ΤΕΙ ΚΡΗΤΗΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σχεδίαση και ανάπτυξη Android εφαρμογής για την οργάνωση των οικονομικών μικρών επιχειρήσεων

ΣΠΟΥΔΑΣΤΗΣ:

ΣΤΡΑΤΑΚΗΣ ΝΙΚΟΛΑΟΣ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:

ΒΙΔΑΚΗΣ ΝΙΚΟΛΑΟΣ

Κρήτη 2015

<<Δηλώνουμε υπεύθυνα ότι το παρόν κείμενο αποτελεί προϊόν προσωπικής μελέτης και εργασίας. Οι πηγές που χρησιμοποιήθηκαν αναφέρονται στην βιβλιογραφία.>>

Περίληψη

Στην παρούσα πτυχιακή εργασία περιγράφονται τεχνικές δημιουργίας εφαρμογών για smartphones με λειτουργικό σύστημα Android με χρήση της εφαρμογής Android Studio και χρησιμοποιώντας τα Android API's (Application Programming Interface), SDK (System Development Kit) και του πρωτοκόλλου REST.

Αρχικά θα δούμε γιατί επιλέξαμε το συγκεκριμένο λειτουργικό σύστημα και αργότερα θα δούμε τα ενδότερα στοιχεία τους. Ακολουθεί μια ιστορική αναδρομή σχετικά με τις εκδόσεις και τα χαρακτηριστικά του Android.

Σε δεύτερο ρόλο θα δούμε τι είναι και γιατί χρησιμοποιήσαμε το πρωτόκολλο REST, και στην συνέχεια θα γίνει μια ανάλυση στα χαρακτηριστικά του και στις ιδιομορφίες του.

Έπειτα προχωράμε στις παραμέτρους του Android και του πρωτοκόλλου REST χρησιμοποιώντας την αντικειμενοστραφής γλώσσα προγραμματισμού JAVA.

Τέλος μετά από μελέτη όλων αυτών και χρησιμοποιώντας τα κατάλληλα εργαλεία καταφέραμε να φτιάξουμε την παρακάτω εφαρμογή.

Στόχος της εφαρμογής είναι να γίνεται ο έλεγχος της τιμολόγησης και η διαχείριση ρευστότητας (εισπράξεων – πληρωμών) μικρών επιχειρήσεων και ελευθέρων επαγγελματιών. Ο χρήστης της εφαρμογής έχει την δυνατότητα δημιουργίας και διαχείρισης επαφών, την προβολή λίστας παραστατικών πωλήσεων και αγορών, δημιουργίας και διαχείρισης εισπράξεων – πληρωμών και προβολής γραφικών αναπαραστάσεων για μία πιο ολοκληρωμένη εικόνα. Το σύστημα επικοινωνεί με ένα Server μέσω του πρωτοκόλλου REST για να λαμβάνει και να αλλάζει τα δεδομένα.

Abstract

The present paper describes techniques to develop and learn how to create applications for smartphones with Android operating system. For the purposes of the paper an android application has developed using the platform Android Studio, as well as the REST protocol.

Initially we will see why we chose this particular operating system and later we will describe its interior details. After that follows a historical overview of publications and features of Android.

Then we will see what is and why we used the REST protocol, then will be an analysis of its characteristics and its peculiarities.

Furthermore, we will move to the deeper parts of Android and REST protocol parameters using JAVA programming language which is Object Oriented.

Finally, after studying all this and using the proper tools we were able to create the application below.

The aim of this Application is to monitor the invoicing, organizing and managing the cash flow (cash receipts-cash payments) of small businesses and freelance. The user of the Application has the possibility of creating and managing contacts, list view of invoices, both for sales and purchases, creating and managing cash receipts – cash payments and display graphic representations for a more complete picture. The system communicates with a Server via the REST protocol to receive and change the data.

Περιεχόμενα

Κεφάλαιο 1

1.1 Τι είναι το Android

Το **Android** είναι λειτουργικό σύστημα για συσκευές κινητής τηλεφωνίας το οποίο τρέχει τον πυρήνα του λειτουργικού *Linux*. Αρχικά αναπτύχθηκε από την *Google* και αργότερα από την [*Handset Alliance Open Handset Alliance*]. Επιτρέπει στους κατασκευαστές λογισμικού να συνθέτουν κώδικα με την χρήση της γλώσσας προγραμματισμού **JAVA**, ελέγχοντας την συσκευή μέσω βιβλιοθηκών λογισμικού ανεπτυγμένων από την *Google*. Το Android είναι κατά κύριο λόγο σχεδιασμένο για συσκευές με οθόνη αφής, όπως τα έξυπνα τηλέφωνα και τα τάμπλετ, με διαφορετικό περιβάλλον χρήσης για τηλεοράσεις (Android TV), αυτοκίνητα (Android Auto) και ρολόγια χειρός (Android Wear). Παρόλο που έχει αναπτυχθεί για συσκευές με οθόνη αφής, έχει χρησιμοποιηθεί σε κονσόλες παιχνιδιών, ψηφιακές φυτογραφικές μηχανές, συνηθισμένους Η/Υ (π.χ. το HP Slate 21) και σε άλλες ηλεκτρονικές συσκευές.

Το Android είναι το πιο ευρέως διαδεδομένο λογισμικό στον κόσμο. Οι συσκευές με Android έχουν περισσότερες πωλήσεις από όλες τις συσκευές Windows, iOS και Mac OS X μαζί.

Η πρώτη παρουσίαση της πλατφόρμας Android έγινε στις 5 Νοεμβρίου 2007, παράλληλα με την ανακοίνωση της ίδρυσης του οργανισμού Open Handset Alliance, μιας κοινοπραξίας 48 τηλεπικοινωνιακών εταιριών, εταιριών λογισμικού καθώς και κατασκευής hardware, οι οποίες είναι αφιερωμένες στην ανάπτυξη και εξέλιξη ανοιχτών προτύπων στις συσκευές κινητής τηλεφωνίας. 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 Ιρίνα Μπλόκ.

Εικόνα 1: Λογότυπο Android

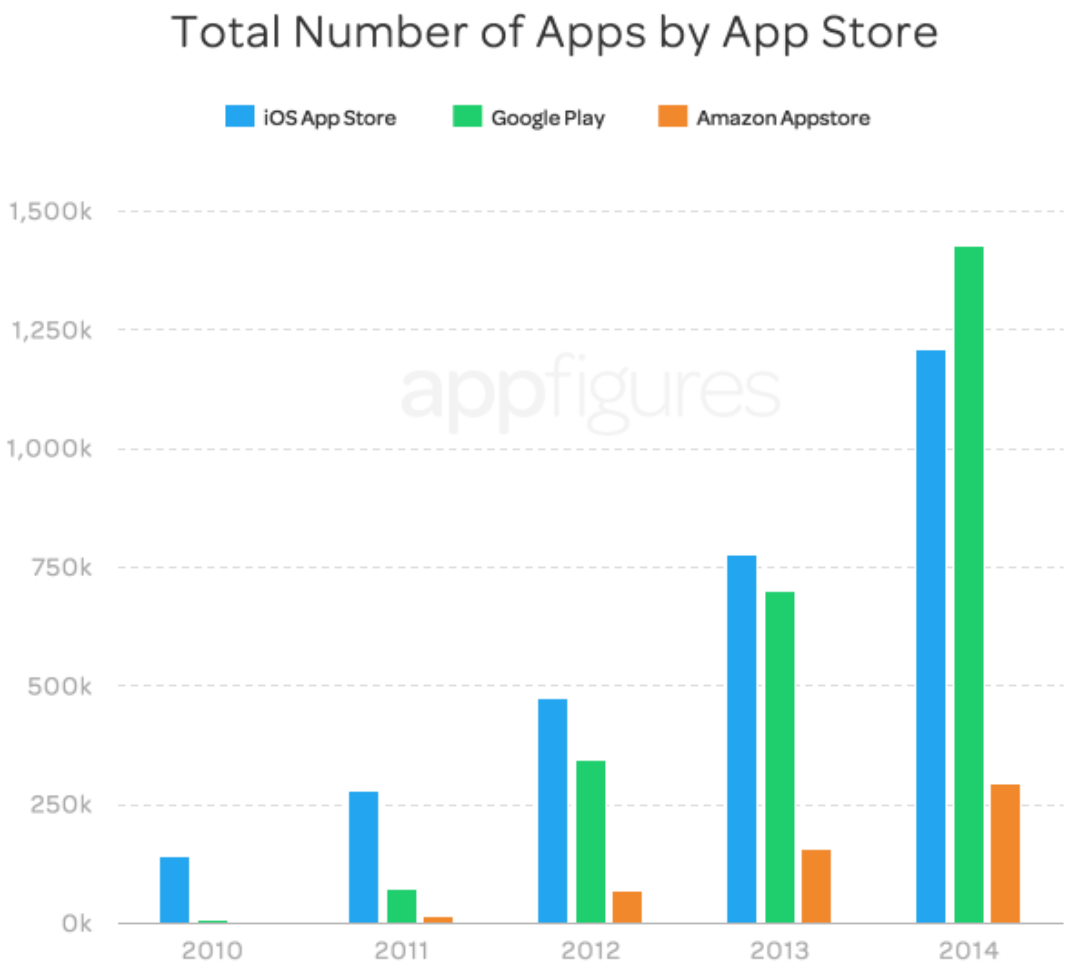
1.2 Χαρακτηριστικά του Android

Τα χαρακτηριστικά του είναι:

- Η πλατφόρμα είναι προσαρμόσιμη σε πολλές αναλύσεις (από VGA, μέχρι 4K).
- Χρήση βάσης δεδομένων SQLite για τις ανάγκες αποθήκευσης.
- Τεχνολογίες συνδεσιμότητας GSM/EDGE, 3G, 4G, CDMA, EV-DO, Bluetooth, NFC,Wi-Fi.
- Διαθέσιμους τρόπους ανταλλαγής μηνυμάτων SMS και MMS.
- Διαθέτει Web browser βασισμένο στην τεχνολογία WebKit.
- Λογισμικό γραμμένο σε JAVA το οποίο γίνεται compile στην εικονική μηχανή Dalvik και ART.
- Υποστηρίζει τις ακόλουθες επεκτάσεις στατικής κινούμενης εικόνας H.263, H.264, 3GP, MP4, MPEG-4 SP, AMR, AMR-WB, JPEG, PNG, GIF, BMP.
- Αρχεία ήχου AAC, HE-AAC, MP3, MIDI, OGG, WAV.
- Χρησιμοποιεί ψηφιακές βιβλιοθήκες και τρισδιάστατα γραφικά χρησιμοποιώντας την τεχνολογία OpenGL ES 3.0+ .
- Μπορεί να διαθέτει γυροσκόπιο, κάμερα, GPS, αισθητήρες επιτάχυνσης, μαγνητόμετρο, δισδιάστατους και τρισδιάστατους επιταχυντές γραφικών.
- Υποστηρίζει την τεχνολογία multi-touch.
- Περιλαμβάνει έναν προσομοιωτή συσκευής, εργαλεία για διόρθωση σφαλμάτων, μνήμη και εργαλεία ανάλυσης της απόδοσης του εκτελέσιμου λογισμικού και ένα επιπρόσθετο για το Eclipse IDE.

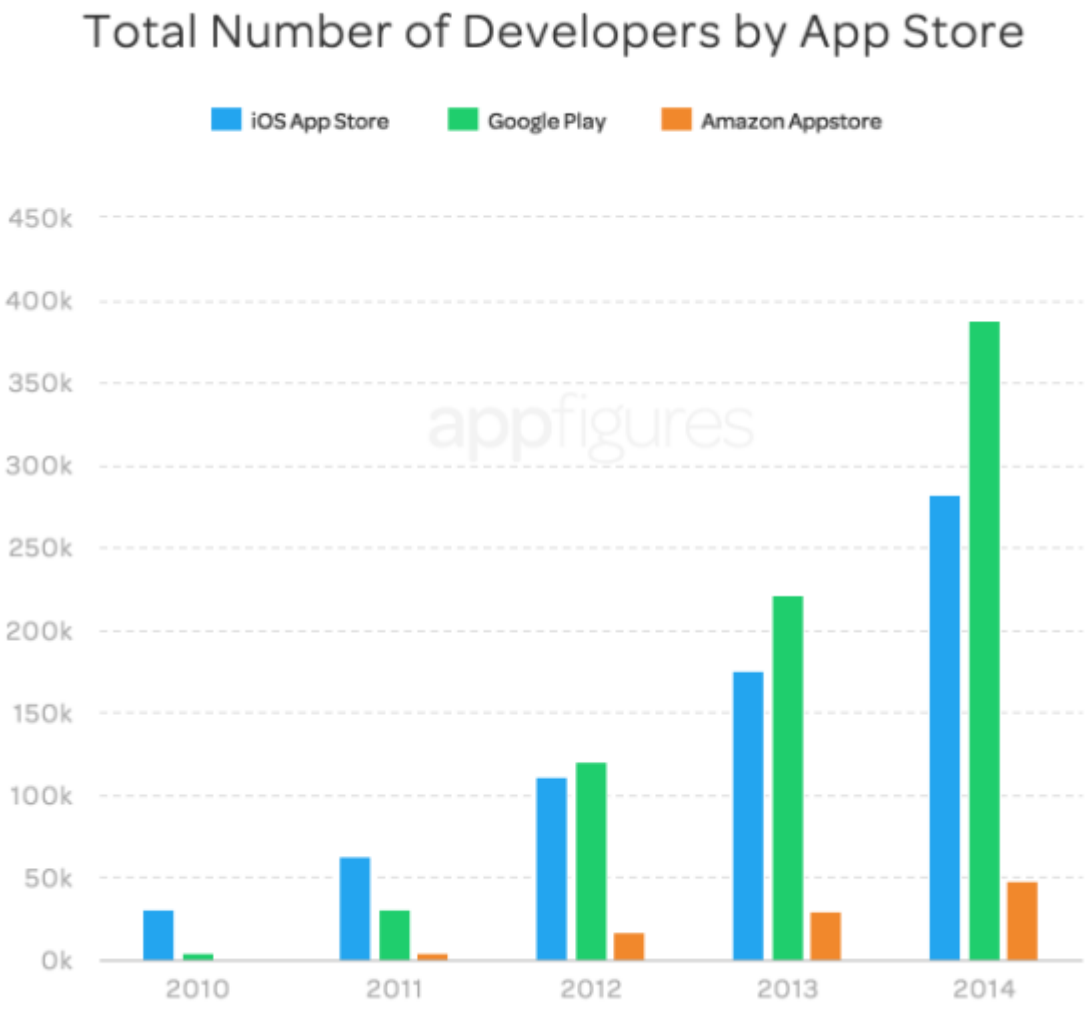
1.3 Γιατί χρησιμοποιούμε Android

Το Android είναι ένα λογισμικό ιδιαίτερος γνωστό και ευρύτερα διαδεδομένο βασικότερα επειδή διαθέτει μια μεγάλη κοινότητα προγραμματιστών και επειδή είναι ανοιχτού κώδικα (Open source). Οι εφαρμογές γράφονται σε μια προσαρμοσμένη έκδοση της JAVA χρησιμοποιώντας τα Android API's τα οποία είναι διαθέσιμα μαζί με τις πλατφόρμες Android Studio και το plugin ADT για το Eclipse. Διαθέτει πάνω από ένα εκατομμύριο εφαρμογές στο Google Play Store και για την ακρίβεια 1.432.671 μέχρι το 2014. Παρακάτω θα δούμε πως μεταβλήθηκαν τα νούμερα αυτά με τα χρόνια:



Εικόνα 2: Διάγραμμα δημοσιευμένων εφαρμογών στο App Store.

Εκτός από την μεγάλη αύξηση στις εφαρμογές, το Google Play Store έχει ανθίσει και στους πολλούς προγραμματιστές Android που ειδικεύονται στην ανάπτυξη και δημοσίευση εφαρμογών στο Google Play Store. Παρακάτω βλέπουμε την αύξηση τα τελευταία χρόνια:



Εικόνα 3: Διάγραμμα προγραμματιστών στο App Store.

1.4 Εξέλιξη του Android

Η φορητότητα στην επικοινωνία έχει γίνει δεκαετίες πριν και έτσι ήρθαν δειλά στην αρχή μα ευρέως και για τις μάζες στην πορεία, τα κινητά τηλέφωνα. Σε αυτό έπαιξε πρωταρχικό ρόλο το Symbian, τ 'οποίο και έτρεχε σε δισεκατομμύρια συσκευές στον πλανήτη (ενώ ακόμα συνεχίζει). Ευρέως γνώστες οι συσκευές της σύγχρονης τεχνολογίας, με οθόνες αφής και τηλέφωνα, τα οποία στην πράξη ήταν μικροί υπολογιστές, έγιναν με το iOS της Apple, χωρίς αυτό να σημαίνει βέβαια, πως η συγκεκριμένη εταιρεία, δημιούργησε κάποια δική τεχνολογία από το μηδέν. Εφάρμοσε τεχνολογίες και πρακτικές, οι οποίες είχαν δημιουργηθεί πολλές δεκαετίες πριν).

Στην εξέλιξη λοιπόν, των λειτουργικών συστημάτων για φορητές συσκευής, ήρθε και το **Android**. Πρώτο έκανε την εμφάνιση του τον Οκτώβριο του 2003, στο Palo Alto, στην πολιτεία της Καλιφόρνιας των ΗΠΑ, από τους: **Andy Rubin, Rich Miner, Nick Sears** και **Chris White**, οι οποίοι ίδρυσαν την: *Android Inc*. Σχεδόν δυο χρόνια αργότερα, τον Ιούλιο του 2005 η Google αποκτά το Android. Στα χρόνια που ακολούθησαν, το Android άλλαξε ριζικά το τοπίο στα smartphones. Πολύ σύντομα το Android αρχίζει να διατίθεται σε αριθμούς που τρομάζουν. Η εξέλιξή του ήταν πολύ μεγάλη και αναμενόμενη λόγω της διαθεσιμότητας του στο ευρύ κοινό χωρίς πληρωμή. Αυτό φαίνεται στο ότι οι 9 κύριες εκδόσεις του έχουν κυκλοφορήσει σε διάστημα 5 ετών από τον Σεπτέμβρη του 2009 με την πρώτη έκδοση Android(Cupcake) έως την πιο πρόσφατη έκδοση Android(Lollipop). Το ποσοστό «διανομής» εκδόσεων

Version	Code name	Release date	API level	Distribution
5.1.x	Lollipop	March 9, 2015	22	7.9%
5.0–5.0.2		November 3, 2014	21	15.6%
4.4–4.4.4	KitKat	October 31, 2013	19	38.9%
4.3.x	Jelly Bean	July 24, 2013	18	4.3%
4.2.x		November 13, 2012	17	14.5%
4.1.x		July 9, 2012	16	11.4%
4.0.3–4.0.4	Ice Cream Sandwich	December 16, 2011	15	3.4%
2.3.3–2.3.7	Gingerbread	February 9, 2011	10	3.8%
2.2–2.2.3	Froyo	May 20, 2010	8	0.2%

Εικόνα 4: Εξέλιξη Android

1.4.1 Android Cupcake (API Level 3 Android Version 1.5)

Τον Απρίλιο του 2009 ήρθε η έκδοση Cupcake με την εισαγωγή νέων χαρακτηριστικών όπως:

- Παγκόσμια αναζήτηση.
- Αναδιάταξη και Επανασχεδιασμός του τότε Android Market με κατηγορίες όπως (Εφαρμογές, Λήψεις, Παιχνίδια) και φίλτρα (Κορυφαίες δωρεάν, Κορυφαίες επι πληρωμή).
- Επέκταση του SDK για να υποστηρίζει κινήσεις με το χέρι (gestures) και ομιλία σε κείμενο (voice to text).

Εικόνα 5: Android Cupcake

1.4.2 Android Donut (API Level 4 Android Version 1.6)

Τον Σεπτέμβριο του 2009 ήρθε το Android Donut το οποίο πρόσθεσε τα εξής χαρακτηριστικά στην διαθεσιμότητα του κοινού:

- Εικονικό πληκτρολόγιο οθόνης.
- Camcorder mode για εγγραφή και προβολή video.
- Εισαγωγή widgets και φακέλων στην αρχική οθόνη.
- Αντιγραφή/Επικόλληση και αναζήτηση στον browser.
- Άμεση μεταφόρτωση στο YouTube και στο Picasa.

Εικόνα 6: Android Donut

1.4.3 Android Éclair (API Level 5 & 6 & 7 Android Versions 2.0 & 2.1)

Τον Οκτώβριο του 2009 έκανε την εμφάνιση του το Android Eclair με αλλαγή Level στο 2.0 φέρνοντας αλλαγές όπως:

- Πολλαπλούς λογαριασμούς χρήστη.
- Υποστήριξη λήψης και αποστολής email.
- Ανεπτυγμένη κάμερα με χρήση flash και ψηφιακό zoom.
- Bluetooth 2.1.
- Ανεπτυγμένο πληκτρολόγιο με προσαρμόσιμο λεξικό και προτάσεις από ονόματα επαφών.
- Live wallpapers.
- Ομιλία σε κείμενο (Speech to text) σε οποιοδήποτε πεδίο κειμένου.

Εικόνα 7: Android Eclair

1.4.4 Android Froyo (API Level 8 Android Version 2.2)

Εικόνα 8: Android Froyo

Τον Μάϊο του 2010 έκανε την εμφάνιση του το Android Froyo εισάγοντας νέες λειτουργίες στην κοινότητα όπως:

- Υποστήριξη Tethering και λειτουργίας Hotspot.
- Αναβάθμιση του Android Market εισάγοντας την δυνατότητα εγκατάστασης εφαρμογών σε εξωτερική κάρτα SD.
- Adobe Flash 10.1.
- Ανέβασμα αρχείων σε πόρους του browser.
- Ανεπτυγμένη υποστήριξη Microsoft Exchange: Πολιτικές Ασφαλείας, Global Διευθύνσεις, συγχρονισμός ημερολογίου, απομακρυσμένη διαγραφή και καθαρισμός.
- Υποστήριξη Bluetooth για φωνητική πληκτρολόγηση και διαμοιρασμό επαφών(contact sharing).

1.4.5 Android Gingerbread (API Level 9 & 10 Android Version 2.3.x)

Εικόνα 9: Android Gingerbread

Τον Δεκέμβριο του 2010 ήρθε έκανε την εμφάνιση του το Android Gingerbread φέρνοντας σημαντικές αλλαγές στην κοινότητα του Android όπως:

- Επανασχεδιασμός του Copy/Paste.
- Υποστήριξη για συμπίεση Web M video.
- NFC (Near Field Communication).
- Αλλαγή σε μπροστινή κάμερα μέσα από την εφαρμογή της κάμερας.
- Συντομεύσεις πληκτρολογίου οθόνης.

1.4.6 Android Honeycomb (API Level 11 & 12 & 13 Android Versions 3.0 & 3.1-3.2.6)

Εικόνα 10: Android Honeycomb

Τον Φεβρουάριο του 2011 έκανε την εμφάνιση της μια ιδιαίτερη έκδοση που απευθυνόταν αποκλειστικά μόνο για tablets, ενσωματώνει ένα νέο σχεδιασμένο με απλούστερο User Interface και υποστήριξη διπύρηνων και τετραπύρηνων επεξεργαστών. Οι καινούργιες αλλαγές που έφερε όμως δεν ήταν μόνο αυτές, έτσι βλέπουμε αλλαγές όπως:

- Υποστήριξη γραφικών 3D.
- Private browsing.
- Δυνατότητα Video Chat με την χρήση της υπηρεσίας Google Talk.
- Δυνατότητα προβολής εικόνων από την γκαλερί σε πλήρης οθόνη.
- Bluetooth Tethering.
- Υποστήριξη περιφερειακών συσκευών όπως πληκτρολόγια και game pads.

- Παραμετροποιήσιμα widgets.
- Δυνατότητα υποστήριξης της λειτουργίας πληρωμή στον δρόμο (Pay as you go) 3G/4G tablets.
- Διορθώσεις σφαλμάτων και σταθερότητας.

1.4.7 Android Ice-cream Sandwich (API Level 14 & 15 Android Version 4.0.x)

Με πάρα πολλές καινούργιες δυνατότητες κάνει την εμφάνιση του τον Οκτώβριος του 2011 το Android Ice-cream Sandwich. Με επανασχεδιασμένο User Interface πιο γρήγορο και αποδοτικό εισάγει νέα εικονικά κουμπιά που αντικαθιστούν τα αφής ή φυσικά που υπήρχαν στις εκάστοτε συσκευές. Οι αλλαγές που έφερε είναι πολλές όπως θα δούμε παρακάτω:

Εικόνα 11: Android Ice-cream Sandwich

- Δημιουργία φακέλων απλά τραβώντας την μια εφαρμογή επάνω σε μια άλλη.
- Διαχωρισμός μεταξύ εφαρμογών και widgets.
- Υποστήριξη λειτουργίας pinch-to-zoom στην εφαρμογή ημερολογίου.
- Επανασχεδιασμός της εφαρμογής Gmail και παράλληλα προσθέτοντας την λειτουργία offline search καθώς επίσης την λειτουργία swipe μεταξύ email μηνυμάτων.
- Επανασχεδιασμός της εφαρμογής Google Chrome η οποία πλέον υποστηρίζει καρτέλες εκτός σύνδεσης, μέχρι 16 συνεχόμενες καρτέλες ανοικτές, συγχρονισμός σελιδοδεικτών.
- Διορθώσεις σφάλματων πληκτρολογίου οθόνης.
- Προσαρμόσιμη οθόνη κλειδώματος, υποστήριξη launcher.
- Εισαγωγή γραμματοσειράς Roboto.
- Νέα λειτουργία διαγραφής με κύλιση.
- Αναπτυγμένη αναγνώριση ομιλίας και λειτουργία Copy/Paste.
- Ξεκλείδωμα οθόνης με λειτουργία αναγνώρισης προσώπου.
- Μετρητής δεδομένων κινητής τηλεφωνίας.
- Λειτουργία απόκρυψης επιλεγμένων εφαρμογών από τον χρήστη.
- Ικανότητα για τερματισμό λειτουργίας εφαρμογών που χρησιμοποιούν πόρους υλικού.
- Πρόσθετες λειτουργίες της εφαρμογής κάμερας: zero shutter lag, continuous focus, zoom while recording, time lapse settings, panorama photos, 1080p recording.
- Δυνατότητα αναγνώρισης προσώπου (Face recognition) μέσα από την κάμερα με ενσωματωμένο Photo Editor.
- Νέος σχεδιασμός του gallery App.
- Δυνατότητα αποστολής άμεσου μηνύματος στην απόρριψη κλήσεων.

- Λειτουργία Android Beam μια ιδιότητα του NFC για αποστολή αρχείων όταν τα κινητά βρίσκονται σε κοντινή απόσταση.

1.4.8 Android Jellybean (API Level 16 & 17 & 18 Android Versions 4.1 & 4.2 & 4.3)

Τον Ιούλιο του 2012 έκανε την εμφάνιση του το Android Jellybean φέρνοντας ραγδαίες αλλαγές όχι μόνο στον σχεδιασμό των εφαρμογών αλλά και σε ολόκληρο το λειτουργικό σύστημα. Οι αλλαγές είναι οι εξής:

Εικόνα 12: Android Jellybean

- Πιο γρήγορη και ομαλή απόδοση με το λεγόμενο Project Butter.
- Επεκτάσιμες ειδοποιήσεις με μεγαλύτερη δια δραστικότητα.
- Φωνητική αναζήτηση με κύλιση από το κάτω μέρος της οθόνης.
- Εισαγωγή της υπηρεσίας Google Now.
- Λειτουργία Offline Dictation.
- Αναπροσαρμοσμένα widgets εφαρμογών.
- Αναβάθμιση της λειτουργίας Android Beam στο να υποστηρίζει μεγαλύτερου όγκου αρχεία όπως εικόνες και video.
- Αναβάθμιση του Android Market σε Google Play Store.
- Φωνητική αναζήτηση μουσικών κομματιών και καλλιτεχνών.
- Υψηλής ανάλυσης φωτογραφίες επαφών.
- Επιλογές μεγαλύτερης προσβασιμότητας.
- Επέκταση υποστήριξης γλωσσών σε Αραβικά και Εβραϊκά.
- Interface Tweaks.
- Δυνατότητας εισαγωγής widgets στην οθόνη κλειδώματος και άνοιγμα κάμερας απευθείας από την οθόνη κλειδώματος.
- Δυνατότητα εισαγωγής μπάρας γρήγορων ρυθμίσεων στο πάνελ ειδοποιήσεων για πιο εύκολη πρόσβαση στο Wi-Fi, Bluetooth, etc.
- Δυνατότητα λειτουργίας Daydream.
- Wireless Display (Miracast).
- Λειτουργίες προσβασιμότητας όπως (τριπλό χτύπημα οθόνης για μεγέθυνση, zoom με δυο δάκτυλα, φωνητική πλοήγηση για τυφλούς).
- Ενοποιημένο User Interface για όλες τις συσκευές.
- Υποστήριξη Bluetooth Low Energy.
- Εντοπισμός θέσης με την υποστήριξη Wi-Fi χωρίς την ενεργοποίηση του Wi-Fi.
- Υποστήριξη οθονών με ανάλυση 4K.

1.4.9 Android KitKat (API Level 19 & 20 Android Version 4.4)

Εισάγοντας νέες αλλαγές στο design τον Οκτώβριο του 2013 έκανε την εμφάνισή του το Android KitKat όπως είναι οι παρακάτω:

- Μεγάλες αλλαγές στο design ειδικότερα για Nexus συσκευές.
- Διαφανής μπάρα κατάστασης για το λειτουργικό σύστημα και για εφαρμογές.
- Νέα λειτουργία immersive mode όπου οι εφαρμογές μπορούν να κρύβουν την πλοήγηση και την μπάρα κατάστασης.
- Μείωση του μεγέθους του λειτουργικού συστήματος για γρηγορότερη λειτουργία και απόδοση σε συσκευές με μικρές ποσότητες μνήμης RAM.
- Ασύρματη εκτύπωση με της υπηρεσίας Google Cloud Print.

Εικόνα 13: Android KitKat

1.4.10 Android Lollipop (API Level 21 Android Version 5.0)

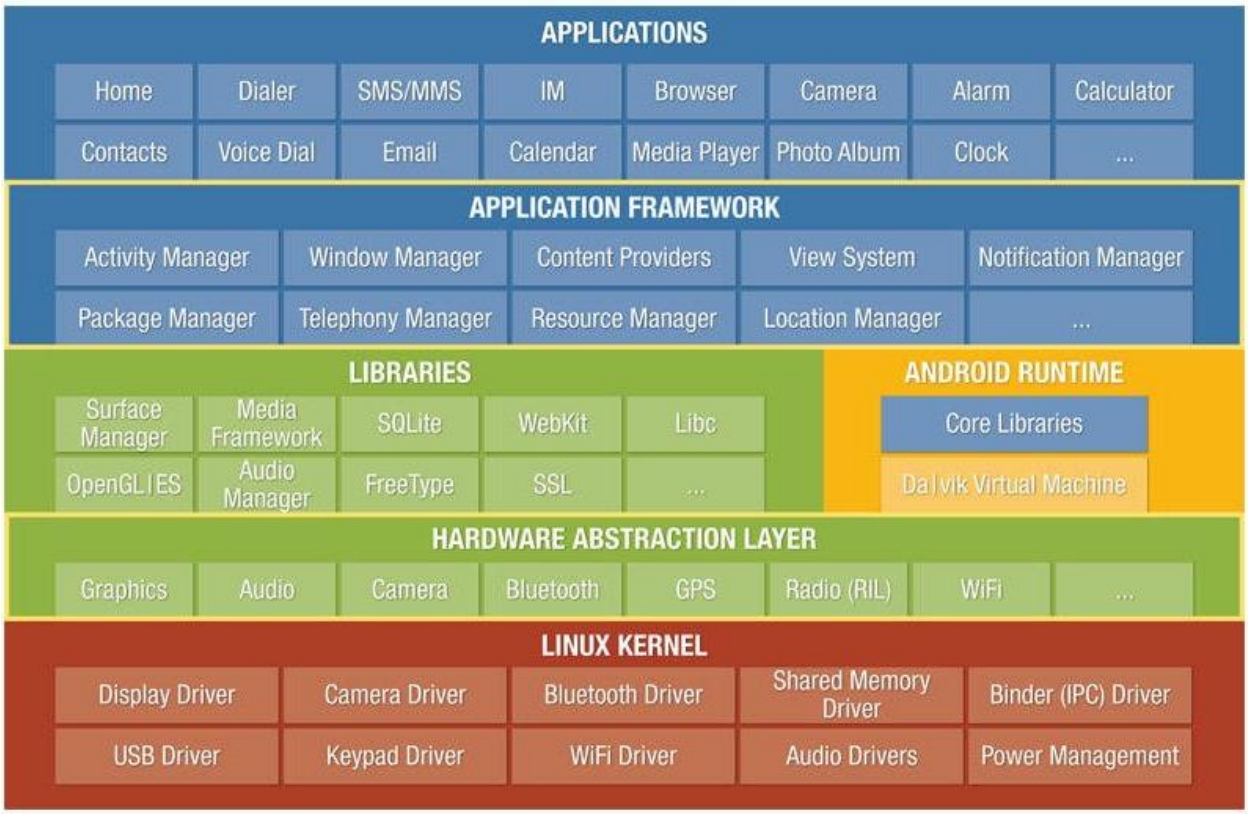
Πολλά υποσχόμενο και νέες αλλαγές έφερε η νέα και πολυσυζητημένη έκδοση Android Lollipop. Πλήρως επανασχεδιασμένο Material Design και ανανεωμένο User Interface έτσι ώστε να παρέχει την βέλτιστη απόδοση για χρήστες με μικρές ποσότητες μνήμης RAM. Οι αλλαγές που θα έγιναν είναι οι παρακάτω:

Εικόνα 14: Android Lollipop

- Ειδοποιήσεις κατευθείαν στην οθόνη κλειδώματος και pop-up μηνύματα.
- Λειτουργία priority mode η οποία εμφανίζει τις πιο σημαντικές ειδοποιήσεις.
- Πολλαπλοί λογαριασμοί χρηστών για smartphones και tablets.
- Νέο menu recent apps με ονομασία Overview.
- Λειτουργία Guest mode.
- Screen Pinning.
- Battery Saver mode.
- Προεπιλεγμένη κρυπτογράφηση συσκευής.
- Έξυπνο κλείδωμα/ξεκλείδωμα ξεκλειδώνει συσκευές Bluetooth ή NFC.

1.5 Αρχιτεκτονική του Android

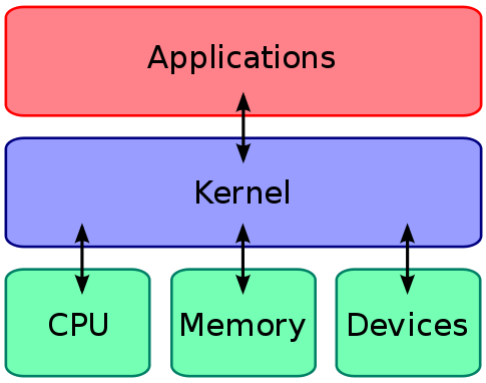
Το Android δεν είναι απλά ένα λειτουργικό σύστημα αλλά είναι μια στοίβα από διεργασίες και υπηρεσίες που επικοινωνούν με τις εφαρμογές γνωστές και ως middleware, τις κύριες εφαρμογές γνωστές και ως core, λειτουργικό σύστημα. Στο παρακάτω γράφημα θα δούμε την αρχιτεκτονική αυτή:



Εικόνα 15: Αρχιτεκτονική του Android

1.5.1 Πυρήνας (Linux Based Kernel)

Android Kernel ονομάζουμε τον πυρήνα του λειτουργικού συστήματος και είναι υπεύθυνος για την διασύνδεση εφαρμογών με το hardware όπως φαίνεται στην παρακάτω εικόνα.



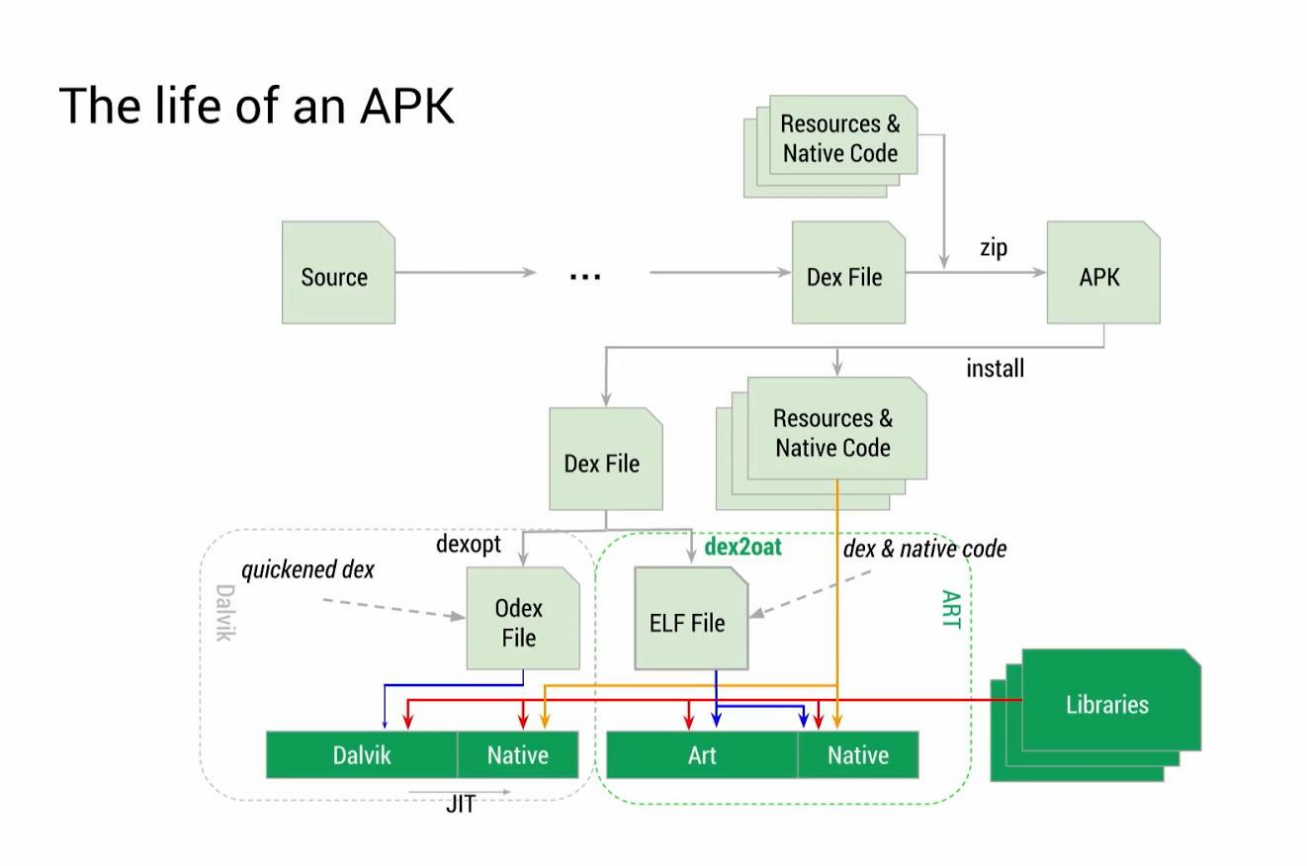
Εικόνα 16: Android Kernel

Η κατασκευάστρια εταιρία κάθε συσκευής πρέπει να ενσωματώνει διαφορετικούς drivers για το hardware πράγμα το οποίο καθιστά το Kernel της κάθε συσκευής διαφορετικό. Πολλοί προγραμματιστές αλλάζουν το Kernel για πολλούς λόγους όπως είναι (bug fixing, overclocking, better RAM performance, etc).

1.5.2 Η Εικονική μηχανή Dalvik

Η Dalvik είναι η εικονική μηχανή που χρησιμοποιεί η κάθε εφαρμογή για να τρέξει. Έτσι γίνεται το λεγόμενο multi-tasking χωρίς οι εικονικές μηχανές να επηρεάζονται μεταξύ τους και με αποτέλεσμα οι εφαρμογές να εκτελούνται ταυτόχρονα. Τα αρχεία της εφαρμογής πρέπει να είναι σε μορφή .dex διότι η Dalvik δεν μπορεί να κάνει compile σε κώδικα BYTE έτσι ώστε να τα τρέξει το Virtual Machine. Μια από τις σημαντικότερες προσθήκες στην έκδοση 4.4.x ήταν η προσθήκη του Compiler ART πράγμα το οποίο πέρασε απαρατήρητο από πολλούς χρήστες κατά το Runtime της συσκευής αλλά και της εφαρμογής. Αρχικά ήρθε σε πειραματικό στάδιο πράγμα το οποίο σήμαινε ότι για να το ενεργοποιήσει κάποιος χρήστης θα έπρεπε να ανοίξει την κρυφή λειτουργία Developer Options και να αλλάξει το Runtime από Dalvik σε ART. Με την έκδοση 5.0 το ART έγινε πλέον το default Runtime των συσκευών.

Παρακάτω θα δούμε μια σύγκριση του Dalvik και του ART Runtime.



Εικόνα 17: Σύγκριση του Dalvik και του ART Runtime

1.5.3 Πλαίσιο Εφαρμογής Android

Το πλαίσιο εφαρμογής περιλαμβάνει τα εξής

1) View System
2) Content Providers
3) Resource Manager
4) Notification Manager
5) Activity Manager

Οι προγραμματιστές έχουν πρόσβαση σε όλα τα API's καθιστώντας έτσι την δημιουργία πολύπλοκων εφαρμογών με συγκεκριμένες άδειες δίνοντας πρόσβαση σε όλα τα μέρη του λειτουργικού συστήματος.

1.5.3.1 *View System*

Είναι όλα τα Γραφικά Περιβάλλοντα του Προγραμματιστή που θα χρησιμοποιηθούν για την δημιουργία της εφαρμογής. Μπορούν να υλοποιηθούν μέσω XML μέσω της εντολής setContentView(int resourceID) η οποία παίρνει όρισμα ένα id του αρχείου XML που υλοποιείται ή προγραμματιστικά μέσω της εντολής:

```
TextView tv = new TextView(this);
tv.setLayoutParams(new ViewGroup.LayoutParams(
ViewGroup.LayoutParams.WRAP_CONTENT,
ViewGroup.LayoutParams.WRAP_CONTENT));
```

Στην οποία βλέπουμε πως υλοποιείται ένα πεδίο κειμένου (ετικέτας). Τα γραφικά περιβάλλοντα μπορούν να περιλαμβάνουν πολύπλοκα GUI's χρησιμοποιώντας πολλούς Layout Managers(Linear Layout, Relative Layout, Table Layout, Listview, Spinner View, Frame Layout, Buttons, Edittext, Textview, Checkbox, Switch, ToggleButton, Imageview, e.t.c.).

1.5.3.2 *Content Providers*

Οι Content Providers δίνουν την δυνατότητα στην εφαρμογή να έχει πρόσβαση σε δομημένα δεδομένα. Ενθυλακώνουν μηχανισμούς παραλαβής, αποστολής και ασφάλισης δεδομένων. Για να μπορεί να γίνει λήψη των δεδομένων χρειάζονται κάποιες συγκεκριμένες άδειες που επιτρέπουν στην εφαρμογή να κάνει συγκεκριμένα πράγματα. Για παράδειγμα αν θέλαμε να δημιουργήσουμε μια εφαρμογή ανταλλαγής μηνυμάτων κειμένου θα έπρεπε να βάλουμε το στο Manifest.xml το hasSystemFeature(PackageManager.FEATURE_TELEPHONY). Και τα permissions.

- <uses-permission android:name=”android.permission.WRITE_SMS”>
- <uses-permission android:name=”android.permission.READ_SMS”>
- <uses-permission android:name=”android.permission.SEND_SMS”>
- <uses-permission android:name=”android.permission.RECEIVE_SMS”>
- <uses-permission android:name=”android.permission.BROADCAST_SMS”>
- <uses-permission android:name=”android.providers.Telephony”>
- <uses-permission android:name=”android.providers.Telephony.SMS”>

1.5.3.3 *Resource Manager*

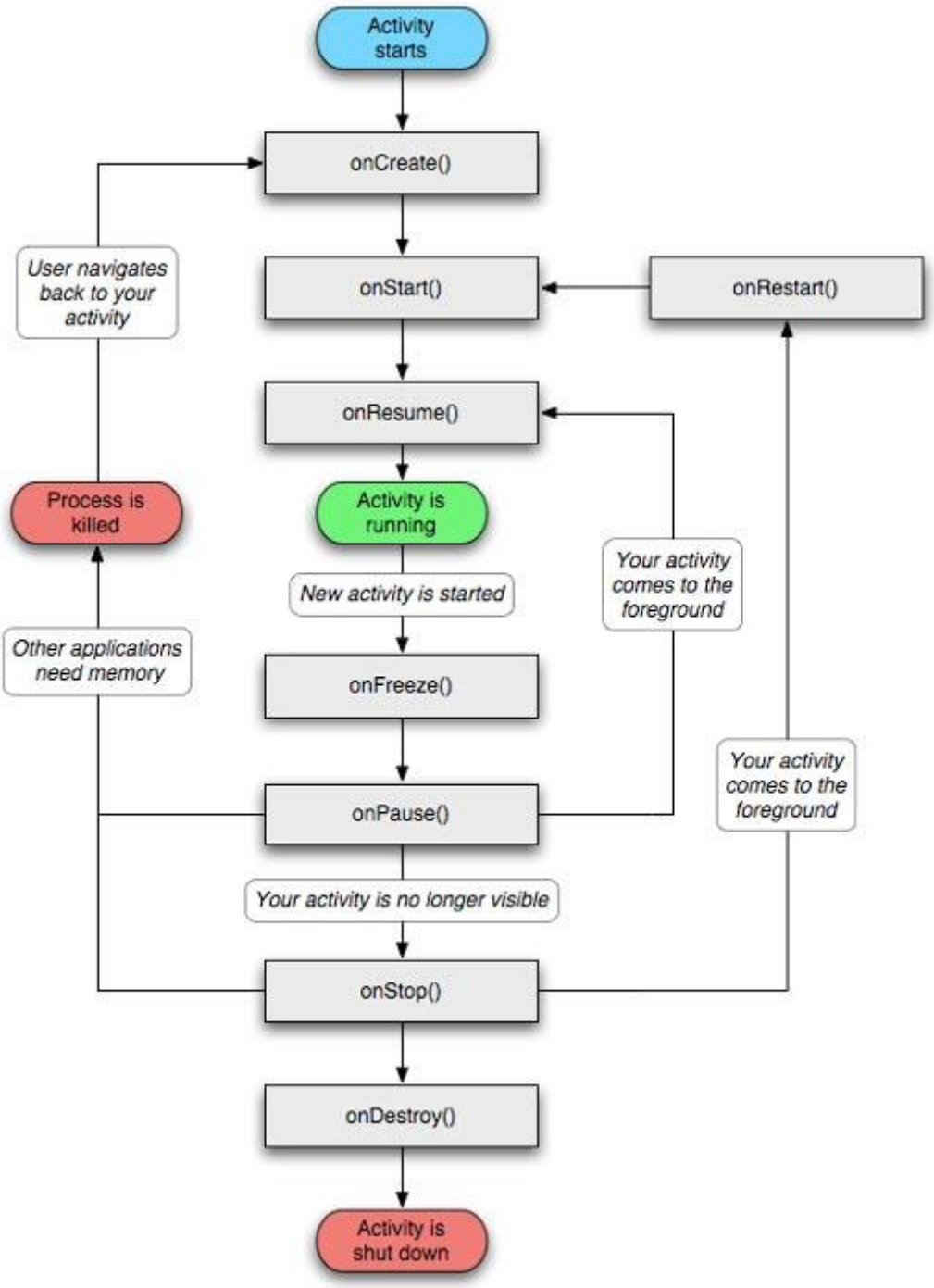
Δίνει πρόσβαση στην εφαρμογή από εξωτερικούς παράγοντες όπως είναι (αρχεία, εικόνες από την μνήμη (εσωτερική ή εξωτερική), αρχεία xml, και άλλα.

1.5.3.4 *Notification Manager*

Δίνει την δυνατότητα στην εφαρμογή να ενημερώνει τον χρήστη για αλλαγές ή νέες προσθήκες στην εφαρμογή χωρίς ο χρήστης να χρειάζεται να ανοίξει την εφαρμογή και να φορτώσει τα δεδομένα απασχολώντας πολύτιμους πόρους του συστήματος. Για παράδειγμα Notifications μας στέλνει το Google Play Store για το ποιες εφαρμογές μας είναι έτοιμες για ενημέρωση χωρίς να ανοίγουμε εμείς κάθε φορά να κάνουμε έλεγχο για το αν κάποια εφαρμογή έχει διαθέσιμη ενημέρωση.

1.5.3.5 *Activity Manager*

Διαχειρίζεται το πως συμπεριφέρεται μια εφαρμογή ανάλογα με το πως χρησιμοποιείται από τον χρήστη. Για παράδειγμα τι μεθόδους θα καλέσει όταν ο χρήστης την έχει ανοικτή, όταν βρίσκεται στο παρασκήνιο και όταν τερματίζεται η λειτουργία της. Θα δούμε τον κύκλο ζωής της εφαρμογής στο παρακάτω διάγραμμα.



Εικόνα 18: Κύκλος ζωής της εφαρμογής

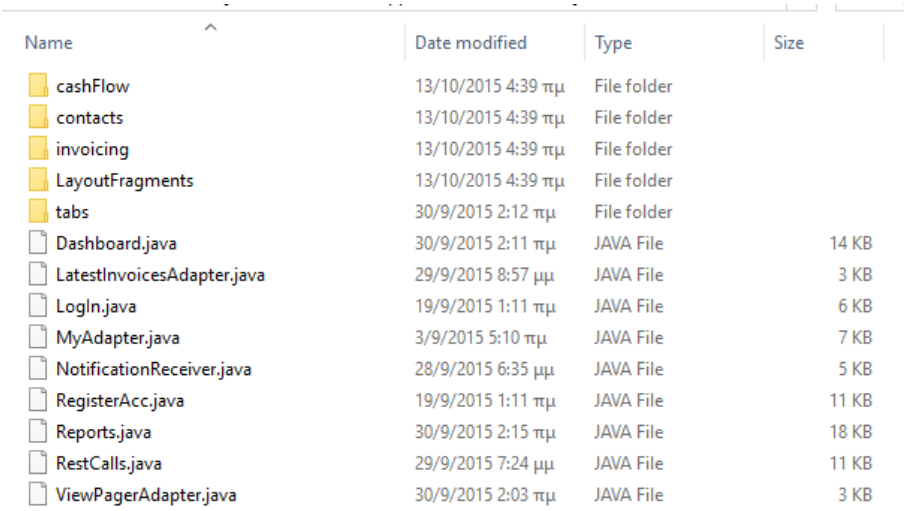
1.6 Εσωτερικό Εφαρμογής Android

Το εσωτερικό μιας εφαρμογής περιλαμβάνει τα ακόλουθα:

- Φάκελος src
- Φάκελος res
- AndroidManifest.xml
- Gradle Scripts

1.6.1 Φάκελος src

Ο πιο σημαντικός φάκελος μέσα στην εφαρμογή. Περιέχει όλα τα αρχεία .java στο κάθε package και είναι υπεύθυνος για τον κώδικα της εφαρμογής, τον φάκελο res, τον φάκελο assets που εμπεριέχει διάφορα fonts, όπως επίσης και το αρχείο AndroidManifest.xml. Αυτά μετατρέπονται σε αρχεία .dex για να μπορούν να γίνουν bytecode από τους compilers Dalvik και ART. Διαμορφώνεται κάπως έτσι:



Εικόνα 19: Φάκελος src/java

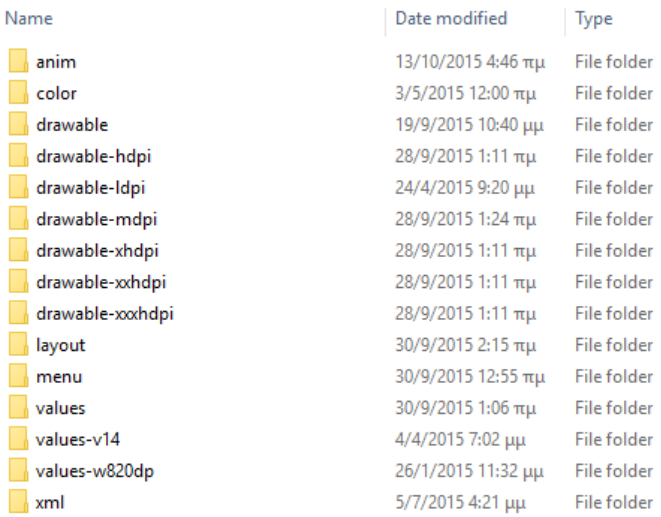
Βλέπουμε τα packages που στο καθένα εμπεριέχονται τα αρχεία .java

1.6.2 Φάκελος res

Ο φάκελος res περιέχει διάφορους φακέλους αναλόγως το περιεχόμενο. Περιέχει τους φακέλους:

- Anim (Καθορίζει τα animations που θα χρησιμοποιηθούν για τις εναλλαγές μεταξύ activities ή για το πώς θα εμφανιστεί ένα πλαίσιο κειμένου και άλλα.)
- Drawable(Περιέχει τους φακέλους εικόνων (hdpi, mdpi, ldpi, xhdpi, xxhdpi, xxxhdpi) καθώς και αρχεία .xml που καθορίζουν συγκεκριμένο στυλ για κάποιες συμπεριφορές σε διάφορα κομμάτια κώδικα)
- Layout(Υπεύθυνος για τα αρχεία .xml που χρησιμοποιεί η εφαρμογή μας)
- Menu(Περιέχονται τα .xml που είναι υπεύθυνα για την δημιουργία μενού και υπό μενού σε συγκεκριμένα σημεία της εφαρμογής μας)
- Values(Περιέχει τα αρχεία (styles.xml, colors.xml, dimens.xml, strings.xml))
- Styles.xml(Περιέχει το πως θα είναι και θα φαίνεται η εφαρμογή στον τελικό χρήστη)
- Colors.xml(Περιέχει ένα πίνακα στατικών χρωμάτων στα οποία μπορούμε να έχουμε πρόσβαση από κάθε σημείο της εφαρμογής μας)
- Dimens.xml(Περιέχει αναφορές σε διάφορα περιθώρια και μετακινήσεις μεταξύ αντικειμένων της εφαρμογής μας)
- Strings.xml(Περιέχει στατικά πεδία κειμένου τα οποία μπορούν να μεταφραστούν με το οποίο μπορούμε να πετύχουμε το globalization και localization της εφαρμογής μας)

Η δομή φαίνεται στην παρακάτω εικόνα:



Name	Date modified	Type
anim	13/10/2015 4:46 πμ	File folder
color	3/5/2015 12:00 πμ	File folder
drawable	19/9/2015 10:40 μμ	File folder
drawable-hdpi	28/9/2015 1:11 πμ	File folder
drawable-ldpi	24/4/2015 9:20 μμ	File folder
drawable-mdpi	28/9/2015 1:24 πμ	File folder
drawable-xhdpi	28/9/2015 1:11 πμ	File folder
drawable-xxhdpi	28/9/2015 1:11 πμ	File folder
drawable-xxxhdpi	28/9/2015 1:11 πμ	File folder
layout	30/9/2015 2:15 πμ	File folder
menu	30/9/2015 12:55 πμ	File folder
values	30/9/2015 1:06 πμ	File folder
values-v14	4/4/2015 7:02 μμ	File folder
values-w820dp	26/1/2015 11:32 μμ	File folder
xml	5/7/2015 4:21 μμ	File folder

Εικόνα 20: Φάκελος res

1.6.3 AndroidManifest.xml

Ίσως το πιο σημαντικό αρχείο μέσα σε όλη την εφαρμογή. Περιέχει το package name, αναφορές στα υπόλοιπα packages, κλάσεις, activities, permissions. Η δομή του φαίνεται παρακάτω:

AndroidManifest.xml

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

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.CALL_PHONE"/>

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/Theme.MyTheme" >
        <receiver
            android:name=".NotificationReceiver"
            android:enabled="true"
            android:exported="true"
            android:label="NotificationReceiver">
            <intent-filter>
                <action android:name="android.intent.action.SCREEN_OFF"/>
                <action android:name="android.intent.action.SCREEN_ON"/>
                <action android:name="android.intent.action.USER_PRESENT"/>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
            </intent-filter>
        </receiver>
        <activity
            android:name=".LogIn"
            android:windowSoftInputMode="adjustPan" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".RegisterAcc"
            android:windowSoftInputMode="adjustPan" />
        <activity
            android:name=".Dashboard"
            android:configChanges="orientation|screenSize"
            android:windowSoftInputMode="adjustPan" />
        <activity
            android:name=".contacts.NewContact"
            android:configChanges="orientation|screenSize"
            android:windowSoftInputMode="adjustPan" />
        <activity
            android:name=".contacts.ViewContact"
            android:windowSoftInputMode="adjustPan" />
        <activity
            android:name=".contacts.EditContact"
```

Εικόνα 21: AndroidManifest.xml

1.6.4 Gradle Scripts

Το σύστημα διαθέτει ένα toolkit το οποίο παρέχει κάποιες συγκεκριμένες δυνατότητες (Τροποποίηση, Ρύθμιση, Επέκταση) της διαδικασίας του build. Παρέχεται η δυνατότητα δημιουργίας πολλών και πολύπλοκων εφαρμογών με την χρήση ίδιων προσθέτων (plugins), βιβλιοθηκών (libraries, depencencies) διαφόρων επεκτάσεων (modules). Το αρχείο build.gradle είναι υπεύθυνο για την προσθήκη όλων των παραπάνω, έτσι ώστε εάν θέλουμε να χρησιμοποιήσουμε μια custom library να μην κάνουμε εισαγωγή όλο το Project στην εφαρμογή μας μόνο μια αναφορά στο groupId, ArtifactId της βιβλιοθήκης που θέλουμε να χρησιμοποιήσουμε.

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

android {
    compileSdkVersion 22
    buildToolsVersion '22.0.1'
    defaultConfig {
        applicationId 'com.elorus'
        minSdkVersion 14
        targetSdkVersion 22
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
        debug {
            debuggable true
        }
    }
    productFlavors {
    }
}

dependencies {
    compile 'de.hdodenhof:circleimageview:1.3.0'
    compile 'com.android.support:recyclerview-v7:22.2.1'
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:22.2.1'
    compile 'com.shamanland:fonticon:0.1.8'
    compile 'org.json:json:20141113'
    compile 'org.apache.httpcomponents:httpclient:4.4'
    compile 'com.android.support:support-v4:22.0.+'
}
```

Εικόνα 22: Gradle Script

1.7 Δομικά μέρη μιας εφαρμογής

Τα δομικά μέρη μιας εφαρμογής περιλαμβάνουν τα εξής:

- Activities
- Services
- Intents
- Broadcast Receivers

1.7.1 Activity

Είναι οι διάφορες οθόνες μιας εφαρμογής. Εκεί μέσα ο κάθε προγραμματιστής επιλέγει πως θα συνδεθεί το γραφικό περιβάλλον της εφαρμογής με τον πηγαίο κώδικα ή εγκέφαλο της εφαρμογής. Όλα τα Activities δηλώνονται στο Manifest καθώς και τα packages στα οποία ανήκουν. Όλα τα Activities μαζί αποτελούν το τελικό αποτέλεσμα που βλέπει ο χρήστης στην συσκευή του.

- *Fragments*

  Ένα fragment αντιπροσωπεύει μια συμπεριφορά ή ένα τμήμα της διεπαφής του χρήστη σε ένα Activity. Πολλά fragments μπορούν να συνδυαστούν σε ένα ενιαίο Activity για την οικοδόμηση μιας multi-window UI και την επαναχρησιμοποίηση ενός fragment σε πολλαπλές δραστηριότητες. Μπορούμε να θεωρήσουμε τα fragments ως αρθρωτά τμήματα ενός Activity, τα οποία το καθένα έχει το δικό του Lifecycle, λαμβάνει τα δικά του input events και τα οποία μπορούν να προστεθούν ή να αφαιρεθούν ενώ το Activity είναι σε λειτουργία (περίπου όπως ένα «υπό Activity» που μπορεί να επαναχρησιμοποιηθεί σε διάφορα activities).

- *Loaders*

  Συστήθηκαν στο Android API 3.0, οι loaders κάνουν εύκολη την φόρτωση την δεδομένων ασύγχρονα σε ένα Activity ή ένα Fragment. Έχουν αυτά τα χαρακτηριστικά:

  - Είναι διαθέσιμοι σε κάθε Activity και Fragment.
  - Παρέχουν ασύγχρονη φόρτωση δεδομένων
  - Παρακολουθούν την πηγή των δεδομένων και παραδίδουν τα νέα αποτελέσματα όταν αυτά. Αλλάξουν.
  - Μπορούν να επανασυνδεθούν στην προηγούμενη θέση του κέρσορα φόρτωσης όταν αναδημιουργούνται μετά την αλλαγή ρυθμίσεων. Έτσι δεν χρειάζεται να γίνει query για τα δεδομένα τους ξανά.

- *Tasks and Back Stack*

  Η εφαρμογή περιέχει συνήθως πολλαπλά Activities. Κάθε ένα από αυτά πρέπει να σχεδιαστεί γύρω από ένα συγκεκριμένο πλάνο όπου ο χρήστης μπορεί να εκτελέσει και να ξεκινήσει άλλα Activities. Για παράδειγμα, μια εφαρμογή ηλεκτρονικού ταχυδρομείου μπορεί να έχει ένα Activity που να δείχνει μια λίστα με τα νέα μηνύματα. Όταν ο χρήστης επιλέξει ένα μήνυμα, ένα νέο Activity ανοίγει για να δει το μήνυμα αυτό.

- *Overview Screen*

  Το Overview screen (αναφέρεται επίσης ως recent screen, recent task list ή recent apps) είναι ένα επίπεδο συστήματος UI που απαριθμεί πρόσφατα προσβάσημα Activities και διεργασίες. Ο χρήστης μπορεί να πλοηγηθεί στην λίστα και να επιλέξει μια διεργασία για να συνεχίσει ή να αφαιρέσει μια διεργασία από την λίστα σύροντάς την. Με την Android έκδοση 5.0, υπάρχουν πολλαπλές εμφανίσεις από το ίδιο Activity που περιέχει διάφορα έγραφα ενδέχεται να εμφανίζονται ως λειτουργίες στο Overview Screen. Για παράδειγμα, το Google Drive μπορεί να έχει μια διεργασία για κάθε ένα από διάφορα έγγραφα της Google. Κάθε έγγραφο εμφανίζεται ως μία διεργασία στο Overview screen.

1.7.2 Services

Πρόκειται για background Threads που τρέχουν για όση διάρκεια απαιτείται για να γίνει μια διεργασία, όπως είναι να φορτωθούν δεδομένα από το διαδίκτυο ή να μετράμε τον χρόνο για κάποια διεργασία ή να παρέχουμε ζωντανές πληροφορίες όλο το 24ώρο. Δηλώνονται και αυτά στο Manifest και δεν μπορεί να τα δει ο χρήστης στο τελικό αποτέλεσμα.

1.7.3 Intents

Πρόκειται για διεργασίες που επιτρέπουν και καθορίζουν την επικοινωνία μεταξύ Activities. Δηλώνονται στο Manifest ή και προγραμματιστικά. Καθορίζουν ποιο Activity θα αρχίσει πρώτο καθώς και ποιο θα ακολουθήσει όπως επίσης χρησιμοποιούνται για να μεταφέρουν δεδομένα από ένα Activity σε ένα άλλο. Χρησιμοποιούνται για πρόσβαση σε άλλες εφαρμογές όπως είναι οι ρυθμίσεις συστήματος της συσκευής ή στο να παραπέμψουν τον χρήστη να εγκαταστήσει μια άλλη εφαρμογή ή πρόσθετο που απαιτείται για την ορθή λειτουργία της εφαρμογής.

1.7.4 Broadcast Receivers

Πρόκειται για υπηρεσίες που χρησιμοποιούνται με στόχο την λήψη, αποστολή, ενημέρωση του χρήστη ή και των άλλων εφαρμογών του συστήματος. Δεν είναι ορατές στον χρήστη και δηλώνονται στο Manifest. Χρησιμοποιούνται συχνά για ανταλλαγή μηνυμάτων, ενημέρωση του χρήστη μέσω push notifications και άλλα.

Κεφάλαιο 2

2.1 Εγκατάσταση Λογισμικού

Για να μπορέσει ο προγραμματιστής να αναπτύξει την δικιά του εφαρμογή θα πρέπει να έχει εγκαταστήσει ένα ολοκληρωμένο περιβάλλον διαχείρισης και δημιουργίας εφαρμογών Android τα λεγόμενα IDE. Τα περιβάλλοντα αυτά διαθέτουν πολλά εργαλεία αποσφαλμάτωσης, built-in Compiler, auto-complete text editor και επίσης αυτόματη δημιουργία όλων των χρήσιμων αρχείων και φακέλων που καθιστούν την εφαρμογή έτοιμη να τρέξει και να χρησιμοποιηθεί από τον τελικό χρήστη.

Για την εγκατάσταση του περιβάλλοντος διαχείρισης απαιτείται να έχει εγκατεστημένο το JDK(Java Development Kit). Μετά θα μπορέσει να επιλέξει όποιο περιβάλλον προγραμματισμού αυτός θεωρεί πιο βολικό (Android Studio, Eclipse IDE (Android Developer Tools Built-in), Netbeans IDE, Intellij IDEA IDE) για να φτιάξει την εφαρμογή του. Θα πρέπει όμως να κατεβάσει όλα τα API's που θεωρεί απαραίτητα για την υλοποίηση καθώς και τα απαραίτητα αρχεία για την εικονική συσκευή που θα γίνει το debugging.

2.2 Διαδικασία Αποσφαλμάτωσης (Debugging)

Το κάθε περιβάλλον προγραμματισμού και δημιουργίας εφαρμογών Android περιλαμβάνει κάποια εργαλεία για την αποσφαλμάτωση της εφαρμογής. Ο προγραμματιστής επιλέγει πως θα υλοποιήσει την εφαρμογή αλλά το λειτουργικό σύστημα θα καθορίσει ποιους πόρους θα χρησιμοποιήσει. Σε αυτήν την διαδικασία χρησιμοποιείται το Logcat το οποίο είναι ένας καταγραφέας για το οτιδήποτε συμβαίνει στην εφαρμογή σου και στους πόρους του συστήματος που χρησιμοποιεί. Είναι υπεύθυνο για να ενημερώσει τον προγραμματιστή για ότι προκύψει (μνήμη που χρησιμοποιεί η εφαρμογή, συντακτικά και λογικά λάθη στον κώδικα, σφάλματα στο μητρώο, παραβίαση αδειών, παραβίαση δεδομένων της εφαρμογής και άλλα). Επίσης ένα άλλο σημαντικό εργαλείο που χρησιμοποιούν οι προγραμματιστές είναι το DDMS το οποίο παρέχει την δυνατότητα τεσταρίσματος της εφαρμογής σε εικονικές συσκευές που δεν έχουν πρόσβαση σε ζωντανά δεδομένα.

Αν πούμε ότι για παράδειγμα φτιάχνουμε μια εφαρμογή ανταλλαγής μηνυμάτων κειμένου και στατικών εικόνων πριν την δοκιμάσουμε σε κανονική συσκευή και μας επιφέρει πρόσθετες χρεώσεις, μόνο για τις δοκιμές μπορούμε να στέλνουμε fake notifications ή και μηνύματα στην εικονική συσκευή με στόχο την βελτίωση. Το ίδιο γίνεται και με την γεωγραφική θέση, κλήση αριθμού, προβολή website, προβολή στατιστικών heap memory allocation, thread memory size, network statistics και άλλα.

2.3 Δημοσίευση Εφαρμογής

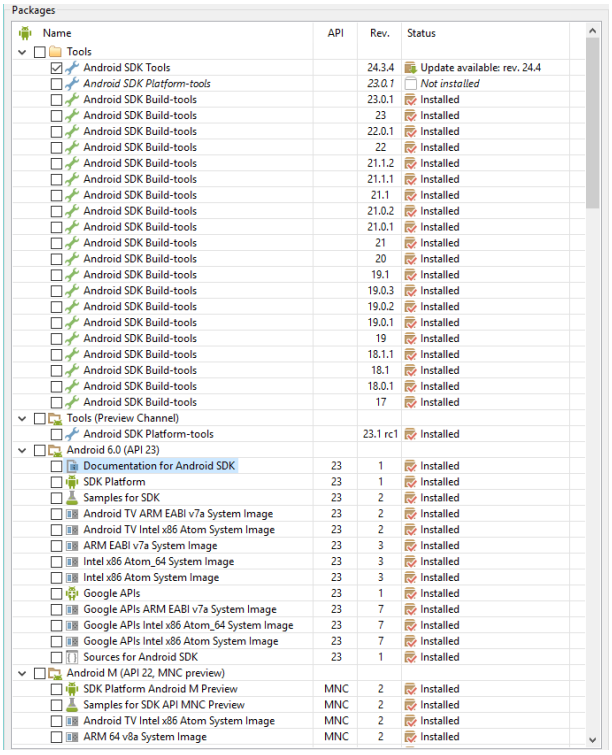
Για να δημοσιεύσουμε μια εφαρμογή χρειαζόμαστε το .apk αρχείο που θα μας δώσει ο Compiler εφόσον περάσει επιτυχώς την διαδικασία της αποσφαλμάτωσης. Αν θέλουμε να δημοσιεύσουμε την εφαρμογή μας στο Google Play Store θα πρέπει να κάνουμε εγγραφή σαν προγραμματιστές και στην συνέχεια δίνοντας μια περιγραφή και μερικά screenshots η εφαρμογή μας δημοσιεύεται και είναι διαθέσιμη για κατέβασμα από τον χρήστη.

Την στιγμή που γράφουμε την παρούσα πτυχιακή εργασία δεν είναι διαθέσιμη η εγγραφή εμπόρου στην Ελλάδα δηλαδή οι προγραμματιστές εφόσον καταβάλλουν τα 25$ που είναι το registration fee της Google για Developer registration θα μπορούν να δημοσιεύουν εφαρμογές αλλά θα διατίθενται δωρεάν στο ευρύ κοινό. Αυτό σημαίνει ότι δεν θα μπορούν να έχουν κέρδος από τα downloads της εφαρμογής αλλά μόνο από τα in-app purchases (για παράδειγμα η αναβάθμιση μια εφαρμογής από trial σε pro έκδοση μέσω paypal ή google wallet) ή το admob (διαδικασία πληρωμής από διαφημίσεις της Google μέσα στην εφαρμογή).

2.4 Android System Development Kit

Για να μπορέσει ο προγραμματιστής να δημοσιεύσει μια εφαρμογή η οποία θα είναι διαθέσιμη στο κοινό θα πρέπει να χρησιμοποιήσει κάποιες βιβλιοθήκες και εργαλεία που είναι διαθέσιμα μέσω του Android SDK. Επίσης θα μπορεί να προβεί σε διαθέσιμες ενημερώσεις ανάλογα με τις εκδόσεις του λειτουργικού συστήματος. Μπορεί να κατεβάσει τα απαραίτητα αυτά εργαλεία από τον SDK Manager που είναι ένα built-in επέκταση σε κάθε περιβάλλον προγραμματισμού που θα επιλέξει να χρησιμοποιήσει.

Το πως είναι διαθέσιμα τα πακέτα ανάλογα με τα API's του μπορούμε να δούμε στην παρακάτω εικόνα:

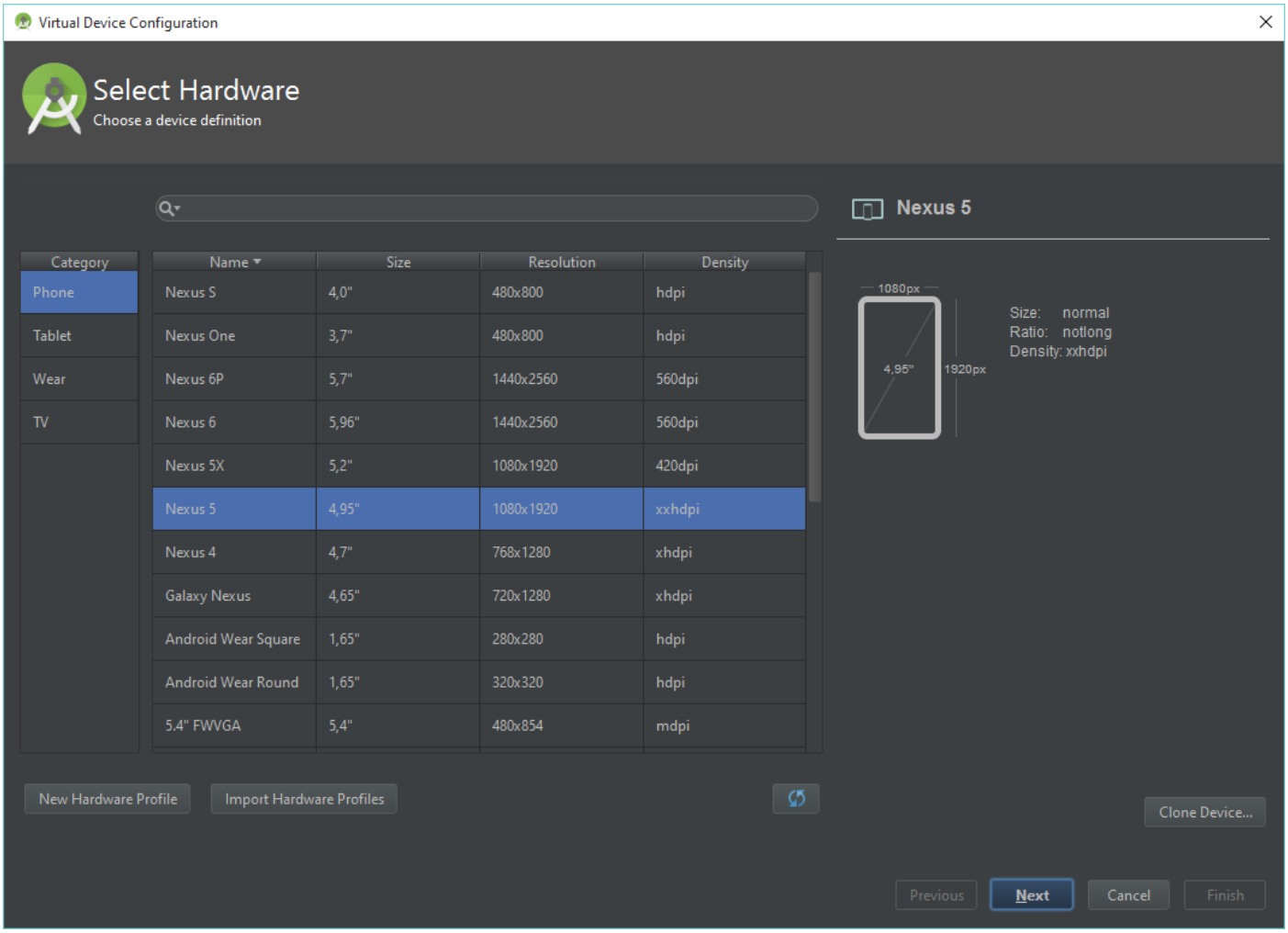


Εικόνα 23: Android SDK Manager

2.5 Android Virtual Device Manager (AVD)

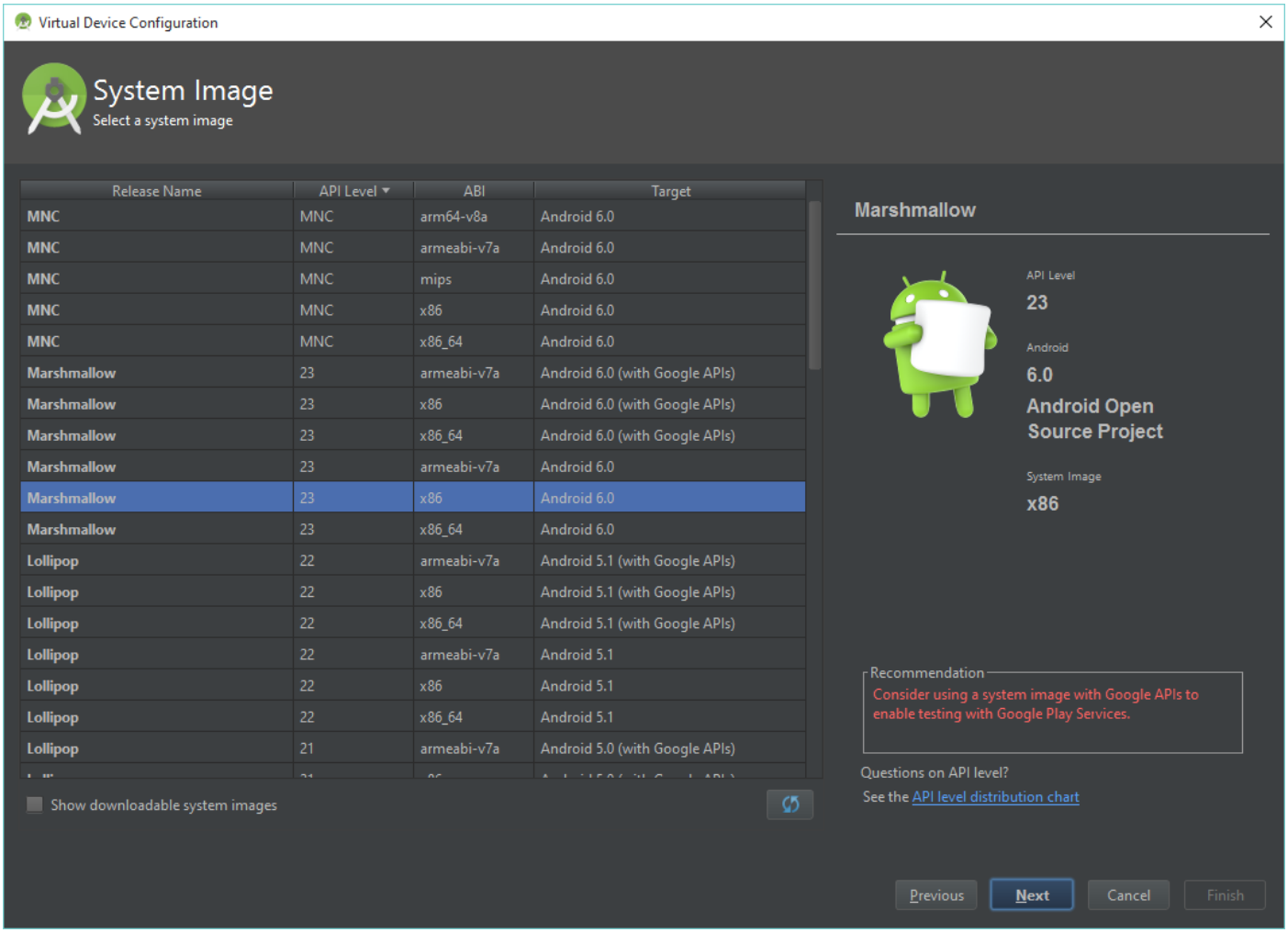
Επίσης θα πρέπει να κατεβάσουμε όλα τα απαραίτητα αρχεία για την δημιουργία εικονικής συσκευής με σκοπό να τεστάρουμε την εφαρμογής μας. Δουλεία η οποία αναλαμβάνει το AVD Android Virtual Device Manager με το κατάλληλο SDK και έκδοση του λειτουργικού συστήματος και με τον κατάλληλο επεξεργαστή. Βεβαίως θα πρέπει να έχουμε εγκατεστημένο το API που θέλουμε να τρέξει και να διαλέξουμε τις ρυθμίσεις οθόνης, μέγεθος μνήμης, χρησιμοποίηση της κάρτας γραφικών του HOST για καλύτερες επιδόσεις ιδιαίτερα σε εφαρμογές με μέγιστη απόδοση γραφικών ή σε παιχνίδια. Παρακάτω θα δούμε πως είναι το AVD:

Διαλέγουμε την συσκευή από την λίστα



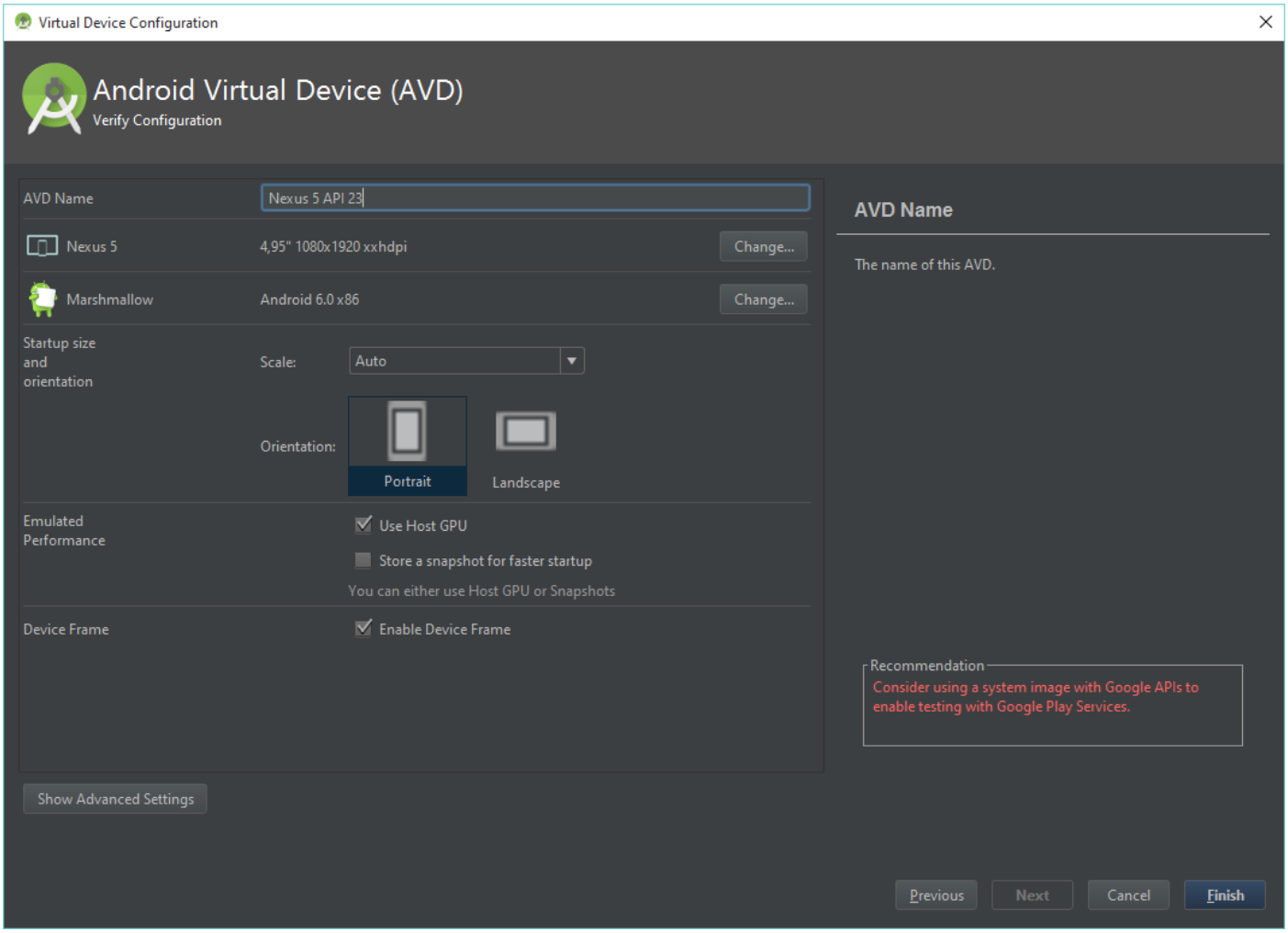
Εικόνα 24: AVD Manager Device Selection

Επιλέγουμε το SDK και την έκδοση καθώς και τον επεξεργαστή που θέλουμε να τρέχει η εικονική συσκευή μας:



Εικόνα 25: SDK and Processor Selection

Επιθεώρηση και δημιουργία της εικονικής συσκευής:



Εικόνα 26: AVD inspection and Creation

2.6 Android Design Guidelines

Βασιζόμενοι στο https://developer.android.com/design/index.html θα προσπαθήσουμε να ακολουθήσουμε τις υποδοχές για τον σχεδιασμό και υλοποίηση με βάση τις προτάσεις της εταιρίας που κατασκεύασε το λειτουργικό σύστημα. Βεβαίως μπορούμε να χρησιμοποιήσουμε τις δικές μας ιδέες απλώς ο σχεδιασμός μιας κακοσχεδιασμένης εφαρμογής θα απαιτεί πολύ περισσότερους πόρους στην συσκευή με σκοπό μια πολύπλοκη εφαρμογή να διακοπεί από το σύστημα εάν δεν είναι διαθέσιμοι οι πόροι η ακόμα και περισσότερο χρόνο αποσφαλμάτωσης για τυχόν λάθη.

Θα πρέπει να αποφασίσουμε αν η εφαρμογή μας θα καλύψει παλαιότερες εκδόσεις του λειτουργικού συστήματος, αν θα υποστηρίζει πολλαπλές συσκευές όπως(smartphones, tablets, tv, wearable) με σκοπό τον σχεδιασμό εικόνων και γραφικών για πολλά μεγέθη οθονών. Οι εταιρίες πάντα θα βγάζουν συσκευές μεγαλύτερων αναλύσεων έτσι εάν θέλουμε η εφαρμογή μας να καλύπτει ένα μεγάλο ποσοστό συσκευών θα πρέπει να αναβαθμίζουμε συνεχώς την εφαρμογή.

Όταν θέλουμε να σχεδιάσουμε εφαρμογή που να υποστηρίζει πολλές συσκευές θα πρέπει να συμπεριλάβουμε υπόψιν τα διάφορα μεγέθη τα οποία φαίνονται παρακάτω:

Density Bucket	Screen Density	Physical Size	Pixel Size
ldpi	120 dpi	0.5 x 0.5 in	0.5 in * 120 dpi = 60 x 60 px
mdpi	160 dpi	0.5 x 0.5 in	0.5 in * 160 dpi = 80 x 80 px
hdpi	240 dpi	0.5 x 0.5 in	0.5 in * 240 dpi = 120 x 120 px
xhdpi	320 dpi	0.5 x 0.5 in	0.5 in * 320 dpi = 160 x 160 px
xxhdpi	480 dpi	0.5 x 0.5 in	0.5 in * 480 dpi = 240 x 240 px

Εικόνα 27: Διαφορές σε διάφορα μεγέθη οθονών

Υπάρχουν διάφορες μονάδες μέτρησης μεγέθους οθόνης όπως είναι:

- Px Είναι το pixel.
- Sp Είναι scale-independent pixels.
- Dip Είναι density-independent pixels.
- Inch Είναι η ίντσα και αντιστοιχεί σε πραγματικές διαστάσεις οθόνης 1inch = 2.54 cm.
- Pt Είναι points και αντιστοιχεί σε πραγματικές διαστάσεις οθόνης 1pt= 1/72 inch.
- Dp Είναι density-independent pixels αλλά είναι μια abstract μονάδα μέτρησης για το μέγεθος. 1dp = 1px in 160 dpi screen.

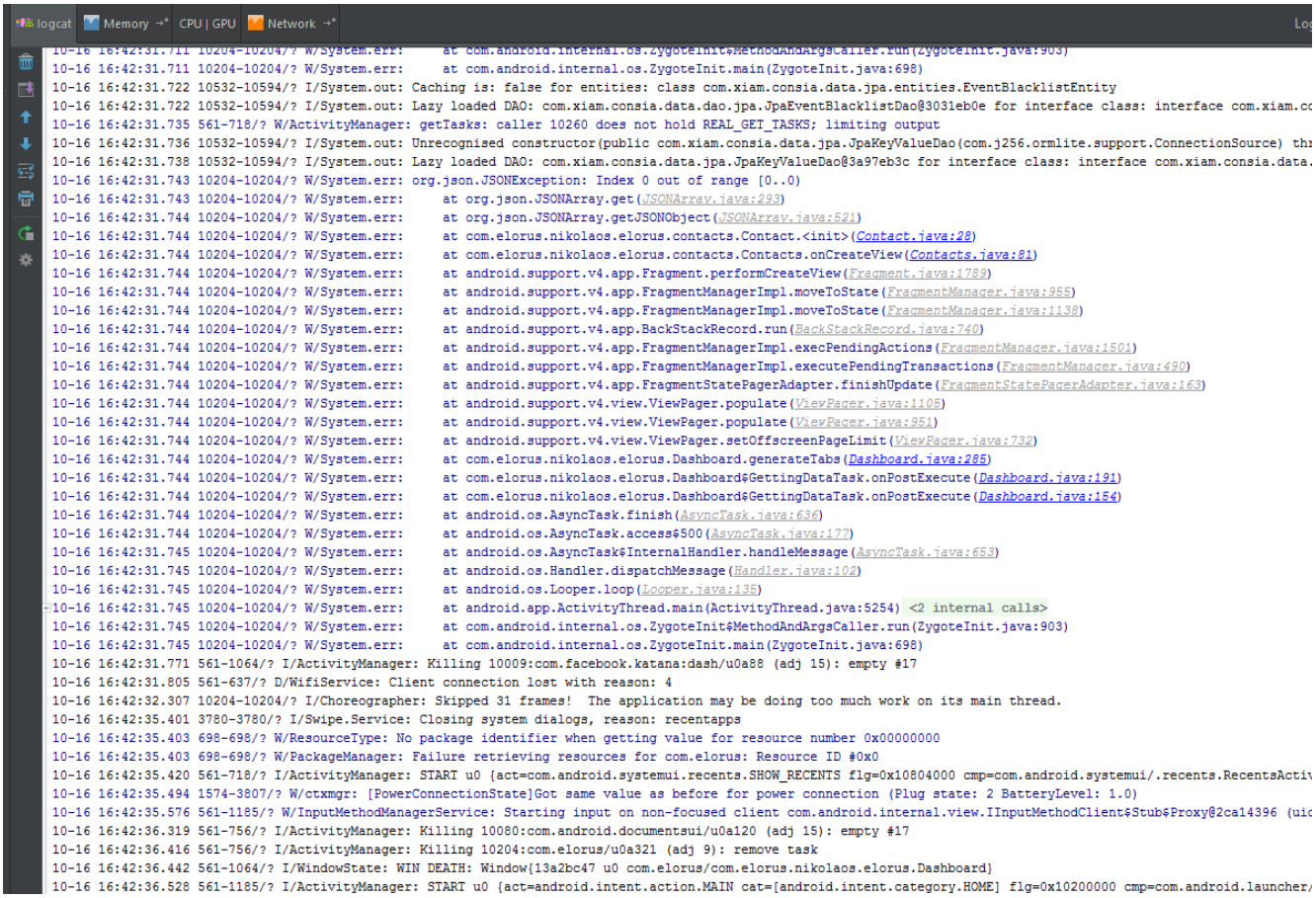
Παρακάτω θα δούμε ένα συνοπτικό πίνακα που εξηγεί καλύτερα τα μεγέθη των διαφόρων οθονών:

Dimension	Description	Units / Physical Inch	Density Independent	Same Physical Size on Every Screen
px	Pixels	Varies	No	No
in	Inches	1	Yes	Yes
mm	Millimeters	25.4	Yes	Yes
pt	Points	72	Yes	Yes
dp	Density independent pixels	~160	Yes	No
sp	Scale independent pixels	~160	Yes	No

Εικόνα 28: Αντιστοιχία μονάδων μέτρησης για μεγέθη οθονών

2.7 Android Logcat – Android Crash Report Manager

Ένα από τα βασικότερα εργαλεία μέσα στα περιβάλλοντα προγραμματισμού είναι το Android Logcat. Παρέχει πληροφορίες για το οτιδήποτε γίνεται μέσα στην εφαρμογή καθώς και τους πόρους, μνήμη, χώρο που καταλαμβάνει στην συσκευή. Επίσης μας παρέχει πληροφορίες για το εάν η εφαρμογή διακοπεί κατά την διάρκεια της αποσφαλμάτωσης κατά την διάρκεια του debugging καθώς και για τον λόγο διακοπής. Μας ενημερώνει για τα permissions που δίνουμε στην εφαρμογή καθώς και για το αν το thread μας καταλαμβάνει πάρα πολύ δουλειά παρόλο που εκτελείται επιτυχώς. Μπορούμε να δούμε το αποτέλεσμα του Logcat με μηνύματα verbose, info, warning και άλλα. Μια οθόνη Logcat είναι η παρακάτω:



Εικόνα 29: Logcat Output

Μπορούμε να καταγράφουμε τα δικά μας δεδομένα για αποσφαλμάτωση χρησιμοποιώντας τις παρακάτω εντολές

- Log.v("INTENT_TAG", "Intent Successfull"); για να δούμε αποτέλεσμα εάν το intent ήταν επιτυχές σε verbose level
- Log.d("INTENT_TAG", "Intent Successfull"); για να δούμε αποτέλεσμα εάν το intent ήταν επιτυχές σε debug level
- Log.e("INTENT_TAG", "Intent Successfull"); για να δούμε αποτέλεσμα εάν το intent ήταν επιτυχές σε error level

- Log.i("INTENT_TAG", "Intent Successfull"); για να δούμε αποτέλεσμα εάν το intent ήταν επιτυχές σε info level
- Log.w("INTENT_TAG", "Intent Successfull"); για να δούμε αποτέλεσμα εάν το intent ήταν επιτυχές σε warning level

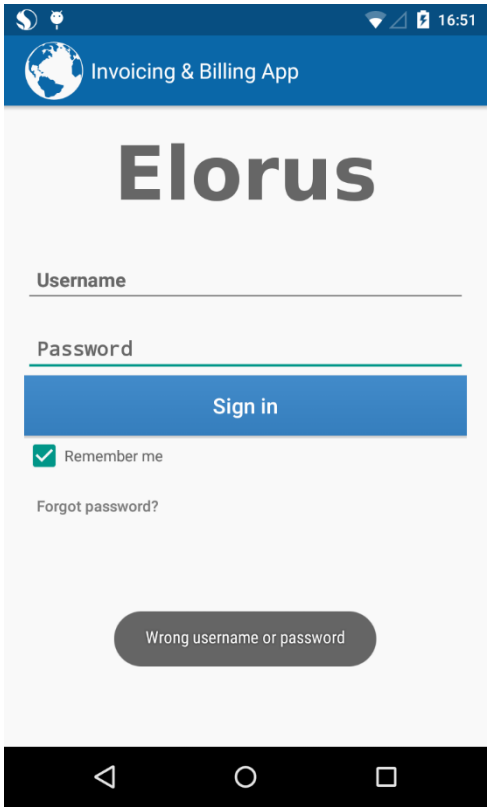
Συνήθως τα χρησιμοποιούμε σε Exception Handlers και Catch Statements.

2.8 Notifications

Πολύ σημαντικό είναι για ένα προγραμματιστή να μπορεί να ενημερώνει τον χρήστη για κάποια λογικά ή συντακτικά λάθη με βάση την ορθή χρήση της εφαρμογής. Αυτό μπορεί να επιτευχθεί με πολλούς τρόπους 3 εξ αυτών είναι οι πιο διαδεδομένοι.

- Toast Message Πρόκειται για floating μηνύματα που προσπαθούν να βελτιώσουν την ορθή λειτουργία της εφαρμογής. Ένα παράδειγμα υποδομής ενός τέτοιου μηνύματος είναι:

```
Toast.makeText(getApplicationContext(), "Wrong username or password", Toast.LENGTH_LONG).show();
```

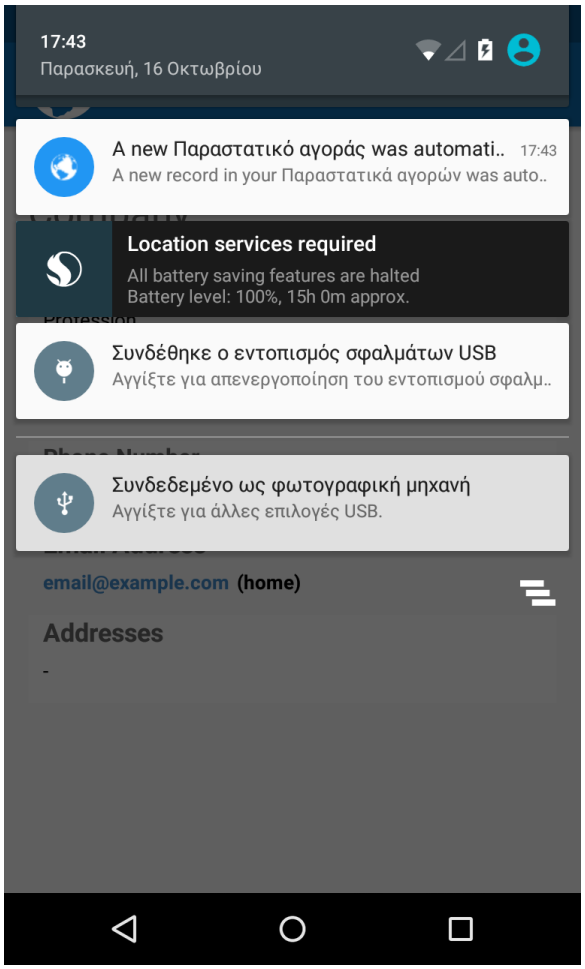


Εικόνα 30: Toast Message

- Push Notifications Χρησιμοποιούνται για την ενημέρωση του χρήστη έτσι ώστε να μην χρειάζεται να ελέγχει την εφαρμογή καθημερινά. Με κώδικα υλοποιείται ως εξής:

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(context)
        .setSmallIcon(getNotificationIcon()) // Set Icon
        .setColor(color) // Set Color
        .setTicker(message) // Set Ticker Message
        .setContentTitle(title) // Set Title
        .setContentText(message) // Set Text
        .setContentIntent(pIntent) // Set PendingIntent into Notification
        .setAutoCancel(false) // Dismiss Notification
        .setDefaults(Notification.DEFAULT_ALL);
        NotificationManager notificationManager = (NotificationManager) context.getSystemService(Context.NOTIFICATION_SERVICE);
        notificationmanager.notify(i, builder.build());
```

Ενώ ο χρήστης βλέπει το τελικό αποτέλεσμα ως εξής:

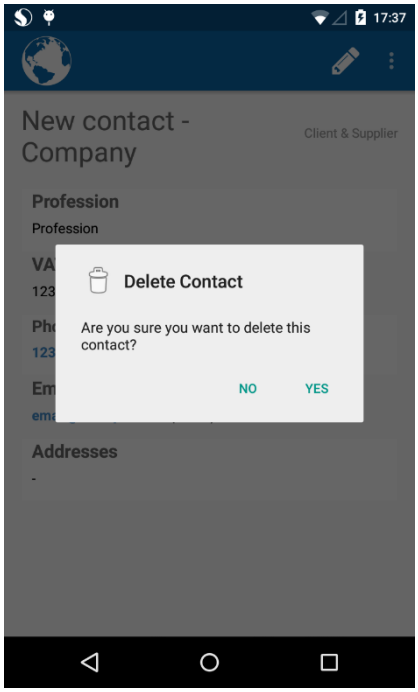


Εικόνα 31: Notification Message

- Alert Dialog Χρησιμοποιείται όταν θέλουμε να εμφανίσουμε μια πληροφορία η ένα μήνυμα λάθους σε μια φόρμα. Με κώδικα υλοποιείται ως εξής:

```
new AlertDialog.Builder(this, R.style.MyAlertDialogStyle)
        .setTitle("Delete Contact")
        .setMessage("Are you sure you want to delete this contact?")
        .setPositiveButton(R.string.yes, new DialogInterface.OnClickListener() {
        @Override
                public void onClick(DialogInterface dialog, int which) {
                 try {
                        deleteContact();  // Do something
                        dialog.cancel();
                }
                catch (MalformedURLException e) {
                         e.printStackTrace();
                } } })
        .setNegativeButton(R.string.no, new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) { dialog.cancel(); } })
        .setIcon(android.R.drawable.ic_menu_delete) .show();
```

Ο χρήστης βλέπει το τελικό αποτέλεσμα ως εξής:



Εικόνα 32: Alert Dialog

2.9 Developing in Android from Programmer View

Παραπάνω αναλύσαμε όλα αυτά τα Guidelines που χρησιμοποιεί ένας προγραμματιστής προκειμένου να μην χάσει ένα μεγάλο μέρος χρηστών διότι αυτοί μπορεί να είναι πιθανοί πελάτες του. Δεν μπορεί να αγνοήσει όλα τα μεγέθη οθόνης γιατί είναι μέρος των χρηστών που απευθύνεται μιας και δεν έχουν όλοι οι χρήστες την οικονομική δυνατότητα να αγοράζουν καινούργιες συσκευές με στόχο να καταργούνται κάποια API's. Βέβαια αυτό μπορεί να αποβεί ασύμφορο για έναν προγραμματιστή, πόσο μάλλον μια εταιρία όπως είναι η Killo Games στην περίπτωση του "Subway Surfers" ενός παιχνιδιού που διατίθεται δωρεάν και περιλαμβάνει in-app purchases για πιο γρήγορο gameplay και λιγότερο ανταγωνισμό. Βέβαια δείτε το λίγο και από την μεριά της εταιρίας, ότι θα πρέπει να αποσύρει κάποιες εκδόσεις αλλιώς θα προβεί σε πτώχευση έτσι όπως εξελίσσεται το λειτουργικό σύστημα.

Το ερώτημα το οποίο πρέπει να απασχολεί κάθε προγραμματιστή είναι "Πόσο εύκολο είναι να δημιουργηθεί μια εφαρμογή σε μια πλατφόρμα?". Ας υποθέσουμε ότι είστε χρήστης των Windows και θα πρέπει απλώς να κατεβάσετε το JDK, έπειτα ένα περιβάλλον προγραμματισμού Eclipse, Android Studio, Netbeans, Intellij Idea, να κατεβάσετε τα απαραίτητα API's, και τα απαραίτητα αρχεία που θα πρέπει να δημιουργήσουν την εικονική συσκευή. Εννοείτε βέβαια ότι το Android παρέχει αρκετό documentation και μια μεγάλη online κοινότητα καθώς και διάφορα φόρουμ όπου θα προσφερθεί βοήθεια όπως το Stack Overflow, Android Enthusiasts, Super User καθώς και blogs με διάφορα tutorials που πλέον παρέχουν κάθε λύση σε κάθε πρόβλημα ή ακόμα και ένα παράδειγμα πολύ κοντά σε αυτό που ψάχνουμε.

Αυτό που αντιμετωπίζει συχνά ένας προγραμματιστής είναι οι κριτικές που δέχονται οι εφαρμογές του και οι πιο πολλές περιλαμβάνουν λέξεις όπως «διακόπηκε», «δεν δουλεύει», «πολλές διαφημίσεις» και άλλα. Για την επιτυχία μιας εφαρμογής πρέπει να παρθούν οι σωστές αποφάσεις στον σωστό χρόνο. Ο προγραμματιστής πρέπει να λάβει υπόψιν ότι μια εφαρμογή δεν γίνεται έτσι απλά να επιτύχει απλώς βεβαιώνοντας τον ότι δουλεύει. Οι χρήστες θα προτιμήσουν εφαρμογές που έχουν καλό performance, low battery consumption, κατανάλωση λιγότερων πόρων.

Η κάθε εφαρμογή είναι ένα Software. Στην αγορά υπάρχουν πολλές εφαρμογές που μπορούν να κάνουν την δικιά σου εφαρμογή να συμπεριφέρεται λανθασμένα. Ένας λόγος που το λειτουργικό σύστημα Android είναι διαδεδομένο είναι ότι βρίσκεται πολύ πιο μπροστά από το iOS and Windows Mobile, παρέχεται δωρεάν η πλατφόρμα ανάπτυξης εφαρμογών σε αντίθεση με τα άλλα 2 που έχουν ετήσια συνδρομή μόνο για να ανάπτυξη εφαρμογής 99$ ετησίως.

Κεφάλαιο 3

3.1 Τί είναι το REST (Representational state transfer)

Στην πληροφορική, το Representational State Transfer (REST) είναι το λογισμικό αρχιτεκτονικής του World Wide Web. Το REST δίνει ένα συντονισμένο σύνολο περιορισμών στον σχεδιασμό των περιεχόμενων σε ένα κατανεμημένο σύστημα υπερμέσων, που μπορεί να οδηγήσει σε μία υψηλότερης απόδοσης και πιο ισορροπημένης αρχιτεκτονικής σύστημα.

Στο βαθμό που τα συστήματα συμμορφώνονται με τους περιορισμούς του REST, μπορούν να ονομαστούν RESTful. Τα RESTful συστήματα συνήθως, αλλά όχι πάντα, επικοινωνούν μέσω του πρωτοκόλλου Hypertext Transfer Protocol με τα HTTP (GET, POST, PUT, DELETE, κ.λπ.) που χρησιμοποιούν προγράμματα περιήγησης στο διαδίκτυο για την ανάκτηση ιστοσελίδων και την αποστολή δεδομένων σε απομακρυσμένους διακομιστές. Το πρωτόκολλο REST διασυνδέεται με εξωτερικά συστήματα χρησιμοποιώντας τους πόρους που προσδιορίζονται από τα URI, για παράδειγμα ***/people/Tom***, η οποία μπορεί να λειτουργήσει κατά τη χρήση συνηθισμένων ρημάτων, όπως ***DELETE /people/Tom***.

Το REST προτάθηκε αρχικά από τον Roy Thomas Fielding στην 2000 διδακτορική διατριβή του «Architectural Styles and the Design of Network-based Software Architectures". Ο Fielding ανέπτυξε την υπόλοιπη αρχιτεκτονική παράλληλα με το HTTP 1.1 1996/1999, βασιζόμενος στον υπάρχουσα σχεδιασμό του HTTP 1.0 το 1996.

3.2 Ιδιότητες Αρχιτεκτονικής

Οι αρχιτεκτονικές ιδιότητες που επηρεάζονται από τους περιορισμούς της αρχιτεκτονικής του REST είναι:

- *Επιδόσεις*: Οι Αλληλεπιδράσεις των συστατικών του μπορεί να είναι ο κυρίαρχος παράγοντας για την απόδοση του δικτύου όπως αντιλαμβάνονται οι χρήστες την αποτελεσματικότητα του δικτιού.
- *Κλιμάκωση* για την υποστήριξη μεγάλου αριθμό συστατικών και αλληλεπιδράσεων μεταξύ αυτών. Ο Roy Fielding , ένας από τους κύριους συντάκτες του HTTP , περιγράφει την επίδραση του REST για επεκτασιμότητα έτσι :

> REST's client–server separation of concerns simplifies component implementation, reduces the complexity of connector semantics, improves the effectiveness of performance tuning, and increases the scalability of pure server components. Layered system constraints allow intermediaries—proxies, gateways, and firewalls—to be introduced at various points in the communication without changing the interfaces between components, thus allowing them to assist in communication translation or improve performance via large-scale, shared caching. REST enables intermediate processing by constraining messages to be self-descriptive: interaction is stateless between requests, standard methods and media types are used to indicate semantics and exchange information, and responses explicitly indicate cacheability.

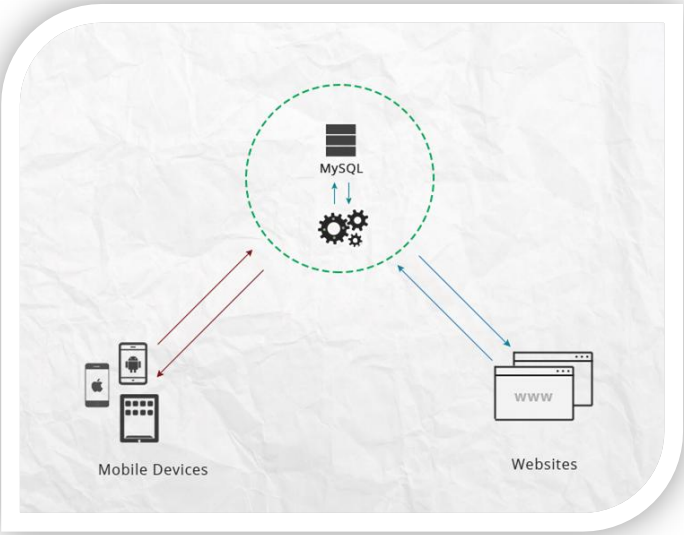
- *Απλότητα* των διεπαφών.
- *Δυνατότητα τροποποιήσεως* των συστατικών του για να ανταποκρίνεται στις μεταβαλλόμενες ανάγκες (ακόμη και όταν η εφαρμογή εκτελείται).
- *Προβολή* της επικοινωνίας μεταξύ των συστατικών από μέσα της υπηρεσίας.
- *Φορητότητα* των συστατικών του με την μετακίνηση του κώδικα προγραμματισμού με τα δεδομένα.
- *Αξιοπιστία* είναι η αντίσταση της αποτυχίας σε επίπεδο συστήματος στην παρουσία των αποτυχιών των συστατικών, υποδοχών , ή δεδομένων.

3.3 Αρχιτεκτονική συστήματος

Οι αρχιτεκτονικές ιδιότητες του REST γίνονται με την εφαρμογή ειδικών περιορισμών αλληλεπιδράσεων με στοιχεία , συνδέσεις και στοιχεία δεδομένων. Μπορεί κανείς να χαρακτηρίσει τις εφαρμογές που χρησιμοποιούν το REST, που περιγράφονται στο παρόν τμήμα, ως «RESTful» . Εάν μια υπηρεσία παραβεί οποιαδήποτε από τους απαιτούμενους περιορισμούς , δεν μπορεί να θεωρηθεί RESTful . Η συμμόρφωση με αυτούς τους περιορισμούς , και, συνεπώς, σύμφωνα με την αρχιτεκτονική του REST, επιτρέπει σε κάθε είδους κατανεμημένου συστήματος υπερμέσων να έχει επιθυμητές μη λειτουργικές ιδιότητες , όπως την απόδοση , την επεκτασιμότητα , την απλότητα , τη δυνατότητα τροποποιήσεως , την προβολή , τη φορητότητα και την αξιοπιστία.

Οι επίσημοι περιορισμοί του REST είναι :

- Client-Server
- Stateless
- Cashable
- Layered system
- Code on demand
- Uniform interface:
  - Identification of resources
  - Manipulation of resources
  - Self-descriptive messages
  - Hypermedia as the engine of application state



Εικόνα 33: αρχιτεκτονική του REST

3.3.1 Client-Server

Μια ενιαία διεπαφή χωρίζει τους πελάτες από τους διακομιστές. Αυτός ο διαχωρισμός σημαίνει ότι, για παράδειγμα , οι πελάτες δεν ασχολούνται με την αποθήκευση δεδομένων , η οποία παραμένει στο εσωτερικό κάθε διακομιστή , έτσι ώστε η δυνατότητα μεταφοράς του κώδικα του πελάτη να βελτιώνεται. Οι διακομιστές δεν ασχολούνται με το περιβάλλον εργασίας χρήστη ή την κατάσταση του χρήστη, ώστε οι διακομιστές να είναι απλούστεροι και πιο επεκτάσιμοι.
Οι διακομιστές και οι πελάτες μπορούν επίσης να αντικατασταθούν και να αναπτυχθούν ανεξάρτητα, όσο η διασύνδεση μεταξύ τους δεν μεταβάλλεται.

3.3.2 Stateless

Η επικοινωνία πελάτη - διακομιστή δεν περιορίζεται περαιτέρω από κάποιο πλαίσιο πελάτης που είναι αποθηκευμένο στο διακομιστή μεταξύ των αιτημάτων . Κάθε αίτηση από οποιονδήποτε πελάτη περιέχει όλες τις απαραίτητες πληροφορίες για την εξυπηρέτηση του αιτήματος, η κατάσταση του session διεξάγεται στον πελάτη. Η κατάσταση περιόδου μπορεί να μεταφερθεί από το διακομιστή σε άλλη υπηρεσία , όπως μια βάση δεδομένων για να διατηρήσει μια επίμονη κατάσταση για μια περίοδο και να επιτρέψει τον έλεγχο ταυτότητας. Ο πελάτης ξεκινά την αποστολή των αιτήσεων όταν είναι έτοιμοι να κάνουν τη μετάβαση σε μια νέα κατάσταση .

Ενώ μία ή περισσότερες αιτήσεις είναι εκκρεμείς , ο πελάτης θεωρείται ότι είναι σε μεταβατικό στάδιο. Η εκπροσώπηση της κάθε κατάστασης της εφαρμογής περιέχει συνδέσμους που μπορούν να χρησιμοποιηθούν την επόμενη φορά που ο πελάτης επιλέγει να ξεκινήσει μια νέα κατάσταση μετάβασης.

3.3.3 Cashable

Όπως στο World Wide Web, οι πελάτες και οι μεσάζοντες μπορεί αποθηκεύσουν απαντήσεις. Οι απαντήσεις πρέπει , ως εκ τούτου , ρητά ή σιωπηρά , να αυτό-προσδιορίζονται ως cacheable , ή όχι , για να αποτρέψει τους πελάτες από την επαναχρησιμοποίηση παλιών ή ακατάλληλων δεδομένων σε απάντηση περαιτέρω αιτημάτων. Καλή Διαχείριση caching, εν μέρει ή εντελώς, εξαλείφει κάποιες αλληλεπιδράσεις του πελάτη - εξυπηρετητή, βελτιώνοντας περαιτέρω την επεκτασιμότητα και την απόδοση.

3.3.4 Layered system

Ένας πελάτης δεν μπορεί να καταλάβει αν συνδέεται άμεσα με το διακομιστή, ή σε ενδιάμεσο διακομιστή. Οι ενδιάμεσοι διακομιστές μπορούν να βελτιώσουν την επεκτασιμότητα του συστήματος με την ενεργοποίηση της εξισορρόπησης φορτίου και με την παροχή shared cashes. Μπορούν επίσης να επιβάλουν τις πολιτικές ασφάλειας.

3.3.5 Code on demand

Οι διακομιστές μπορούν να παρατείνουν προσωρινά ή να προσαρμόσουν τη λειτουργικότητα ενός πελάτη από τη μεταφορά του εκτελέσιμο κώδικα. Παραδείγματα για αυτό μπορεί να περιλαμβάνουν compiled components, όπως μικροεφαρμογές Java και client-side scripts όπως JavaScript .Η "Code on demand" είναι η μόνη προαιρετική περιοριστική της αρχιτεκτονικής του REST.

3.3.6 Uniform interface

Ο περιορισμός μιας ομοιόμορφης διεπαφής είναι θεμελιώδους σημασίας για τον σχεδιασμό μιας υπηρεσίας REST. Η ομοιόμορφη διεπαφή απλοποιεί και αποσυνδέει την αρχιτεκτονική, η οποία επιτρέπει κάθε μέρος να εξελιχθεί ανεξάρτητα.

3.4 Εφαρμογές σε Web Services

Τα APIs των Web Services που συμμορφώνονται με τους αρχιτεκτονικούς περιορισμούς του REST ονομάζονται RESTful APIs . Τα RESTful APIs που βασίζονται στο HTTP ορίζονται με αυτές τις πτυχές :

- Base URI όπως ***http://example.com/resources/***
- Ένα internet media type για τα δεδομένα. Πιο συχνά αυτό είναι το JSON αλλά μπορεί να είναι και άλλο media type (XML, Atom, microformats, images, etc).
- HTTP methods.
- Hypertext links to reference state.
- Hypertext links to reference-related resources.

3.4.1 Http methods

Ένα καλά σχεδιασμένο RESTful API υποστηρίζει τις μεθόδους Http με την πιο συχνή χρήση(GET, POST, PUT και DELETE). Υπάρχουν και άλλες Http μέθοδοι όπως OPTION, HEAD αλλά χρησιμοποιούνται πιο σπάνια. Κάθε μέθοδος πρέπει να χρησιμοποιείται εξαρτώμενη από τον τύπο της εφαρμογής που χρησιμοποιούμε.

Παρακάτω βλέπουμε τις μεθόδους του HTTP που εμπεριέχει ένα RESTful API:

Resource	GET	PUT	POST	DELETE
Collection URI, such as http://example.com/resources	**List** collection's members	**Replace** collection with another collection	**Create** new entry in the collection	**Delete** the entire collection
Element URI, such as http://example.com/resources/item17	**Retrieve** the member of the collection	**Replace** the member of the collection	Not generally used	**Delete** the member of the collection

Εικόνα 34: Μέθοδοι του Http

3.4.2 Http Status

Τα status codes του Http στο response body υποδεικνύει στην εφαρμογή του χρήστη για τι λειτουργία πρέπει να χρησιμοποιηθεί όσο αφορά το response. Για παράδειγμα ένα response code 200, σημαίνει ότι από την μεριά του server η αίτηση έχει επεξεργαστεί επιτυχώς και σαν απάντηση μπορούμε να περιμένουμε ανανέωση στα δεδομένα. Όπως ακόμα αν το status code είναι 401, η αίτηση δεν επιτρέπεται. Ένα παράδειγμα για να πάρουμε status code 401 είναι όταν έχουμε βάλει εσφαλμένο API κλειδί. Τα πιο συνηθισμένα status codes είναι:

Code	Description	Code	Description
200	OK	400	Bad Request
201	Created	401	Unauthorized
202	Accepted	403	Forbidden
301	Moved Permanently	404	Not Found
303	See Other	410	Gone
304	Not Modified	500	Internal Server Error
307	Temporary Redirect	503	Service Unavailable

Εικόνα 35: Http status codes

3.4.3 URL Structure

Στην σχεδίαση του REST τα τερματικά URLs πρέπει να είναι ορθά δομημένα και εύκολα στην κατανόηση. Κάθε URL για κάθε πόρο πρέπει να αναγνωρίζεται μοναδικά. Αν το API χρειάζεται ένα κλειδί για να παρέχεται πρόσβαση, αυτό το κλειδί πρέπει να υπάρχει στα headers του Http αντί να το προσθέτουμε στο URL. Για παράδειγμα:

- GET **http://api.elorus-staging.com:/v1.0/contacts/id/** - Θα πάρουμε μια συγκεκριμένη επαφή.
- POST **http://api.elorus-staging.com:/v1.0/contacts** - Θα δημιουργήσει μια νέα επαφή.

3.4.4 API Versioning

Υπάρχει τεράστια συζήτηση στις εκδόσεις των API. Αν θα υποστηρίζουν τα URL τις εκδόσεις των API ή αν θα υπάρχουν στα headers του Http. Αν και ενδείκνυται να περιλαμβάνονται στα request headers, είναι το ίδιο ασφαλής να τα χρησιμοποιούμε μέσα στο URL αφού είναι πολύ βολικό από την μεριά του client να «μεταναστεύει» από την μία έκδοση στην άλλη. Για παράδειγμα:

- **http://api.elorus-staging.com:/v1.0/contacts**
- **http://api.elorus-staging.com:/v2.0/contacts**

3.4.5 Content Type

Το content type στα Http headers καθορίζει το είδος των δεδομένων που πρέπει να σταλθούν μεταξύ στον server και τον client. Εξαρτάται από τα δεδομένα που υποστηρίζει το API το πώς χρειάζεται να ρυθμίσουμε το content type. Για παράδειγμα, το JSON Mime type πρέπει να είναι **Content-type:application/json**, για το XML είναι **Content-Type:application/xml**.

3.4.6 API key

Για την δημιουργία ενός ιδιωτικού API, όπου και χρειάζεται να υπάρχει αυστηρή πρόσβαση ή μείωση σε ιδιωτική πρόσβαση, η καλύτερη προσέγγιση είναι να ασφαλιστεί το API με ένα κλειδί. Ο κάθε χρήστης αναγνωρίζεται από το κλειδί που έχει στο API και όλες οι χρήσεις που είναι εφικτό να γίνουν, είναι για τους πόρους που ανήκουν σε αυτόν.

Το API key πρέπει να κρατιέται στο Authorization header της αίτησης, αντί να εμπεριέχεται στο URL.

Authorization: bf45c093e542f057caee68c47787e7d6

3.5 Android JSON Parsing

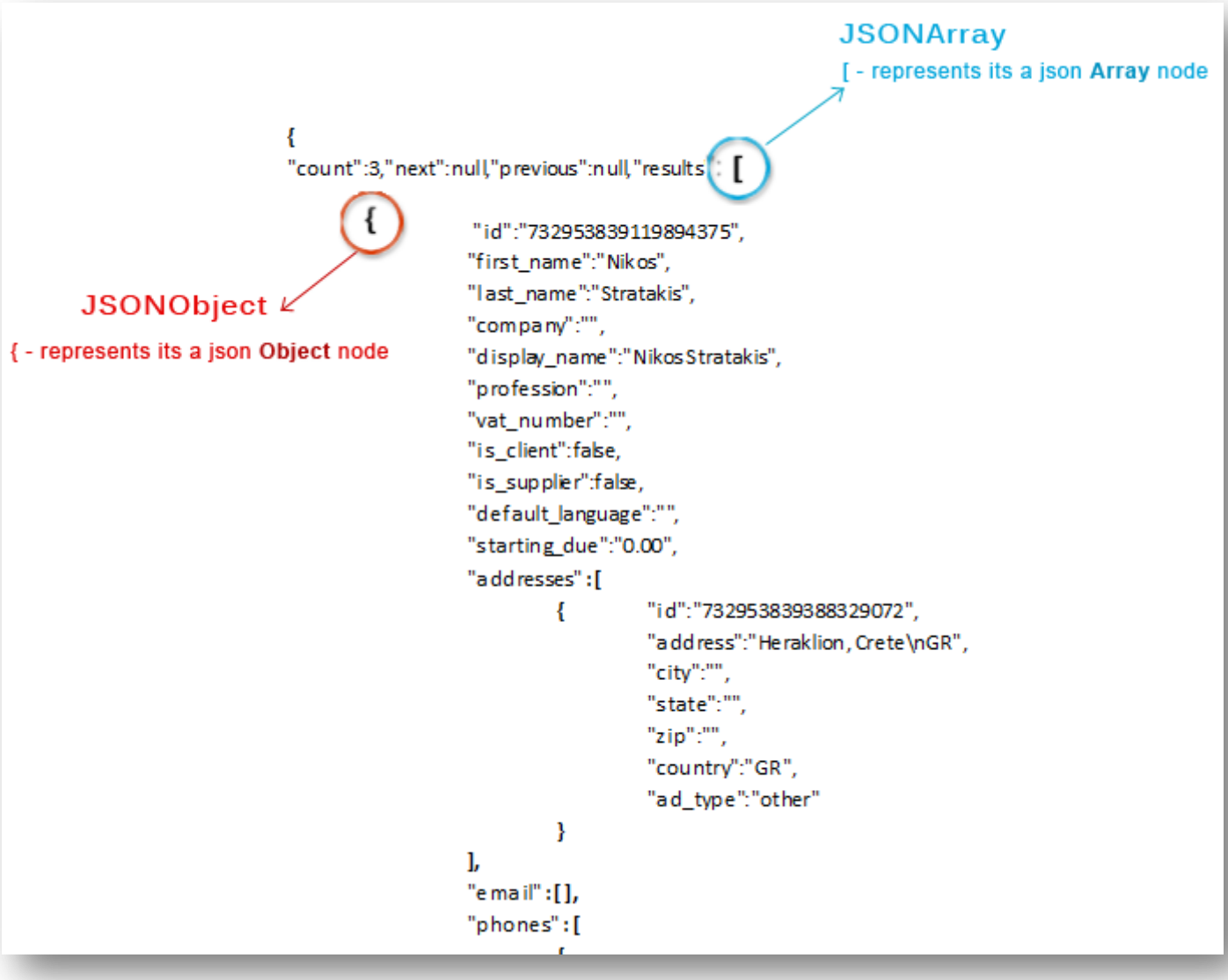
3.5.1 The Json Sample

Παρακάτω υπάρχει ένα δείγμα JSON που θα γίνει μία μικρή ανάλυση. Είναι ένα απλό JSON που μας δίνει μια λίστα από επαφές όπου κάθε κόμβος περιέχει στοιχεία επαφών όπως το όνομα, το email, την διεύθυνση, το τηλέφωνό του, κλπ.


```
{
"count":3,"next":null,"previous":null,"results" :    [
        {
                "id":"732953839119894375",
                "first_name":"Nikos",
                "last_name":"Stratakis",
                "company":"",
                "display_name":"Nikos Stratakis",
                "profession":"",
                "vat_number":"",
                "is_client":false,
                "is_supplier":false,
                "default_language":"",
                "starting_due":"0.00",
                "addresses" : [
                        {       "id":"732983839388329072",
                                "address":"Heraklion, Crete\nGR",
                                "city":"",
                                "state":"",
                                "zip":"",
                                "country":"GR",
                                "ad_type":"other"
                        }
                ],
                "email" : [ ],
                "phones" : [
                        {
                                "id":"732953839270888867",
                                "number":"6947137228",
                                "phone_type":"mobile"
                        }
                 ],
                "organization":"134450992913793052"
        },
        {
                "id":"732955124925132648",
                "first_name":"example first name",
                "last_name":"example last name",
                "company":"","display_name":"example company",
                "profession":""
                ……..
        }
}
```

3.5.2 Η διαφορά ανάμεσα στο «[» και στο «{» (Square brackets and Curly brackets)

Γενικά όλοι οι κόμβοι του JSON αρχίζουν με μια αγκύλη ή ένα άγκιστρο. Η διαφορά ανάμεσά τους είναι ότι η αγκύλη αναφέρεται σε έναν κόμβο **JSONArray** ενώ το άγκιστρο σε ένα **JSONObject**. Συνεπώς για να εισχωρήσουμε στους κόμβους αυτούς πρέπει να καλέσουμε και την κατάλληλη μέθοδο, ώστε να έχουμε πρόσβαση στα δεδομένα. Αν το JSON αρχίζει με το «[» πρέπει να χρησιμοποιήσουμε την getJSONArray() μέθοδο. Ενώ με το «{» την getJSONObject() μέθοδο.



Εικόνα 36: Definition of JSONObject and JSONArray

Κεφάλαιο 4

4.1 Ανάλυση απαιτήσεων για την δημιουργία της εφαρμογής

Ο σκοπός είναι η ανάπτυξη μιας εφαρμογής για την διαχείριση ρευστότητας και τον έλεγχο τιμολόγησης μικρών επιχειρήσεων. Θα χρησιμοποιήσουμε έναν RESTful API για την διαχείριση των δεδομένων και το documentation του για το πώς θα τα επεξεργαστούμε.

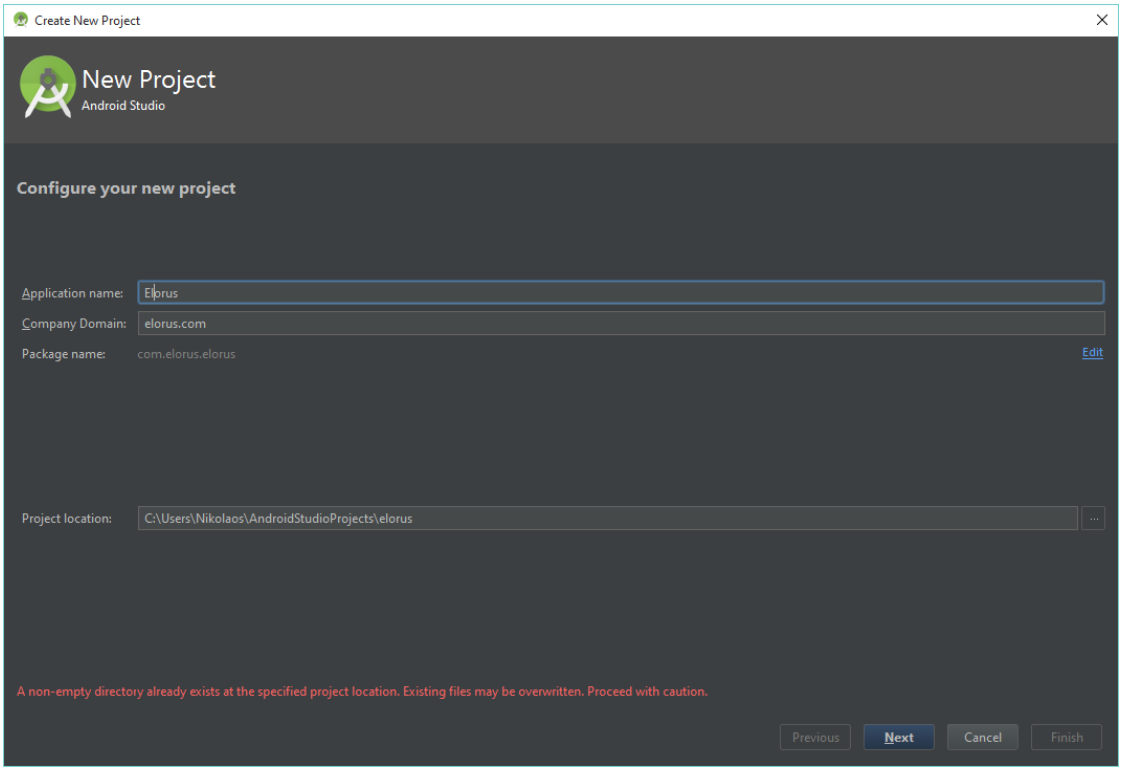
Η εφαρμογή αυτή θα πρέπει να επιβλέπει τις εισπράξεις και τις πληρωμές της επιχείρησης. Ο χρήστης της εφαρμογής θα βλέπει μια λίστα από τα παραπάνω, καθώς και θα έχει την επιλογή να δημιουργεί, να διαγράφει και να τις επεξεργάζεται τις πληρωμές και τις εισπράξεις.

Σε δεύτερο στάδιο, ο χρήστης πρέπει να έχει ένα τυπικό έλεγχο για τα παραστατικά και τα πιστωτικά παραστατικά των πωλήσεων και των αγορών όπως και τις προσφορές των πωλήσεων. Πιο συγκεκριμένα θα μπορεί να βλέπει μια λίστα από αυτά.

Τέλος υπάρχει η δυνατότητα των επαφών. Ο χρήστης θα μπορεί να βλέπει μια λίστα με τις επαφές-πελάτες του. Έχει την δυνατότητα να δημιουργήσει καινούργια επαφή μέσα από την εφαρμογή ή να ενσωματώσει όποια θέλει από τις επαφές της Android συσκευής του. Επίσης δίνεται η δυνατότητα να βλέπει τα απαραίτητα στοιχεία των επαφών όπως και η επιλογή της διαγραφής και της επεξεργασίας τους. Μια extra λειτουργία που παρέχεται είναι η αποστολή email μέσω της εφαρμογής, επιλέγοντας τον κατάλληλο email server, όπως και η λειτουργία να παίρνει τηλέφωνο την επιλεγόμενη επαφή, αν ο αριθμός είναι έγκυρος.

4.2 Δημιουργία Project στο Android Studio

Ξεκινάμε δημιουργώντας το καινούριο μας project στο android studio, πατώντας Create New Project όπως βλέπουμε παρακάτω:



Εικόνα 37: Δημιουργία νέου project

Στην συνέχεια επιλέγουμε τα SDK και την πλατφόρμα που θέλουμε να δημιουργήσουμε την εφαρμογή. Σε αυτή την περίπτωση επιλέγουμε το Android API 22: Lollipop όπως και την επιλογή κινητό ή tablet. Τέλος για την ολοκλήρωση επιλέγουμε την Blank Activity επιλογή και την ονομάζουμε LogIn activity. Δημιουργήσαμε έτσι το LogIn.java και το activity_log_in.xml αρχείο.

4.3 Δημιουργία φόρμα εισόδου

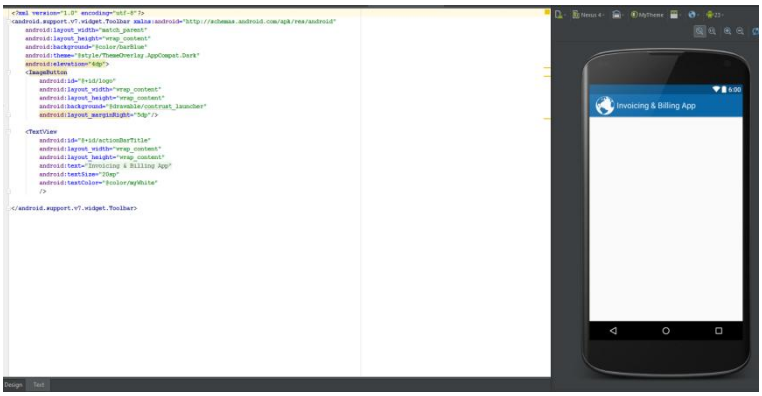
Για αρχή θα δημιουργήσουμε μία φόρμα εισόδου που ο κάθε χρήστης θα μπορεί να κάνει Log in για να έχει πρόσβαση στην εφαρμογή. Θα αποτελείται από 2 EditText, ένα για το όνομα χρήστη και ένα για τον κωδικό του, ένα κουμπί και ένα Check box για την επιλογή Remember me. Επειδή δεν έχουμε πρόσβαση για πολλούς χρήστες από το RESTful API που θα χρησιμοποιήσουμε, η είσοδος θα γίνεται για έναν χρήστη με συγκεκριμένο όνομα και κωδικό.

Εικόνα 38: Είσοδος χρήστη

```
    <EditText
        android:id="@+id/usernameTextField"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginBottom="13dp"
        android:hint="Username"
        android:inputType="text"
        android:paddingLeft="10dp"
        android:textStyle="bold"
        android:textColorHint="@color/myDarkGrey"/>

    <EditText
        android:id="@+id/passwordTextField"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:hint="Password"
        android:inputType="textVisiblePassword|textPassword"
        android:paddingLeft="10dp"
        android:password="true"
        android:textStyle="bold"
        android:textColorHint="@color/myDarkGrey" />
</LinearLayout>

<Button

    android:id="@+id/signInButton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/button_change_bg"
    android:singleLine="true"
    android:text="Sign in"
    android:textAllCaps="false"
    android:textColor="@drawable/button_change_text"
    android:textSize="20sp" />

<CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Remember me"
    android:id="@+id/rememberAccount"
```

Εικόνα 39: activity_log_in.xml

Στην συνέχεια θα φτιάξουμε ένα custom Toolbar χρησιμοποιώντας την βιβλιοθήκη compile **'com.android.support:support-v4:22.0.+'**. Το οποίο θα έχει την εξής μορφή:



Εικόνα 40: Δημιουργία custom Toolbar

Το Toolbar που δημιουργήσαμε θα περιέχει το Logo της εφαρμογής και μία κατάλληλη λεζάντα.

4.3.1 Κώδικας λειτουργίας της οθόνης εισόδου

Πρέπει να ξέρουμε αν υπάρχει πρόσβαση του χρήστη στο διαδίκτυο. Γι' αυτό πρέπει να πάρουμε τα κατάλληλα permissions από το Android. Δηλαδή πρόσβαση μέσω WIFI ή μέσω του provider. Αυτά πρέπει να προστεθούν στο Android Manifest.xml

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Το 1^{ο} για να έχουμε πρόσβαση στο Internet και το 2^{ο} για να πραγματοποιείται έλεγχος για την κατάσταση του δικτύου. Αφού το Android μας επιτρέπει να χρησιμοποιήσουμε τις άδειες, κάνουμε έναν έλεγχο για να διαπιστώσουμε αν η συσκευή είναι συνδεδεμένη στο Internet. Το οποίο γίνεται μέσω μίας Background υπηρεσίας AsyncTask για να εμφανίσουμε το κατάλληλο μήνυμα στον χρήστη στις περιπτώσεις που είναι ή δεν είναι συνδεδεμένος. Στην περίπτωση που είναι μεταβαίνει στην κύρια οθόνη της εφαρμογής.

```
public class ConnectingTask extends AsyncTask<Void, Void, Void> {
    ConnectivityManager cm = (ConnectivityManager) getApplicationContext().getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo activeNetwork = cm.getActiveNetworkInfo();

    @Override
    protected void onPreExecute() {
        StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder().permitAll().build();
        StrictMode.setThreadPolicy(policy);
        if (activeNetwork != null && activeNetwork.isConnectedOrConnecting()) {
            progressDialog = ProgressDialog.show(LogIn.this, "Logging In", "Please Wait...", true);

        } else {
            negativeProgressDialog = ProgressDialog.show(LogIn.this, "Something went wrong", "No internet connection!", true);
            (Thread) run() -> {
                    try {
                        sleep(2500);
                    } catch (Exception e) {
                        Log.e("tag", e.getMessage());
                    }
                    negativeProgressDialog.dismiss();
            }.start();
        }
        super.onPreExecute();

    }

    @Override
    protected void onPostExecute(Void s) {
        if (activeNetwork != null && activeNetwork.isConnectedOrConnecting()) {
            finish();
        }
        super.onPostExecute(s);
    }

    @Override
    protected Void doInBackground(Void... params) {
        if (activeNetwork != null && activeNetwork.isConnectedOrConnecting()) {
            startActivity(new Intent(LogIn.this, Dashboard.class));
        }
        return null;
    }
}
```

Εικόνα 41: έλεγχος διαθεσιμότητας internet

4.3.2 Στοιχεία φόρμας εισόδου & λειτουργεία υπενθύμισης

Ο χρήστης πρέπει να εισάγει τα σωστά στοιχεία εισόδου. Επειδή χρειαζόμαστε μόνο έναν, αυτά είναι “admin” για το όνομα εισόδου kai “admin” για τον κωδικό. Αν ο χρήστης εισχωρήσει κάτι άλλο από αυτά τότε δεν γίνεται δεκτή η πρόσβαση και ενημερώνεται με κατάλληλο μήνυμα.

```
if (username.getText().toString().equals("admin") && password.getText().toString().equals("admin"))
        new ConnectingTask().execute();
else Toast.makeText(getApplicationContext(), "Wrong username or password", Toast.LENGTH_LONG).show();
```

Αν ο χρήστης πατήσει την επιλογή του checkbox, τότε γίνεται η λειτουργία του Remember me και πρέπει να αποθηκεύονται τα δεδομένα που έχει εισχωρήσει για την επόμενη φορά που ανοίξει την εφαρμογή. Είτε αυτά είναι αληθείς είτε όχι. Αυτό επιτυγχάνεται χρησιμοποιώντας τα Shared Preferences που η ιδιότητές τους είναι να αποθηκεύουν συγκεκριμένες τιμές για μελλοντική χρήση.

```
private SharedPreferences loginPreferences;
private SharedPreferences.Editor loginPrefsEditor;
private Boolean saveLogin;
private String usernameString,passwordString;
……
saveLogin = loginPreferences.getBoolean("saveLogin", false);
if (saveLogin) {
        username.setText(loginPreferences.getString("username", ""));
        password.setText(loginPreferences.getString("password", ""));
        rememberMe.setChecked(true);  }
usernameString = username.getText().toString();
passwordString = password.getText().toString();
if (rememberMe.isChecked()) {
        loginPrefsEditor.putBoolean("saveLogin", true);
        loginPrefsEditor.putString("username", usernameString);
        loginPrefsEditor.putString("password", passwordString);
        loginPrefsEditor.commit();
}
else {
        loginPrefsEditor.clear();
        loginPrefsEditor.commit();
}
……
```

4.4 Κύρια οθόνη

Εφόσον ο χρήστης συμπληρώσει σωστά την φόρμα εισόδου, εισέρχεται στην κύρια οθόνη της εφαρμογής. Στο ξεκίνημά της επιβάλλεται να γίνει μία κλήση στον server του API για να πάρουμε τα δεδομένα που θα χρειαστεί να δείξουμε στον χρήστη. Πριν αρχίσουμε να αναλύουμε την κύρια οθόνη πρέπει να δούμε το πώς γίνονται αυτές οι «κλήσεις» στον server με το χρησιμοποιώντας το πρωτόκολλο REST.

4.4.1 Δημιουργία κλάσης για τα REST calls

Δημιουργήσαμε μια κλάση JAVA, όπου την ονομάσαμε RestCalls.java, για την απλοποίηση και την αποφυγή επαναλήψεων κώδικα. Η οποία, εμπεριέχει όλα τα απαραίτητα δεδομένα για τις μεθόδους που θα χρησιμοποιούμε, όταν αυτό είναι απαραίτητο, ώστε να πετύχουμε την επιθυμητή επικοινωνία με τον Server.

4.4.1.1 *Μέθοδος GET*

Ίσως η πιο συχνά χρησιμοποιημένη μέθοδος. Χρησιμοποιείται για να Πάρουμε δεδομένα από τον server. Την υλοποιήσαμε σύμφωνα με τους κανόνες επικοινωνίας ενός Web Service.

```
public void GetCall() {
        try {
                URL restUrl = new URL("http://api.elorus-staging.com:/v1.0/" + address);
                request = (HttpURLConnection) restUrl.openConnection();
                request.setRequestProperty("Authorization", "Token ed8ee7b7c485128dfedf7be1c5a1a31536ed8fe9");
                request.setRequestProperty("X-Elorus-Organization", "134450922913793052");
                request.setRequestProperty("Content-Type", "application/json");
                request.connect();
                InputStream in = new BufferedInputStream(request.getInputStream());
                BufferedReader reader = new BufferedReader(new InputStreamReader(in));
                String line;
                while ((line = reader.readLine()) != null) {
                        sb.append(line).append("\n");
                }
                in.close();
                setJsonString(sb.toString());
        }
        catch (Exception e) {
                Log.e("myTag4", "exception", e);
        }
        finally {
                request.disconnect();
        }
}
```

Για αρχή φτιάξαμε το URL, του οποίου το String προστίθεται ένα address για το κατάλληλο φάκελο. Έτσι έχουμε πετύχει ένα γενικό πλάνο για να καλείται η μέθοδος GET με την κατάλληλη

κατάληξη διεύθυνσης. Έπειτα χρησιμοποιούμε την HttpURLConnection κλάση για να γίνει η σύνδεση με το URL. Σε αυτή την σύνδεση επιβάλλεται σύμφωνα με τους κανόνες του REST να βάλουμε κάποια χαρακτηριστικά που εμπεριέχονται στο API. Για αρχή το **κλειδί** "**Token**" για το Authorization, το **id** του οργανισμού μας, αφού επιβάλλεται από το συγκεκριμένο API και τέλος το **Content-Type** που θα είναι το application/json. Αφού έχουμε φτιάξει τα απαραίτητα χαρακτηριστικά για μία επιτυχής σύνδεση, την πραγματοποιούμε με τις κλάσεις InputStream και BufferedReader για να πάρουμε δεδομένα από τον server. Τα οποία τα αποθηκεύουμε σε ένα String που είναι public για την χρήση του σε οποιαδήποτε java κλάση. Τέλος κλείνουμε την σύνδεσή μας με τον server.

4.4.1.2 *Μέθοδος POST*

Την μέθοδο Post την χρησιμοποιούμε όταν πρέπει να δημιουργήσουμε κάτι και να το στείλουμε στον server. Αυτή υλοποιείται όπως και η GET αλλά με κάποιες διαφορές.

```
public void PostCall() {
        try {
                URL restUrl = new URL("http://api.elorus-staging.com:/v1.0/" + address);
                response = (HttpURLConnection) restUrl.openConnection();
                response.setRequestProperty("Authorization", "Token ed8ee7b7c485128dfedf7be1c5a1a31536ed8fe9");
                response.setRequestProperty("X-Elorus-Organization", "134450922913793052");
                response.setRequestProperty("Content-Type", "application/json");
                response.setRequestMethod("POST"); response.setDoOutput(true); response.setUseCaches(false);
                response.connect();
                OutputStreamWriter out = new OutputStreamWriter(response.getOutputStream(), "UTF-8");
                out.write(jsonString);
                out.flush();
                out.close();
                if (response.getResponseCode() == 400) {
                        BufferedReader br = new BufferedReader(new InputStreamReader(response.getErrorStream()));
                        String line;
                        while ((line = br.readLine()) != null) {
                                sb.append(line).append("\n");
                        }
                        br.close();
                }
                if (response.getResponseMessage().equals("CREATED")) {
                        response.disconnect();
                }
        }
        catch (Exception e) {
                response.disconnect();
        }
        finally {
                response.disconnect();
        }
}
```

Όπως και η GET, έτσι και στη POST, δηλώνονται τα απαραίτητα χαρακτηριστικά του REST με τις προσθήκες ότι πρέπει να πούμε ποια μέθοδο θα χρησιμοποιήσουμε, καθώς η GET είναι η default μέθοδος και δεν χρειάστηκε πριν να την δηλώσουμε. Για να στείλουμε δεδομένα στον server χρησιμοποιούμε την κλάση OutputStreamWriter με το String που θέλουμε να στείλουμε, τα οποία είναι public για να αξιοποιούνται από όλες τις κλάσεις Java της εφαρμογής μας. Τέλος πρέπει να κλείσουμε το OutputStream που ανοίξαμε. Αφού πάρουμε την απάντηση από τον server με το κατάλληλο response κωδικό πράττουμε αναλόγως. Αν είναι επιτυχής, κλείνουμε την HttpURLConnection σύνδεσή μας. αν πάρουμε 400 διαβάζουμε το μήνυμα για να δούμε τί πήγε κακά και μετά κλείνουμε την HttpURLConnection σύνδεσή μας.

4.4.1.3 *Μέθοδος PUT*

Η PUT χρησιμοποιείται όταν πρέπει να ενημερώσουμε με αλλαγές τον server για κάποια εγγραφή στα δεδομένα του, όπως για παράδειγμα να αλλάξουμε τα δεδομένα μιας επαφής. Συντάζεται όπως τη POST.

```
public void PutCall() {
        try {
                URL restUrl = new URL("http://api.elorus-staging.com/v1.0/" + address);
                put = (HttpURLConnection) restUrl.openConnection();
                put.setRequestMethod("PUT");
                put.setRequestProperty("Authorization", "Token ed8ee7b7c485128dfedf7be1c5a1a31536ed8fe9");
                put.setRequestProperty("X-Elorus-Organization", "134450922913793052");
                put.setRequestProperty("Content-Type", "application/json");
                put.setDoOutput(true);
                put.setUseCaches(false);
                put.connect();
                OutputStreamWriter out = new OutputStreamWriter(put.getOutputStream(), "UTF-8");
                out.write(jsonString);
                out.flush();
                out.close();
                if (put.getResponseCode() != 200) {
                        BufferedReader br = new BufferedReader(new InputStreamReader(put.getErrorStream()));
                        String line;
                        while ((line = br.readLine()) != null) {
                                sb.append(line).append("\n");
                        }
                        br.close();
                }
        }
        catch (Exception e) {
                put.disconnect();
        }
        finally {
                put.disconnect();
        }
}
```

4.4.1.4 *Μέθοδος DELETE*

Η μέθοδος DELETE χρησιμοποιείται για την διαγραφή ενός ή πολλών δεδομένων από τον server. Στην περίπτωση αυτή γίνεται μόνο έλεγχος αν δεν το response code δεν είναι επιτυχής για να δούμε τί δεν πήγε καλά σύμφωνα με το response μήνυμα που θα στείλει ο server.

```
public void DeleteCall(URL urls) {
        try {
                delete = (HttpURLConnection) urls.openConnection();
                delete.setRequestMethod("DELETE");
                delete.setRequestProperty("Authorization", "Token ed8ee7b7c485128dfedf7be1c5a1a31536ed8fe9");
                delete.setRequestProperty("X-Elorus-Organization", "134450922913793052");
                delete.setRequestProperty("Content-Type", "application/json");
                delete.setUseCaches(false);
                delete.connect();
                if (delete.getResponseCode() != 200) {
                        BufferedReader br = new BufferedReader(new InputStreamReader(delete.getErrorStream()));
                        String line;
                        while ((line = br.readLine()) != null) {
                                sb.append(line).append("\n");
                        }
                        br.close();
                }
        }
        catch (Exception e) {
                Log.e("myTag4", "exception", e);
                delete.disconnect();
        }
        finally {
        delete.disconnect();
        }
}
```

4.4.1.5 *Μέθοδος PUT για Notifications*

Τέλος υπάρχει και μια extra μέθοδος η οποία κάνει put τα notifications για να κάνει mars-as-read το κάθε notification που διαβάζεται. Η συγκεκριμένη μέθοδος χρειάζεται για να αλλάξουμε συγκεκριμένο πεδίο από τα δεδομένα των notifications. Το μόνο που αλλάζει από την PUT είναι το URL.

```
URL restUrl = new URL("http://api.elorus-staging.com/v1.0/notifications/"+address+"/mark-read/");
```

4.4.2 Αρχιτεκτονική κύριας οθόνης

Πριν δούμε τα περιεχόμενα της κύριας οθόνης πρέπει να δούμε την αρχιτεκτονική της. Θέλουμε στην κύρια οθόνη να εμφανίζονται όλα τα περιεχόμενα για να είναι user-friendly και να υπάρχει εύκολη πλοήγηση στον χρήστη.

Για αρχή γίνεται η δημιουργία και προσθήκη ενός custom Tollbar που περιέχεται το λογότυπο της εφαρμογής.

Έπειτα γίνεται η προσθήκη ενός RecyclerView και ενός SliderLayout από Tabs, στο Activity της κύριας οθόνης, με τις κατηγορίες των δεδομένων που έχει πρόσβαση στον χρήστη. Για την υλοποίηση του Drawer γίνεται η χρήση της κλάσης RecyclerView από την βιβλιοθήκη **'com.android.support:support-v4:22.0.+'** και η SlidingTabLayout που είναι εξωτερική βιβλιοθήκη του Android.

```
<android.support.v4.widget.DrawerLayout
    android:id="@+id/DrawerLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:elevation="7dp">

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

        <tabs.SlidingTabLayout
            android:id="@+id/tabs"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="@color/barBlue"
            android:elevation="2dp" />

        <android.support.v4.view.ViewPager
            android:id="@+id/pager"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1" />
    </LinearLayout>

    <android.support.v7.widget.RecyclerView
        android:id="@+id/RecyclerView"
        android:layout_width="320dp"
        android:layout_height="match_parent"
        android:layout_gravity="left"
        android:background="#ffffff"
        android:scrollbars="vertical" />
</android.support.v4.widget.DrawerLayout>
```

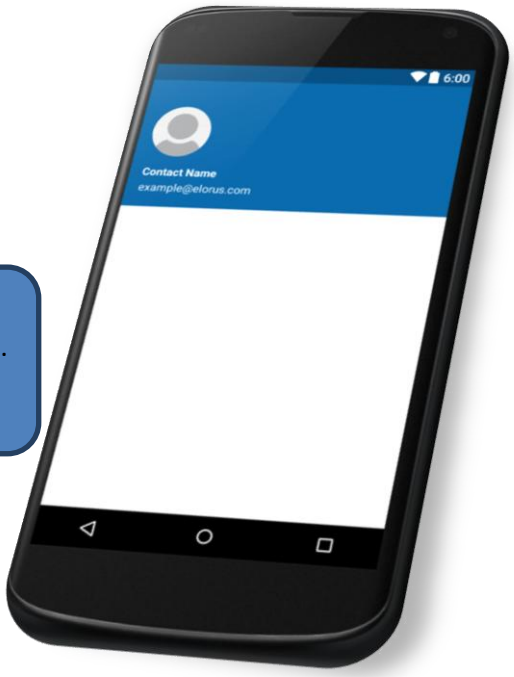
Εικόνα 42: DrawerLayout

4.4.2.1 *Recycler View*

Για την χρησιμοποίηση του RecyclerView πρέπει να φτιάξουμε ένα Adapter για να βάλουμε μέσα τα περιεχόμενά του. Έτσι φτιάχνουμε ένα custom Adapter και καλούμε τον Constructor με τα στοιχεία που πρέπει όπως τα Titles, τα Icons, το όνομα και το email του χρήστη και το εικονίδιό του.

```
MyAdapter mAdapter = new MyAdapter(TITLES, ICONS, NAME, EMAIL, PROFILE, this);
```

Ο Adapter αυτός είναι ένα RecyclerView.Adapter<MyAdapter.ViewHolder> το οποίο αποτελείται από ένα header και ένα body. Όπως βλέπουμε παρακάτω το Header του έχει την παρακάτω σχεδίαση:



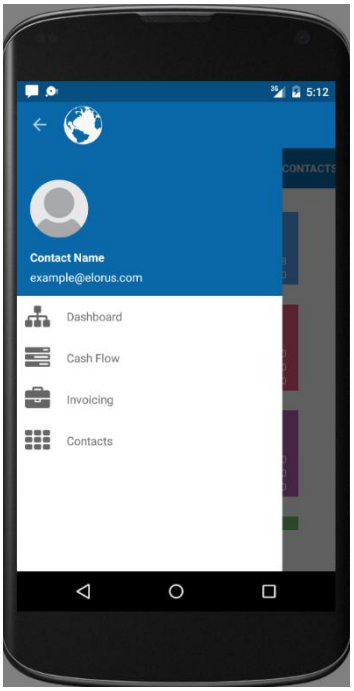
Εικόνα 43: RecyclerView Header

Αυτό γίνεται με την διείσδυση ενός ξεχωριστού layout στο layout του RecyclerView με την εντολή:

```
View v =
LayoutInflater.from(parent.getContext()).inflate(R.layout.
header,parent,false);
```

Το Header αποτελείται από το εικονίδιο, το όνομα και το email του χρήστη που δώσαμε σαν όρισμα στον constructor του Adapter.

Το Body του RecyclerView περιέχει τα εικονίδια και ονόματα από τις κατηγορίες της πλοήγησης. Τα οποία τα σχεδιάζουμε σε ξεχωριστό layout και τα ενσωματώνουμε, όπως είδαμε και προηγουμένως, κάτω από το Header του View. Η τελική μορφή του RecyclerView είναι έτσι:



Εικόνα 44: RecyclerView

Το RecyclerView εμφανίζεται είτε πατώντας στο κουμπί πάνω αριστερά του Toolbar, είτε με το σύρσιμο (swipe) του δακτύλου από οπουδήποτε αριστερά της οθόνης προς τα δεξιά. Εξαφανίζεται πατώντας το κουμπί στο Toolbar ή swipe από τα δεξιά της οθόνης προς τα αριστερά. Η λειτουργία του RecyclerView διευκολύνει τον χρήστη στην πλοήγηση στις κατηγορίες της βασικής οθόνης. Με ένα click στο όνομα της κατηγορίας γίνεται η πλοήγηση στην αντίστοιχη θέση και έπειτα κλείνει το RecyclerView. Για την λειτουργία αυτή πρέπει να βάλουμε ένα Listener στην περίπτωση που πατηθεί ένα αντικείμενο μέσα στο Drawer.

```
mRecyclerView.addOnItemTouchListener(new RecyclerView.OnItemTouchListener() {
    @Override
    public void onRequestDisallowInterceptTouchEvent(boolean b) {
    }

    @Override
    public boolean onInterceptTouchEvent(RecyclerView recyclerView, MotionEvent motionEvent) {
        View child = recyclerView.findChildViewUnder(motionEvent.getX(), motionEvent.getY());
        onTouchDrawer(recyclerView.getChildAdapterPosition(child));


        if (child != null && mGestureDetector.onTouchEvent(motionEvent)) {
            Drawer.closeDrawers();

            return true;
        }
        return false;
    }

    @Override
    public void onTouchEvent(RecyclerView recyclerView, MotionEvent motionEvent) {

    }
});
```

Εικόνα 45:RecyclerView Listener

Όπου η μέθοδος onTouchDrawer που δημιουργήσαμε, μας επιτρέπει να πούμε σε ποια θέση θα γίνεται η πλοήγηση αναλόγως το αντικείμενο που πατιέται.

```
//Method for tab selection through drawer buttons
public void onTouchDrawer(final int position) {
    if (position == 1) {
        //selection for Contacts
        pager.setCurrentItem(0);
    } else if (position == 2) {
        //selection for Cashflow
        pager.setCurrentItem(1);
    } else if (position == 3) {
        //selection for Invoicing
        pager.setCurrentItem(2);
    } else if (position == 4) {
        //selection for Reports
        pager.setCurrentItem(3);
    }
}
```

Εικόνα 46: Λειτουργία του onTouchDrawer

4.4.2.2 *SlidingTabLayout*

Για την υλοποίηση του SlidingTabLayout πρέπει να ενσωματώσουμε 2 Java κλάσεις στον φάκελο Java της εφαρμογής, Τη SlidingTabLayout.java και την SlidingTabStrip.java, διότι χρειάζονται για την υλοποίηση των Tabs-πλοηγών. Η πρώτη χρειάζεται για την λειτουργία των Tabs και η δεύτερη για την σχεδίασή τους. Βεβαίως μπορούμε να πειράξουμε τον έτοιμο κώδικα για την μορφοποίησή τους αλλά στην συγκεκριμένη εφαρμογή δεν χρειάστηκε.

Πρώτα πρέπει να καλέσουμε την SlidingTabLayout στο xml του Activity που θα την χρησιμοποιήσουμε. Μετά φτιάχνουμε έναν ViewPager για να μορφοποιήσουμε τα Tabs, όπως είδαμε στην πιο πάνω εικόνα μέσα στο DrawerLayout.

Στο αρχείο java της κύριας οθόνης φτιάξαμε μια μέθοδο όπου ορίζεται ο ViewPager Adapter και το SlidingLayout, με σκοπό να την καλούμε όποτε χρειάζεται για την δημιουργία των tabs και των περιεχόμενών τους.

```
public void generateTabs() {
    // Creating The ViewPagerAdapter and Passing Fragment Manager, Titles fot the Tabs and Number Of Tabs.
    adapter = new ViewPagerAdapter(getSupportFragmentManager(), Titles, Numboftabs);
    // Assigning ViewPager View and setting the adapter
    pager = (ViewPager) findViewById(R.id.pager);
    pager.setAdapter(adapter);
    pager.setOffscreenPageLimit(4);
    // Assiging the Sliding Tab Layout View
    tabs = (SlidingTabLayout) findViewById(R.id.tabs);
    // To make the Tabs Fixed set this true, This makes the tabs Space Evenly in Available width
    tabs.setDistributeEvenly(true);
    // Setting Custom Color for the Scroll bar indicator of the Tab View
    tabs.setCustomTabColorizer((position) -> {
            return getResources().getColor(R.color.tabsScrollColor);
    });
    // Setting the ViewPager For the SlidingTabsLayout
    tabs.setViewPager(pager);
}
```

Εικόνα 47: Φόρτωση των tabs

Ο ViewPager Adapter που δημιουργήσαμε είναι τύπου FragmentStatePagerAdapter διότι τα Tabs από το SlidingTabLayout είναι Fragments. Ο Adapter περιέχει τρείς κύριες μεθόδου. Την getItem για την επιλογή των fragments (Tabs) στην κατάλληλη θέση, την getPageTitle για την θέση των τίτλων των Tabs και την getCount για τον αριθμό των Tabs που δημιουργήθηκαν.

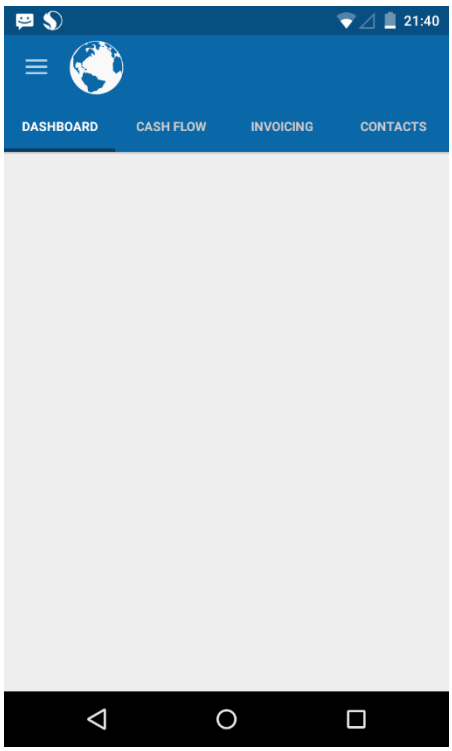
```
//This method return the fragment for the every position in the View Pager
@Override
public Fragment getItem(int position) {
    final Fragment result;
    switch (position) {
        case 0:
            // First Fragment of First Tab
            result = new Reports();
            break;
        case 1:
            // First Fragment of Second Tab
            result = new Cashflow();
            break;
        case 2:
            // First Fragment of Third Tab
            result = new Invoicing();
            break;
        case 3:
            // First Fragment of Third Tab
            result = new Contacts();

            break;
        default:
            result = null;
            break;
    }

    return result;
```

Εικόνα 49: getItem

```
// This method return the titles for the Tabs in the Tab Strip

@Override
public CharSequence getPageTitle(int position) { return Titles[position]; }

// This method return the Number of tabs for the tabs Strip

@Override
public int getCount() { return NumbOfTabs; }
```

Εικόνα 48: getPageTitle & getCount

Κάνοντας “swipe” δεξιά ή αριστερά μετακινούμαστε στα Tabs του SlidingTabLayout, ή πατώντας πάνω στα ονόματά τους. Το τελικό αποτέλεσμα του SlidingTabLayout φαίνεται στην παρακάτω εικόνα:



Εικόνα 50: SlidingTabLayout

4.4.3 Λειτουργία κύριας οθόνης

Για την λειτουργία της κύριας οθόνης πρέπει να γίνεται η εμφάνιση των περιεχομένων των Tabs πλοήγησης. Πριν δούμε το περιεχόμενο και το πώς φαίνεται πρέπει να προσδιορίσουμε ότι το κάθε Tab είναι και ένα Fragment. Αυτό συμβαίνει επειδή πρέπει το κάθε Tab να θεωρηθεί σαν μια εικονική οντότητα που έχει τις δικές του λειτουργίες και περιεχόμενα. Οπότε όλα τα Tabs είναι μέσα στο Activity της κύριας οθόνης και μπορούν να μοιραστούν τις ιδιότητες και τα χαρακτηριστικά του.

Έχουμε τέσσερα Tabs, το καθένα εμφανίζει διαφορετικά πράγματα, όμως όλα εμφανίζουν περιεχόμενο που χρειάζεται δεδομένα από τον server. Έτσι δημιουργούμε μία εσωτερική κλάση η οποία είναι τύπου AsyncTask. Εκεί γίνεται η επικοινωνία με τον server για την απολαβή δεδομένων. Η κλάση αυτή έχει τρείς βασικές μεθόδους την onPreExecute για να κάνει κάτι πριν εκτελεστεί, την onPostExecute για να κάνει κάτι μετά την εκτέλεση και την doInBackground στην οποία γίνεται η εκτέλεση.

Στην onPreExecute εμφανίζουμε ένα Alert Dialog για την πρόοδο της εκτέλεσης αλλά και για να βλέπει κάτι ο χρήστης όσο περιμένει να γίνει η απολαβή των δεδομένων. Εκεί γίνονται και διεργασίες σε UI. Το Alert Dialog είναι ακυρώσιμο, στην περίπτωση που ο χρήστης δεν θέλει να βλέπει αλλά να κάνει κάτι με το UI της εφαρμογής.

```
@Override
protected void onPreExecute() {

    progressDialog = ProgressDialog.show(Dashboard.this, "Fetching data", "Please Wait...", true, true
    StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder().permitAll().build();
    StrictMode.setThreadPolicy(policy);
    thisIntent = getIntent();
    myCalendar = Calendar.getInstance();
    myCalendar.set(Calendar.DAY_OF_MONTH, myCalendar.getActualMaximum(Calendar.DAY_OF_MONTH));
    dateFormat = new SimpleDateFormat("yyyy-MM-dd", Locale.getDefault());
    oldCalendar = Calendar.getInstance();
    oldCalendar.add(Calendar.MONTH, -2);
    oldCalendar.set(Calendar.DAY_OF_MONTH, oldCalendar.getActualMinimum(Calendar.DAY_OF_MONTH));
}
```

Εικόνα 51: onPreExecute

Εικόνα 52: Progress Dialog

Στην doInBackground γίνονται τα Rest Calls. Σύμφωνα με αυτά γίνεται η επικοινωνία και παίρνουμε τα απαραίτητα δεδομένα από τον server. Η μέθοδος αυτή εκτελείται στο παρασκήνιο για να μπορεί ο χρήστης να ασχοληθεί με το UI της εφαρμογής.

```
@Override
protected Void doInBackground(Void... params) {
        if (getIntent().getStringExtra("notificationId") != null) {
            getCall = new RestCalls(getIntent().getStringExtra("notificationId"));
            getCall.PutNotificaton();
        }
        getCall = new RestCalls("invoices?period_from=" + dateFormat.format(oldCalendar.getTime()) + "&period_to=" + dateFormat.format(myCalendar.getTime()) + "&draft__exact=0");
        getCall.GetCall();
        periodInvoices = getCall.getJsonString();
        getCall = new RestCalls("creditnotes?period_from=" + dateFormat.format(oldCalendar.getTime()) + "&period_to=" + dateFormat.format(myCalendar.getTime()) + "&draft__exact=0");
        getCall.GetCall();
        periodCreditNotes = getCall.getJsonString();
        getCall = new RestCalls("purchases?period_from=" + dateFormat.format(oldCalendar.getTime()) + "&period_to=" + dateFormat.format(myCalendar.getTime()) + "&draft__exact=0");
        getCall.GetCall();
        periodPurchaseInvoices = getCall.getJsonString();
        getCall = new RestCalls("purchasecreditnotes?period_from=" + dateFormat.format(oldCalendar.getTime()) + "&period_to=" + dateFormat.format(myCalendar.getTime()) + "&draft__exact=0");
        getCall.GetCall();
        periodPurchaseCreditNotes = getCall.getJsonString();
        getCall = new RestCalls("contacts/");
        getCall.GetCall();
        contacts = getCall.getJsonString();
        getCall = new RestCalls("cashpayments/");
        getCall.GetCall();
        cashpayments = getCall.getJsonString();
        getCall = new RestCalls("cashreceipts/");
        getCall.GetCall();
        cashreceipts = getCall.getJsonString();
        getCall = new RestCalls("invoices/");
        getCall.GetCall();
        invoices = getCall.getJsonString();
        getCall = new RestCalls("creditnotes/");
        getCall.GetCall();
        creditNotes = getCall.getJsonString();
        getCall = new RestCalls("estimates/");
        getCall.GetCall();
        estimates = getCall.getJsonString();
        getCall = new RestCalls("purchases/");
        getCall.GetCall();
        purchases = getCall.getJsonString();
        getCall = new RestCalls("purchasecreditnotes/");
        getCall.GetCall();
        purchasesCreditNotes = getCall.getJsonString();

        thisIntent.putExtra("contacts", contacts)
                .putExtra("cashpayments", cashpayments)
                .putExtra("cashreceipts", cashreceipts)
                .putExtra("invoices", invoices)
                .putExtra("creditnotes", creditNotes)
                .putExtra("estimates", estimates)
                .putExtra("purchases", purchases)
                .putExtra("purchasecreditnotes", purchasesCreditNotes)
```

Εικόνα 53: doInBackground

Αφού έχει τελειώσει την εκτέλεσή της η doInBackground εκτελείται η onPostExecute που εκεί κάνουμε dismiss το Alert Dialog, και δείχνουμε τις αλλαγές στο UI σύμφωνα με τα δεδομένα που πήραμε. Δηλαδή για την εφαρμογή αυτή καλούμε την generateTabs().

```
@Override
protected void onPostExecute(Void s) {
    super.onPostExecute(s);
    generateTabs();
    progressDialog.dismiss();
}
```

Εικόνα 54: onPostExecute

Αφού γίνουν τα calls στην doInBackground διαβιβάζουμε ως extra στοιχεία τις μεταβλητές που αποθηκεύσαμε τα δεδομένα που πήραμε από τον server. Αυτό επιτυγχάνεται καλώντας το Intent της κύριας οθόνης και με την μέθοδο *putExtra("String","String").*

```
thisIntent.putExtra("contacts", contacts) .putExtra("cashpayments", cashpayments)
        .putExtra("cashreceipts", cashreceipts) .putExtra("invoices", invoices)
        .putExtra("creditnotes", creditNotes) .putExtra("estimates", estimates)
        .putExtra("purchases", purchases) .putExtra("purchasecreditnotes",
        purchasesCreditNotes)
        .putExtra("periodInvoices", periodInvoices)
        .putExtra("periodCreditNotes", periodCreditNotes)
        .putExtra("periodPurchaseInvoices", periodPurchaseInvoices)
        .putExtra("periodPurchaseCreditNotes", periodPurchaseCreditNotes);
```

Με αυτόν τον τρόπο μπορούμε να πάρουμε πιο εύκολα και γρήγορα τα δεδομένα για την διαχείρισή τους στα fragments (Tabs) και να τα χρησιμοποιήσουμε για την επίδειξη των περιεχομένων τους. Το AsyncTask αυτό εκτελείται στην onCreate μέθοδο του Activity.

```
new GettingDataTask().execute();
```

Για να έχουμε σωστή διατύπωση δεδομένων, η εντολή αυτή έχει τοποθετηθεί να εκτελείται και στην onRestart μέθοδο του Activity, όπου σύμφωνα με το Activity lifecycle επιθυμούμε να γίνεται ανανέωση στα περιεχόμενα της κύριας οθόνης στις περιπτώσεις που θέλουμε να επιστρέψουμε σε αυτή από κάπου αλλού, για παράδειγμα ένα άλλο Activity.

4.5 Reports

Το πρώτο fragment της εφαρμογής είναι τα Reports. Εφόσον είναι το πρώτο fragment και στην ουσία η πρώτη οθόνη που βλέπει ο χρήστης μόλις ανοίξει η κύρια οθόνη της εφαρμογής, πρέπει να έχει μία όμορφη, εύκολη στο μάτι απεικόνιση. Αυτό επιτυγχάνεται εμφανίζοντας μερικά στατιστικά και τιμές από σύνολα, σε γραφήματα ώστε να έχει μια ολοκληρωμένη και γενική εικόνα της διαχείρισης των λογιστικών του. Πιο συγκεκριμένα, εμφανίζεται ένας πίνακας με τα Ταμειακά διαθέσιμα του χρήστη το οποίο αποτελείται από την γενική τιμή των εισπράξεων και των πληρωμών. Ένας πίνακας πωλήσεων και ένας πληρωμών με τις τιμές τους τρείς τελευταίους μήνες και ένας πίνακας με τα 5 τελευταία παραστατικά. Όλα τα δεδομένα για την απεικόνιση των τιμών τα παίρνουμε από τα Calls που κάναμε στο Activity.

4.5.1 Γενικό πλάνο απεικόνισης

Η σχεδίαση του xml του fragment έχει γίνει με την μορφοποίηση τριών LinearLayout και ενός ListView που αυτά είναι παιδία ενός ScrollView για την δυνατότητα στον χρήστη να κάνει scroll down/up. Τα LinearLayouts έχουν μορφοποιηθεί ώστε να μοιάζουν με πίνακες που περιέχουν ένα εικονίδιο, TextViews, που απεικονίζουν το τί βλέπει ο χρήστης καθώς και TextViews που μπαίνουν οι τιμές των δεδομένων από τον server. Για την τοποθέτηση και απεικόνιση των παραστατικών στο ListView έχει δημιουργηθεί μια κλάση Java όπου είναι ένας Adapter για την εξοικονόμηση ταχύτητας των αντικειμένων. Παρακάτω βλέπουμε το γενικό πλάνο σχεδίασης του fragment:

Εικόνα 55: tab_reports.xml

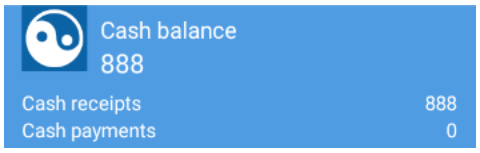
Όπως βλέπουμε στην θέση των εικονιδίων υπάρχει ένα γράμμα το οποίο είναι στην ουσία ο κωδικός για το κατάλληλο εικονίδιο που έχουν σε ένα .ttf αρχείο. Για την σωστή απεικόνιση πρέπει να φτιάξουμε ένα Typeface με αυτό το γράμμα. Για παράδειγμα για το πρώτο εικονίδιο:

```
<TextView
    android:id="@+id/ying_yang_icon_textView"
    android:layout_width="40dp"
    android:layout_height="40dp"
    android:background="@color/myBlueTableIconBackground"
    android:gravity="center"
    android:text="K"
    android:textAllCaps="false"
    android:textColor="@color/myWhite" />
```

Εικόνα 56: Ying Yang Icon TextView

```
viewHolder.yingYang = (TextView) v.findViewById(R.id.ying_yang_icon_textView);
viewHolder.yingYang.setTypeface(Typeface.createFromAsset(viewHolder.yingYang.getContext().g
etAssets(), "fonts/elorus-font.ttf"));
```

Όπου το αποτέλεσμα θα φαίνεται κάπως έτσι:



4.5.2 Latest Invoices Adapter

Για την τοποθέτηση όπως προαναφέρθηκε έχει δημιουργηθεί ένας Adapter οποίος είναι τύπου ArrayAdapter<ManageInvoice>. Σε αυτόν όπως και σε άλλους Adapters της εφαρμογής που θα δούμε πιο κάτω έχει χρησιμοποιηθεί η μέθοδος του ViewHolder. Σύμφωνα με αυτή την μέθοδο ορίζουμε στοιχεία σε μία στατική ViewHolder μέθοδο. Έπειτα καλώντας τα στοιχεία αυτά δίνουμε τους ορισμούς που θέλουμε και μεταβάλουμε τις τιμές τους για να τις χρησιμοποιήσουμε, όπως βλέπουμε παρακάτω:

```
@Override
public View getView(final int position, View convertView, ViewGroup parent) {
    // Get the data item for this position
    final ManageInvoice manageInvoice = getItem(position);
    // Check if an existing view is being reused, otherwise inflate the view
    ViewHolder viewHolder; // view lookup cache stored in tag
    if (convertView == null) {
        viewHolder = new ViewHolder();
        LayoutInflater inflater = LayoutInflater.from(getContext());
        convertView = inflater.inflate(R.layout.latest_invoices_row, parent, false);
        viewHolder.representations = (TextView) convertView.findViewById(R.id.latest_invoices_representations);
        viewHolder.date = (TextView) convertView.findViewById(R.id.latest_invoices_date);
        viewHolder.eye = (TextView) convertView.findViewById(R.id.latest_invoices_eye_icon);
        viewHolder.amount = (TextView) convertView.findViewById(R.id.latest_incoices_amount);
        convertView.setTag(viewHolder);
    } else {
        viewHolder = (ViewHolder) convertView.getTag();
    }
```

```
private static class ViewHolder {
    TextView representations;
    TextView date;
    TextView eye;
    TextView amount;
}
```

Η μέθοδος αυτή μας επιτρέπει την προ δημιουργία των στοιχείων μέχρι να κάνουμε κάτι σε αυτά. Επιτυγχάνουμε αύξηση ταχύτητας της λειτουργίας που θέλουμε να επιτύχουμε. Στην συγκεκριμένη περίπτωση εισχωρούμε στα στοιχεία που έχει το κάθε αντικείμενο από το κατάλληλο xml αρχείο και τοποθετούμε τις τιμές των δεδομένων.

Ο Adapter είναι μια λίστα από ManageInvoice. Εισχωρούμε στο view του, δηλαδή στο view του κάθε αντικειμένου, τα κατάλληλα στοιχεία στα σωστά πεδία.

```
// Populate the data into the template view using the data object
viewHolder.date.setText(manageInvoice.date);
//viewHolder.date.setOnClickListener gia na pame se view mode
viewHolder.representations.setText(manageInvoice.representation);
viewHolder.eye.setTypeface(Typeface.createFromAsset(viewHolder.eye.getContext().getAssets(), "fonts/elorus-font.ttf"));
viewHolder.amount.setText(manageInvoice.total);

// Return the completed view to render on screen
return convertView;
}
```

4.5.3 REST calls του fragment Reports

Τα REST Calls που χρειάζονται για την επίδειξη των δεδομένων είναι 4 Get Calls τα οποία γίνονται με βάση συγκεκριμένες περιόδους. Για τον σκοπό αυτόν πρέπει να παίρνουμε την διάρκεια, από τον τωρινό μήνα έως δύο μήνες πριν, από την συσκευή Android του χρήστη με την κλάση Calendar.

```
myCalendar = Calendar.getInstance();
myCalendar.set(Calendar.DAY_OF_MONTH, myCalendar.getActualMaximum(Calendar.DAY_OF_MONTH));
dateFormat = new SimpleDateFormat("yyyy-MM-dd", Locale.getDefault());
oldCalendar = Calendar.getInstance();
oldCalendar.add(Calendar.MONTH, -2);
oldCalendar.set(Calendar.DAY_OF_MONTH, oldCalendar.getActualMinimum(Calendar.DAY_OF_MONTH));
```

Τα Calls έχουν την εξής μορφή:

```
getCall = new RestCalls("invoices?period_from=" + dateFormat.format(oldCalendar.getTime()) + "&period_to=" +
        dateFormat.format(myCalendar.getTime()) + "&draft__exact=0");
getCall.GetCall();
getCall = new RestCalls("creditnotes?period_from=" + dateFormat.format(oldCalendar.getTime()) + "&period_to=" +
        dateFormat.format(myCalendar.getTime()) + "&draft__exact=0");
getCall.GetCall();
getCall = new RestCalls("purchases?period_from=" + dateFormat.format(oldCalendar.getTime()) + "&period_to=" +
        dateFormat.format(myCalendar.getTime()) + "&draft__exact=0");
getCall.GetCall
getCall = new RestCalls("purchasecreditnotes?period_from=" + dateFormat.format(oldCalendar.getTime()) +
        "&period_to=" + dateFormat.format(myCalendar.getTime()) + "&draft__exact=0");
getCall.GetCall();
```

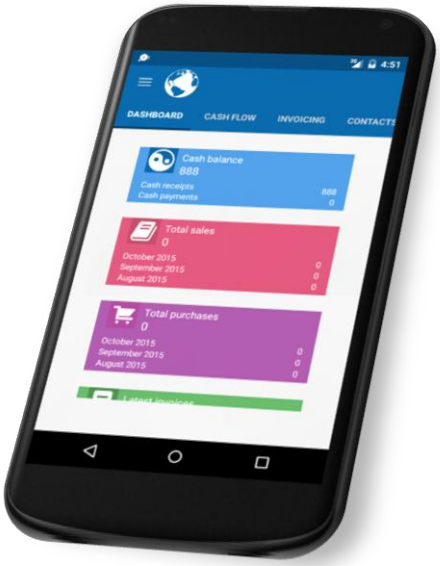
Εφόσον μεταβιβάσαμε τα δεδομένα από τα calls αυτά στο Intent του Activity σαν Extras, μπορούμε να τα πάρουμε και να τα χρησιμοποιήσουμε στο fragment:

```
Intent thisIntent = getActivity().getIntent();
String invoices = thisIntent.getStringExtra("periodInvoices");
String creditNotes = thisIntent.getStringExtra("periodCreditNotes");
String purchaseInvoices = thisIntent.getStringExtra("periodPurchaseInvoices");
String purchaseCreditNotes = thisIntent.getStringExtra("periodPurchaseCreditNotes");
```

Αυτά τα Strings είναι JSON αντικείμενα που μπορούμε να πάρουμε τα δεδομένα από τα κατάλληλα πεδία με την χρήση των κλάσεων JSONObject και JSONArray. Για παράδειγμα για να πάρουμε την τιμή του κάθε Cash Receipt, να τις προσθέσουμε για το συνολικό άθροισμα και να το τοποθετήσουμε στο text του Textview, χρησιμοποιούμε τις κλάσης έτσι:

```
double cashReceiptsVal = 0;
DecimalFormat decimalFormat = new DecimalFormat("###,###,###,###.##");
jsonObject = new JSONObject(cashReceipts);
for (int i = 0; i <jsonObject.getInt("count"); i++) {
        cashReceiptsVal = cashReceiptsVal +
        Double.parseDouble(jsonObject.getJSONArray("results").getJSONObject(i).getString("amount")
); }
cashReceiptsValue.setText(decimalFormat.format(cashReceiptsVal).replace(",", "."));
```

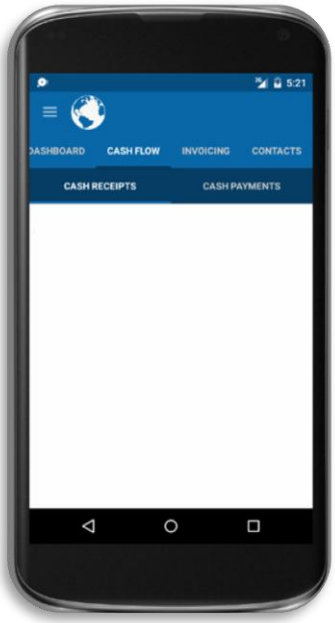
Με τον τρόπο αυτόν αξιοποιούμε όλα τα στοιχεία των δεδομένων που χρειαζόμαστε για την υλοποίηση του fragment. Το πρώτο fragment έχει την εξής τελική μορφή:



Εικόνα 57: Tab Reports

4.6 Cash Flow

Το δεύτερο Tab (fragment) έχει δύο κατηγορίες, τις εισπράξεις (Cash Receipts) και τις πληρωμές (Cash Payments). Η διαχείριση είναι η ίδια και στις δύο κατηγορίες. Το μόνο που αλλάζει είναι τα δεδομένα που ενδείκνυνται. Για την εμφάνιση των κατηγοριών θα χρησιμοποιήσουμε τις κλάσεις SlidingTabLayout και ViewPager για να δημιουργήσουμε υπό πλοηγούς Tabs στο fragment αυτό.




```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <tabs.SlidingTabLayout
        android:id="@+id/tabs_cash_flow"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:elevation="2dp"
        android:background="@color/tabsScrollColor"/>

    <android.support.v4.view.ViewPager
        android:id="@+id/pager_cash_flow"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1" />

</LinearLayout>
```

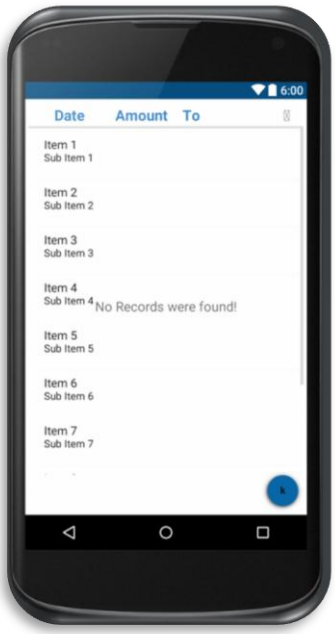
Εικόνα 58: Cash Flow Tabs

4.6.1 Λειτουργία του fragment Cash Flow

Για την λειτουργία του fragment δημιουργήσαμε έναν Adapter όπως και στο 1ο fragment, το οποίο τοποθετεί στα κατάλληλα Tabs τις κλάσεις των Cash Receipts και Cash Payments τα οποία είναι με την σειρά τους νέα fragments παιδία του Fragment Cash Flow.

```
//This method return the fragment for the every position in the Vie
@Override
public Fragment getItem(int position) {

    if (position == 0) // if the position is 0 we are returning the
    {
        return new CashReceipts();
    }
    if (position == 1)            // As we are having 2 tabs if the
    {
        return new CashPayments();
    } else
        return null;
}
```

Εικόνα 60: Τοποθέτηση της θέσης των Tabs στο ViewPager



Εικόνα 59: Fragments Raw list

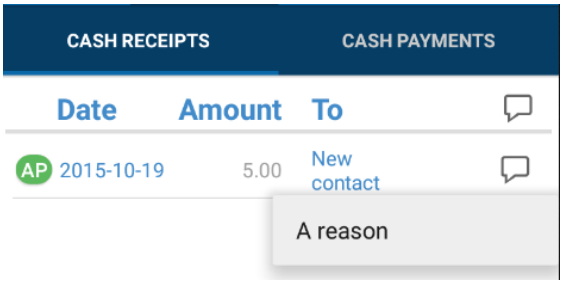
4.6.1.1 *Σχεδίαση των Fragment παιδιών*

Η απεικόνιση των fragments παιδιών, Cash Receipts και Cash Payments, βασίζεται στην δημιουργία μίας λίστας για το καθένα με την βοήθεια του ListView. Αν η λίστα είναι κενή, δηλαδή αν δεν υπάρχει καταχώρηση δεδομένων στο Server, εμφανίζεται ένα TextView με το απαραίτητο μήνυμα «No Records were found!» αλλιώς το TextView γίνεται αόρατο με την μέθοδο *.setVisibility(View.INVISIBLE);*.

Σε αυτή την οθόνη υπάρχει και ένα κουμπί στο τέλος της λίστας για την δημιουργία νέου Cash Receipt ή Cash Payment την οποία θα αναλύσουμε παρακάτω. Το εικονίδιο του κουμπιού μορφοποιείται με ένα Typeface από το .ttf που έχουμε τα icons. Πατώντας το, εμφανίζεται ένα popup menu με επιλογή για την δημιουργία μιας νέας καταχώρισης, ξεκινώντας ένα νέο Activity για τον σκοπό αυτό.

```
public void CircleImageButton(View v) {
    circleImageButton = (Button) v.findViewById(R.id.circleImageButton_cash_receipts);
    circleImageButton.setTypeface(Typeface.createFromAsset(circleImageButton.getContext().getAssets(), "fonts/elorus-font.ttf"));
    circleImageButton.setTextSize(TypedValue.COMPLEX_UNIT_SP, 35);
    circleImageButton.setTextColor(circleImageButton.getResources().getColor(R.color.myWhite));
    circleImageButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            PopupMenu cashReceiptsPopMenu = new PopupMenu(getActivity(), circleImageButton);
            cashReceiptsPopMenu.getMenuInflater().inflate(R.menu.cashflow_fragment_receipts, cashReceiptsPopMenu.getMenu());
            cashReceiptsPopMenu.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
                @Override
                public boolean onMenuItemClick(MenuItem item) {
                    if (item.getTitle().equals(getString(R.string.create_new_cash_receipt)))
                        startActivity(new Intent(CashReceipts.this.getActivity(), NewCashReceipt.class).putExtra("contacts",jsonContact));
                    return false;
                }
            });
            cashReceiptsPopMenu.show();
        }
    });
}
```

Για την τοποθέτηση των αντικειμένων των ListView, χρησιμοποιούμε έναν Adapter όπου εισχωρούμε, στο xml των αντικειμένων, και τοποθετούμε τα δεδομένα στα κατάλληλα πεδία. Και αυτός ο adapter χρησιμοποιεί την μέθοδο του ViewHolder για να εξασφαλίσουμε ταχύτητα στην απεικόνιση του UI των ListView. Το κάθε αντικείμενο έχει μια ημερομηνία, μια τιμή, το όνομα του πελάτη, ένα κουμπί (αν υπάρχει) για τον λόγο και ένα εικονίδιο με τον τύπο της κάθε είσπραξης ή πληρωμής. Στο κουμπί έχει τοποθετηθεί ένας onClickListener όπου εμφανίζει σαν popup το κείμενο του λόγου.



4.6.1.2 *Υλοποίηση των Fragment παιδιών*

Για την λειτουργία της απεικόνισης των fragments, πρέπει να επεξεργαστούμε τα κατάλληλα JSON δεδομένα που πρέπει να χρησιμοποιήσουμε. Για αρχή, στην onCreateView του κάθε fragment δηλώνουμε το view, δηλαδή το αρχείο .xml που περιέχεται το πως θα φαίνεται στον χρήστη το καθένα και τα πεδία που θα χρησιμοποιήσουμε. Αυτό γίνεται με την εντολή *View v = inflater.inflate(R.layout.tab_cash_receipts, container, false);* και στο τέλος επιστρέφουμε αυτό το view. Έπειτα παίρνουμε τα JSON δεδομένα από το Activity της κύριας οθόνης με την Intent *thisIntent = getActivity().getIntent();* Και την μέθοδο .getStringExtra("String"). Αφού τα αποθηκεύουμε σε String μεταβλητές, χρησιμοποιούμε τις μεθόδους JSONObject και JSONArray για να πάρουμε τα απαραίτητα στοιχεία από τα πεδία των JSON.

```
try {
    JSONObject getResults = new JSONObject(jsonContact);
    JSONArray contactList = getResults.getJSONArray("results");

    JSONObject json = new JSONObject(jsonString);
    json.getJSONArray("results");
    json.getInt("count");
    JSONArray cashReceiptsFields = json.getJSONArray("results");
    ListView listView = (ListView) v.findViewById(R.id.list_view_cash_receipts);

    if (json.getInt("count")>0) {
        TextView message = (TextView) v.findViewById(R.id.no_records_cash_receipts_textView);
        message.setVisibility(View.GONE);
        ArrayList<CashReceipt> arrayOfCashReceipts = new ArrayList<~>();

        // Create the adapter to convert the array to views
        CashReceiptsAdapter adapter = new CashReceiptsAdapter(this.getActivity(), arrayOfCashReceipts);
        // Attach the adapter to a ListView

        listView.setAdapter(adapter);
        reason = (Button) v.findViewById(R.id.reason_button_cash_receipts);
        reason.setTypeface(Typeface.createFromAsset(reason.getContext().getAssets(), "fonts/elorus-font.ttf"));
```

Εικόνα 61: Διαχείριση JSON δεδομένων

Για την σωστή διαχείριση των αντικειμένων, Cash Receipts και Cash Payments, πρέπει να τα δούμε σαν ξεχωριστές οντότητες και να δημιουργήσουμε μια Java κλάση για το καθένα με τα στοιχεία, τον constructor τους και τις μεθόδους τους. Όπως, για παράδειγμα, βλέπουμε στην παρακάτω εικόνα κάθε Cash Payment θα έχει μοναδικά χαρακτηριστικά.

```
public class CashPayment {
    public String date;
    public String amount;
    public String contactId;
    public String title;
    public String id;
    public String transaction_type;


    public CashPayment(String transaction_type,String id,String date, String amount, String contactId, String title) throws JSONException {
        this.date = date;
        this.amount = amount;
        this.contactId = contactId;
        this.title = title;
        this.id=id;
        this.transaction_type=transaction_type;

        // CallContact();
    }

    public CashPayment(JSONObject object) {
        try {
            this.date = object.getString("date");
            this.amount = object.getString("amount");
            this.contactId = object.getString("contact");
            this.title = object.getString("title");
            this.id=object.getString("id");
            this.transaction_type=object.getString("transaction_type");

        } catch (JSONException e) {
            e.printStackTrace();
        }
    }

    public static ArrayList<CashPayment> fromJson(JSONArray jsonObjects) {
        ArrayList<CashPayment> cashPayments = new ArrayList<CashPayment>();
        for (int i = 0; i < jsonObjects.length(); i++) {
            try {
                cashPayments.add(new CashPayment(jsonObjects.getJSONObject(i)));
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
        return cashPayments;
    }
```

Εικόνα 62: Παράδειγμα οντότητας Cash Payment

Στην συνέχεια πρέπει να γίνει ο έλεγχος για πόσα JSONObject από Cash Payments ή Cash Receipts υπάρχουν αντίστοιχα. Σε αυτό μας βοηθάει ένα πεδίο του JSONObject των δεδομένων, το “count”. Μας δίνει τον αριθμό από τα Cash Receipts ή Cash Payments που υπάρχουν. Για κάθε ένα από αυτά δημιουργούμε μια οντότητα Cash Receipt ή Cash Payment που ορίζεται με βάση τον Constructor τους. Στο τέλος προσθέτουμε στον Adapter τις οντότητες αυτές.

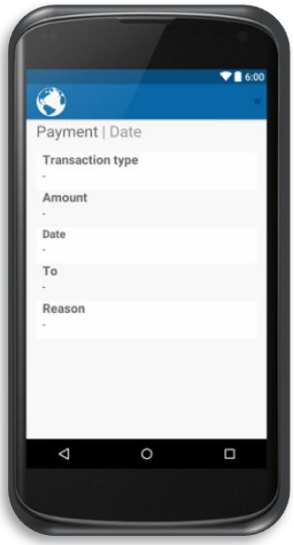
```
for (int i = 0; i < json.getInt("count"); i++) {
    String displayName= null;
    for (int j = 0; j < getResults.getInt("count"); j++)
        if(contactList.getJSONObject(j).getString("id").equals(cashReceiptsFields.getJSONObject(i).getString("contact")))
            displayName=contactList.getJSONObject(j).getString("display_name");

    CashReceipt cashReceipt = new CashReceipt(cashReceiptsFields.getJSONObject(i).getString("transaction_type"),cashReceiptsFields.getJSONObject(i).getString("id"),cashReceiptsFields.getJSONObject(i).getStr
    adapter.add(cashReceipt);
}
```

4.6.2 Εμφάνιση στοιχείων συναλλαγής

Όταν ο χρήστης κάνει click πάνω σε κάποιο αντικείμενο από την λίστα του ListView καλείται ένα νέο Activity για την εμφάνιση των περιεχόμενων αλλά και λεπτομερειών΄, που δεν εμφανίζονται στο ListView, του κάθε Cash Receipt ή Cash Payment. Αυτή η λειτουργία υπάρχει στους Adapter των Cash Payment και Cash Receipt. Για παράδειγμα, για να δούμε ένα Cash Receipt:

```
Intent viewCashReceipt = new Intent(v.getContext(), ViewCashReceipt.class).putExtra("id",
cashReceipt.id).putExtra("contacts", ((Activity) v.getContext()).getIntent().getStringExtra("contacts"));
getContext().startActivity(viewCashReceipt);
```

Έτσι εμφανίζουμε το ViewCashReceipt ή το ViewCashPayment Activity. Στην παρακάτω εικόνα βλέπουμε την σχεδιαστική μορφή που θα έχει το View του Activity για ένα Cash Payment. Αποτελείται από ένα custom Toolbar που έχει το logo της εφαρμογής, στο οποίο υπάρχει link για να την μετάβαση στην κύρια οθόνη. Στα αριστερά του Toolbar τοποθετείται ένα εικονίδιο για την επιλογή της επεξεργασίας του Cash Payment ή του Cash Receipt. Για να δείξουμε τα σωστά πράγματα στον χρήστη, πρέπει να κάνουμε ένα Get Call με το συγκεκριμένο αντικείμενο. Αυτό γίνεται επειδή έχουμε μεταβεί σε ένα δεύτερο Activity και δεν μπορούμε να πάρουμε μεταβιβασμένα δεδομένα από το intent της κύριας οθόνης. Γι' αυτό όταν γίνεται η κλίση του Activity αυτού, βάζουμε ως extra στοιχείο το id του αντικειμένου του ListView που επιλέχθηκε.

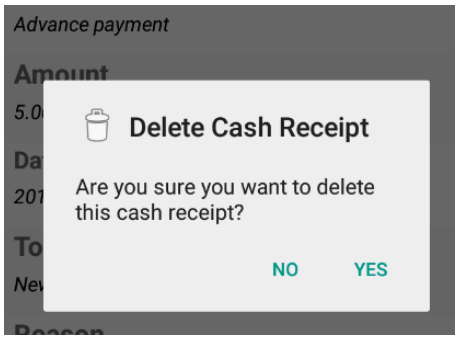


Εικόνα 63: View Cash Payment Raw view

```
RestCalls getCall = new RestCalls("cashpayments/" + id);
getCall.GetCall();
jsonString = getCall.getJsonString();

try {
    JSONObject json = new JSONObject(jsonString);
    String selectedVal = getResources().getStringArray(R.array.transaction_array)[Arrays.asList(get

    //tranfer values of cash payment to Edit Cash Payment Intent
    editCashPayment.putExtra("transaction_type", selectedVal);
    editCashPayment.putExtra("title", json.getString("title"));
    editCashPayment.putExtra("amount", json.getString("amount"));
    editCashPayment.putExtra("date", json.getString("date"));
    editCashPayment.putExtra("contact", json.getString("contact"));
    editCashPayment.putExtra("contacts", jsonContact);

    payment = (TextView) findViewById(R.id.payment_view);
    payment.setText(selectedVal);

    dateView = (TextView) findViewById(R.id.date_view);
    if (!json.getString("date").equals(""))
        dateView.setText(json.getString("date"));

    transaction = (TextView) findViewById(R.id.transaction_type_view);
    transaction.setText(selectedVal);

    amount = (TextView) findViewById(R.id.amount_value_view);
    if (!json.getString("amount").equals(""))
        amount.setText(json.getString("amount"));

    date = (TextView) findViewById(R.id.date_value_view);
    if (!json.getString("date").equals(""))
        date.setText(json.getString("date"));

    to = (TextView) findViewById(R.id.to_value_view);
    JSONObject getResults = new JSONObject(jsonContact);
    JSONArray contactList = getResults.getJSONArray("results");
    for (int j = 0; j < getResults.getInt("count"); j++) {
        String displayName;
        if (!json.getString("contact").equals("")) {
            if (contactList.getJSONObject(j).getString("id").equals(json.getString("contact"))) {
                displayName = contactList.getJSONObject(j).getString("display_name");
                to.setText(displayName);
            }
        }
    }
    reason = (TextView) findViewById(R.id.reason_value_view);
    if (!json.getString("title").equals(""))
        reason.setText(json.getString("title"));
```

Εικόνα 64: Εμφάνιση δεδομένων στο View του αντικειμένου

Στο View του activity υπάρχει ένα Options Menu όπου ο χρήστης μπορεί να διαγράψει το συγκεκριμένο Cash Payment ή Cash Receipt. Γίνεται η εμφάνιση popup μηνύματος “delete”, όπου αν επιλεχθεί εμφανίζεται ένα Alert Dialog για να ρωτήσει τον χρήστη αν είναι σίγουρος για την διαγραφή.



Εικόνα 65: Μήνυμα διαγραφής αντικειμένου

```
        new AlertDialog.Builder(this, R.style.MyAlertDialogStyle)//styling the dialog
                .setTitle("Delete Cash Payment")
                .setMessage("Are you sure you want to delete this cash payment?")
                .setPositiveButton("Yes", (dialog, which) -> {
                        try {
                            deleteCashPayment();
                            dialog.cancel();
                        } catch (MalformedURLException e) {
                            e.printStackTrace();
                        }
                })
                .setNegativeButton("No", (dialog, which) -> {
                        dialog.cancel();
                })
                .setIcon(android.R.drawable.ic_menu_delete)
                .show();

        return true;
    }

    return super.onOptionsItemSelected(item);
}

public void deleteCashPayment() throws MalformedURLException {
    URL restUrl = new URL("http://api.elorus-staging.com/v1.0/cashpayments/" + getId() + "/");
    new DeleteTask().execute(restUrl);
  //  Log.d("ID", "from DELETE : " + getId());
}

public class DeleteTask extends AsyncTask<URL, Void, String> {
    ProgressDialog progressDialog;

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        progressDialog = ProgressDialog.show(ViewCashPayment.this, "Deleting Cash Payment", "Please Wait...", true);
    }

    @Override
    protected void onPostExecute(String s) {
        super.onPostExecute(s);
        progressDialog.dismiss();
    }

    @Override
    protected String doInBackground(URL... urls) {
        RestCalls deleteCall = new RestCalls(null);
        deleteCall.DeleteCall(urls[0]);
        finish();
        return null;
```

Εικόνα 66: Λειτουργία διαγραφής αντικειμένου

Εάν η επιλογή είναι θετική και ο χρήστης διαγράψει το αντικείμενο, γίνεται ένα Delete Call μέσα από το Activity με την χρήση ενός AsyncTask. Στέλνουμε δηλαδή στον server ένα delete request με το id του αντικειμένου που πρέπει να διαγραφεί.

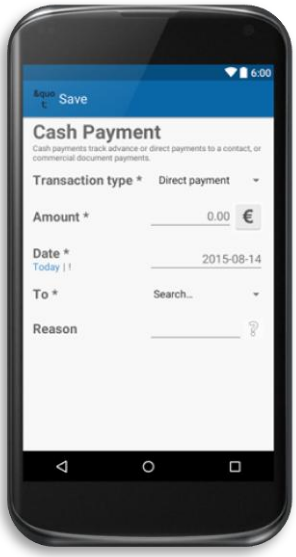
Τέλος το Activity τερματίζεται αφού, στην ουσία, έχει διαγραφεί και επιστρέφουμε στην κύρια οθόνη. Το Activity της έρχεται στην κατάσταση onRestart όπου έχουμε προσθέσει την επαναδημιουργία των Tabs της. Επιπροσθέτως επειδή ξαναδημιουργείται το Activity αυτό, γίνονται ορθά τα REST Calls για τα δεδομένα της εφαρμογής. Με την αναδημιουργία των tabs, θα εμφανιστεί η λίστα από το tab Cash Receipt ή Cash Payment χωρίς το υπάρχον αντικείμενο.

Εάν ο χρήστης θελήσει να επεξεργαστεί τα στοιχεία του Cash Payment ή Cash Receipt, τότε επιλέγει το εικονίδιο με αυτή την λειτουργία. Μόλις γίνει αυτή η ενέργεια, δημιουργείται ένα νέο Activity το EditCashPayment ή EditCashReceipt και το Activity του View του αντικειμένου τερματίζεται.

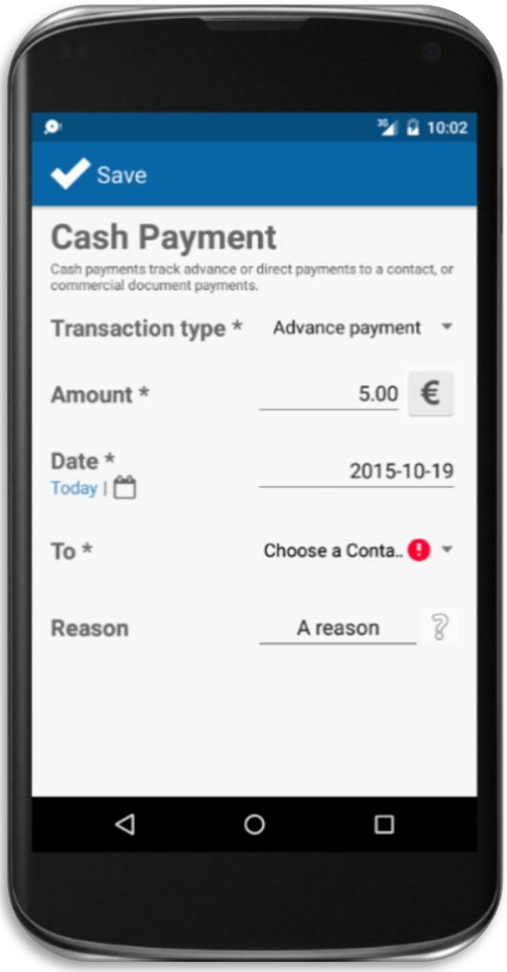
```
editButton = (TextView) findViewById(R.id.editButton);
editButton.setTypeface(Typeface.createFromAsset(editButton.getContext().getAssets(), "fonts/elorus-font.ttf"));
editButton.setTextSize(TypedValue.COMPLEX_UNIT_SP, 35);
editButton.setTextColor(editButton.getResources().getColor(R.color.myWhite));
editButton.setOnClickListener((v) -> {
        startActivity(editCashPayment);
        finish();
});
```

4.6.3 Εμφάνιση φόρμας επεξεργασίας συναλλαγής

Στο Activity που δημιουργείται για την επεξεργασία της συναλλαγής, ο χρήστης μπορεί να αλλάξει ή και να προσθέσει στοιχεία στα κατάλληλα πεδία της φόρμας. Ο σχεδιασμός της φόρμας γίνεται με τα απαραίτητα πεδία που απαιτεί να δεχθεί ο server. Επιπλέον έχει δημιουργηθεί ένα custom Toolbar με ένα κουμπί για την αποθήκευση των δεδομένων και ένα μήνυμα της ενέργειας αυτής “Save”.



Όπως είναι λογικό εφόσον μιλάμε για μια φόρμα με την απαίτηση συμπλήρωσης στοιχείων, επιβάλλεται να υπάρχει και validation στις επιλογές του χρήστη. Το κάθε πεδίο έχει ρυθμιστεί για να προσδίδει στον χρήστη τα σωστά inputs αλλά και να τον περιορίσει στις λάθος επιλογές. Για παράδειγμα άμα δεν έχει επιλέξει πελάτη. Στα validation αυτά εμφανίζεται ένα μήνυμα σφάλματος στον χρήστη για να το διορθώσει.

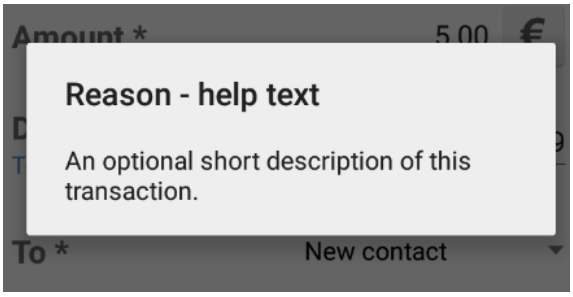



```
public void ValidationContact(final Spinner contact) {
    contact.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
        int check = 0;

        @Override
        public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
            check = check + 1;
            if (check > 1) {
                if (Build.VERSION.SDK_INT > Build.VERSION_CODES.GINGERBREAD_MR1) {
                    if (position == 0) {
                        TextView selectedTextView = (TextView) contact.getSelectedView();
                        String errorString = "This field is required.";
                        selectedTextView.setError(errorString);
                    }
                } else {
                    if (position == 0) {
                        TextView selectedTextView = (TextView) contact.getSelectedView();
                        String errorString = "This field is required.";
                        selectedTextView.setError(errorString);
                    }
                }
            }

        @Override
        public void onNothingSelected(AdapterView<?> parent) {

        }
    });
}

public void ValidationOnClick(Spinner contact) {
    if (Build.VERSION.SDK_INT > Build.VERSION_CODES.GINGERBREAD_MR1) {
        if (!contact.isSelected() && contact.getSelectedItemPosition() < 1) {
            TextView selectedTextView = (TextView) contact.getSelectedView();
            String errorString = "This field is required.";
            selectedTextView.setError(errorString);

        }
    } else {
        if (!contact.isSelected() && contact.getSelectedItemPosition() < 1) {
            TextView selectedTextView = (TextView) contact.getSelectedView();
            String errorString = "This field is required.";
            selectedTextView.setError(errorString);
        }
    }
}
```

Εικόνα 68: Έλεγχος λάθους στη φόρμα

Εικόνα 69: Επιλογή από ημερολόγιο

Για την υλοποίηση του του πρώτου πεδίου έχουμε δημιουργήσει ένα Spinner από 2 επιλογές, Advance Payment και direct Payment όπου επιλέγει ο χρήστης τον τύπο της συναλλαγής. Στο δεύτερο πεδίο πρέπει να τοποθετηθεί ένας αριθμός για το ποσό της συναλλαγής με εισαγωγή ενός Double. Το τρίτο πεδίο έχει σαν εισαγωγή μια ημερομηνία. Αν ο χρήστης επιθυμεί να αλλάξει την υπάρχουσα ημερομηνία του αντικειμένου, μπορεί είτε να βάλει μια καινούρια από το πληκτρολόγιο, είτε να πατήσει την επιλογή “Today” όπου θα εισχωρηθεί η σημερινή ημερομηνία σύμφωνα με την Android συσκευή, είτε να πατήσει το εικονίδιο με το Calendar όπου εμφανίζεται σαν popup ένα ημερολόγιο για να επιλέξει ο χρήστης την ημερομηνία που επιθυμεί. Στο τέταρτο πεδίο ο χρήστης μπορεί να αλλάξει τον πελάτη της συναλλαγής μέσω ενός Spinner, με επιλογές τις επαφές της εφαρμογής. Αν αλλάξει την προκαθορισμένη επιλογή και δεν επιλέξει έναν πελάτη, τότε εμφανίζεται το κατάλληλο validation μήνυμα. Τέλος στο τελευταίο πεδίο ο χρήστης μπορεί να αλλάξει τον λόγο της συναλλαγής αν επιθυμεί ή να βάλει έναν αν δεν είχε το επιλεγμένο για επεξεργασία αντικείμενο. Σε αυτό το πεδίο υπάρχει ένα κουμπί που απεικονίζεται από ένα ερωτηματικό όπου εμφανίζεται ένα μήνυμα σε popup μορφή για την επεξήγηση του πεδίου.



Εικόνα 70: Μήνυμα επεξήγησης

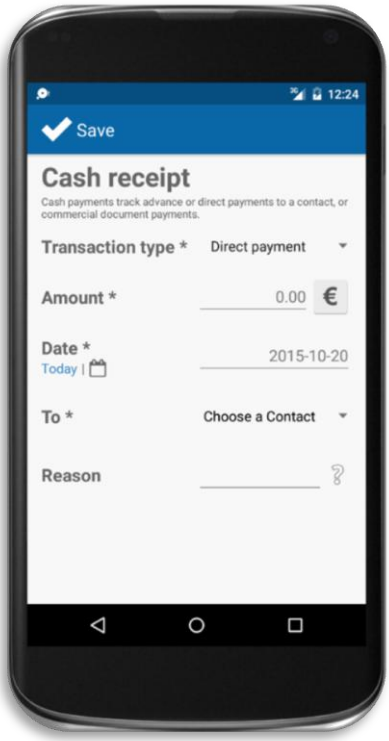
Με την σωστή επεξεργασία των πεδίων ο χρήστης καλείται να τα αποθηκεύσει στον server. Πατώντας το εικονίδιο για την αποθήκευση, αφού γίνει ο έλεγχος των validations, γίνεται PUT Call στον server μέσω ενός AsyncTask. Με το call αυτό έχουμε την δυνατότητα να αλλάξουμε τα στοιχεία των δεδομένων που έχει ο server για αυτή την καταχώρηση. Από το AsyncTask εμφανίζεται ένα Alert dialog που ειδοποιεί τον χρήστη για την διαδικασία αλλαγής των δεδομένων. Όταν τερματιστεί το AsyncTask αυτό κλείνει το Activity της επεξεργασίας και δημιουργείται ένα νέο Activity ViewCashPayment ή ViewCashReceipt αντίστοιχα, με τα νέα στοιχεία της συναλλαγής. Επιπλέον το ίδιο συμβαίνει και όταν ο χρήστης πατήσει το πλήκτρο Back με την μόνη διαφορά ότι δεν γίνεται αλλαγή στα δεδομένα αφού δεν καλείται το AsyncTask για την επεξεργασία της συναλλαγής.

```
@Override
public void onBackPressed() {
    Intent viewCashPayment = new Intent(EditCashPayment.this, ViewCashPayment.class).putExtra("id", idExtra).putExtra("contacts", jsonContact);
    startActivity(viewCashPayment);
    finish(); //close intent
}
```

```
public class ResponseTask extends AsyncTask<String, Void, String> {
    ProgressDialog progressDialog;
    RestCalls PutCall;
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        progressDialog = ProgressDialog.show(EditCashPayment.this, "Updating Cash Payment", "Please Wait...", true);
    }


    @Override
    protected void onPostExecute(String s) {
        super.onPostExecute(s);
        progressDialog.dismiss();
        try {
            if (PutCall.getPut().getResponseMessage().equals("OK"))
                finish();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    protected String doInBackground(String... params) {
        String accountInfo = params[0];
        PutCall = new RestCalls("cashpayments/" + idExtra + "/");
        PutCall.setJsonString(accountInfo);
        PutCall.PutCall();
        try {
            if (PutCall.getPut().getResponseMessage().equals("OK")) {
                PutCall.getPut().disconnect();
                Intent viewCashPayment = new Intent(EditCashPayment.this, ViewCashPayment.class).putExtra("id", idExtra).putExtra("contacts", jsonContact);
                startActivity(viewCashPayment);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

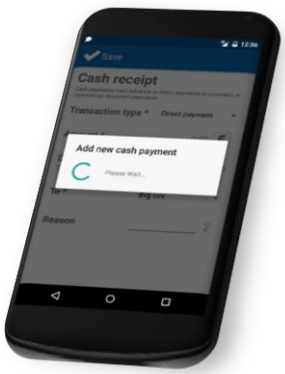
4.6.4 Εμφάνιση φόρμας δημιουργίας συναλλαγής

Όταν επιλέξει ο χρήστης να δημιουργήσει νέα συναλλαγή, δημιουργείται ένα νέο Activity όπως και στην εμφάνιση κάποιας συναλλαγής. Το οποίο έχει τον ίδιο σχεδιασμό με την φόρμα επεξεργασίας, με την διαφορά ότι τα πεδία είναι άδεια για την συμπλήρωση του χρήστη με καινούριες επιλογές.



Εικόνα 71: Φόρμα δημιουργίας συναλλαγής

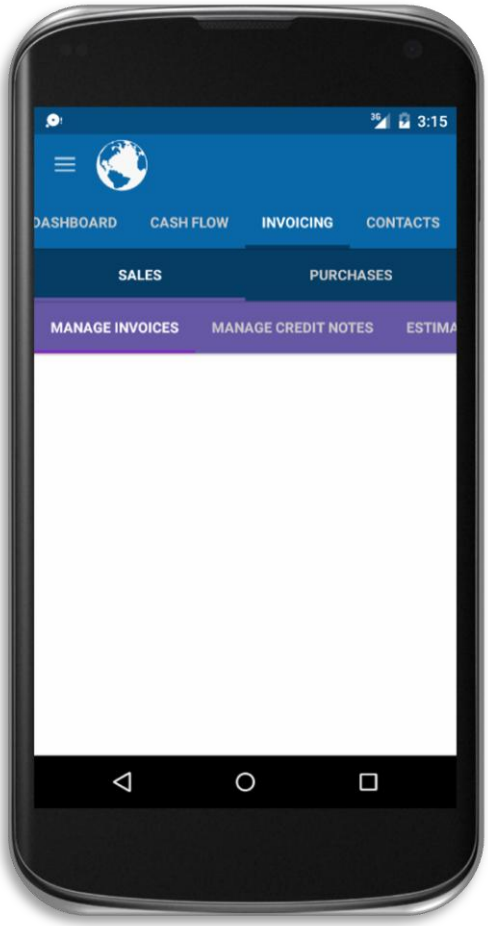
Ο χρήστης έχει την απαίτηση να διαλέξει μόνο πελάτη για την αποθήκευσή της συναλλαγής. Χωρίς να βάλει ημερομηνία ο χρήστης επιλέγεται η σημερινή σύμφωνα με την συσκευή Android και η προεπιλεγμένη τιμή είναι 0. Για την δημιουργία νέας καταχώρησης συναλλαγής στον server γίνεται ένα POST CALL μόλις πατηθεί το αρμόδιο εικονίδιο μέσω ενός AsyncTask. Ο χρήστης ενημερώνεται για την διαδικασία αποστολής δεδομένων στον server και μόλις τελειώσει η διαδικασία τερματίζεται το Activity και μεταβαίνουμε στην κύρια οθόνη.



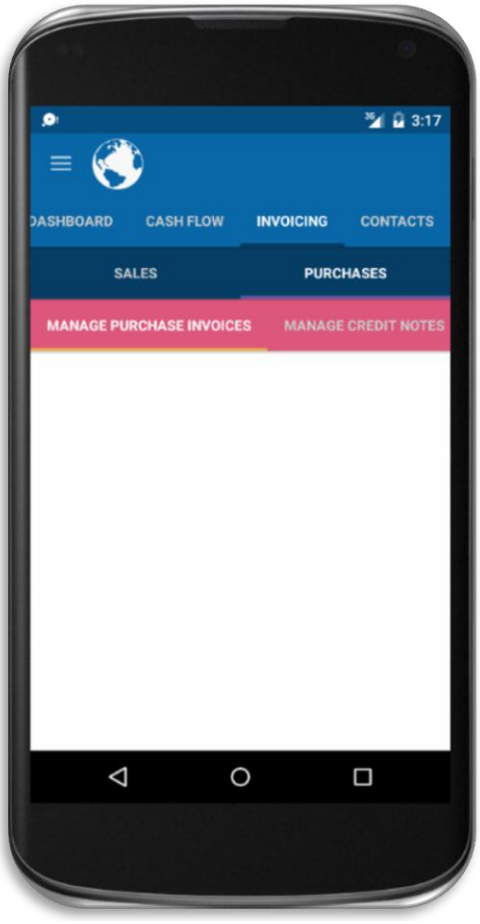
Εικόνα 72: Δημιουργία συναλλαγής

4.7 Invoicing

Στο τρίτο Tab (fragment) γίνεται η παρακολούθηση των παραστατικών. Περιλαμβάνει δύο κατηγορίες, τις πωλήσεις και τις αγορές. Σε αυτές μπορούμε να δούμε τις λίστες των παραστατικών και πιστωτικών παραστατικών πωλήσεων και αγορών αλλά και την λίστα προσφορών πωλήσεων. Για την υλοποίηση και σχεδίαση των κατηγοριών αυτών χρησιμοποιούνται οι κλάσεις SlidingTabLayout και ViewPager όπου η κάθε μια εμφανίζει τις δικές τις κατηγορίες που αναφέραμε με την βοήθεια των κλάσεων αυτών. Η μόνη διαφορά είναι στην απεικόνιση των δεδομένων που παίρνουμε από τον server με τα GET Calls που γίνονται στο Activity της κύριας οθόνης.



Εικόνα 74: Fragment Πωλήσεων

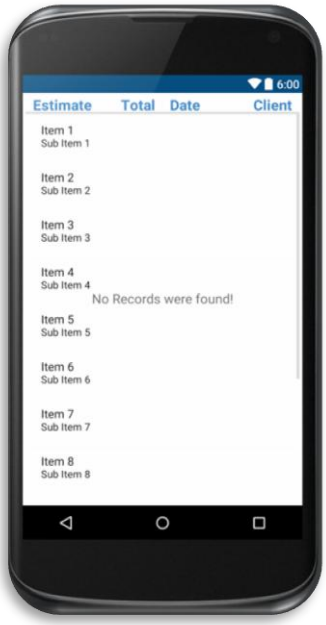


Εικόνα 73: Fragment Αγορών

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <tabs.SlidingTabLayout
        android:id="@+id/tabs_invoicing"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:elevation="2dp"
        android:background="@color/tabsScrollColor"/>

    <android.support.v4.view.ViewPager
        android:id="@+id/pager_invoicing"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1" />

</LinearLayout>
```

4.7.1 Λειτουργία του fragment Invoicing

Για την λειτουργία του fragment δημιουργήσαμε έναν Adapter όπως και στα προηγούμενα fragments, το οποίο τοποθετεί στα κατάλληλα Tabs τις κλάσεις των Sales και Purchases τα οποία είναι με την σειρά τους νέα fragments παιδία του Fragment Invoicing. Τα fragments παιδιά αυτά έχουν δικά τους fragments παιδιά με τις κατηγορίες τους, τα οποία τοποθετούνται στις καρτέλες γονείς με την χρήση Adapters.



Εικόνα 75: Raw Estimates Fragment List

```
//This method return the fragment for the every position in the View Pager
@Override
public Fragment getItem(int position) {

    if (position == 0) // if the position is 0 we are returning the First ta
    {
        return new Sales();
    }
    if (position == 1)            // As we are having 2 tabs if the position
    {
        return new Purchases();
    } else
        return null;
```

4.7.2 Σχεδίαση των fragment παιδιών Sales και Purchases

Η απεικόνιση των fragments παιδιών των Sales και Purchases, βασίζεται στην δημιουργία μίας λίστας για το καθένα με την βοήθεια του ListView, βλέπε εικόνα 75. Αν η λίστα είναι κενή, δηλαδή αν δεν υπάρχει καταχώρηση δεδομένων στο Server, εμφανίζεται ένα TextView με το απαραίτητο μήνυμα «No Records were found!» αλλιώς το TextView γίνεται αόρατο με την μέθοδο *.setVisibility(View.INVISIBLE);*.

```
<TextView
    android:id="@+id/no_records_estimates_textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:text="No Records were found!"
    android:textAppearance="?android:attr/textAppearanceMedium" />
```

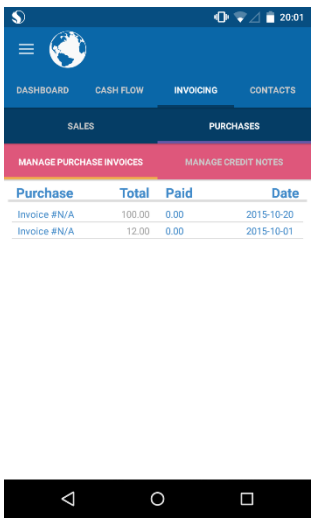
4.7.3 Υλοποίηση των fragment παιδιών Sales και Purchases

Για την λειτουργία της απεικόνισης των fragments, πρέπει να επεξεργαστούμε τα κατάλληλα JSON δεδομένα που πρέπει να χρησιμοποιήσουμε. Για αρχή, στην onCreateView του κάθε fragment δηλώνουμε το view, δηλαδή το αρχείο .xml που περιέχεται το πως θα φαίνεται στον χρήστη το καθένα και τα πεδία που θα χρησιμοποιήσουμε. Αυτό γίνεται με την εντολή *View v = inflater.inflate(R.layout.fragment_estimates, container, false);* και στο τέλος επιστρέφουμε αυτό το view. Έπειτα παίρνουμε τα JSON δεδομένα από το Activity της κύριας οθόνης με την Intent *thisIntent = getActivity().getIntent();* Και την μέθοδο .getStringExtra("String"). Αφού τα αποθηκεύουμε σε String μεταβλητές, χρησιμοποιούμε τις

μεθόδους JSONObject και JSONArray για να πάρουμε τα απαραίτητα στοιχεία από τα πεδία των JSON.

```
Intent thisIntent = getActivity().getIntent();
// Log.d("CHECK",""+thisIntent.getStringExtra("estimates"));
jsonString = thisIntent.getStringExtra("estimates");
try {
    JSONObject json = new JSONObject(jsonString);
    json.getJSONArray("results");
    json.getInt("count");
    JSONArray estimatesFields = json.getJSONArray("results");
    ListView listView = (ListView) v.findViewById(R.id.list_view_estimates);

    if (json.getInt("count")>0) {
        TextView message = (TextView) v.findViewById(R.id.no_records_estimates_textView);
        message.setVisibility(View.GONE);
        ArrayList<Estimate> arrayOfEstimates = new ArrayList<~>();

        // Create the adapter to convert the array to views
        EstimatesAdapter adapter = new EstimatesAdapter(this.getActivity(), arrayOfEstimates);
        // Attach the adapter to a ListView

        listView.setAdapter(adapter);
        for (int i = 0; i < json.getInt("count"); i++) {
            Estimate estimate = new Estimate(estimatesFields.getJSONObject(i));
            adapter.add(estimate);
        }
    }
} catch (Exception e) {
    Log.e("EXCEPTION", "exception", e);
}
return v;
```

Για την τοποθέτηση των αντικειμένων των ListView, χρησιμοποιούμε έναν Adapter όπου εισχωρούμε, στο xml των αντικειμένων, και τοποθετούμε τα δεδομένα στα κατάλληλα πεδία. Και αυτός ο adapter χρησιμοποιεί την μέθοδο του ViewHolder για να εξασφαλίσουμε ταχύτητα στην απεικόνιση του UI των ListView. Το κάθε αντικείμενο έχει το όνομα του παραστατικού ή της προσφοράς, μια τιμή του ποσού του παραστατικού, μια τιμή από το ποσό που έχει πληρωθεί και μια ημερομηνία.



Οι επιλογές του χρήστη στο Tab Invoicing είναι μόνο για παρακολούθηση των καταχωρίσεων από τα δεδομένα που λαμβάνουμε από τον server. Δεν του δίνεται η δυνατότητα να δημιουργήσει ή να επεξεργαστεί μέσω της εφαρμογής κάποιο παραστατικό ή προσφορά. Η δημιουργία και επεξεργασία γίνεται από την online πλατφόρμα του συγκεκριμένου API.

Η παρακολούθηση των παραστατικών από την εφαρμογή, δίνει στον χρήστη την δυνατότητα να έχει την πλήρη επίγνωση για όλες τις αγορές και πωλήσεις που έχει ή έχουν δημιουργηθεί αλλά και την εξόφλησή τους. Με τελικό σκοπό την πρόοδο της επιχείρησής του.

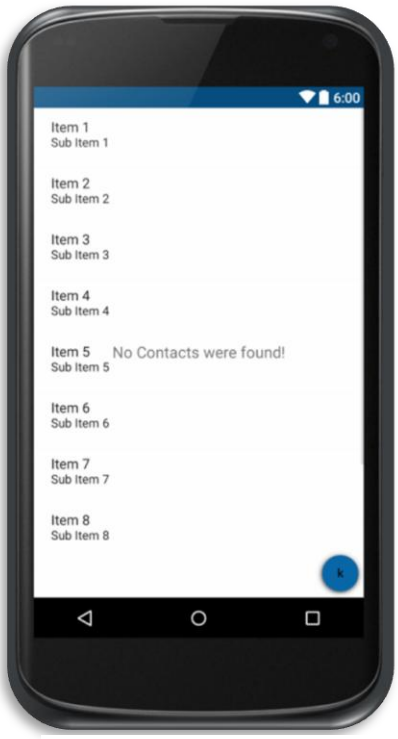
4.8 Contacts

Το τέταρτο Tab (fragment) της κύριας οθόνης αποτελείται από τις επαφές της εφαρμογής. Ίσως η πιο χρήσιμη λειτουργία αφού για να δημιουργήσει μια συναλλαγή ή ένα παραστατικό μέσω της online πλατφόρμας πρέπει να γίνει η σύνδεσή τους με μια επαφή.

Σε αυτή την καρτέλα ο χρήστης μπορεί δει τις επαφές του, να δημιουργήσει καινούριες ή να προσθέσει επαφές από την Android συσκευή του, να στείλει σε κάποια επαφή email ή να πάρει τηλέφωνο αλλά και να επεξεργαστεί τα στοιχεία της. Σαν πρώτη όψη ο χρήστης βλέπει μια λίστα από επαφές με το όνομα και το email της κάθε μιας όπως και ένα εικονίδιο για την προσθήκη καινούριας. Η εμφάνιση της λίστας των επαφών γίνεται με ένα ListView που με την βοήθεια ενός Adapter προστίθεται κάθε επαφή σαν αντικείμενο της λίστας.

```
public class ContactsAdapter extends ArrayAdapter<Contact> {
    // View lookup cache
    private static class ViewHolder {
        TextView displayName;
        TextView email;
    }
    public ContactsAdapter(Context context, ArrayList<Contact> contacts) {
        super(context, 0, contacts);
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        // Get the data item for this position
        final Contact contact = getItem(position);
        final ViewHolder viewHolder; // view lookup cache stored in tag
        if (convertView == null) {
            viewHolder = new ViewHolder();
            LayoutInflater inflater = LayoutInflater.from(getContext());
            convertView = inflater.inflate(R.layout.contacts_row, parent, false);
            viewHolder.displayName = (TextView) convertView.findViewById(R.id.contactsDisplayName);
            viewHolder.email=(TextView) convertView.findViewById(R.id.contacts_email);

            convertView.setTag(viewHolder);
        }else {
            viewHolder = (ViewHolder) convertView.getTag();
        }
```

Εικόνα 77: Υλοποίηση Adapter επαφών

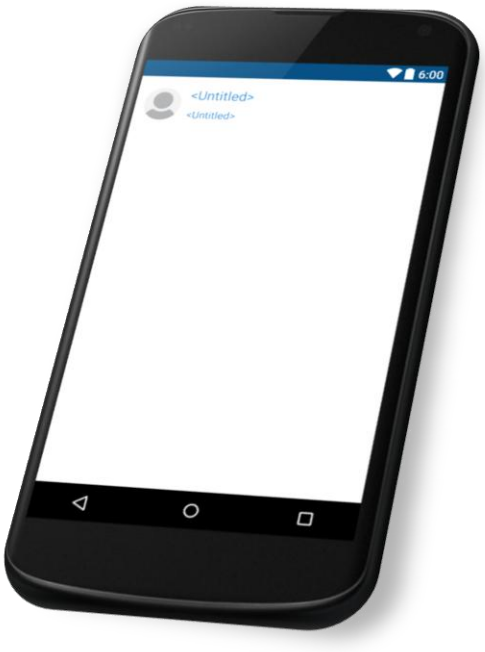


Εικόνα 76: Raw λίστα για τις επαφές

4.8.1 Σχεδίαση Επαφών

Για την σχεδίαση των επαφών έχει δημιουργηθεί ένα Layout που έχει ένα ImageView για το εικονίδιο της κάθε επαφής, ένα Textview με το όνομα και ένα Textview με το email της.

```
<de.hdodenhof.circleimageview.CircleImageView
    android:id="@+id/contactsIcon"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:layout_marginTop="10dp"
    android:contentDescription="@android:string/untitled"
    android:src="@drawable/ic_profile" />

<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#ffffff"
    android:orientation="vertical"
    android:layout_gravity="center">

    <TextView
        android:id="@+id/contactsDisplayName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:paddingTop="10dp"
        android:paddingRight="10dp"
        android:text="@android:string/untitled"
        android:textSize="18sp"
        android:textColor="@color/myBlueButton"/>
    <TextView
        android:id="@+id/contacts_email"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:paddingTop="10dp"
        android:paddingRight="10dp"

        android:text="@android:string/untitled"
        android:textSize="14sp"
        android:textColor="@color/myBlueButton"/>
</LinearLayout>
</LinearLayout>
```

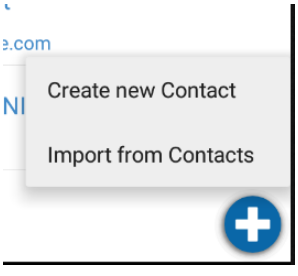


Εικόνα 78: Σχεδίαση επαφής

4.8.2 Υλοποίηση επαφών

Για αρχή πρέπει να δηλώσουμε το Layout του fragment με την εντολή *View v = inflater.inflate(R.layout.tab_contacts, container, false);* Να μορφοποιήσουμε το εικονίδιο για την προσθήκη επαφών αλλά και να του δώσουμε την λειτουργία να εμφανίζεται ένα popup μήνυμα με τις επιλογές δημιουργίας νέας επαφής και προσθήκης από την συσκευή Android.

```
public void CircleImageButton(View v) {
    circleImageButton = (Button) v.findViewById(R.id.circleImageButton);
    circleImageButton.setTypeface(Typeface.createFromAsset(circleImageButton.getContext().getAssets(), "fonts/elorus-font.ttf"));
    circleImageButton.setTextSize(TypedValue.COMPLEX_UNIT_SP, 35);
    circleImageButton.setTextColor(circleImageButton.getResources().getColor(R.color.myWhite));
    circleImageButton.setOnClickListener((v) -> {
            PopupMenu contactPopMenu = new PopupMenu(getActivity(), circleImageButton);
            contactPopMenu.getMenuInflater().inflate(R.menu.contacts_fragment, contactPopMenu.getMenu());
            contactPopMenu.setOnMenuItemClickListener((item) -> {
                    if (item.getTitle().equals("Create new Contact"))
                        startActivity(new Intent(Contacts.this.getActivity(), NewContact.class));

                    if (item.getTitle().equals("Import from Contacts")) {
                        addContact = new Intent(Intent.ACTION_PICK, ContactsContract.Contacts.CONTENT_URI);

                        startActivityForResult(addContact, PICK_CONTACT);
                    }
                    return false;
            });
            contactPopMenu.show();
    });
}
```



Εικόνα 79: Μορφοποίηση εικονιδίου προσθήκης επαφής

Έπειτα πρέπει να πάρουμε από το JSON με τις επαφές, από το Activity, τα JSONObject των επαφών και με την βοήθεια του Adapter να γεμίσουμε την λίστα. Αν δεν υπαρχει κάποια επαφή η λιστα θα είναι κενή και εμφανίζεται το μήνυμα “No Records were found!” από ένα TextView. Αλλιώς το TextView εξαφανίζεται και μαζί του το μήνυμα με την μέθοδο *.setVisibility(View.INVISIBLE);*. Για την απεικόνιση των δεδομένων πρέπει να επεξεργαστούμε τα JSON αντικείμενα που πήραμε από τα Get Calls στο Activity της κύριας οθόνης.

```
Intent thisIntent = getActivity().getIntent();
jsonString = thisIntent.getStringExtra("contacts");
try {
    JSONObject json;
    json = new JSONObject(jsonString);
    JSONArray contactFields = json.getJSONArray("results");

//   Log.d("myJason", "Json Tag for COUNT" + json.getInt("count"));
 //  Log.d("myJason", "Json Tag for RESULTS" + json.getJSONArray("results"));
   if (json.getInt("count") > 0) {
       message.setVisibility(View.GONE);
       // Construct the data source
       ArrayList<Contact> arrayOfContacts = new ArrayList<>();
       // Create the adapter to convert the array to views
       ContactsAdapter adapter = new ContactsAdapter(Contacts.this.getActivity(), arrayOfContacts);
       // Attach the adapter to a ListView
       scrollbar.setAdapter(adapter);

       for (int i = 0; i < json.getInt("count"); i++) {
           Contact newContact = new Contact(contactFields.getJSONObject(i));
           adapter.add(newContact);

       }
   }
```

4.8.3 Οντότητα μιας επαφής

Για να μπορέσει ο adapter να επεξεργαστεί τα στοιχεία της κάθε επαφής πρέπει να φτιάξουμε μια Java κλάση με τον σκοπό η κάθε επαφή να θεωρηθεί σαν ξεχωριστή οντότητα. Να έχει δηλαδή τα δικά της μοναδικά στοιχεία, τον constructor της και τις μεθόδους της.

```
public class Contact {
    public String displayName;
    public String email;
    public String id;

    public Contact(String displayName, String id) {
        this.displayName = displayName;
        this.id = id;
    }

    public Contact(JSONObject object) {
        try {
            this.displayName = object.getString("display_name");
            this.id = object.getString("id");
            this.email=object.getJSONArray("email").getJSONObject(0).getString("email");
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }


    public static ArrayList<Contact> fromJson(JSONArray jsonObjects) {
        ArrayList<Contact> contacts = new ArrayList<>();
        for (int i = 0; i < jsonObjects.length(); i++) {
            try {
                contacts.add(new Contact(jsonObjects.getJSONObject(i)));
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
        return contacts;
    }
}
```

Εικόνα 80: Οντότητα επαφής

4.8.4 Λειτουργία αποστολής email

Όταν ο χρήστης επιλέξει το email που απεικονίζεται σε κάθε επαφή, του δίνεται η δυνατότητα αποστολής email σε αυτή. Αυτό γίνεται με επιλογή από email servers/providers που υπάρχουν στην συσκευή Android για παράδειγμα, Gmail, Yahoo mail, Skype κλπ. Η υλοποίηση της λειτουργίας αυτής γίνεται στον Adapter των επαφών.

```
// Populate the data into the template view using the data object
viewHolder.email.setText(contact.email);
viewHolder.email.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent email = new Intent(Intent.ACTION_SEND);
        email.putExtra(Intent.EXTRA_EMAIL, new String[]{viewHolder.email.getText().toString()});
        email.setType("message/rfc822");
        v.getContext().startActivity(Intent.createChooser(email, "Choose an Email client :"));
    }
});
```

Με το που στείλει ή επιλέξει να φύγει από την λειτουργία αυτή ο χρήστης, μεταβαίνετε πίσω στην εφαρμογή και πιο συγκεκριμένα στο Activity της κύριας οθόνης.

4.8.5 Λειτουργία προβολής επαφής

Όταν ο χρήστης επιλέξει το όνομα μιας επαφής, δημιουργείται ένα νέο Activity (ViewContact) όπου ο χρήστης μπορεί να δει περισσότερα στοιχεία της επαφής. Του δίνεται η επιλογή της επεξεργασίας και της διαγραφής της επαφής αυτής.

Εικόνα 81: View Contact Raw design

Σε αυτό το Activity έχει σχεδιαστεί ένα custom Toolbar με το Logo της εφαρμογής πού έχει link μετάβασης στο Activity της κεντρικής οθόνης και ένα εικονίδιο για την επεξεργασία των στοιχείων της εφαρμογής. Επιπλέον στα στοιχεία των TextView του Activity της προβολής της επαφής, τοποθετούνται τα κατάλληλα texts με βάση τα δεδομένα που έχει η κάθε επαφή από τα JSON που έχουμε πάρει από τα Get Call της κύριας οθόνης. Αυτά μεταβιβάζονται ως extra στοιχεία όταν ξεκινάει το Activity.

```
//Show the toolbar in order to show the menu!
toolbar = (Toolbar) findViewById(R.id.tool_bar_view_contact);
setSupportActionBar(toolbar);
assert getSupportActionBar() != null;
ImageButton logo = (ImageButton) toolbar.findViewById(R.id.logo);
logo.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        finish();
    }
});
```

Για την λειτουργία του Activity (ViewContact) κάνουμε ένα Get Call με το id της επαφής που θέλει ο χρήστης να δει. Το id μεταβιβάζεται από το προηγούμενο Activity με την μέθοδο .getStringExtra("String"). Από το call, διαχειριζόμαστε το JSONObject του «contact» και βάζουμε στα απαραίτητα TextViews τα στοιχεία της επαφής.

```
try {
    JSONObject json = new JSONObject(jsonString);
    //tranfer values of contact to Edit Contact Intent
    editContact.putExtra("first_name", json.getString("first_name"));
    editContact.putExtra("last_name", json.getString("last_name"));
    editContact.putExtra("is_client", json.getBoolean("is_client"));
    editContact.putExtra("is_supplier", json.getBoolean("is_supplier"));
    editContact.putExtra("profession", json.getString("profession"));
    editContact.putExtra("company", json.getString("company"));
    editContact.putExtra("vat_number", json.getString("vat_number"));

    displayName = (TextView) findViewById(R.id.displayName);
    displayName.setText(json.getString("display_name"));

    amp = (TextView) findViewById(R.id.amp);
    isClient = (TextView) findViewById(R.id.client);
    if (!json.getBoolean("is_client")) {
        isClient.setVisibility(View.GONE);
        amp.setVisibility(View.GONE);
    }

    isSupplier = (TextView) findViewById(R.id.supplier);
    if (!json.getBoolean("is_supplier")) {
        amp.setVisibility(View.GONE);
        isSupplier.setVisibility(View.GONE);
    }

    profession = (TextView) findViewById(R.id.professionName);
    if (!json.getString("profession").equals(""))
        profession.setText(json.getString("profession"));
    vatNumber = (TextView) findViewById(R.id.vatNumber);
    if (!json.getString("vat_number").equals(""))
        vatNumber.setText(json.getString("vat_number"));

    addresses = (LinearLayout) findViewById(R.id.addressParentLayout);
    phones = (LinearLayout) findViewById(R.id.phoneParentLayout);
    emails = (LinearLayout) findViewById(R.id.emailParentLayout);

    ArrayList<String> numberArray = new ArrayList();
    ArrayList<String> phoneTypeArray = new ArrayList();
    for (int i = 0; i < json.getJSONArray("phones").length(); i++) {
        phoneEmpty = (TextView) findViewById(R.id.phoneEmpty);
        phoneEmpty.setVisibility(View.GONE);
        View phonesView = getLayoutInflater().inflate(R.layout.contact_phone, null);
        final TextView phone = (TextView) phonesView.findViewById(R.id.phone);
        phone.setText(json.getJSONArray("phones").getJSONObject(i).getString("number"));
        numberArray.add(i, json.getJSONArray("phones").getJSONObject(i).getString("number"));
```

```
    phone.setOnClickListener((v) -> {
            Intent callIntent = new Intent(Intent.ACTION_CALL);
            callIntent.setData(Uri.parse("tel:"+phone.getText().toString()));
            startActivity(callIntent);
    });

    TextView phoneType = (TextView) phonesView.findViewById(R.id.phoneType);
    phoneType.setText(json.getJSONArray("phones").getJSONObject(i).getString("phone_type"));
    phoneTypeArray.add(i, json.getJSONArray("phones").getJSONObject(i).getString("phone_type"));
    phones.addView(phonesView);
}
editContact.putStringArrayListExtra("number", numberArray);
editContact.putStringArrayListExtra("phone_type", phoneTypeArray);

ArrayList<String> emailArray = new ArrayList();
ArrayList<String> emailTypeArray = new ArrayList();
for (int i = 0; i < json.getJSONArray("email").length(); i++) {
    emailEmpty = (TextView) findViewById(R.id.emailEmpty);
    emailEmpty.setVisibility(View.GONE);
    View emailView = (View) getLayoutInflater().inflate(R.layout.contact_email, null);
    final TextView email = (TextView) emailView.findViewById(R.id.email);
    email.setText(json.getJSONArray("email").getJSONObject(i).getString("email"));
    emailArray.add(i, json.getJSONArray("email").getJSONObject(i).getString("email"));
    email.setOnClickListener((v) -> {
            Intent emailSend = new Intent(Intent.ACTION_SEND);
            emailSend.putExtra(Intent.EXTRA_EMAIL, new String[]{email.getText().toString()});
            emailSend.setType("message/rfc822");
            v.getContext().startActivity(Intent.createChooser(emailSend, "Choose an Email client :"));
    });

    TextView emailType = (TextView) emailView.findViewById(R.id.emailType);
    emailType.setText(json.getJSONArray("email").getJSONObject(i).getString("email_type"));
    emailTypeArray.add(i, json.getJSONArray("email").getJSONObject(i).getString("email_type"));
    emails.addView(emailView);
}
editContact.putStringArrayListExtra("email", emailArray);
editContact.putStringArrayListExtra("email_type", emailTypeArray);

ArrayList<String> addressArray = new ArrayList();
ArrayList<String> countryArray = new ArrayList();
ArrayList<String> adTypeArray = new ArrayList();
for (int i = 0; i < json.getJSONArray("addresses").length(); i++) {
    addressesEmpty = (TextView) findViewById(R.id.addressesEmpty);
    addressesEmpty.setVisibility(View.GONE);
    View addressView = getLayoutInflater().inflate(R.layout.contact_addresses, null);
    TextView address = (TextView) addressView.findViewById(R.id.address);
    address.setText(json.getJSONArray("addresses").getJSONObject(i).getString("address"));
    addressArray.add(i, json.getJSONArray("addresses").getJSONObject(i).getString("address"));
```

Εάν η επαφή έχει στα στοιχεία της κάποια email ο χρήστης μπορεί να στείλει ένα mail απλά πατώντας πάνω σε ένα από αυτά. Επίσης έχει την δυνατότητα να καλέσει την επαφή του, αν υπάρχει στοιχείο με αριθμό τηλεφώνου. Για να την λειτουργία της τηλεφωνικής κλίσης πρέπει να πάρουμε Permissions από την συσκευή Android, δηλώνοντας στο AndroidManifest ***<uses-permission android:name="android.permission.CALL_PHONE"/>***

4.8.5.1 *Διαγραφή επαφής*

Ο χρήστης μπορεί να διαγράψει την επαφή από την επιλογή του menu του Activity που τοποθετήται δεξιά του Toolbar. Εκεί εμφανίζεται τα OptionsMenu με popup μήνυμα «delete». Επιλέγοντας την διαγραφή της εφαρμογής, εμφανίζεται ένα Alert Dialog για την επαλήθευση της ενέργειας. Αν ο χρήστης την υλοποιήσει, τότε πραγματοποιείται η διαγραφή της επαφής με ένα AsyncTask. Το AsyncTask αυτό εμφανίζει ένα μήνυμα ενημέρωσης (Progress Dialog) της προόδου της διαγραφής, και στο παρασκήνιο εκτελείται ένα Delete Call στον server με το id της επαφής που υπόκειται διαγραφή. Στο τέλος το μήνυμα εξαφανίζεται και το Activity τερματίζεται, επιστρέφοντας στο Activity της κύριας οθόνης όπου αναδημιουργούνται τα Tabs από τα νέα Get Calls στον server και η επαφή δεν υπάρχει πια.

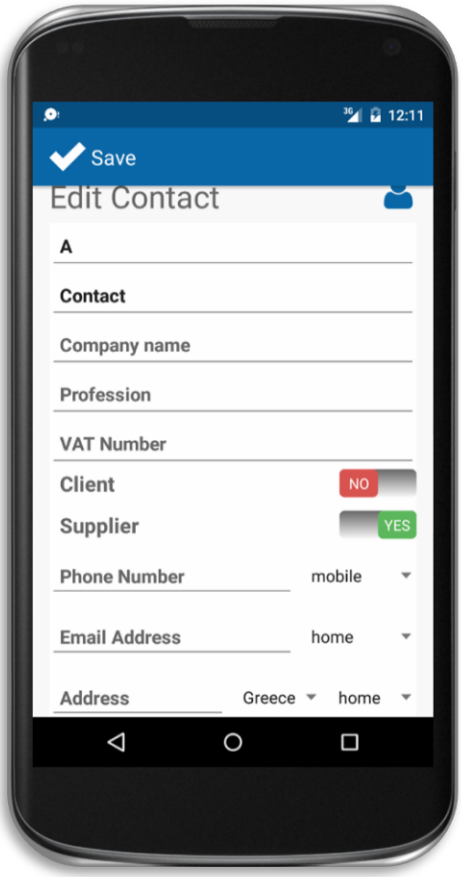
```
            new AlertDialog.Builder(this, R.style.MyAlertDialogStyle)//styling the dialog
                    .setTitle("Delete Contact")
                    .setMessage("Are you sure you want to delete this contact?")
                    .setPositiveButton("Yes", (dialog, which) -> {
                            try {
                                deleteContact();
                                dialog.cancel();
                            } catch (MalformedURLException e) {
                                e.printStackTrace();
                            }
                    })
                    .setNegativeButton("No", (dialog, which) -> {
                            dialog.cancel();
                    })
                    .setIcon(android.R.drawable.ic_menu_delete)
                    .show();
            return true;
        }

        return super.onOptionsItemSelected(item);
    }

    public void deleteContact() throws MalformedURLException {
        URL restUrl = new URL("http://api.elorus-staging.com/v1.0/contacts/" + getId() + "/");
        new DeleteTask().execute(restUrl);
        Log.d("ID", "from DELETE : " + getId());

    }

    public class DeleteTask extends AsyncTask<URL, Void, String> {
        ProgressDialog progressDialog;

        @Override
        protected void onPreExecute() {
            super.onPreExecute();
            progressDialog = ProgressDialog.show(ViewContact.this, "Deleting Contact", "Please Wait...", true);
        }

        @Override
        protected void onPostExecute(String s) {
            super.onPostExecute(s);
            progressDialog.dismiss();
        }

        @Override
        protected String doInBackground(URL... urls) {
            RestCalls deleteCall = new RestCalls(null);
            deleteCall.DeleteCall(urls[0]);
            finish();
```

Εικόνα 82: Διαγραφή επαφής

4.8.5.2 *Επεξεργασία επαφής*

Εάν ο χρήστης επιλέξει το εικονίδιο για την επεξεργασία των στοιχείων της επαφής, δημιουργείται ένα νέο Activity και το ViewContact Activity τερματίζεται.

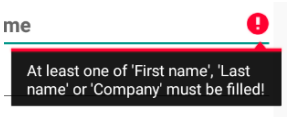
```
editButton = (TextView) findViewById(R.id.editButton);
editButton.setTypeface(Typeface.createFromAsset(editButton.getContext().getAssets(), "fonts/elorus-font.ttf"));
editButton.setTextSize(TypedValue.COMPLEX_UNIT_SP, 35);
editButton.setTextColor(editButton.getResources().getColor(R.color.myWhite));
editButton.setOnClickListener((v) -> {
        startActivity(editContact);
        finish();
});
```



Εικόνα 83: Παράδειγμα φόρμας επεξεργασίας επαφής

Το νέο Activity αυτό (EditContact), εμφανίζει μια φόρμα από πεδία, από τα οποία, έχουν συμπληρωθεί με στοιχεία της επαφής που υπήρχαν στα δεδομένα, τα υπόλοιπα είναι κενά.

Εμφανίζονται διάφορα πεδία από TextViews και Spinners. Τα πεδία αυτά είναι το όνομα, το επίθετο, το όνομα της εταιρίας, το επάγγελμα, το ΑΦΜ, εάν είναι πελάτης ή προμηθευτής, ο αριθμός τηλεφώνου και το είδος του, η διεύθυνση email και το είδος της, και η διεύθυνση κατοικίας με την χώρα και το είδος της. Όλα αυτά τα στοιχεία είναι στοιχεία του API του server και ο χρήστης μπορεί να συμπληρώσει ή να αλλάξει όποια θέλει. Ο μόνος περιορισμός είναι να βάλει αναγκαστικά ένα όνομα, επίθετο ή όνομα εταιρίας εάν το αφήσει κενό. Όπως επίσης αν αλλάξει κάποιο πεδίο με τις πολλαπλές επιλογές (τηλέφωνο, email, διεύθυνση) και το πεδίο είναι κενό εμφανίζεται μήνυμα ειδοποίησης ή αν η διεύθυνση email δεν έχει σωστή μορφή. Για την σωστή λειτουργία και για να είναι user friendly έχουν τοποθετηθεί Validations με τα κατάλληλα μηνύματα.




```
public void Validation(EditText firstName, EditText lastName, EditText companyName) {
    if (Build.VERSION.SDK_INT > Build.VERSION_CODES.GINGERBREAD_MR1) {
        if (firstName.getText().toString().length() == 0 && lastName.getText().toString().length() == 0 && companyName.getText().toString().length() == 0) {
            firstName.setError("At least one of 'First name', 'Last name' or 'Company' must be filled!");
            lastName.setError("At least one of 'First name', 'Last name' or 'Company' must be filled!");
            companyName.setError("At least one of 'First name', 'Last name' or 'Company' must be filled!");
        }

    } else
    {
        if (firstName.getText().toString().length() == 0 && lastName.getText().toString().length() == 0 && companyName.getText().toString().length() == 0) {
            firstName.setError(Html.fromHtml("<font color='red'>At least one of 'First name', 'Last name' or 'Company' must be filled!</font>"));
            lastName.setError(Html.fromHtml("<font color='red'>At least one of 'First name', 'Last name' or 'Company' must be filled!</font>"));
            companyName.setError(Html.fromHtml("<font color='red'>At least one of 'First name', 'Last name' or 'Company' must be filled!</font>"));
        }
    }
}

public void ValidationEmail(EditText emailadd) {
    if (Build.VERSION.SDK_INT > Build.VERSION_CODES.GINGERBREAD_MR1) {
        if (emailadd.getText().toString().length() == 0)
            emailadd.setError("Email Address may be required!");
        else if (!emailadd.getText().toString().matches(EXP))
            emailadd.setError("Not a valid email address!");
    } else
    {
        if (emailadd.getText().toString().length() == 0)
            emailadd.setError(Html.fromHtml("<font color='red'>Email Address may be required!</font>"));
        else if (!emailadd.getText().toString().matches(EXP))
            emailadd.setError(Html.fromHtml("<font color='red'>Not a valid email address!</font>"));
```

Τα πεδία με το τηλέφωνο, την διεύθυνση email και την διεύθυνση κατοικίας πρέπει να μην είναι μοναδικά για την περίπτωση οπού η επαφή περιέχει παραπάνω από ένα ή ο χρήστης επιθυμεί να προσθέσει κάποιο. Η λύση για αυτή την λειτουργία απαιτεί την χρήση μικρών fragments. Το κάθε πεδίο από αυτά είναι ένα fragment όπου συμπεριλαμβάνεται κάτω από τα παραπάνω πεδία. Όταν υπάρχουν περισσότερες καταχωρίσεις από μια, δημιουργείται ένα νέο fragment ίδιας κατηγορίας κάτω από αυτό για την εισχώρηση νέων στοιχείων. Όπως καις την περίπτωση που δεν υπάρχει για παράδειγμα καταχώριση από τηλέφωνο, ο χρήστης βλέπει ένα τέτοιο fragment-πεδίο. Από την στιγμή που τελειώσει την συμπλήρωση από το πληκτρολόγιο, εμφανίζεται ένα νέο κάτω από το προηγούμενο.

Το ίδιο συμβαίνει και στα άλλα δύο πεδία που επιτρέπουν πολλαπλές καταχωρίσεις.

```
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.new_contact_phone_layout, container, false);
    setPhone((EditText) v.findViewById(R.id.phoneNumberEditText));
    setPhoneSpinner((Spinner) v.findViewById(R.id.phoneSpinner));


    SpinnerValidation(getPhoneSpinner(), getPhone());

    phone.setOnEditorActionListener((v, actionId, event) -> {

            if (count == 0) {
                if (!v.getText().toString().equals("")) {
                    ValidationPhone(getPhone());

                    FragmentManager fragmentManager = getFragmentManager();
                    FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
                    NewContactPhoneFragment phoneFragment = new NewContactPhoneFragment();
                    phoneFragment.setI(i + 1);
                    int newI = getI();
                    fragmentTransaction.add(R.id.phoneNewLayout, phoneFragment, "" + newI);
                    fragmentTransaction.commit();
                    count++;

                }
            }
            return false;
    });

    return v;
```

Εικόνα 84: Παράδειγμα fragment τηλεφώνου

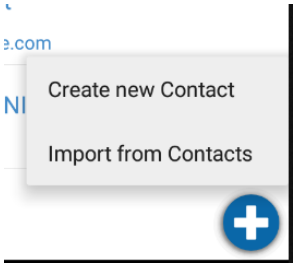
Μόλις ο χρήστης τελειώσει με την επεξεργασία της επαφής, μπορεί να πατήσει να αποθηκεύσει τα νέα στοιχεία της επαφής πατώντας το εικονίδιο αριστερά του custom Toolbar που δημιουργήσαμε για αυτό το Activity. Μόλις γίνει αυτό εκτελείται ένα PUT Call στον server για την ενημέρωση της επαφής. Αυτό γίνεται με την συγκέντρωση των στοιχείων των πεδίων σε ένα JSONObject που στέλνουμε στον server.

Για την εκτέλεση της διεργασίας αυτής χρησιμοποιείται ένα AsyncTask. Το AsyncTask εμφανίζει ένα progress dialog στον χρήστη για την πρόοδο της διεργασίας και στο παρασκήνιο, γίνεται η εκτέλεση του call. Τέλος το progress Dialog εξαφανίζεται, το Activity κλείνει και δημιουργείται ένα νέο Activity για την προβολή της επαφής με τα νέα στοιχεία. Έτσι ο χρήστης μπορεί να έχει σωστή εμφάνιση στην σειρά των οθονών. Η ίδια εναλλαγή σε Activities γίνεται εάν ο χρήστης πατήσει το back button για να επιστρέψει στην πίσω οθόνη.

```
public class ResponseTask extends AsyncTask<String, Void, String> {
    ProgressDialog progressDialog;

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        progressDialog = ProgressDialog.show(EditContact.this, "Updating contact", "Please Wait...", true);
    }

    @Override
    protected void onPostExecute(String s) {
        super.onPostExecute(s);
        progressDialog.dismiss();
    }

    @Override
    protected String doInBackground(String... params) {
        String accountInfo = params[0];
        RestCalls PutCall = new RestCalls("contacts/" + id + "/");
        PutCall.setJsonString(accountInfo);
        PutCall.PutCall();
        try {
            if (PutCall.getPut().getResponseMessage().equals("OK")) {
                PutCall.getPut().disconnect();
                Intent viewContact = new Intent(EditContact.this, ViewContact.class).putExtra("id", id);
                startActivity(viewContact);
                finish();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
}

@Override
public void onBackPressed() {
    Intent viewContact = new Intent(EditContact.this, ViewContact.class).putExtra("id", id);
    startActivity(viewContact);
    finish(); //close intent
}
```

4.8.6 Λειτουργία προσθήκης νέας επαφής

Όταν ο χρήστης επιλέξει το εικονίδιο κάτω από την λίστα των επαφών εμφανίζονται δύο popup επιλογές. Να δημιουργήσει μια καινούρια ή να προσθέσει μια από τις υπάρχουσες στην συσκευή Android.



Εάν επιλέξει να δημιουργήσει μια καινούρια, τότε δημιουργείται ένα νέο Activity με την ίδια φόρμα με την επεξεργασία της επαφής, αφού τα στοιχεία είναι τα ίδια. Οι μόνες διαφορές είναι ότι είναι κενή και ότι η μέθοδος REST για την αποστολή δεδομένων στον server είναι η POST αφού θέλουμε να δημιουργήσουμε κάτι καινούριο.

Εάν επιλέξει να προσθέσει μια από τον κατάλογο με τις επαφές android τότε εμφανίζεται ένα Activity με τις επαφές της συσκευής και όταν ο χρήστης επιλέξει κάποια, στέλνονται το όνομα, και αν υπάρχουν τα τηλέφωνα, τα email, οι διευθύνσεις και ο οργανισμός.

Για την προσθήκη των ήδη υπάρχων επαφών δεν χρειάζεται η δημιουργία java κλάσης (Activity) αφού τα δεδομένα για την απεικόνιση υπάρχουν έτοιμα σε Activity του Android. Το έτοιμο Activity αυτό θα καλεστεί με την

```
addContact = new Intent(Intent.ACTION_PICK, ContactsContract.Contacts.CONTENT_URI);
startActivityForResult(addContact, PICK_CONTACT);
```

Που με την σειρά της καλεί την μέθοδο onActivityResult.

```
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    String company = "";
    String firstName = "";
    String lastName = "";
    switch (requestCode) {
        case (PICK_CONTACT):
            if (resultCode == Activity.RESULT_OK) {
                Uri contactData = data.getData();
                Cursor c = getActivity().getContentResolver().query(contactData, null, null, null, null);
                if (c.moveToFirst()) {
                    String contactId = c.getString(c.getColumnIndexOrThrow(ContactsContract.Contacts._ID));

                    String[] nameWhereParams = new String[]{contactId,
                            ContactsContract.CommonDataKinds.StructuredName.CONTENT_ITEM_TYPE};
                    Cursor nameCursor = getActivity().getContentResolver().query(ContactsContract.Data.CONTENT_URI, null,
                            ContactsContract.Data.CONTACT_ID + " = ? AND " + ContactsContract.Data.MIMETYPE + " = ?", nameWhereParams, null);
                    if (nameCursor.moveToNext()) {
                        firstName = nameCursor.getString(nameCursor.getColumnIndex(ContactsContract.CommonDataKinds.StructuredName.GIVEN_NAME));
                        lastName = nameCursor.getString(nameCursor.getColumnIndex(ContactsContract.CommonDataKinds.StructuredName.FAMILY_NAME));
                    }
                   // Log.d("FIRST ", "name is: " + firstName);
                   // Log.d("LAST ", "name is: " + lastName);
                    nameCursor.close();

                    Cursor phonesCursor = getActivity().getContentResolver().query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null,
                            ContactsContract.CommonDataKinds.Phone.CONTACT_ID + " = " + contactId, null, null);
                    while (phonesCursor.moveToNext()) {
                        String number = phonesCursor.getString(phonesCursor.getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMBER));
                     //   Log.d("NUMBER ", "number is: " + number);

                        int type = phonesCursor.getInt(phonesCursor.getColumnIndex(ContactsContract.CommonDataKinds.Phone.TYPE));
                        switch (type) {
                            case ContactsContract.CommonDataKinds.Phone.TYPE_HOME:
                              //  Log.d("NUMBER ", "type is home");
                                JsonPhoneData(number, "home");
                                break;
                            case ContactsContract.CommonDataKinds.Phone.TYPE_MOBILE:
                               // Log.d("NUMBER ", "type is mobile");
                                JsonPhoneData(number, "mobile");
                                break;
                            case ContactsContract.CommonDataKinds.Phone.TYPE_WORK:
                              //  Log.d("NUMBER ", "type is work");
                                JsonPhoneData(number, "work");
                                break;

                            case ContactsContract.CommonDataKinds.Phone.TYPE_FAX_HOME:
                              //  Log.d("NUMBER ", "type is fax home");
                                JsonPhoneData(number, "fax");
```

Μέσα σε αυτή την μέθοδο δημιουργούμε έναν δείκτη cursor που μας επιτρέπει να δείχνει σε ότι θέλουμε. Για την συγκεκριμένη περίπτωση, μας δείχνει στο id του contact που θα επιλέξουμε από τα ContactsContract.Contacts._ID . Δηλώνουμε τις παραμέτρους που θέλουμε να εμφανιστούν, δηλαδή τα ContactsContract.Data.CONTENT_URI και έπειτα με την βοήθεια εσωτερικών Cursor επιλέγονται τα δεδομένα της επιλεγμένης επαφής που χρειάζονται για την ενσωμάτωση.

Τέλος αποθηκεύουμε τα δεδομένα σε ένα JSONObject που το στέλνουμε με την μέθοδο POST στον server. Η εκτέλεση της διεργασίας γίνεται με ένα AsyncTask και ο χρήστης ενημερώνεται για την πρόοδο με ένα progress Dialog.

4.9 Notifications

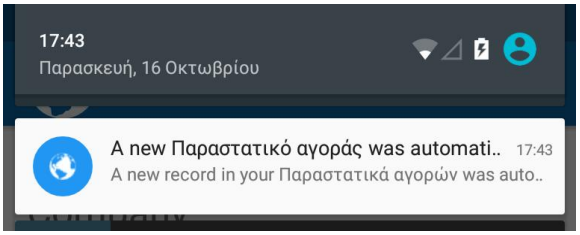
Μια απαραίτητη λειτουργία της εφαρμογής για την ενημέρωση του χρήστη σημαντικών γεγονότων. Για την δημιουργία των Notifications χρησιμοποιήσαμε έναν Broadcast Receiver ο οποίος εκτελείται όταν ο χρήστης είναι παρών στην συσκευή του και όταν την ανοίξει. Αυτό γίνεται με την απόκτηση permissions από την Android συσκευή και δηλώνονται στο Android Manifest.

```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
```

Γίνεται ένας έλεγχος για το αν υπάρχει δίκτυο με τα κατάλληλα permissions, και αν ο χρήστης χρησιμοποιεί την συσκευή.

```
@Override
public void onReceive(Context context, Intent intent) {
    Log.d("CHECK", "checking if BOOT is completed");
    if ("android.intent.action.BOOT_COMPLETED".equals(intent.getAction()) ||
            "android.intent.action.SCREEN_ON".equals(intent.getAction()) ||
            "android.intent.action.SCREEN_OFF".equals(intent.getAction()) ||
            "android.intent.action.USER_PRESENT".equals(intent.getAction())) {
        Log.d("CHECK", "checking if network is up");
        if (isNetworkAvailable(context)) {
            try {
                StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Bui
                StrictMode.setThreadPolicy(policy);
                Notification(context);
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    }
```

Επειδή το notification που θα λαμβάνει η εφαρμογή είναι της μορφής JSON μέσω ενός GET call, πρέπει να το υλοποιήσουμε κατάλληλα. Γίνεται ένα GET Call το οποίο αποθηκεύεται σε ένα JSONObject από αυτό παίρνουμε τα πεδία title και message για να τα δείξουμε στο notification. Έπειτα κατασκευάζουμε ένα Notification με την χρήση του NotificationCompat.Builder. Το μορφοποιούμε και το δείχνουμε στον χρήστη. Τέλος όταν ο χρήστης το διαβάσει, μεταβαίνετε στην εφαρμογή αλλά και γίνεται ένα PUT Call να ειδοποιήσουμε τον server ότι διαβάστηκε.



```
public void Notification(Context context) throws JSONException {
    RestCalls notificationsCall = new RestCalls("notifications");
    notificationsCall.GetCall();
    String notification = notificationsCall.getJsonString();
    if (notification != null) {
        JSONObject json = new JSONObject(notification);
        JSONArray results = new JSONArray(json.getString("results"));
        int color = context.getResources().getColor(R.color.notificationColor);
        for (int i = 0; i < json.getInt("count"); i++) {
            if (results.getJSONObject(i).getString("read").equals("null")) {
                String title = results.getJSONObject(i).getString("title");
                String message = results.getJSONObject(i).getString("message");

                // Open NotificationView Class on Notification Click
                Intent intent = new Intent(context, Dashboard.class).putExtra("notificationId", results.getJSONObject(i).getString("id"));
                // Open NotificationView.java Activity
                PendingIntent pIntent = PendingIntent.getActivity(context, i, intent, PendingIntent.FLAG_UPDATE_CURRENT);
                // Create Notification using NotificationCompat.Builder
                NotificationCompat.Builder builder = new NotificationCompat.Builder(context)
                        // Set Icon
                        .setSmallIcon(getNotificationIcon())
                        .setColor(color)
                                // Set Ticker Message
                        .setTicker(message)
                                // Set Title
                        .setContentTitle(title)
                                // Set Text
                        .setContentText(message)
                                // Set PendingIntent into Notification
                        .setContentIntent(pIntent)
                                // Dismiss Notification
                        .setAutoCancel(false)
                        .setDefaults(Notification.DEFAULT_ALL);

                // Create Notification Manager
                NotificationManager notificationmanager = (NotificationManager) context.getSystemService(Context.NOTIFICATION_SERVICE);
                // Build Notification with Notification Manager
                notificationmanager.notify(i, builder.build());

                if (existAlarm(i, context) == null) {
                    RestCalls getCall = new RestCalls(results.getJSONObject(i).getString("id"));
                    getCall.PutNotificaton();
```

```
// Check for network availability
private boolean isNetworkAvailable(Context context) {
    ConnectivityManager connectivityManager = (ConnectivityManager) context
            .getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo activeNetworkInfo = connectivityManager
            .getActiveNetworkInfo();
    return activeNetworkInfo != null;
}

private int getNotificationIcon() {
    boolean whiteIcon = (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.LOLLIPOP);
    return whiteIcon ? R.drawable.globe_silhouette_md : R.drawable.ic_launcher;
}

public PendingIntent existAlarm(int id,Context context) {
    Intent intent = new Intent(context, Dashboard.class);
    intent.setAction(Intent.ACTION_VIEW);
    return PendingIntent.getBroadcast(context, id , intent, PendingIntent.FLAG_NO_CREATE);
}
```

Κεφάλαιο 5

5.1 Μελλοντική Εξέλιξη

Η εφαρμογή βρίσκεται στο επίπεδο του monitoring με κάποια βασικά features. Μελλοντικός σκοπός είναι στην δημιουργία παραστατικών για την πλήρη διαχείριση των επιχειρήσεων σε λογιστικό επίπεδο. Την δημιουργία πολλαπλών χρηστών για να έχει ο καθένας το δικό του περιβάλλον διαχείρισης. Τέλος την εμφάνιση και λειτουργία συστήματος πληρωμής μέσω Paypal των υπηρεσιών που προσφέρει η εφαρμογή.

5.2 Συμπεράσματα

Η ανάπτυξη της εφαρμογής Android είναι μια ιδιαίτερα δύσκολη διαδικασία, ειδικά όταν θα πρέπει να μάθεις πολλά πράγματα μόνος/μόνη εφόσον δεν συμπεριλαμβάνονται σε μάθημα της σχολής. Θα πρέπει να σου κεντρίσει τον ενδιαφέρον για να ασχοληθείς και θέλει τεράστια προσπάθεια και συμβουλές από φόρουμ με εμπειρία στον χώρο. Υπήρχαν αρκετές δυσκολίες όσο αναφορά την φόρτωση δεδομένων από JSON αντικείμενα αλλά και την διαχείριση των επαφών της Android συσκευής. Παρόλα αυτά όμως στο τέλος βγήκε ένα πολύ καλό αποτέλεσμα.

Μέσα στην εφαρμογή υπάρχουν πολλές χρήσιμες λειτουργίες, την δημιουργία και επεξεργασία επαφών και αντικειμένων γενικά, επικοινωνία με server για την απόκτηση, αλλαγή και δημιουργία δεδομένων, την δημιουργία και απεικόνιση notifications, τις μεθόδους για την καλύτερη δυνατή εμπειρία του χρήστη, με την επιτάχυνση εμφάνισης UI και επικοινωνίας με τον server, την δημιουργία και λειτουργία φόρμας εισόδου και μεθόδου υπενθύμισης των στοιχείων εισαγωγής.

Κεφάλαιο 6

Βιβλιογραφία

1. J. Friesen, 2010, “Learn Java for Android Development”, Apress
2. M. Murphy, 2011, "Android Programming Tutorials, 3rd Edition”, CommonsWare
3. C. Hasenan, 2008, “Android Essentials”, Firstpress
4. R. Meier, 2010, “Professional Android 2 Application Development”, Wrox
5. Android Training Page
6. Design Guidelines Page
7. Stack Overflow
8. Android Hive
9. GitHub
10. Android for Devs
11. Icon-Fonts in Android Page