //Log.i(tag, "Rest Client POST response: " + responseString);
```

Αν ο τύπος επιστροφής από τον server είναι html ή η τύπου json απάντηση περιέχει λάθη επέστρεψε λάθος

```
            if (responseString.startsWith("<!DOCTYPE HTML PUBLIC"))
                                    return "ERROR";
            }
            } catch (JSONException ex) {
                        Log.e(tag, "JSONException :" + ex.getMessage());
                        return "ERROR" + ex.getMessage();
            } catch (UnsupportedEncodingException ex) {
                        Log.e(tag, "UnsupportedEncodingException :" +
                        ex.getMessage());
                        return "ERROR" + ex.getMessage();
            } catch (ClientProtocolException ex) {
                        Log.e(tag, "ClientProtocolException :" +
                        ex.getMessage());
                        return "ERROR" + ex.getMessage();
            } catch (IOException ex) {
                        Log.e(tag, "IOException :" + ex.getMessage());
                        return "ERROR" + ex.getMessage();
            }
                  return responseString;
            }
```

Έκδοση επικοινωνίας για πρωτόκολλο HTTP

```
public final static class RestClientNoSSL {
      private String URL;
      RestClientNoSSL(String url) {
            this.URL = url;
      }

public String sendJson(ArrayList<NameValuePair> params) throws
KeyStoreException, NoSuchAlgorithmException, CertificateException,
                        IOException, KeyManagementException,
UnrecoverableKeyException {
      Log.i(tag, "Starting RestClientNoSSL instance for URL: " + URL);
```

Ρυθμίσεις και επικεφαλίδες πρωτοκόλου https, cookies, πόρτας επικοινωνίας, timeout αναμονής από server κτλ.

```
      HttpClient httpclient = new DefaultHttpClient();
      HttpPost httppost = new HttpPost(URL);
      String responseString = null;
      HttpResponse response = null;

      try {
            // Create a local instance of cookie store
            CookieStore cookieStore = new BasicCookieStore();
           // Create local HTTP context
            HttpContext localContext = new BasicHttpContext();
            // Bind custom cookie store to the local context
            localContext.setAttribute(ClientContext.COOKIE_STORE,
            cookieStore);
```

Μετατροπή λίστας κλειδιών και τιμών σε δεδομένα POST, Αποστολή δεδομένων και λήψης απάντησης από τον server

```
            if (params != null) {
```

```
httppost.setEntity(new UrlEncodedFormEntity(params));
response = httpclient.execute(httppost,localContext);
            } else { //No params so use httpget instead
      HttpGet httpGet = new HttpGet(URL);
      response = httpclient.execute(httpGet,localContext);
}
//Checking response
if (response != null) {
InputStream is = response.getEntity().getContent();
BufferedReader bReader = new BufferedReader(new
InputStreamReader(is));

StringBuilder sb = new StringBuilder();
String line = null;
while ((line = bReader.readLine()) != null) {
      sb.append(line);
}
responseString = sb.toString();
//Log.i(tag, "Rest Client POST response: " + responseString);
```

Έλεγχος ορθότητας λήψης απάντησης

```
if (responseString.startsWith("<!DOCTYPE HTML PUBLIC"))
      return "ERROR";
}
} catch (UnsupportedEncodingException ex) {
Log.e(tag, "UnsupportedEncodingException :" + ex.getMessage());
      return "ERROR" + ex.getMessage();
} catch (ClientProtocolException ex) {
Log.e(tag, "ClientProtocolException :" + ex.getMessage());
      return "ERROR" + ex.getMessage();
} catch (IOException ex) {
      Log.e(tag, "IOException :" + ex.getMessage());
      return "ERROR" + ex.getMessage();
}
      return responseString;
}
}
```

Μέθοδος εμφάνισης κωδικού λάθους στον χρήστη

```
public void showErrorDialog(int code) {
            AlertDialog.Builder alertDialogBuilder = new
            AlertDialog.Builder(ctx);
            alertDialogBuilder.setTitle("Σφάλμα");
            alertDialogBuilder.setMessage(getMessageForErrorCode(ctx,
            code)).setCancelable(false)
            .setPositiveButton("OK", new DialogInterface.OnClickListener() {

            public void onClick(DialogInterface alertDialog, int id) {
                  alertDialog.dismiss();
            }
            });
            AlertDialog alertDialog = alertDialogBuilder.create();
            alertDialog.show();
}
```

Μέθοδος εμφάνισης μηνύματος λάθους στον χρήστη

```
/** ----------- Error dialog displaying message ------------- **/
public void showErrorDialog(String msg) {
	AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(ctx);
		alertDialogBuilder.setTitle("Σφάλμα");
alertDialogBuilder.setMessage(msg).setCancelable(false).setPositiveButton("OK",
new DialogInterface.OnClickListener() {




public void onClick(DialogInterface alertDialog, int id) {
				alertDialog.dismiss();
	}
});
		AlertDialog alertDialog = alertDialogBuilder.create();
		alertDialog.show();
	}
```

Υπερφόρτωση συνάρτησης εφμάνισης μηνύματος λάθους με τίτλο

```
/** ----------- Dialog displaying message ------------- **/
public void showDialog(String title, String msg) {
	AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(ctx);
		alertDialogBuilder.setTitle(title);

	alertDialogBuilder.setMessage(msg).setCancelable(false).setPositiveButton
	("OK", new DialogInterface.OnClickListener() {
			public void onClick(DialogInterface alertDialog, int id) {
				alertDialog.dismiss();
			}
		});
		AlertDialog alertDialog = alertDialogBuilder.create();
		alertDialog.show();
	}

	/** ---------- Return a string corresponding to the error code given ----
	private String getMessageForErrorCode(Context ctx, int code) {
		String message = "";
		Resources res = ctx.getResources();
		switch (code) {
		case 1:
			message = res.getString(R.string.errorCode_1);
			break;
		case 2:
			message = res.getString(R.string.errorCode_2);
			break;
		case 3:
			message = res.getString(R.string.errorCode_3);
			break;
		case 4:
			message = res.getString(R.string.errorCode_4);
			break;
		case 5:
```

```
		message = res.getString(R.string.errorCode_5);
				break;
			case 6:
				message = res.getString(R.string.errorCode_6);
				break;
			case 7:
				message = res.getString(R.string.errorCode_7);
				break;
			default:
				message = res.getString(R.string.errorCode_x);
				break;
			}
			return message;
		}
```

Μέθοδος μετατροπής κειμένου σε bytes

```
public static byte[] StringToByteArray(String str) {
			try {
				byte[] bytes = new byte[str.length()];
				for (int i = 0; i < str.length(); i++)
					bytes[i] = (byte) str.charAt(i);
				return bytes;
			} catch (Exception e) {
				Log.e(tag, e.getMessage());
				return null;
			}
}
```

Μέθοδος προσάρτησης πίνακα byte σε εικόνα

```
//@TargetApi(Build.VERSION_CODES.FROYO)
public void setImageDrawable(byte[] bytes, ImageView iv) {
			if (bytes != null) {
				bytes = Base64.decode(bytes, Base64.DEFAULT);
				Bitmap bmp = BitmapFactory.decodeByteArray(bytes, 0,
bytes.length);
				iv.setImageBitmap(bmp);
			} else {
		iv.setBackgroundDrawable(ctx.getResources().getDrawable(R.drawable.add));
			}
		}
```

Εμφάνιση μηνύματος διαλόγου με δυνατότητα επιλογής ok ή cancel

```
public void showOKCancelDialog(String title, String msg, final DialogActions
da) {
		AlertDialog.Builder myAlertDialog = new AlertDialog.Builder(ctx);
		myAlertDialog.setTitle(title);
		myAlertDialog.setMessage(msg);

		myAlertDialog.setPositiveButton("OK", new
		DialogInterface.OnClickListener() {
				public void onClick(DialogInterface arg0, int arg1) {
					if (da.deleteCar())
						da.refreshUI();
				}
```

```
	});
myAlertDialog.setNegativeButton("Ακυρο", new DialogInterface.OnClickListener()
{
	public void onClick(DialogInterface arg0, int arg1) {}
});

	myAlertDialog.show();
}

interface DialogActions {
		abstract boolean deleteCar();
		abstract void refreshUI();
}
```

Λήψη δεδομένων Json από αίτηση σε webserver ορισμένου url

```
public static JSONArray getJSONFromUrl(String url) {
		InputStream is = null;
		JSONArray jArray = null;
		String json = null;
		// Making HTTP request
		try {
```

Αίτηση σε webserver

```
			// defaultHttpClient
			DefaultHttpClient httpClient = new DefaultHttpClient();
			HttpGet httpGet = new HttpGet(url);

			HttpResponse httpResponse = httpClient.execute(httpGet);
			HttpEntity httpEntity = httpResponse.getEntity();
			is = httpEntity.getContent();

		} catch (UnsupportedEncodingException e) {
			e.printStackTrace();
		} catch (ClientProtocolException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}
```

Λήψη Απάντησης από webserver

```
		try {
			BufferedReader reader = new BufferedReader(new
			InputStreamReader(is, "UTF-8"), 8);
			StringBuilder sb = new StringBuilder();
			String line = null;
			while ((line = reader.readLine()) != null) {
				sb.append(line + "\n");
			}
			is.close();
			json = sb.toString();
		} catch (Exception e) {
```

```
                Log.e("Buffer Error", "Error converting result " +
                e.toString());
        }
```

Μετατροπή απάντησης σε μορφή json

```
        // try parse the string to a JSON Array (changed from object)
        try {
                jArray = new JSONArray(json);
        } catch (JSONException e) {
                Log.e("JSON Parser", "Error parsing data " + e.toString());
        }

        // return JSON Array
        return jArray;
}
```

Λήψη δεδομένων οχήματος από τον server. Χρήσιμη συνάρτηση για συγχρονισμό εφαρμογής με webserver

```
public static JSONObject getJSONObjectCarQueryAPI(String url) {
        Log.i(tag, "Getting JSON from " + url);
        InputStream is = null;
        JSONObject jObj = null;
        String json = null;
        // Making HTTP request
        try {
                // defaultHttpClient
                DefaultHttpClient httpClient = new DefaultHttpClient();
                HttpGet httpGet = new HttpGet(url);
```

Αίτηση σε webserver και λήψη απάντησης

```
HttpConnectionParams.setConnectionTimeout(httpClient.getParams(),
CONNECTION_TO_ms);
HttpConnectionParams.setSoTimeout(httpClient.getParams(), SOCKET_TO_ms);
                HttpResponse httpResponse = httpClient.execute(httpGet);
                HttpEntity httpEntity = httpResponse.getEntity();
                is = httpEntity.getContent();

        } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
        } catch (ClientProtocolException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }

        try {
                BufferedReader reader = new BufferedReader(new
                InputStreamReader(is, "UTF-8"), 8);
                StringBuilder sb = new StringBuilder();
                String line = null;
                while ((line = reader.readLine()) != null) {
                        sb.append(line + "\n");
                }
                is.close();
                json = sb.toString();
```

```
                //Remove invalid characters
                json = json.substring(3, json.length() - 2);

        } catch (Exception e) {
                Log.e(tag, "Buffer Error: Error converting result " +
                e.toString());
        }
```

Μετατροπή δεδομένων σε μορφή json, επιστροφή του αντικειμένου στην καλούσα μέθοδο.

```
        try {
                if (json != null)
                        jObj = new JSONObject(json);
                else
                        return null;
        } catch (JSONException e) {
                Log.e(tag, "JSON Parser: Error parsing data " +
                e.toString());
        }
        // return JSON Object
        return jObj;
}
```

Κλάση progressbar για εμφάνιση στον χρήστη μπάρας προόδου με κειμένο

```
public static class CustomProgressDialog extends ProgressDialog {

        public CustomProgressDialog(Context context) {
                super(context);
        }

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                try {
                        Method method =
                        TextView.class.getMethod("setVisibility",
                        Integer.TYPE);

                        Field[] fields =
                        this.getClass().getSuperclass().getDeclaredFields();

                        for (Field field : fields) {
                if (field.getName().equalsIgnoreCase("mProgressNumber")) {
                                field.setAccessible(true);
                                TextView textView = (TextView) field.get(this);
                                        method.invoke(textView, View.GONE);
                                }
                        }
                } catch (Exception e) {
                        Log.e(tag, "Failed to invoke the progressDialog
                        method 'setVisibility' and set 'mProgressNumber' to
                        GONE.", e);
                }
        }
}
```

Έλεγχος κενού κειμένου,null ή κειμένου null. Χρήσιμη συνάρτηση για διάφορες περιπτώσεις λήψης κενής τιμής απο τον server

```
	public static String checkNullOrEmpty(String string) {
	if (string != null && string.length() > 0 && !string.equals("null"))
			return string;
		else
			return "N/A";
	}
```

Επιστροφή κειμένου μέγιστου ορισμένου μήκους. Αν το κειμενο είναι μεγαλύτερο επιστρέφετε το οριζόμενο max με τρεις τελείες. Χρησιμοποιείται για την τακτοποιημένη εμφάνιση των κειμένων

```
	public static String checkLength(String string) {
		if (string.length() > STRING_MAX_LENGTH)
			return string.substring(0, STRING_MAX_LENGTH - 3) + "...";
		else
			return string;
	}
	public static int getApiVersionNumber() {
		return android.os.Build.VERSION.SDK_INT;
	}

}
```

3.4 Αρχείο DatabaseHandler.java

Περιέχει Συναρτήσεις διαχείρισης βάσης δεδομένων όπως εισαγωγής, ενημέρωσης και ανάκτησης εγγραφών. Επίσης κατά την πρώτη εκτέλεση της εφαρμογής δημιουργείται **On the fly** η βάση δεδομένων. Η κλάση κληρονομεί στην SqliteOpenHelper ώστε να διαχειριστεί βασικές λειτουργίες βάσεων δεδομένων όπως άνοιγμα βάσης, εκτέλεσης ερωτημάτων, ανάκτησης δεδομένων κτλ.

```
public class DatabaseHandler extends SQLiteOpenHelper {
```

Ορισμός των ονομάτων πινάκων και των πεδίων ώστε να κατασκευαστούν on the fly το σχήμα της βάσης στην τοπική sqlite

```
	Context contextInDB;
	public static int DB_VERSION = 0;
	static String SP_DB_VERSION = "DataBase Version";
	static final String dbName = "VehicleManagerDB";
	static final String tag = "Debug DatabaseHandler";

	// TABLE NAMES
	static final String carTable = "CarTable";
	static final String manufacturerTable = "ManufacturerTable";
	static final String modelTable = "ModelTable";
	static final String serviceTable = "ServiceTable";
	static final String fuelTable = "FuelTable";
```

```
static final String routeTable = "RouteTable";
static final String routeRecordsTable = "RouteRecordsTable";

//CarTable Fields
static final String CAR_id = "Id";
static final String CAR_modelId = "ModelId";
static final String CAR_manufacturer = "Manufacturer";
static final String CAR_model = "Model";
static final String CAR_licencePlate = "LicencePlate";
static final String CAR_modelYear = "ModelYear";
static final String CAR_icon = "Icon";
static final String CAR_fuelType = "FuelType";
static final String CAR_insuranceDateFrom = "InsuranceDateFrom";
static final String CAR_insuranceDateTo = "InsuranceDateTo";

//ManufacturerTable Fields
static final String MAN_cqa_id = "CQAId";
static final String MAN_name = "Manufacturer";
static final String MAN_country = "Country";
static final String MAN_isFullySynched = "IsFullySynched";
static final String MAN_minYearSynched = "MinYearSynched";
static final String MAN_maxYearSynched = "MaxYearSynched";

//ModelTable Fields
static final String MOD_id = "Id";
static final String MOD_make_id = "MakeId";
static final String MOD_name = "Name";
static final String MOD_trim = "Trim";
static final String MOD_year = "Year";
static final String MOD_body = "Body";

static final String MOD_engine_position = "EnginePosition";
static final String MOD_engine_cc = "EngineCC";
static final String MOD_engine_cyl = "EngineCyl";
static final String MOD_engine_type = "EngineType";
static final String MOD_engine_valves_per_cyl = "EngineValvesPerCyl";
static final String MOD_engine_power_ps = "EnginePowerPS";
static final String MOD_engine_power_rpm = "EnginePowerRPM";
static final String MOD_engine_torque_nm = "EngineTorqueNM";
static final String MOD_engine_torque_rpm = "EngineTorqueRMP";
static final String MOD_engine_bore_mm = "EngineBoremm";
static final String MOD_engine_stroke_mm = "EngineStrokemm";
static final String MOD_engine_compression = "EngineCompression";
static final String MOD_engine_fuel = "EngineFuel";
static final String MOD_top_speed_kph = "TopSpeed";
static final String MOD_0_to_100_kph = "ZeroTo100kph";
static final String MOD_drive = "Drive";
static final String MOD_transmission_type = "TransmissionType";
static final String MOD_seats = "Seats";
static final String MOD_doors = "Doors";
static final String MOD_weight_kg = "WeightKg";
static final String MOD_length_mm = "Lengthmm";
static final String MOD_width_mm = "Widthmm";
static final String MOD_height_mm = "Heightmm";
static final String MOD_wheelbase_mm = "Wheelbasemm";
static final String MOD_lkm_hwy = "LkmHwy";
static final String MOD_lkm_mixed = "LkmMixed";
static final String MOD_lkm_city = "LkmCity";
```

```
static final String MOD_fuel_cap_l = "FuelCapL";
static final String MOD_sold_in_us = "SoldInUs";
static final String MOD_co2 = "Co2";
static final String MOD_display = "Display";
static final String MOD_make_display = "MakeDisplay";
static final String MOD_country = "Country";

// ServiceTable Fields
static final String SER_id = "Id";
static final String SER_date = "Date";
static final String SER_time = "Time";
static final String SER_kms = "Kms";
static final String SER_price = "Price";
static final String SER_reason = "Reason";
static final String SER_comments = "Comments";
static final String SER_carId = "CarId";
static final String SER_dateRev = "OrderingDate";

// FuelTable Fields
static final String FUE_id = "Id";
static final String FUE_date = "Date";
static final String FUE_time = "Time";
static final String FUE_kms = "Kms";
static final String FUE_price_gasoline = "PriceGasoline";
static final String FUE_price_gas = "PriceGas";
static final String FUE_price_diesel = "PriceDiesel";
static final String FUE_amountPaid = "AmountPaid";
static final String FUE_type = "Type";
static final String FUE_carId = "CarId";
static final String FUE_dateRev = "OrderingDate";

//RouteTable Fields
static final String ROU_id = "Id";
static final String ROU_startLat = "StartLat";
static final String ROU_startLon = "StartLon";
static final String ROU_endLat = "EndLat";
static final String ROU_endLon = "EndLon";
static final String ROU_title = "Title";

//RouteRecordsTable Fields
static final String RRT_id = "Id";
static final String RRT_routeId = "RouteId";
static final String RRT_carId = "CarId";
static final String RRT_time = "Time";
static final String RRT_date = "Date";
static final String RRT_dateRev = "DateRev";
static final String RRT_startTime = "StartTime";
static final String RRT_endTime = "EndTime";
```

Ορισμός των ερωτημάτων δημιουργίας πινάκων. Αυτά θα χρησιμοποιηθούν από συναρτήσεις εκτέλεσης sql ώστε να δημιουργηθούν οι πίνακες την πρώτη φορά εκτέλεσης της εφαρμογής

```
// CarTable String
static final String DATABASE_CREATE_CAR_TABLE = "CREATE TABLE if not exists " +
carTable + "(" + CAR_id + " INTEGER PRIMARY KEY AUTOINCREMENT, " + CAR_modelId
+ " TEXT, " + CAR_manufacturer + " TEXT NOT NULL, " + CAR_model
```

```
+ " TEXT NOT NULL, " + CAR_licencePlate + " TEXT, " + CAR_modelYear + " TEXT, "
+ CAR_icon + " BLOB, " + CAR_fuelType
                    + " INTEGER NOT NULL, " + CAR_insuranceDateFrom + " TEXT, "
+ CAR_insuranceDateTo + " TEXT " + ");";
      // ManufacturerTable String
      static final String DATABASE_CREATE_MANUFACTURER_TABLE = "CREATE TABLE if
not exists " + manufacturerTable + "(" + MAN_cqa_id
+ " TEXT PRIMARY KEY, " + MAN_name + " TEXT NOT NULL, " + MAN_country + " TEXT
NOT NULL, " + MAN_isFullySynched + " INTEGER NOT NULL, " + MAN_minYearSynched +
" INTEGER, " + MAN_maxYearSynched + " INTEGER " + ");";
      // ModelTable String

static final String DATABASE_CREATE_MODEL_TABLE = "CREATE TABLE if not exists "
+ modelTable + "(" + MOD_id + " INTEGER PRIMARY KEY, "
+ MOD_make_id + " STRING, " + MOD_name + " STRING, " + MOD_trim + " STRING, " +
MOD_year + " STRING, " + MOD_body + " STRING, "
+ MOD_engine_position + " STRING, " + MOD_engine_cc + " STRING, " +
MOD_engine_cyl + " STRING, " + MOD_engine_type
+ " STRING, " + MOD_engine_valves_per_cyl + " STRING, " + MOD_engine_power_ps +
" STRING, " + MOD_engine_power_rpm
+ " STRING, " + MOD_engine_torque_nm + " STRING, " + MOD_engine_torque_rpm + "
STRING, " + MOD_engine_bore_mm + " STRING, "
+ MOD_engine_stroke_mm + " STRING, " + MOD_engine_compression + " STRING, " +
MOD_engine_fuel + " STRING, " + MOD_top_speed_kph
+ " STRING, " + MOD_0_to_100_kph + " STRING, " + MOD_drive + " STRING, " +
MOD_transmission_type + " STRING, " + MOD_seats
+ " STRING, " + MOD_doors + " STRING, " + MOD_weight_kg + " STRING, " +
MOD_length_mm + " STRING, " + MOD_width_mm
+ " STRING, " + MOD_height_mm + " STRING, " + MOD_wheelbase_mm + " STRING, " +
MOD_lkm_hwy + " STRING, " + MOD_lkm_mixed
+ " STRING, " + MOD_lkm_city + " STRING, " + MOD_fuel_cap_l + " STRING, " +
MOD_sold_in_us + " STRING, " + MOD_co2
+ " STRING, " + MOD_display + " STRING, " + MOD_make_display + " STRING, " +
MOD_country + " STRING " + ");";

// ServiceTable String
static final String DATABASE_CREATE_SERVICE_TABLE = "CREATE TABLE if not exists
" + serviceTable + "(" + SER_id
+ " INTEGER PRIMARY KEY AUTOINCREMENT, " + SER_date + " TEXT NOT NULL, " +
SER_time + " TEXT NOT NULL, " + SER_kms
+ " INTEGER NOT NULL, " + SER_price + " REAL NOT NULL, " + SER_reason + " TEXT
, " + SER_comments + " TEXT,  " + SER_carId
+ " INTEGER,  " + SER_dateRev + " TEXT NOT NULL, " + "FOREIGN KEY (" +
SER_carId + ") REFERENCES " + carTable + " (" + CAR_id
                    + ") ON DELETE CASCADE" + ");";

// FuelTable String
static final String DATABASE_CREATE_FUEL_TABLE = "CREATE TABLE if not exists "
+ fuelTable + "(" + FUE_id + " INTEGER PRIMARY KEY AUTOINCREMENT, " + FUE_date
+ " TEXT NOT NULL, " + FUE_time + " TEXT NOT NULL, " + FUE_kms
+ " INTEGER NOT NULL, " + FUE_price_gasoline + " REAL, " + FUE_price_gas + "
REAL, " + FUE_price_diesel + " REAL, "
+ FUE_amountPaid + " REAL NOT NULL, " + FUE_type + " INTEGER NOT NULL, " +
FUE_carId + " INTEGER, " + FUE_dateRev
+ " TEXT NOT NULL, " + "FOREIGN KEY (" + FUE_carId + ") REFERENCES " + carTable
+ " (" + CAR_id + ") ON DELETE CASCADE" + ");";

// RouteTable String
```

```
static final String DATABASE_CREATE_ROUTE_TABLE = "CREATE TABLE if not exists "
+ routeTable + "(" + ROU_id + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
ROU_startLat + " REAL NOT NULL, " + ROU_startLon + " REAL NOT NULL, " +
ROU_endLat + " REAL NOT NULL, " + ROU_endLon + " REAL NOT NULL, " + ROU_title +
" TEXT NOT NULL" + ");";

// RouteRecordsTable String
static final String DATABASE_CREATE_ROUTE_RECORDS_TABLE = "CREATE TABLE if not
exists " + routeRecordsTable + "(" + RRT_id + " INTEGER PRIMARY KEY
AUTOINCREMENT, " + RRT_routeId + " INTEGER NOT NULL, " + RRT_carId + " INTEGER
NOT NULL, " + RRT_time + " INTEGER NOT NULL, " + RRT_date + " TEXT NOT NULL, "
+ RRT_dateRev + " TEXT NOT NULL, " + RRT_startTime + " TEXT NOT NULL, "
+ RRT_endTime + " TEXT NOT NULL, " + "FOREIGN KEY (" + RRT_routeId + ")
REFERENCES " + routeTable + " (" + ROU_id
+ ") ON DELETE CASCADE, " + "FOREIGN KEY (" + RRT_carId + ") REFERENCES " +
carTable + " (" + CAR_id + ") ON DELETE CASCADE"
                    + ");";

//////////////////////////END OF STRING DECLARATIONS/////////////////////
```

Δήλωση βάσης δεδομένων και έκδοσης της μέσω του δομητή της κλάσης.

```
      //CONSTRUCTOR
      public DatabaseHandler(Context context, int version) {
            super(context, dbName, null, version);
            Log.i("DataBase Constructor", "DataBase Version: " + version);
            contextInDB = context;
            DB_VERSION = version;
      }
```

Άνοιγμα Βάσης δεδομένων και επιστροφή αντικειμένου.

```
public SQLiteDatabase getDatabase(String path, String name) {
            return SQLiteDatabase.openDatabase(path + name, null,
            SQLiteDatabase.NO_LOCALIZED_COLLATORS);
}
```

Η συνάρτηση onCreate καλείται όταν η βάση που έχει οριστεί στον constructor δεν υπάρχει. Στην περίπτωση αυτή δημιουργείται και έπειτα εκτελούνται τα queries δημιουργίας πινάκων

```
public void onCreate(SQLiteDatabase db) {
            db.execSQL(DATABASE_CREATE_CAR_TABLE);
            db.execSQL(DATABASE_CREATE_SERVICE_TABLE);
            db.execSQL(DATABASE_CREATE_FUEL_TABLE);
            db.execSQL(DATABASE_CREATE_MANUFACTURER_TABLE);
            db.execSQL(DATABASE_CREATE_MODEL_TABLE);
            db.execSQL(DATABASE_CREATE_ROUTE_TABLE);
            db.execSQL(DATABASE_CREATE_ROUTE_RECORDS_TABLE);
}// /onCreate() END
```

Κατά το άνοιγμα της βάσης ελέγχεται η πρόσβαση της και έπειτα ενεργοποιείται ο έλεγχος ξένων κλειδιών.

```
@Override
```

```
public void onOpen(SQLiteDatabase db) {
	super.onOpen(db);
		if (!db.isReadOnly()) {
			//Log.i(tag, "Enabling foreign key constraints");
			db.execSQL("PRAGMA foreign_keys=ON;");
		}
	}
```

Εάν η έκδοση της βάσης είναι διαφορετική από αυτή που δίνεται ως παράμετρο στον constructor θα εκτελεστούν οι παρακάτω εντολής διαγραφής πινάκων και έπειτα η επαναδημιουργία τους.

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
	Log.i(tag, "Upgrading database from version " + oldVersion + " to
version" + newVersion);
	//db.execSQL("DROP TABLE IF EXISTS " + carTable);
	//db.execSQL("DROP TABLE IF EXISTS " + serviceTable);
	//db.execSQL("DROP TABLE IF EXISTS " + fuelTable);
	db.execSQL("DROP TABLE IF EXISTS " + manufacturerTable);
	db.execSQL("DROP TABLE IF EXISTS " + modelTable);
	onCreate(db);
}

public void attachDatabase() {
		Log.i(tag, "Attaching Database...");
		SQLiteDatabase db = this.getWritableDatabase();
		db.execSQL("attach database ? as initDB", new String[] {
		Environment.getExternalStorageDirectory().getAbsolutePath() + "/"
		+ getCarDBTask.fileName });
		db.execSQL(DATABASE_CREATE_MANUFACTURER_TABLE);
		db.execSQL(DATABASE_CREATE_MODEL_TABLE);
		db.execSQL("INSERT INTO " + manufacturerTable + " SELECT  * FROM
		initDB." + manufacturerTable);
		db.execSQL("INSERT INTO " + modelTable + " SELECT  * FROM initDB."
		+ modelTable);
		try {
			String selectQuery = "SELECT  * FROM initDB.YearsTable";
			Cursor cursor = db.rawQuery(selectQuery, null);
			if (cursor != null && cursor.moveToFirst()) {
				int minYear = cursor.getInt(1);

				int maxYear = cursor.getInt(2);
				if (minYear > 0)
					MainActivity.sp.edit().putInt("min_year",
					minYear).commit();
				if (maxYear > 0)
					MainActivity.sp.edit().putInt("max_year",
					maxYear).commit();
			}

			db.close();
		} catch (NumberFormatException e) {
			Log.e(tag, "getCar: " + e.getMessage());
			db.close();
		}
	}
```

Λήψη έκδοσης της βάσης δεδομένων

```
public int getDBVersion() {
	SQLiteDatabase db = this.getWritableDatabase();
	int version = 0;
	try {
		String query = "pragma user_version";
		Cursor cursor = db.rawQuery(query, null);
		if (cursor.moveToFirst())
			version = cursor.getInt(0);
		db.close();
		return version;
	} catch (NumberFormatException e) {
		Log.e(tag, "getAllCars: " + e.getMessage());
		db.close();
		return 0;
	}
}
```

Προσθήκη οχήματος στην βάση δεδομένων.

```
/* ------------------------| | CAR TABLE CRUD | */
// Adding new car
boolean addCar(CarClass x) {
```

Λήψη βάσης δεδομένων

```
	SQLiteDatabase db = this.getWritableDatabase();
	try {
```

Εκχώρηση τιμής για κάθε πεδίο της βάσης δεδομένων λαμβάνοντας τις τιμές του αντικειμένου οχήματος

```
		ContentValues values = new ContentValues();

		values.put(CAR_modelId, x.getModelId());
		values.put(CAR_manufacturer, x.getManufacturer());
		values.put(CAR_model, x.getModel());
		values.put(CAR_licencePlate, x.getLicencePlate());
		values.put(CAR_modelYear, x.getModelYear());
		values.put(CAR_icon, x.getIcon());
		values.put(CAR_fuelType, x.getFuelType());
		values.put(CAR_insuranceDateFrom, x.getInsuranceDateFrom());
		values.put(CAR_insuranceDateTo, x.getInsuranceDateTo());

		// Inserting Row
```

Προσθήκη εγγραφής στον πίνακα οχημάτων της βάσης δεδομένων

```
		db.replace(carTable, null, values);
		db.close(); // Closing database connection
		Log.i(tag, "Successfuly added car");
		return true;
	} catch (Exception e) {
		Log.e(tag, "addCar: " + e.getMessage());
		db.close();
		return false;
	}
}
```

Ενημέρωση οχήματος της βάση δεδομένων.

```
// Updating single car
public boolean updateCar(CarClass x) {
```

Λήψη βάσης δεδομένων

```
	SQLiteDatabase db = this.getWritableDatabase();
	try {
```

Εκχώρηση τιμής για κάθε πεδίο της βάσης δεδομένων λαμβάνοντας τις τιμές του αντικειμένου οχήματος

```
		ContentValues values = new ContentValues();

		values.put(CAR_modelId, x.getModelId());
		values.put(CAR_manufacturer, x.getManufacturer());
		values.put(CAR_model, x.getModel());
		values.put(CAR_licencePlate, x.getLicencePlate());
		values.put(CAR_modelYear, x.getModelYear());
		values.put(CAR_icon, x.getIcon());
		values.put(CAR_fuelType, x.getFuelType());
		values.put(CAR_insuranceDateFrom, x.getInsuranceDateFrom());
		values.put(CAR_insuranceDateTo, x.getInsuranceDateTo());
```

Ενημέρωση εγγραφής στον πίνακα οχημάτων της βάσης δεδομένων δεδομένου id οχήματος.

```
		// Updating row
		db.update(carTable, values, CAR_id + " = ?", new String[] {
		String.valueOf(x.getId()) });
		db.close();
		return true;
	} catch (Exception e) {
		Log.e(tag, "updateCar: " + e.getMessage());
		db.close();
		return false;
	}

}
```

Ενημέρωση εικόνας ενός οχήματος στην βάση δεδομένων

```
// Updating car's img
public boolean updateCarImg(long carId, Bitmap bmp) {
	SQLiteDatabase db = this.getWritableDatabase();
	try {
		ContentValues values = new ContentValues();
```

Μετατροπή Bitmap σε byteArray ώστε να είναι συμβατή η αποθήκευση στην βάση δεδομένων

```
		ByteArrayOutputStream stream = new ByteArrayOutputStream();
		bmp.compress(Bitmap.CompressFormat.PNG, 100, stream);
		byte[] byteArray = stream.toByteArray();
		values.put(CAR_icon, byteArray);
```

Αποθήκευση εικόνας οχήματος στην βάση δεδομένων

```
		// Updating row
		db.update(carTable, values, CAR_id + " = ?", new String[] {
		String.valueOf(carId) });
		db.close();
		return true;
	} catch (Exception e) {
		Log.e(tag, "updateCar: " + e.getMessage());
		db.close();
		return false;
	}

}
```

Λήψη δεδομένων οχήματος συγκεκριμένου id

```
// Getting single car
public CarClass getCar(long id) {
	CarClass x = null;
	SQLiteDatabase db = this.getReadableDatabase();
	try {
		Ανάκτηση δεδομένων του οχήματος
		Cursor cursor = db.query(carTable, new String[] { CAR_id,
		CAR_modelId, CAR_manufacturer, CAR_model, CAR_licencePlate,
		CAR_modelYear, CAR_icon, CAR_fuelType,
		CAR_insuranceDateFrom, CAR_insuranceDateTo }, CAR_id + "=?",
		new String[] { String.valueOf(id) }, null, null, null,
		null);

		if (cursor != null)
			cursor.moveToFirst();

		Δημιουργία αντικειμένου οχήματος με τιμές αυτές που
		ανακτήθηκαν από την βάση δεδομένων
		x = new CarClass(cursor.getLong(0), cursor.getString(1),
		cursor.getString(2), cursor.getString(3),
		cursor.getString(4),
			cursor.getString(5), cursor.getBlob(6),
			cursor.getInt(7), cursor.getString(8),
			cursor.getString(9));
		db.close();

		Επιστροφή του οχήματος
		return x;

	} catch (NumberFormatException e) {
		Log.e(tag, "getCar: " + e.getMessage());
		x = null;
		db.close();
		return x;
	}
}
```

Λήψη όλων των οχημάτων της βάσης δεδομένων με μορφή λίστας

```
// Getting All cars
public List<CarClass> getAllCars() {
	List<CarClass> xList = null;
	SQLiteDatabase db = this.getWritableDatabase();
	try {
```

```
		xList = new ArrayList<CarClass>();

		Query ανάκτησης όλων των οχημάτων
		// Select All Query
		String selectQuery = "SELECT * FROM " + carTable;
		Cursor cursor = db.rawQuery(selectQuery, null);

		Προσθήκη όλων των οχημάτων σε λίστα οχημάτων
		// looping through all rows and adding to list
		if (cursor.moveToFirst()) {
			do {
		CarClass x = new CarClass(cursor.getLong(0),
		cursor.getString(1), cursor.getString(2),
		cursor.getString(3),
		cursor.getString(4), cursor.getString(5), cursor.getBlob(6),
		cursor.getInt(7), cursor.getString(8),
		cursor.getString(9));
		xList.add(x);
			} while (cursor.moveToNext());
		}
		db.close();

		Επιστροφή της λίστας οχημάτων που ανακτήθηκαν από την
		βάση δεδομένων
		return xList;

	} catch (NumberFormatException e) {
		Log.e(tag, "getAllCars: " + e.getMessage());
		xList = null;
		db.close();
		return xList;
	}
}

Διαγραφή οχήματος από την βάση δεδομένων
// Deleting single car
public boolean deleteCar(CarClass x) {
	SQLiteDatabase db = this.getWritableDatabase();
	try {
		Εκτέλεση ερωτήματος διαγραφής στην βάση
		db.delete(carTable, CAR_id + " = ?", new String[] {
		String.valueOf(x.getId()) });
		db.close();
		return true;
	} catch (Exception e) {
		Log.e(tag, "deleteCar: " + e.getMessage());
		db.close();
		return false;
	}
}

// Getting car Count
public int getCarCount() {
	String countQuery = "SELECT  * FROM " + carTable;
	SQLiteDatabase db = this.getReadableDatabase();
	Cursor cursor = db.rawQuery(countQuery, null);
	int count = cursor.getCount();
```

```
        cursor.close();
        db.close();
        // return count
        return count;
}


 * ----------------------| | MANUFACTURER TABLE CRUD | |---------------|
```

Προσθήκη κατασκευαστή στην βάση

```
// Adding new manufacturer
long addManufacturer(DBManufacturer x) {
        SQLiteDatabase db = this.getWritableDatabase();
        try {
                ContentValues values = new ContentValues();
```

Προετοιμασία δεδομένων προς αποστολή στην βάση

```
                values.put(MAN_cqa_id, x.getCqa_id());
                values.put(MAN_name, x.getName());
                values.put(MAN_country, x.getCountry());
                values.put(MAN_isFullySynched, x.isFullySynched() ? 1 : 0);
                values.put(MAN_minYearSynched, x.getMinYearSynched());
                values.put(MAN_maxYearSynched, x.getMaxYearSynched());
```

Εκτέλεση εισαγωγής εγγραφής κατασκευαστή στον πίνακα δεδομένων

```
                // Inserting Row
                long id = db.replace(manufacturerTable, null, values);
                db.close(); // Closing database connection
                //Log.i(tag, "Successfuly added manufacturer");
                return id;
        } catch (Exception e) {
                Log.e(tag, "addManufacturer: " + e.getMessage());
                db.close();
                return -1;
        }
}
```

Ενημέρωση κατασκευαστή στον πίνακα δεδομένων

```
// Updating single manufacturer
public boolean updateManufacturer(DBManufacturer x) {
        SQLiteDatabase db = this.getWritableDatabase();
        try {
                ContentValues values = new ContentValues();

                //values.put(MAN_cqa_id, x.getCqa_id());
                values.put(MAN_name, x.getName());

                values.put(MAN_country, x.getCountry());
                values.put(MAN_isFullySynched, x.isFullySynched() ? 1 : 0);
                values.put(MAN_minYearSynched, x.getMinYearSynched());
                values.put(MAN_maxYearSynched, x.getMaxYearSynched());

                // Updating row
                db.update(manufacturerTable, values, MAN_cqa_id + " = ?",
                new String[] { x.getCqa_id() });
```

```
			db.close();
			return true;
		} catch (Exception e) {
			Log.e(tag, "updateManufacturer: " + e.getMessage());
			db.close();
			return false;
		}

	}
```

Λήψη κατασκευαστή από την βάση

```
	// Getting single manufactuer
	DBManufacturer getManufacturer(String cqa_id) {
		DBManufacturer x = null;
		SQLiteDatabase db = this.getReadableDatabase();
		try {
```

Ερώτημα στην βάση δεδομένων

```
			Cursor cursor = db.query(manufacturerTable, new String[] {
			MAN_cqa_id, MAN_name, MAN_country, MAN_isFullySynched,
			MAN_minYearSynched, MAN_maxYearSynched }, MAN_cqa_id + "=?",
			new String[] { cqa_id }, null, null, null, null);
			if (cursor != null)
				cursor.moveToFirst();
```

Δημιουργία αντικειμένου κατασκευαστή με τιμές που ανακτήθηκαν από τον πίνακα κατασκευαστών της βάσης δεδομένων

```
			x = new DBManufacturer(cursor.getString(0),
			cursor.getString(1), cursor.getString(2), cursor.getInt(3)
			== 1 ? true : false,
					cursor.getInt(4), cursor.getInt(5));
			db.close();
			return x;
		} catch (NumberFormatException e) {
			Log.e(tag, "getManufacturer: " + e.getMessage());
			x = null;
			db.close();
			return x;
		}
	}
```

Λήψη όλων των κατασκευαστών της βάσης δεδομένων με μορφή λίστας

```
	// Getting All manufacturers
	public List<DBManufacturer> getAllManufacturers() {
		List<DBManufacturer> xList = null;
		SQLiteDatabase db = this.getWritableDatabase();
		try {
			xList = new ArrayList<DBManufacturer>();
```

Query ανάκτησης όλων των κατασκευαστών

```
			// Select All Query
			String selectQuery = "SELECT  * FROM " + manufacturerTable;
```

```
		Cursor cursor = db.rawQuery(selectQuery, null);

		Προσθήκη όλων των κατασκευαστών σε λίστα οχημάτων
		if (cursor.moveToFirst()) {
			do {
				DBManufacturer x = new
				DBManufacturer(cursor.getString(0),
				cursor.getString(1), cursor.getString(2),
							cursor.getInt(3) == 1 ? true :
				false, cursor.getInt(4), cursor.getInt(5));
				xList.add(x);
			} while (cursor.moveToNext());
		}
		db.close();

		Επιστροφή της λίστας κατασκευαστών οχημάτων που
		ανακτήθηκαν από την βάση δεδομένων

		return xList;

	} catch (NumberFormatException e) {
		Log.e(tag, "getAllManufacturers: " + e.getMessage());
		xList = null;
		db.close();
		return xList;
	}
}
```

Διαγραφή κατασκευαστή οχημάτων από την βάση δεδομένων

```
// Deleting single manufacturer
public boolean deleteManufacturer(DBManufacturer x) {
	SQLiteDatabase db = this.getWritableDatabase();
	try {
		Εκτέλεση ερωτήματος διαγραφής στην βάση
		db.delete(manufacturerTable, x.getCqa_id() + " = ?", new
		String[] { x.getCqa_id() });
		db.close();
		return true;
	} catch (Exception e) {
		Log.e(tag, "deleteManufacturer: " + e.getMessage());
		db.close();
		return false;
	}
}

// Getting manufacturer Count
public int getManufacturerCount() {
	String countQuery = "SELECT  count(*) FROM " + manufacturerTable;
	SQLiteDatabase db = this.getReadableDatabase();
	Cursor cursor = db.rawQuery(countQuery, null);
	int count = 0;
	if (cursor.moveToFirst())
		count = cursor.getInt(0);
	cursor.close();
	db.close();
	return count;
```

```
}
```

Με την ίδια λογική προσθήκης οχήματος και κατασκευαστή λειτουργεί και η προσθήκη μοντέλου οχήματος. Πρώτα δημιουργείται υποδοχέας τιμών για το ερώτημα και έπειτα εκτελείται ερώτημα εισαγωγής στον πίνακα

```
// Adding new Model
boolean addModel(DBModel x) {
	SQLiteDatabase db = this.getWritableDatabase();
	try {
		ContentValues values = new ContentValues();

		values.put(MOD_id, x.getModel_id());

		values.put(MOD_make_id, x.getModel_make_id());
		values.put(MOD_name, x.getModel_name());
		values.put(MOD_trim, x.getModel_trim());
		values.put(MOD_year, x.getModel_year());
		values.put(MOD_body, x.getModel_body());
		values.put(MOD_engine_position,
		x.getModel_engine_position());
		values.put(MOD_engine_cc, x.getModel_engine_cc());
		values.put(MOD_engine_cyl, x.getModel_engine_cyl());
		values.put(MOD_engine_type, x.getModel_engine_type());
		values.put(MOD_engine_valves_per_cyl,
		x.getModel_engine_valves_per_cyl());
		values.put(MOD_engine_power_ps,
		x.getModel_engine_power_ps());
		values.put(MOD_engine_power_rpm,
		x.getModel_engine_power_rpm());
		values.put(MOD_engine_torque_nm,
		x.getModel_engine_torque_nm());
		values.put(MOD_engine_torque_rpm,
		x.getModel_engine_torque_rpm());
		values.put(MOD_engine_bore_mm, x.getModel_engine_bore_mm());
		values.put(MOD_engine_stroke_mm,
		x.getModel_engine_stroke_mm());
		values.put(MOD_engine_compression,
		x.getModel_engine_compression());
		values.put(MOD_engine_fuel, x.getModel_engine_fuel());
		values.put(MOD_top_speed_kph, x.getModel_top_speed_kph());
		values.put(MOD_0_to_100_kph, x.getModel_0_to_100_kph());
		values.put(MOD_drive, x.getModel_drive());
		values.put(MOD_transmission_type,
		x.getModel_transmission_type());
		values.put(MOD_seats, x.getModel_seats());
		values.put(MOD_doors, x.getModel_doors());
		values.put(MOD_weight_kg, x.getModel_weight_kg());
		values.put(MOD_length_mm, x.getModel_length_mm());
		values.put(MOD_width_mm, x.getModel_width_mm());
		values.put(MOD_height_mm, x.getModel_height_mm());
		values.put(MOD_wheelbase_mm, x.getModel_wheelbase_mm());
		values.put(MOD_lkm_hwy, x.getModel_lkm_hwy());
		values.put(MOD_lkm_mixed, x.getModel_lkm_mixed());
		values.put(MOD_lkm_city, x.getModel_lkm_city());
		values.put(MOD_fuel_cap_l, x.getModel_fuel_cap_l());
		values.put(MOD_sold_in_us, x.getModel_sold_in_us());
```

```
                values.put(MOD_co2, x.getModel_co2());
                values.put(MOD_display, x.getModel_make_display());
                values.put(MOD_make_display, x.getMake_display());
                values.put(MOD_country, x.getMake_country());

                // Inserting Row
                db.replace(modelTable, null, values);
                db.close(); // Closing database connection
                //Log.i(tag, "Successfuly added Model");
                return true;
        } catch (Exception e) {
                Log.e(tag, "addModel: " + e.getMessage());
                db.close();
                return false;
        }
}
```

Ομοίως με την λογική ενημέρωσης οχήματος και κατασκευαστή οχημάτων λειτουργεί και η ενημέρωση μοντέλου οχήματος. Πρώτα δημιουργείται υποδοχέας τιμών για το ερώτημα και έπειτα εκτελείται ερώτημα ενημέρωσης μοντέλου οχημάτων στον πίνακα

```
// Updating single Model
public boolean updateModel(DBModel x) {
        SQLiteDatabase db = this.getWritableDatabase();
        try {
                ContentValues values = new ContentValues();

                //values.put(MOD_id, x.getModel_id());
                values.put(MOD_make_id, x.getModel_make_id());
                values.put(MOD_name, x.getModel_name());
                values.put(MOD_trim, x.getModel_trim());
                values.put(MOD_year, x.getModel_year());
                values.put(MOD_body, x.getModel_body());
                        values.put(MOD_engine_position,
                x.getModel_engine_position());
                values.put(MOD_engine_cc, x.getModel_engine_cc());
                values.put(MOD_engine_cyl, x.getModel_engine_cyl());
                values.put(MOD_engine_type, x.getModel_engine_type());
                values.put(MOD_engine_valves_per_cyl,
                x.getModel_engine_valves_per_cyl());
                values.put(MOD_engine_power_ps,
                x.getModel_engine_power_ps());
                values.put(MOD_engine_power_rpm,
                x.getModel_engine_power_rpm());
                values.put(MOD_engine_torque_nm,
                x.getModel_engine_torque_nm());
                values.put(MOD_engine_torque_rpm,
                x.getModel_engine_torque_rpm());
                values.put(MOD_engine_bore_mm, x.getModel_engine_bore_mm());
                values.put(MOD_engine_stroke_mm,
                x.getModel_engine_stroke_mm());
                values.put(MOD_engine_compression,
                x.getModel_engine_compression());
                values.put(MOD_engine_fuel, x.getModel_engine_fuel());
                values.put(MOD_top_speed_kph, x.getModel_top_speed_kph());
                values.put(MOD_0_to_100_kph, x.getModel_0_to_100_kph());
```

```
                values.put(MOD_drive, x.getModel_drive());
                values.put(MOD_transmission_type,
                x.getModel_transmission_type());
                values.put(MOD_seats, x.getModel_seats());
                values.put(MOD_doors, x.getModel_doors());
                values.put(MOD_weight_kg, x.getModel_weight_kg());
                values.put(MOD_length_mm, x.getModel_length_mm());
                values.put(MOD_width_mm, x.getModel_width_mm());
                values.put(MOD_height_mm, x.getModel_height_mm());
                values.put(MOD_wheelbase_mm, x.getModel_wheelbase_mm());
                values.put(MOD_lkm_hwy, x.getModel_lkm_hwy());
                values.put(MOD_lkm_mixed, x.getModel_lkm_mixed());
                values.put(MOD_lkm_city, x.getModel_lkm_city());
                values.put(MOD_fuel_cap_l, x.getModel_fuel_cap_l());
                values.put(MOD_sold_in_us, x.getModel_sold_in_us());
                values.put(MOD_co2, x.getModel_co2());
                values.put(MOD_display, x.getModel_make_display());
                values.put(MOD_make_display, x.getMake_display());
                values.put(MOD_country, x.getMake_country());

                // Updating row
                db.update(modelTable, values, MOD_id + " = ?", new String[]
                { x.getModel_id() });
                db.close();
                return true;
        } catch (Exception e) {
                Log.e(tag, "updateModel: " + e.getMessage());
                db.close();
                return false;
        }

}
```

Λήψη μοντέλου οχημάτων συγκεκριμένου id

```
// Getting single model
DBModel getModel(int id) {
        DBModel x = null;
        SQLiteDatabase db = this.getReadableDatabase();
        try {
                Cursor cursor = db.query(modelTable, new String[] { MOD_id,
                MOD_make_id, MOD_name, MOD_trim, MOD_year, MOD_body,
                MOD_engine_position, MOD_engine_cc, MOD_engine_cyl,
                MOD_engine_type, MOD_engine_valves_per_cyl,
                MOD_engine_power_ps,
                MOD_engine_power_rpm, MOD_engine_torque_nm,
                MOD_engine_torque_rpm, MOD_engine_bore_mm,
                MOD_engine_stroke_mm,
                MOD_engine_compression, MOD_engine_fuel, MOD_top_speed_kph,
                MOD_0_to_100_kph, MOD_drive, MOD_transmission_type,
                MOD_seats, MOD_doors, MOD_weight_kg, MOD_length_mm,
                MOD_width_mm, MOD_height_mm, MOD_wheelbase_mm, MOD_lkm_hwy,
                MOD_lkm_mixed, MOD_lkm_city, MOD_fuel_cap_l, MOD_sold_in_us,
                MOD_co2, MOD_display, MOD_make_display, MOD_country },
                                MOD_id + "=?", new String[] {
                        String.valueOf(id) }, null, null, null, null);

                if (cursor != null)
                        cursor.moveToFirst();
```

```
                x = new DBModel(cursor.getString(0), cursor.getString(1),
                cursor.getString(2), cursor.getString(3),
                cursor.getString(4), cursor.getString(5),
                cursor.getString(6), cursor.getString(7),
                cursor.getString(8), cursor.getString(9),
                cursor.getString(10), cursor.getString(11),
                cursor.getString(12), cursor.getString(13),
                cursor.getString(14), cursor.getString(15),
                cursor.getString(16), cursor.getString(17),
                cursor.getString(18), cursor.getString(19),
                cursor.getString(20), cursor.getString(21),
                cursor.getString(22), cursor.getString(23),
                cursor.getString(24), cursor.getString(25),
                cursor.getString(26), cursor.getString(27),
                cursor.getString(28), cursor.getString(29),
                cursor.getString(30), cursor.getString(31),
                cursor.getString(32), cursor.getString(33),
                cursor.getString(34), cursor.getString(35),
                cursor.getString(36), cursor.getString(37),
                cursor.getString(38));
                db.close();

                Επιστροφή μοντέλου οχήματος στην καλούσα μέθοδο
                return x;
        } catch (NumberFormatException e) {
                Log.e(tag, "getModel: " + e.getMessage());
                x = null;
                db.close();
                return x;
        }
}
```

Λήψη όλων των μοντέλων οχημάτων της βάσης δεδομένων με μορφή λίστας

```
// Getting All Models
public List<DBModel> getAllModels() {
        List<DBModel> xList = null;
        SQLiteDatabase db = this.getWritableDatabase();
        try {
                xList = new ArrayList<DBModel>();

                Query ανάκτησης όλων των μοντέλων οχημάτων
                // Select All Query
                String selectQuery = "SELECT  * FROM " + modelTable;
                Cursor cursor = db.rawQuery(selectQuery, null);

                Προσθήκη όλων των μοντέλων οχημάτων σε λίστα
                if (cursor.moveToFirst()) {
                        do {
                                DBModel x = new DBModel(cursor.getString(0),
                                cursor.getString(1),cursor.getString(2),cursor.
                                getString(3),cursor.getString(4),cursor.getStri
                                ng(5), cursor.getString(6),cursor.getString(7),
                                cursor.getString(8), cursor.getString(9),
                                cursor.getString(10), cursor.getString(11),
                                cursor.getString(12), cursor.getString(13),
```

```
                        cursor.getString(14), cursor.getString(15),
                        cursor.getString(16), cursor.getString(17),
                        cursor.getString(18), cursor.getString(19),
                        cursor.getString(20), cursor.getString(21),
                        cursor.getString(22), cursor.getString(23),
                        cursor.getString(24), cursor.getString(25),
                        cursor.getString(26), cursor.getString(27),
                        cursor.getString(28), cursor.getString(29),
                        cursor.getString(30), cursor.getString(31),
                        cursor.getString(32), cursor.getString(33),
                        cursor.getString(34), cursor.getString(35),
                        cursor.getString(36), cursor.getString(37),
                        cursor.getString(38));
                        xList.add(x);
                } while (cursor.moveToNext());
            }
            db.close();
```

Επιστροφή της λίστας μοντέλων οχημάτων που ανακτήθηκαν από την βάση δεδομένων

```
            return xList;

        } catch (NumberFormatException e) {
            Log.e(tag, "getAllModels: " + e.getMessage());
            xList = null;
            db.close();
            return xList;
        }
    }
```

Λήψη όλων των τύπων καυσίμων της βάσης δεδομένων με μορφή λίστας

```
    // Getting All FuelTypes DEBUG
    public List<String> getAllModelsEngineFuel() {
        List<String> xList = null;
        SQLiteDatabase db = this.getWritableDatabase();
        try {
            xList = new ArrayList<String>();
```

Query ανάκτησης όλων των τύπων καυσίμων

```
            // Select All Query
            String selectQuery = "SELECT  DISTINCT " + MOD_engine_fuel +
            " FROM " + modelTable;
            Cursor cursor = db.rawQuery(selectQuery, null);
```

Προσθήκη όλων των τύπων καυσίμων σε λίστα

```
            if (cursor.moveToFirst()) {
                do {
                    String x = new String(cursor.getString(0));
                    xList.add(x);
                } while (cursor.moveToNext());
            }
            db.close();
```

Επιστροφή της λίστας τύπων καυσίμων που ανακτήθηκαν από την βάση δεδομένων

```
                return xList;

        } catch (NumberFormatException e) {
                Log.e(tag, "getAllModelsEngineFuel: " + e.getMessage());
                xList = null;
                db.close();
                return xList;
        }
}
```

Η συνάρτηση αυτή είναι υπερφορτωμένη με παράμετρο τον κωδικό το κατασκευαστή. Έχει παρόμοια λειτουργία με την βασική συνάρτηση με την διαφορά ότι στο ερώτημα στην βάση δεδομένων το πεδίο του κατασκευαστή των μοντέλων οχημάτων θα πρέπει να ταιριάζει με αυτό που δόθηκε ως παράμετρο.

```
// Getting All Models by manufacturer
public List<DBModel> getAllModels(String manId) {
        List<DBModel> xList = null;
        SQLiteDatabase db = this.getWritableDatabase();
        try {
                xList = new ArrayList<DBModel>();
                // Select All Query
```

Η διαφοροποίηση της συνάρτησης πραγματοποιείται εδώ γράφοντας επιπλέον την συνθήκη “where MOD_make_id=value”

```
                String selectQuery = "SELECT  * FROM " + modelTable + "
                WHERE " + MOD_make_id + "=" + manId;
                Cursor cursor = db.rawQuery(selectQuery, null);
                if (cursor.moveToFirst()) {
                        do {
```

Εδώ φαίνεται η πολλαπλή απόδοση τιμών ενώς αντικειμένου. Θα μπορούσε να δημιουργηθεί constructor για την απόδοση τιμών απευθείας του πίνακα τιμών αλλα για την αποφυγή λανθασμένης αντιστοίχησης ορισμάτων τιμών και πεδίων του πίνακα προτιμήθηκε η πολλαπλή παραμετροποίηση.

```
                DBModel x = new DBModel(cursor.getString(0),
                cursor.getString(1), cursor.getString(2),
                cursor.getString(3), cursor.getString(4),
                cursor.getString(5), cursor.getString(6),
                cursor.getString(7), cursor.getString(8),
                cursor.getString(9), cursor.getString(10),
                cursor.getString(11), cursor.getString(12),
                cursor.getString(13), cursor.getString(14),
                cursor.getString(15), cursor.getString(16),
                cursor.getString(17), cursor.getString(18),
                cursor.getString(19), cursor.getString(20),
                cursor.getString(21), cursor.getString(22),
```

```
                cursor.getString(23), cursor.getString(24),
                cursor.getString(25), cursor.getString(26),
                cursor.getString(27), cursor.getString(28),
                cursor.getString(29), cursor.getString(30),
                cursor.getString(31), cursor.getString(32),
                cursor.getString(33),cursor.getString(34),
                cursor.getString(35), cursor.getString(36),
                cursor.getString(37), cursor.getString(38));
                        xList.add(x);
                } while (cursor.moveToNext());
                }
                db.close();
                return xList;

        } catch (NumberFormatException e) {
                Log.e(tag, "getAllModelsByManufacturer: " + e.getMessage());
                xList = null;
                db.close();
                return xList;
        }
}
```

Επιπλέον υπερφόρτωση της συνάρτησης getAllModels σύμφωνα με τον χρόνο κατασκευής και τον κατασκευαστή των οχημάτων

```
// Getting All Models by manufacturer AND year
public List<DBModel> getAllModels(String year, String manId) {
        List<DBModel> xList = null;
        SQLiteDatabase db = this.getWritableDatabase();
        try {
                xList = new ArrayList<DBModel>();
                // Select All Query
                String selectQuery = "SELECT  * FROM " + modelTable + "
                WHERE " + MOD_make_id + "='" + manId + "' AND " + MOD_year +
                "='" + year + "'" + " ORDER BY " + MOD_name;
                Cursor cursor = db.rawQuery(selectQuery, null);

        if (cursor.moveToFirst()) {
                do {
                DBModel x = new DBModel(cursor.getString(0),
                cursor.getString(1), cursor.getString(2),
                cursor.getString(3),cursor.getString(4),
                cursor.getString(5), cursor.getString(6),
                cursor.getString(7), cursor.getString(8),
                cursor.getString(9), cursor.getString(10),
                cursor.getString(11), cursor.getString(12),
                cursor.getString(13), cursor.getString(14),
                cursor.getString(15), cursor.getString(16),
                cursor.getString(17), cursor.getString(18),
                cursor.getString(19), cursor.getString(20),
                cursor.getString(21), cursor.getString(22),
                cursor.getString(23), cursor.getString(24),
                cursor.getString(25), cursor.getString(26),
                cursor.getString(27), cursor.getString(28),
                cursor.getString(29), cursor.getString(30),
                cursor.getString(31), cursor.getString(32),
                cursor.getString(33), cursor.getString(34),
```

```
                cursor.getString(35), cursor.getString(36),
                cursor.getString(37), cursor.getString(38));
                        xList.add(x);
                } while (cursor.moveToNext());
            }
            db.close();
            return xList;

        } catch (NumberFormatException e) {
            Log.e(tag, "getAllModelsByManufacturer: " + e.getMessage());
            xList = null;
            db.close();
            return xList;
        }
    }
```

Επιστρέφει τις χρονίες που κατασκευαστής με συκεκριμένο id έχει κατασκευάσει μοντέλα αυτές τις χρονιές.

```
    // Getting all active years for a MANUFACTURER
    public List<String> getManufacturerActiveYears(String manId) {
        List<String> xList = null;
        SQLiteDatabase db = this.getWritableDatabase();
        try {
            xList = new ArrayList<String>();
```

Βρές τις χρονιές κατασκευής μοντέλων που κατασκευάστηκαν από συγκεκριμένο κατασκευαστή, Το DISTINCT διασφαλίζει ότι δεν θα υπάρχουν δύο ίδιες χρονιές, μιας που υπάρχουν πολλές περιπτώσεις ένας κατασκευαστής να έχει κατασκευάσει παραπάνω του ενώς μοντέλου την ίδια χρονιά. Η χρονιές θα επιστραφούν χρονολογικά από τις παλαιότερες προς τις πιο πρόσφατες.

```
            // Select All Query
            String selectQuery = "SELECT DISTINCT " + MOD_year + " FROM
            " + modelTable + " WHERE " + MOD_make_id + "='" + manId +
            "'" + " ORDER BY " + MOD_year + " ASC ";
            Cursor cursor = db.rawQuery(selectQuery, null);
```

Για κάθε μια εγγραφή που βρέθηκε, προσθήκη στην λίστα String η οποία θα επιστραφεί στην καλούσα συνάρτηση.

```
            if (cursor.moveToFirst()) {
                do {
                    String x = cursor.getString(0);
                    if (x != null && x.length() > 0)
                        xList.add(x);
                } while (cursor.moveToNext());
            }
            db.close();
            return xList;
        } catch (NumberFormatException e) {
            Log.e(tag, "getAllModelsByManufacturer: " + e.getMessage());
            xList = null;
```

```
		db.close();
		return xList;
	}
}
```

Διαγραφή μοντέλου από την βάση δεδομένων

```
// Deleting single Model
public boolean deleteModel(DBModel x) {
	SQLiteDatabase db = this.getWritableDatabase();
	try {
```

Διέγραψε μοντέλο σύμφωνα με το id του αντικειμένου x που δώθηκε ως παράμετρο στην μέθοδο deleteModel

```
		db.delete(modelTable, MOD_id + " = ?", new String[] {
		String.valueOf(x.getModel_id()) });
		db.close();
		return true;
	} catch (Exception e) {
		Log.e(tag, "deleteModel: " + e.getMessage());
		db.close();
		return false;
	}
}
```

Μέθοδος εύρεσης του πλήθους των μοντέλων στην βάση δεδομένων

```
// Getting Model Count
public int getModelCount() {
	String countQuery = "SELECT  count(*) FROM " + modelTable;
	SQLiteDatabase db = this.getReadableDatabase();
	Cursor cursor = db.rawQuery(countQuery, null);
	int count = 0;
	if (cursor.moveToFirst())
		count = cursor.getInt(0);
	cursor.close();
	db.close();
	return count;
}


/* --------------------| | SERVICE TABLE CRUD | |--------------    */
```

Προσθήκη υπηρεσίας όπως σέρβις οχήματος, οδική βοήθεια κτλ στην βάση δεδομένων.

```
// Adding new service
public boolean addService(ServiceClass x) {
	SQLiteDatabase db = this.getWritableDatabase();
	try {
		ContentValues values = new ContentValues();
```

Ορισμός τιμών στο αντικείμενο ContentValues σύμφωνα με τις τιμές που έχει το αντικείμενο x

```
		values.put(SER_date, x.getDate());
```

```
                values.put(SER_time, x.getTime());
                values.put(SER_kms, x.getKms());
                values.put(SER_price, x.getAmountPaid());
                values.put(SER_reason, x.getReason());
                values.put(SER_comments, x.getComments());
                values.put(SER_carId, x.getCarId());
```

Βρές την μέρα, μήνα και χρόνο σύμφωνα με το αλφαριθμητικό της μορφής day/month/year, οπότε διέσπασε το κείμενο σύμφωνα με τον χαρακτήρα /. Έπειτα σύνθεσε την ημερομηνία σύμφωνα με την μορφή year/month/day. Η διαδικασία αυτή χρησιμοποιείται γιατί ο χρήστης δίνει την ημερομηνία με την πρώτη μορφή αλλά στην βάση η αποθήκευση πρέπει να γίνεται με την δεύτερη (year/month/day).

```
                String[] dateArray = x.getDate().split("/");
                values.put(SER_dateRev, dateArray[2] + "/" + dateArray[1] +
                "/" + dateArray[0]);
                // Inserting Row
                db.replace(serviceTable, null, values);
                db.close(); // Closing database connection
                Log.i(tag, "Successfuly added service");
                return true;
        } catch (Exception e) {
                Log.e(tag, "addService: " + e.getMessage());
                db.close();
                return false;
        }
}
```

Ενημέρωση υπηρεσίας με id το id του αντικειμένου x. Η συνέρτηση αυτή έχει παρόμοιες λειτουργίες με αυτές της addService

```
// Updating single service
public boolean updateService(ServiceClass x) {
        SQLiteDatabase db = this.getWritableDatabase();
        try {
                ContentValues values = new ContentValues();

                values.put(SER_date, x.getDate());
                values.put(SER_time, x.getTime());
                values.put(SER_kms, x.getKms());
                values.put(SER_price, x.getAmountPaid());
                values.put(SER_reason, x.getReason());
                values.put(SER_comments, x.getComments());
                values.put(SER_carId, x.getCarId());

                String[] dateArray = x.getDate().split("/");
                values.put(SER_dateRev, dateArray[2] + "/" + dateArray[1] +
                "/" + dateArray[0]);
```

Ενημέρωση τιμών εγγραφής, σύμφωνα με το id που προσδιορίζεται στο τρίτο όρισμα της συνάρτησης update. Το ερώτημα που παράγετε ουσιαστικά από την παρακάτω κλήση

είναι update table serviceTable (attr1,attr2,…) VALUES (val1,val2,…) WHERE Ser_id=value of x.id)

```
		// Updating row
		db.update(serviceTable, values, SER_id + " = ?", new
		String[] { String.valueOf(x.getId()) });
		db.close();
		return true;
	} catch (Exception e) {
		Log.e(tag, "updateService: " + e.getMessage());
		db.close();
		return false;
	}

}
```

Λήψη υπηρεσίας σύμφωνα με το id της

```
// Getting single service
ServiceClass getService(int id) {
	ServiceClass x = null;
	SQLiteDatabase db = this.getReadableDatabase();
	try {
```

Λήψη τιμών των πεδίων συγκεκριμένης υπηρεσίας από την βάση δεδομένων. Το πεδίο SER_reason προσδιορίζει τον τύπο της υπηρεσίας όπως σέρβις οχήματος, οδική βοήθεα κτλ.

```
		Cursor cursor = db.query(serviceTable, new String[] {
		SER_id, SER_date, SER_time, SER_kms, SER_price, SER_reason,
		SER_comments, SER_carId, SER_dateRev }, SER_id + "=?", new
		String[] { String.valueOf(id) }, null, null, null, null);
		if (cursor != null)
			cursor.moveToFirst();
```

Δημιουργία νέου αντικειμένου υπηρεσίας και εκχώριση τιμών που ανακτήθηκαν από τον πίνακα της βάσης δεδομέων

```
		x = new ServiceClass(cursor.getLong(0), cursor.getString(1),
		cursor.getString(2), cursor.getLong(3), cursor.getDouble(4),
		cursor.getString(5), cursor.getString(6), cursor.getLong(7),
		cursor.getString(8));
		db.close();
		return x;
	} catch (NumberFormatException e) {
		Log.e(tag, "getClient: " + e.getMessage());
		x = null;
		db.close();
```

Επιστροφή του αντικειμένου τύπου ServiceClass

```
		return x;
	}
}
```

Λήψη υπηρεσιών συγκεκριμένου οχήματος και περιορισμένου πλήθους εγγραφών. Χρήσιμη μέθοδος για να μην φορτώνονται όλες οι υπηρεσίες ενός οχήματος στο γραφικό περιβάλον της εφαρμογής.

```
// Getting Top N services
public List<ServiceClass> getTopServices(int howMany, long carId) {
	List<ServiceClass> xList = null;
	SQLiteDatabase db = this.getWritableDatabase();
	try {
		xList = new ArrayList<ServiceClass>();
		Δημιουργία ερωτήματος ανάκτησης δεδομένων υπηρεσιών
		συγκεκριμένου οχήματος και μέγισοτυ πλήθους εγγραφών.
		// Select All Query
		String selectQuery = "SELECT  * FROM " + serviceTable + "
		WHERE " + SER_carId + "=" + carId + " ORDER BY " +
		SER_dateRev
		+ " LIMIT " + howMany;
		Cursor cursor = db.rawQuery(selectQuery, null);

		Για όλες τις εγγραφές δημιούργησε αντικέμενα υπηρεσιών και
		πρόσθεσε τα σε λίστα αντικειμένων τύπου ServiceClass
		// looping through all rows and adding to list
		if (cursor.moveToFirst()) {
			do {
				ServiceClass x = new
				ServiceClass(cursor.getLong(0),
				cursor.getString(1), cursor.getString(2),
				cursor.getLong(3),cursor.getDouble(4),
				cursor.getString(5), cursor.getString(6),
				cursor.getLong(7), cursor.getString(8));

				xList.add(x);
			} while (cursor.moveToNext());
		}
		db.close();

		Επιστροφή της λίστας αποτελεσμάτων
		return xList;

	} catch (NumberFormatException e) {
		Log.e(tag, "getAllClients: " + e.getMessage());
		xList = null;
		db.close();
		return xList;
	}
}
```

Ανάκτηση από την βάση δεδομένων και επιστροφή όλων των υπηρεσιών συγκεκριμένου οχήματος

```
// Getting All services
public List<ServiceClass> getAllServices(long carId) {
	List<ServiceClass> xList = null;
	SQLiteDatabase db = this.getWritableDatabase();
```

Όμοια με της άλλες μεθόδους ανάκτησης πολλαπλών εγραφών από πίνακα της βάσης δεδομένων. Δημιουργία κατάλληλου ερωτήματος, εκτέλεση, ανάκτηση δεδομένων και δημιουργία λίστας αντικειμένων.

```
        try {
                xList = new ArrayList<ServiceClass>();
                // Select All Query
                String selectQuery = "SELECT  * FROM " + serviceTable + "
                WHERE " + SER_carId + "=" + carId + " ORDER BY " +
                SER_dateRev;
                Cursor cursor = db.rawQuery(selectQuery, null);

        // looping through all rows and adding to list
        if (cursor.moveToFirst()) {
        do {
                ServiceClass x = new ServiceClass(cursor.getLong(0),
                cursor.getString(1), cursor.getString(2), cursor.getLong(3),
                cursor.getDouble(4), cursor.getString(5),
                cursor.getString(6), cursor.getLong(7),
                cursor.getString(8));
        xList.add(x);
        } while (cursor.moveToNext());
}
db.close();
Log.i(tag, "Found " + xList.size() + " SERVICE entries for carid=" +
carId);
        return xList;

        } catch (NumberFormatException e) {
                Log.e(tag, "getAllClients: " + e.getMessage());
                xList = null;
                db.close();
                return xList;
        }
}
```

Διαγραφή υπηρεσίας οχήματος από την βάση δεδομένων

```
// Deleting single service
public boolean deleteService(ServiceClass x) {
        SQLiteDatabase db = this.getWritableDatabase();
        try {
        db.delete(serviceTable, SER_id + " = ?", new String[] {
        String.valueOf(x.getId()) });
                db.close();
                return true;
        } catch (Exception e) {
                Log.e(tag, "deleteClient: " + e.getMessage());
                db.close();
                return false;
        }
}
```

Εύρεση του πλήθους υπηρεσιών συγκεκριμένου οχήματος

```
// Getting service Count
public int getServiceCount(long carId) {


	String countQuery = "SELECT  * FROM " + serviceTable + " WHERE " +
	SER_carId + "=" + carId;
	SQLiteDatabase db = this.getReadableDatabase();
	Cursor cursor = db.rawQuery(countQuery, null);
	int count = cursor.getCount();
	cursor.close();
	db.close();
	// return count
	return count;
}
```

Εύρεση του πλήθους υπηρεσιών όλων των οχημάτων του χρήστη. Χρήσιμο για στατιστικά στοιχεία.

```
// Getting service Count
public int getServiceCount() {
	String countQuery = "SELECT  * FROM " + serviceTable;
	SQLiteDatabase db = this.getReadableDatabase();
	Cursor cursor = db.rawQuery(countQuery, null);
	int count = cursor.getCount();
	cursor.close();
	db.close();
	// return count
	return count;
}


/* -----------------| | FUEL TABLE CRUD | |------------------ **/
```

Προσθήκη καυσίμου στην βάση δεδομένων. Το αντικείμενο-παράμετρος που δέχεται η μέθοδος περιέχει το id του οχήματος για το οποίο αναφέρετε η πετρέλευση αυτή και επομένως δεν χρειάζεται να περάστει επιπλέον παράμετρος το id του οχήματος.

```
// Adding new fuel
public boolean addFuel(FuelClass x) {
	SQLiteDatabase db = this.getWritableDatabase();
	try {
		ContentValues values = new ContentValues();
```

Οι λειτουργίες είναι παρόμοιες με αυτές των προηγούμενων μεθόδων που σηζητήθηκαν.

```
		values.put(FUE_date, x.getDate());
		values.put(FUE_time, x.getTime());
		values.put(FUE_kms, x.getKms());
		values.put(FUE_price_gasoline, x.getPriceGasoline());
		values.put(FUE_price_gas, x.getPriceGas());
		values.put(FUE_price_diesel, x.getPriceDiesel());
		values.put(FUE_amountPaid, x.getAmountPaid());
		values.put(FUE_type, x.getFuelType());
		values.put(FUE_carId, x.getCarId());
```

```
                String[] dateArray = x.getDate().split("/");
                values.put(FUE_dateRev, dateArray[2] + "/" + dateArray[1] +
                "/" + dateArray[0]);

                db.replace(fuelTable, null, values);
                db.close(); // Closing database connection
                Log.i(tag, "Successfuly added fuel");
                return true;
        } catch (Exception e) {
                Log.e(tag, "addFuel: " + e.getMessage());
                db.close();
                return false;
        }
}
```

Ενημέρωση τιμών καυσίμου συγκεκριμένης εγγραφής. Χρήσιμη μέθοδος για διόρθωση τιμών από τον χρήστη.

```
// Updating single fuel
public boolean updateFuel(FuelClass x) {
        SQLiteDatabase db = this.getWritableDatabase();
        try {
                ContentValues values = new ContentValues();

                values.put(FUE_date, x.getDate());
                values.put(FUE_time, x.getTime());
                values.put(FUE_kms, x.getKms());
                values.put(FUE_price_gasoline, x.getPriceGasoline());
                values.put(FUE_price_gas, x.getPriceGas());
                values.put(FUE_price_diesel, x.getPriceDiesel());
                values.put(FUE_amountPaid, x.getAmountPaid());
                values.put(FUE_type, x.getFuelType());
                values.put(FUE_carId, x.getCarId());

                String[] dateArray = x.getDate().split("/");
                values.put(FUE_dateRev, dateArray[2] + "/" + dateArray[1] +
                "/" + dateArray[0]);
```

Ενημέρωση τιμών συγκεκριμένης πετρέλευσης με id το id οτυ αντικειμένου x (x.getId())

```
                // Updating row
                db.update(fuelTable, values, FUE_id + " = ?", new String[] {
                String.valueOf(x.getId()) });
                db.close();
                Log.i(tag, "updatingFuel ID=" + x.getId());
                return true;
        } catch (Exception e) {
                Log.e(tag, "updateFuel: " + e.getMessage());
                db.close();
                return false;
        }

}
```

Λήψη πετρέλευσης συγκεκριμένου id

```
// Getting single fuel
```

```
FuelClass getFuel(int id) {
	FuelClass x = null;
	SQLiteDatabase db = this.getReadableDatabase();
	try {
		Ανάκτηση δεδομένων από την βάση
		Cursor cursor = db.query(fuelTable, new String[] { FUE_id,
		FUE_date, FUE_time, FUE_kms, FUE_price_gasoline,
		FUE_price_gas,
		FUE_price_diesel, FUE_amountPaid, FUE_type, FUE_carId,
		FUE_dateRev }, FUE_id + "=?",
	new String[] { String.valueOf(id) }, null, null, null, null);
		if (cursor != null)
			cursor.moveToFirst();

		Δημιουργία και επιστροφή αντικειμένου
		x = new FuelClass(cursor.getLong(0), cursor.getString(1),
		cursor.getString(2), cursor.getLong(3), cursor.getDouble(4),
		cursor.getDouble(5), cursor.getDouble(6),
		cursor.getDouble(7), cursor.getInt(8), cursor.getLong(9),
		cursor.getString(10));
		db.close();
		return x;
	} catch (NumberFormatException e) {
		Log.e(tag, "getfuel: " + e.getMessage());
		x = null;
		db.close();
		return x;
	}
}
```

Οι μεθόδοι που ακολουθούν έχουν παρόμοια λειτουργικότητα με αυτές που σηζητήθηκαν παραπάνω οπότε αναφέρονται επιγραμματικά. Ουσιαστικά τα μόνα χαρακτηστικά που διαφέρουν είναι οι τύποι παραμέτρων τα ερωτήματα όσον αναφορά τα ονόματα των πινάκων και των τιμών επιστροφής.

```
// Getting Top N fuels
public List<FuelClass> getTopFuels(int howMany, long carId)

// Getting Top N fuels
public List<Double> getLastFuelPrices(long carId)

// Getting All fuels
public List<FuelClass> getAllFuels(long carId)

// Deleting single fuel
public boolean deleteFuel(FuelClass x)

// Getting fuel Count
public int getFuelCount(long carId

// Getting fuel Count
public int getFuelCount()

// Adding new Route
public long addRoute(DBRoute x)

// Updating single Route
```

```
    public boolean updateRoute(DBRoute x)

    // Getting single Route
    public DBRoute getRoute(int id)

    // Getting All Routes
    public List<DBRoute> getAllRoutes()

    // Deleting single Route
    public boolean deleteRoute(DBRoute x)

    // Getting Route Count
    public int getRouteCount()

    // Adding new RouteRecords
    public boolean addRouteRecords(DBRouteRecords x)

    // Updating single RouteRecords
    public boolean updateRouteRecords(DBRouteRecords x)

    // Getting single RouteRecords
    DBRouteRecords getRouteRecords(int id)


    // Getting All RouteRecords for a given Route
    public List<DBRouteRecords> getAllRouteRecordsForRoute(long routeId)


    // Getting All RouteRecords for a given Car
    public List<DBRouteRecords> getAllRouteRecordsForCar(long carId)

    // Deleting single RouteRecords
    public boolean deleteRouteRecords(DBRouteRecords x)

    // Getting RouteRecords Count for a given route
    public int getRouteRecordsCount(long routeId)

    // Getting RouteRecords Count for a given route -> Inside Call
    public int getRouteRecordsCount(long routeId, SQLiteDatabase db

    // Getting RouteRecords Count
    public int getRouteRecordsCount()

    // Getting RouteRecords Count -> Inside Call
    public int getRouteRecordsCount(SQLiteDatabase db)

}
```

3.5 StartFragment.java (πακέτου **vehiclemanager.fuels.)**

Η κλάση αυτή είναι χρήσιμη για την διακπεραίωση των δεδομένων των οχημάτων που επιλέγει ο χρήστης.

```
package gr.topalidis.android.vehiclemanager.fuels;
```

Όπως φαίνεται παρακάτω χρησιμοποιούνται πολλές κλάσεις ως βιβλιοθήκες μιας και η κλάση αυτή αλληλεπιδρά με φόρμες διαλόγου, άλλα γραφικά συστατικά καθώς και άλληλεπίδραση με βάση δεδομένων.

Χρήση άλλων κλάσεων του προγραμματιστή

```
import gr.topalidis.android.vehiclemanager.DatabaseHandler;
import gr.topalidis.android.vehiclemanager.MainActivity;
import gr.topalidis.android.vehiclemanager.R;
import gr.topalidis.android.vehiclemanager.Utils;
import gr.topalidis.android.vehiclemanager.adapters.CarAdapter;
import gr.topalidis.android.vehiclemanager.pojos.CarClass;
import gr.topalidis.android.vehiclemanager.pojos.FuelClass;
import gr.topalidis.android.vehiclemanager.pojos.ServiceClass;
import gr.topalidis.android.vehiclemanager.pojos.UnionClass;
```

Χρήση κλάσεων γραφικής διασύνδεσης της java

```
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Collections;
import java.util.GregorianCalendar;
import java.util.List;
import java.util.Locale;
```

Χρήση κλάσεων java άλλων κατασκευαστών

```
import android.annotation.SuppressLint;
import android.app.Dialog;
import android.app.ProgressDialog;
import android.content.Context;
import android.graphics.Color;
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentActivity;


import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentStatePagerAdapter;
import android.support.v4.view.PagerAdapter;
import android.support.v4.view.ViewPager;
import android.util.Log;
import android.view.Gravity;
import android.view.Menu;
import android.view.View;

import android.view.View.OnClickListener;

import android.view.ViewGroup.LayoutParams;
import android.view.Window;
import android.view.WindowManager;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.TextView;
```

```
import android.widget.TimePicker;
```

Υλοποίηση της τάξης StartFragment, των ιδιοτήτων και των μεθόδων της.

```
public class StartFragment extends FragmentActivity {
```

Διάφορες ιδιότητες - παραμετροποιήσεις γραφικών συστατικών και ιδιοτήτων της κλάσης και των μεθόδων της.

```
        public static final String tag = "Debug StartFragment";
        private static final int NUM_PAGES = 4;
        static ProgressDialog pd;

        public static List<UnionClass> unionData;
        public static List<FuelClass> fuelData;
        public static List<ServiceClass> serviceData;
        static ServiceClass lastService;
        static FuelClass lastFuel;

        private static ViewPager mPager;
        private static PagerAdapter mPagerAdapter;
        Button btn_service, btn_fuel;//, btn_stats;
        public static DecimalFormat dateFormat = new DecimalFormat("00");
        public static CarClass selectedCar;
        DecimalFormat df = new DecimalFormat("#.###");
        DecimalFormat df2 = new DecimalFormat("#.##");
```

Κατά την εκκίνηση του activity θα εκτελεστεί ο παρακάτω κώδικας ο οποίος περιλαμβάνει αρχικοποιήσεις τιμών.

```
        @Override
        protected void onCreate(Bundle savedInstanceState) {
```

Κλήση της default onCreate της υπερτάξης

```
            super.onCreate(savedInstanceState);
```

Δήλωση οτι στην οθόνη και για την συγκεκριμένη activity δεν θα εμφανιστεί τίτλος. Έπειτα δηλώνονται και άλλες ρυθμίσεις εμφάνισης περιεχομένου.

```
            requestWindowFeature(Window.FEATURE_NO_TITLE);
            getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN);
            setContentView(R.layout.activity_start_fragment);

            // ------> Var Init
```

Λήψη δεδομένων που έχουν οριστεί από τον προγραμματιστή. Αυτή η μέθοδος χρησιμοποιείται για την επικοινωνία και ανταλλαγή δεδομένων ανάμεσα σε δύο ή παραπάνω activities. Στην συγκεκριμένη περίπτωση χρησιμοποιείται για να ανακτηθεί το όχημα που έχει ήδη επιλέξει ο χρήστης.

```
            Bundle b = getIntent().getExtras();
            selectedCar = b.getParcelable(CarAdapter.TAG_SELECTED_CAR);
```

```
        Log.i("Debug", "BundleHead: " + selectedCar.getModel());
```

Κλήση συνάρτησης αρχικοποίησης των πρετρελεύσεων που έχουν ήδη εισαχθεί για το συγκεκριμένο όχημα.

```
        initializeData(StartFragment.this);
```

Αρχικοποίηση διάφορων γραφικών συστατικών

```
        // ------>  Instantiate a ViewPager and a PagerAdapter.
        mPager = (ViewPager) findViewById(R.id.pager);
        mPagerAdapter = new
        ScreenSlidePagerAdapter(getSupportFragmentManager());
        mPager.setAdapter(mPagerAdapter);
        mPager.setOffscreenPageLimit(NUM_PAGES - 1);
        //mPager.setPageTransformer(true, new ZoomOutPageTransformer());

        // ------> The bottom buttons
        btn_service = (Button) findViewById(R.id.btn_main_service);
        btn_fuel = (Button) findViewById(R.id.btn_main_fuel);
        //btn_stats = (Button) findViewById(R.id.btn_main_stats);
```

Listener για το γεγονός click του κουμπιού «service» σε αυτή την περίπτωση θα εμφανιστεί φόρμα διαλόγου για καταχώρηση κάποιου service του οχήματος.

```
        btn_service.setOnClickListener(new OnClickListener() {
              @Override
              public void onClick(View v) {
                    showServiceDialog();
              }
        });
```

Listener για το γεγονός click του κουμπιού «καύσιμα» όμοια με το κουμπί «service» θα εμφανιστεί φόρμα διαλόγου για νέα κίνηση καυσίμου.

```
        btn_fuel.setOnClickListener(new OnClickListener() {
              @Override
              public void onClick(View v) {
                    showFuelDialog();
              }
        });

}
```

Σε οποιαδίποτε περίπτωση που σταματήσει η λειτουργία του τρέχων activity να γίνει παύση και απόκρυψη της progressive bar όταν και έχει εμφανιστεί ήδη.

```
@Override
public void onPause() {
      super.onPause();
```

```
		if (pd != null && pd.isShowing())
			pd.dismiss();
	}
```

Ορισμός κλάσης ασύγχρονης διεργασίας που μπορεί να τρέξει στο background της εφαρμογής. Χρήσιμη για την εκτέλεση ενεργειών χωρίς να σταματήσει να εκτελείται κάποια άλλη ενέργεια. (multithreading)

```
	public static class reloadUITask extends AsyncTask<Void, Void, Void> {
		int viewPagerPosition;
		Context ctx;

		public reloadUITask(Context ctx) {
			this.ctx = ctx;
		}

		Εμφάνιση μηνύματος κατά την εκτέλεση της διεργασίας
		@Override
		protected void onPreExecute() {
			viewPagerPosition = mPager.getCurrentItem();
			pd = new ProgressDialog(ctx);
			pd.setTitle("Ενημέρωση");
			pd.setMessage("Υπολογιμός νέων δεδομένων...");
			pd.setCancelable(false);
			pd.show();
		}
```

Εκτέλεση της κύριας εργασίας. Αυτή μπορεί να εκτελείται ενώ το μήνυμα να συνεχίζει να εμφανίζεται στον χρήστη

```
		@Override
		protected Void doInBackground(Void... params) {
			initializeData(ctx);
			return null;
		}
```

Μόλις ολοκληρωθεί η διεργασία ενημέρωσε και τα άλλα χειριστήρια

```
		@Override
		protected void onPostExecute(Void result) {
			pd.dismiss();
			mPagerAdapter.notifyDataSetChanged();
			try {
				Thread.sleep(100);
			} catch (InterruptedException e) {
				Log.e(tag, e.getMessage());
			}

			if (viewPagerPosition >= 0)
				mPager.setCurrentItem(viewPagerPosition);
		}

	}
```

Εδώ επιτυγχάνεται η ίδια διαδικασία χωρίς να επιρεάζει της άλλες λειτουργίες της εφαρμογής. Με αυτό τον τρόπο ο χρήστης μπορεί να

αλληλεπιδράσει με άλλα χειρηστήρια και λειτουργίες χωρίς απαραίτητα να έχει ολοκληρωθεί αυτή η λειτουργία.

```
public static void initializeData(Context ctx) {

        Χρήση της τοπικής βάσης δεδομένων
        DatabaseHandler dbh = new DatabaseHandler(ctx,
        DatabaseHandler.DB_VERSION);

        Κλήση συνάρτησης και ανάκτηση όλων των κινήσεων καυσίμου για
        το επιλεγμένο όχημα

        fuelData = dbh.getAllFuels(StartFragment.selectedCar.getId());

        Όμοια και με τα services του οχήματος

        serviceData =
        dbh.getAllServices(StartFragment.selectedCar.getId());
        dbh.close();

        Τοποθέτηση όλων των ενεργειών κινήσεων καυσίμων αλλα και
        services σε μία λίστα. Για να είναι συμβατό το merging (UnionClass)
        Θα πρέπει ο τύπος δεδομένων να είναι συμβατός και για το
        FuelClass αλλά και για το ServiceClass. Για αυτό το λόγο
        υλοποιήθηκε η τάξη UnionClass

        unionData = new ArrayList<UnionClass>(getAllUnionData());

        Ταξινόμηση των πεπλεγμένων στοιχείων (Fuel and Service)
        σύμφωνα με την ημερομηνία της δημιουργίας τους.

        Collections.sort(unionData, UnionClass.dateCompRev);

        Εύρεση των τελευταίων ανα τύπο υπηρεσιών
        setLastData();
}
```

Ενοποίηση όλων των τύπων (καύσιμα - services) ενεργειών του οχήματος, σε μία λίστα. Η ενοποίηση γίνεται σειριακά, πρώτα τα κάυσιμα και έπειτα τα services

```
private static List<UnionClass> getAllUnionData() {
        List<UnionClass> unionData = new ArrayList<UnionClass>();

        Τοποθέτηση  κάθε καυσίμου στην λίστα
        for (FuelClass x : fuelData)

        unionData.add(new UnionClass(x, null, true));
        Έπειτα τοποθέτηση  κάθε service στην λίστα

        for (ServiceClass x : serviceData)
                unionData.add(new UnionClass(null, x, false));
```

Όπως φαίνεται παραπάνω η σειρά με την οποία τοποθετούνται οι τιμές στον constructor καθορίζει και το είδος της ενέργειας. Ως πρώτη παράμετρος θεωρείται το καύσιμο, δεύτερη το service και τρίτη παράμετρος αν πρόκειται για καύσιμο ή service. Η διαδικασία θα μπορούσε να γίνει πιο δομημένη (ώστε η UnionClass να ανιχνεύει αν το αντικείμενο είναι fuel ή service) αλλά προτιμήθηκε πιο απλή λύση

```
		return unionData;
}
```

Εύρεση της τελευταίας (πρόσφατης) κίνησης καυσίμου και service στην ενιαία λίστα

```
private static void setLastData() {
	boolean fuelSet = false, serviceSet = false;
	lastFuel = null;
	lastService = null;

	Για όλα τα αντικείμενα που έχουν τοποθετηθεί στην unionData
	for (UnionClass x : unionData) {
		Εάν είναι καύσιμο και δεν έχει οριστεί ακόμα
		τρέχον καύσιμο όρισε το ως το πιο πρόσφατο.

		if (x.isFuel() && !fuelSet) {
			lastFuel = x.getFc();
			fuelSet = true;

		Διαφορετικά αν είναι υπηρεσία και δεν έχει οριστεί
		ακόμα τρέχον υπηρεσία όρισε την ως την πιο
		πρόσφατη.

		} else if (!x.isFuel() && !serviceSet) {
			lastService = x.getSc();
			serviceSet = true;
		}

		Αν έχουν οριστεί και service και καύσιμο μην
		ψάξεις περαιτέρω

		if (fuelSet && serviceSet)
			break;
	}

	Ο παραπάνω αλγόριθμος αναζητά και εκχωρεί σε δύο
	μεταβλητές τα πρώτα αντικείμενα τύπου καυσίμου και
	service που θα βρεί στην λίστα. Λόγω της ταξινόμησης

	τους ημερομηνιακά από τα πιο πρόσφατα προς τα
	παλαιότερα τα πρώτα είναι και τα τελευταία που
	καταχωρήθηκαν ημερομηνιακά.
	}
```

```
        Αποθήκευση service οχήματος στην βάση δεδομένων

        private void saveService(String date, String time, long kms,
        double price, String reason, String comments) {
                    Επικοινωνία με την βάση δεδομένων
                    DatabaseHandler dbh = new
                    DatabaseHandler(StartFragment.this,
                    DatabaseHandler.DB_VERSION);

                    Δημιουργία στιγμιότυπου ServiceClass ώστε να
                    περαστεί ως παράμετρος σε μέθοδο της
                    DatabaseHandler

                    ServiceClass x = new ServiceClass(date, time, kms,
                    price, reason, comments, selectedCar.getId());

                    Η addService έχει αναφερθεί παραπάνω.
                    Λαμβάνει τα δεδομένα το αντικειμένου x τα
                    μετατρέπει σε εντολή sql ώστε να
                    αποθυκευτούν στην βάση δεδομένων.

                    dbh.addService(x);
                    dbh.close();
                }
        });

    Εμφάνιση φόρμας διαλόγου καταχώρησης καυσίμου.

    @SuppressLint("InlinedApi")
    private void showFuelDialog() {

        Δημιουργία διαλόγου με ορισμένα χαρακτηριστικά
        final Dialog dialog = new Dialog(StartFragment.this,
        android.R.style.Theme_Translucent_NoTitleBar_Fullscreen);
        dialog.requestWindowFeature(Window.FEATURE_LEFT_ICON);
        dialog.getWindow().setGravity(Gravity.CENTER);

        dialog.getWindow().setLayout(LinearLayout.LayoutParams.MATCH_PAREN
        T, LinearLayout.LayoutParams.MATCH_PARENT);

        Ορισμός του layout της οθόνης
        dialog.setContentView(R.layout.dialog_fuel);

        Έλεγχος των πλήκτρων ώστε να δημιουργηθούν events
        final Button btn_fuel_date = (Button)
        dialog.findViewById(R.id.btn_fuel_date);
        final Button btn_fuel_time = (Button)
        dialog.findViewById(R.id.btn_fuel_time);



        Δημιουργία ημερολογίου για ανάκτηση τρέχουσας
        ημερομηνίας και ώρας
```

```
GregorianCalendar cal = new
GregorianCalendar(Locale.getDefault());

Ορισμός κειμένου της τρέχουσας ημερομηνίας του
συστήματος
btn_fuel_date.setText(formatDate(cal.get(GregorianCalendar.DAY_OF_
MONTH), cal.get(GregorianCalendar.MONTH) + 1,
cal.get(GregorianCalendar.YEAR)));

Ορισμός κειμένου της τρέχουσας ώρας του συστήματος
btn_fuel_time.setText(formatTime(cal.get(GregorianCalendar.HOUR_OF
_DAY), cal.get(GregorianCalendar.MINUTE)));


Listener για το γεγονός click. Όταν ο χρήστης κάνει
κλικ στην ημερομηνία, θα εμφανιστεί φόρμα διαλόγου με
ημερομηνίες για να επιλέξει ο χρήστης διαφορετική
ημερομηνία.
btn_fuel_date.setOnClickListener(new OnClickListener() {
	@Override
	public void onClick(View v) {
		Κλήση της μεθόδου που εμφανίζει την φόρμα
		διαλόγου με ημερομηνίες. Η μέθοδος αυτή
		δέχεται παράμετρο το χειρηστήριο ώστε να
		μπορεί να τροποποιήσει την ημερομηνία
		αναλόγως με αυτή που έχει επιλέξει ο
		χρήστης.

		showDateDialog(btn_fuel_date);
	}
});

Όμοια με τον listener της ημερομηνίας. Η διαφορά είναι
ότι γίνεται επιλογή της ώρας αντί της ημερομηνίας.
btn_fuel_time.setOnClickListener(new OnClickListener() {
	@Override
	public void onClick(View v) {
		showTimeDialog(btn_fuel_time);
	}
});

Ορισμός αντικειμένων ώστε να είναι δυνατή η πρόσβαση
στα gui-χειρηστήρια της εφγαρμογής

final EditText et_kms = (EditText)
dialog.findViewById(R.id.et_fuel_kms);
final EditText et_gasoline_price = (EditText)
dialog.findViewById(R.id.et_gasoline_price);
final EditText et_gasoline_amountPaid = (EditText)
dialog.findViewById(R.id.et_gasoline_amountPaid);
final EditText et_gas_price = (EditText)
dialog.findViewById(R.id.et_gas_price);
```

```
final EditText et_gas_amountPaid = (EditText)
dialog.findViewById(R.id.et_gas_amountPaid);
final EditText et_diesel_price = (EditText)
dialog.findViewById(R.id.et_diesel_price);
final EditText et_diesel_amountPaid = (EditText)
dialog.findViewById(R.id.et_diesel_amountPaid);
TextView tv_last_fuel_kms = (TextView)
dialog.findViewById(R.id.tv_last_fuel_kms);
TextView tv_last_fuel_amount_paid = (TextView)
dialog.findViewById(R.id.tv_last_fuel_amount_paid);
TextView tv_last_fuel_litres = (TextView)
dialog.findViewById(R.id.tv_last_fuel_litres);
```

Διάφορες εκχωρήσεις τιμών. Αν υπάρχει καταχωρημένη κίνηση καυσίμου, τα textbox θα ανακτήσουν τις τιμές της τελευταίας κίνησης. Η λειτουργία αυτή είναι χρήσιμη για να μην απαιτείται η επαναπληκτρολόγηση όλων των τιμών της κίνησης εφόσον οι τιμές αυτές δεν αλλάζουν συχνά (π.χ. τιμή καυσίμου, τελική τιμή κτλ). Προφανώς στις περισσότερες περιπτώσεις κάποιες από αυτές τις τιμές αλλάζουν (η τιμή καυσίμου δεν είναι σταθερή τόσο από πρατήριο σε πρατήριο όσο και από μέρα σε μέρα), ενώ η τελική τιμή συνήθως είναι σταθερή (ο χρήστης γεμίζει π.χ. με 20ευρώ σταθερα ανεξαρτήτου τιμής λίτρου). Η χρησιμότητα του εγκειται στο γεγονός ότι δεν αλλάζουν τοσο συχνά όλες οι ποσότητες ταυτόχρονα και ότι ο χρήστης βλέπει τις τιμές της τελευταίας κίνησης ώστε να μπορεί να τις συγκρίνει.

```
if (lastFuel != null) {
	tv_last_fuel_kms.setText(lastFuel.getKms() + " km");

tv_last_fuel_amount_paid.setText(df2.format(lastFuel.getAmountPaid
()) + " € (" + lastFuel.getPrice() + " €/lt)");

tv_last_fuel_litres.setText(df2.format(lastFuel.getAmountPaid() /
lastFuel.getPrice()) + " lt");
}
```

Μια άλλη διατύπωση της εφαρμογής που εμφανίζονται προτινόμενες τιμές στον χρήστη ανα τύπο καυσίμου.

```
//Get suggested fuel prices
DatabaseHandler dbh = new DatabaseHandler(StartFragment.this,
DatabaseHandler.DB_VERSION);
try {
	List<Double> suggestedFuelPricesList =
	dbh.getLastFuelPrices(selectedCar.getId());
	if (suggestedFuelPricesList != null &&
suggestedFuelPricesList.size() == 3) {
	et_gasoline_price.setText(String.valueOf(suggestedFuelPrices
List.get(0)));

et_gas_price.setText(String.valueOf(suggestedFuelPricesList.get(1)
));
```

```
et_diesel_price.setText(String.valueOf(suggestedFuelPricesList.get
(2)));
	}
} catch (Exception e) {
	Log.e(tag, e.getMessage());
} finally {
	dbh.close();
}
```

Καθορισμός χαρακτηριστικών των textbox αναλόγως της συμβατότητα διάφορων τύπων καυσίμου με το συγκεκριμένο όχημα του χρήστη. Για παράδειγμα αν ένας τύπος καυσίμου δεν είναι συμβατός με το όχημα τότε αυτό απενεργοποιείται και χρωματίζεται με ανοιχρό κόκκινο.

```
switch (selectedCar.getFuelType()) {
case 0:
	et_gas_amountPaid.setEnabled(false);
	et_gas_amountPaid.setHintTextColor(Color.parseColor("#ff6161
	61"));
	et_diesel_amountPaid.setEnabled(false);
	et_diesel_amountPaid.setHintTextColor(Color.parseColor("#ff6
	16161"));
	break;
case 1:
	et_diesel_amountPaid.setEnabled(false);
	et_diesel_amountPaid.setHintTextColor(Color.parseColor("#ff6
	16161"));
	break;
case 2:
	et_gas_amountPaid.setEnabled(false);
	et_gas_amountPaid.setHintTextColor(Color.parseColor("#ff6
	16161"));
	et_gasoline_amountPaid.setEnabled(false);

	et_gasoline_amountPaid.setHintTextColor(Color.parseColor("#f
	f616161"));
	break;
default:
	break;
}


Button btn_ok = (Button) dialog.findViewById(R.id.btn_ok);
Button btn_cancel = (Button) dialog.findViewById(R.id.btn_cancel);
```

Listener για το κουμπί «ok» ώστε να εκτελεστούν οι απαραίτητες ενέργειες για την αποθήκευση στην τοπική βάση δεδομένων

```
btn_ok.setOnClickListener(new OnClickListener() {
	@Override
	public void onClick(View v) {
```

```
Αναλόγως του τύπου καυσίμου που δέχεται το
επιλεγμένο όχημα αποθήκευονται τα αντίστοιχα
δεδομένα

if (checkInputFields(selectedCar.getFuelType())) {

      switch (selectedCar.getFuelType()) {
      case 0:

      Κλήση μεθόδου αποθήκευσης κίνησης
      πετρελαίου, στην βάση αποθηκεύονται η
      ημερομηνία, η ώρα, τα τρέχων χιλιόμετρα
      του οχήματος, η τιμή λίτρου, η τελική
      τιμή που πληρώθηκε για την συνολική
      ποσότητα καθώς και ο τύπος του
      καυσίμου.

      saveGasolineOrDiesel(btn_fuel_date.getText().to
      String(), btn_fuel_time.getText().toString(),
     Long.parseLong(et_kms.getText().toString()),
     Double.parseDouble(et_gasoline_price.getText().
     toString()),
     Double.parseDouble(et_gas_price.getText().toStr
     ing()),Double.parseDouble(et_diesel_price.getTe
     xt().toString()),
      Double.parseDouble(et_gasoline_amountPaid.getTe
     xt().toString()), selectedCar.getFuelType());
            break;
      case 1:

      Ομοίως για το καυσίμο αερίου

      saveGas(btn_fuel_date.getText().toString(),
      btn_fuel_time.getText().toString(),
      Long.parseLong(et_kms.getText().toString()),
      Double.parseDouble(et_gasoline_price.getText().
      toString()), Double
            .parseDouble(et_gasoline_amountPaid.getT
      ext().toString().equals("") ? "0" :
      et_gasoline_amountPaid.getText().toString()),
      Double.parseDouble(et_gas_price.getText().toStr
      ing()), Double
            .parseDouble(et_gas_amountPaid.getText()
      .toString().equals("") ? "0" :
      et_gas_amountPaid.getText().toString()),
      Double.parseDouble(et_diesel_price.getText().to
      String()), selectedCar.getFuelType());
            break;

      case 2:
      Όμοια και για το καυσίμο βενζίνη. Η
      συνάρτηση αποθήκευσης είναι ίδια με
      εκείνη του πετρελαίου. Η διαφοροποίηση
      γίνεται μόνο στην τελευταία παράμετρο
```

```
                //που δηλώνεται ο τύπος του καυσίμου (0 ή 2)

                saveGasolineOrDiesel(btn_fuel_date.getText().toString(), btn_fuel_time.getText().toString(), Long.parseLong(et_kms.getText().toString()),

                Double.parseDouble(et_gasoline_price.getText().toString()),Double.parseDouble(et_gas_price.getText().toString()),Double.parseDouble(et_diesel_price.getText().toString()),Double.parseDouble(et_gasoline_amountPaid.getText().toString().equals("") ?
                et_diesel_amountPaid.getText().toString() :
                et_gasoline_amountPaid.getText().toString()),
                selectedCar.getFuelType());
                        break;
                default:
                        break;
                }

                //Η εισαγωγή των δεδομένων στην τοπική βάση έχει ολοκληρωθεί οπότε η φόρμα κλείνει και επαναφορτώνεται το τρέχον fragment.

                dialog.dismiss();
                new reloadUITask(StartFragment.this).execute();
        }
}

//Μέθοδος ελέγχου- επικύρωσης των δεδομένων κίνησης καυσίμου.
private boolean checkInputFields(int fuelId) {
        int errorCount = 0;
        String responseString = "Δεν έχετε εισάγει: ";

        //Η μέθοδος αυτή ελέγχει ένα προς ένα τα δεδομένα που έχει εισάγει ο χρήστης. Αναλόγως τα μη έγκυρα δεδομένα δημιουργεί και το αντίστοιχο μήνυμα. Το παραγόμενο μήνυμα σφάλματος θα περιέχει όλα τα επιμέρους μηνύματα σφάλματος για κάθε μη έγκυρο πεδίο.

        //Αν ο χρήστης δεν έχει εισάγει χιλιόμετρα συνένωσε το βασικό μήνυμα λάθους (Δεν έχετε εισάγει:) με το περιγραφικό «Χιλιόμετρα»

        if (et_kms.getText().toString().equals("")) {
                responseString += "Χιλιόμετρα";
                errorCount++;
        }
```

```
switch (selectedCar.getFuelType()) {

Αναλόγως τον τύπο του καυσίμου που
χρησιμοποιείται από το όχημα γίνεται και ο
αντίστοιχος έλεγχος

Τύπος καυσίμου – βενζίνη


case 0:

      Ο χρήστης δεν έχει ορίσει ποσό χρήματος
      if
      (et_gasoline_amountPaid.getText().toString().eq
      uals("")) {

      Αν δεν υπάρχει ήδη σφάλμα απλά πρόσθεσε
      το μήνυμα διαφορετικά διαχώρισε το
      δεύτερο (ή τρίτο μήνυμα) με κόμμα.

      if (errorCount == 0)
            responseString += "Ποσό € Βενζίνης";
            else
            responseString += ", Ποσό € Βενζίνης";
            errorCount++;
      }
      break;
case 1:
Ομοίως αν ο τύπος καυσίμου είναι αέριο ή
βενζίνη.
      if
      (et_gas_amountPaid.getText().toString().equals(
      "") &&
      et_gasoline_amountPaid.getText().toString().equ
      als("")) {
            if (errorCount == 0)
      responseString += "Ποσό € Αερίου ή Βενζίνης";
            else
      responseString += ", Ποσό € Αερίου ή Βενζίνης";
            errorCount++;
      }
      break;
case 2:
Ομοίως αν ο τύπος καυσίμου είναι πετρέλαιο.
      if
      (et_diesel_amountPaid.getText().toString().equa
      ls("")) {
            if (errorCount == 0)
            responseString += "Ποσό € Πετρελαίου";
            else
            responseString += ", Ποσό € Πετρελαίου";
            errorCount++;
      }
      break;
default:
```

```
                break;
        }

        Αντίστοιχα προστίθεται μήνυμα λάθους σε
        περίπτωση που ο χρήστης δεν έχει ορίσει τιμή
        καυσίμου.
        if
        (et_gasoline_price.getText().toString().equals("")) {
                if (errorCount == 0)
                        responseString += "Τιμή lt Βενζίνης";
                else

                        responseString += ", Τιμή lt Βενζίνης";
                errorCount++;
        }

        if (et_gas_price.getText().toString().equals("")) {
                if (errorCount == 0)
                        responseString += "Τιμή lt Αερίου";
                else
                        responseString += ", Τιμή lt Αερίου";
                errorCount++;
        }

        if (et_diesel_price.getText().toString().equals(""))
        {
                if (errorCount == 0)
                        responseString += "Τιμή lt Πετρελαίου";
                else
                responseString += ", Τιμή lt Πετρελαίου";
                errorCount++;
        }

        Εάν έχει βρεθεί τουλάχιστον ένα μη έγκυρο
        πεδίο εμφάνισε μήνυμα στον χρήστη και
        επέστρεψε false στην καλούσα μέθοδο ώστε να
        διαχειριστεί αντίστοιχα το γεγονός,
        διαφορετικά δεν εμφανίζεται μήνυμα λάθους
        και επιστρέφεται true.

        if (errorCount > 0)
        Utils.showToast(StartFragment.this, responseString);

        return (errorCount == 0 ? true : false);
}

Η παρακάτω μέθοδος δέχεται ως παραμέτρους τις
τιμές κίνησης που πρέπει να αποθηκεύσει στην
τοπική βάση δεδομένων
Τα αποιτούμενα δεδομένα όπως περιγράφηκαν παραπάνω
κατά την κλήση της είναι ημερομηνία,ώρα,τρέχον
χιλιόμετρα οχήματος,τιμή βενζίνης ή και αερίου ανα
μονάδα μέτρησης, συνολική ποσότητα και τύπος.
```

```
        private void saveGasolineOrDiesel(String date, String time,
        long kms, double priceGasoline, double priceGas,
        double priceDiesel, double amountPaid, int fuelType) {

              Δημιουργία χειρισμού βάσης δεδομένων
              DatabaseHandler dbh = new
              DatabaseHandler(StartFragment.this,
              DatabaseHandler.DB_VERSION);

              Δημιουργία αντικειμένου - καύσιμο
              FuelClass x = new FuelClass(date, time, kms,
              priceGasoline, priceGas, priceDiesel, amountPaid,
              fuelType, selectedCar.getId());


              Κλήση προσθήκης του παραπάνω αντικειμένου-
              καυσίμου στην βάση μέσω της addFuel που
              περιγράφηκε παραπάνω
              dbh.addFuel(x);
              dbh.close();
        }

      Προσθήκη Listener χειρισμού onclick στο κουμπί «cancel»
      Στην περίπτωση αυτή η φόρμα διαλόγου εισαγωγής κίνησης
      καυσίμου θα κλείσει και ο χρήστης θα μπορεί να επιλέξει
      από την αρχή μια από της βασικές δραστηριότητες του
      οχήματος. (η να επιλέξει άλλο όχημα)

      btn_cancel.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                  dialog.dismiss();
            }
      });



      LinearLayout ll_dialog_content = (LinearLayout)
      dialog.findViewById(R.id.tl_dialog_fuel);

      Εμφάνιση διαλόγου και κίνησης (εφφέ) οχήματος
      dialog.show();
      ll_dialog_content.startAnimation(MainActivity.slideInLeftDelay);
}

Μέθοδος εμφάνισης φόρμας διαλόγου επιλογής ημερομηνίας
private void showDateDialog(final Button tv) {

      Δημιουργία διαλόγου με οριζόμενο layout. Το layout έχει
      δημιουργηθεί όπως και όλα τα άλλα από τον κατασκευαστή
      της εφαρμογής.

      final Dialog dialog = new Dialog(StartFragment.this,
      android.R.style.Theme_DeviceDefault_Dialog);
      dialog.getWindow().setGravity(Gravity.CENTER);
```

```
        // dialog.getWindow().setLayout(800, 400);
        dialog.setTitle("ΕΠΙΛΕΞΤΕ ΗΜΕΡΟΜΗΝΙΑ");
        dialog.setContentView(R.layout.dialog_set_date);
        dialog.setCancelable(true);

        Χειρισμός των χειριστηρίων όπως το DatePicker κουμπιά
        κτλ
        final DatePicker d_dp = (DatePicker)
        dialog.findViewById(R.id.dialog_datePicker);
        Button d_ok_btn = (Button)
        dialog.findViewById(R.id.dialog_datePicker_ok_btn);
        Button d_cancel_btn = (Button)
        dialog.findViewById(R.id.dialog_datePicker_cancel_btn);




        Click Listener για τα κουμπιού ok και cancel. Στην
        περίπτωση του πλήκτρου «ok» η επιλεγμένη τιμή από το
        χειριστήριο της ημερομηνίας εκχωρείται στο textbox tv.
        d_ok_btn.setOnClickListener(new OnClickListener() {
              public void onClick(View arg0) {
                    tv.setText(getSelectedDate(d_dp));
                    dialog.dismiss();
              }
        });

        d_cancel_btn.setOnClickListener(new OnClickListener() {
              public void onClick(View arg0) {
                    dialog.dismiss();
              }
        });

        dialog.show();
}
```

Όμοια με την επιλογή ημερομηνίας, επιτυγχάνεται και η επιλογή της ώρας.

```
protected void showTimeDialog(final Button tv) {
        final Dialog dialog = new Dialog(StartFragment.this,
        android.R.style.Theme_DeviceDefault_Light_Panel);
        dialog.getWindow().setGravity(Gravity.CENTER);
        LayoutParams params = dialog.getWindow().getAttributes();
        params.width = LayoutParams.WRAP_CONTENT;

        dialog.getWindow().setAttributes((android.view.WindowManager.Layou
        tParams) params);
        // dialog.getWindow().setLayout(200, 300);
        dialog.setTitle("Επιλέξτε Ώρα");
        dialog.setContentView(R.layout.time_picker_dialog);
        dialog.setCancelable(true);

        Button ok = (Button)
        dialog.findViewById(R.id.dialog_timePicker_ok_btn);
```

```
		final TimePicker tp = (TimePicker)
		dialog.findViewById(R.id.dialog_timePicker);
		tp.setIs24HourView(true);
		Calendar cal = new GregorianCalendar();
		tp.setCurrentHour(cal.get(Calendar.HOUR_OF_DAY));

		ok.setOnClickListener(new OnClickListener() {
			public void onClick(View v) {

			String hourString = tp.getCurrentHour() < 10 ? "0" +
			tp.getCurrentHour() : tp.getCurrentHour() + "";
			String minString = tp.getCurrentMinute() < 10 ? "0" +
			tp.getCurrentMinute() : tp.getCurrentMinute() + "";
			tv.setText(hourString + ":" + minString);
				dialog.dismiss();
			}
		});

		dialog.show();
	}
```

Η μέθοδος αυτή δέχεται ένα χειριστήριο τύπου DatePicker και επιστρέφει την ημερομηνία που έχει εκχωρηθεί από τον χρήστη, σε συγκεκριμένο format.
Το format αυτό όπως φαίνεται είναι κείμενο της μορφής day/month/year

```
	public static String getSelectedDate(DatePicker _dp) {
		String date = "";
		int year = _dp.getYear();
		int month = _dp.getMonth() + 1;
		int day = _dp.getDayOfMonth();
		date = formatDate(day, month, year);
		return date;
	}
```

Επιστρέφει κείμενο της μορφής day/month/year σύμφωνα με της παραμέτρους που δέχεται

```
	public static String formatDate(int day, int month, int year) {
		return String.valueOf(dateFormat.format(day)) + "/" +
		String.valueOf(dateFormat.format(month)) + "/" +
		Integer.toString(year);
	}
```

Όμοια και με την ώρα, μορφής HH::min

```
	private String formatTime(int hour, int min) {
		return String.valueOf(dateFormat.format(hour)) + ":" +
		String.valueOf(dateFormat.format(min));
	}

}
```

ΚΕΦΑΛΑΙΟ 4

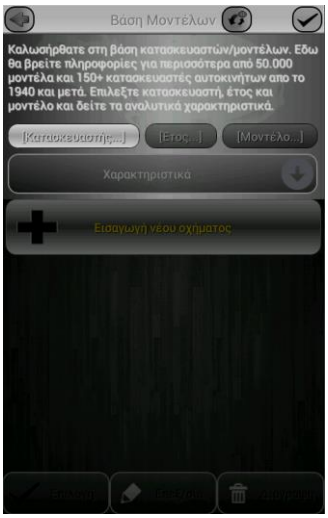
4.1 ΟΔΗΓΟΣ ΧΡΗΣΗΣ ΛΟΓΙΣΜΙΚΟΥ - Περιήγηση Εφαρμογής

Παρακάτω παρουσιάζεται με εικόνες η εφαρμογή και οι χρήσεις λειτουργίας της. Η κάθε εικόνα είναι και μία περίπτωση αρχικής ή ενδιάμεσης χρήσης λειτουργίας η οποία μπορεί να συσχετίζεται με προηγούμενες ενέργειες του χρήστη.

Στην διπλανή εικόνα φαίνεται η πρώτη οθόνη αλληλεπίδρασης με τον χρήστη. Το βασικό κουμπί προτρέπει τον χρήστη για εισαγωγή ενος οχήματος.

Τα πλήκτρα που βρίσκονται στο κάτω μέρος της οθόνης είναι απενεργοποιημένα μιας και δεν υπάρχει ακόμα κανένα επιλεγμένο όχημα.

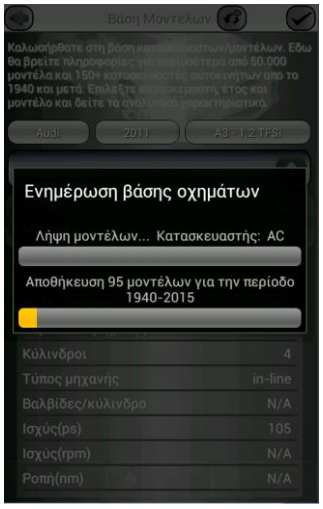
Κάνοντας κλικ στο κουμπί εισαγωγής νέου οχήματος εμφανίζεται η επόμενη οθόνη



Στις παραπάνω εικόνες φαίνονται οι περιπτώσεις χρήσης τις εφαρμογής κατά την εισαγωγή του πρώτου οχήματος. Εμφανίζεται μήνυμα υποδοχής κα έπειτα ο χρήστης (δεξιά εικόνα) μπορεί να επιλέξει τον κατασκευαστή του οχήματος του.

Στην διπλανή οθόνη εμφανίζονται οι επιλογές που έχει ο χρήστης κατά την εισαγωγή ή την τροποποίηση του οχήματος του, είτε πρόκειται για το βασικό όχημα ή για περισσότερα του ενός οχήματος. Μπορεί να επιλέξει αρχικά τον κατασκευαστή και το μοντέλο του αυτοκινήτου. Είναι χρήσιμη η επιλογή του κατασκευαστή ώστε να περιοριστεί η αναζήτηση των μοντέλων αυτοκινήτων τα οποία απαριθμούνται πάνω από 50000. Έπειτα ο χρήστης πληκτρολογεί την πινακίδα του οχήματος και επιλέγει έτος κατασκευής του αυτοκινήτου του. Στην συνέχεια επιλέγει ημερομηνία ασφάλειας και τον τύπο καυσίμου του οχήματος του. Πλέον είναι έτοιμος για την καταχώρηση των δεδομένων πατώντας το κουμπί οκ.

Συγχρονισμός – ενημέρωση δεδομένων



Στις παραπάνω εικόνες φαίνεται ο συγχρονισμός των δεδομένων. Οι επιλεγμένοι από τον χρήστη κατασκευαστές και όλα τα μοντέλα οχημάτων αυτών των κατασκευαστών θα συγχρονιστούν με τον server (download - updates) Η

διαδικασία αυτή μπορεί να διαρκέσει μερικά λεπτά αναλόγως με το πλήθος των μοντέλων αυτών που πρέπει να συγχρονιστούν και την ταχύτητα του δικτύου.

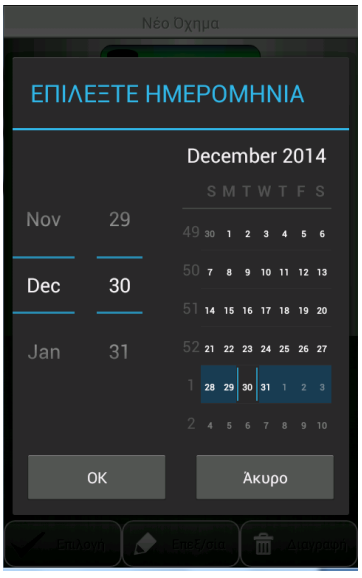
Όπως φαίνεται στις παραπάνω εικόνες η επιλογή της χρονολογίας έκδοσης κυκλοφορίας γίνεται με επιλογή απο λίστα. Η εισαγωγή θα μπορούσε να επιτευχθεί απο τον χρήστη πληκτρολογόντας την ημερομηνία. Όμως με αυτό τον τρόπο θα υπήρχε ενδεχόμενο να μην υπάρχει χρονολογία για το συγκεκριμένο όχημα. Προκειμένου να γίνει πιο εύκολο και ασφαλή για τον χρήστη με λιγότερες περιπτώσεις χρήστης, υλοποιήθηκε η επιλογή μέσω λίστας. Με την ίδια λογική πραγματοποιείται η επιλογή του μοντέλου οχήματος. Με αυτόν τον τρόπο αποφεύγεται η λανθασμένη πληκτρολόγηση, ορθογραφικά λάθη καθώς και πραγματοποιείται η ευκολότερη αναγνώριση την εγγραφής και της συσχέτισης του μοντέλου με την βάση δεδομένων.

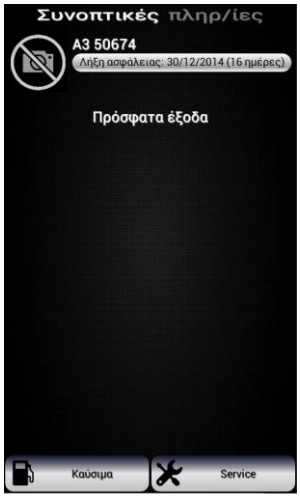
Μετά τις επιλογές του χρήστη η οθόνη του λίγο πριν την καταχώρηση θα φαίνεται όπως στην εικόνα δεξιά. Μετά την αποθήκευση του οχήματος μπορεί να πραγματοποιηθεί η επεξεργασία του επιλέγοντας το πάνελ του και πιέζοντάς το πλήκτρο επεξεργασία. Η οθόνη που θα εμφανιστεί θα είναι και πάλι η ίδια έχοντας πλέον την δυνατότητα αλλαγής χαρακτηριστικών του συγκεκριμένου οχήματος.

Έχοντας και πάλι επιλεγμένο το επιθυμητό όχημα μπορούμε να επιλέξουμε το κουμπί "κάυσιμα" (εικόνα δεξιά). Έπειτα θα εμφανιστεί η οθόνη της παρακάτω εικόνας.

Στην οθόνη αυτή ο χρήστης μπορεί επιλέξει ημερομηνία τις κίνησης γεμίσματος του ντεπόζιτου του οχήματος, αν αυτή ειναι διαφορετική απο την τρέχον ημερομηνία και να πληκτρολογίσει τα τρέχων χιλιόμετρα του οχήματος. Έπειτα αναλόγως το είδος των καυσίμων που μπορεί να δεχτεί το όχημα, ο χρήστης μπορεί να πληκτρολογήσει το ποσό που έδωσε για καύσιμο και την τιμή του αντίστοιχου καυσίμου ανα λίτρο. Μετα την πληκτρολόγηση ο χρήστης καταχωρεί την κίνηση.

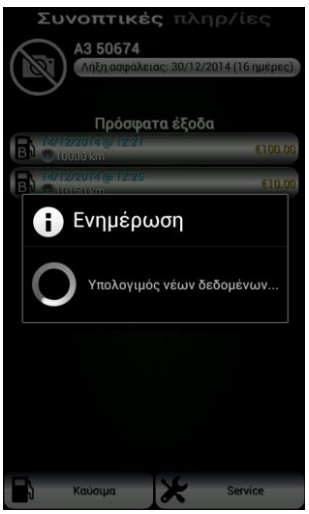
Περίπτωση χρήσης επιλογής διαφορετικής από την τρέχων ημερομηνία μέσω πλαίσιο διαλόγου.

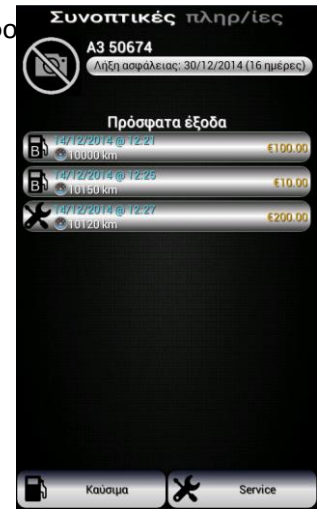




Οι παραπάνω εικόνες δείχνουν το πρίν και το μετά της καταγραφής κινήσεων καυσίμων

Εάν ο χρήστης αντι για το κουμπί καυσίμων πιέσει το πλήκτρο service θα του εμφανιστεί η οθόνη της δεξιάς εικόνας. Εδώ μπορεί να επιλέξει ημερομηνία, χιλιόμετρα, κόστος και είδος υπηρεσίας δίνοντας και μία μικρή περιγραφή για αυτή την υπηρεσία. Έπειτα καταχωρεί το γεγονός-service αυτό στην εφαρμογή



φο


Μετά την προσθήκη νεου service εμφανίζονται στον χρήστη οι παραπάνω εικόνες. Όπως φαίνεται ο χρήστης βλέπει ταυτόχρονα τις κινήσεις καυσίμων και services με ημερομηνιακή σειρά ωστε να έχει καλύτερη εποπτιά των εξόδων του οχήματος του

ΣΥΜΠΕΡΑΣΜΑΤΑ – ΕΠΙΛΟΓΟΣ

Οι ιδιαιτερότητες του σύγχρονου τρόπου ζωής και των απαιτήσεων που προκύπτουν, οδηγούν την τεχνολογία των υπολογιστών του web και της κινητής τηλεφωνίας δια μέσου του προγραμματισμού σε νέες βελτιωμένες εκδόσεις και τεχνολογίες. Άμεση συνέπεια των απαιτήσεων αυτών προκύπτει και η κατασκευή προγραμμάτων χρήσιμων για τον άνθρωπο.

Η σύλληψη της επι τόπου καταχώρησης κινήσεων καυσίμων και διάφορων εξόδων όπως service, ασφάλειες οχημάτων με εύκολο τρόπο και χρησιμοποιώντας υπάρχουσα για τον χρήστη συσκευή όπως είναι το κινητό τηλέφωνο, οδήγησε στην κατασκευή αυτής της εφαρμογής. Με βάση τις ιδιαίτερες ανάγκες των θεμάτων της δομής και λειτουργικότητας του, τέθηκαν στόχοι οι οποίοι καθόρισαν το σχεδιασμό και την τεχνολογία ανάπτυξης και υλοποίησης που ακολουθήθηκε.

Έγιναν προσπάθειες ώστε το λογισμικό που αναπτύχθηκε να έχει κύρια χαρακτηριστικά την φιλικότητα και την ευχρηστία, ενώ ταυτόχρονα να είναι προσιτό και ευκολονόητο στον χρήστη, γεγονός που πρέπει να αποδειχτεί με την χρήση και αξιολόγηση της εφαρμογής αυτής.

Λαμβάνοντας υπό όψη τα παραπάνω, ελπίζουμε ότι το λογισμικό «Τα Οχήματα μου – εξοδολόγια» θα αποτελέσει κίνητρο από καθηγητές και σπουδαστές για την επέκταση του και σε επαγγελματικό και πρακτικό επίπεδο.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Bruce Eckel (2006), Thinking in Java (4th Edition) Paperback – February 20, 2006

Jeff Friesen (2013) Learn Java for Android Development (2nd edition) February 20th 2013

Micheal Burton, Donn Felker (2012) Android Application Development For Dummies (2nd edition) October 23th 2012

Quentin Zervaas (2007) Practical Web 2.0 Applications with PHP

Samisa Abeysinghe (2008) RESTful PHP Web Services