Εξώφυλλο Αναφοράς Πτυχιακής Εργασίας

Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Σχολή Τεχνολογικών Εφαρμογών

Τμήμα Μηχανικών Πληροφορικής

Πτυχιακή Εργασία

Τίτλος: Εφαρμογή αποστολής/λήψης άμεσων μηνυμάτων σε περιβάλλον Android

---

Νικολόπουλος Αλέξανδρος (3011)

Ράλλης Ιωάννης (2752)

Επιβλέπων καθηγητής : ***Παπαδάκης Νικόλαος***

Επιτροπή Αξιολόγησης : ***Κορνάρος Γιώργος, Βιδάκης Νικόλαος***

Ημερομηνία παρουσίασης : ***28/8/2015***

Ευχαριστίες

Θα θέλαμε να ευχαριστήσουμε τον επιβλέποντα καθηγητή Δρ. Νικόλαο Παπαδάκη για την ανάθεση αυτής της πτυχιακής σε εμάς. Εν συνεχεία, θα θέλαμε να ευχαριστήσουμε τις οικογένειές μας για την κατανόηση και υποστήριξη που μας έδειξαν κατά την περίοδο της δημιουργίας της εφαρμογής αυτής αλλά και για την συγγραφή αυτής της πτυχιακής εργασίας.

Abstract

Aim of this thesis is the development of an application which will be compatible with the most recent versions of Android operating system. This application has been very carefully developed in the simplicity of User Interface, so that anyone could use this application without having specific knowledge.

This application helps the user with his everyday activities. One such action is sending instant text messages or photographs sharing daily moments with one's contacts or the insertion of meetings between the contacts one has added to one's list of friends. Another extra feature that this application offers is that the user can choose the profile picture that will be shown in his contacts of friends.

For the needs of this application several devices with the most updated android software were used so that we could fix as many errors as possible in this application. Our idea was picked up by a real application that is released on the android system and we have worked on very few features in the application, since the aim of this thesis is to show how an application like this works.

Σύνοψη

Ο στόχος αυτής της πτυχιακής εργασίας είναι η ανάπτυξη μίας εφαρμογής η οποία θα είναι συμβατή με τις πιο πρόσφατες εκδόσεις του λειτουργικού συστήματος Android. Η εφαρμογή αυτή έχει αναπτυχθεί με ιδιαίτερη προσοχή στην απλότητα της Διεπαφής Χρήστη (User Interface), έτσι ώστε οποιοσδήποτε να μπορεί να χρησιμοποιήσει την εφαρμογή αυτή χωρίς να έχει ιδιαίτερες γνώσεις.

Η εφαρμογή αυτή βοηθάει τον χρήστη στις καθημερινές του δραστηριότητες Μια τέτοια δραστηριότητα είναι η αποστολή άμεσων μηνυμάτων κειμένου η φωτογραφιών μοιράζοντας τις στιγμές της καθημερινότητας με τις επαφές του ή η εισαγωγή ραντεβού μεταξύ των επαφών που έχει προσθέσεις στην λίστα φίλων του. Κάποια έξτρα χαρακτηριστικά που προσφέρει η εφαρμογή είναι ο χρήστης να μπορεί να επιλέξει την φωτογραφία προφίλ που θα φαίνεται στις επαφές των φίλων του.

Για τις ανάγκες της εφαρμογής αυτής χρησιμοποιήθηκαν αρκετές συσκευές με τα ποιο σύγχρονα λογισμικά android για να μπορέσουμε να διορθώσουμε όσο το δυνατόν περισσότερα σφάλματα στην εφαρμογή. Η ιδέα μας πάρθηκε από πραγματική εφαρμογή που κυκλοφορεί στα συστήματα android και έχουμε δουλέψει πάνω σε ελάχιστα χαρακτηριστικά στην εφαρμογή, αφού, ο σκοπός της πτυχιακής εργασίας είναι να δείξουμε πως μία τέτοια εφαρμογή δουλεύει.

Πίνακας περιεχομένων

Ευρετήριο Εικόνων

Ευρετήριο Πινάκων

1. Εισαγωγή

Η συγκεκριμένη πτυχιακή εργασία αφορά την ανάπτυξη και υλοποίηση μιας εφαρμογής για λειτουργικό Android η οποία θα επιτρέπει στους χρήστες την εύκολη και γρήγορη ανταλλαγή μηνυμάτων στο πλαίσιο της καθημερινή τους επικοινωνίας και την δημιουργία ραντεβού για τις προγραμματισμένες συναντήσεις τους.

1.1. Περίληψη

Ο σκοπός της εφαρμογής είναι να διευκολύνει τους χρήστες στην καθημερινή τους επικοινωνία μέσα από ένα εύχρηστο, διασκεδαστικό και ειδικά προσαρμοσμένο γραφικό περιβάλλον.

Η εφαρμογή αναπτύχθηκε για το λειτουργικό Android και αποτελεί μια εφαρμογή πελάτη που επικοινωνεί με τον Server για την ανταλλαγή μηνυμάτων, την αποδοχή και αποστολή αιτημάτων φιλίας και την δημιουργία και αποδοχή ραντεβού. Ο χρήστης έχει την δυνατότητα να κάνει εγγραφή στο σύστημα και στην συνέχεια να συνδεθεί για να εισαχθεί και να δει τα περιεχόμενα της ή να εκτελέσει μία από τις διαθέσιμες ενέργειες.

Από την πλευρά του Server έγινε χρήση της διαδεδομένης scripting γλώσσας PHP και βάσης δεδομένων MySQL. Όλοι οι χρήστες και τα δεδομένα τους όπως ραντεβού μηνύματα κτλ. είναι αποθηκευμένα σε αντίστοιχους πίνακες της βάσης δεδομένων.

Ο χρήστης έχει την δυνατότητα εκτός από μήνυμα απλού κειμένου να αποστείλει και φωτογραφία, να διαγράψει και να κάνει ολοκληρωμένη προβολή της εικόνας που έστειλε ή έλαβε, να δημιουργήσει ραντεβού και να αποδεχτεί ή να ακυρώσει εισερχόμενα ραντεβού, να κάνει αναζήτηση φίλων και να αποδεχτεί ή να αποστείλει αιτήματα φιλίας.

1.2. Σκοπός και Στόχος της πτυχιακής εργασίας

Οι στόχοι της πτυχιακής εργασίας είναι τέσσερις και συνοψίζονται παρακάτω:

1. Ο χρήστης μπορεί να δημιουργήσει ένα λογαριασμό μέσα από ανάλογη φόρμα.
2. Ο χρήστης μπορεί να στείλει αίτημα φιλίας σε χρήστες ή να αποδεχτεί εισερχόμενα αιτήματα φιλίας.
3. Ο χρήστης μπορεί να δημιουργήσει και να αποστείλει μηνύματα σε φίλους του με κείμενο ή και εικόνα.
4. Ο χρήστης μπορεί να δημιουργήσει ραντεβού με φίλους του και να αποδεχτεί ή να απορρίψει εισερχόμενα ραντεβού.

Συνοψίζοντας όλα τα παραπάνω σκοπός της πτυχιακής εργασίας είναι η δημιουργία μιας εύχρηστης εφαρμογής που θα επιτρέπει την καθημερινή αλληλεπίδραση φίλων μέσω μηνυμάτων και δημιουργίας ραντεβού με ένα κατανοητό διαδραστικό τρόπο συνδυασμένη με την κατάλληλη γραφική διεπαφή. Η εφαρμογή πρέπει να χρησιμοποιεί λειτουργικό Android και πρότυπα της Google.

1.3. Δομή της εργασίας

Το **Κεφάλαιο 2** ασχολείται με την εισαγωγική περιγραφή του λειτουργικού Android, την βασική αρχιτεκτονική του και τα βασικά συστατικά μιας Android εφαρμογής. Το **Κεφάλαιο 3** ασχολείται με τις λειτουργικές απαιτήσεις τις εφαρμογής, ενώ το **Κεφάλαιο 4** παρουσιάζει τα εργαλεία ανάπτυξης της εφαρμογής. Στο **Κεφάλαιο 5** παρουσιάζονται αναλυτικά οι διαδικασίες ανάπτυξης με επεξήγηση σε βασικά κομμάτια του κώδικα και στο **Κεφάλαιο 6** παρουσιάζονται τα αποτελέσματα από την διαδικασία ανάπτυξης.

2. Το λειτουργικό Android OS

Το android αποτελεί λειτουργικό σύστημα βασισμένο σε πυρήνα Linux το οποίο αναπτύχθηκε από την Google και είναι ειδικά σχεδιασμένο για οθόνες αφής κινητών τηλεφώνων και tablet. Ανάλογα με τις δραστηριότητες, το περιβάλλον και τις ανάγκες των χρηστών έχουν αναπτυχθεί εξειδικευμένες εκδόσεις android με γραφικές διεπαφές ειδικά προσαρμοσμένες σε περιβάλλον τηλεόρασης Android TV, αυτοκινήτου Android Auto και ρολόγια καρπού Android Wear. Επιπλέον το android ενσωματώνεται εύκολα και χρησιμοποιείται από συσκευές καθημερινής χρήσης όπως ψηφιακές κάμερες, κονσόλες βιντεοπαιχνιδιών ακόμα και σε σταθερού υπολογιστές.

Η Google έχει εκδώσει (release) τον κώδικα του λειτουργικού Android με άδειες ανοικτού κώδικα(open source licenses) το οποίο το καθιστά ως ένα ιδιαίτερα δημοφιλές λειτουργικό σε συσκευές υψηλής τεχνολογίας ενώ ενθαρρύνει τους προγραμματιστές και τμήματα ανάπτυξης να διανέμουν εκδόσεις με ακόμα ευρύτερη λειτουργικότητα βρίσκοντας εφαρμογή σε έργα του δημόσιου και του ιδιωτικού φορέα.

2.1. Αρχιτεκτονική - Συστατικά Μέρη Android OS

Το Android αποτελείται από μια διαστρωμάτωση επιμέρους λογισμικών συστατικών. Κάθε στρώμα αποτελείται από ένα σύνολο λογισμικών ενοτήτων το οποίο προσφέρει στο αμέσως επόμενο ένα σύνολο από υπηρεσίες.

Linux Kernel: Όλο το λειτουργικό android έχει δομηθεί πάνω σε προσαρμοσμένο πυρήνα Linux το οποίο αποτελείται από τους απαραίτητους οδηγούς(drivers) για την αλληλεπίδραση με το υλικό της συσκευής.
Android Runtime: Σε αυτό το επίπεδο αποτελείται από τον Dalvik Virtual Machine και βιβλιοθήκες Java (JAVA SE και JAVA ME).

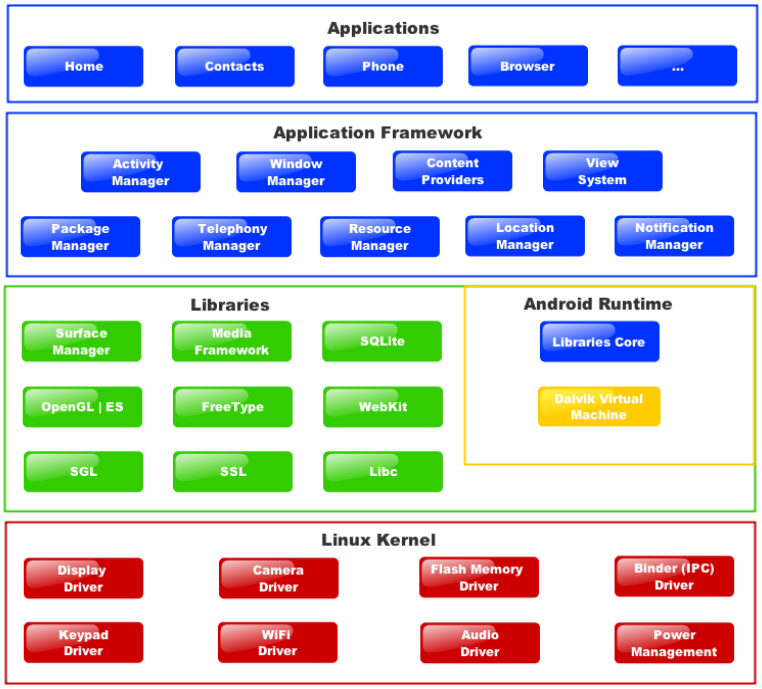
Libraries: Σε αυτό το επίπεδο περιέχονται βιβλιοθήκες γραμμένες σε γλώσσα C/C++ οι οποίες έχουν αναπτυχθεί για να διαχειρίζονται διαφορετικού τύπου δεδομένα. Μερικές από αυτές τις βιβλιοθήκες είναι οι:

1. **Media Framework:** Παρέχει μία σειρά από κωδικοποιητές Media επιτρέποντας την αναπαραγωγή ή την εγγραφή διάφορων τύπων αρχείων media.
2. **SQLite:** Είναι η βάση δεδομένων και μαζί με όλες τις επιμέρους λειτουργικότητες για την αποθήκευση δεδομένων στο Android.
3. **WebKit:** Είναι η βιβλιοθήκη που χρησιμοποιείται για την εμφάνιση HTML κώδικα σε browsers.
4. **OpenGL:** Χρησιμοποιείται για την δημιουργία 2D και 3D γραφικών στην οθόνη.
5. **Dalvik Virtual Machine:** Αποτελεί έναν τύπο JVM για android συσκευές για να εκτελούνται εφαρμογές καταναλώνοντας όσο τον δυνατόν λιγότερους πόρους της συσκευής.
6. **Java (JAVA SE και JAVA ME):** Παρέχει την λειτουργικότητα που συναντάμε σε JAVA SE βιβλιοθήκες.

Application Framework: Αποτελείται από ενότητες λογισμικού και προγραμματιστικά εργαλεία που επιτρέπουν στην εφαρμογές μας να επικοινωνούν με πόρους του συστήματος και να προσφέρουν λειτουργικότητα στις εφαρμογές που αναπτύσσονται για android. Μερικά παραδείγματα είναι:

1. **Content Providers:** Διαχειρίζονται τα δεδομένα που διαμοιράζονται μεταξύ τους οι εφαρμογές.
2. **Telephony Manager:** Διαχειρίζεται τις κλάσεις ήχου και την πρόσβαση σε υπηρεσίες βιντεοκλήσης.
3. **Location Manager:** Ανάκτηση της τοποθεσία από τους GPS αισθητήρες ή από δίκτυο κυψελών.
4. **Resource Manager:** Διαχειρίζεται διάφορους τύπους από πόρους όπως φωτογραφίες σε png, jpeg κτλ. ή xml layouts που χρησιμοποιούν οι εφαρμογές.

Applications: Είναι το επίπεδο το οποίο βρίσκεται στην κορυφή τη διαστρωμάτωσης και αποτελείται από τις εφαρμογές που είναι εγκατεστημένες στο κινητό μας. Λόγω της ιδιότητα του android ως ελεύθερου λογισμικού μπορούμε να αντικαταστήσουμε πολλές από τις προεγκατεστημένες εφαρμογές του κινητούς μας όπως SMS Client app, Web Browser, ContactManger κτλ.



Εικόνα 1: Android Architecture Diagram

2.2. Συστατικά Μέρη Android Εφαρμογής

Activity: Αυτή η κλάση αντιστοιχεί στην οθόνη που εμφανίζεται στον χρήστη και φορτώνει όλους τους απαραίτητους πόρους όπως xml layout, drawables κτλ. για την εμφάνιση της γραφικής διεπαφής με την οποία θα αλληλεπιδρά. Με την υλοποίηση των κατάλληλων διεπαφών μπορούμε να διαχειριζόμαστε γεγονότα (events) όταν αυτά εμφανίζονται και σχετίζονται με τον τρόπο που αλληλεπιδρά ο χρήστης με την οθόνη, τέτοιες διεπαφές είναι για παράδειγμα η onClickeListener, onTouchelistener κτλ.

Κάθε οθόνη που δημιουργούμε πρέπει να επεκτείνει (extend) την κλάση Activity και διαθέτει ένα συγκεκριμένο κύκλο ζωής ο οποίος υλοποιείται μέσα από τις αντίστοιχες callback μεθόδους: onCreate(),onStart(),onPause(),onStop(),onDestroy().

Πριν η οθόνη καλέσει τη onDestroy() δηλαδή στην onPause() θα πρέπει να έχουμε φροντίσει να αποθηκεύσουμε τα δεδομένα τα οποία θέλουμε να παραμείνουν στην εφαρμογή

μας, αυτό μπορεί να γίνει είτε με την κλάση SharedPreferences η οποία αποθηκεύει μικρού όγκου δεδομένα κυρίως για τις ρυθμίσεις (setting) της εφαρμογής, είτε σε βάση δεδομένων sqlite με χρήση της κλάσης SqliteHelper όπου μας επιτρέπει την αποθήκευση δεδομένων ως εγγραφές σε ανάλογους πίνακες, είτε σε αρχεία xml ή txt που αποθηκεύονται στο σύστημα αρχείων της κινητής συσκευής.

Fragment: Ένα fragment αποτελεί ένα τμήμα από την γραφική διεπαφή ενός Activity και μπορεί να επαναχρησιμοποιηθεί σε παραπάνω από ένα Activity. Έχει τον δικό του κύκλο ζωής το οποίο επηρεάζεται άμεσα από το Activity στο οποίο φιλοξενείται.

Ο κύκλος ζωής ενός fragment υλοποιείται μέσα από τις αντίστοιχες callback μεθόδους όπως onCreate(), onCreateView(), onActivityCreated(), onStart(), onStart(), onPause(), onStop(), onDestroyView(), onDestroy(), onDetached().

Service: Ένα Service αποτελεί το συστατικό εκείνο της εφαρμογής το οποίο μπορεί να εκτελεί λειτουργίες που διαρκούν αρκετή ώρα στο παρασκήνιο όπως μεταφόρτωση/αναφόρτωση δεδομένων στο διαδίκτυο κτλ. ενώ δεν παρέχει κάποια γραφική διεπαφή. Κάθε συστατικό της εφαρμογής μπορεί να καλέσει την bindservice() και να δημιουργηθεί έτσι μια επικοινωνία τύπου client-server όπου ο client μπορεί να είναι ένα Activity ενώ server ένα service και η επικοινωνία μεταξύ τους να επιτυγχάνεται με την υλοποίηση κατάλληλων διεπαφών.

Ένα service ξεκινάει με την κλήση startService() και σταματάει με την stopService(). Κάθε service έχει το δικό του lifecycle το οποίο υλοποιείται από τις αντίστοιχες callback μεθόδους: onStartCommand(), onCreate(), onDestroy(), onPause(), onStop(), onCreate(), onBind(), onUnbind().

Sqlite Helper: Αποτελεί την βοηθητική κλάση η οποία διαχειρίζεται την βάση δεδομένων και τις εκδόσεις της για αναβάθμιση όποτε αυτό χρειάζεται. Περιέχει όλες τις απαραίτητες μεθόδους για την εισαγωγή δεδομένων στους πίνακες και την ανάκτηση των εγγραφών τους με την εκτέλεση κατάλληλων ερωτημάτων.

BroadCastReceiver: Ένας BroadcastReceiver είναι το συστατικό εκείνο της εφαρμογή που είναι προσανατολισμένο στο να λαμβάνει συγκεκριμένα γεγονότα όταν αυτά παρουσιάζονται. Ένα παράδειγμα είναι όταν το σύστημα μας ολοκληρώσει την επανεκκίνηση του, μπορούμε να γράψουμε ένα BroadcastReceiver και να τον αρχικοποιήσουμε έτσι ώστε να ακούει για το event ACTION_BOOT_COMPLETED.

Υπάρχουν δύο τρόποι για να δημιουργήσουμε έναν BroadcastReceiver. Ο ένας είναι να τον δηλώσουμε στο AndroidManifest φάκελο και με το tag <intent-filter> να προσδιορίσουμε για τα actions που είναι προγραμματισμένος να λαμβάνει όπως προαναφέραμε π.χ. <action android:name="android.intent.action.BOOT_COMPLETED" /> όπου αποτελεί ένα action που το στέλνει αυτόματα το Runtime σύστημα του android όταν ολοκληρωθεί η εκκίνηση του συστήματος.

Ένας δεύτερος τρόπος είναι να κάνουμε την έγγραφή προγραμματιστικά με την μέθοδο registerReceiver() με δύο παραμέτρους, η πρώτη είναι ένα instance της κλάσης που έχουμε δημιουργήσει να επεκτείνει την BroadCastReceiver και να υλοποιεί την onReceive() και η δεύτερη είναι ένα instance IntentFilter όπου του έχουμε θέσει όταν τα actions τα οποία πρέπει να ακούει και μπορούν να προέρχονται και από άλλες οντότητες της εφαρμογής όπως π.χ. ένα Service.

3. Περιγραφή Λειτουργικών Απαιτήσεων

3.1.1. Εγγραφή στο σύστημα

Η εφαρμογή επιτρέπει σε μη εγγεγραμμένους χρήστες να εγγραφούν στο σύστημα και να αποκτήσουν όνομα χρήστη και κωδικό πρόσβασης. Σε περίπτωση που το όνομα χρήστη και ο κωδικός υπάρχουν είδη η εφαρμογή ενημερώνει τον χρήστη ανάλογα.

- Ο χρήστης οδηγείται στην αντίστοιχη φόρμα εγγραφής και εισάγει τα πεδία username, email και password της επιλογής του. Με το πάτημα του κουμπιού 'Register' τα δεδομένα αποστέλλονται στον server όπου γίνεται ο έλεγχος για την διαθεσιμότητα των εισαχθέντων στοιχείων.

- Σε περίπτωση επιτυχημένη εγγραφής η εφαρμογή εγγράφει αυτόματα τον χρήστη στον αντίστοιχο πίνακα σε αντίθετη περίπτωση τον ενημερώνει για την αποτυχία δημιουργία λογαριασμού χρήστη.

Πίνακας 1: Εγγραφή στο σύστημα

Inputs	Processing	Outputs
Ο χρήστης εισάγει μέσα από την φόρμα εγγραφής τα στοιχεία Email(String), Password(String) και πιέζει το κουμπί Register.	Η εφαρμογή από την πλευρά του server παραλαμβάνει τα δεδομένα και κάνει έλεγχο αν υπάρχει καταχωρημένος χρήστης με τα στοιχεία που αποστάλθηκαν, αν δεν υπάρχει δημιουργεί εγγραφή στον πίνακα της βάσης δεδομένων με τα στοιχεία του χρήστη.	Σε περίπτωση επιτυχημένης εγγραφής ο χρήστης ενημερώνεται για την επιτυχία της με αποστολή μηνύματος Response(String). Σε αντίθετη περίπτωση ενημερώνεται ανάλογα για την αποτυχία δημιουργίας λογαριασμού.

3.1.2. Σύνδεση στο σύστημα

Μια από τις κύριες λειτουργίες της εφαρμογής είναι η σύνδεση του χρήστη. Ο εγγεγραμμένος χρήστης εισάγει τα στοιχεία του email και password στην αντίστοιχη φόρμα και πιέζει το κουμπί Login:

- Γίνεται τοπικά ο έλεγχος για την εγκυρότητα των στοιχείων και στην συνέχει αποστέλλονται στον server.

- Η εφαρμογή από την πλευρά του server ελέγχει αν υπάρχει καταχωρημένος χρήστης με τα στοιχεία που έχουν εισαχθεί και αποστέλλει ανάλογο μήνυμα για επιτυχημένη ή μη επιτυχημένη σύνδεση.

Πίνακας 2: Σύνδεση στο σύστημα

Inputs	Processing	Outputs
Ο χρήστης εισάγει μέσα από την φόρμα σύνδεσης τα στοιχεία Email(String),Password(String)	Η εφαρμογή από την πλευρά του server κάνει έλεγχο αν υπάρχει καταχωρημένος χρήστης με τα στοιχεία που αποστάλθηκαν	Σε περίπτωση επιτυχημένης εγγραφής ο χρήστης ενημερώνεται για την επιτυχία της εγγραφής του με αποστολή μηνύματος Response(String)

3.1.3. Αποστολή μηνυμάτων

Ο χρήστης έχει την δυνατότητα να αποστείλει σε φίλο του στο σύστημα μήνυμα με κείμενο ή εικόνα:

- Ο χρήστης πληκτρολογεί κείμενο στην αντίστοιχη φόρμα και επιλέγει από το photo album μία εικόνα της αρεσκείας του.

- Με την επιλογή του κουμπιού Send από το menu ο χρήστης αποστέλλει τα δεδομένα του μηνύματος στον server όπου καταχωρούνται στην βάση δεδομένων.

Πίνακας 3: Αποστολή μηνυμάτων

Inputs	Processing	Outputs
Ο χρήστης εισάγει μέσα από την φόρμα αποστολή μηνυμάτων τα στοιχεία Message(String), Photo(base64String), ReceiverId(Integer), ProviderId(Integer)	Η εφαρμογή από την πλευρά του server λαμβάνει τα δεδομένα και τα αποθηκεύει στον αντίστοιχο πίνακα των μηνυμάτων.	Σε περίπτωση επιτυχημένης ή αποτυχημένης αποστολής μηνυμάτων ο χρήστης ενημερώνεται με αντίστοιχο μήνυμα Response(String)

3.1.4. Προβολή ιστορικού μηνυμάτων

Ο χρήστης έχει την δυνατότητα να δει το ιστορικό μηνυμάτων του με έναν χρήστη.

- Επιλέγει από την διαθέσιμη λίστα συνομιλιών μια συνομιλία με ένα φίλο.

- Η εφαρμογή κάνει ανάκτηση όλων των μηνυμάτων του φίλου με τον χρήστη και τα προβάλει.

Πίνακας 4: Προβολή ιστορικού μηνυμάτων

Inputs	Processing	Outputs
Ο χρήστη επιλέγει μία συνομιλία και στέλνει το id του φίλου στην οθόνη μηνυμάτων	Η εφαρμογή του πελάτη ανακτά από την βάση όλα τα μηνύματα που έχουν σταλεί από και προς τον συγκεκριμένο φίλο.	Εμφανίζεται η λίστα με όλο το ιστορικό μηνυμάτων.

3.1.5. Αποστολή αιτήματος φιλίας

Ο χρήστης αποδέχεται εισερχόμενο αίτημα φιλίας:

- χρήστης πατώντας το κουμπί αποδοχή με την ένδειξη ‘Accept’ αποστέλλει στον server δεδομένα για την αποδοχή του αιτήματος φιλίας και ο server ανανεώνει την εγγραφή του αιτήματος φιλίας με το αντίστοιχο status.

Πίνακας 5: Αποστολή αιτήματος φιλίας

Inputs	**Processing**	**Outputs**
ProviderId(Integer), ReceiverId(Integer), Status(String)	Η εφαρμογή θέτει το status του αιτήματος φιλίας σε APPROVED και το ανανεώνει την εγγραφή	Response(String) όπου ενημερώνεται ο χρήστης για την επιτυχημένη αποδοχή του αιτήματος φιλίας.

3.1.6. Αποστολή ραντεβού

Ο χρήστης μπορεί να δημιουργήσει ένα ραντεβού και να το αποστείλει στον φίλο του:

- χρήστης συμπληρώνοντας την αντίστοιχη φόρμα με το ραντεβού στην συνέχεια επιλέγει το κουμπί Send από το menu και αποστέλλει στον server τα δεδομένα του ραντεβού που έχει δημιουργήσει. Η εφαρμογή στον server αναλαμβάνει να δημιουργήσει εγγραφή στην βάση για το αντίστοιχο ραντεβού.

Πίνακας 6: Αποστολή ραντεβού

Inputs	Processing	Outputs
ProviderId(Integer), ReceiverId(Integer), Location(String), Cause(String), FromDateTime(Date), ToDateTime(Date)	Η εφαρμογή παραλαμβάνει τα δεδομένα και τα καταχωρεί τον αντίστοιχο πίνακα με το status PENDING.	Response(String) όπου ενημερώνεται ο χρήστης για την επιτυχημένη αποστολή των ραντεβού.

3.1.7. Αποδοχή ή απόρριψη ραντεβού

Ο χρήστης μπορεί να αποδεχτεί ή να απορρίψει ένα εισερχόμενο ραντεβού:

- Σε περίπτωση αποδοχής αποστέλλονται στον server τα δεδομένα του ραντεβού ο οποίος ενημερώνει την εγγραφή του συγκεκριμένου ραντεβού ως αποδεκτή.

- Σε περίπτωση μη αποδοχής αποστέλλονται στον server τα δεδομένα του ραντεβού ο οποίος ενημερώνει την εγγραφή ως μη αποδεκτή από τον φίλο.

Πίνακας 7: Αποδοχή ή απόρριψη ραντεβού

Inputs	Processing	Outputs
ProviderId(Integer), ReceiverId(Integer), Status(String)/Values (APPROVED,DECLINE)	Η εφαρμογή παραλαμβάνει τα δεδομένα και ανανεώνει το status του καταχωρημένου ραντεβού.	Response(String) όπου ενημερώνεται ο χρήστης για την επιτυχημένη αποστολή των ραντεβού.

3.1.8. Προβολή καρτέλας φίλου

Ο χρήστης επιλέγοντας έναν φίλο από τη λίστα φίλων οδηγείται στην καρτέλα του όπου προβάλλονται τα στοιχεία του όπως το email και η εικόνα του προφίλ του. Από εκεί μπορεί να προσκαλέσει ένα φίλο σε ραντεβού πιέζοντας το κουμπί Invite Friend ή να ανοίξει μία συνομιλία πιέζοντας το κουμπί Send Message.

Πίνακας 8: Προβολή καρτέλας φίλου

Inputs	Processing	Outputs
Το userid(Integer) του φίλου που επιλέχθηκε από την λίστα.	Η εφαρμογή android ανοίγει την φόρμα με τα στοιχεία του φίλου τα οποία φορτώνει από τα δεδομένα που έχει αποθηκεύσει στην βάση τοπικά.	Η καρτέλα του φίλου με τα στοιχεία του όπως το email, το username του και την εικόνα του προφίλ του.

3.1.9. Τροποποίηση στοιχείων εγγεγραμμένου χρήστη

Ο χρήστη επιλέγοντας να οδηγηθεί στην καρτέλα του προφίλ του μπορεί να κάνει επεξεργασία συγκεκριμένα στοιχεία όπως το email του και την εικόνα του προφίλ του.

- Επιλέγοντας τη επιλογή Edit μπορεί να επιλέξει μία φωτογραφία της αρεσκείας του από το photo album και να αλλάξει το email του λογαριασμού του. Στη συνέχεια επιλέγοντας Ok αποστέλλει τα δεδομένα στον server.

Πίνακας 9: Τροποποίηση στοιχείων εγγεγραμμένου χρήστη

Inputs	Processing	Outputs
Το userid(Integer) του χρήστη, imagerprofile(base64String), email (String).	Η εφαρμογή στον server παραλαμβάνει τα δεδομένα και κάνει update τα αντίστοιχα πεδία.	Μήνυμα που ενημερώνει τον χρήστη για την επιτυχημένη ανανέωση στοιχείων.

4. Τεχνολογίες/Εργαλεία ανάπτυξης εφαρμογής

4.1. Android client εφαρμογή

Για την ανάπτυξη της εφαρμογή android χρησιμοποιήσαμε το περιβάλλον προγραμματισμού Android Studio, το Android SDK Tools το οποίο περιέχει όλα τα απαραίτητα εργαλεία για ανάπτυξη και αποσφαλμάτωση μιας Android εφαρμογής και την γλώσσα προγραμματισμού Java. Για την ανάπτυξη της εφαρμογής χρησιμοποιήθηκε το Android SDK version 15 το οποίο υποστηρίζεται από την έκδοση του Android λειτουργικού 4.0.3 ενώ η εφαρμογή μας μπορεί να εκτελεστεί ομαλά ακόμα και σε Android λογισμικό έκδοσης 5.0.1.

Το Android Studio παρέχει έναν γρήγορο και εξελιγμένο editor βασισμένο στην τεχνολογία IntelJ IDEA. Παρέχει επίσης ένα γραφικό περιβάλλον ειδικό για την σύνθεση γραφικών διεπαφών για Android εφαρμογές το οποίο μας επιτρέπει με ένα απλό drag n drop να δημιουργήσουμε το γραφικό περιβάλλον κάθε οθόνης. Το Android Studio έχει επίσης ενσωματωμένο Android SDK Manager το οποίο μας επιτρέπει να κατεβάσουμε APIs, usb drivers και χρήσιμες βιβλιοθήκες για την ανάπτυξη Android εφαρμογών. Ιδιαίτερα χρήσιμο είναι επίσης και το Android Device Manager όπου εκεί μπορούμε να δημιουργήσουμε προσομοιωτές και να κάνουμε αποσφαλμάτωση σε εικονικές συσκευές Android με διαφορετικές εκδόσεις Android λογισμικού.

4.2. Server side εφαρμογή

Για την ανάπτυξη της εφαρμογής από την πλευρά του server χρησιμοποιήσαμε όπως έχουμε προαναφέρει την γλώσσα PHP και την βάση δεδομένων MySql. Τα μηνύματα που αποστέλλουμε από τον server είναι μορφοποιημένα σε json format ενώ τα δεδομένα που αποστέλλονται από την εφαρμογή αποθηκεύονται στον κορμό(body) της HTTP κλήσης ως ζεύγη .

Η PHP αποτελεί την πιο διαδεδομένη scripting γλώσσα για εφαρμογές που εκτελούν σε servers. Ένα από τα προτερήματα της php είναι ότι διαθέτει μια αρκετά μεγάλη κοινότητα και ένα ισχυρό documentation όπου ωθεί συνεχώς νέους προγραμματιστές να ενασχοληθούν με την εν λόγω scripting γλώσσα. Οι περισσότεροι web servers διαθέτουν το απαραίτητο PHP Module Interface SAPI το οποίο δίνει την δυνατότητα διερμηνείας PHP scripts. Η έκδοση php που χρησιμοποιούμε είναι η 5.0.1 η οποία υποστηρίζει τον προγραμματισμό προσανατολισμένο σε αντικείμενα ενώ διαθέτει και τα απαραίτητα MySql extendsions για την επικοινωνία με την βάση δεδομένων.

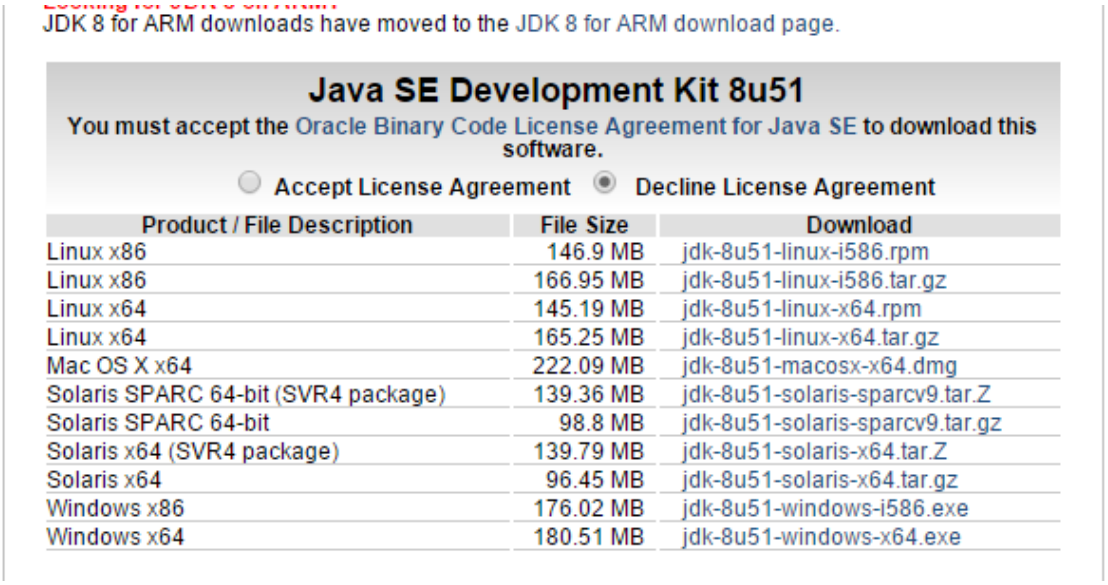
Για την ανάπτυξη των php script χρησιμοποιήσαμε τον Notepadd ++ editor.

Για την αποθήκευση των δεδομένων χρησιμοποιήσαμε την τεχνολογία MySql όπου αποτελεί ένα σύστημα σχεσιακή βάσης δεδομένων(RDBMS) που υποστηρίζει την γλώσσα ερωτημάτων SQL. Επίσης χρησιμοποιήσαμε το εργαλείο phpmyadmin το οποίο μας επιτρέπει την διαχείριση της βάσης δεδομένων μέσα από έναν ευχάριστο γραφικό περιβάλλον.

5. Υλοποίηση εφαρμογής

5.1. Υλοποίηση android εφαρμογής

Για την υλοποίηση της android εφαρμογής είναι απαραίτητο να εγκαταστήσουμε την java στον υπολογιστή μας. Μεταβαίνουμε στη σελίδα http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html και επιλέγουμε την έκδοση jsdk που αντιστοιχεί στον υπολογιστή μας:



JDK 8 for ARM downloads have moved to the JDK 8 for ARM download page.

Java SE Development Kit 8u51

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

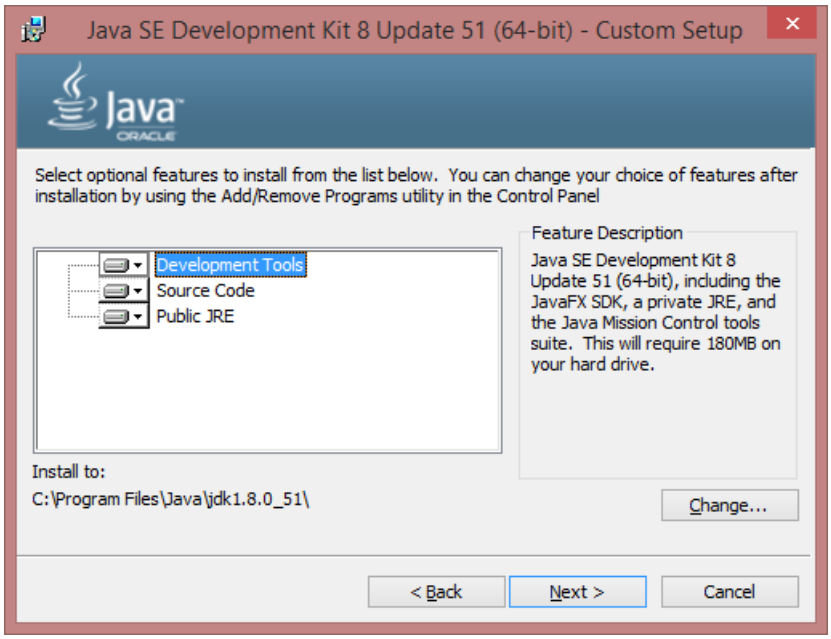
○ Accept License Agreement ◉ Decline License Agreement

Product / File Description	File Size	Download
Linux x86	146.9 MB	jdk-8u51-linux-i586.rpm
Linux x86	166.95 MB	jdk-8u51-linux-i586.tar.gz
Linux x64	145.19 MB	jdk-8u51-linux-x64.rpm
Linux x64	165.25 MB	jdk-8u51-linux-x64.tar.gz
Mac OS X x64	222.09 MB	jdk-8u51-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	139.36 MB	jdk-8u51-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	98.8 MB	jdk-8u51-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	139.79 MB	jdk-8u51-solaris-x64.tar.Z
Solaris x64	96.45 MB	jdk-8u51-solaris-x64.tar.gz
Windows x86	176.02 MB	jdk-8u51-windows-i586.exe
Windows x64	180.51 MB	jdk-8u51-windows-x64.exe

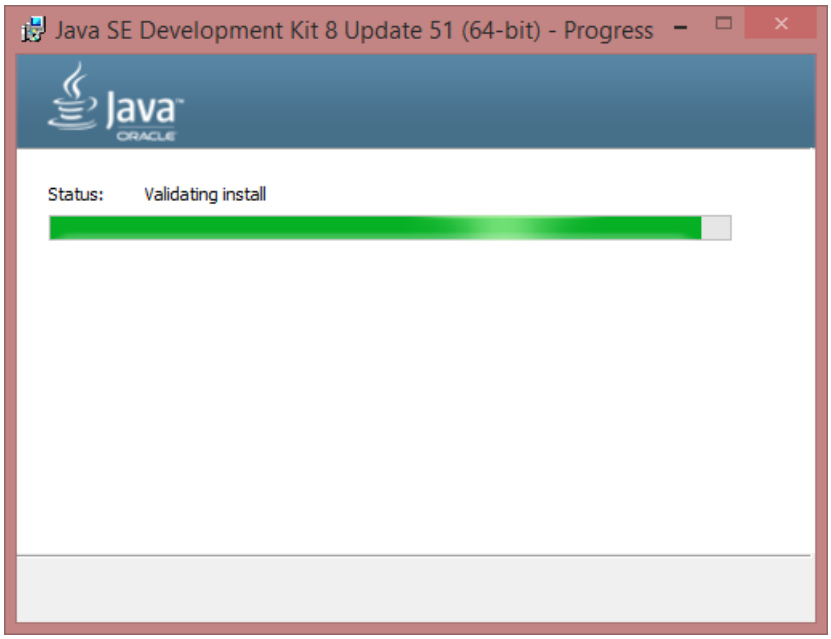
Εικόνα 2: Download JDK

Εφόσον εγκαταστήσουμε το αντίστοιχο αρχείο .exe το εκτελούμε στον υπολογιστή μας και ακολουθούμε τα βήματα εγκατάστασης.

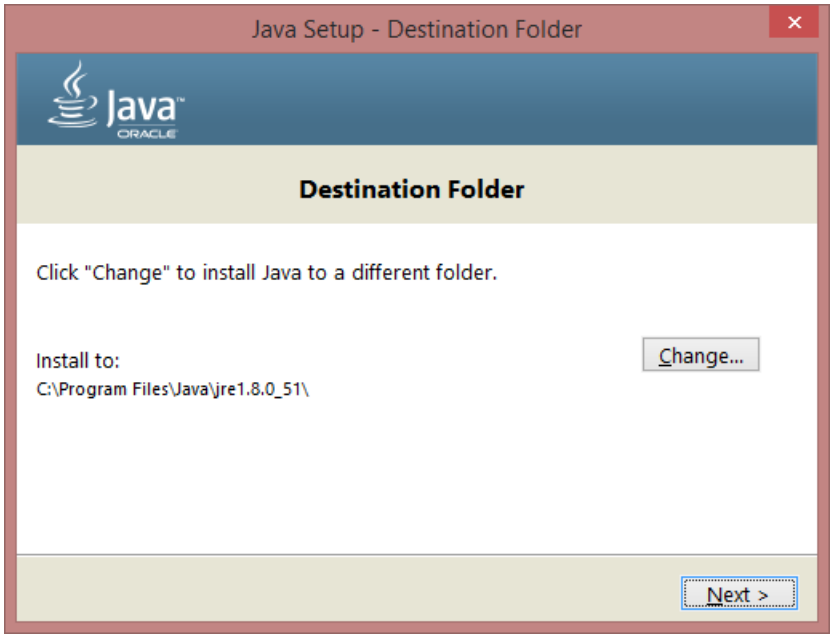
Εικόνα 3: Εγκατάσταση JDK



Εικόνα 4: Εγκατάσταση JDK



Εικόνα 5: Εγκατάσταση JDK

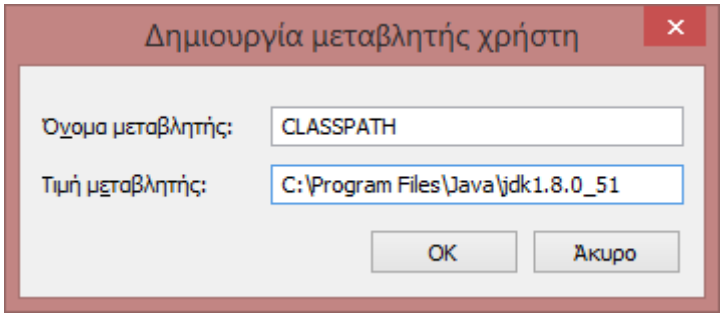


Εικόνα 6: Εγκατάσταση JDK

Εικόνα 7: Εγκατάσταση JDK

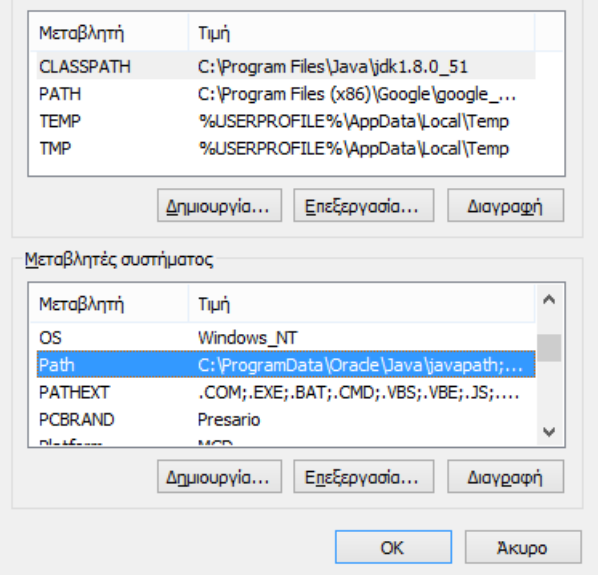
Εικόνα 8: Εγκατάσταση JDK

Ορίζουμε την μεταβλητή συστήματος **CLASSPATH** με τιμή το **path** που είναι εγκατεστημένο το **jdk** που εγκαταστήσαμε.



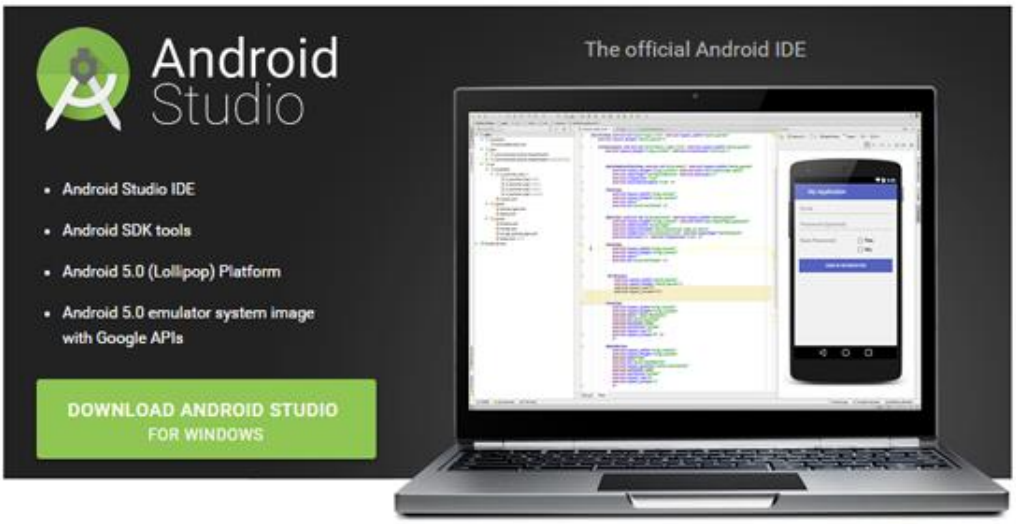
Εικόνα 9: Μεταβλητή Συστήματος

Στην συνέχεια θέτουμε στην μεταβλητή **path** το **path** που είναι εγκατεστημένη **java** στον υπολογιστή μας.



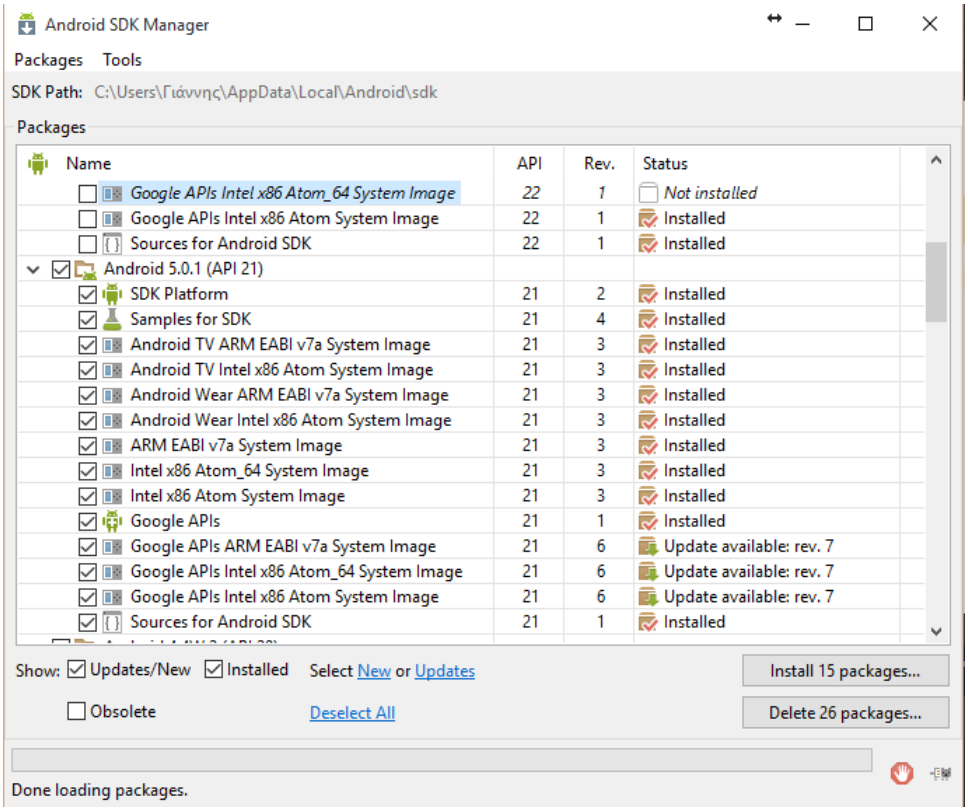
Εικόνα 10: Μεταβλητή Συστήματος

Για την ανάπτυξη android εφαρμογής κατεβάζουμε το Android Studio από την σελίδα https://developer.android.com/sdk/index.html



Εικόνα 11: Download Android Studio

Στη συνέχεια εκτελούμε το .exe του android studio και κατεβάζουμε από το android sdk manager τις εκδόσεις που θέλουμε για την ανάπτυξη της εφαρμογής.



Εικόνα 12: Android SDK Manager

5.2. Υλοποίηση android client εφαρμογής

Το βασικό σύστημα αρχείων μίας Android εφαρμογής αποτελείται από τα .java αρχεία που βρίσκονται στο directory java, τα γραφικά στοιχεία που βρίσκονται μέσα στο αρχείο res, το AndroidManifest.xml στο οποί ορίζουμε τα Services, Activities και όλες τις απαραίτητες άδειες για την χρήση αντίστοιχων πόρων κινητού.

Παρακάτω παρουσιάζουμε και κάνουμε μία σύντομη ανάλυση στα πακέτα που περιέχουν τα αρχεία .java που αποτελούν το πιο λειτουργικό κομμάτι της εφαρμογής μας:

Interfaces: Εδώ βρίσκονται όλα τα interfaces που υλοποιεί η εφαρμογή μας όπως FriendListener, AppointmentListener, MessageListener και UserListener. Η χρήση των interfaces είναι απαραίτητη για την ενημέρωση της γραφικής διεπαφής κάθε φορά που λαμβάνουμε απόκριση από μία HTTP κλήση.

Όταν αποστέλλουμε ένα αίτημα φιλίας με την sendFriendRequest, τότε αυτή εκτελείται σε έαν Thread(background thread) διαφορετικό από το Main UI Thread στο οποίο έχει αρχικοποιηθεί η γραφική διεπαφή. Για να ενημερώσουμε στην συνέχεια την γραφική διεπαφή αποστέλλουμε ένα μήνυμα στον αντίστοιχο Handler που είναι συνδεδεμένος με το UI Thread και του ορίζουμε να καλέσει την αντίστοιχη μέθοδο του interface που υλοποιεί το Activity δηλαδή την onSendFriendRequest() της διεπαφής FriendListener.

Services: Εδώ βρίσκονται τα Services που εκτελούν λειτουργίες παρασκηνίου όπως την μεταφόρτωση ή την αποστολή μηνυμάτων, αιτημάτων φιλίας και ραντεβού, FriendService, AppointmentService, MessageService. Ένα Service εκτελείται ακόμα και όταν ο χρήστης

μεταβεί σε άλλη οθόνη ή ανοίξει άλλη εφαρμογή. Ο λόγος που χρησιμοποιήσαμε Services είναι για να μπορούμε να κατεβάζουμε από τον Server περιοδικά τα δεδομένα μας ανεξάρτητα από τι ενέργειες εκτελεί ο χρήστης και να τα παρέχουμε με την μορφή λίστας όταν αυτός το απαιτήσει.

Content: Εδώ έχουμε αποθηκεύσει όλες τις κλάσεις που χρησιμοποιούμε για να αποθηκεύσουμε δεδομένα ή να εξάγουμε εγγραφές από την τοπική βάση όπως η IMContentManager. Η συγκεκριμένη κλάση επεκτείνει την SQLiteOpenHelper και είναι υπέυθυνη για την δημιουργία ή την αναβάθμιση της βάσης δεδομένων και την εκτέλεσει ερωτημάτων στους πίνακες της βάσης δεδομένων.

Controllers: Έχουμε αποθηκεύσει όλες τις κλάσεις που είναι απαραίτητες για προσφέρουν την απαιτούμενη λειτουργικότητα σε κάθε fragment που έχουμε δημιουργήσει, η κλάση FriendController παρέχει τις μεθόδους για την φόρτωση και προβολή λίστας φίλων, παρέχει εγγραφές φίλων και μεθόδους για την ασύγχρονη αποστολή αιτημάτων φιλία ή αποδοχή φιλίας. Η κλάση AppointmentCotnroller αντίστοιχα παρέχει μεθόδους για την προβολή λίστας από ραντεβού, την ασύγχρονη αποστολή http κλήσεων για αποστολή των ραντεβού ή επιβεβαίωση ή ακύρωση των ραντεβού στον server. UserController περιέχει μεθόδους για την μεταφόρτωση χρηστών στο σύστημα. MessageControler παρέχει μεθόδους για ασύγχρονες HTTP κλήσεις υπεύθυνες για την αποστολή και την ανάκτηση του ιστορικού μηνυμάτων από και προς τον server.

Views: Εδώ βρίσκονται όλες οι κλάσεις που σχετίζονται με την προβολή στον χρήστη της αντίστοιχης γραφικής διεπαφή. Οι συγκεκριμένες κλάσεις αποτελούν Fragment ή Activities και προβάλλουν τα δεδομένα μας σε λίστες ή και φόρμες κανοητές για τον χρήστη. Μερικές από αυτές είναι AppointmentForm, AppointmnetListFragment και MessageList.

Handlers: Εδώ έχουμε αποθηκεύσει όλες τι κλάσεις που επεκτείνουν την κλάσης Handler που μας παρέχει το Android framework. Οι Handlers λαμβάνουν μηνύματα και εκτελούν αντίστοιχες ενέργειες ανάλογα με το μήνυμα που έχουν παραλάβει. Η επεξεργασία τον μηνυμάτων γίνεται στην onHandleMessage(Message) και η αποστολή των μηνυμάτων γίνεται με την εκτέλεση της μεθόδου sendMessage(Message). Η συνήθης ενέργεια που εκτελούν οι Handlers είναι να καλέσουν μεθόδους των interfaces που υλοποιούν τα Activities για να ενημερωθεί με ασφάλει η γραφική διεπαφή της εφαρμογή εφόσον τα αντικείμενα των Handlers τα έχουμε αρχικοποιήσεις στο Thread που έχει αρχικοποιηθεί και η γαφική διεπαφή.

Utils: Εδώ έχουμε αποθηκεύσει τις κλάσεις που παρέχουν βοηθητική επαναλαμβανόμενη λειτουργία σε όλη την εφαρμογή όπως η κυκλική εμφάνιση της φωτογραφίας του προφίλ με την χρήση της κλάσης CircleImageView, την δημιουργία και την εκτέλεση HTTP κλήσεων με την μέθοδο POST με την χρήση της μεθόδου makeHttpRequest της κλάσης HttpExcecutor. Επιπλέον η κλάση CommontUtilities παρέχει μία σειρά από βοηθητικές σταστικές μεθόδους όπως είναι η sendBroadcast για την αποστολή broadcast μηνυμάτων, την printDifference για τον υπολογισμό της διάρκειας δοθέντων δύο ημερομηνιών κτλ.

Adapters: Μέσα σε αυτό το πακέτο έχουμε αποθηκεύσει τις κλάσεις που επεκτείνουν την κλάση Adapter. Αυτές οι κλάσεις είναι απαραίτητες για να τροφοδοτούν τα γραφικά στοιχεία ListView με τα αντίστοιχα δεδομένα όπως ραντεβού,μηνύματα και φίλους.Τέτοιες κλάσεις είναι οι MessageAdapter,RequestListAdapter κτλ. Κάθε κλάση που λειτουργεί ώς Adapter υλοποιεί μέσα στην getView όλες τις απαραίτητες ενέργειες για να συμπληρώσει με τιμές αντίστοιχα πεδία όπως TextView ή ImageView σε κάθε γραμμη του ListView.

Timers: Μέσα σε αυτό το πακέτο έχουμε αποθηκεύσει όλες τις κλάσεις που είναι υπεύθυνες για το download ή upload δεδομένων περιοδικά. Αποτελούνται από μία inner κλάση TimerTask όπου εκεί στην μέθοδο run εκτελούν ασύγχρονες http κλήσεις με τις αντίστοιχες παραμέτρους για την παραλαβή ή την αποστολή παραμέτρων. Με την μέθοδο start() ξεκινάει η ασύγχρονη εκτέλεση των http κλήσεων.

5.3. Υλοποίηση από την πλευρά του server

Από την πλευρά του server είναι απαραίτητο να επιλέξουμε μία υπηρεσία για host έτσι ώστε να έχουμε την δυνατότητα να αποθηκεύσουμε τα αρχεία της php τα οποία είναι απαραίτητα για την επικοινωνία με την android client εφαρμογή. Πρέπει λοιπόν να κατοχυρώσουμε ένα domain name έτσι ώστε να μπορούμε να αποκτούμε πρόσβαση διαδικτυακά στο αρχείο .index της php που λειτουργεί ως διαμοιραστής dispatcher στα HTTP αιτήματα που αποστέλλει η εφαρμογή μας με τη μέθοδο POST.

Μια υπηρεσία που παρέχει free hosting και δωρεάν subdomain είναι η υπηρεσία 00webhost.com. Εφόσον κάνει κάποιος εγγραφή δίνοντας τα στοιχεία του και δηλώνοντας το subdomain που επιθυμεί αυτόματα δημιουργείται λογαριασμός στον server με control panel για να έχει πρόσβαση στο σύστημα αρχείων και στον MySql Server.

Εικόνα 13: Web Hoster

5.3.1. Δομή και αρχεία της php εφαρμογής στον server

Η εφαρμογή της php αποτελείται από τα αρχεία:

index.php: Λειτουργεί σαν διαμοιραστής (dispatcher) για τις παραμέτρου που λαμβάνει μέσα από τις http κλήσης με την μέθοδο POST. Ανάλογα με τις παραμέτρους καλεί και τις αντίστοιχε μεθόδους για την εισαγωγή εγγραφών ή την εξαγωγή εγγραφών σε μορφή json. Το αρχείο index.php ελέγχει αν έχει τεθεί η παράμετρος tag στις POST μεταβλητές με αυτό το τμήμα κώδικα if(isset($_POST['tag'])). Η παράμετρος tag ορίζει στην ουσία ποια μέθοδος θα

κληθεί από το αρχείο db_functions.php. Παρακάτω παραθέτουμε ένα πίνακα με τις τιμές που πρέπει να πάρει η tag και τις αντίστοιχες μεθόδους που εκτελούνται από το αρχείο db_functions.php:

Πίνακας 10: Κλήση μεθόδων

Tag	Other Post Parameters	Method
allusers	-	getAllUsers
message	'provider'receiver'], 'created', attachment' text'	insertMessage
login	$'email,username	getUserByEmailAndPassword
register	'username'], ['email'], ['password']	storeUser
infriendrequests	'userid']	getIncomingFriendRequests
friendslist	'userid']	getFriendsList
appointment	'provider' 'receiver 'location' fromdatetime' status todatetime' 'cause' guid	createAppointment
getappointments	Userid, guid, check_guid	getIncomingAppointments
sendfriendrequest	provider, receiver, created	sendFriendsRequest
getallmessages		
approvefriend	useridfriendid	approvefriend

db_functions.php: Σε αυτό το αρχείο έχουμε γράψει όλες τις μεθόδους που εκτελούν ερωτήματα στην βάση δεδομένων.

db_connect.php: Σε αυτό το αρχείο έχουμε δημιουργήσει την κλάση που είναι υπεύθυνη για την σύνδεση με τον MySQL server. Παρέχει τις μεθόδους connect() η οποία είναι υπεύθυνη να ανοίξει μια σύνδεση με την βάση δεδομένων και close() η οποία κλείνει την ενεργή σύνδεση με την βάση.

config.php: Σε αυτό το αρχείο δηλώνουμε σε σταθερές μεταβλητές όλες τις παραμέτρους για την σύνδεση με την βάση δεδομένων όπως DB_HOST,DB_USERNAME_DB_PASSWORD,DB_DATABASE

2.Βάση δεδομένων του εξυπηρετητή (server)

Για την αποθήκευση των δεδομένων δημιουργήθηκε η κατάλληλη βάση στον εξυπηρετητή όπου αποθηκεύονται οι χρήστες, φίλοι, τα μηνύματα και τα ραντεβού. Για την βάση δεδομένων χρησιμοποιήθηκε MySql server. Για την επικοινωνία της εφαρμογής έχει αναπτυχθεί το ανάλογο PHP script όπου αναλαμβάνει να παραλάβει τα δεδομένα που

αποστέλλονται και να εκτελέσει τις αντίστοιχες εγγραφές ή ανανεώσεις εγγραφών στους πίνακες της βάσης δεδομένων.

Παρακάτω παρουσιάζουμε τους πίνακες της βάσης δεδομένων που αποθηκεύουν δεδομένα που αποστέλλονται από την android εφαρμογή.

Πίνακας 10: Πίνακας ραντεβού (appointment)

Χαρακτηριστικό	Τύπος
id	integer
provider	integer
receiver	integer
fromDateTime	datetime
toDateTime	datetime
status	varchar(30)
cause	varchar(30

Πίνακας 11: Πίνακας μηνυμάτων (message)

Χαρακτηριστικό	Τύπος
userid	integer
text	varchar(30)
created	datetime
attachment	varchar(30)

Πίνακας 12: Πίνακας χρηστών (users)

Χαρακτηριστικό	Τύπος
id	integer
username	varchar(30)
password	varchar(30)
salt	varchar(30)
imgprofile	varchar(30)

Πίνακας 13: Πίνακας φίλων (friends)

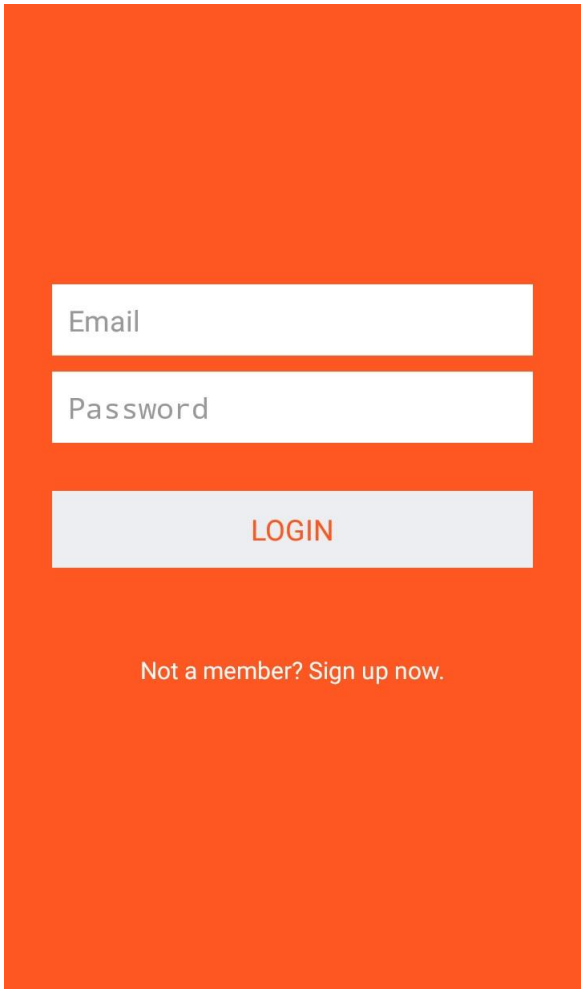
Χαρακτηριστικό	Τύπος
id	integer
username	varchar(30)
password	varchar(30)
salt	varchar(30)
imgprofile	varchar(30)

6. Αποτέλεσμα Ανάπτυξης της Εφαρμογής

Παρακάτω παρουσιάζουμε στιγμιότυπα της ανάπτυξης της Android εφαρμογής μέσα από την ακολουθία βημάτων και την δημιουργία κώδικα που προαναφέραμε. Θα παρουσιάσουμε σημαντικά κομμάτια κώδικα που χρειάζονται για την πραγματοποίηση σημαντικών δυνατοτήτων της εφαρμογής.

6.1. Εισαγωγή του χρήστη στο σύστημα

Ανοίγοντας την εφαρμογή εμφανίζεται το LoginSceen. Ο χρήστης εισάγει το email και το password και επιλέγει το κουμπί Login από την φόρμα LoginScreen. Α δεν έχει δημιουργήσει λογαριασμό μπορεί να επιλέξει την επιλογή **not a member? Sign up now**.



Εικόνα 14: LoginSceen

Αφού τοποθετήσαμε τα στοιχεία μας στην φόρμα εισόδου μας πετάει ένα message Dialog “ Autthentication Login”

```
@Override
public void onClick(View v) {
   if (v.getId() == R.id.btnLogin) {
      mDialog = new ProgressDialog(LoginScreen.this);
      mDialog.setMessage("Authentication in action....");
      mDialog.setCancelable(false);
```

Το dialog αυτό παίζει όσο η εφαρμογή ψάχνει στην βάση στο server να ταυτοποιήσει τα δεδομένα τα οποία έχουμε εισάγει.

```
String email = inputEmail.getText().toString().trim();
      String password = inputPassword.getText().toString().trim();
      if (checkFields(email, password)) {
         Crouton.makeText(LoginScreen.this, "" + resultFields, Style.ALERT).show();
         resultFields = "";
      } else {
         manager.authenicate(email, password);
```

Σε περίπτωση που τα δεδομένα έχουν γραφτεί λάθος τυπογραφικά έχουμε τοποθετήσει έναν έλεγχο ο οποίος θα πετάξει σφάλμα αν τα στοιχεία είναι λάθος.

```
@Override
public void onLoginError(String s) {
   Crouton.makeText(LoginScreen.this, s, Style.ALERT).show();

   mDialog.dismiss();
}

public void onFailedLogin() {
   mDialog.dismiss();
   Crouton.makeText(LoginScreen.this, "Incorrect username or password",
Style.ALERT).show();
}
```

Αφότου γίνει με επιτυχία το Login η εφαρμογή αυτόματα μας πετάει μέσα στο κυρίως μενού.

6.2. Κυρίως μενού και καρτέλες.

Στο κυρίως μενού της εφαρμογής υπάρχουν τρεις καρτέλες και ένα υπό-μενού. Οι καρτέλες είναι οι εξής: Καρτέλα με όλους τους φίλους, καρτέλα με τα μηνύματα και καρτέλα με τα Ραντεβού. Το υπό-μενού που βρίσκεται κάτω δεξιά είναι για το προφίλ του κάθε χρήστη. Εκεί υπάρχουν πληροφορίες για το e-mail του, αν υπάρχουν καινούργια αιτήματα φιλίας και υπάρχει η δυνατότητα αλλαγής εικόνας προφίλ.

Στο κυρίως μενού τρέχει η κλάση MainActivity.java η οποία αρχικοποιεί τα tabs της εφαρμογής.

Η κλάση **MainActivity** επεκτείνει την κλάση **SherlockFragmentActivity** κάτι το οποίο δηλώνει ότι η κλάση **MainActivity** φιλοξενεί **fragments**. Για τη αρχικοποίηση του tab menu προχωράμε στη δήλωση:

```
mActionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
```

Στην συνέχεια δημιουργούμε κάθε tab και του θέτουμε την διεπαφή που υλοποιεί η κλάση MainActivity, έτσι είμαστε σε θέση να γνωρίζουμε ποιο tab έχει επιλεχθεί κάθε φορά που αλληλεπιδρά ο χρήστης με το Tab Menu και να εκτελέσουμε τις απαραίτητε ενέργειες αν υπάρχουν.

```
ActionBar.Tab friendsTab = mActionBar.newTab();
friendsTab.setTabListener(this);
friendsTab.setCustomView(renderTabView(0));
```

Υλοποίηση της μεθόδου του interface **ActionBar.TabListener** στην κλάση **MainActivity** για να ενημερώνουμε τον ViewPager για το τρέχον Tab αυτό δηλαδή που έχει επιλέξει ο χρήστης**.**

```
@Override
public void onTabSelected(ActionBar.Tab tab, FragmentTransaction ft) {
  mPager.setCurrentItem(tab.getPosition());
  }
```

Για να εμπλουτίσουμε την εφαρμογή μας με την δυνατότητα ο χρήστης να μεταβαίνει από το ένα tab στο άλλο με σύρσιμο του δακτύλου και να παρέχουμε σε κάθε tab το αντίστοιχο fragment είναι απαραίτητο να χρησιμοποιήσουμε την κλάση **ViewPager view** το οποίο παρέχεται από την βιβλιοθήκη **android.support.v4**.

```
<android.support.v4.view.ViewPager
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/pager"
   android:layout_width="match_parent"
   android:layout_height="match_parent" />
```

Στην συνέχεια αρχικοποιούμε τον κατάλληλο **ViewPagerAdapter**. Ο **ViewPagerAdapter** τροφοδοτεί την **ViewPager** με το αντίστοιχο **fragment** μέσα από την μέθοδο **getItem()**. Ανάλογα με την θέση που βρίσκεται επιστρέφει και το αντίστοιχο **fragment instance**:

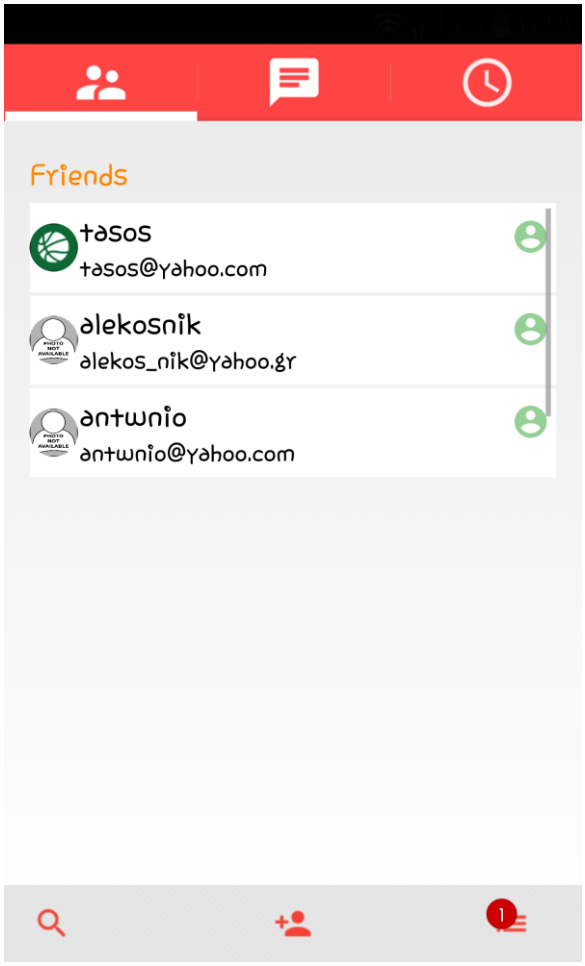
```
@Override
public Fragment getItem(int position) {
      Bundle data = new Bundle();
      switch (position){
         case 0:
            FriendList friendsList =new FriendList();
            data.putInt("current_page", position);
            friendsList.setArguments(data);
            return  friendsList;

         case 1:
            ConversationList chatList =new ConversationList();
            data.putInt("current_page", position);
            chatList.setArguments(data);
            return  chatList;

         case 2:

            AppointmentList imAppointmentListFragment =new AppointmentList();
            data.putInt("current_page", position);
```

```
            return  imAppointmentListFragment;


        }
        return null;
    }
```

Μέσα στην κλάση **MainActivity** κάνουμε εγγραφή αντικείμενο τύπου **BroadCastReceiver** και είναι το αντικείμενο της κλάσης NotificationReceiver που αναλαμβάνει να παραλάβει μηνύματα από τα Services στην onReceive και να ενημερώνει αντίστοιχα το UI.

```
onCreate(){

//…code

MainActivity.this.registerReceiver(notifyReceiver, filter);

}

public class NotificationReceiver extends BroadcastReceiver {
      @Override
      public void onReceive(Context context, Intent intent) {
      if (intent.getAction().equals(CommonUtilities.NEW_MESSAGE)) {
         Toast.makeText(MainActivity.this,"MESSAGES
ARRIVED",Toast.LENGTH_LONG).show();
          if(mPager.getCurrentItem()!=2){
              int size=manager.getMessages("status='"+ Message.UNREAD +"'").size();
              showCustomCrouton(size);
              View view=  mActionBar.getTabAt(1).getCustomView();
              TextView bandge =(TextView)view.findViewById(R.id.tab_badge);
              updateTabBadge(bandge);
            }
…….
```

Κάποιες ενδεικτικές εικόνες από την εφαρμογή είναι οι παρακάτω:



Εικόνα 15: Λίστα φίλων

Εδώ μπορούμε να δούμε τους φίλους του κάθε χρήστη στην συγκεκριμένη περίπτωση είναι τρεις. Στο μενού της λίστα φίλων (στο κάτω μέρος της οθόνης) ο χρήστη μπορεί να κάνει

1) αναζήτηση φίλων
2) αποστολή αιτήματος φιλίας και τέλος
3) να δει το προφίλ του.

Στην καρτέλα Friends για να εμφανίσουμε την λίστα φίλων χρησιμοποιούμε την κλάση **FriendList** η οποία επεκτείνει την **SherlockFragment**.

Αρχικοποιούμε τον αντίστοιχο FriendController και φορτώνουμε την λίστα με τις δηλώσεις:

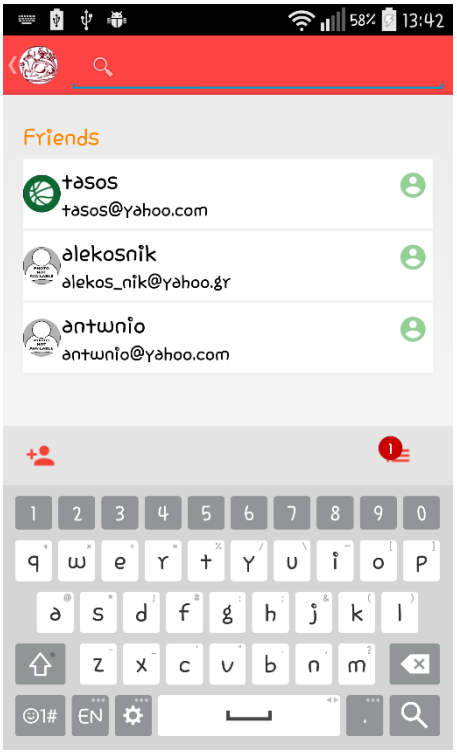
```
@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {

//…code….

FriendHandler handler = new FriendHandler(this);

controller = new FriendController(getSherlockActivity(), handler);

ls = (ListView) rootView.findViewById(R.id.listView);
controller.loasList(ls,linlaHeaderProgress);
```

Δημιουργούμε την κλάση **FriendReceiver** που επεκτείνει την **BroadCastReceiver** για να λαμβάνει μηνύματα από το Service FriendService στην onReceive και εκτελεί αντίστοιχες ενέργες.

```
public class FriendReceiver extends BroadcastReceiver {

   @Override

   public void onReceive(Context context, Intent intent)

//..code here
 if
(intent.getAction().equals(FriendHandler.FriendCodes.ON_BEFORE_GET_FRIENDS.na
me())) {
            if (controller.getFriends(null).size() == 0) {
               linlaHeaderProgress.setVisibility(View.VISIBLE);
               ls.setVisibility(View.GONE);
            } else {
               linlaHeaderProgress.setVisibility(View.GONE);
               ls.setVisibility(View.VISIBLE);
            }
         }
            }

}
```

6.3. Αναζήτηση φίλων

Ο χρήστης μπορεί να πληκτρολογήσει γράμματα που ταιριάζουν με τον φίλο που αναζητεί και να εκτελεστεί αυτόματη αναζήτηση με ανανέωση της λίστας.



Εικόνα 16: Αναζήτηση φίλων

Για την αναζήτηση των φίλων δημιουργούμε από το αντίστοιχο xml το κατάλληλο μενού. Η διαδικασία δημιουργίας του μενού εκτελείται στην **onCreateOptionsMenu** και με την εκτέλεση του κώδικα **inflater.inflate(R.menu.*menu*, menu)** παραμετρόποιούμε το μενού ανάλογα με τις δηλώσεις που έχουμε γράψει στο αντίστοιχο xml δηλαδή αυτό που βρίσκεται στον φάκελο **menu/menu.xml**

Το μενού περιέχει στην επιλογή search και το αντίστοιχο SearchView που εμφανίζεται όταν ο χρήστης προτιμήσει αυτή την επιλογή.

```
<item android:id="@+id/search"
 android:title="search"
 android:orderInCategory="1"
 android:icon="@drawable/ic_action_action_search"
 android:showAsAction="always|collapseActionView"
 android:actionViewClass="com.actionbarsherlock.widget.SearchView" />
```

Το SearchView όπως φαίνεται στην παραπάνω εικόνα αποτελείται από ένα EditText πεδίο και ένα κουμπί αναζήτησης με το αντίστοιχο εικονίδιο. Στο SearchView έχουμε κάνε τις εξής δηλώσεις έτσι ώστε να μπορούμε να εκτελούμε αναζήτηση στην λίστα κάθε φορά που ο χρήστης επιλέγει το κουμπί αναζήτησης ή πληκτρολογεί μία φράση

SearchView searchView = (SearchView) menu.findItem(R.id.*search*).getActionView();

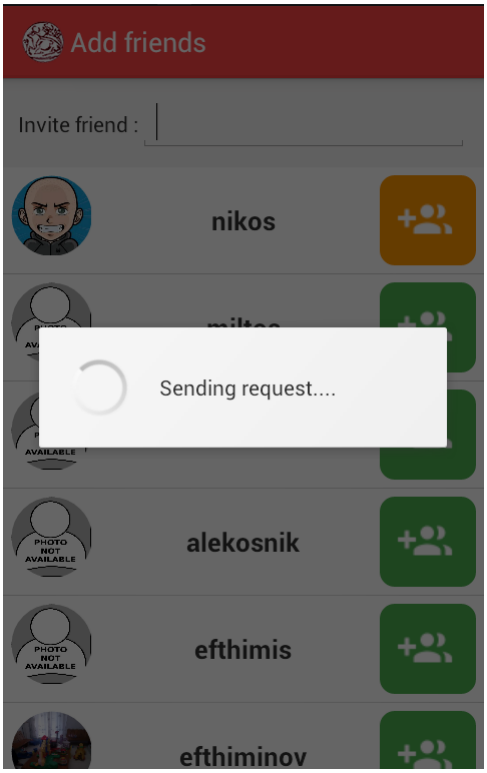
```
searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
 @Override
 public boolean onQueryTextSubmit(String query) {
 if (controller.getAdapter() != null) {
 controller.getAdapter().getFilter().filter(query);
 }
 return false;
 }

 @Override
 public boolean onQueryTextChange(String newText) {
 if (controller.getAdapter() != null) {
 controller.getAdapter().getFilter().filter(newText);
 }

 return false;
 }
});
```

Το φιλτράρισμα εκτελείται με την χρήση της κλάσης FriendFilter που επεκτείνει την κλάση Filter στην οποία όταν καλούμε την μέθοδο filter() σε οποιοδήποτε αντικείμενο της τότε έχουμε ορίσει να εκτελείται το εξής κομμάτι κώδικα μέσα στην FriendListAdapter

```
@Override
protected FilterResults performFiltering(CharSequence constraint) {
 constraint = constraint.toString().toLowerCase();
 FilterResults result = new FilterResults();
 if (constraint != null && constraint.toString().length() > 0) {
 ArrayList<Friend> filteredItems = new ArrayList<Friend>();
 for (int i = 0, l = originalList.size(); i < l; i++) {
 Friend friend = originalList.get(i);
 if (friend.username.toString().toLowerCase().contains(constraint))
 filteredItems.add(friend);
 // Toast.makeText(context, friend.username, Toast.LENGTH_LONG).show();
 }

 result.count = filteredItems.size();
 result.values = filteredItems;
 } else {

 synchronized (this) {
 result.values = originalList;
 result.count = originalList.size();
 }


 }
 return result;
}
```

6.4. Αποστολή αιτήματος φιλίας

Ο χρήστης μπορεί να επιλέξει οποιωνδήποτε χρήστη του συστήματος και να του αποστείλει ένα αίτημα φιλίας. Υπάρχει και εδώ η δυνατότητα search.



Εικόνα 17: Αποστολή αιτήματος φιλίας

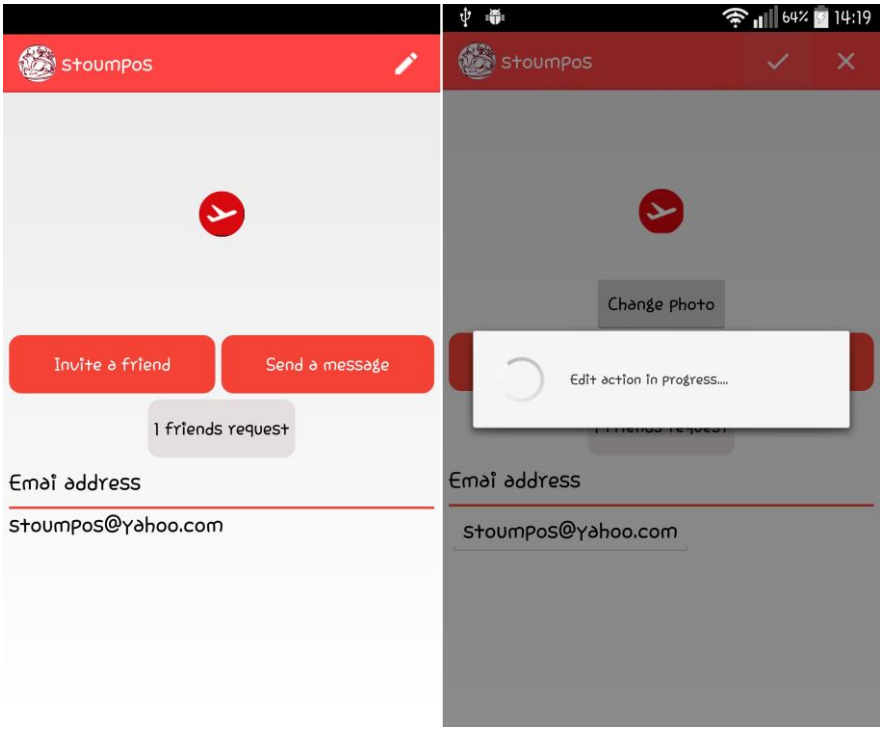
Εφόσον ο χρήστης επιλέξει το κουμπί αποστολής αιτήματος σε ένα οποιονδήποτε χρήστη της λίστας τότε καλείται η μέθοδος sendFriendRequest της κλάσης FriendController η οποία εκτελεί ασύγχρονα σε άλλο Thread μία HTTP κλήση για την αποστολή του αιτήματος στον Server.

```
new Thread(){

 @Override
 public void run() {

 try{
 List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(3);
 nameValuePairs.add(new BasicNameValuePair("tag", "sendfriendrequest"));
 nameValuePairs.add(new BasicNameValuePair("provider",
Integer.toString(manager.getUserId())));
 nameValuePairs.add(new BasicNameValuePair("receiver", Integer.toString(receiver)));
 HttpExecutor.makeHttpRequest(nameValuePairs);

CommonUtilities.raiseEvent(FriendHandler.FriendCodes.ON_SEND_FRIEND_REQUES
T,handler);

 }catch (Exception e){

CommonUtilities.raiseEvent(FriendHandler.FriendCodes.ON_ERROR_FRIEND_REQUE
```

```
ST,handler);
 }
 }
}.start();
```

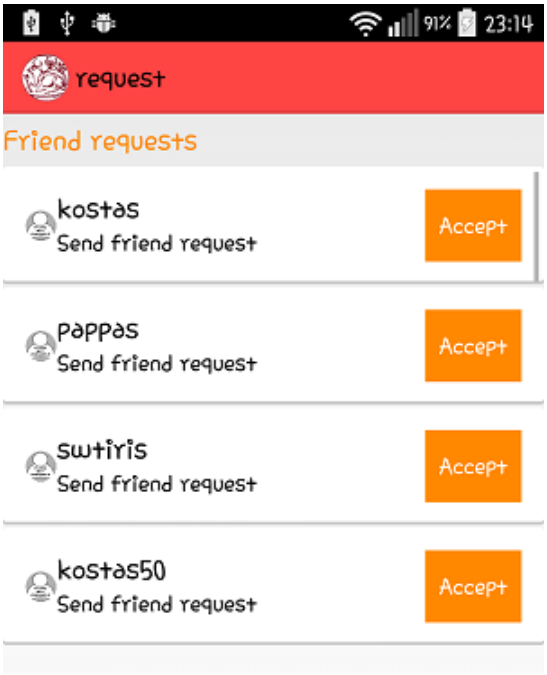
6.5. Προβολή προφίλ

Ο χρήστης έχει την δυνατότητα να δει πληροφορίες του προφίλ του όπως ο αριθμός των εισερχόμενων αιτημάτων φιλίας, την διεύθυνση του email του και την φωτογραφία του προφίλ του. Επίσης μπορεί να κάνει τροποποίηση της email address και της φωτογραφίας του προφίλ του. Στην προκειμένη περίπτωση επεξεργαστήκαμε την εικόνα προφίλ του χρήστη πατώντας το μολυβάκι στην επάνω δεξιά γωνία. Αφού το πατήσουμε μας εμφανίζει δύο επιλογές ένα '√' και ένα 'X' για να καταχωρίσουμε ή για να αποκρύψουμε τις αλλαγές αντίστοιχα. Για να γίνουν όλα αυτά καλείτε η **UserProfile.java** αφού πατήσει "ok" ο χρήστης καλείτε η μέθοδος **editUser** της **UserController** και αποστέλλονται τα τροποποιημένα δεδομένα στον Server.



Εικόνα 18: Προβολή προφίλ

6.6. Αποδοχή αιτήματος φιλίας



Εικόνα 19: Αιτήματα φιλίας

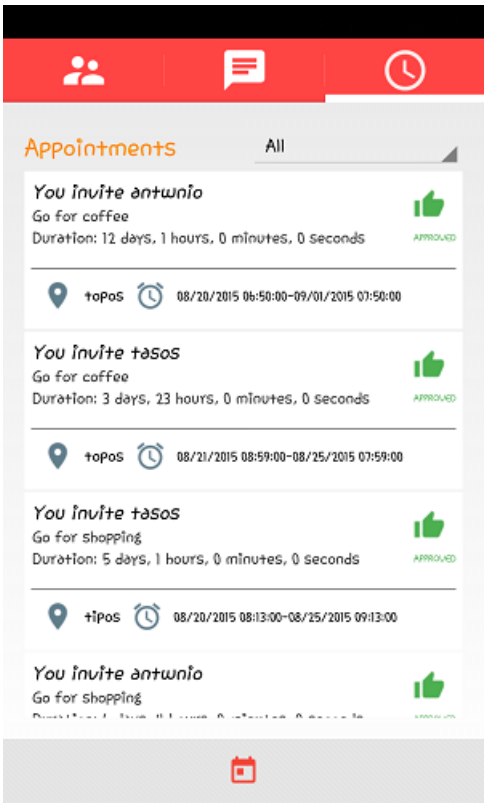
Στην διαδικασία αποδοχή αιτήματος φιλίας καλούμε την μέθοδο approveFriend της κλάσης FriendController η οποία εκτελείται και αυτή ασύγχρονα και αποστέλλει το ανάλογο event μόλις λάβει απόκριση από τον Server.

```
public void approveFriend(final int friendid){
   new Thread(){
      @Override
      public void run() {
         try{

            List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(3);
            nameValuePairs.add(new BasicNameValuePair("tag","approvefriend"));
            nameValuePairs.add(new
BasicNameValuePair("friendid",Integer.toString(friendid)));
            nameValuePairs.add(new BasicNameValuePair("userid",
Integer.toString(manager.getUserId())));

         HttpExecutor.makeHttpRequest(nameValuePairs);

CommonUtilities.raiseEvent(FriendHandler.FriendCodes.ON_APPROVE_FRIEND,handl
er);
          }catch (Exception e){

CommonUtilities.raiseEvent(FriendHandler.FriendCodes.ON_ERROR_APPROVE_FRIE
ND,e.getMessage(),handler);
```

```
        }

    }
}.start();
```

6.7. Προβολή Ραντεβού

Μέσα στην κλάση **AppointmentList**(η οποία αποτελεί ένα Fragment εφόσον επεκτείνει την SherlockFragment) εκτελούνται όλες οι απαραίτητες ενέργειες για την προβολή των ραντεβού. Στην συγκεκριμένη καρτέλα ο χρήστης μπορεί:
1) Να επιλέξει να δει την φόρμα ενός καταχωρημένου ραντεβού.
2) Να κάνει αναζήτηση φίλου για να δημιουργήσεις στην συνέχεια καινούργιο ραντεβού.
3) Ναι φιλτράρει τα ραντεβού ανάλογα με την κατάσταση του ή την διάρκεια τους.



Εικόνα 20: Λίστα των ραντεβού

Εφόσον αρχικοποιήσουμε τον **AppointmentController** και τον **AppointmentHandler** στην **onCreate()** στην συνέχεια καλούμε την μέθοδο **loadList(ListView ls)** της κλάσης **AppointmentController** η οποία αναλαμβάνει να ανακτήσει τα **appointments** που υπάρχουν υποθηκευμένα τοπικά στην **sqlite** βάση δεδομένων **IMServices**, να αρχικοποιήσει **instance** της κλάσης **AppointmentAdapter** με την λίστα των ραντεβού και να το θέσει στο **ListView** για προβολή.

```
@Override
public View onCreateView(LayoutInflater inflater, final ViewGroup container, Bundle
savedInstanceState) {
/…code…
```

```
AppointmentHandler appointmentHandler =new AppointmentHandler(this);
controller=new AppointmentController(getSherlockActivity(),appointmentHandler);
```

Φορτώνουμε στο ListView όλα τα ραντεβού με την δήλωση:

```
ListView listView=(ListView)rootView.findViewById(R.id.listView);
controller.loadList(listView);
listView=(ListView)rootView.findViewById(R.id.listView);

TextView title =(TextView)rootView.findViewById(R.id.dateType);
title.setText("Appointments");
```

Δημιουργούμε μια αναφορά στο **Spinner view** που έχουμε δηλώσει στο layout και αποτελεί ένα dropdown menu το οποίο τροφοδοτούμε στην συνέχεια με Strings:

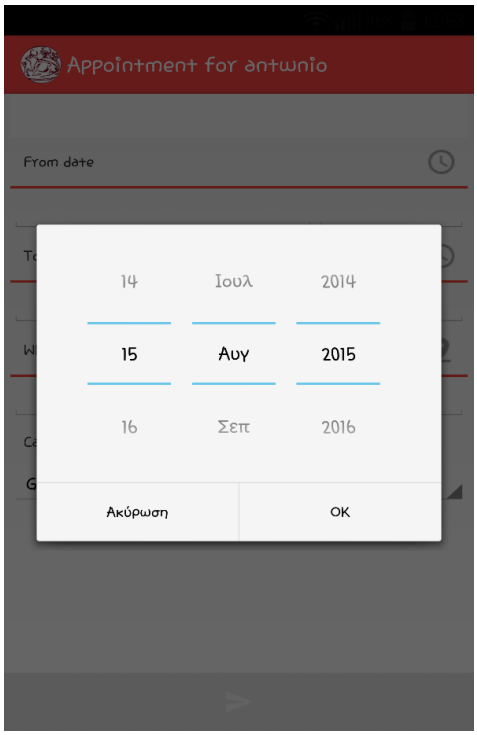
```
sp=(Spinner)rootView.findViewById(R.id.sort);

ArrayList<String> menuItems=new ArrayList<String>();
menuItems.add("All");
menuItems.add("Pending");
menuItems.add("Approved");
menuItems.add("Declined");
menuItems.add("Duration");
```

Θέτουμε στον **Spinner** ένα ανώνυμο instance της διεπαφής **onItemSelectedListener** και υλοποιούμε την μέθοδο **onItemSlected** για να εφαρμόσουμε το αντίστοιχο φιλτράρισμα στην λίστα ανάλογα με την επιλογή του χρήστη.

```
public static String FITER;

public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
   switch (position) {
      case 0:
         FITER = "ALL";
         controller.loadList(listView);
         break;
      case 1:
         FITER = "PENDING";
         controller.loadList(listView, FITER);
         break;
      case 2:
         FITER = "APPROVED";
         controller.loadList(listView, FITER);
         break;
      case 3:
         FITER = "DECLINE";
         controller.loadList(listView, FITER);
         break;
      case 4:
         FITER = "DURATION";
         controller.loadList(listView, FITER);
         break;
```

```
    }
}
```

6.8. Δημιουργία ραντεβού



Εικόνα 21: Δημιουργία ραντεβού

Αφού έχουμε επιλέξει το άτομο στο οποίο θέλουμε να στείλουμε το ραντεβού βάζουμε ημερομηνία και ώρα συνάντησης, ημερομηνία και ώρα λήξης και τέλος την τοποθεσία.
Για να ορίζουμε την ημερομηνία έχουμε κάνει import τις μεθόδους, οι οποίες βρίσκεται μέσα στην κλάση **AppointmentForm**

```
import android.app.DatePickerDialog;
import android.app.TimePickerDialog;
```

Η κλάση **AppointmentForm** χρησιμοποιείται για την δημιουργία Appointment. Ο χρήστης συμπληρώνει τα απαραίτητα πεδία και πατώντας την επιλογή Send αποστέλλει ένα ραντεβού στο server. Η επικοινωνία με το **AppointmentService** γίνεται με την υλοποίηση της **onServiceConnection** της κλάσης **AppointmentServiceConection** η οποία καλείται με την επιτυχημένη κλήση της **bindService**.

```
bindService(new Intent(this, AppointmentService.class),
appointmentServiceConnection, BIND_AUTO_CREATE);


private AppointmentService.AppointmentServiceInterface serviceInterface;


@Override

public void onServiceConnected(ComponentName name, IBinder service) {
```

```
    serviceInterface = (AppointmentService.AppointmentServiceInterface) service;

    serviceInterface.setAppointmentClientListener(appointmentClientListener);
}
```

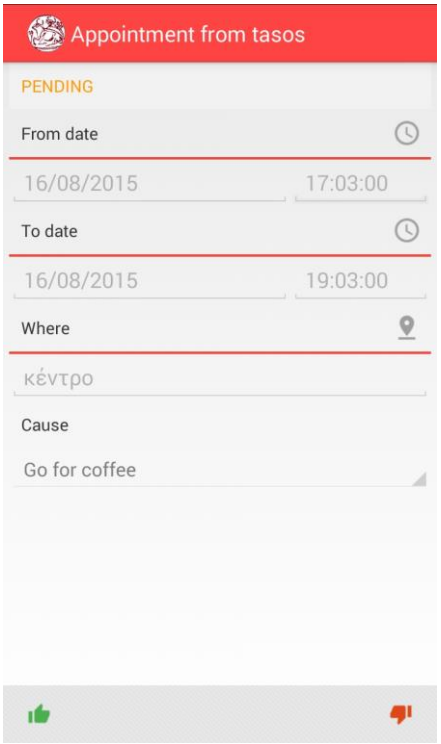
Η συγκεκριμένη κλάση **AppointmentService.java** είναι υπεύθυνη για την αποστολή και την μεταφόρτωση ραντεβού. Στην **onStartCommand** δημιουργούμε instance τις κλάσεις **AppointmentHandler** και **AppointmentController**.

```
handler = new AppointmentHandler(AppointmentService.this);
appointmentController = new AppointmentController(AppointmentService.this,
handler);
```

Αρχικοποιούμε τον **AppointmentDownloader** instance και καλούμε την μέθοδο **start()**:

```
AppointmentsDownloader appointmentsDownloader = new
AppointmentsDownloader(this);

appointmentsDownloader.registerController(appointmentController);

appointmentsDownloader.start();
```

6.9. Αποδοχή ή απόρριψη ραντεβού



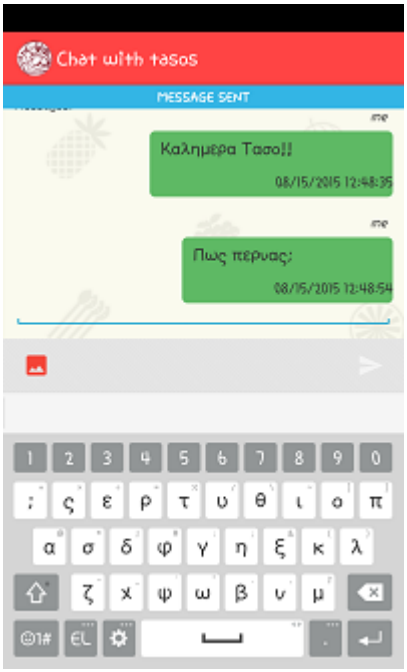
Εικόνα 22: Αίτημα για ραντεβού

Αφού στείλουμε το αίτημα στον χρήστη εμφανίζεται ένα Notification ότι έλαβε ένα ραντεβού αφότου πατήσει πάνω σε αυτό θα ανοίξει μια νέα σελίδα και θα τον οδηγήσει στο σημείο που θα τον ρωτήσει αν θέλει ή αν δεν θέλει να συμμετέχει στην συνάντηση.

Δημιουργούμε την **inner** κλάση **AppointmentReceiver** που επεκτείνει την **BroadCastReceiver** και λαμβάνει **BroadCast intents** από το **AppointmentService** που είναι υπεύθυνο για upload ή download των ραντεβού από τον server.

```
public class AppointmentReceiver extends BroadcastReceiver{
   @Override
   public void onReceive(Context context, Intent intent) {
      if(intent.getAction().equals(Appointment.ARRIVE)){
         int count =intent.getIntExtra("count",-1);
  String msg= count>1?" APPOINTMENTS ARRIVED":" APPOINTMENT ARRIVED";
   Configuration configuration = new Configuration.Builder().setDuration(
               Configuration.DURATION_INFINITE).build();

         mCrouton=Crouton.makeText(getSherlockActivity(), Integer.toString(count) +
msg, Style.INFO);

         mCrouton.show();

         controller.notifyList(FITER);
      }if(intent.getAction().equals(Appointment.APPROVED)){
         int count =intent.getIntExtra("count",-1);
         String msg= count>1?"APPOINTMENTS APPROVED":"APPOINTMENT
APPROVED";
         mCrouton=Crouton.makeText(getSherlockActivity(), Integer.toString(count) +
msg, Style.INFO);

         mCrouton.show();
         controller.notifyList(FITER);
      }if(intent.getAction().equals(Appointment.DECLINE)){
         int count =intent.getIntExtra("count",-1);
         String msg= count>1?"APPOINTMENTS DECLINED":"APPOINTMENT
DECLINED";
         mCrouton=Crouton.makeText(getSherlockActivity(), Integer.toString(count) +
msg, Style.INFO);

         mCrouton.show();
         controller.notifyList(FITER);

}if(intent.getAction().equals(AppointmentHandler.AppointmentCodes.ON_APPOINTME
NT_REFRESH_ERRROR)){
         Crouton.makeText(getSherlockActivity(), intent.getStringExtra("error"),
Style.INFO).show();
      }
   }
}
```

6.10. Αποστολή μηνυμάτων



Εικόνα 23: Αποστολή μηνυμάτων

Στην κλαση **MessageService.java** στην **onStartCommand** αρχικοποιούμε τα instance της κλάσεις **MessageHandler** και δημιουργούμε το αντίστοιχο instance **MessageController**.

```
public int onStartCommand(Intent intent, int flags, int startId) {

    msgHandler = new MessageHandler(this);
    msgController = new MessageController(MessageService.this, msgHandler);
```

Στην συνέχεια αρχικοποιούμε το instance της κλάσης **MessageDownloader**, θέτουμε με τη **registerController** τη μεταβλητή τη κλάσης με την τιμή το αντικείμενο **MessageController** και καλούμε την μέθοδο **start()**.

```
MessageDownloader msgDownloader = new MessageDownloader(this);
msgDownloader.registerController(msgController);
//START TIMER TASK
msgDownloader.start();
```

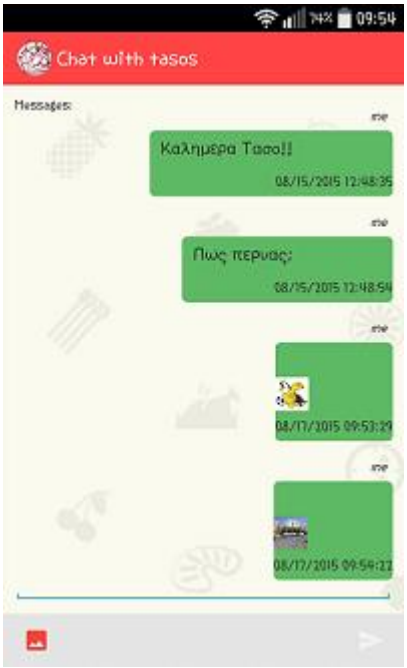
Δημιουργούμε στην συνέχεια την κλάση **MessageService** interface το οποίο επεκτείνει την κλάση **Binder**. Ένα instance της κλάσης επιστρέφεται στην **onBind()** το οποίοι είναι απαραίτητο για την σύνδεση της κλάσης **MessageActivity** με το αντίστοιχο service.

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

Η κλάση **MessageService** interface υλοποιεί μία σειρά από μεθόδους:

```
public void sendMessage(Message m) {
   msgController.sendMessage(m, this);
}


public class MessageServiceInterface extends Binder {

   public void sendMessage(Message m) {
      MessageService.this.sendMessage(m);
   }

   public void setReceiverId(int friendId) {
      MessageService.this.receiverId = friendId;
   }

   public void setClientListener(MessageClientListener listener) {
      MessageService.this.clientListener = listener;
   }
}
```

Αυτές οι τρεις μέθοδοι χρησιμοποιούνται από την client κλάση **MessageActivity** για την αποστολή μηνύματος **sendMessage(Message m)**, την αρχικοιποίηση της μετιαβλητής **receiverId** και την αρχικοποίηση της μεταβλητής **MessageClientListener clientListener** με το instance του **MessageClientListener** που έχει δημιουργηθεί στην **MessageActivity**. Με αυτό τον τρόπο επιτρέπεται η αμοιβαία επικοινωνία μεταξύ της **MessageActivity** και **MessageService**.



Εικόνα 24: Αποστολή μηνυμάτων

Η κλάση **MessageActivity** είναι υπεύθυνη για την προβολή του ιστορικού μηνυμάτων του χρήστη με ένα φίλο. Μπορεί επίσης να δημιουργήσει ένα μήνυμα κειμένου ή να αποστείλει μία φωτογραφία μέσα από τις επιλογές που διαθέτει στο menu.

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

```
MessageHandler handler=new MessageHandler(this);
msgController =new MessageController(this, handler);

ListView ls =(ListView)findViewById(R.id.chatList);
msgController.loadList(ls,providerId);

}
```

Στην **onResume** καλούμε την **bindService(new Intent(this, MessageService.class), messageServiceConnection, 0)** για να ανακτήσουμε μία αναφορά στο instance τύπου **MessageServiceInterface** της κλάσης **MessageService** και να έχουμε πρόσβαση σε μεθόδους της **MessageService** όπως την **sendMessage(Message m)**. Οι απαραίτητες αρχικοποιήσεις γίνονται μέσα από την υλοποίηση της μεθόδου **onServiceConnection** της **inner** κλάσης **MessageServiceConnection** η οποία υλοποιεί την **ServiceConnection** και καλείται με την επιτυχημένη κλήση της **bindService**.

```
@Override
   public void onServiceConnected(ComponentName name, IBinder service) {
      messageServiceInterface = (MessageService.MessageServiceInterface) service;
      messageServiceInterface.setClientListener(messageClientListener);
      messageServiceInterface.setReceiverId(providerId);
   }
}
```

Στην **onCreateOptionMenu** δημιουργούμε το menu με επιλογές την αποστολή γραπτού μηνύματος και την ανάκτηση φωτογραφίας.

```
Override
public boolean onCreateOptionsMenu(Menu menu) {
   menu.add("Photo")
         .setOnMenuItemClickListener(this.LikeButtonClickListener)
         .setIcon(R.drawable.ic_action_editor_insert_photo) // Set the menu icon
         .setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);


   menu.add("Send")
         .setOnMenuItemClickListener(this.sendButtonClickListener)
         .setIcon(R.drawable.ic_action_content_send) // Set the menu icon
         .setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);

   return super.onCreateOptionsMenu(menu);
}
```

Όταν ο χρήστης πατήσει την επιλογή Send τότε καλούμε την **sendMessage** της **MessageService**:

```
messageServiceInterface.sendMessage(m);
```

Αντίστοιχα όταν πατήσει την επιλογή Photo τότε ανοίγεται η εφαρμογή Photo Album για την επιλογή φωτογραφίας την οποία στην συνέχεια παραλαμβάνουμε ως Uri στην **onActivityResult**. Για να το υλοποιήσουμε αυτό όταν ο χρήστης πατήσει την επιλογή Photo τότε εκτελούμε το εξής κώδικα:

```
MenuItem.OnMenuItemClickListener LikeButtonClickListener = new
MenuItem.OnMenuItemClickListener() {

   public boolean onMenuItemClick(MenuItem item) {
```

```
        // getMessagesByProvider(providerId);
        Intent intent = new Intent(Intent.ACTION_PICK,
MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
        startActivityForResult(intent,CommonUtilities.RESULT_LOAD_IMAGE);
        return false;

    }
};
```

Στην **onActivityResult** εκτελούμε το αντίστοιχο **query** στο **global instance ContentResolver** με τη **getContentResolver.query(uri, projection, selection, selectionArgs, sortOrder)** η οποία επιστρέφει έναν **Cursor instance** για να διατρέξουμε τα δεδομένα που επιστρέφονται από το **query**. Εφόσον πάρουμε από το **query** το **filepath** της φωτογραφία που επιλέξαμε τότε το περνάμε σαν παράμετρο στην **displayDialog(String filepath)**.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == CommonUtilities.RESULT_LOAD_IMAGE && resultCode ==
RESULT_OK) {

        Uri selectedImage = data.getData();
        String[] filePathColumn = {MediaStore.Images.Media.DATA};
        Cursor cursor = getContentResolver().query(selectedImage,
                filePathColumn, null, null, null);
        if (cursor == null || cursor.getCount() < 1) {
            return; // no cursor or no record. DO YOUR ERROR HANDLING
        }

        cursor.moveToFirst();
        int columnIndex = cursor.getColumnIndex(filePathColumn[0]);

        if (columnIndex < 0) // no column index
            return;
        String picturePath = cursor.getString(columnIndex);
        cursor.close();

        displayDialog(picturePath);

    }
```

Στην **displayDialog** εξάγουμε το bitmap από το **filepath** της φωτογραφίας και το αποστέλλουμε στον server εφόσον ο χρήστη πατήσει Send κάνοντας το **decode** σε **base64 String**. Η κωδικοποίηση σε **base64** μεταφράζει στην ουσία **binary** δεδομένα σε χαρακτήρες **ASCII**.

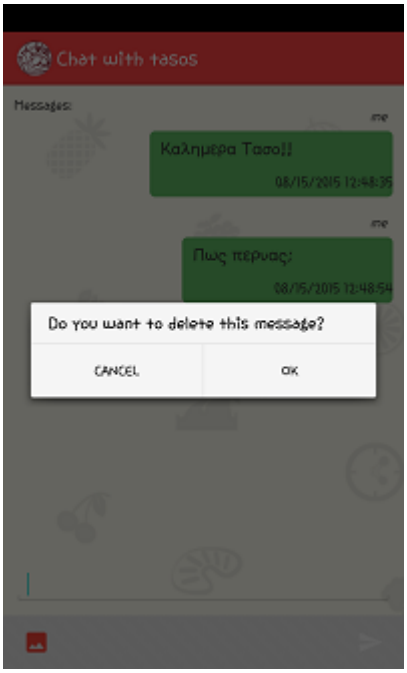
Κάθε αρχικό γράμμα ή σύμβολο αντιστοιχίζεται στον **ASCII** κωδικό του, που με τη σειρά του αντιστοιχίζεται στο δυαδικό του αντίστοιχο 8μπιτο αριθμό και τέλος, συνεχόμενες τριάδες τέτοιων 8-μπιτων αριθμών δημιουργούν ένα 24μπιτο αριθμό ο οποίος σπάει σε 4 εξάδες (6-bit) από αριστερά προς τα δεξιά. Κάθε μια εξάδα αντιστοιχεί σε έναν από τους 2^6 = 64 δυνατούς αριθμούς ο οποίος τέλος μεταφράζεται σε ένα από τα 64 σύμβολα του Base64 (συνήθως a-z, A-Z, 0-9, +, =και/).

```
public void displayDialog(String picturePath) {
        final Dialog d = new Dialog(this);
        d.setCancelable(false);
        d.setTitle("IMAGE PREVIEW");
        d.requestWindowFeature(android.view.Window.FEATURE_NO_TITLE);
        d.setContentView(R.layout.custom_dialog_preview);
```

```
        ImageView img = (ImageView) d.findViewById(R.id.selectedImage);

        try {
            Bitmap bm = BitmapFactory.decodeStream(
                    getContentResolver().openInputStream(Uri.parse("file://" + picturePath)));

            Bitmap scaledbitmap =
CommonUtilities.decodeSampledBitmapFromResourceMemOpt(new
FileInputStream(picturePath), 800, 600);
//decodeSampledBitmapFromResourceMemOpt,new FileInputStream(picturePath), 800,
600);
            capturedImage = scaledbitmap;


            img.setImageDrawable(CommonUtilities.getDrawable(bm,
MessageActivity.this));
            img.setAdjustViewBounds(true);

            img.setScaleType(ImageView.ScaleType.FIT_XY);
            d.show();


        } catch (Exception e) {

        }
        final EditText text = (EditText) d.findViewById(R.id.textMessage);

        Button declinde = (Button) d.findViewById(R.id.decline);
        declinde.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                capturedImage = null;
                d.dismiss();
            }
        });

        Button accept = (Button) d.findViewById(R.id.accept);
        accept.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                Message m = new Message(text.getText().toString(), manager.getUserId(),
                        providerId,
                        Utils.convertCalendarToString(),
                        manager.getUserId(),
                        Utils.encodeTobase64(capturedImage), -1, Message.READ,
conversationManager.guid);
                System.gc();
                //   manager.sendMessage(m);
                setSupportProgressBarIndeterminateVisibility(true);
                ls.setClickable(false);
                mText.setEnabled(false);
                messageServiceInterface.sendMessage(m);
                d.dismiss();
            }
        });
    }
}
```

Η δυνατότητα διαγραφής μηνυμάτων υλοποιείται και αυτή στην **MessageActivity** χρησιμοποιώντας την μέθοδο **deleteMessage** έχουμε αρχικοποιήσει τον **MessaageController** και έχουμε κάνει instanciate το **flagMessageDelete** από το **IMContentmanager** και το **notifylist**.

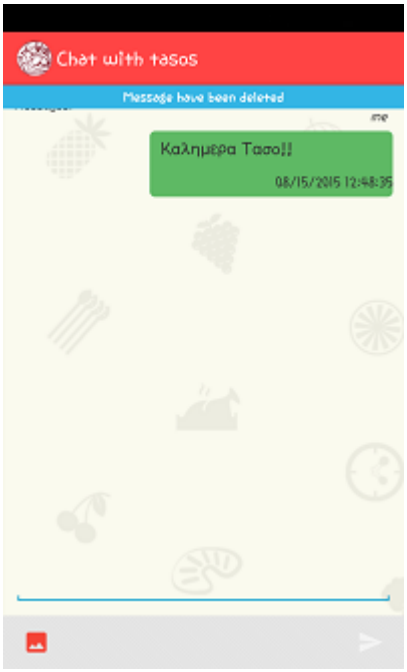
```
public void setFlagDelete(int messageId) throws Exception{
   SQLiteDatabase db = DatabaseManager.getInstance().openDatabase();
//getWritableDatabase();
   db.beginTransaction();

   try {
      ContentValues values = new ContentValues();
      values.put("status", "DELETED");
      db.update("messages", values,"messageId="+messageId,null);
      db.setTransactionSuccessful();
   } catch (Exception e) {
      throw e;
   } finally {
      db.endTransaction();
      DatabaseManager.getInstance().closeDatabase();

   }
}
```



Εικόνα 25: Διαγραφή μηνυμάτων

```
public static AlertDialog createOptionsDialog(final Context context, final int
messageId) {
      AlertDialog.Builder builder = new AlertDialog.Builder(context);
// Add the buttons
      builder.setMessage("Do you want to delete this message?");
      builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
         public void onClick(DialogInterface dialog, int id) {
            // User clicked OK button
            ((MessageActivity) context).deleteMessage(messageId);
         }
      });
```

```
        builder.setNegativeButton("CANCEL", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
                // User cancelled the dialog
                //controller.dele
            }
        });
        // Create the AlertDialog
        AlertDialog dialog = builder.create();
        return dialog;
    }
```



Εικόνα 26: Διαγραφή μηνυμάτων

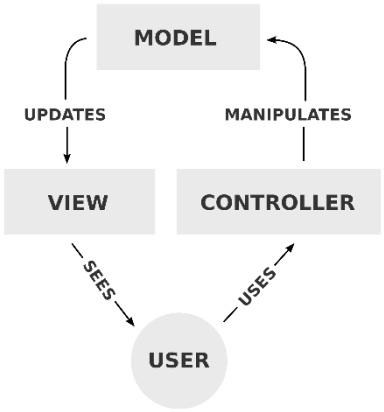
```
public void deleteMessage(int messageId) {
    msgController.flagMessageDelete(messageId);
    msgController.notifyList(providerId);
    Crouton.makeText(MessageActivity.this, "Message have been deleted",
Style.INFO).show();
}
```

6.11. Πρόσθετα Χαρακτηριστικά

Παρακάτω σας παραθέτουμε κάποια πρόσθετα χαρακτηριστικά που χρησιμοποιήσαμε για την δημιουργία του προγράμματος. Αυτά είναι το **MVC Architecture** ,**LRU Cash η βιβλιοθήκη ActionBarSherlock και η βιβλιοθήκη Crouton** .
Ποιο αναλυτικά:

- ***MVC Architecture***

Λίγο πολύ η διαδικασία που χρησιμοποιήσαμε για να κάνουμε εφικτή αυτήν την εφαρμογή μοιάζει με την MVC (Model-view-controller)



Εικόνα 27: MVC Architecture

Στο μοντέλο αυτό η εφαρμογή διαιρείται σε τρία διασυνδεδεμένα μέρη ώστε να διαχωριστεί η παρουσίαση της πληροφορίας στον χρήση από την μορφή που έχει αποθηκευτεί στο σύστημα. Το κύριο μέρος του μοντέλου είναι το αντικείμενο Model το οποίο διαχειρίζεται την ανάκτηση/αποθήκευση των δεδομένων στο σύστημα. Το αντικείμενο View χρησιμοποιείται μόνο για να παρουσιάζεται η πληροφορία στον χρήστη (π.χ. με γραφικό τρόπο). Το τρίτο μέρος είναι ο Controller ο οποίος δέχεται την είσοδο και στέλνει εντολές στο αντικείμενο Model και στο View.

Εκτός από το να διαιρείται η εφαρμογή σε τρία μοντέλα, η σχεδίαση model–view–controller ορίζει και τις αλληλεπιδράσεις των μοντέλων

a) Ο **controller** μπορεί να στέλνει εντολές στο μοντέλο και να ενημερώνει την κατάσταση του μοντέλου. Μπορεί επίσης να στέλνει εντολές ώστε να γίνει η αντίστοιχη αναπαράσταση των δεδομένων του μοντέλου μέσω του View.
b) Το **model** ενημερώνει τις αντίστοιχες αναπαραστάσεις views και τους controllers όταν υπάρχει αλλαγή στα δεδομένα. Αυτή η ενημέρωση επιτρέπει στα views να ενημερώνουν την γραφική απεικόνιση.
c) Το **view** αναπαριστά με γραφικό τρόπο την πληροφορία που περιέχει το model δημιουργώντας γραφική παρουσίαση στο χρήστη.

- ***LRU Cash***

Φορτώνοντας μια εικόνα στο user interface (UI) είναι εύκολο, όταν όμως θέλουμε να φορτώσουμε ένα πλήθος από εικόνες με μιας τα πράγματα γίνονται ποιο περίπλοκα. Η ποσότητα των εικόνων που βρίσκονται στην οθόνη με τα αυτές που θα ακολουθήσουν αργότερα είναι άπειρες.

Η χρήση της μνήμης κρατιέται σε χαμηλά επίπεδα όταν κάποια components ανακυκλώνονται όπως τα child Views. ο σκουπιδοφάγος ελευθερώνει μνήμη χωρίς με τις φορτωμένες εικόνες. Θέλοντας όμως εσύ να κρατήσεις τα επίπεδα λειτουργείας σε καλή λειτουργεία και το γρήγορο φόρτωμα του UI, πρέπει να αποφύγεις τέτοιου είδους συνεχόμενες διαδικασίες. Μια μνήμη cache μπορεί συχνά να βοηθήσει εδώ, επιτρέποντας να φορτώσετε γρήγορα επεξεργασμένες εικόνες.

Μια κρυφή μνήμη προσφέρει γρήγορη πρόσβαση σε εικόνες bitmap χωρίς να τραβάει διαθέσιμη μνήμη συστήματος. Η LRUCache (διαθέσιμη στην βιβλιοθήκη) είναι κατάλληλα εξοπλισμένη στο να φιλάει προσωρινά εικόνες bitmap.

Για να μπορέσουμε να διαλέξουμε τον κατάλληλο μέγεθος της LRUCache θα πρέπει να ληφθεί ένας αριθμός από παράγοντες υπόψη, για παράδειγμα:

a. Πόσο απαιτητικές σε μνήμη είναι η δραστηριότητες μιας εφαρμογής ή η εφαρμογή.
b. Πόσες φωτογραφίες θα είναι στην οθόνη με μιας ή το πόσες θα πρέπει να είναι σε διαθεσιμότητα για να προβληθούν στην οθόνη.
c. Ποιο είναι το μέγεθος και η πυκνότητα της συσκευής, μια έξτρα υψηλής πυκνότητας οθόνη θα χρειαστεί μεγαλύτερη cache να κρατήσει τον ίδιο αριθμό εικόνων.
d. Τι διαστάσεις και διαμόρφωση είναι οι εικόνες bitmap και, ως εκ τούτου πόση μνήμη αναλάβει για την κάθε μια.
e. Πόσο συχνά μια εικόνα θα γίνει accessed
f. Ισορροπία ποιότητας με ποσότητα. Μερικές φορές είναι ποιο εφικτό να φυλάξεις μεγαλύτερο αριθμό εικόνων μικρότερης ποιότητας, ενδεχομένως μεγαλύτερης ποιότητας εικόνων γίνεται σε άλλο διεργασία στο παρασκήνιο.

- **ActionBarSherlock**

Αποτελεί την βιβλιοθήκη η οποία προσφέρει την δυνατότητα δημιουργίας ActionBar ακόμα και σε εφαρμογές που τρέχουν σε λειτουργικό Android έκδοσης 2.3.3. Ο λόγος που χρησιμοποιήσαμε την συγκεκριμένη βιβλιοθήκη είναι ότι μας επιτρέπεται μέσω του Web εργαλείου Android Asset Studio https://romannurik.github.io/AndroidAssetStudio/ να δημιουρήσουμε ειδικά για αυτή την βιβλιοθήκη όλα τα απαραίτητα png και themes με έναν αυτοματοποιημένο και γρήγορο τρόπο.

- **Crouton**

Η συγκεκριμένη βιβλιοθήκη μας επιτρέπει να δημιουργήσουμε αναδυόμενα μηνύματα με ένα εύκολο τρόπο και με μία αισθητική πολύ καλύτερη από αυτήν που προσφέρει η κλάση Toast για τα αναδυόμενα μηνύματα.

Βιβλιογραφία

1. **Google**. Android. Development. [Ηλεκτρονικό] http://developer.android.com/.
2. **Wikipedia**. [Ηλεκτρονικό] http://en.wikipedia.org/wiki/Main_Page.
3. **Android Developers Blog**. Android Development. Blog. [Ηλεκτρονικό] http://android-developers.blogspot.gr/.
4. **StackOverflow**. [Ηλεκτρονικό] http://stackoverflow.com/.
5. **Amal, Raj**. Learn2Crack. [Ηλεκτρονικό] http://www.learn2crack.com/.
6. **Tamada, Ravi**. AndroidHive. [Ηλεκτρονικό] http://www.androidhive.info/.
7. **Vogel, Lars**. Vogella. [Ηλεκτρονικό] http://www.vogella.com/.
8. **Azzola, Francesco**. SurvivingWithAndroid. [Ηλεκτρονικό] http://www.survivingwithandroid.com/.
9. **M, Adnan A**. MyAndroidTuts Blogspot. [Ηλεκτρονικό] http://myandroidtuts.blogspot.gr/.
10. **FoamyGuy και nostra13**. GitHub. [Ηλεκτρονικό] https://github.com/.
11. **Mathew, George**. wpTrafficAnalyzer Blog. [Ηλεκτρονικό] http://wptrafficanalyzer.in/blog/.
12. **Oracle**. Java. JDK. [Ηλεκτρονικό] http://www.oracle.com/technetwork/articles/javase/index-jsp-138363.html.
13. **000 Web Host**. [Ηλεκτρονικό] http://www.000webhost.com/order.php.
14. **W3Schools**. [Ηλεκτρονικό] http://www.w3schools.com/.
15. **ActionBarSherlock**. [Ηλεκτρονικό] http://actionbarsherlock.com/.
16. **Grokking Android**. [Ηλεκτρονικό] http://www.grokkingandroid.com/
17. **EazyTutz**. [Ηλεκτρονικό] http://www.eazytutz.com/