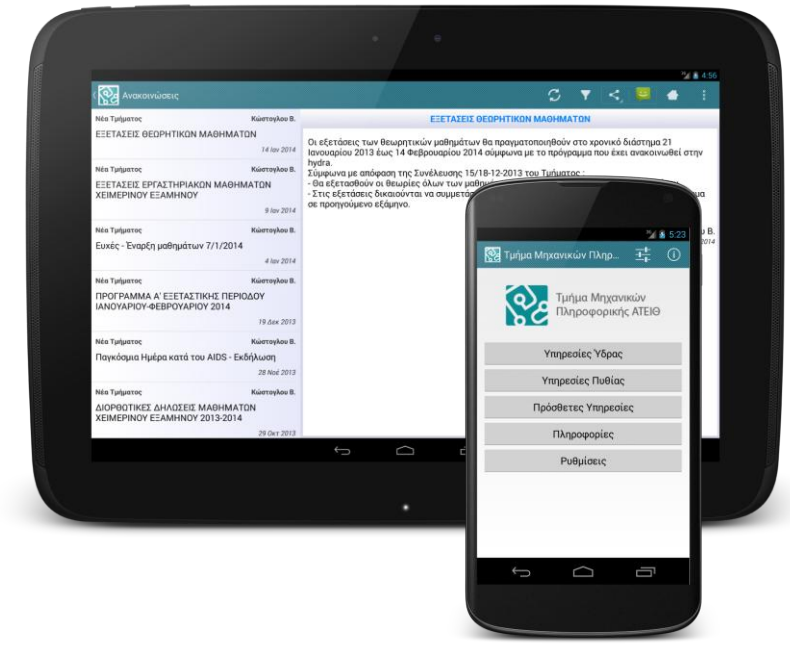
ΑΛΕΞΑΝΔΡΕΙΟ ΤΕΧΝΟΛΟΓΙΚΟ
ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Τμήμα Μηχανικών Πληροφορικής ΑΤΕΙΘ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Εφαρμογή Τμήματος Μηχανικών Πληροφορικής ΑΤΕΙΘ στην πλατφόρμα Android



Του φοιτητή

Κυριμλίδη Νικόλαου

Αρ. Μητρώου: 04/2475

Επιβλέπων καθηγητής

Κεραμόπουλος Ευκλείδης

Θεσσαλονίκη 2014

Περίληψη

Σκοπός της πτυχιακής εργασίας ήταν η ανάπτυξη εφαρμογής του Τμήματος Μηχανικών Πληροφορικής του ΑΤΕΙΘ για την πλατφόρμα Android.

Αρχικά γίνεται μια εισαγωγή στο σύστημα Android την ιστορία του και την εξέλιξη του καθώς και τα βασικά χαρακτηριστικά του. Παρουσιάζονται οι δυνατότητες της πλατφόρμας και τεχνικές ανάπτυξης εφαρμογών. Έμφαση δίνεται στην προς τα πίσω συμβατότητα καθώς χρησιμοποιούνται χαρακτηριστικά που εισήχθηκαν στις νεότερες εκδόσεις αλλά κυρίως στην ανάπτυξη για βέλτιστη υποστήριξη των διαφόρων μεγεθών οθονών καθώς το Android τρέχει σε μεγάλο αριθμό διαφορετικών συσκευών.

Παρουσιάζονται τα βασικά στοιχεία ανάπτυξης που αποτελούν κομμάτι κάθε εφαρμογής, η ανάπτυξη διεπιφανειών χρήστη, υπηρεσιών και εργασιών παρασκηνίου. Περιγράφονται οι τρόποι ανάκτησης και αποθήκευσης των δεδομένων μας και τέλος παρουσιάζεται η εφαρμογή μας δίνοντας την περιγραφή των λειτουργιών και πώς αυτές υλοποιούνται με τις μεθόδους που παρουσιάσαμε.

Abstract

The purpose of the thesis was the development of an application for the Department of Information Technology of the ATEITH on the Android platform.

Initially we introduce the Android system, its history, evolution and basic features. Capabilities and application development techniques are presented. Attention was given to retain backwards compatibility since we use a lot of the features that were only introduced in the recent editions as well as support for various screen sizes since Android runs on devices with various configurations.

We present the basic elements of the development that is part of any Android application, the development of the user interface, services and background tasks. We describe the methods of retrieving and storing our data and close with the presentation of our application giving a detailed description of its features and the way they are implemented.

ΕΙΣΑΓΩΓΗ

Στην παρούσα πτυχιακή εργασία παρουσιάζονται οι δυνατότητές της πλατφόρμας Android και με τη χρήση της αναπτύχθηκε εφαρμογή για smartphones και tablets.

Η εφαρμογή παρουσιάζει το τμήμα Μηχανικών Πληροφορικής του ΑΤΕΙΘ και παρέχει πρόσβαση στις υπηρεσίες του τμήματος όπως είναι οι ανακοινώσεις και οι βαθμολογίες αλλά και σε πρόσθετες υπηρεσίες όπως ο χάρτης της σχολής.

Κύριος στόχος της πτυχιακής εργασίας ήταν η υλοποίηση της εφαρμογής αυτής προσπαθώντας παράλληλα να γίνει χρήση των νεώτερων λειτουργιών διατηρώντας την συμβατότητα με τις παλιότερες εκδόσεις της πλατφόρμας.

Η εφαρμογή παρέχει διαφορετικές προβολές του περιεχόμενου, ανάλογα με την μέγεθος της οθόνης της συσκευής, εκμεταλλευόμενη τις μεγαλύτερες οθόνες των tablets και τις δυνατότητες που εισήγαγαν οι νεότερες εκδόσεις. Προσοχή έπρεπε όμως να δοθεί ώστε να διατηρηθεί η συμβατότητα και με τις παλιότερες εκδόσεις καθώς διατηρούν ακόμα ένα μεγάλο κομμάτι της αγοράς. Σε αυτό βοήθησε και η βιβλιοθήκη υποστήριξης(Android Support Library) η οποία επιτρέπει την χρήση πολλών εκ των νέων χαρακτηριστικών.

Υλοποιήθηκε με τη χρήση του πακέτου Android Development Tools(ADT) Bundle το οποίο περιλαμβάνει το Eclipse IDE, Android SDK και όλα τα απαραίτητα εργαλεία για την ανάπτυξη εφαρμογών στην πλατφόρμα Android. Χρησιμοποιήθηκαν επίσης οι υπηρεσίες χαρτών και πλοήγησης της Google καθώς και οι εξωτερικές Java βιβλιοθήκες JSoup(HTML parser), και η μεταφορά του Java Mail API για χρήση στο Android.

Πίνακας Περιεχομένων

Κεφάλαιο 1: Εισαγωγή στο Android

Το Android είναι στη βάση του ένα λειτουργικό σύστημα ανοιχτού κώδικα βασισμένο στο Linux το οποίο έχει σχεδιαστεί με βασικό στόχο φορητές συσκευές αφής όπως smart phones και tablets. Αναπτύχθηκε αρχικά από την Android Inc. η οποία υποστηρίχτηκε οικονομικά και έπειτα εξαγοράστηκε από την Google το 2005. Το 2007 σχηματίστηκε η Open Handset Alliance, μια κοινοπραξία 34 εταιριών με πρωτοστάτη τη Google με σκοπό να εξελίξουν περεταίρω το σύστημα και να αναπτύξουν ανοιχτά πρότυπα ώστε να προάγουν την καινοτομία, να βελτιώσουν την εμπειρία των χρηστών και να μειώσουν τα κόστη παραγωγής στον τομέα των φορητών συσκευών.

Το Android δεν είναι μόνο το λειτουργικό σύστημα αλλά ένα ολόκληρο οικοσύστημα που αποτελείται από τον συνδυασμό τριών στοιχείων:

- Ένα ελεύθερο, ανοιχτού κώδικα λειτουργικό σύστημα
- Μια ανοιχτού κώδικα πλατφόρμα ανάπτυξης λογισμικού
- Συσκευές, οι οποίες χρησιμοποιούν το λειτουργικό καθώς και οι εφαρμογές του

1.1 Ιστορία και εξέλιξη του Android

Η Android Inc. ιδρύθηκε στο Palo Alto, της California τον Οκτώβριο του 2003 από τους Andy Rubin, Rich Miner, Nick Sears και Chris White με σκοπό να αναπτύξουν κατά τα λεγόμενα του Rubin "έξυπνες συσκευές οι οποίες θα είχαν επίγνωση της τοποθεσίας του χρήστη και των προτιμήσεων του". Παρότι προηγούμενα επιτεύγματα των ιδρυτών και των αρχικών εργαζομένων ήταν δημοσίως γνωστά η εταιρία λειτούργησε με άκρα μυστικότητα αποκαλύπτοντας μόνο ότι δουλεύει πάνω σε λογισμικό για κινητά τηλέφωνα.

Στις 17 Αυγούστου 2005 η Google εξαγόρασε την Android Inc. κάνοντας την θυγατρική της. Τα κύρια μέλη της εταιρίας παρέμειναν και εκεί ανέπτυξαν μια πλατφόρμα για φορητές συσκευές βασισμένη στον πυρήνα του Linux. Η Google διέθεσε την πλατφόρμα στους κατασκευαστές τηλεφώνων με την υπόσχεση να παρέχει ένα ευέλικτο και αναβαθμίσιμο σύστημα καθώς και ότι ήταν διατεθειμένη να συνεργαστεί με κατασκευαστές υλικού και λογισμικού για την περεταίρω ανάπτυξη και βελτίωση του.

Στις 5 Νοεμβρίου 2007 παρουσιάστηκε στο ευρύ κοινό η Open Handset Alliance, μια κοινοπραξία 34 εταιριών στο τομέα των τεχνολογιών πληροφορικής και επικοινωνιών με πρωτεργάτη τη Google και περιλάμβανε κατασκευαστές συσκευών, ολοκληρωμένων κυκλωμάτων καθώς και παρόχων τηλεφωνικών και ασύρματων υπηρεσιών. Ως κύριος στόχος τέθηκε η ανάπτυξη ανοιχτών προτύπων για κινητές συσκευές. Την ίδια μέρα παρουσιάστηκε και το Android ως το πρώτο τους προϊόν, μια πλατφόρμα για κινητές συσκευές βασισμένη στην έκδοση 2.6 του πυρήνα του Linux. Το πρώτο κινητό τηλέφωνο(HTC Dream) που χρησιμοποιούσε το νέο λειτουργικό έγινε εμπορικά διαθέσιμο στις 22 Οκτωβρίου 2008.

Εικόνα 1: HTC Dream(G1) η πρώτη συσκευή με λειτουργικό Android

Από το 2008 έχουν γίνει πολυάριθμες αναβαθμίσεις οι οποίες βελτίωσαν σταδιακά το λειτουργικό σύστημα, πρόσθεσαν νέες δυνατότητες και διόρθωσαν προβλήματα των

προηγούμενων εκδόσεων. Από το 2010 η Google ξεκίνησε τη δική της σειρά smart phones και tablets, κατασκευασμένα σε συνεργασία με μεγάλους κατασκευαστές κινητών, για να λειτουργήσουν ως η ναυαρχίδα για την παρουσίαση των νέων εκδόσεων του λειτουργικού συστήματος.

Σήμερα πλέον το Android κατέχει το 80% της αγοράς smart phones με περισσότερες από 1 εκατομμύριο ενεργοποιήσεις την ημέρα. Η OHA αριθμεί πλέον 84 μέλη και συνεχίζει να εξελίσσει ενεργά την πλατφόρμα. Η χρήση όμως του λειτουργικού δεν περιορίζεται στις συσκευές κινητής τηλεφωνίας κυκλοφορεί ήδη μεγάλος αριθμός άλλων συσκευών όπως tablets, e-readers αλλά και media players, netbooks, κονσόλες και τηλεοράσεις.

1.2 Ιστορικό εκδόσεων - Χαρακτηριστικά

Το Android βρίσκετε υπό συνεχή ανάπτυξη και καινούργιες εκδόσεις του κυκλοφορούν κάθε μερικούς μήνες, συνήθως 6 με 9, παρέχοντας βελτιώσεις και νέες δυνατότητες. Από την έκδοση 1.5 και μετά κάθε σημαντική αναβάθμιση παίρνει το όνομα ενός "γλυκού" πχ Cupcake, Donut. Τα παρακάτω δεν είναι εκτενής λίστα των δυνατοτήτων του Android άλλα μια παρουσίαση κάποιων βασικότερων χαρακτηριστικών που εισήγαγε κάθε έκδοση.

Android 1.0: Η πρώτη εμπορική έκδοση του λειτουργικού κυκλοφόρησε στις 23 Σεπτεμβρίου 2008. Περιλαμβάνει τα:

- Android Market για την λήψη και αναβάθμιση εφαρμογών
- Web browser ο οποίος παρέχει βασικές λειτουργίες προβολής HTML ιστοσελίδων
- Βασικές εφαρμογές για τις υπηρεσίες της Google(Gmail, Maps, YouTube player κλπ) με τις οποίες το λειτουργικό στενά ολοκληρωμένο.

Android 1.5 Cupcake: Κυκλοφόρησε στις 27 Απριλίου 2009 και βασιζόταν στον πυρήνα Linux 2.6.27. Αυτή ήταν η πρώτη έκδοση που χρησιμοποιεί επίσημα μια κωδική ονομασία που βασίζεται σε ένα επιδόρπιο ("Cupcake"), ένα θέμα το οποίο θα χρησιμοποιείται για όλες τις εκδόσεις του πλέον. Η ενημερωμένη έκδοση περιλαμβάνει αρκετά νέα χαρακτηριστικά και τροποποιήσεις στο γραφικό περιβάλλον:

- Παρέχει υποστήριξη για «εικονικά» πληκτρολόγια με πρόβλεψη κειμένου και λεξικό για προσθήκη νέων λέξεων.
- Προστέθηκαν τα Widgets, μικρές εφαρμογές που προσθέτονται στην αρχική οθόνη και μπορούν να ανανεώνονται τακτικά ώστε να προβάλουν ενημερωμένο περιεχόμενο.
- Εγγραφή και αναπαραγωγή βίντεο σε μορφή MPEG-4 και 3GP.
- Υποστήριξη για Bluetooth (προφίλ A2DP και AVRCP).
- Επιλογή για αυτόματη εναλλαγή προσανατολισμού.
- Δυνατότητα για upload περιεχομένου σε YouTube και Picasa.

Android 1.6 Donut: Κυκλοφόρησε στις 15 Σεπτεμβρίου 2009 και βασιζόταν στον πυρήνα Linux 2.6.29.

- Πολύγλωσση μηχανής σύνθεσης ομιλίας που επιτρέπει σε κάθε εφαρμογή του Android να «μιλήσει»(Text to Speech).
- Βελτίωση προβολής και αναζήτησης στο Android Market.

- Gallery, φωτογραφική μηχανή και βιντεοκάμερα έχουν ενσωματωθεί πλήρως, με ταχύτερη πρόσβαση στη κάμερα.
- Ενημέρωση υποστήριξης της τεχνολογίας για CDMA / EVDO, 802.1x, VPNs.
- Υποστήριξη για μεγαλύτερη ανάλυση οθόνης(WVGA)
- Επεκταμένο πλαίσιο χειρονομιών(Gestures) και νέο εργαλείο ανάπτυξης(GestureBuilder)
- Δυνατότητα για τους προγραμματιστές να συμπεριλάβει το περιεχόμενό τους στα αποτελέσματα αναζητήσεων.

Android 2.0/2.1 Éclair: Κυκλοφόρησε στις 26 Οκτωβρίου 2009 και βασιζόταν στον πυρήνα Linux 2.6.29.

- Υποστήριξη Bluetooth 2.1
- Υποστήριξη Microsoft Exchange
- Προσθήκη live wallpapers.
- Βελτιστοποιημένη ταχύτητα υλικού και ανανεωμένο UI.
- Υποστήριξη για περισσότερα μεγέθη οθονών και αναλύσεων.
- Ανανεωμένο UI στον Browser και υποστήριξη για HTML5.

Android 2.2 Froyo: Κυκλοφόρησε στις 20 Μάιου 2010 και βασιζόταν στον πυρήνα Linux 2.6.32.

- Βελτιστοποιήσεις ταχύτητας, μνήμης και απόδοσης.
- Βελτίωση της απόδοσης των εφαρμογών με χρήση JIT(Just In Time) compilation.
- Προσθήκη V8 JavaScript Engine του Chrome στον Browser.
- Βελτιωμένη υποστήριξη Microsoft Exchange.
- Υποστήριξη για την εγκατάσταση εφαρμογών στην επεκτάσιμη μνήμη.
- Η υποστήριξη για οθόνες υψηλής πυκνότητας (μέχρι 320 ppi) , όπως 4" οθόνες 720p.

Android 2.3.x Gingerbread: Κυκλοφόρησε στις 6 Δεκεμβρίου 2010 και βασιζόταν στον πυρήνα Linux 2.6.35.

- Απλοποίηση του γραφικού περιβάλλοντος για αύξηση της λειτουργικότητας και της απόδοσης.
- Προσθήκη Download Manager για ευκολότερη πρόσβαση στα ληφθέντα αρχεία.
- Υποστήριξη για Near Field Communication (NFC).
- Βελτιωμένη διαχείριση ενέργειας.
- Το σύστημα αρχείων άλλαξε από YAFFS σε ext4.
- Concurrent garbage collection για αυξημένη απόδοση.
- Εγγενής υποστήριξη για περισσότερους αισθητήρες όπως γυροσκόπια και βαρόμετρα.

Android 3.0 - 3.2 Honeycomb: Κυκλοφόρησε στις 22 Φεβρουαρίου 2011 η πρώτη Android ενημέρωση μόνο για tablets, βασίζεται σε πυρήνα Linux 2.6.36.

- Βελτιωμένη υποστήριξη tablet με μια νέα εικονική και "ολογραφική" διεπαφή χρήστη.
- Προσθήκη Μπάρας Συστήματος(System Bar ή Navigation Bar) στο κάτω μέρος της οθόνης, η οποία διαθέτει γρήγορη πρόσβαση στις κοινοποιήσεις, το καθεστώς, και τα πλήκτρα πλοήγησης τα οποία αντικαθιστούν τα πραγματικά πλήκτρα που υπήρχαν σε παλιότερες συσκευές.

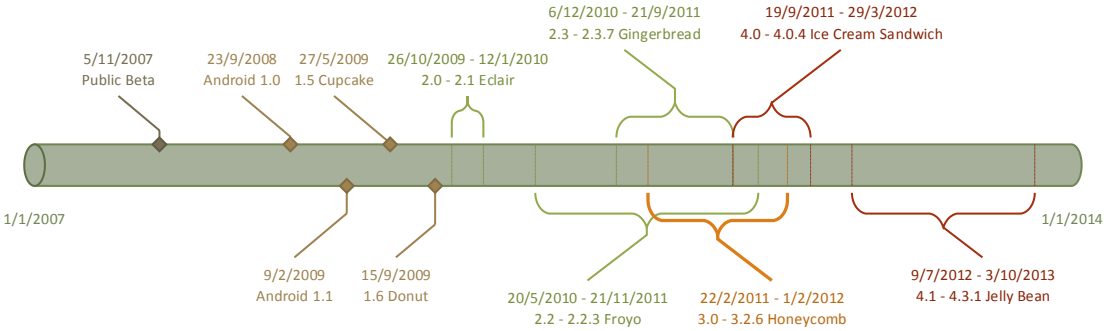
- Προσθήκη Μπάρας Ενεργειών(Action Bar), δίνοντας πρόσβαση σε επιλογές ανάλογα με το τρέχων περιεχόμενο, πλοήγηση, widgets στο πάνω μέρος της οθόνης.
- Υποστήριξη επεξεργαστών πολλαπλών πυρήνων.
- Ενημερωμένο UI
    - Προσαρμόσιμη Αρχική Οθόνη(Home Screen)
    - Προβολή πρόσφατων εφαρμογών.
    - Νέα διάταξη πληκτρολογίου
- Νέο API για την διαχείριση των διαφόρων τύπων οθονών(ανάλυση, πυκνότητα)
    - Νέα προσδιοριστικά πόρων
    - Fragments API
    - Λειτουργία συμβατότητας για παλιότερες εφαρμογές

Android 4.0-4.0.4 Ice-cream Sandwich: Κυκλοφόρησε στις 19 Οκτωβρίου 2011 και βασίζεται σε πυρήνα Linux 3.0.1. Ήταν θεωρητικά συμβατό με κάθε συσκευή που υποστήριζε την έκδοση 2.3.x. Αυτή η έκδοση εισήγαγε πολλά νέα στοιχεία αλλά και ενοποίησε πολλά από αυτά που εμφανιστήκαν στην έκδοση 3 επιτρέποντας την χρήση τους και σε τηλέφωνα εκτός από tablets.

- Ενοποιημένο πλαίσιο γραφικού περιβάλλοντος.
    - Action Bar
    - Multi-pane layouts, Fragments API
    - Οικογένεια γραμματοσειρών Roboto
    - Κοινές πρακτικές σχεδιασμού
- Υποστήριξη Wi-Fi Direct
- Android VPN Framework
- Εφαρμογή Android Beam για την ανταλλαγή δεδομένων μέσω NFC.
- Εμφανίζεται για πρώτη φορά στο Android ο Chrome browser.

Android 4.1/4.2/4.3 Jelly Bean: Κυκλοφόρησε στις 9 Ιουλίου 2012 και βασίζεται σε πυρήνα Linux 3.0.31. Η ενημέρωση είχε ως πρωταρχικό σκοπό τη βελτίωση της λειτουργικότητας και της απόδοσης του γραφικού περιβάλλοντος. Ο σκοπός αυτός επιτεύχθηκε μέσω του “Project Butter”, το οποίο χρησιμοποιεί touch anticipation, triple buffering, v-sync και ένα σταθερό ρυθμό καρέ στα 60fps για να δημιουργήσει ένα ομαλό και ευχάριστο UI.

- Υποστήριξη για ασύρματες σύνδεση με οθόνες μέσω Miracast.
- Google Chrome είναι πλέον ο default browser του συστήματος.
- Επεκτάσιμες ειδοποιήσεις
- Βελτιώσεις στη Οθόνη Κλειδώματος(Lock Screen) για προσθήκη widgets.
- Πολλαπλοί λογαριασμοί χρηστών(έκδοση 4.2 και μόνο για tablets)
- Υποστήριξη για 4K αναλύσεις(έκδοση 4.3)
- Υποστήριξη Open GLES 3.0



Εικόνα 2: Χρονοδιάγραμμα εκδόσεων

1.3 Επίπεδο API

Το επίπεδο API είναι ένας ακέραιος αριθμός που προσδιορίζει μοναδικά την έκδοση API του πλαισίου που προσφέρεται από μια έκδοση της πλατφόρμας Android.

Η πλατφόρμα Android παρέχει όπως είπαμε ένα πλαίσιο με το οποίο οι εφαρμογές μπορούν να χρησιμοποιούν για να αλληλεπιδρούν με το σύστημα Android. Κάθε διαδοχική έκδοση της πλατφόρμας Android μπορεί να περιλαμβάνει ενημερώσεις για το πλαίσιο εφαρμογών που προσφέρει. Οι ενημερώσεις σχεδιάζονται ώστε η νέα έκδοση να εξακολουθεί να είναι συμβατή με προηγούμενες εκδόσεις του API. Δηλαδή, οι περισσότερες αλλαγές στο API είναι προσθήκες νέων λειτουργιών ή αντικατάσταση παλαιότερων. Καθώς νέα μέρη προστίθενται τα παλιότερα αντικαθιστώνται, αυτά ορίζονται ως παρωχημένα αλλά δεν αφαιρούνται, ώστε οι υπάρχουσες εφαρμογές να μπορούν να τα χρησιμοποιούν ακόμα. Σε ένα πολύ μικρό αριθμό περιπτώσεων, τα μέρη του API μπορεί να τροποποιηθούν ή να αφαιρεθούν, αν και συνήθως τέτοιες αλλαγές γίνονται μόνον για να εξασφαλιστεί η σταθερότητα και η ασφάλεια του συστήματος. Παρόλα αυτά αλλαγές στις υλοποιήσεις μπορεί να αλλάξουν τον τρόπο λειτουργίας μεταξύ των διαφόρων εκδόσεων.

Το API πλαίσιο που η πλατφόρμα Android προσφέρει προσδιορίζεται χρησιμοποιώντας ένα ακέραιο αναγνωριστικό που ονομάζεται "επίπεδο API". Κάθε έκδοση της πλατφόρμας Android υποστηρίζει ακριβώς ένα επίπεδο API, αν και η υποστηρίζει όλα τα προηγούμενα επίπεδα (μέχρι και το επίπεδο API 1).

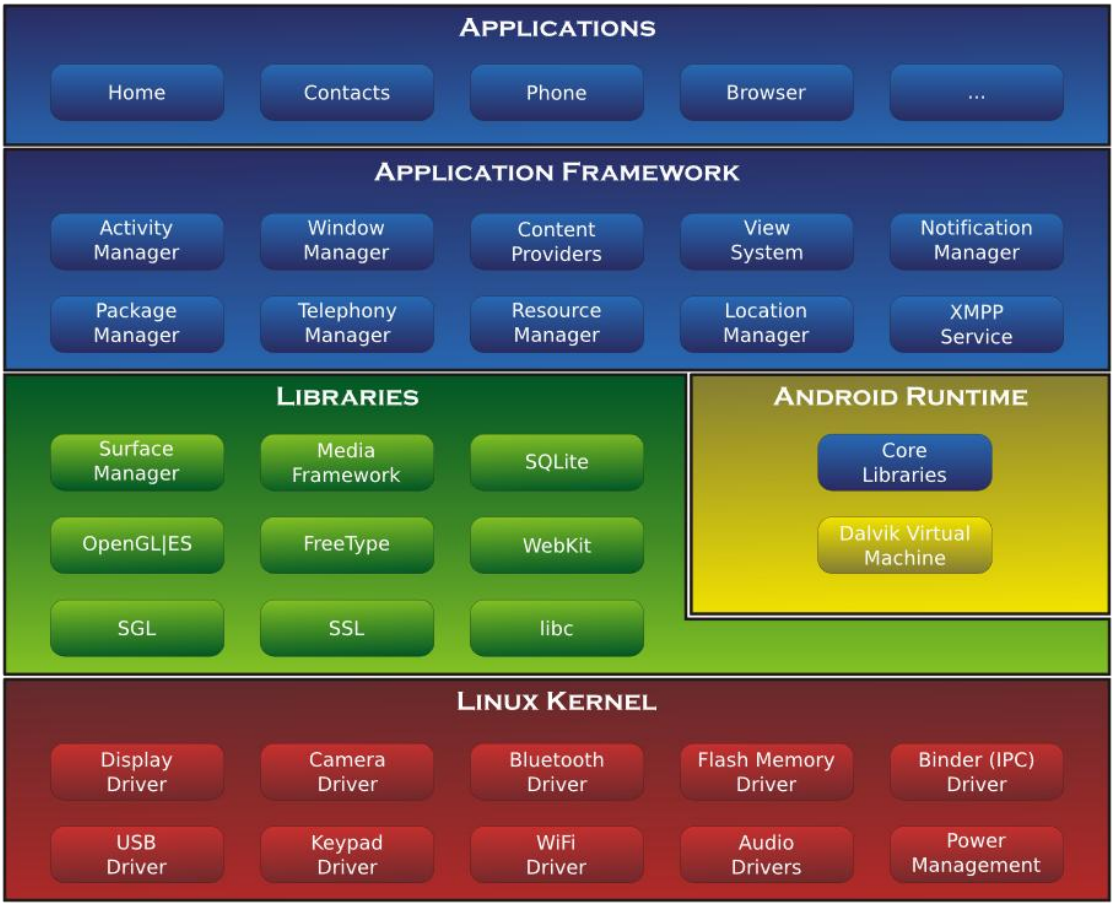
Το επίπεδο API χρησιμοποιείτε για να:

- Επιτρέψει στις εφαρμογές να δηλώσουν ποιο είναι το ελάχιστο επίπεδο που απαιτούν.
- Δηλώσουν ποιο είναι το επίπεδο για το οποίο είναι ακριβώς σχεδιασμένες, ώστε να εκμεταλλευτούν τις δυνατότητες της νέας έκδοσης.
- Επιτρέψει στο σύστημα να ελέγχει την εγκατάσταση εφαρμογών στη συσκευή του χρήστη έτσι ώστε να μην εγκαθίστανται ασύμβατες εφαρμογές.

1.4 Η Αρχιτεκτονική του συστήματος

Το Android είναι μια πλατφόρμα, μια "στοίβα λογισμικού" (software stack), που αποτελείτε από το λειτουργικό σύστημα στη βάση του, το ενδιάμεσο λογισμικό(middleware) και τις βασικές εφαρμογές. Το Android SDK παρέχει τα εργαλεία και APIs τα οποία είναι απαραίτητα για την ανάπτυξη εφαρμογών για την πλατφόρμα Android χρησιμοποιώντας τη γλώσσα προγραμματισμού Java.

Παρακάτω θα παρουσιάσουμε την αρχιτεκτονική του. Τα διάφορα επίπεδα από τα οποία απαρτίζεται και το πώς αυτά αλληλεπιδρούν μεταξύ τους καθώς και τις τεχνολογίες που χρησιμοποιούνται για την υλοποίηση τους. Όπως είπαμε το android είναι ένα ενσωματωμένο(embedded) ΛΣ βασισμένο στον πυρήνα του Linux δεν είναι όμως embedded έκδοση του Linux. Οι περισσότερες εφαρμογές αναπτύσσονται με τη χρήση της γλώσσας προγραμματισμού Java δεν είναι όμως Java εφαρμογές, δεν υποστηρίζονται ολοκληρωμένα πλαίσια όπως Java Standard Edition ή Micro Edition.



Εικόνα 3: Η αρχιτεκτονική του Android

1.4.1 Πυρήνας Linux(Linux Kernel)

Στη βάση του συστήματος βρίσκεται ο πυρήνας του Linux. Αρχικά χρησιμοποιήθηκε η σειρά 2.6 αλλά πλέον, από την έκδοση 4.0 και μετά, έχει αναβαθμιστεί στη σειρά 3.0. Έχουν γίνει όμως αλλαγές πέρα από την τυπική ανάπτυξη του πυρήνα. Το X Window System δεν περιλαμβάνεται και δεν υποστηρίζεται το πλήρες σετ των βιβλιοθηκών GNU, αυτό κάνει δύσκολη τη μεταφορά εφαρμογών ή βιβλιοθηκών από το Linux στο Android. Περεταίρω αλλαγές, όπως ένα νέο σύστημα διαχείρισης ενέργειας, προστέθηκαν για να καλύψουν ιδιαίτερες ανάγκες. Ο πυρήνας είναι αυτός που αλληλεπιδρά με το υλικό της συσκευής και διαχειρίζεται το σύστημα αρχείων. Περιλαμβάνει όλους τους οδηγούς όπως Bluetooth και Wi-Fi καθώς και ιδιόκτητους οδηγούς για το υλικό του κάθε κατασκευαστή το οποίο και αναπτύσσεται από τον ίδιο. Λειτουργεί σαν επίπεδο αφαίρεσης μεταξύ του υλικού και του υπόλοιπου λογισμικού. Αυτή ακριβώς η χρήση του πυρήνα Linux το οποίο είναι ανοιχτό λογισμικό καθιστά ευκολότερη τη χρήση του ΛΣ σε διαφορετικές συσκευές.

1.4.2 Εγγενείς Βιβλιοθήκες(Native Libraries)

Το επόμενο επίπεδο είναι οι εγγενείς βιβλιοθήκες(native libraries) οι οποίες χρησιμοποιούνται από διάφορα υποσυστήματα του Android και επιτρέπουν στη συσκευή να διαχειρίζεται τα διάφορα είδη δεδομένων. Αυτές είναι γραμμένες σε C/C++. Οι δυνατότητες που προσφέρουν γίνονται διαθέσιμες στους προγραμματιστές μέσω του πλαισίου εφαρμογών(framework) Android. Μερικές από τις σημαντικότερες είναι:

- **Surface Manager**: Διαχειρίζεται την πρόσβαση στο υποσύστημα οθόνης και συνθέτει 2D και 3D επιφάνειες γραφικών από πολλαπλές εφαρμογές.

Εικόνα 4: Διαχειριστής επιφάνειας

- **Media framework:** Παρέχει τους διάφορους κωδικοποιητές/αποκωδικοποιητές για την αναπαραγωγή και εγγραφή των διάφορων μορφών ήχου και εικόνας, συμπεριλαμβανομένων των MPEG4, H.264, MP3, AAC, AMR, JPG, και PNG. Βασισμένο στην πλατφόρμα PacketVideo openCore.
- **SQLite**: Ένα απλό, ελαφρύ και ισχυρό σχεσιακό σύστημα διαχείρισης βάσης δεδομένων για την αποθήκευση δεδομένων διαθέσιμο για όλες τις εφαρμογές.
- **WebKit**: Μια μηχανή φυλλομετρητή για την προβολή HTML περιεχομένου. Μπορεί να χρησιμοποιηθεί και σαν ένα πλαίσιο για την ανάπτυξη ενός πλήρους προγράμματος περιήγησης.
- **OpenGL**: Χρησιμοποιείτε για την προβολή δισδιάστατου και τρισδιάστατου περιεχομένου στην οθόνη κάνοντας χρήση του υλικού 3D επιτάχυνσης όταν είναι διαθέσιμο.
- **SGL**: Η βασική 2D μηχανή γραφικών.
- **FreeType**: Bitmap και διανυσματική απεικόνιση γραμματοσειρών.
- **Libc(System C library)**: Μια προσαρμοσμένη έκδοση, βελτιστοποιημένη για χρήση σε embedded συστήματα. Μικρή σε μέγεθος και με ενσωματωμένη υποστήριξη για υπηρεσίες του android. Δεν υποστηρίζει κάποια χαρακτηριστικά POSIX.

1.4.3 Περιβάλλον Χρόνου Εκτέλεσης(Runtime Environment)

Το περιβάλλον χρόνου εκτέλεσης αποτελείται από την Εικονική μηχανή Dalvik και τις Βασικές βιβλιοθήκες Java. Κάνοντας χρήση αυτού του περιβάλλοντος εκτελούνται οι πλειοψηφία των εφαρμογών στο Android.

1.4.3.1 Εικονική Μηχανή Dalvik

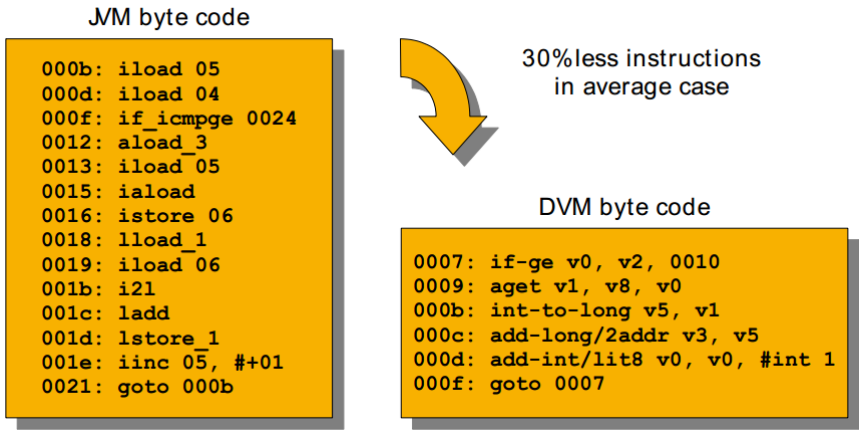
Είναι ένας τύπος εικονικής μηχανής Java που χρησιμοποιείτε για να τρέχει τις εφαρμογές και είναι βελτιστοποιημένη για χαμηλή κατανάλωση και συσκευές με λίγη μνήμη. Σχεδιάστηκε ώστε

να επιτρέπεται η εκτέλεση πολλαπλών στιγμιότυπων της μηχανής. Βασίζεται στο ΛΣ για τη διαχείριση της μνήμης, των νημάτων και των διαδικασιών. Αρχικά είχε μόνο interpreter από την έκδοση 2.2 όμως και μετά χρησιμοποιείτε JIT(Just In Time) compiler. Χρησιμοποίει αρχιτεκτονική βασισμένη σε καταχωρητές(register machine) σε αντίθεση με άλλες JVMs που είναι μηχανές στοίβας(Stack machines). Για αυτούς τους λόγους δεν εκτελεί το στάνταρ java byte code αλλά μετατρέπει τα java .class αρχεία σε δικό του τύπο .dex(Dalvik executables).

- Πολλαπλές κλάσεις συγκεντρώνονται σε ένα μόνο .dex αρχείο για να εξοικονομηθεί χώρος.

Εικόνα 5: Από πηγαίο κώδικα σε .dex

- Αλλάζει το σετ εντολών που χρησιμοποιείτε, τα .dex αρχεία αποτελούνται από λιγότερες εντολές με συνεπεία να έχουν και μικρότερο μέγεθος. Ένα ασυμπίεστο .dex αρχείο έχει ελαφρώς μικρότερο μέγεθος από ένα συμπιεσμένο .jar με τα ίδια .class αρχεία.



Εικόνα 6: Πλήθος εντολών byte codes

- Από τις βιβλιοθήκες που έχουμε εισάγει μόνο οι κλάσεις που χρησιμοποιούμε θα συμπεριληφθούν διατηρώντας έτσι μικρότερο το μέγεθος.
- Τα αρχεία αυτά μπορεί να τροποποιηθούν περεταίρω κατά την εγκατάσταση της εφαρμογής ώστε να βελτιστοποιηθούν με βάση το υλικό στο οποίο εγκαθίστανται.

1.4.3.2 Εικονική μηχανή ART

Ένας νέος τύπος περιβάλλοντος χρόνου εκτέλεσης έκανε την εμφάνιση του με την κυκλοφορία της έκδοσης 4.4 KitKat. Η νέα αυτή μηχανή βρίσκεται σε πειραματικό στάδιο και δεν χρησιμοποιείτε εξ ορισμού. Χρησιμοποιεί Ahead Of Time(AOT) compilation για να κάνει pre compile σε γλώσσα μηχανής τις εφαρμογές κατά την εγκατάσταση τους. Αυτό βελτιώνει το χρόνο εκτέλεσης αλλά αυξάνει το χρόνο που απαιτείται για την εγκατάσταση καθώς και τον απαραίτητο αποθηκευτικό χώρο.

1.4.4 Βασικές βιβλιοθήκες(Core libraries)

Είναι οι βασικές βιβλιοθήκες που παρέχουν το μεγαλύτερο μέρος της λειτουργικότητας της Java SE. Κάθε συσκευή θα πρέπει να υποστηρίζει αυτές τις λειτουργίες. Βασίζεται σε ένα υποσύνολο

του Apache Harmony, μιας ανοιχτού κώδικα υλοποίησης της Java με προσθήκες και τροποποιήσεις από την Google. Αποτελείτε από 3 κύρια μέρη:

1. Βιβλιοθήκες ειδικά για την Dalvik VM(dalvik.*)
2. Κλάσεις Java, περιλαμβάνει βασικές και βοηθητικές(java.*, javax.*)
3. Βοηθητικές βιβλιοθήκες από τρίτους, όπως Apache HttpClient 4(org.apache.http.*)

1.4.5 Πλαίσιο Εφαρμογών(Application Framework)

Παρέχοντας μια ανοιχτή πλατφόρμα ανάπτυξης το Android προσφέρει στους προγραμματιστές την ικανότητα να αναπτύσσουν εξαιρετικά πλούσιες και καινοτόμες εφαρμογές. Οι προγραμματιστές είναι ελεύθεροι να επωφεληθούν από το υλικό της συσκευής, τις πληροφορίες τοποθεσίας πρόσβασης(GPS), να εκτελέσουν υπηρεσίες στο παρασκήνιο, να προσθέσουν ειδοποιήσεις στην γραμμή κατάστασης και πολλά άλλα.

Οι προγραμματιστές έχουν πλήρη πρόσβαση στα ίδια APIs που χρησιμοποιούνται από τις βασικές εφαρμογές. Η αρχιτεκτονική έχει σχεδιαστεί για να απλοποιήσει την επαναχρησιμοποίηση των συστατικών μερών(components). Οποιαδήποτε εφαρμογή μπορεί να δημοσιεύσει τις δυνατότητές της και οποιαδήποτε άλλη εφαρμογή μπορεί να κάνει τότε χρήση αυτών των δυνατοτήτων (που υπόκεινται σε περιορισμούς ασφαλείας που επιβάλλονται από το πλαίσιο). Αυτός ο ίδιος μηχανισμός επιτρέπει αυτά τα συστατικά να αντικατασταθούν από το χρήστη.

Πίσω από όλες τις εφαρμογές είναι ένα σύνολο υπηρεσιών και συστημάτων, όπως οι εξής:

- **Activity Manager**: Διαχειρίζεται τον κύκλο ζωής των εφαρμογών και παρέχει μια κοινή δυνατότητα πλοήγησης.
- **View System**: Ένα πλούσιο και επεκτάσιμο σύνολο Views που μπορούν να χρησιμοποιηθούν για τη δημιουργία μιας εφαρμογής, συμπεριλαμβανομένων των πινάκων, πλαίσια κειμένου, κουμπιά, και ακόμη και ενός ενσωματωμένου προγράμματος περιήγησης.
- **Content Providers**: Επιτρέπουν στις εφαρμογές πρόσβαση άλλων εφαρμογών, πχ Επαφές, ή να μοιραστούν δικά τους δεδομένα.
- **Resource Manager**: Παρέχει πρόσβαση σε μη-εκτελέσημους πόρους όπως γραφικά και αρχεία διάταξης.
- **Notification Manager**: Επιτρέπει σε όλες τις εφαρμογές να προβάλουν ειδοποιήσεις στη μπάρα κατάστασης.

1.4.6 Εφαρμογές

Το τελευταίο επίπεδο της "στοίβας" είναι οι εφαρμογές, εδώ ανήκουν τόσο οι εφαρμογές που έρχονται προ-εγκατεστημενες στο λειτουργικό όσο και αυτές που αναπτύσσονται από χρήστες και διανέμονται μέσω των καταστημάτων εφαρμογών και του ιντερνέτ.

Το Android κυκλοφορεί πάντα με ένα σύνολο από βασικές εφαρμογές συμπεριλαμβανομένου μιας εφαρμογής ηλεκτρονικού ταχυδρομείου, πρόγραμμα SMS, ημερολόγιο, χάρτες, πρόγραμμα περιήγησης, επαφές, και άλλα. Όλες αυτές οι εφαρμογές έχουν γραφτεί με τη χρήση της γλώσσας προγραμματισμού Java.

1.5 Τα θεμελιώδη των εφαρμογών

Το Android SDK περιλαμβάνει όλα τα εργαλεία για τη μεταγλώττιση του κώδικα και τη συγκέντρωση του μαζί με όλα τα δεδομένα και τα αρχεία των πόρων σε ένα πακέτο Android, ένα συμπιεσμένο αρχείο με την κατάληξη .apk. Όλος ο κώδικας σε ένα ενιαίο αρχείο .apk θεωρείται ότι είναι μία εφαρμογή και το αρχείο χρησιμοποιείτε από τις συσκευές για να εγκαταστήσουν την εφαρμογή.

Με τον τρόπο αυτό, το σύστημα εφαρμόζει την "αρχή των ελάχιστων προνομίων". Δηλαδή, κάθε εφαρμογή, εξ ορισμού, έχει πρόσβαση μόνο στα στοιχεία που απαιτούνται για να κάνει τη δουλειά του και τίποτα περισσότερο. Αυτό δημιουργεί ένα πολύ ασφαλές περιβάλλον στο οποίο μια εφαρμογή δεν μπορεί να έχει πρόσβαση στα μέρη του συστήματος για τα οποία δεν έχει δοθεί η άδεια.

Ωστόσο, υπάρχουν τρόποι για τις εφαρμογές να μοιράζονται δεδομένα μεταξύ τους και για να έχουν πρόσβαση στις υπηρεσίες του συστήματος:

- Είναι δυνατόν για δύο εφαρμογές να μοιράζονται το ίδιο user ID, οπότε είναι σε θέση να έχουν πρόσβαση στα αρχεία του άλλου. Για να εξοικονόμηση πόρων συστήματος εφαρμογές με το ίδιο όνομα χρήστη μπορεί να τρέξουν με την ίδια διαδικασία και μοιράζονται το ίδιο VM (οι εφαρμογές θα πρέπει να υπογραφούν επίσης με το ίδιο πιστοποιητικό).
- Η εφαρμογή μπορεί να ζητήσει την άδεια να έχει πρόσβαση στα δεδομένα της συσκευής, όπως επαφές του χρήστη, τα μηνύματα SMS, την κάρτα SD, φωτογραφική μηχανή, Bluetooth και πολλά άλλα. Όλα τα δικαιώματα της εφαρμογής θα πρέπει να γίνουν δεκτά από το χρήστη κατά την εγκατάσταση.

1.5.1 Συστατικά Εφαρμογών(Application Components)

Τα συστατικά εφαρμογών είναι τα βασικά δομικά στοιχεία μιας εφαρμογής. Κάθε στοιχείο είναι ένα διαφορετικό σημείο μέσω του οποίου το σύστημα μπορεί να εκκινήσει την εφαρμογή. Δεν είναι όλα τα στοιχεία σημεία εισόδου για το χρήστη και ορισμένα εξαρτώνται από κάποιο άλλο, το καθένα όμως υπάρχει ως μια ξεχωριστή οντότητα και παίζει ένα συγκεκριμένο ρόλο και βοηθά να καθορίσει τη συνολική συμπεριφορά της εφαρμογής.

Υπάρχουν τέσσερις διαφορετικοί τύποι συστατικών εφαρμογής. Κάθε τύπος εξυπηρετεί ένα ξεχωριστό σκοπό και έχει ένα ξεχωριστό κύκλο ζωής που ορίζει τον τρόπο που δημιουργείται και καταστρέφεται.

Αυτοί είναι οι τέσσερις τύποι συστατικών μιας εφαρμογής:

- **Activities**: Μια δραστηριότητα αντιπροσωπεύει μια μοναδική οθόνη με μια διεπαφή χρήστη. Για παράδειγμα, μια εφαρμογή ηλεκτρονικού ταχυδρομείου μπορεί να έχει μία δραστηριότητα που δείχνει μια λίστα με τα νέα μηνύματα ηλεκτρονικού ταχυδρομείου, μια άλλη δραστηριότητα για τη σύνθεση νέου μηνύματος, καθώς και άλλη δραστηριότητα για την ανάγνωση των μηνυμάτων. Αν και οι δραστηριότητες λειτουργούν μαζί για να σχηματίσουν μια συνολική εμπειρία χρήστη κάθε μία είναι ανεξάρτητη από τις άλλες. Ως εκ τούτου μια άλλη εφαρμογή μπορεί να ξεκινήσει οποιαδήποτε από τις δραστηριότητες αυτές. Για παράδειγμα η εφαρμογή της κάμερας

μπορεί να εκκινήσει τη δραστηριότητα που συνθέτει νέο e-mail έτσι ώστε ο χρήστης να αποστείλει μια εικόνα.

- **Services**: Μια υπηρεσία είναι ένα στοιχείο που τρέχει στο παρασκήνιο για να εκτελέσει μακροχρόνιες λειτουργίες ή να εκτελέσει εργασίες για απομακρυσμένες διαδικασίες. Μια υπηρεσία δεν παρέχει διεπαφή χρήστη. Για παράδειγμα, μια υπηρεσία μπορεί να παίξει μουσική στο παρασκήνιο ενώ ο χρήστης βρίσκεται σε διαφορετική εφαρμογή, ή θα μπορούσε να φέρει τα δεδομένα μέσω του δικτύου χωρίς να εμποδίζει την αλληλεπίδραση του χρήστη με μια άλλη δραστηριότητα. Ένα άλλο στοιχείο, όπως μια δραστηριότητα, μπορεί να εκκινήσει μια υπηρεσία και να αλληλεπιδράσει μαζί της.
- **Content providers**: Ένας πάροχος περιεχομένου διαχειρίζεται ένα κοινό σύνολο δεδομένων μιας εφαρμογής. Μπορούμε να αποθηκεύσουμε τα δεδομένα στο σύστημα αρχείων, μια βάση δεδομένων SQLite, στο διαδίκτυο, ή οποιαδήποτε άλλη θέση αποθήκευσης μπορεί να έχει πρόσβαση η εφαρμογή. Μέσα από τον πάροχο περιεχομένου άλλες εφαρμογές μπορεί να ανακτήσουν ή ακόμα και να τροποποιήσουν τα δεδομένα (αν ο πάροχος περιεχομένου το επιτρέπει). Για παράδειγμα το σύστημα Android παρέχει μια υπηρεσία παροχής περιεχομένου που διαχειρίζεται τα στοιχεία επικοινωνίας του χρήστη. Ως εκ τούτου, κάθε εφαρμογή με τα κατάλληλα δικαιώματα μπορεί να υποβάλει ερωτήματα στον παρόχο περιεχομένου ώστε να διαβάσει και να γράψει πληροφορίες σχετικές με ένα συγκεκριμένο άτομο.
- **Broadcast receivers:** Ένας δέκτης εκπομπής είναι ένα στοιχείο που ανταποκρίνεται στη μετάδοση προθέσεων(`Intent listeners`). Πολλές εκπομπές προέρχονται από το σύστημα για παράδειγμα, μια εκπομπή που ανακοινώνει ότι η οθόνη έχει απενεργοποιηθεί ή η μπαταρία είναι χαμηλή. Οι εφαρμογές μπορούν επίσης να εκκινήσουν εκπομπές για παράδειγμα, για να επιτρέψουν σε άλλες εφαρμογές να γνωρίζουν ότι κάποια δεδομένα έχουν ληφθεί και είναι διαθέσιμα για χρήση. Αν και οι `Broadcast receivers` δεν εμφανίζουν μια διεπαφή χρήστη μπορούν να δημιουργήσουν μια ειδοποίηση στη γραμμή κατάστασης που να ειδοποιεί το χρήστη για ένα "συμβάν εκπομπής". Πιο συχνά όμως ένας δέκτης εκπομπής είναι απλά μια "πύλη" για άλλα συστατικά και έχει ως στόχο να κάνει μια πολύ ελάχιστη ποσότητα της εργασίας.

Μια μοναδική πτυχή του σχεδιασμού του συστήματος Android είναι ότι οποιαδήποτε εφαρμογή μπορεί να εκκινήσει συστατικά μιας άλλης εφαρμογής. Για παράδειγμα, αν ο χρήστης θέλει να τραβήξει μια φωτογραφία με την κάμερα της συσκευής, υπάρχει πιθανότατα ήδη μια άλλη εφαρμογή που το κάνει και η εφαρμογή μπορεί να το χρησιμοποιήσει. Δεν χρειάζεται να ενσωματώσει ούτε και να συνδεθεί με τον κώδικα της εφαρμογή της κάμερας. Αντ'αυτού, ξεκινάμε απλά τη δραστηριότητα της εφαρμογή της κάμερας που τραβάει τη φωτογραφία. Όταν ολοκληρωθεί η φωτογραφία θα επιστραφεί στην εφαρμογή μας και για τον χρήστη θα φαίνεται ότι η κάμερα αποτελεί μέρος της εφαρμογής μας.

Όταν το σύστημα ξεκινά ένα συστατικό αρχίζει η διαδικασία(process) για την εφαρμογή, αν δεν είναι ήδη σε λειτουργία, και αρχικοποιεί τις κλάσεις που απαιτούνται. Για παράδειγμα, αν η εφαρμογή μας ξεκινά τη δραστηριότητα στην εφαρμογή της κάμερας που τραβάει τις φωτογραφίες η δραστηριότητα εκτελείται στο πλαίσιο της διαδικασίας που ανήκει στην εφαρμογή της κάμερας, όχι στη διαδικασία της εφαρμογής μας. Έτσι, σε αντίθεση με τις

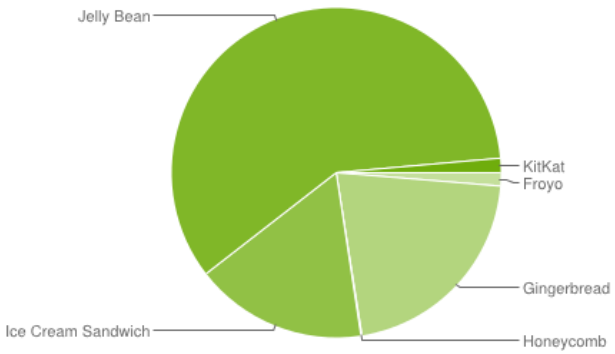
εφαρμογές για τα περισσότερα άλλα συστήματα, οι Android εφαρμογές δεν έχουν ένα ενιαίο σημείο εισόδου (δεν υπάρχει συνάρτηση `main()`, για παράδειγμα).

Επειδή το σύστημα εκτελεί κάθε εφαρμογή σε μια ξεχωριστή διαδικασία με άδειες αρχείων που περιορίζουν την πρόσβαση σε άλλες εφαρμογές, η εφαρμογή μας δεν μπορεί να ενεργοποιήσει άμεσα ένα component από μια άλλη εφαρμογή, το σύστημα Android όμως μπορεί. Έτσι, για να ενεργοποιήσουμε ένα component σε κάποια άλλη εφαρμογή θα πρέπει να παραδώσουμε ένα μήνυμα στο σύστημα που καθορίζει την πρόθεσή να ξεκινήσουμε ένα συγκεκριμένο component. Το σύστημα ενεργοποιεί στη συνέχεια το στοιχείο.

1.6 Υποστήριξη διαφορετικών εκδόσεων

Παρόλο που οι νέες εκδόσεις εισάγουν πολλές νέες δυνατότητες και βελτιώνουν τις υπάρχουσες η εφαρμογή μας θα πρέπει να συνεχίζει να υποστηρίζει τις παλιότερες. Όχι μόνο για να μη δυσαρεστήσουμε τους υπάρχοντες χρήστες αλλά και για να μην αποτρέψουμε νέους οι οποίοι χρησιμοποιούν ακόμα κάποια παλιότερη έκδοση. Οι εκδόσεις παλιότερες της 4.0, που ενοποίησε αρκετά στοιχεία και έθεσε τις βάσεις για μια συγκεκριμένη πορεία ανάπτυξης, διατηρούν ακόμα πάνω από το 20% της αγοράς.

Version	Codename	API	Distribution
2.2	Froyo	8	1.3%
2.3.3 - 2.3.7	Gingerbread	10	21.2%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	16.9%
4.1.x	Jelly Bean	16	35.9%
4.2.x		17	15.4%
4.3		18	7.8%
4.4	KitKat	19	1.4%



Εικόνα 7: Κατανομή εκδόσεων

Προκειμένου να παρέχουμε τα νεότερα χαρακτηριστικά και λειτουργικότητα και να διατηρήσουμε τη συμβατότητα σε διάφορες εκδόσεις του Android μπορούμε να κάνουμε τα εξής:

- Θα πρέπει να χρησιμοποιήσουμε τη Βιβλιοθήκη Υποστήριξης Android(Android Support Library) η οποία μας επιτρέπει να χρησιμοποιούμε τα περισσότερα από τα πιο πρόσφατα APIs σε παλαιότερες εκδόσεις.
- Να ελέγχουμε στον κώδικα μας την έκδοση κατά τον χρόνο εκτέλεσης και να χρησιμοποιούμε τις λειτουργίες μόνο όταν αυτές είναι διαθέσιμες. Τα XML στοιχεία που δεν υποστηρίζονται απλώς θα αγνοηθούν από το σύστημα.
- Να χρησιμοποιήσουμε διαφορετικούς πόρους για την εφαρμογή μας ανάλογα με την έκδοση. Περισσότερα για τους πόρους της εφαρμογής θα δούμε παρακάτω

Η εφαρμογή μας σχεδιάστηκε ώστε να υποστηρίζει όλες τις εκδόσεις μετά την 2.3(API 9). Αυτή παρέχει αρκετές από τις λειτουργίες που χρειαζόμαστε και η βιβλιοθήκη υποστήριξης καλύπτει τις περισσότερες από τις νεότερες.

1.7 Ασφάλεια στο Android

Από τα αρχικά στάδια ανάπτυξης του Android αναγνωρίστηκε η ανάγκη για ένα ισχυρό μοντέλο ασφάλειας το οποίο θα επέτρεπε την ανάπτυξη ενός δυναμικού οικοσυστήματος συσκευών και εφαρμογών καθώς και για την υποστήριξη υπηρεσιών νέφους(cloud services). Καθ' όλη τη διάρκεια λοιπόν του κύκλου ανάπτυξης της πλατφόρμας εφαρμόζεται ένα πρόγραμμα ασφαλείας το οποίο περιλαμβάνει:

- Αναθεώρηση σχεδιασμού: Η διαδικασία ασφαλείας αρχίζει νωρίς στον κύκλο ανάπτυξης με τη δημιουργία ενός πλούσιου και ενός παραμετροποιήσιμου μοντέλου ασφάλειας και σχεδιασμού.
- Δοκιμές διείσδυσης και αναθεώρηση κώδικα: Κατά την ανάπτυξη της πλατφόρμας, τμήματα του Android καθώς και ανοιχτού κώδικα υπόκεινται σε έντονους ελέγχους ασφαλείας. Οι έλεγχοι αυτοί εκτελούνται τόσο από τμήματα ασφάλειας της Google όσο και από ανεξάρτητους συμβούλους ασφαλείας.
- Ανασκόπηση από μέλη της κοινότητας: Η ανοιχτή φύση της πλατφόρμας επιτρέπει τον ευρύ έλεγχο από κάθε ενδιαφερόμενο.
- Απόκριση συμβάντος: Ακόμη και με όλες αυτές τις προφυλάξεις, θέματα ασφάλειας μπορούν να προκύψουν μετά τη κυκλοφορία αμέσως μετά την ανακάλυψη των οποίων λαμβάνονται μέτρα για την αντιμετώπιση τους.

1.7.1 Αρχιτεκτονική ασφαλείας

Το Android επιδιώκει να είναι όσο πιο ασφαλές παραμένοντας ένα εύχρηστο λειτουργικό σύστημα για κινητές πλατφόρμες με αναπροσανατολισμό των παραδοσιακών ελέγχων ασφαλείας του λειτουργικού συστήματος για να προστατέψει τα δεδομένα των χρηστών και τους πόρους του συστήματος.

Για την επίτευξη αυτών των στόχων, το Android παρέχει αυτά τα βασικά χαρακτηριστικά ασφαλείας:

- Ισχυρή ασφάλεια σε επίπεδο ΛΣ μέσω του πυρήνα Linux
- Υποχρεωτική εφαρμογή sandbox(απομόνωση) για όλες τις εφαρμογές
- Ασφαλής επικοινωνία μεταξύ διεργασιών
- Ψηφιακά υπογεγραμμένες εφαρμογές
- Δικαιώματα που ορίζονται από την εφαρμογή και χορηγούνται από το χρήστη.

Ασφάλεια πυρήνα Linux

Όπως είπαμε η βάση του ΛΣ είναι ο πυρήνας Linux. Βρίσκεται σε ευρεία χρήση για πολλά χρόνια, και χρησιμοποιείται σε εκατομμύρια ευαίσθητα περιβάλλοντα. Έχει γίνει έτσι πολύ σταθερός και ασφαλής και θεωρείτε αξιόπιστος από πολλές εταιρίες και επαγγελματίες στο χώρο της ασφάλειας.

Τα βασικά χαρακτηριστικά ασφαλείας είναι τα εξής:

- Χρησιμοποιεί μοντέλο δικαιωμάτων με βάση το χρήστη
- Μηχανισμός απομόνωσης διαδικασιών
- Επεκτάσιμος μηχανισμός για την ασφαλή επικοινωνία μεταξύ διεργασιών(IPC)
- Η ικανότητα να αφαιρούνται περιττά και ενδεχομένως ανασφαλή τμήματα του πυρήνα

Απομόνωση εφαρμογών(Application sandbox)

Η πλατφόρμα εκμεταλλεύεται την προστασία βάση-χρηστών του Linux ως ένα μέσο για τον εντοπισμό και την απομόνωση των εφαρμογών. Το σύστημα εκχωρεί ένα μοναδικό αναγνωριστικό χρήστη (UID) για κάθε εφαρμογή και την εκτελεί σαν αυτόν το χρήστη σε μια ξεχωριστή διαδικασία. Αυτή η προσέγγιση είναι διαφορετική από τα άλλα λειτουργικά συστήματα (συμπεριλαμβανομένης και της παραδοσιακής διαμόρφωση του Linux), όπου πολλαπλές εφαρμογές τρέχουν με τα ίδια δικαιώματα χρήστη.

Έτσι επιτυγχάνεται η απομόνωση των εφαρμογών(sandbox). Ο πυρήνας επιβάλλει την ασφάλεια μεταξύ των εφαρμογών και του συστήματος στο επίπεδο της διαδικασίας μέσω των αναγνωριστικών που έχουν εκχωρηθεί στις εφαρμογές. Εξ ορισμού, οι εφαρμογές δεν μπορούν να αλληλεπιδράσουν μεταξύ τους και έχουν περιορισμένη πρόσβαση στο λειτουργικό σύστημα.

Δεδομένου ότι η απομόνωση εφαρμογών γίνεται στο επίπεδο του πυρήνα αυτό το μοντέλο ασφαλείας εφαρμόζεται και στον εγγενή κώδικα και στις εφαρμογές του ΛΣ. Η απομόνωση αυτή περιορίζει και τυχόν σφάλματα μνήμης(memory corruption) τα οποία θέτουν σε κίνδυνο την ασφάλεια της συσκευής.

Όπως όλα τα χαρακτηριστικά ασφαλείας έτσι και το sandbox δεν είναι απαραβίαστο. Παρόλα αυτά σε κατάλληλα διαμορφωμένη συσκευή θα πρέπει πρώτα να παραβιαστεί η ίδια η ασφάλεια του πυρήνα του Linux.

Διαμέρισμα συστήματος

Το διαμέρισμα του συστήματος περιέχει τον πυρήνα του Android, καθώς και τις βιβλιοθήκες του λειτουργικού συστήματος, το περιβάλλον χρόνου εκτέλεσης, το πλαίσιο εφαρμογών, και τις βασικές εφαρμογές. Αυτό το διαμέρισμα έχει οριστεί μόνο για ανάγνωση και εξασφαλίζει ότι ο χρήστης μπορεί να εκκινήσει το τηλέφωνό του σε ένα περιβάλλον που είναι ελεύθερο από λογισμικό τρίτων.

Δικαιώματα αρχείων συστήματος

Σε ένα περιβάλλον τύπου UNIX τα δικαιώματα αρχείων εξασφαλίζουν ότι ένας χρήστης δεν μπορεί να αλλάξει ή να διαβάσει αρχεία άλλου χρήστη. Στην περίπτωση του Android, κάθε εφαρμογή τρέχει ως ένας διαφορετικός χρήστης. Εκτός αν ο δημιουργός της εφαρμογής επιτρέψει ρητά την χρήση αρχείων σε άλλες εφαρμογές τα αρχεία που δημιουργούνται από μια εφαρμογή δεν μπορούν να διαβαστούν ή να τροποποιηθούν από κάποια άλλη.

Κρυπτογράφηση

Το Android παρέχει ένα σύνολο κρυπτογραφικών APIs για χρήση από εφαρμογές. Αυτές περιλαμβάνουν υλοποιήσεις τυποποιημένων και πολυχρησιμοποιημένων κρυπτογραφικών εργαλείων όπως AES, RSA, DSA και SHA. Επιπλέον, τα APIs παρέχουν πρωτόκολλα υψηλότερου επιπέδου, όπως SSL και HTTPS.

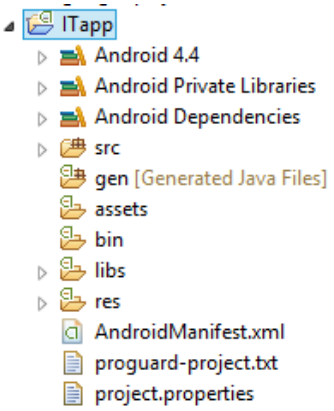
Κεφάλαιο 2: Βασικά στοιχεία εφαρμογών

Παραπάνω είδαμε επιγραμματικά τα βασικά δομικά στοιχεία μιας εφαρμογής τώρα θα δούμε κάποια βασικά στοιχεία τα οποία είναι κοινά σε όλες οι εφαρμογές και πως μπορούμε να τα

χρησιμοποιήσουμε για να βελτιώσουμε την διαδικασία της ανάπτυξης της εφαρμογής αλλά και κυρίως την εμπειρία χρήσης της.

2.1 Κατάλογος έργου εφαρμογής

Κάθε έργο(project) για την ανάπτυξη μιας εφαρμογής Android ακολουθεί μια συγκεκριμένη διαδικασία που απαιτεί μια συγκεκριμένη ιεραρχία αρχείων. Αποτελείτε από:



Εικόνα 8: Ιεραρχία έργου

- Το αρχείο AndroidManifest.xml, το οποίο περιλαμβάνει βασικές πληροφορίες για την εφαρμογή μας.
- Κατάλογος `src`, ο οποίος περιέχει τον κώδικα της εφαρμογής μας. Τα αρχεία με τον κώδικα Java όλων των Activities, Services κλπ, δομημένα σε πακέτα.
- Κατάλογος `res`, εδώ μπαίνουν όλοι οι πόροι της εφαρμογής που δεν έχουν να κάνουν με τον εκτελέσιμο κώδικα της εφαρμογής. Αρχεία όπως εικόνες, layouts των δραστηριοτήτων κλπ για τα οποία θα δημιουργηθεί αυτομάτως ένα αναγνωριστικό για εύκολη πρόσβαση στο κώδικα μας μέσω της κλάσης `R`.
- Κατάλογος `assets`, και εδώ μπορούν να τοποθετηθούν διάφοροι τύποι αρχείων χωρίς όμως να δημιουργηθούν αναγνωριστικά οπότε η πρόσβαση στο κώδικα γίνεται με βάση το όνομα του αρχείου και μπορούμε να έχουμε μια ιεραρχία αρχείων.
- Άλλοι κατάλογοι και αρχεία που δημιουργούνται από το περιβάλλον ανάπτυξης που χρησιμοποιούμε.

2.2 Το αρχείο AndroidManifest.xml

Κάθε εφαρμογή Android πρέπει να περιέχει το αρχείο manifest, πάντα με το ακριβές όνομα AndroidManifest.xml, στη ρίζα της ιεραρχίας του έργου μας. Καθορίζει τη δομή, τα μεταδεδομένα, τα συστατικά και τις απαιτήσεις της εφαρμογής μας, πληροφορίες που το σύστημα πρέπει να έχει για να μπορέσει να τρέξει οποιοδήποτε κώδικα της εφαρμογής. Είναι υπεύθυνο μεταξύ άλλων για να:

- Ορίζει το όνομα του πακέτου Java για την εφαρμογή. Το όνομα λειτουργεί ως ένα μοναδικό αναγνωριστικό για την εφαρμογή.
- Περιγράφει τα συστατικά της εφαρμογής τις δραστηριότητες, τις υπηρεσίες, κλπ από τα οποία αποτελείτε. Είναι τα ονόματα των κλάσεων που υλοποιούν καθένα από τα συστατικά και δημοσιεύει τις δυνατότητές τους για παράδειγμα, ποιές «προθέσεις» μπορούν να χειριστούν. Οι δηλώσεις αυτές επιτρέπουν στο σύστημα να γνωρίζει ποια είναι τα συστατικά και κάτω από ποιες συνθήκες μπορεί να τα εκτελέσει.
- Καθορίζει τις διαδικασίες που θα φιλοξενήσουν τα στοιχεία της εφαρμογής.
- Δηλώνει ποια δικαιώματα απαιτεί η εφαρμογή προκειμένου να αποκτήσει πρόσβαση σε προστατευμένα μέρη του API και να αλληλεπιδράσει με άλλες εφαρμογές.
- Δηλώνει επίσης τα δικαιώματα που οι άλλοι πρέπει να έχουν για να αλληλεπιδράσουν με τα στοιχεία της εφαρμογής.
- Δηλώνει το ελάχιστο επίπεδο του API του Android που απαιτεί η εφαρμογή.

- Παραθέτει τις βιβλιοθήκες με τις οποίες πρέπει να συνδέεται η εφαρμογή.
- Προσδιορίζει τα μεταδεδομένα της εφαρμογή όπως το επίσημο όνομα της εφαρμογής, εικονίδιό της, τον αριθμό έκδοσης, και το θέμα.

2.2.1 Δομή του αρχείου Manifest

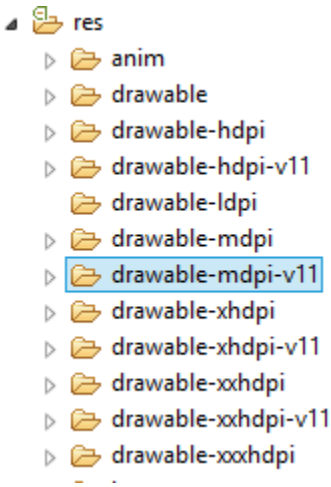
Ο παρακάτω κώδικας δείχνει τη γενική δομή του αρχείου manifest και κάθε στοιχείο που μπορεί να περιέχει. Πολλά από τα στοιχεία αυτά θα τα αναλύσουμε καθώς θα βλέπουμε τα συστατικά με τα οποία συσχετίζονται.

```
<?xml version="1.0" encoding="utf-8"?>

<manifest>

    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <compatible-screens />
    <supports-gl-texture />

    <application>

        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>

        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>

        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>

        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>

        <provider>
            <grant-uri-permission />
            <meta-data />
            <path-permission />
        </provider>
```

```
        <uses-library />

    </application>

</manifest>
```

2.3 Πόροι εφαρμογής

Κατά την ανάπτυξη μιας εφαρμογής Android γίνεται διαχωρισμός του εκτελέσιμου κώδικα από την εμφάνιση, πχ διατάξεις των Activities, γραφικά, σταθερά αλφαριθμητικά κλπ. Αυτό πέρα από το ότι αποτελεί καλή πρακτική καθώς γίνεται ευκολότερη η διατήρηση και η διαχείριση του έργου της εφαρμογής επιτρέπει στο ίδιο το σύστημα Android να διαχειρίζεται τους πόρους, να επιλέγει τα κατάλληλα στοιχεία ανάλογα με τη διαμόρφωση, το υλικό και τις ρυθμίσεις στις οποίες εκτελεί την εφαρμογή μας.



Εικόνα 9: Οι διάφορες κατηγορίες πόρων της εφαρμογής

Οι πόροι της εφαρμογής όπως είπαμε αποθηκεύονται στο κατάλογο `res` στην ιεραρχία του έργου μας. Κάθε ένας από τους διαθέσιμους τύπους πόρων είναι αποθηκευμένος σε υποκατάλογους στους οποίους ομαδοποιούνται ανάλογα με το είδος τους. Περαιτέρω κατηγοριοποίηση γίνεται με βάση τη διαμόρφωση του περιβάλλοντος στο οποίο θέλουμε να χρησιμοποιηθούν οι συγκεκριμένοι πόροι, όπως η γλώσσα ή η έκδοση.

Οι βασικές ομάδες στις οποίες θα κατηγοριοποιηθούν οι πόροι. Για κάθε ομάδα υπάρχει και ένας συγκεκριμένος υποκατάλογος στον κατάλογο `res`:

- `anim`: αρχεία XML που καθορίζουν animations κατά την μετάβαση μεταξύ Activities.
- `color`: αρχεία XML που καθορίζουν μια λίστα χρωμάτων.
- `drawables`: Γραφικά, εικόνες, αρχεία XML που καθορίζουν σχήματα.
- `layout`: αρχεία XML που καθορίζουν τη διάταξη διεπαφών χρήστη.
- `menu`: αρχεία XML που καθορίζουν μενού εφαρμογών, όπως ένα μενού επιλογών ή context menu.
- `values`: αρχεία XML που περιέχουν απλές τιμές, όπως αλφαριθμητικά, ακέραιοι, διαστάσεις, styles. Οι απλές τιμές είναι αποθηκευμένες μέσα σε XML αρχείο στο φάκελο.
    - arrays.xml για πίνακες πόρων(αλφαριθμητικά, ακέραιοι).
    - colors.xml για τις τιμές χρωμάτων.
    - dimens.xml για τις τιμές διαστάσεων.
    - strings.xml για τις τιμές συμβολοσειράς.
    - styles.xml για το στυλ.
- `xml`: Διάφορα αρχεία XML τα οποία χρησιμοποιούνται για τον ορισμό λειτουργιών αλλά δεν ανήκουν στις άλλες κατηγορίες.
- `raw`: Διάφορα αρχεία τα οποία θέλουμε να συμπεριλάβουμε στην εφαρμογή μας.

Περαιτέρω κατηγοριοποίηση γίνεται όταν θέλουμε να ξεχωρίσουμε τους πόρους σε κατηγορίες ανά συγκεκριμένη διαμόρφωση πχ, `drawables-hdpi`, `drawables-xhdpi` για χρήση διαφορετικών γραφικών σε συσκευές με διαφορετική πυκνότητα οθόνης. Το σύστημα θα χρησιμοποιήσει αυτομάτως τους πόρους από τον κατάλληλο κατάλογο. Μπορούν επίσης να μπουν πολλαπλά προσδιοριστικά πχ `drawable-mdpi-v11` το οποίο καθορίζει τόσο την πυκνότητα όσο και την ελάχιστη έκδοση API.

2.3.1 Παραδείγματα ορισμού πόρων

Ορισμός σταθερών αλφαριθμητικών

```
<resources>
      <string name="title_activity_main">Τμήμα Πληροφορικής</string>
      <string name="title_activity_hydra">Υπηρεσίες Ύδρας</string>
      ...
</resources>
```

Ορισμός πινάκα αλφαριθμητικών

```
<resources>
      <string-array name="intervals">
        <item>5 λεπτά</item>
        <item>15 λεπτά</item>
        ...
      </string-array>
</resources>
```

2.3.2 Πρόσβαση στους πόρους

Μέσα στον κώδικα μας έχουμε πρόσβαση στους πόρους μας μέσω της κλάσης `R` η οποία δημιουργείτε από το περιβάλλον ανάπτυξης και βασίζεται στα αρχεία που έχουμε στον κατάλογο `res`. Περιέχει στατικές κλάσεις για κάθε τύπο πόρου που έχουμε προσθέσει. Κάθε μία από τις κλάσεις εντός της R εκθέτει συναφείς πόρους ως μεταβλητές, με τα ονόματα των μεταβλητών να ταιριάζουν με τα αναγνωριστικά πόρων, τα ονόματα δηλαδή που έχουμε ορίσει στα XML αρχεία μας ή το ίδιο το όνομα αρχείου.

Χρήση ενός layout(`res/layout/activity_main.xml`) στο Activity μας

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

Ανάκτηση ενός μηνύματος λάθους(από `res/values/strings.xml`)

```
String msg = getString(R.string.app_error);
```

Μπορούμε επίσης να χρησιμοποιήσουμε τις αναφορές των πόρων, όπως τιμές των χαρακτηριστικών μέσα σε άλλους πόρους. Αυτό είναι ιδιαίτερα χρήσιμο για layouts και styles, επιτρέποντάς μας να δημιουργήσουμε εξειδικευμένες παραλλαγές. Για την πρόσβαση χρησιμοποιούμε το παρακάτω κομμάτι κώδικα.

```
attribute="@[packagename:]resourcetype/resourceidentifier"
```

Εδώ ορίζουμε το style του TextView στο layout μας σε ένα συγκεκριμένο style

```
<TextView
        android:id="@+id/tvTitle"
        style="@style/listTitleTextView"
/>
```

2.4 Κλάση Application

Τη στιγμή που εκτελείτε η εφαρμογή μας δημιουργείτε ένα, και μοναδικό, αντικείμενο της κλάσης Application το οποίο και παραμένει στη μνήμη για όσο χρονικό διάστημα είναι ενεργή η εφαρμογή μας. Αν και δεν είναι τυπικά απαραίτητο μπορούμε να επεκτείνουμε την κλάση ώστε να:

- Χειριστούμε συμβάντα της εφαρμογής, όπως η έναρξη της.
- Μεταφέρουμε αντικείμενα μεταξύ των συστατικών της εφαρμογής(Global objects).
- Διαχειριστούμε τους πόρους που χρησιμοποιούνται από διάφορα στοιχεία της εφαρμογής.
- Παρέχουμε πρόσβαση στο Application `Context`(το γενικό πλαίσιο με το οποίο συσχετίζεται η εφαρμογή) από κλάσεις που δεν έχουν άμεση πρόσβαση σε αντίθεση με τα Activities.

Κλάση Context

Η κλάση Context αποτελεί ένα interface στις γενικές πληροφορίες σχετικά με το περιβάλλον της εφαρμογής. Αυτή είναι μια αφηρημένη κλάση η υλοποίηση της οποίας παρέχεται από το σύστημα Android. Επιτρέπει την πρόσβαση σε συγκεκριμένους πόρους και κλάσεις της εφαρμογής καθώς και κλήσεις για εργασίες σε επίπεδο εφαρμογής όπως η έναρξη των δραστηριοτήτων, μετάδοση και λήψη προθέσεων, κ.λπ.

2.4.1 Υλοποίηση και χρήση της κλάσης

Για να χρησιμοποιήσουμε την δικιά μας υλοποίηση της κλάσης θα πρέπει να το δηλώσουμε στο manifest ορίζοντας το όνομα της κλάσης στο στοιχείο `application` στην ιδιότητα `android:name`.

```
<application
        android:name="com.nkyrim.itapp.ITapp"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        …
</application>
```

Ο παρακάτω κώδικας επεκτείνει την κλάση Application παρέχοντας μας πρόσβαση στο Context και υπερβαίνοντας την `OnCreate` μέθοδο χειριζόμαστε την εκκίνηση της εφαρμογής αρχικοποιώντας τις ρυθμίσεις μας.

```
public class ITapp extends Application {
      private static Context context;

      public static Context getAppContext() {
            return context;
      }

      @Override
      public void onCreate() {
            super.onCreate();
            context = this.getApplicationContext();

            Settings.setupFirstRun();
      }
}
```

2.5 Ενεργοποίηση Συστατικών

Τρεις από τους τέσσερις τύπους συστατικών: δραστηριότητες, υπηρεσίες, δέκτες εκπομπών ενεργοποιούνται από ένα ασύγχρονο μήνυμα που ονομάζεται πρόθεση(`Intent`). Οι προθέσεις συνδέουν μεμονωμένα συστατικά μεταξύ τους κατά το χρόνο εκτέλεσης, είτε το στοιχείο ανήκει στην εφαρμογή μας είτε σε κάποια άλλη.

Η πρόθεση δημιουργείται με ένα αντικείμενο `Intent`, το οποίο καθορίζει ένα μήνυμα για να ενεργοποιήσουμε είτε ένα συγκεκριμένο συστατικό, ένα συγκεκριμένο τύπο συστατικού ή σαν περιγραφή για κάτι που έχει γίνει και τώρα ανακοινώνεται.

- Μπορούμε να ξεκινήσουμε μια δραστηριότητα ή να της δώσουμε κάτι νέο να κάνει με το πέρασμα ενός αντικειμένου `Intent` στην `startActivity()` ή στην `startActivityForResult()` όταν θέλουμε η δραστηριότητα να επιστρέψει ένα αποτέλεσμα.
- Μπορούμε να ξεκινήσουμε μια υπηρεσία ή να δώσουμε νέες οδηγίες σε μία ενεργή υπηρεσία με το πέρασμα μιας πρόθεσης στην `startService()`. Μπορούμε επίσης να συνδεθούμε με την υπηρεσία με το πέρασμα μιας πρόθεσης στην `bindService()`.
- Μπορούμε να ξεκινήσουμε μια εκπομπή με το πέρασμα μιας πρόθεσης στις μεθόδους `sendBroadcast()`, `sendOrderedBroadcast()`, ή `sendStickyBroadcast()`.

Σε κάθε περίπτωση το σύστημα βρει το κατάλληλο συστατικό που θα ανταποκριθεί στην πρόθεση και πάντα τον κατάλληλο τύπο, πχ η `startActivity()` θα εκκινήσει σίγουρα μια δραστηριότητα και όχι υπηρεσία.

2.5.1 Δομή μιας Πρόθεσης

Ένα αντικείμενο `Intent` είναι ένα σύνολο πληροφοριών και αποτελείτε από τα παρακάτω μέρη χωρίς να είναι όμως απαραίτητο να τα περιέχει όλα:

- Όνομα συστατικού: Το όνομα του συστατικού(όνομα κλάσης) που πρέπει να χειριστεί την πρόθεση. Η πρόθεση θα παραδοθεί σε ένα στιγμιότυπο της συγκεκριμένης κλάσης.
- Ενέργεια(action): Μια συμβολοσειρά κατονομάζει την ενέργεια που θα εκτελεστεί ή στην περίπτωση εκπομπής των προθέσεων την ενέργεια που έλαβε χώρα και τώρα γίνεται η αναφορά της. Η κλάση `Intent` ορίζει μια σειρά από σταθερές για συγκεκριμένες ενέργειες μπορούμε όμως να ορίζουμε και δικές μας.
- Δεδομένα(data): Το URI των δεδομένων προς επεξεργασία και ο τύπος MIME των εν λόγω στοιχείων. Οι διάφορες δράσεις σε συνδυασμό με διάφορα είδη των προδιαγραφών των δεδομένων. Διαφορετικά είδη ενεργειών συνδυάζονται με διαφορετικά δεδομένα.
- Κατηγορία(category): Μία συμβολοσειρά που περιέχει πρόσθετες πληροφορίες σχετικά με το είδος του στοιχείου που πρέπει να χειριστεί την πρόθεση. Όπως στις ενέργειες ετσι και εδώ ορίζονται μια σειρά από σταθερές για συγκεκριμένες κατηγορίες.
- Extras: Ζευγάρια κλειδί-τιμή για πρόσθετες πληροφορίες που θα πρέπει να παραδοθούν στο συστατικό που θα χειριστεί την πρόθεση.
- Flags: Flags διαφόρων ειδών, πολλές δίνουν οδηγίες στο σύστημα Android για το πώς να ξεκινήσει μια δραστηριότητα.

Για τις δραστηριότητες και τις υπηρεσίες η πρόθεση καθορίζει τις ενέργειες που θα εκτελέσει (για παράδειγμα, να "δει" ή "να αποστείλει" κάτι) και μπορεί να καθορίσει το URI των δεδομένων πάνω στα οποία ενεργεί. Για παράδειγμα, μια πρόθεση μπορεί να μεταφέρει ένα αίτημα σε μια δραστηριότητα για να δείξει μια εικόνα ή για να ανοίξει μια ιστοσελίδα. Σε ορισμένες περιπτώσεις μπορούμε να εκκινήσουμε μια δραστηριότητα για να λάβουμε ένα αποτέλεσμα στην περίπτωση αυτή η δραστηριότητα επιστρέφει επίσης το αποτέλεσμα σε ένα αντικείμενο πρόθεσης(`Intent`) .

Ο άλλος τύπος συστατικού, ο πάροχος περιεχομένου, δεν ενεργοποιείτε από προθέσεις. Αντίθετα ενεργοποιείται όταν δέχεται ένα αίτημα από έναν `ContentResolver`. Αυτός χειρίζεται όλες τις άμεσες συναλλαγές με τον πάροχο του περιεχομένου έτσι ώστε να μη χρειάζεται να το κάνει το συστατικό και αντ'αυτού καλεί μεθόδους πάνω στο αντικείμενο `ContentResolver`. Μπορούμε να εκτελέσουμε ένα ερώτημα σε έναν πάροχο περιεχομένου καλώντας την `query()` σε έναν `ContentResolver`.

2.5.2 Ανάλυση πρόθεσης

Υπάρχουν δύο κύριες μορφές προθέσεων που χρησιμοποιούνται.

- Ρητές Προθέσεις: Έχουν καθορίσει ένα συστατικό (δηλώνοντας το όνομα), το οποίο παρέχει την ακριβή κλάση που θα εκτελεστεί. Συχνά, αυτές δεν θα περιλαμβάνουν οποιαδήποτε άλλη πληροφορία, απλά είναι ένας τρόπος για μια εφαρμογή να ξεκινήσει διάφορες εσωτερικές δραστηριότητες με τις οποίες ο χρήστης αλληλεπιδρά.
- Έμμεσες Προθέσεις: Αυτές δεν έχουν καθορίσει ένα συγκεκριμένο συστατικό αντ' αυτού πρέπει να περιλαμβάνουν επαρκείς πληροφορίες ώστε το σύστημα να μπορεί να καθορίσει ποια από τα διαθέσιμα συστατικά είναι καλύτερο να εκτελεστεί για αυτό το σκοπό. Αυτά είναι τα συστατικά τα οποία έχουν ορίσει συγκεκριμένα `Intents` τα οποία μπορούν να χειριστούν(`Intent Filters`) είτε στο κώδικα είτε στο manifest ώστε το σύστημα να γνωρίζει για αυτά πριν εκτελεστούν για πρώτη φορά.

Κλάση PendingIntent

Για να περάσουμε μια πρόθεση σε μία άλλη εφαρμογή ή υπηρεσία η οποία θα αναλάβει να την εκτελέσει «εσωκλείουμε» το `Intent` μας σε ένα αντικείμενο `PendingIntent`. Δίνοντας το αντικείμενο αυτό σε μια άλλη εφαρμογή της χορηγούμε το δικαίωμα να εκτελέσει τη λειτουργία που έχουμε ορίσει σαν να ήταν δικιά μας η άλλη εφαρμογή, με τα ίδια δηλαδή δικαιώματα και ταυτότητα.

2.5.3 Παραδείγματα χρήσης

Χρήση ενός ρητού Intent για την εκκίνηση του επόμενου Activity

```
Intent intent = new Intent(this, InfoActivity.class);
startActivity(intent);
```

Προσθήκη Extras για το πέρασμα πληροφοριών στο Activity για τη σύνδεση

```
Intent intent = new Intent(this, LoginActivity.class);
intent.putExtra(LoginActivity.EXTRA_ACCOUNT, accountOption);
startActivityForResult(intent, accountOption);
```

Δημιουργία ενός Intent για την κοινοποίηση μιας ανακοίνωσης

```
Intent shareIntent = new Intent(Intent.ACTION_SEND);
shareIntent.setType("text/plain");
```

```
shareIntent.putExtra(Intent.EXTRA_TEXT, tvText.getText());
```

Intent filter το οποίο ορίζεται για ένα Broadcast Receiver της εφαρμογής μας και «ακούει» το συμβάν εκκίνησης της συσκευής

```
<receiver
   android:name="com.nkyrim.itapp.bcreceiveres.DeviceBootReceiver" >
   <intent-filter>
       <action android:name="android.intent.action.BOOT_COMPLETED" />
   </intent-filter>
</receiver>
```

Κεφάλαιο 3: Δημιουργία διεπιφανειών χρήστη

Το κυριότερο στοιχείο μιας εφαρμογής, ανεξαρτήτως της πλατφόρμας στην οποία εκτελείτε, είναι η αλληλεπίδραση της με το χρήστη. Αυτό επιτυγχάνετε με τη χρήση διεπαφών χρήστη. Στο Android το κύριο συστατικό για τη δημιουργία τους είναι οι Δραστηριότητες(`Activities`). Κάθε `Activity` αποτελείτε από `Views`, που είναι τα διάφορα στοιχεία ελέγχου, `ViewGroups` για τη διάταξη των στοιχείων και `Fragments` τα οποία αντιπροσωπεύουν ένα αρθρωτό τμήμα της δραστηριότητας ενθυλακώνοντας μια λειτουργία ή ένα κομμάτι της διεπαφής.

3.1 Δραστηριότητες(Activities)

Κάθε δραστηριότητα αντιπροσωπεύει μια οθόνη την οποία η εφαρμογή μας θα παρουσιάσει στους χρήστες. Μια εφαρμογή αποτελείται συνήθως από πολλαπλές δραστηριότητες οι οποίες δεν συνδέονται άμεσα μεταξύ τους. Τυπικά μία δραστηριότητα σε μια εφαρμογή καθορίζεται ως η «κύρια» η οποία παρουσιάζεται στο χρήστη κατά την εκκίνηση της εφαρμογής για πρώτη φορά. Όσο πιο περίπλοκη γίνεται η εφαρμογή μας, τόσο περισσότερες οθόνες είναι πιθανό να χρειαστεί.

Κάθε δραστηριότητα μπορεί να ξεκινήσει μια άλλη, προκειμένου να εκτελέσει διάφορες ενέργειες. Κάθε φορά που μια νέα ξεκινά, η προηγούμενη σταματάει, αλλά το σύστημα διατηρεί τη δραστηριότητα σε μια στοίβα (“back stack”). Όταν ξεκινά μια νέα δραστηριότητα ωθείται στην back stack και έρχεται στο προσκήνιο. Η back stack συμμορφώνεται με το βασικό “last in, first out” μηχανισμό της στοίβας έτσι όταν ο χρήστης τελειώνει με την τρέχουσα δραστηριότητα και πιέζει το κουμπί «Πίσω» η τρέχουσα δραστηριότητα βγαίνει από τη στοίβα και καταστρέφεται και η προηγούμενη δραστηριότητα επανέρχεται στο προσκήνιο.

3.1.1 Δημιουργία μιας δραστηριότητας

Το πρώτο πράγμα που πρέπει να κάνουμε κατά τη δημιουργία μιας δραστηριότητας είναι να την δηλώσουμε στο `manifest`. Αυτό θα επιτρέψει στο σύστημα να «γνωρίζει την ύπαρξη της» ώστε να είναι σε θέση να την εκκινήσει όταν αυτό θα ζητηθεί από κάποιο συστατικό της εφαρμογής μας ή από κάποια άλλη εξωτερική εφαρμογή.

Αυτή είναι η δήλωση της αρχικής δραστηριότητας της εφαρμογής μας. Η ιδιότητα `android:name`, το όνομα που θα έχει η κλάση μας δηλαδή, είναι η μόνη απαραίτητη. Συνήθως όμως θα ορίσουμε και τον τίτλο, `android:label`, που θα έχει. Για την αρχική μας δραστηριότητα θα πρέπει να ορίσουμε και ένα `Intent filter` το οποίο δηλώνει ότι η δραστηριότητα μας ανταποκρίνεται στην ενέργεια `main`, το σημείο εισαγωγής δηλαδή στην

εφαρμογή μας, καθώς και την κατηγορία `LAUNCHER` ώστε να εμφανίζεται στον εκκινητή εφαρμογών(application launcher) του συστήματος.

```
<activity
      android:name="com.nkyrim.itapp.ui.activities.MainActivity"
      android:label="@string/title_activity_main"
      android:launchMode="standard" >
      <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
</activity>
```

Για να δημιουργήσουμε την κλάση μας θα επεκτείνουμε την κλάση `Activity` ή μία εκ των `FragmentActivity` και `ActionBarActivity` αν θέλουμε να χρησιμοποιήσουμε τα `Fragments` και την `ActionBar` σε παλαιότερες εκδόσεις. Σε αυτήν θα πρέπει να υλοποιήσουμε τις μεθόδους επανάκλησης που καλεί το σύστημα κατά τις μεταβάσεις μεταξύ των διαφόρων καταστάσεων του κύκλου ζωής της δραστηριότητας.

Εδώ υλοποιούμε μόνο την σημαντικότερη μέθοδο που είναι η δημιουργία της δραστηριότητας και δηλώνουμε το XML αρχείο που θα χρησιμοποιηθεί μέσω του αναγνωριστικού του στην κλάση `R` όπου έχουμε ορίσει την εμφάνιση καλώντας την μέθοδο `setContentView()`. Σε κάθε κλήση μεθόδου του κύκλου ζωής θα πρέπει να καλούμε την αντίστοιχη μέθοδο της υπέρ-κλάσης.

```
public class MainActivity extends Activity {
      @Override
      public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);
}
```

Θα δούμε την δημιουργία του ίδιου του περιεχομένου παρακάτω όταν θα αναλύσουμε τα `Views` και `ViewGroups`.

3.1.2 Διαχείριση του κύκλου ζωής της δραστηριότητας

Ένα από τα σημαντικότερα ζητήματα της ανάπτυξης εφαρμογών στο Android είναι η διαχείριση του κύκλου ζωής των δραστηριοτήτων μας καθώς έχουν πολλές ιδιαιτερότητες τις οποίες δεν συναντάμε αλλού. Είναι λοιπόν απαραίτητο να κατανοήσουμε αυτή τη διαδικασία καθώς είναι ζωτικής σημασίας για την ανάπτυξη μιας ισχυρής και ευέλικτης εφαρμογής. Ο κύκλος ζωής μιας δραστηριότητας επηρεάζεται άμεσα από τη σχέση της με άλλες δραστηριότητες, το back stack, τις αλλαγές στη διαμόρφωση αλλά και τον τρόπο λειτουργίας του ίδιου του συστήματος.

Μπορεί να βρεθεί σε 3 καταστάσεις:

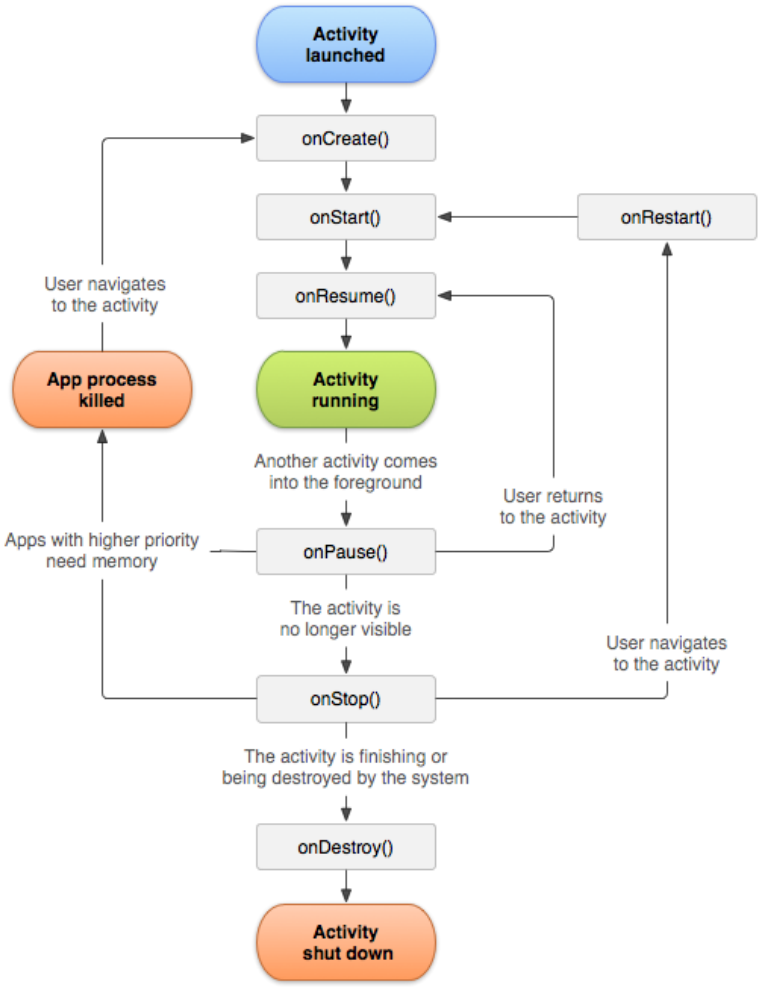
1. Σε εκτέλεση(*Resumed/Running*): Η δραστηριότητα είναι στο προσκήνιο και αλληλεπιδρά με τον χρήστη.
2. Σε παύση(*Paused*): Μια άλλη δραστηριότητα είναι στο προσκήνιο, αλλά η συγκεκριμένη είναι ακόμη ορατή. Αυτό συμβαίνει όταν η δραστηριότητα στο προσκήνιο δεν καλύπτει εντελώς την οθόνη ή είναι διαφανής.

3. Σταματημένη(*Stopped*): Η δραστηριότητα επισκιάζεται από μια άλλη και είναι τώρα στο «παρασκήνιο».

Μια δραστηριότητα που είναι σε παύση ή σταματημένη το σύστημα μπορεί να αποφασίσει να την τερματίσει για να ελευθερώσει μνήμη. Όταν μετά από αυτό η δραστηριότητα ξαναρχίσει, ο χρήστης δηλαδή επιστρέψει σε αυτήν, θα πρέπει να δημιουργηθεί από την αρχή.

3.1.2.1 Επεξήγηση των μεθόδων του κύκλου ζωής

Κατά τις μεταβάσεις μεταξύ των παραπάνω καταστάσεων η δραστηριότητα μας ενημερώνεται μέσω διαφόρων μεθόδων επανάκλησης(callbacks). Υπερβαίνοντας αυτές τις μεθόδους μπορούμε να διαχειριστούμε τις αλλαγές στην κατάσταση των δραστηριοτήτων μας.



Εικόνα 10: Κύκλος ζωής δραστηριότητας

- `onCreate`(): Καλείτε όταν η δραστηριότητα δημιουργείται για πρώτη φορά. Εδώ κάνουμε την αρχικοποίηση, ορίζουμε τα Views που θα χρησιμοποιηθούν, συνδεόμαστε με τα δεδομένα κλπ. Αυτή η μέθοδος χρησιμοποιεί ένα αντικείμενο `Bundle` που περιέχει την προηγούμενη κατάσταση της δραστηριότητας αυτής, αν υπάρχει. Πάντα ακολουθείται από την `onStart`().

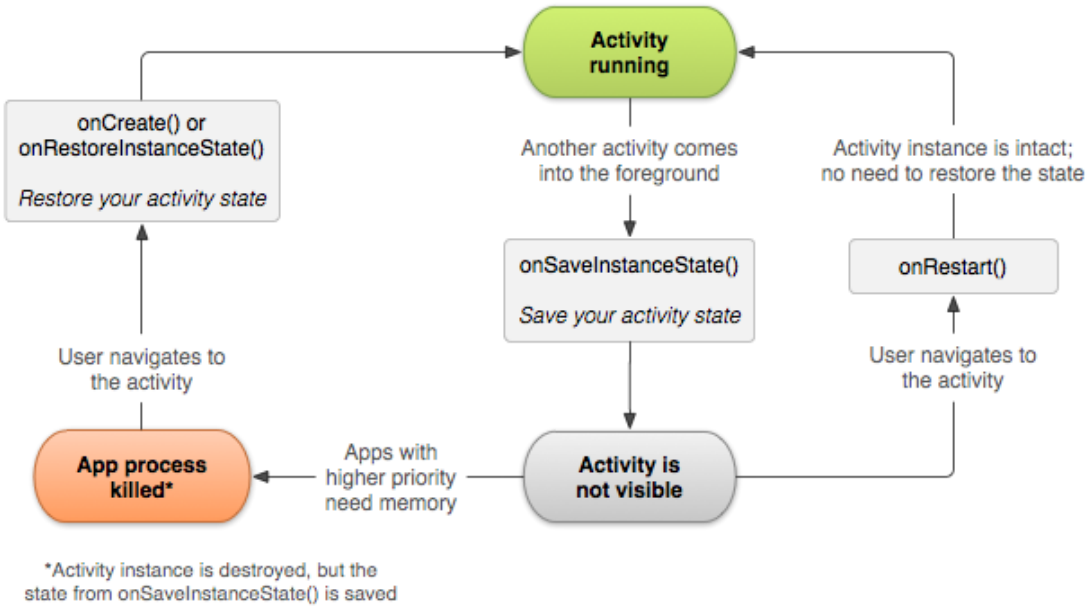
- `onRestart`(): Καλείτε αφού η δραστηριότητα έχει σταματήσει, ακριβώς πριν αρχίσει και πάλι.
- `onStart`(): Καλείτε λίγο πριν γίνει ορατή στο χρήστη.
- `onResume`(): Καλείτε ακριβώς πριν αρχίσει η δραστηριότητα να αλληλεπιδρά με το χρήστη. Σε αυτό το σημείο η δραστηριότητα είναι στην κορυφή της στοίβας.
- `onPause`(): Καλείτε όταν το σύστημα είναι έτοιμο να ξεκινήσει μια άλλη δραστηριότητα. Η μέθοδος αυτή χρησιμοποιείται συνήθως για την αποθήκευση δεδομένων, σταμάτημα εργασιών, κα. Θα πρέπει να ολοκληρώσουμε γρήγορα όμως γιατί η επόμενη δραστηριότητα δεν θα ξεκινήσει μέχρι να ολοκληρωθεί αυτή η μέθοδος.
- `onStop`(): Καλείτε όταν η δραστηριότητα δεν είναι πλέον ορατή στο χρήστη. Αυτό μπορεί να συμβεί γιατί καταστρέφεται είτε επειδή μια άλλη δραστηριότητα την καλύπτει.
- `onDestroy`(): Η τελευταία μέθοδος που καλείτε πριν η δραστηριότητα καταστραφεί είτε γιατί έχει ολοκληρωθεί, ο χρήστης την εγκαταλείπει, είτε η εφαρμογή την καταστρέφει προσωρινά.

3.1.2.2 Διατήρηση κατάστασης δραστηριότητας

Κατά τη διάρκεια που μια δραστηριότητα είναι σε παύση ή σταματημένη παραμένει στη μνήμη και η κατάσταση της διατηρείτε όπως ήταν όταν βρισκόταν σε εκτέλεση. Έτσι όταν επιστρέψει στο προσκήνιο η κατάσταση της με όλες τις αλλαγές που είχαν γίνει παραμένει η ίδια.

Παρόλα αυτά το σύστημα μπορεί να αναγκαστεί να τερματίσει την δραστηριότητα για να ανακτήσει μνήμη καταστρέφοντας έτσι το αντικείμενο και επομένως την τρέχουσα κατάσταση του. Συνεπώς σε περίπτωση που επιστρέψουμε σε αυτή τη δραστηριότητα το σύστημα δεν μπορεί απλώς να την συνεχίσει(`onResume`) θα πρέπει να την ξαναδημιουργήσει. Ο χρήστης όμως δεν το γνωρίζει αυτό, περιμένει επιστρέφοντας στη προηγούμενη εργασία του να τα βρει όπως ήταν. Σε αυτήν την περίπτωση για να εξασφαλίσουμε ότι οι σημαντικές πληροφορίες σχετικά με την κατάσταση της δραστηριότητα διατηρούνται χρησιμοποιούμε μια επιπλέον μέθοδο επανάκλησης, την `onSaveInstanceState()` που μας επιτρέπει να αποθηκεύσουμε αυτές τις πληροφορίες.

Το σύστημα καλεί την `onSaveInstanceState()` πριν καταστεί δυνατή η καταστροφή της δραστηριότητας. Το σύστημα περνά σε αυτή τη μέθοδο ένα αντικείμενο τύπου `Bundle` όπου μπορούμε να αποθηκεύσουμε τις πληροφορίες κατάστασης σχετικά με τη δραστηριότητα ως ζεύγη ονόματος-τιμής χρησιμοποιώντας μεθόδους όπως `putString()` και `putInt()`. Στη συνέχεια αν το σύστημα σκοτώσει τη διαδικασία μας και ο χρήστης γυρίσει πίσω σε αυτήν το σύστημα αναδημιουργεί την δραστηριότητα και περνάει το `Bundle` στην `onCreate()` και στην `onRestoreInstanceState()`. Χρησιμοποιώντας οποιαδήποτε από αυτές τις μεθόδους, μπορούμε να αποκαταστήσουμε την κατάσταση της δραστηριότητας. Το `Bundle` είναι `null` όταν η δραστηριότητα δημιουργείται για πρώτη φορά. Να σημειώσουμε ότι εδώ αποθηκεύουμε την τρέχουσα κατάσταση η οποία θέλουμε να παραμείνει αν η δραστηριότητα δεν καταστραφεί από δική μας υπαιτιότητα, δεν φύγουμε δηλαδή πατώντας το back, σε διαφορετική περίπτωση θα πρέπει να αποθηκεύσουμε τα δεδομένα μας στην `onPause`().



Εικόνα 11: Διατήρηση κατάστασης δραστηριότητας

3.2 Θραύσματα(Fragments)

Ένα `Fragment` αντιπροσωπεύει μια συμπεριφορά ή ένα τμήμα της διεπαφής χρήστη σε μια δραστηριότητα. Μπορούμε να συνδυάσουμε πολλά Fragments σε μια ενιαία δραστηριότητα για την κατασκευή ενός πολύ-τμηματικού UI ή να επαναχρησιμοποιήσουμε ένα κομμάτι σε πολλαπλές δραστηριότητες. Μπορούμε να σκεφτούμε ένα Fragment ως ένα αρθρωτό τμήμα μιας δραστηριότητας, το οποίο έχει το δικό του κύκλο ζωής, λαμβάνει τα δικά του γεγονότα εισόδου και το οποίο μπορούμε να προσθέσουμε ή να αφαιρέσουμε ενώ η δραστηριότητα εκτελείται.

Τα `Fragments` πρέπει πάντα να ενσωματώνονται σε μια δραστηριότητα και ο κύκλος ζωής τους επηρεάζεται άμεσα από τον κύκλο ζωής της δραστηριότητας της οποίας αποτελούν τμήμα. Καθ' όλη τη διάρκεια που η δραστηριότητα είναι σε εκτέλεση μπορούμε να χειριστούμε τα `Fragments`, το καθένα ξεχωριστά, προσθέτοντας ή αφαιρώντας τα ενώ μπορούμε παράλληλα να χρησιμοποιήσουμε το δικό τους back stack που έχει τρόπο λειτουργίας παρόμοιο με των δραστηριοτήτων.

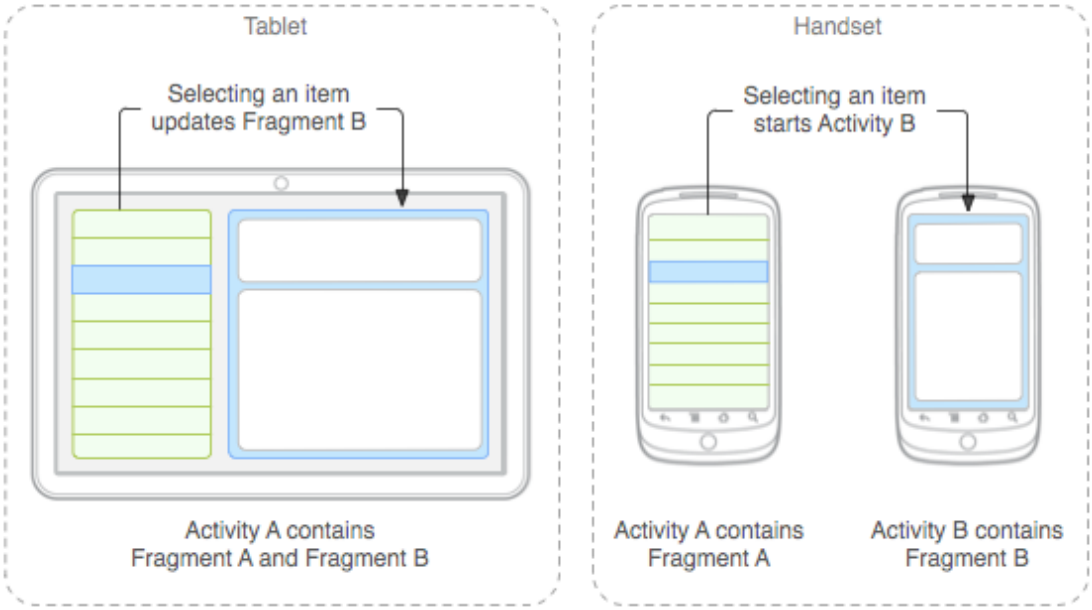
Κάθε φορά που προσθέτουμε ένα Fragment αυτό «ζει» σε ένα `ViewGroup` στην ιεραρχία των UI στοιχείων της δραστηριότητας παρέχοντας και αυτό με τη σειρά του την δικιά του ιεραρχία στοιχείων. Παρ' όλα αυτά μπορούμε να έχουμε και Fragments χωρίς UI τα οποία χρησιμοποιούνται σαν «αόρατοι εργάτες» για την δραστηριότητα μας. Αυτή η κατηγορία θα μας φανεί πολύ χρήσιμη στη διαχείριση των αλλαγών στη διαμόρφωση όπως θα δούμε παρακάτω.

3.2.1 Σχεδιαστική φιλοσοφία

Τα Fragments εισήχθηκαν στην έκδοση 3.0, κυρίως για να υποστηρίξουν μια πιο δυναμική και ευέλικτη σχεδίαση UI για μεγάλες οθόνες όπως των tablets. Επειδή η οθόνη ενός tablet είναι πολύ μεγαλύτερη από εκείνη του τηλεφώνου υπάρχει μεγαλύτερη επιφάνεια για παρουσίαση περιεχομένου και παροχή περισσότερης λειτουργικότητας. Τα θραύσματα επιτρέπουν αυτά τα σχέδια χωρίς την ανάγκη για εμάς να διαχειριστούμε πολύπλοκες αλλαγές στην ιεραρχία των UI

στοιχείων. Με τη διαίρεση της διάταξης μιας δραστηριότητας σε τμήματα μπορούμε να τροποποιήσουμε την εμφάνιση των δραστηριοτήτων κατά το χρόνο εκτέλεσης.

θα πρέπει να σχεδιάζουμε κάθε `Fragment` ως ένα αρθρωτό και επαναχρησιμοποιήσιμο τμήμα μιας δραστηριότητας. Δηλαδή, επειδή το κάθε `Fragment` ορίζει τη δική του διάταξη, συμπεριφορά και κύκλο ζωής, μπορούμε να το συμπεριλάβουμε σε πολλαπλές δραστηριότητες, έτσι θα πρέπει κατά το σχεδιασμό να αποφύγουμε τον απευθείας χειρισμό ενός `Fragment` από ένα άλλο. Κατά το σχεδιασμό της εφαρμογής για να υποστηρίξουμε τόσο τα tablets όσο και τα κινητά τηλέφωνα μπορούμε να επαναχρησιμοποιήσουμε κομμάτια σε διαφορετικές διαμορφώσεις διάταξης για να βελτιώσουμε την εμπειρία χρήσης με βάση το διαθέσιμο χώρο της οθόνης.



Εικόνα 12: Μέθοδος δημιουργίας πολλαπλών διατάξεων

3.2.2 Δημιουργία Fragments

Για να δημιουργήσουμε ένα θραύσμα θα πρέπει να επεκτείνουμε την κλάση `Fragment` ή κάποια από τις υπάρχουσες υποκλάσεις της οι οποίες παρέχουν επιπλέον λειτουργικότητα όπως η `DialogFragment`, `ListFragment` και `PreferenceFragment`. Πολλές από τις μεθόδους του είναι παρόμοιες με τις μεθόδους του `Activity`, πχ `onCreate()`, `onPause()` κλπ.

Ο κώδικας για την δημιουργία ενός απλού Fragment με το δικό του UI. Υπερβαίνουμε την `onCreateView()` μέθοδο για να δηλώσουμε το UI το οποίο θα έχει το fragment μας.

```
public class MailFragment extends Fragment {

  public View onCreateView(LayoutInflater inflater,
                           ViewGroup container,
                           Bundle savedInstanceState) {
    return inflater.inflate(R.layout.fragment_mail,container,false);
  }
}
```

Μπορούμε να προσθέσουμε το `Fragment` στο `Activity` μας με δύο διαφορετικούς τρόπους:

1. Δηλώνοντας το `Fragment` στο xml αρχείο διάταξης της δραστηριότητας μας όπως θα κάναμε με οποιοδήποτε άλλο στοιχείο ελέγχου(θα δούμε λεπτομερώς τα αρχεία layout παρακάτω). Η ιδιότητα `android:name` καθορίζει την κλάση που θα αρχικοποιηθεί και τη μέθοδο `onCreateView()` του οποίου θα καλέσει για να πάρει το στοιχείο ελέγχου(`View`) που θα προσθέσει στην ιεραρχία.

```
<?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="horizontal" >

    <fragment
        android:id="@+id/fragment1"
        android:name="com.nkyrim.itapp.ui.fragments.MailFragment "
        android:layout_width="match_parent"
        android:layout_height="match_parent"

</LinearLayout>
```

Κάθε `Fragment` απαιτεί ένα μοναδικό αναγνωριστικό που το σύστημα μπορεί να χρησιμοποιήσει για να το επαναφέρει εάν η δραστηριότητα ξαναδημιουργηθεί (και το οποίο μπορούμε να χρησιμοποιήσουμε για να το διαχειριστούμε). Υπάρχουν τρεις τρόποι για να αποδοθεί ένα αναγνωριστικό:

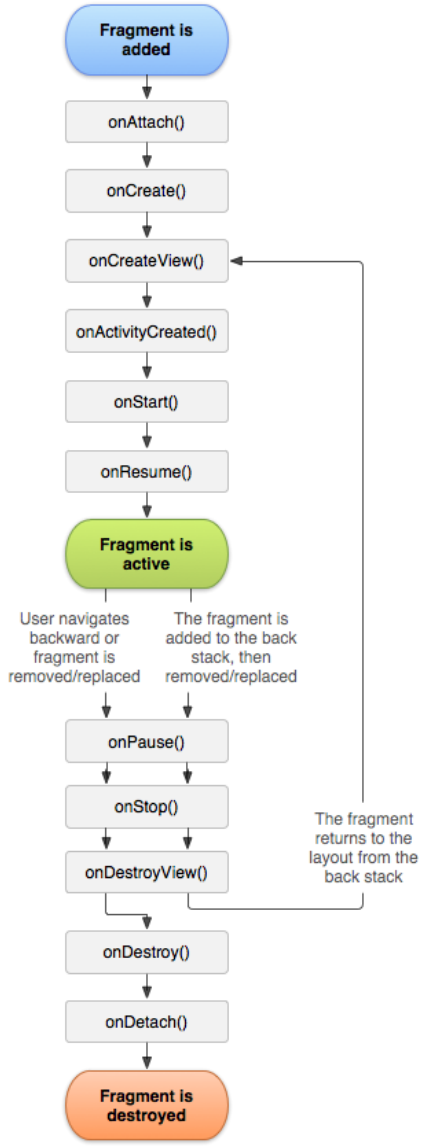
- Δηλώνοντας την ιδιότητα `android:id`.
- Δηλώνοντας την ιδιότητα `android:tag`.
- Εάν δεν δηλώσουμε κανένα από τα δύο προηγούμενα, το σύστημα χρησιμοποιεί το ID του στοιχείου UI που το περιέχει.

2. Προγραμματιστικά, δημιουργώντας το `Fragment` στον κώδικα μας και προσθέτοντας το σε ένα `ViewGroup`. Για να το επιτύχουμε αυτό χρησιμοποιούμε ένα `FragmentTransaction` μέσω του `FragmentManager` της δραστηριότητας μας. Μπορούμε έτσι να προσθέσουμε, να αφαιρέσουμε ή να αντικαταστήσουμε υπάρχοντα `Fragments`.

```
FragmentManager fm = getFragmentManager();
FragmentTransaction ft = fm.beginTransaction();

MailFragment fragment = new MailFragment();
ft.add(R.id.fragment_container, fragment);
ft.commit();
```

3.2.3 Διαχείριση του κύκλου ζωής του Fragment

Ο κύκλος ζωής των Fragments είναι παρόμοιος με των Activities και επηρεάζεται από αυτών καθώς αποτελούν μέρος τους.



Εικόνα 13: Κύκλος ζωής θραύσματος

Ακριβώς όπως οι δραστηριότητες τα `Fragments` μπορούν να βρεθούν στις τρείς ίδιες καταστάσεις:

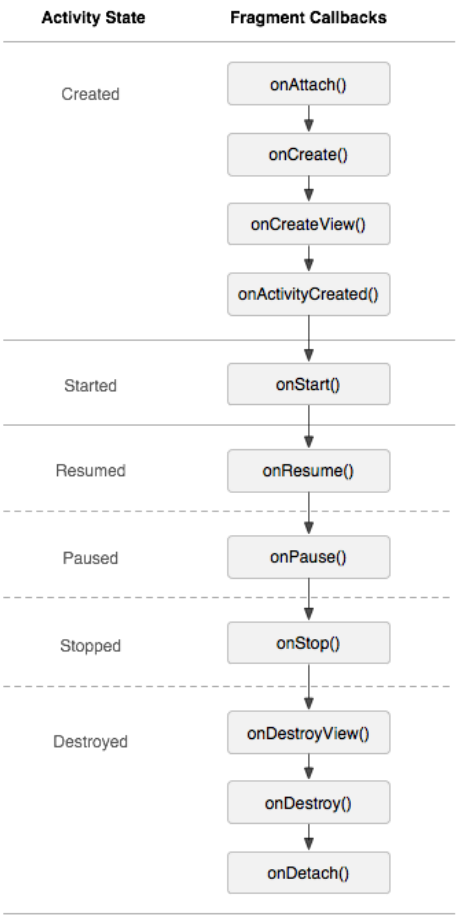
1. Σε εκτέλεση(*Resumed/Running*): Το Fragment είναι ορατό και αλληλεπιδρά με τον χρήστη.
2. Σε παύση(*Paused*): Μια άλλη δραστηριότητα είναι στο προσκήνιο, αλλά η συγκεκριμένη είναι ακόμη ορατή. Αυτό συμβαίνει όταν η δραστηριότητα στο προσκήνιο δεν καλύπτει εντελώς την οθόνη ή είναι διαφανής.
3. Σταματημένη(*Stopped*): Η δραστηριότητα επισκιάζεται από μια άλλη, η δραστηριότητα είναι τώρα στο «παρασκήνιο» ή μόνο το συγκεκριμένο Fragment έχει «αντικατασταθεί» από κάποιο άλλο και έχει περάσει στο back stack.

Οι καταστάσεις αυτές όμως δεν είναι απαραίτητο να είναι πάντα συγχρονισμένες με τις αντίστοιχες της δραστηριότητας. Μόλις η δραστηριότητα φθάνει στην κατάσταση «σε εκτέλεση», μπορούμε ελεύθερα να προσθέσουμε και να αφαιρέσουμε Fragments, έτσι η κατάσταση τους μπορεί να αλλάξει και ανεξάρτητα.

Τα Fragments διαθέτουν επίσης το δικό τους back stack ανεξάρτητο από τα Activities το οποίο λειτουργεί με παρόμοιο τρόπο.

3.2.3.1 Συντονισμός με τον κύκλο ζωής της Δραστηριότητας

Η κύκλος ζωής της δραστηριότητας στην οποία το `Fragment` ζει επηρεάζει άμεσα τον κύκλο ζωής του έτσι κάθε μέθοδος του κύκλου ζωής της δραστηριότητας καταλήγει σε ανάλογη μέθοδο για κάθε `Fragment`. Για παράδειγμα, όταν η δραστηριότητα λαμβάνει την `onPause()` κάθε `Fragment` στη δραστηριότητα λαμβάνει την `onPause()`.



Εικόνα 14: Αντιστοίχηση καταστάσεων Activity - Fragment

Τα `Fragments` έχουν μερικές επιπλέον μεθόδους κύκλου ζωής ωστόσο που χειρίζονται την αλληλεπίδραση με τη δραστηριότητα προκειμένου να εκτελούν ενέργειες όπως η κατασκευή και η καταστροφή του UI του θραύσματος. Αυτές οι πρόσθετες μέθοδοι είναι:

- `onAttach()`: Καλείται όταν το θραύσμα έχει συσχετισθεί με τη δραστηριότητα.
- `onCreateView()`: Καλείται για να δημιουργήσει την ιεραρχία UI που σχετίζεται με το `Fragment`.
- `onActivityCreated()`: Καλείται όταν η μέθοδος `onCreate()` της δραστηριότητας αυτής έχει ολοκληρωθεί.
- `onDestroyView()`: Καλείται όταν η ιεραρχία UI που σχετίζεται με το `Fragment` αφαιρείται.
- `onDetach()`: Καλείται όταν το `Fragment` διαχωρίζεται από τη δραστηριότητα.

3.2.3.3 Διατήρηση κατάστασης Fragment

Για να διατηρήσουμε την κατάσταση των `Fragments` μας μπορούμε να χρησιμοποιήσουμε τον ίδιο μηχανισμό με των Activities. Υπερβαίνοντας δηλαδή την `onSaveInstanceState()` και αποθηκεύοντας την τρέχουσα κατάσταση σε ένα αντικείμενο `Bundle` από το οποίο μπορούμε έπειτα να επαναφέρουμε την κατάσταση σε μια εκ των `onCreate()`, `onCreateView()` και `onActivityCreated()`. Συνήθως όμως σε μια εκ των δύο τελευταίων έτσι ώστε να έχει δημιουργηθεί πρώτα η ιεραρχία των στοιχείων UI.

Ένας άλλος τρόπος είναι να χρησιμοποιήσουμε ένα headless fragment τα οποία θα δούμε αμέσως και τα οποία διατηρούν την κατάσταση τους μετά από αναδημιουργία της δραστηριότητας. Παρ' όλα αυτά όμως η κατάσταση μπορεί να χαθεί αν το σύστημα καταστρέψει τη δραστηριότητα λόγω έλλειψης μνήμης.

3.2.4 Fragments χωρίς UI

Στις περισσότερες περιπτώσεις τα Fragments χρησιμοποιούνται για να αποτελέσουν ένα τμήμα του UI της εφαρμογής. Ωστόσο, μπορούμε επίσης να δημιουργήσουμε ένα Fragments χωρίς UI για την παροχή «συμπεριφοράς παρασκηνίου» που εξακολουθεί να υφίσταται ακόμα και μετά από αναδημιουργία της δραστηριότητας. Αυτά τα Fragments λέγονται και "headless".

Μπορούμε να ορίσουμε ένα Fragment να διατηρήσει το τρέχον στιγμιότυπο του όταν η μητρική δραστηριότητα αναδημιουργείται χρησιμοποιώντας τη μέθοδο `setRetainInstance()`. Αφού καλέσουμε τη μέθοδο αυτή ο κύκλος ζωής του `Fragment` θα αλλάξει. Παρόλο που μπορούμε να το κάνουμε αυτό και σε ένα `Fragment` με UI αυτό δεν προτείνεται και δεν θα μπορούμε να το χρησιμοποιήσουμε με το back stack.

Μετά την δημιουργία του το Fragment θα διατηρηθεί όταν η δραστηριότητα επανεκκινήσει και θα αποκοπεί από την δραστηριότητα(εκτελείτε η `onDetach()` αλλά όχι η `onDestroy()`). Όταν η νέα δραστηριότητα δημιουργηθεί θα κληθούν οι μέθοδοί εκτός από την `onCreate()`.

Ένα βοηθητικό fragment που χρησιμοποιούμε στην εφαρμογή μας για την διατήρηση της κατάστασης σε κάποιες δραστηριότητες.

```
public class StoreFragment extends Fragment {
	private Map<String, Object> store =
                                        new ConcurrentHashMap<String, Object>();

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setRetainInstance(true);

	}

	@Override
	public View onCreateView(LayoutInflater inflater,
                                ViewGroup container,
                                Bundle savedInstanceState) {
		return null;
	}

	public void put(String key, Object value) {
		if (key == null)
			throw new IllegalArgumentException("key cannot be null");

		if (value != null) store.put(key, value);
		else return;
```

```
	}

	public Object get(String key) {
		if (key != null) return store.get(key);
		else throw new IllegalArgumentException("key cannot be null");
	}

	public void remove(String key) {
		if (key != null) store.remove(key);
		else throw new IllegalArgumentException("key cannot be null");
	}
}
```

3.2.5 Επικοινωνία μεταξύ Fragments και Activities

Όπως είπαμε ένα `Fragment` μπορεί να αποτελεί τμήμα πολλών δραστηριοτήτων αλλά ταυτόχρονα να αφαιρείται από αυτήν κατά τον χρόνο εκτέλεσης, πχ λόγο αλλαγής προσανατολισμού της συσκευής. Συνεπώς τα Fragments δεν θα πρέπει να επικοινωνούν απ' ευθείας μεταξύ τους αλλά μέσω της δραστηριότητας η οποία και θα συντονίζει την επικοινωνία, αποστέλλοντας πχ το αίτημα στο κατάλληλο Fragment ή εκκινώντας μια νέα δραστηριότητα αν είναι απαραίτητο.

Για να το επιτύχουμε αυτό το σκοπό θα πρέπει το Fragment να ορίζει ένα Interface σαν ένα εσωτερικό τύπο και να απαιτεί από την δραστηριότητα να υλοποιεί αυτόν τον τύπο. Με αυτό τον τρόπο το Fragment δεν χρειάζεται να γνωρίζει ούτε την ύπαρξη άλλων Fragments ούτε περαιτέρω πληροφορίες για την δραστηριότητα.

Στο `BulletinListFragment` της εφαρμογής μας ορίζουμε το `interface` με το οποίο θα αποστέλλουμε την ανακοίνωση προς εμφάνιση.

```
public class BulletinFragment extends Fragment {
	...
	...
	private OnItemSelectedListener listener;

	public interface OnItemSelectedListener {
		public void onItemSelected(Bulletin bulletin);
	}
```

Σιγουρευόμαστε επίσης κατά την προσάρτηση του `Fragment` ότι η δραστηριότητα όντως υλοποιεί το interface μας.

```
public void onAttach(Activity activity) {
	super.onAttach(activity);

	if (activity instanceof OnItemSelectedListener) {
		listener = (OnItemSelectedListener) activity;
	} else  {
		throw new ClassCastException(activity.toString() +
		" must implemenet BulletinFragment.OnItemSelectedListener");
	}
}
```

Για να το χρησιμοποιήσουμε τώρα αρκεί να καλέσουμε την μέθοδο της τρέχουσας δραστηριότητας.

```
public void onItemClick(AdapterView<?> p, View v, int pos, long id) {
	listener.onItemSelected((Bulletin) p.getItemAtPosition(pos));
}
```

Τέλος η δραστηριότητα θα είναι αυτή που θα αποφασίσει το «επόμενο βήμα». Στη συγκεκριμένη περίπτωση αν υπάρχει ήδη το Fragment για την προβολή, είμαστε δηλαδή σε layout με δύο panes, ή αν θα πρέπει να δημιουργηθεί νέο αντικαθιστώντας τη προβολή της λίστας και σπρώχνοντας την στο back stack.

```
public void onItemSelected(Bulletin bulletin) {

	BulletinDetailFragment detail = (BulletinDetailFragment)
					fm.findFragmentByTag("DETAIL");

	if (!twoPane) {
		detail = new BulletinDetailFragment();
		fm.beginTransaction()
			.replace(R.id.paneMaster, detail,"DETAIL")
			.addToBackStack(null)
			.commit();
		fm.executePendingTransactions();
	}

	detail.showContent(bulletin);
}
```

3.3 Στοιχεία ελέγχου γραφικού περιβάλλοντος

Όλα τα στοιχεία διεπαφής χρήστη σε μια εφαρμογή Android είναι κατασκευασμένα χρησιμοποιώντας αντικείμενα τύπου `View` και `ViewGroup`. Το `View` είναι ένα αντικείμενο που «ζωγραφίζει» κάτι στην οθόνη με το οποίο ο χρήστης μπορεί να αλληλεπιδράσει. Το `ViewGroup` είναι ένα αντικείμενο που κρατά άλλα `Views`(και `ViewGroups`) αντικείμενα, προκειμένου να ορίσει τη διάταξη της διεπαφής.

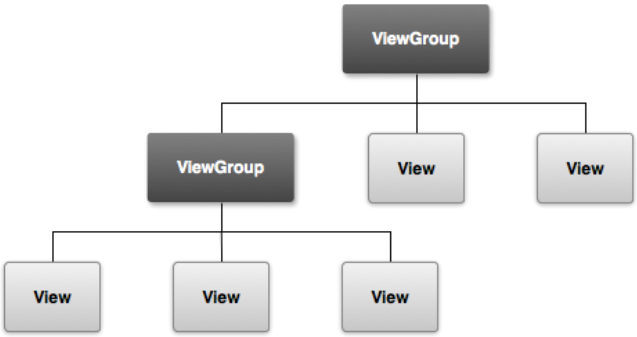
Το Android παρέχει μια συλλογή από στοιχεία και από τις δυο κατηγορίες `View` και `ViewGroup` που προσφέρουν στοιχεία ελέγχου εισόδου(όπως κουμπιά και πεδία κειμένου) και διάφορα μοντέλα διάταξης (όπως γραμμικό, πίνακας κα).

Ένα πολύ σημαντικό κομμάτι του γραφικού περιβάλλοντος είναι η `ActionBar`, ένα στοιχείο το οποίο εισήχθηκε για πρώτη φορά στην έκδοση 3.0 και είναι διαθέσιμο και σε παλιότερες εκδόσεις μέσω τις βιβλιοθήκης υποστήριξης και συμβάλει σε μεγάλο βαθμό στο να έχουμε μια «κοινή και οικεία» διεπαφή σε διαφορετικές εφαρμογές.

3.3.1 Στοιχεία Διάταξης

Η διεπαφή χρήστη για κάθε στοιχείο της εφαρμογής μας ορίζεται χρησιμοποιώντας μια ιεραρχία `View` και `ViewGroup` αντικείμενων. Κάθε `ViewGroup` είναι ένα αόρατο κοντέινερ που οργανώνει τα `Views` τα οποία εμφανίζουν κάποιο μέρος του UI.

Για να ορίσουμε τη διάταξή μας, μπορούμε να δημιουργήσουμε αντικείμενα `View` στον κώδικα και να οικοδομήσουμε την ιεραρχία των στοιχείων αλλά ο ευκολότερος και πιο αποτελεσματικός τρόπος για να καθοριστεί η διάταξη είναι με ένα αρχείο XML. Αυτό προσφέρει μια αναγνώσιμη από τον άνθρωπο δομή για τη διάταξη, παρόμοια με την HTML αλλά κυρίως μας επιτρέπει να διαχωρίσουμε την εμφάνιση από τον εκτελέσιμο κώδικα.

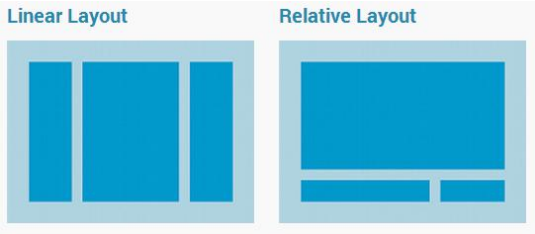


Εικόνα 15: Ιεραρχία στοιχείων ελέγχου

3.3.1.1 Τύποι Διατάξεων

Κάθε τύπος `ViewGroup` παρέχει ένα μοναδικό τρόπο για την προβολή των στοιχείων που περιλαμβάνει. Μπορούμε να έχουμε πολλαπλά εμφωλευμένα `ViewGroup` για να επιτύχουμε πιο περίπλοκες διατάξεις, θα πρέπει όμως να προσπαθούμε να έχουμε όσο το δυνατόν λιγότερα ώστε να έχουμε καλύτερη απόδοση και να γίνεται γρηγορότερα η εμφάνιση των στοιχείων.

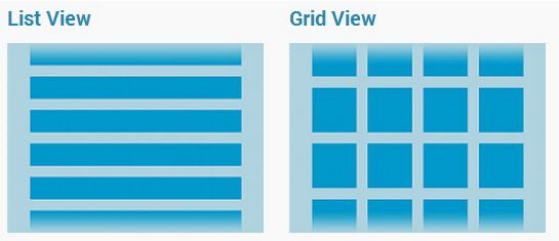
- `LinearLayout`: Μια διάταξη που οργανώνει τα περιλαμβανόμενα στοιχεία της σε μια ενιαία οριζόντια ή κάθετη γραμμή. Μπορούμε να ορίσουμε την κατεύθυνση με το χαρακτηριστικό `android:orientation`.
- `RelativeLayout`: Οργανώνει τα στοιχεία σε θέσεις σχετικές μεταξύ τους ή με την ίδια τη διάταξη, πχ στο κέντρο της διάταξης ή αριστερά από ένα άλλο `View`.
- `TableLayout`: Μια διάταξη που οργανώνει τα στοιχεία της σε γραμμές και στήλες. Αποτελείται από έναν αριθμό `TableRow` αντικειμένων με το καθένα να ορίζει μια γραμμή. Η διάταξη περιλαμβάνει τόσες στήλες όσες η γραμμή με τα περισσότερα στοιχεία με κάθε στοιχείο να μπορεί να πιάσει περισσότερες από μία στήλες.
- `GridLayout`: Προστέθηκε με την έκδοση 4.0 αλλά είναι διαθέσιμη και σε παλιότερες εκδόσεις. Παρουσιάζει τα στοιχεία σε ένα ορθογώνιο πλέγμα. Μία πολύ ευέλικτη διάταξη που έχει πολλές δυνατότητες αλλά και ιδιαιτερότητες.



Πέρα από τους 4 βασικούς τύπους που είδαμε υπάρχουν και τύποι διατάξεων που βασίζονται σε `Adapters`(κλάσεις controllers) οι οποίοι είναι υπεύθυνοι για το «γέμισμα» των διατάξεων με στοιχεία κατά το χρόνο εκτέλεσης. Είναι πολύ βασικοί καθώς χρησιμοποιούν μόνο τους απαραίτητους πόρους για τα στοιχεία που φαίνονται τη συγκεκριμένη στιγμή στην οθόνη. Τους χρησιμοποιούμε λοιπόν όταν δεν γνωρίζουμε τον αριθμό των στοιχείων εκ των προτέρων ή όταν έχουμε πολύ μεγάλο αριθμό στοιχείων να παρουσιάσουμε.

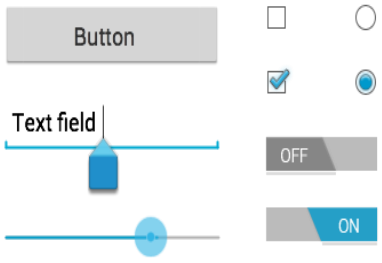
- `ListView`: Παρόμοιο με ένα οριζόντιο `LinearLayout`, παρουσιάζει μια λίστα στοιχείων.

- `ExpandableListView`: Μία οριζόντια λίστα δύο επιπέδων. Επιτρέπει την ομαδοποίηση στοιχείων σε ξεχωριστές λίστες.
- `GridView`: Παρουσιάζει ένα πλέγμα στοιχείων.



3.3.3 Στοιχεία ελέγχου

Τα στοιχεία ελέγχου ή widgets είναι τα αντικείμενα στο γραφικό περιβάλλον με τα οποία αλληλεπιδρά ο χρήστης. Το Android μας παρέχει μια μεγάλη συλλογή στάνταρ `Views` τα οποία μπορούμε να χρησιμοποιήσουμε για την ανάπτυξη της εφαρμογής μας, όπως κουμπιά, πεδία κειμένου κα. Μπορούμε επίσης να δημιουργήσουμε δικά μας στοιχεία ή να επεκτείνουμε και να συνδυάσουμε τα ήδη υπάρχοντα.



Εικόνα 16: Διάφορα Views

Κατά τη δημιουργία του αρχείου διάταξης προσθέτουμε τα στοιχεία ελέγχου σαν XML στοιχεία στο αρχείο μας.

3.3.4 Δημιουργία διατάξεων

Το πρώτο βήμα για την δημιουργία των διατάξεων είναι να ορίσουμε τη δομή χρησιμοποιώντας το XML λεξιλόγιο του Android με τον ίδιο τρόπο που δημιουργούσαμε μια σελίδα με HTML. Κάθε αρχείο διάταξης πρέπει να περιέχει ακριβώς ένα στοιχείο ρίζας, η οποία πρέπει να είναι ένα View ή ViewGroup αντικείμενο. Αφού ορίσουμε το στοιχείο ρίζας μπορούμε να προσθέσουμε επιπλέον αντικείμενα διάταξης ή widgets ώστε να χτίσουμε σταδιακά μια ιεραρχία Views που θα ορίζει τη διάταξή μας.

Εδώ είναι ο κώδικας ο οποίος ορίζει την αρχική δραστηριότητα της εφαρμογής (`activity_main.xml`). Το `LinearLayout` είναι το `ViewGroup` στοιχείο ρίζας που περιλαμβάνει τα υπόλοιπα στοιχεία ελέγχου που θα δούμε παρακάτω όπως το `ImageView` για το logo και τα `Buttons` για την μετάβαση στις αντίστοιχες δραστηριότητες.

```
<?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="wrap_content"
        android:orientation="vertical"
        android:padding="10dp" >

        <ImageView
            android:id="@+id/imgLogo"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:adjustViewBounds="true"
```

```
            android:contentDescription="@string/app_name"
            android:src="@drawable/ic_main_logo" />

        <Button
            android:id="@+id/btnHydra"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:onClick="showHydra"
            android:text="@string/hydra_services"/>

        <Button
            android:id="@+id/btnPithia"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:onClick="showPithia"
            android:text="@string/pithia_services"/>
    ...
    ...
</LinearLayout>
```

Στη δραστηριότητα μας θα πρέπει ορίσουμε το αρχείο που δημιουργήσαμε με τη μέθοδο `setContent()` και μπορούμε να αποθηκεύσουμε αναφορές προς τα στοιχεία που ορίσαμε στο XML αρχείο μας με την μέθοδο `findViewById()` κατά την `onCreate()` φάση του κύκλου ζωής.

```
public class MainActivity extends BaseActivity {
    private Button btnInfo;
    private Button btnServices;
    ...

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    btnInfo = (Button) findViewById(R.id.btnInfo);
    btnServices = (Button) findViewById(R.id.btnServices);
    ...
}
```

Εικόνα 17: Εμφάνιση της MainActivity

3.4 Υποστήριξη διαφορετικών μεγεθών οθόνης

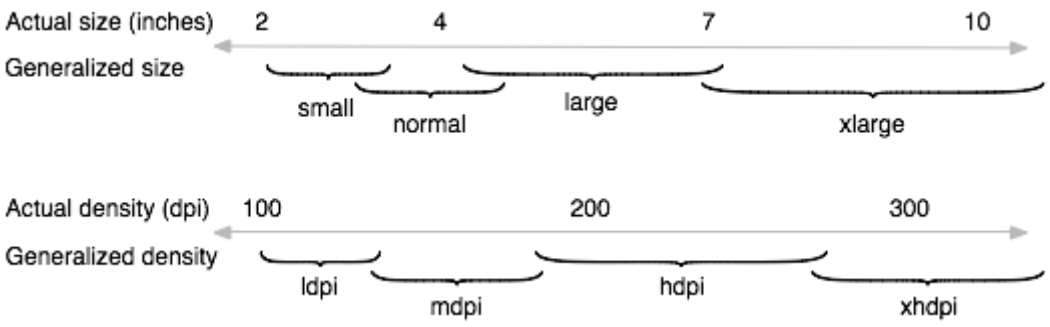
Μέχρι τώρα είδαμε τα βασικά συστατικά και στοιχεία που απαιτούνται για τη δημιουργία διεπιφανειών καθώς και τον τρόπο οργάνωσης των πόρων της εφαρμογής. Τώρα θα δούμε πως μπορούμε να παρέχουμε την καλύτερη δυνατή εμφάνιση και λειτουργικότητα για την εφαρμογή μας υποστηρίζοντας τα διάφορα μεγέθη και πυκνότητες των οθονών.

Αρχικά θα πρέπει να ξεκαθαρίσουμε κάποιες έννοιες:

- Μέγεθος οθόνης: Το πραγματικό μέγεθος, που μετράται ως διαγώνιος της οθόνης.
- Ανάλυση: Ο συνολικός αριθμός των pixels σε μια οθόνη.
- Πυκνότητα οθόνης: Η ποσότητα των pixels σε μια φυσική περιοχή της οθόνης. Συνήθως αναφέρεται ως **dpi**, κουκίδες ανά ίντσα.
- Προσανατολισμός: Ο προσανατολισμός της οθόνης από την άποψη του χρήστη του. Μπορεί να είναι είτε οριζόντια(***landscape***) ή κάθετα(***portrait***). Ο προσανατολισμός μπορεί να αλλάξει κατά το χρόνο εκτέλεσης όταν ο χρήστης περιστρέφει τη συσκευή.
- Ανεξάρτητο πυκνότητας pixel(density independent pixel, dip ή **dp**): Μια εικονική μονάδα pixel που θα πρέπει να χρησιμοποιούμε κατά τον καθορισμό διατάξεων UI η οποία εκφράζει τα μεγέθη με ανεξάρτητο από την τρέχουσα πυκνότητα τρόπο. Το **dp** είναι ισοδύναμο με ένα φυσικό pixel σε μια οθόνη 160 dpi η οποία θεωρείτε ως η βασική πυκνότητα από το σύστημα.
- Ανεξάρτητο κλίμακας pixel(scale independent pixel **sp**): Μια μονάδα για τον καθορισμό του μεγέθους των γραμματοσειρών. Συνυπολογίζει τις προτιμήσεις του χρήστη.

Για να απλοποιήσει τον τρόπο που σχεδιάζουμε διεπαφές χρήστη για πολλαπλές οθόνες το Android χωρίζει το φάσμα των μεγεθών οθόνης και πυκνοτήτων σε:

- Ένα σύνολο τεσσάρων γενικευμένων μεγεθών: small, normal, large, and xlarge. Αυτά χρησιμοποιούνται στις εκδόσεις πριν της 3.2 για τα αρχεία διατάξεων.
- Ένα σύνολο τεσσάρων γενικευμένων πυκνοτήτων: ldpi (low), mdpi (medium), hdpi (high), and xhdpi (extra high). Στις νεότερες εκδόσεις προστέθηκαν και οι xxhdpi και xxxhdpi.
- Νέα προσδιοριστικά μεγέθους που εισήχθηκαν με την έκδοση 3.2 και ορίζουν κατηγορίες με βάση μεγέθη σε **dp**.
    - Ελάχιστο πλάτος: Το ελάχιστο πλάτος ανεξάρτητος προσανατολισμού(`sw<N>dp`)
    - Πλάτος: Το ελάχιστο πλάτος με βάση τον προσανατολισμό(`w<N>dp`)
    - Ύψος: Το ελάχιστο ύψος με βάση τον προσανατολισμό(`η<N>dp`)



Εικόνα 18: Αντιπαράθεση μεγεθών - πυκνοτήτων

3.4.1 Χρήση γραφικών διαφορετικών αναλύσεων

Θα πρέπει πάντα να παρέχουμε πόρους γραφικών που προσαρμόζονται σωστά σε κάθε μία από τις γενικευμένες κατηγορίες πυκνότητας: χαμηλή, μεσαία, υψηλή και πολύ υψηλή πυκνότητα. Αυτό βοηθά να επιτευχθεί καλή ποιότητα γραφικών και υψηλές επιδόσεις σε όλες τις πυκνότητες της οθόνης.

Παράδειγμα ορισμού του logo της εφαρμογής:

- root/
  - res/
    - drawable-xhdpi/
      - ic_main_logo.png
    - drawable-hdpi/
      - ic_main_logo.png
    - drawable-mdpi/
      - ic_main_logo.png
    - drawable-ldpi/
      - ic_main_logo.png

Έπειτα στον κώδικα μας μπορούμε να το ανακτήσουμε καλώντας απλώς `R.drawable.ic_main_logo` ή στα αρχεία διάταξης ορίζοντας αναφορές προς το `@drawable/ic_main_logo` και το σύστημα θα αναλάβει να φέρει την κατάλληλη έκδοση ανάλογα με την πυκνότητα της οθόνης.

3.4.2 Χρήση διαφορετικών αρχείων διατάξεων

Για να ορίσουμε 2 διαφορετικά αρχεία διατάξεων θα πρέπει να τα τοποθετήσουμε στον κατάλληλο κατάλογο με τους προσδιορισμούς μεγέθους και προσανατολισμού.

Αρχείο διάταξης για την δραστηριότητα λίστας email σε διάταξη ενός παραθύρου. Όπως βλέπουμε υπάρχει μόνο το βασικό Fragment της λίστας

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragMailList"
    android:name="com.nkyrim.itapp.ui.fragments.MailFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Το αρχείο για τη διάταξη 2 παραθύρων. Περιλαμβάνει τόσο τη λίστα όσο και την προβολή.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:baselineAligned="false"
    android:orientation="horizontal" >

    <fragment
        android:id="@+id/fragMailList"
        android:name="com.nkyrim.itapp.ui.fragments.MailFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="2" />

    <fragment
        android:id="@+id/fragMailDetail"
        android:name="com.nkyrim.itapp.ui.fragments.MailDetailFragment"
        android:layout_width="match_parent"
```

```
        android:layout_height="match_parent"
        android:layout_weight="1" />

</LinearLayout>
```

Ορίζουμε ουσιαστικά 3 διαφορετικές περιπτώσεις. Ξεκινώντας από κάτω προς τα πάνω. Ορίζουμε το 2-pane για τις συσκευές με ελάχιστο πλάτος 720(sw720dp), συνήθως 10” tablets, και για ελάχιστο πλάτος 600 σε οριζόντιο προσανατολισμό(sw6000dp-land), 7” tablets σε landscape. Τέλος μένουν όλα τα υπόλοιπα, ο κατάλογος layout, δηλαδή κινητά με μικρότερες οθόνες και 7” tablets σε κάθετο προσανατολισμό. Και οι δύο αυτές κατηγορίες χρησιμοποιούν την 1-pane διάταξη.

- res/
  - layout/
    - activity_mail.xml
  - layout-sw600dp-land/
    - activity_mail.xml
  - layout-sw720dp/
    - activity_mail.xml

Για τις παλιότερες εκδόσεις θα πρέπει να προσθέσουμε και τους καταλόγους με τις κατηγορίες μεγεθών αντικαθιστώντας τα `sw600dp` και `sw720dp` με `large` και `xlarge` αντίστοιχα.

3.5 ActionBar

Η ActionBar είναι ένα χαρακτηριστικό παράθυρου που προσδιορίζει τη θέση του χρήστη στο πρόγραμμα, του παρέχει ενέργειες που μπορεί να εκτελέσει και μεθόδους πλοήγησης. Εισήχθηκε για πρώτη φορά στην έκδοση 3.0 αλλά είναι διαθέσιμη προς τα πίσω μέχρι και την 2.1. Είναι ένα πάνελ πλοήγησης το οποίο αντικαθιστά την παλιότερη γραμμή τίτλου στο πάνω μέρος της οθόνης. Η ActionBar “επισημοποίησε” ένα κοινό πρότυπο σχεδίασης που προσφέρει στους χρήστες ένα οικείο περιβάλλον μεταξύ διαφορετικών εφαρμογών το οποίο το σύστημα προσαρμόζει για διαφορετικές διαμορφώσεις οθόνης.

Εικόνα 19: Τυπική ActionBar

Η ActionBar είναι ενεργοποιημένη εξ ορισμού για τις εφαρμογές που προορίζονται για εκδόσεις άνω της 3.0 και χρησιμοποιούν το θέμα `Holo`. Για την εφαρμογή μας όμως θα πρέπει να χρησιμοποιήσουμε την βιβλιοθήκη υποστήριξης και να δηλώσουμε ρητά την χρήση της.

3.5.1 Προσθέτοντας την ActionBar

Για να προσθέσουμε την ActionBar στην δραστηριότητα μας αυτή θα πρέπει κατ’ αρχήν να επεκτείνει την κλάση `ActionBarActivity`. Θα πρέπει επίσης να ορίσουμε το θέμα της εφαρμογής μας ή της δραστηριότητας να χρησιμοποιεί ή να επεκτείνει ένα από τα βασικά θέματα `Theme.AppCompat.Light`, `Theme.AppCompat.Light.DarkActionBar`, `Theme.AppCompat.Dark`.

```
<application
```

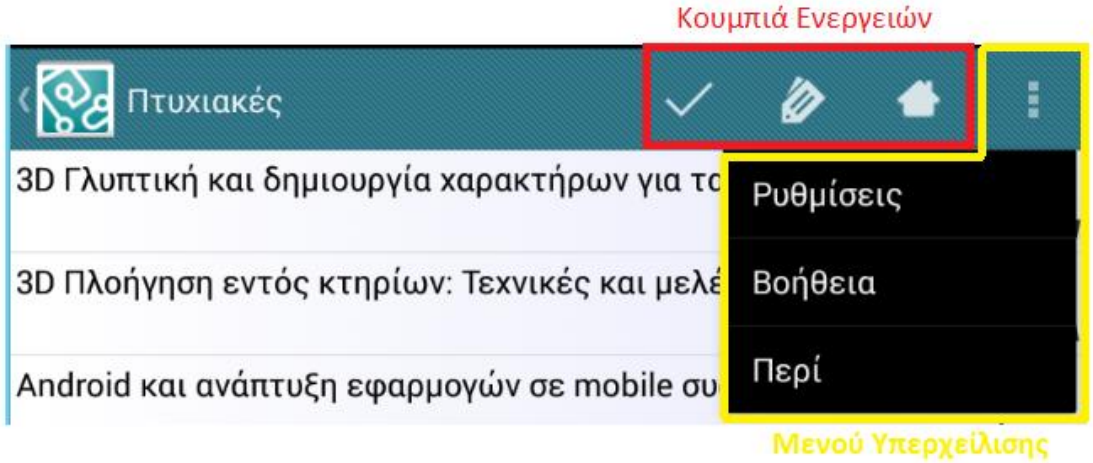
```
        android:name="com.nkyrim.itapp.ITapp"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/Theme.AppCompat.Light.DarkActionBar"
        ...
</application>
```

Ο τίτλος που θα εμφανίζεται στην ActionBar είναι ο τίτλος που έχουμε ορίσει για την δραστηριότητά μας και το εικονίδιο στην αριστερή μεριά είναι το εικονίδιο της εφαρμογής.

3.5.1.1 Προσθέτοντας Action Items

Η δεξιά πλευρά του ActionBar χρησιμοποιείται για την εμφάνιση των «ενεργειών» και του αντίστοιχου «μενού υπερχείλισης». Στις εκδόσεις πριν της 3.0 οι συσκευές έπρεπε υποχρεωτικά να έχουν ένα πλήκτρο menu το οποίο χρησιμοποιούταν για την εμφάνιση επιλογών και λειτουργιών χωρίς να απαιτείται έτσι συνέχεια χώρος στην οθόνη, με κάθε δραστηριότητα να έχει το δικό της μενού. Μετά την έκδοση 3.0 το πλήκτρο έγινε πλέον προαιρετικό και οι επιλογές αντικαταστάθηκαν από τις ενέργειες και το μενού υπερχείλισης.

Οι επιλογές πλέον εμφανίζονται σαν μία από δύο κατηγορίες, «κουμπιά ενεργειών»(Action buttons) και επιλογές του «μενού υπερχείλισης»(overflow menu).



Εικόνα 20: Τύποι αντικειμένων ενεργειών(Action Items)

Για να ορίσουμε της ενέργειες θα πρέπει πρώτα να δημιουργήσουμε ένα XML αρχείο στον υποκατάλογο menu των πόρων της εφαρμογής μας. Ο παρακάτω κώδικας αποτελεί το βασικό μενού της εφαρμογής και είναι κομμάτι ουσιαστικά κάθε δραστηριότητας. Αποτελείται από 4 αντικείμενα ενεργειών(“action items”) και το καθένα ορίζει κάποια βασικά χαρακτηριστικά.

```
res/menu/basic.xml
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:itapp="http://schemas.android.com/apk/res-auto">

    <item
        android:id="@+id/home"
        android:icon="@drawable/ic_menu_home"
        android:orderInCategory="97"
        android:title="@string/home"
        itapp:showAsAction="ifRoom"></item>
```

```
    <item
        android:id="@+id/settings"
        android:icon="@drawable/ic_menu_settings"
        android:orderInCategory="98"
        android:title="@string/settings"></item>
    <item
        android:id="@+id/help"
        android:icon="@drawable/ic_menu_help"
        android:orderInCategory="99"
        android:title="@string/help"></item>
    <item
        android:id="@+id/about"
        android:icon="@drawable/ic_menu_about"
        android:orderInCategory="100"
        android:title="@string/about"></item>
</menu>
```

Ανάλυση χαρακτηριστικών του στοιχείου `item`:

- `android:id`: Ορίζει το id με το οποίο θα μπορούμε να χρησιμοποιήσουμε το συγκεκριμένο menu item στον κώδικα μας.
- `android:icon`: Ορίζει το εικονίδιο που θα έχει αν εμφανίζεται σαν κουμπί ενεργειών(ή στο μενού των παλαιότερων συσκευών).
- `android:title`: Ο τίτλος του item είτε εμφανίζεται σαν κουμπί είτε σαν επιλογή στο μενού υπερχείλισης. Θα πρέπει λοιπόν να τον ορίζουμε πάντα.
- `android:orderInCategory`: Ένας θετικός ακέραιος οποίος ορίζει την προτεραιότητα εμφάνισης του συγκεκριμένου item. Όσο μικρότερος ο αριθμός τόσο μεγαλύτερη η προτεραιότητα.
- `itapp:showAsAction`: Εδώ να τονίσουμε πρώτα ότι χρησιμοποιείτε ο δικός μας ονοματοχώρος `itapp` λόγο της χρήσης της βιβλιοθήκης υποστήριξης. Η ιδιότητα `showAsAction` ορίζει την χρήση του συγκεκριμένου item σαν κουμπί ενεργειών. Μπορεί να πάρει μια από τις τιμές "ifRoom", "never", "always" οι οποίες ορίζουν πότε θα φαίνεται σαν κουμπί και την τιμή "withText" για τον αν θα εμφανίζει και τον τίτλο. Πολλαπλές τιμές χωρίζονται με «|», πχ “always|withText”.

Εδώ ορίζουμε χαμηλή προτεραιότητα καθώς αυτές είναι οι λιγότερο σημαντικές επιλογές και θέλουμε να μπουν χαμηλά στο μενού υπερχείλισης. Με το item `home` να εμφανίζεται σαν κουμπί μόνο αν υπάρχει χώρος.

Υπάρχουν και άλλες ιδιότητες τις οποίες όμως δεν θα αναφέρουμε εδώ. Να σημειώσουμε επίσης πως το στοιχείο `item` μπορεί να περιέχει ένα άλλο στοιχείο `menu` με τα δικά του `items` δημιουργώντας έτσι μια ιεραρχία επιλογών με υπομενού.

Στη δραστηριότητα μας τώρα θα πρέπει να υπερφορτώσουμε την μέθοδο `onCreateOptionsMenu()` για να προσθέσουμε τις ενέργειες με τη χρήση του `MenuInflater` της δραστηριότητας.

```
public boolean onCreateOptionsMenu(Menu menu) {
      MenuInflater inflater = getMenuInflater();
      Inflater.inflate(R.menu.basic, menu);
      return true;
}
```

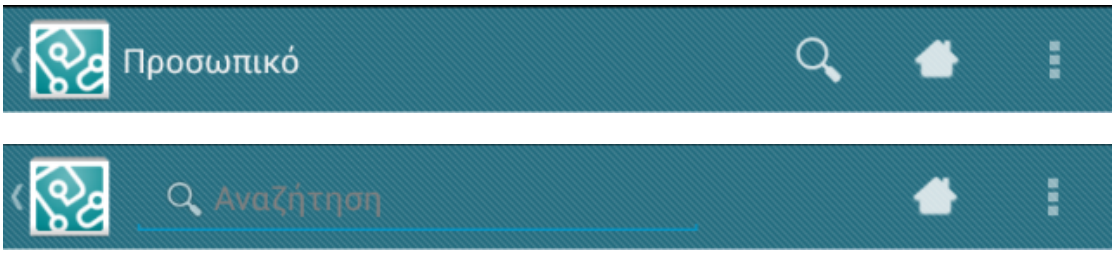
Τώρα θα πρέπει να ορίσουμε και την ουσιαστική δουλειά που θα κάνουν οι συγκεκριμένες επιλογές. Για αυτό το σκοπό θα υπερβούμε την μέθοδο `onOptionsItemSelected()`. Όταν ο χρήστης επιλέξει μια ενέργεια το σύστημα καλεί τη μέθοδο `onOptionsItemSelected()` της δραστηριότητα μας. Χρησιμοποιώντας το `MenuItem` που πέρασε με αυτή τη μέθοδο μπορούμε να προσδιορίσουμε την ενέργεια καλώντας την `getItemId()` η οποία επιστρέφει το μοναδικό αναγνωριστικό που παρέχεται από την `id` ιδιότητα του ώστε να εκτελεστούν οι κατάλληλες εντολές. Η μέθοδος θα πρέπει να επιστρέψει true ή false αναλόγως με το αν θα χειριστεί το γεγονός η θα αφήσει την εκτέλεση να συνεχίσει χωρίς να προβεί σε κάποια ενέργεια.

```
public boolean onOptionsItemSelected(MenuItem item) {
	switch (item.getItemId()) {
	case R.id.home:
		intent = new Intent(this,MainActivity.class);
		intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
		startActivity(intent);
		return true;
	case R.id.settings:
		Intent intent = new Intent(this, PreferencesActivity.class);
		startActivity(intent);
		return true;
	case R.id.about:
		intent = new Intent(this, AboutActivity.class);
		startActivity(intent);
		return true;
	...
	...
	default:
		return super.onOptionsItemSelected(item);
	}
}
```

Εδώ να προσθέσουμε πως τα Fragments συμμετέχουν και αυτά στην ActionBar και μπορούν και αυτά να προσθέσουν τις δικές τους ενέργειες παρέχοντας μας έτσι τις κατάλληλες ενέργειες μόνο όταν είναι απαραίτητες, όταν δηλαδή το Fragment αποτελεί μέρος της δραστηριότητας μας. Χρησιμοποιούν τις ίδιες ακριβώς μεθόδους με τις δραστηριότητες με τον ίδιο τρόπο. Ωστόσο, η δραστηριότητα έχει την ευκαιρία να χειριστεί την περίπτωση πρώτα, οπότε το σύστημα καλεί πρώτα την `onOptionsItemSelected()` της δραστηριότητας πριν από την κλήση της ίδιας μεθόδου για το `Fragment`. Για να εξασφαλιστεί ότι όλα τα `Fragments` στη δραστηριότητα θα έχουν την ευκαιρία να χειριστούν το γεγονός, χρησιμοποιούμε πάντα την κλήση στην υπερκλάση ως προεπιλεγμένη συμπεριφορά αντί να επιστρέφουμε `false` όταν δεν χειριζόμαστε το στοιχείο.

3.5.1.2 Προσθέτοντας Action Views

Ένα action view είναι ένα στοιχείο ελέγχου το οποίο εμφανίζεται στην Actionbar και αντικαθιστά ένα κουμπί. Προσφέρει πρόσθετη λειτουργικότητα στη δραστηριότητα μας χωρίς την ανάγκη να προσθέσουμε Fragments ή να αλλάξουμε τη διάταξη ή την ίδια την Actionbar. Τα στοιχεία μπορούν να αποτελούν εξ' αρχής και μόνιμα μέρος της ActionBar συνήθως όμως θα θέλουμε να συμπτύσσονται σε ένα κουμπί και να εμφανίζονται όταν τα επιλέγουμε ώστε να μη καταλαμβάνουν χώρο ο οποίος θα μπορούσε να χρησιμοποιηθεί για να εμφανίσει επιπλέον action items ή views.



Εικόνα 21: Συμπτυγμένο και επεκταμένο Action View

Για να προσθέσουμε ένα action view θα πρέπει να προσθέσουμε κάποιες επιπλέον ιδιότητες στο στοιχείο item του μενού μας.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:itapp="http://schemas.android.com/apk/res-auto" >

    <item
        android:id="@+id/search"
        android:icon="@drawable/ic_menu_search"
        android:orderInCategory="1"
        android:title="@string/add"
        itapp:actionViewClass="android.support.v7.widget.SearchView"
        itapp:showAsAction="always|collapseActionView">
    </item>

</menu>
```

Για να προσθέσουμε το `View` μας προσθέτουμε την ιδιότητα `actionViewClass`, εάν θέλουμε να χρησιμοποιήσουμε ένα `View` ή `actionLayout`, εάν θέλουμε να χρησιμοποιήσουμε ένα αρχείο διάταξης. Εδώ προσθέτουμε το στοιχείο `SearchView` το οποίο είναι ένα πεδίο κειμένου προορισμένο για αναζήτηση.

Η δεύτερη ιδιότητα είναι η `showAsAction` την οποία είδαμε και παραπάνω, εδώ όμως προσθέτουμε την τιμή `collapseActionView` που ορίζει πως το `View` μας θα πρέπει να συμπτυχθεί σε ένα κουμπί. Επίσης έχουμε επιλέξει να είναι πάντα ορατό ακόμα όμως και αν έμπαινε στο μενού υπερχείλισης όταν το επιλέξει ο χρήστης το `View` θα γίνει ορατό στην `ActionBar`.

Μπορούμε επίσης να τροποποιήσουμε το View μας, να προσθέσουμε event listeners καθώς και να χειριστούμε την επέκταση και σύμπτυξη του κατά την εκτέλεση της `onCreateOptionsMenu()`. Εδώ ορίζουμε listeners για την εισαγωγή ερωτημάτων στο `SearchView` ώστε να γίνεται αμέσως η αναζήτηση καθώς επίσης και καθαρισμός της αναζήτησης όταν το `View` συμπτυχθεί. Να σημειώσουμε επίσης την χρήση της κλάσης `MenuItemCompat` για λόγους προς τα πίσω συμβατότητας, στις νεότερες εκδόσεις οι λειτουργίες περιλαμβάνονται στην `MenuItem`.

```
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
  inflater.inflate(R.menu.staff_list, menu);
  final MenuItem si = menu.findItem(R.id.search);
  final SearchView sv = (SearchView) MenuItemCompat.getActionView(si);

  sv.setQueryHint(getString(R.string.search));
  sv.setOnQueryTextListener(new OnQueryTextListener() {
    public boolean onQueryTextChange(String arg0) {
```

```
      if (adapter != null) adapter.getFilter().filter(sv.getQuery());
      return true;
    }

    public boolean onQueryTextSubmit(String arg0) {
      if (adapter != null) adapter.getFilter().filter(sv.getQuery());
      return true;
    }
  });

  MenuItemCompat.setOnActionExpandListener(si, new
    MenuItemCompat.OnActionExpandListener() {

      public boolean onMenuItemActionCollapse(MenuItem arg0) {
        if (adapter != null) adapter.getFilter().filter("");
        return true;
      }

       public boolean onMenuItemActionExpand(MenuItem arg0) {
        // Do nothing here...
        return true;
      }
  });

  super.onCreateOptionsMenu(menu, inflater);
}
```

Μπορούμε επίσης να ορίσουμε, να επεκτείνουμε και να συμπτύξουμε action views στον κώδικα μας με την χρήση μεθόδων της κλάσης `MenuItemCompat`. Η μέθοδος `setActionView()` ορίζει ένα `View` το οποίο θα αντικαταστήσει ένα menu item. Οι μέθοδοι `expandActionView` και `collapseActionView` επεκτείνουν και συμπτύσσουν το menu item.

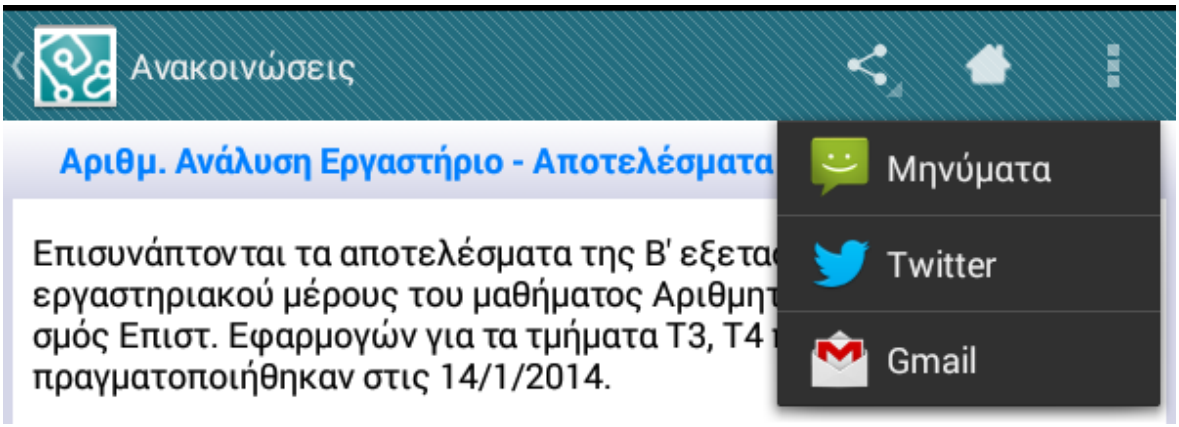
Επειδή το σύστημα επεκτείνει το action view όταν ο χρήστης επιλέγει την ενέργεια δεν χρειάζεται να χειριστούμε το γεγονός στην `onOptionsItemSelected()`. Το σύστημα εξακολουθεί να την καλεί αλλά αν επιστρέψει true, που δείχνει ότι έχουμε χειριστεί το γεγονός αντ' αυτού, τότε το `View` δεν θα επεκταθεί.

Το σύστημα συμπτύσσει επίσης το View όταν ο χρήστης πατήσει το πλήκτρο «Πίσω» ή το κουμπί «Πάνω» που θα δούμε παρακάτω.

3.5.1.3 Προσθέτοντας Action Providers

Στην έκδοση 4.0 προστέθηκε ακόμη ένα είδος αντικειμένων που μπορούμε να προσθέσουμε. Είναι οι `ActionProviders` οι οποίοι προσφέρουν λειτουργικότητα παρόμοια με τα action views ενθυλακώνουν όμως τόσο την εμφάνιση όσο και τη συμπεριφορά του αντικειμένου.

Μπορούμε να δημιουργήσουμε το δικό μας `action provider` με την επέκταση της κλάσης `ActionProvider`, αλλά το Android παρέχει κάποιους έτοιμους όπως ο `ShareActionProvider` για την κοινοποίηση πληροφοριών με εφαρμογές που υποστηρίζουν αποστολή μηνυμάτων τον οποίο χρησιμοποιούμε και στην εφαρμογή μας για την κοινοποίηση των ανακοινώσεων.



Εικόνα 22: Χρήση του ShareActionProvider

Όπως για κάθε αντικείμενο που θέλουμε να προσθέσουμε στην ActionBar ορίζουμε το menu αρχείο μας.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:itapp="http://schemas.android.com/apk/res-auto" >
    <item
        android:id="@+id/action_share"
        android:title="@string/share"
        itapp:showAsAction="ifRoom"
itapp:actionProviderClass="android.support.v7.widget.ShareActionProvider"
    />
</menu>
```

Όπως βλέπουμε δεν ορίζουμε εικονίδιο για το αντικείμενο μας αφού όπως είπαμε ο `ActionProvider` είναι υπεύθυνος για την εμφάνιση και συμπεριφορά, ορίζουμε όμως τον τίτλο τον οποίο θα έχει. Προσθέσαμε επίσης την ιδιότητα `actionProviderClass` η οποία ορίζει την κλάση του Provider μας.

Το μόνο που μένει είναι να περάσουμε τα δεδομένα πάνω στα οποία θα ενεργήσει ο Provider, στη συγκεκριμένη περίπτωση να ορίσουμε το `Intent` το οποίο έχουμε δημιουργήσει από πριν, και το οποίο θα μεταδώσει.

```
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
  inflater.inflate(R.menu.bulletin_details, menu);

  // Set up ShareActionProvider's share intent
  MenuItem share = menu.findItem(R.id.action_share);
  shareProvider = (ShareActionProvider)MenuItemCompat.getActionProvider(share);
  shareProvider.setShareIntent(shareIntent);

  super.onCreateOptionsMenu(menu, inflater);
}
```

Επειδή κάθε `ActionProvider` καθορίζει τη δικιά του συμπεριφορά δεν χρειάζεται να χειριζόμαστε την ενέργεια στην μέθοδο `onOptionsItemSelected()` η οποία δεν πρόκειται να κληθεί καν αν ο `ActionProvider` παρέχει υπομενού επιλογών.

3.5.2 Πλοήγηση με τη χρήση του εικονιδίου της εφαρμογής

Μπορούμε να ορίσουμε το εικονίδιο της εφαρμογής μας στην ActionBar να λειτουργεί σαν ένα «Επάνω» κουμπί το οποίο θα μας μεταφέρει την μητρική δραστηριότητα της τρέχουσας. Όχι

απαραίτητα στην προηγούμενη στην οποία γυρνάμε πάντα με το πλήκτρο «Πίσω» αλλά σε μια δραστηριότητα την οποία έχουμε ορίσει στο manifest.

Εικόνα 23: Πλήτρο "Επάνω"(4.3+ μαζί με τον τίτλο)

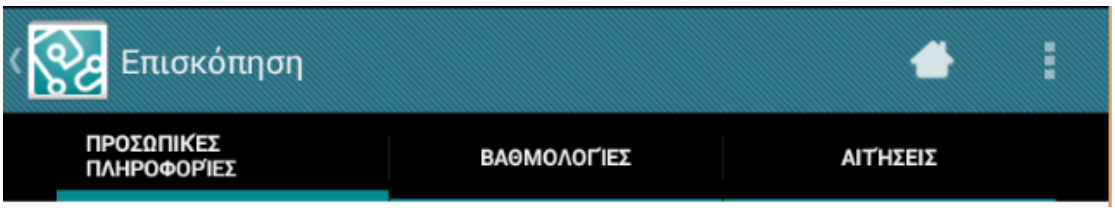
Θα πρέπει να ορίσουμε αρχικά μια υπάρχουσα δραστηριότητα ως μητρική της δραστηριότητας που θέλουμε στο manifest της εφαρμογής μας. Εδώ ορίζουμε την `MainActivity` ως θυγατρική της `HydraActivity`.

```
<application
  ...
  <activity
    android:name="com.nkyrim.itapp.ui.activities.MainActivity"
    android:configChanges="orientation|keyboardHidden|screenSize"
    android:label="@string/title_activity_main"
    android:launchMode="standard" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
             </intent-filter>
    </activity>
    <activity
        android:name="com.nkyrim.itapp.ui.activities.HydraActivity"
        android:label="@string/title_activity_hydra"
        android:parentActivityName="com.nkyrim.itapp.ui.activities.MainActivity">
        <meta-data
          android:name="android.support.PARENT_ACTIVITY"
          android:value="com.nkyrim.itapp.ui.activities.MainActivity"
         />
  </activity>
</application>
```

Κάνουμε τη δήλωση ουσιαστικά δύο φορές. Μία με την ιδιότητα `parentActivityName` για της εκδόσεις άνω της 4.1 και μία με το στοιχείο `metadata` για τις παλιότερες εκδόσεις.

3.5.3 Πλοήγηση με χρήση Tabs

Τα Tabs στην Actionbar καθιστούν εύκολη για τους χρήστες την εξερεύνηση και εναλλαγή μεταξύ διαφορετικών οθονών στην εφαρμογή μας. Οι καρτέλες που παρέχονται από το ActionBar είναι ιδανικές επειδή μπορούν να προσαρμοστούν σε διαφορετικά μεγέθη οθόνης. Για παράδειγμα όταν η οθόνη είναι ευρέα οι καρτέλες εμφανίζονται στη γραμμή παράλληλα με τα κουμπιά ενεργειών, πχ σε ένα tablet, ενώ όταν σε μια στενή οθόνη εμφανίζονται σε μία ξεχωριστή μπάρα, γνωστή ως "stacked ActionBar". Σε ορισμένες περιπτώσεις το σύστημα θα δείξει αντί για tabs μία drop-down λίστα για να εξασφαλιστεί η καλύτερη προσαρμογή.



Εικόνα 24: Τρόποι προσαρμογής της ActionBar

Για το παράδειγμα μας θα δούμε τη χρήση των tabs σε συνδυασμό με τα στοιχεία `ViewPager` και `PagerAdapter` τα οποία θα προσδώσουν την δυνατότητα εναλλαγής των tabs με κινήσεις, αριστερό και δεξί swipe. Θα χρησιμοποιήσουμε επίσης και τα `Fragments` που υλοποιούν τις υπηρεσίες και χρησιμοποιούνται στις αντίστοιχες δραστηριότητες. Η χρήση του `PagerAdapter` και πιο συγκεκριμένα ενός `FragmentStatePagerAdapter` θα διευκολύνει το χειρισμό των `Fragments` καθώς θα είναι αυτό υπεύθυνο για τη διαχείριση τους.

Κατ' αρχήν στο αρχείο διάταξης της δραστηριότητας προσθέτουμε το στοιχείο `ViewPager` το οποίο είναι και το μόνο που χρειαζόμαστε.

```
<android.support.v4.view.ViewPager
        android:id="@+id/pager"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
```

Δεν υπάρχει κάτι το ιδιαίτερο στον ορισμό του, το στοιχείο θα καταλαμβάνει όλο το ύψος και πλάτος και έχει το id `pager`.

Έπειτα επεκτείνουμε την κλάση `FragmentStatePagerAdapter` σαν εσωτερική ιδιωτική κλάση της δραστηριότητας μας.

```
private class OverviewPagerAdapter extends FragmentStatePagerAdapter {
      private String[] titles =
            getResources().getStringArray(R.array.overview_titles);

      public OverviewPagerAdapter(FragmentManager fm) {
            super(fm);
      }

      public int getCount() {
            return 3;
      }

      public Fragment getItem(int i) {
            Fragment fragment = null;
            if (i == 0) fragment = new PersonalInfoFragment();
            else if (i == 1) fragment = new GradesFragment();
            else if (i == 2) fragment = new RequestListFragment();

            return fragment;
      }

      public CharSequence getPageTitle(int position) {
            return titles[position];
      }
}
```

Η πρώτη εντολή ανακτά τους τίτλους που έχουμε ορίσει σε ένα αρχείο πόρων σε ένα πεδίο της κλάσης μας. Ο δομητής παίρνει σαν παράμετρο τον `FragmentManager` που θα του περάσουμε από την δραστηριότητα μας. Η μέθοδος `getCount()` επιστρέφει τον συνολικό

αριθμό των tabs. Η μέθοδος `getItem()` επιστρέφει το ουσιαστικό περιεχόμενο δημιουργώντας το κατάλληλο `Fragment`, ανάλογα με τη τρέχουσα θέση. Η `getPageTitle()` επιστρέφει τον τίτλο πάλι ανάλογα με τη θέση.

Τέλος η κλάση της δραστηριότητας μας θα πρέπει να υλοποιήσει το interface `ActionBar.TabListener` ώστε η δραστηριότητα να «ακούει» για την περίπτωση που ο χρήστης επιλέγει απ' ευθείας κάποιο tab και δεν κάνει swipe σε αυτό.

```
public class PithiaOverviewActivity extends BaseActivity implements
ActionBar.TabListener {

private ViewPager pager;
private OverviewPagerAdapter pagerAdapter;
private ActionBar actionBar;

@Override
protected void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	setContentView(R.layout.activity_pithia_overview);
	actionBar = getSupportActionBar();
	actionBar.setDisplayHomeAsUpEnabled(true);

	pager = (ViewPager) findViewById(R.id.pager);

	pagerAdapter = new OverviewPagerAdapter(getSupportFragmentManager());

	actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
	pager.setAdapter(pagerAdapter);
	pager.setOnPageChangeListener(
		new ViewPager.SimpleOnPageChangeListener() {

		public void onPageSelected(int position) {
			actionBar.setSelectedNavigationItem(position);
		}
	});

	// Προσθήκη tab για κάθε τμήμα
	for (int i = 0; i < pagerAdapter.getCount(); i++) {
		ActionBar.Tab tab = actionBar.newTab();
		tab.setText(pagerAdapter.getPageTitle(i));
		tab.setTabListener(this);
		actionBar.addTab(tab);
	}
}

public void onTabReselected(Tab tab, FragmentTransaction ft) {
	// Do nothing
}

public void onTabSelected(Tab tab, FragmentTransaction ft) {
	// Μετάβαση στο κατάλληλο τμήμα όταν ο χρήστης επιλέγει το ανάλογο tab
	if (pager != null) pager.setCurrentItem(tab.getPosition());
}

public void onTabUnselected(Tab tab, FragmentTransaction ft) {
	// Do nothing
}
```

3.6 Στοιχεία διαλόγου και Toasts

Τα πλαίσια διαλόγου είναι ένα κοινό στοιχείο UI σε διάφορους τύπους εφαρμογών desktop, web ή mobile. Χρησιμοποιείται για να βοηθήσει τους χρήστες να απαντήσουν σε ερωτήσεις, να κάνουν επιλογές, να επιβεβαιώσουν ενέργειες και εμφανίζουν μηνύματα προειδοποίησης ή

σφαλμάτων. Τα παράθυρα διαλόγου στο Android είναι μερικώς διαφανή στοιχεία UI που επισκιάζουν μερικώς το γραφικό περιβάλλον από το οποίο κλήθηκαν.

Υπάρχουν τρεις τρόποι για να υλοποιήσουμε ένα πλαίσιο διαλόγου στο Android:

- Χρησιμοποιώντας την κλάση `Dialog`(ή τις επεκτάσεις της): Εκτός από την γενικής χρήσης κλάση `AlertDialog` το Android περιλαμβάνει μια σειρά από κλάσεις που επεκτείνουν την `Dialog`. Κάθε μια έχει σχεδιαστεί για να παρέχει συγκεκριμένη λειτουργικότητα. Τα στοιχεία διαλόγου δημιουργούνται με αυτόν τον τρόπο διαχειρίζονται εξ' ολοκλήρου μέσα στο πλαίσιο της καλούσας δραστηριότητας, έτσι δεν χρειάζεται να δηλωθούν στο manifest.
- Δραστηριότητες με θέμα πλαισίου διαλόγου: Μπορούμε να εφαρμόσουμε ένα θέμα πλαισίου διαλόγου σε τυπική δραστηριότητα για να δώσει την εμφάνιση ενός παράθυρου διαλόγου.
- `Toast`: Τα Toasts είναι ειδικά non-modal πλαίσια μηνυμάτων, που χρησιμοποιούνται συχνά από τους δέκτες εκπομπών και τις υπηρεσίες για να ενημερώσουν τους χρήστες για τα γεγονότα που συνέβησαν στο παρασκήνιο αλλα και από τις δραστηριότητες για απλές περιπτώσεις.

3.6.1 Δημιουργία πλαισίων διαλόγου

Αν και η κλάση `Dialog` είναι η βασική κλάση για να δημιουργήσουμε πλαίσια διαλόγου προτείνεται να αποφεύγεται η απ' ευθείας χρήση της. Αντ' αυτού μπορούμε να χρησιμοποιήσουμε την γενικού σκοπού `AlertDialog` με την οποία δημιουργούμε τα δικά μας πλαίσια καθώς επίσης και τις συγκεκριμένες υποκλάσεις `CharacterPickerDialog`, `DatePickerDialog`, `TimePickerDialog` και `ProgressDialog`.

3.6.1.1 Χρήση Dialog και υποκλάσεων

Με τη χρήση της κλάσης `AlertDialog.Builder` «χτίζουμε» το πλαίσιο μας ανάλογα με τις ανάγκες μας. Μπορούμε να προσθέσουμε τον τίτλο, ένα μήνυμα κειμένου αν θέλουμε κάτι απλό ή μπορούμε να ορίσουμε το δικό μας αρχείο διάταξης σαν περιεχόμενο του διαλόγου, να ορίσουμε μια λίστα μονής ή πολλαπλής επιλογής καθώς και να ορίσουμε μέχρι 3 κουμπιά επιλογών.

Εδώ δημιουργούμε ένα πλαίσιο το οποίο θα παρουσιάσει την λίστα των κατηγοριών ανακοινώσεων. Στο παράδειγμα μας προσθέτουμε τον τίτλο, την λίστα με τις κατηγορίες και τον Listener που θα διαχειριστεί την επιλογή μας. Δημιουργούμε και δείχνουμε απ' ευθείας το πλαίσιο διαλόγου.

```
AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

builder.setTitle(R.string.select_board)
       .setSingleChoiceItems(R.array.bulletin_boards, sf,
                             new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int which) {
                  selectedFilters.clear();
                  selectedFilters.add(which);
                  setListFilter();
                  dialog.dismiss();
      }
});
```

```
builder.create().show();
```

Εδώ βλέπουμε τη χρήση ενός `ProgressDialog` το οποίο εμφανίζεται κατά την διάρκεια της σύνδεσης σε κάποια υπηρεσία.

```
ProgressDialog pd = new ProgressDialog(MainActivity.this);
pd.setTitle(R.string.login);
pd.setIcon(R.drawable.ic_dialog_info);
pd.setIndeterminate(true);
pd.setMessage(getString(R.string.wait));
pd.setOnCancelListener(new DialogInterface.OnCancelListener() {
      public void onCancel(DialogInterface dialog) {
            cancel(true);
      }
});
pd.show();
```

3.6.1.2 Χρήση Δραστηριοτήτων

Όπως είπαμε μπορούμε να ορίσουμε μια δραστηριότητα ώστε να έχει την εμφάνιση ενός πλαισίου διαλόγου ορίζοντας το θέμα στο manifest εκεί που έχουμε ορίσει και την ίδια την δραστηριότητα. Πέρα από την εμφάνιση όμως ο τρόπος λειτουργίας και η διαχείριση της δεν αλλάζει σε σχέση με μια τυπική δραστηριότητα.

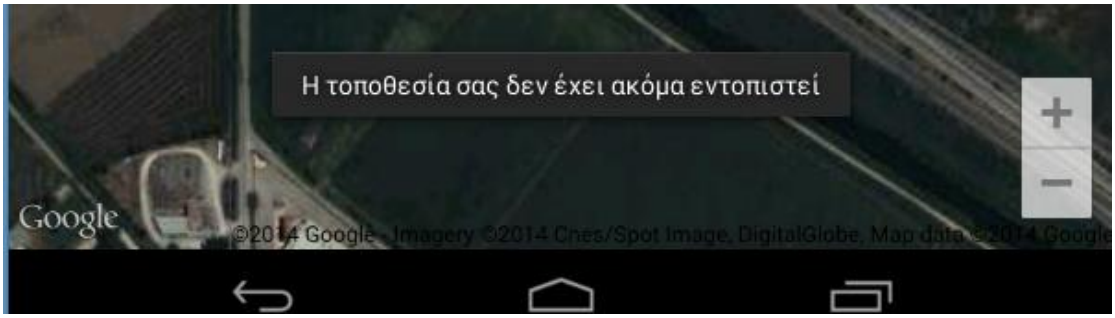
Εδώ ορίζουμε την δραστηριότητα υπεύθυνης για την σύνδεση στην εκάστοτε υπηρεσία. Το θέμα είναι δικό μας βασίζεται όμως στο θέμα διαλόγου ανάλογα με την έκδοση στην οποία τρέχει η εφαρμογή.

```
<activity
      android:name="com.nkyrim.itapp.ui.activities.LoginActivity"
      android:configChanges="orientation|keyboardHidden|screenSize"
      android:excludeFromRecents="true"
      android:theme="@style/DialogTheme" >
</activity>
```

3.6.1.3 Χρήση Toasts

Εάν το μόνο που επιθυμούμε είναι να δείξουμε ένα μήνυμα στον χρήστη χωρίς να θέλουμε να τον διακόψουμε και χωρίς να απαιτείται κάποια αλληλεπίδραση μαζί του χρησιμοποιούμε την κλάση `Toast`. Χρησιμοποιείτε και από τις υπηρεσίες για να ενημερώνουν τον χρήστη.

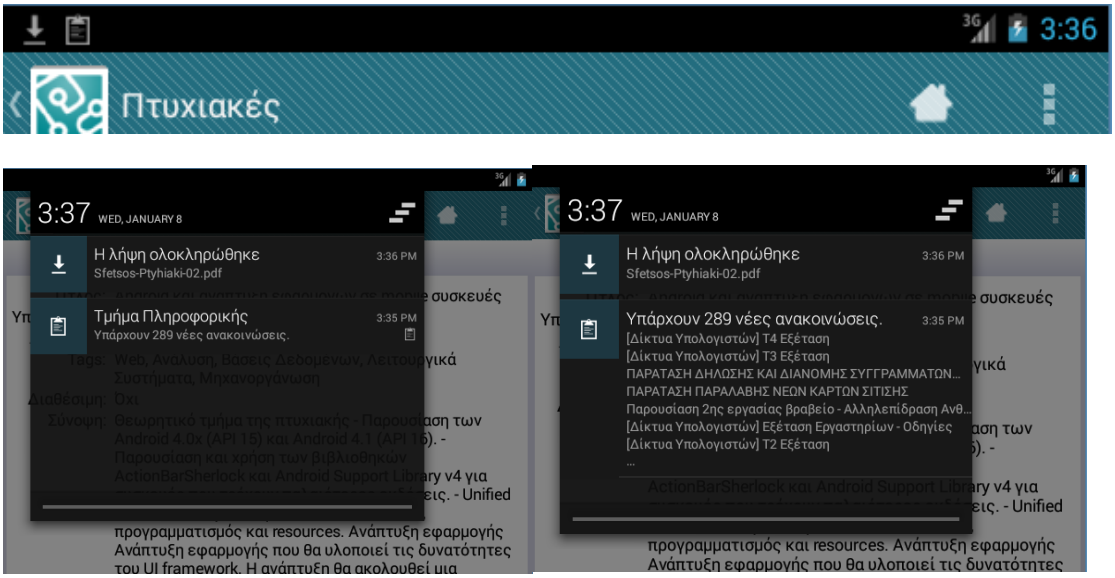
```
Toast.makeText(this, getMesg(), Toast.LENGTH_LONG).show();
```



Εικόνα 25: Χρήση Toast που ενημερώνει για την κατάσταση του εντοπισμού θέσης

3.7 Ειδοποιήσεις

Η ειδοποίηση(notification) είναι ένα μήνυμα που μπορεί να εμφανίσει η εφαρμογή στον χρήστη εκτός των κανονικό UI, έξω δηλαδή από τις δραστηριότητες μας. Όταν ζητάμε από το σύστημα να εμφανίσει μια ειδοποίηση αυτή εμφανίζεται για πρώτη φορά ως εικονίδιο στην περιοχή ειδοποιήσεων. Για να δει τις λεπτομέρειες της ειδοποίησης ο χρήστης ανοίγει το «συρτάρι» ειδοποιήσεων. Τόσο η περιοχή ειδοποιήσεων όσο και το συρτάρι ειδοποιήσεων είναι περιοχές που ελέγχονται από το σύστημα ότι ο χρήστης μπορεί να δει ανά πάσα στιγμή.



Εικόνα 26: Διάφορες καταστάσεις μιας ειδοποίησης

Οι ειδοποιήσεις χειρίζονται από τον Διαχειριστή Ειδοποιήσεων και έχουν τη δυνατότητα να:

- Εμφανίζουν ένα εικονίδιο στη γραμμή κατάστασης(περιοχή ειδοποιήσεων)
- Ενεργοποιούν τα φώτα/LEDs
- Ενεργοποιούν τη δόνηση του τηλέφωνου
- Στέλνουν ηχητικά σήματα(ringtones κλπ)
- Εμφανίσουν πρόσθετες πληροφορίες εντός του συρταριού
- Εκπέμπουν προθέσεις χρησιμοποιώντας αλληλεπιδραστικά στοιχεία ελέγχου μέσα από το συρτάρι ειδοποιήσεων

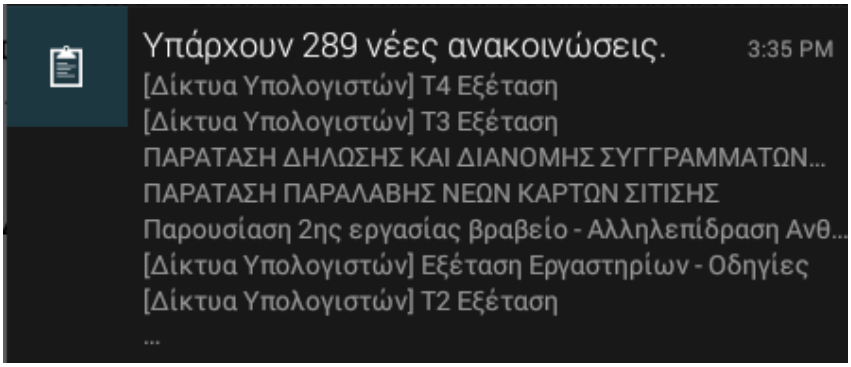
Οι ειδοποιήσεις είναι ο προτιμώμενος μηχανισμός για τα αόρατα στοιχεία της εφαρμογής, δέκτες εκπομπών, υπηρεσίες κλπ, για να ειδοποιούν τους χρήστες πως έχουν λάβει χώρα γεγονότα που μπορεί να απαιτούν την προσοχή τους.

3.7.1 Τύποι ειδοποιήσεων

Οι ειδοποιήσεις στο συρτάρι ειδοποιήσεων μπορεί να εμφανιστούν σε μία από τις δύο οπτικές μορφές, ανάλογα με την έκδοση και την κατάσταση του συρταριού:

- Κανονική προβολή: Η κανονική προβολή των κοινοποιήσεων στο συρτάρι κοινοποίησης.

- Μεγάλη προβολή: Μια μεγάλη προβολή που είναι ορατή όταν η ειδοποίηση επεκταθεί. Διατίθεται από την έκδοση 4.1 και μετά.



3.7.2 Δημιουργία και εμφάνιση ειδοποιήσεων

Αν και μπορούμε να δημιουργήσουμε απ' ευθείας `Notification` αντικείμενα είναι προτιμότερο να χρησιμοποιήσουμε το Notification Builder που εισήχθηκε με την έκδοση 3.0 αλλά είναι διαθέσιμος και για παλιότερες εκδόσεις μέσω της βιβλιοθήκης υποστήριξης. Με αυτό τον τρόπο μπορούμε όχι μόνο να δημιουργήσουμε πιο εύκολα τις ειδοποιήσεις μας αλλά και να αξιοποιήσουμε λειτουργίες που εισήχθηκαν στις νεότερες εκδόσεις διατηρώντας την συμβατότητα με τις παλιότερες εκδόσεις.

Στο παράδειγμα μας δημιουργούμε αρχικά την ειδοποίηση μας χρησιμοποιώντας ένα `NotificationCompat.Builder` αντικείμενο. Ορίζουμε τον τίτλο, το περιεχόμενο κείμενο που θα αναφέρει τον αριθμό των νέων ανακοινώσεων, τα εικονίδια, την πρόθεση που θα εκτελεστεί όταν ο χρήστης επιλέξει την ειδοποίηση και τέλος ορίζουμε να χρησιμοποιηθούν όλες οι εξ ορισμού ρυθμίσεις, πχ ήχος, δόνηση κλπ.

```
NotificationCompat.Builder builder=new NotificationCompat.Builder(this);
  builder.setContentTitle(getString(R.string.app_name))
         .setContentText(getString(R.string.noti_new_bulletins,
                         String.valueOf(count)))
         .setSmallIcon(R.drawable.ic_stat_news)
         .setLargeIcon(BitmapFactory.decodeResource(getResources(),
                       R.drawable.ic_stat_news))
         .setContentIntent(pintent)
        .setDefaults(Notification.DEFAULT_ALL);
```

Έπειτα ορίζουμε την «μεγάλη προβολή» που θα περιέχει μια λίστα με τις νεώτερες ανακοινώσεις και δημιουργούμε το τελικό αντικείμενο μας.

```
NotificationCompat.InboxStyle bigView = new
NotificationCompat.InboxStyle();

bigView.setBigContentTitle(
      getString(R.string.noti_new_bulletins,
      String.valueOf(count)));

for (String s : titles) {
      bigView.addLine(s);
}

builder.setStyle(bigView);

Notification noti = builder.build();
```

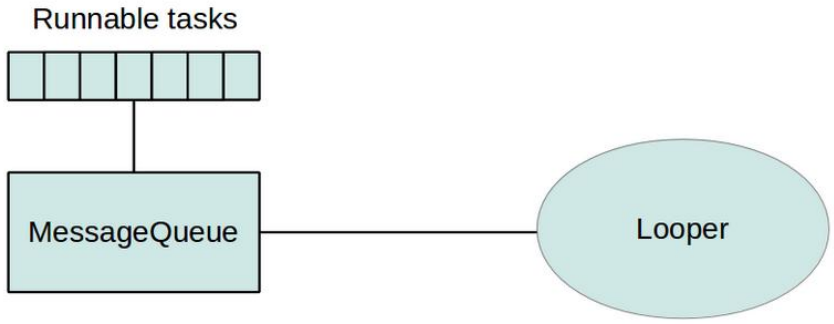
Για να την εμφανίσουμε τώρα θα πρέπει να χρησιμοποιήσουμε την υπηρεσία του συστήματος `NotificationManger και να δηλώσουμε ένα id`. Χρησιμοποιούμε ένα σταθερό id έτσι ώστε αν εμφανίσουμε μια νεότερη ειδοποίηση αυτή να αντικαταστήσει την προηγούμενη και να μη προσθέσει νέα. Επίσης κάθε ανακοίνωση έχει το δικό της σταθερό id ώστε πχ οι ειδοποιείσεις των ανακοινώσεων να μην αντικαθιστούν αυτές των λήψεων αρχείων.

```
NotificationManager nm = (NotificationManager)
                          getSystemService(NOTIFICATION_SERVICE);

nm.notify(Constants.NOTI_ID_BULLETINS, noti);
```

Κεφάλαιο 4: Εργασίες στο παρασκήνιο

Όπως κάθε σύγχρονη εφαρμογή έτσι και στις εφαρμογές του Android υπάρχει η απαίτηση να μπορούμε να εκτελούμε χρονοβόρες λειτουργίες διατηρώντας παράλληλα ένα γραφικό περιβάλλον το οποίο συνεχίζει να αποκρίνεται στα διάφορα συμβάντα ή εντολές εισόδου χωρίς να μπλοκάρει. Το Android υποστηρίζει τον τύπο πολυδιεργασίας με βάση τα Νήματα(Threads).

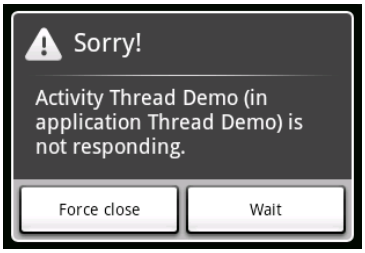
Το σύστημα διαχειρίζεται το γραφικό περιβάλλον καθώς και τις εντολές και τα συμβάντα εισόδου στο βασικό νήμα(main thread) ή UI νήμα όπως αποκαλείτε πολύ συχνά, εκτελώντας σειριακά τη μια εντολή μετά την άλλη. Όλα τα συμβάντα που πρέπει να τρέξουν στο UI νήμα συλλέγονται σε μια ουρά(`MessageQueue`) και επεξεργάζονται από ένα αντικείμενο της κλάσης `Looper`.



Εικόνα 27: Μέθοδος εκτέλεσης ενεργειών στο UI νήμα

Όλες οι χρονοβόρες εργασίες θα πρέπει να εκτελούνται ασύγχρονα μέσω τον διαφόρων δομών παράλληλης επεξεργασίας που μας παρέχει η πλατφόρμα και τα αποτελέσματα τους να στέλνονται στην ουρά ώστε να εμφανίζονται κατάλληλα.

Το Android επιβάλλει ένα μέγιστο χρόνο αντίδρασης των αιτήσεων. Αν μια δραστηριότητα δεν αντιδράσει μέσα σε 5 δευτερόλεπτα μετά από κάποια αλληλεπίδραση με το χρήστη, το σύστημα εμφανίζει τον διάλογο «Η εφαρμογή δεν αποκρίνεται». Από αυτό το παράθυρο διαλόγου, ο χρήστης μπορεί να επιλέξει να τερματίσει την εφαρμογή.



Εικόνα 28: Μήνυμα ANR

4.1 Νήματα και η κλάση Handler

Στο χαμηλότερο επίπεδο για την υποστήριξη της πολυδιεργασίας το Android μας παρέχει τα Νήματα(κλάση `Thread`) όπως αυτά χρησιμοποιούνται και στη Java καθώς και άλλες βοηθητικές κλάσεις που βρίσκονται στο πακέτο `java.util.concurrent`. Η χρήση όμως αποκλειστικά αυτών των κλάσεων έχει μειονεκτήματα όπως είναι ο συγχρονισμός με το UI νήμα και ο χειρισμός των αλλαγών στη διαμόρφωση του συστήματος(configuration changes). Για να ξεπεράσουμε αυτά τα προβλήματα μας βοηθάει η κλάση `Handler` η οποία αναλαμβάνει τον χειρισμό των «μηνυμάτων» που θέλουμε να στείλουμε. Χρησιμοποιείτε συνήθως μέσω των μεθόδων `post()` της κλάσης `View` και `runOnUiThread()` της `Activity` οι οποίες δέχονται ένα `Runnable` αντικείμενο σαν παράμετρο και το προσθέτουν στην ουρά εκτέλεσης.

4.2 Κλάση AsyncTask

Η κλάση `AsyncTask` είναι μία πολύ ισχυρή κλάση η οποία μας επιτρέπει να δημιουργήσουμε παραμετροποιήσημες εργασίες παρασκηνίου. Μας επιτρέπει να ενθυλακώσουμε όλη την διαδικασία δημιουργίας νημάτων και συγχρονισμού με το UI νήμα ενώ παράλληλα μας δίνει την δυνατότητα να αναφέρουμε και την πρόοδο των ενεργών εργασιών μας χωρίς να χρειάζεται να χειριστούμε μόνοι μας τα νήματα.

4.2.1 Ορισμός της κλάσης

Η `AsyncTask(android.os.AsyncTask<Params, Progress, Result>)` είναι μια generic abstract κλάση την οποία και θα πρέπει να επεκτείνουμε. Ορίζεται από τρείς generic τύπους:

1. `Params`: Ο τύπος των παραμέτρων που μπορούμε να περάσουμε κατά την έναρξη της εργασία.
2. `Progress`: Ο τύπος που θα χρησιμοποιηθεί για την αναφορά της προόδου.
3. `Result`: Ο τύπος του αποτελέσματος.

Δεν χρησιμοποιούνται όλοι οι τύποι πάντα από μια ασύγχρονη εργασία. Για να επισημάνουμε έναν τύπο ως αχρησιμοποίητο, χρησιμοποιούμε τον τύπο `Void`.

Και από 4 στάδια, οι μέθοδοι που θα εκτελεστούν κατά τη διάρκεια της εργασίας.

1. `onPreExecute`: Εκτελείτε στο UI νήμα πριν εκτελεστεί η εργασία. Αυτό το βήμα χρησιμοποιείται συνήθως για να ρυθμίσουμε τη εργασία, για παράδειγμα εμφανίζοντας μια μπάρα προόδου.
2. `doInBackground`: Εκτελείτε σε νήμα στο παρασκήνιο αμέσως μετά την `onPreExecute`. Εδώ ορίζεται και πραγματοποιείτε η ουσιαστική εργασία. Αυτή είναι και η μόνη μέθοδος που πρέπει να υπερβούμε υποχρεωτικά, όπως είναι λογικό άλλωστε.
3. `onProgressUpdate`: Εκτελείτε στο UI νήμα για να αναφέρουμε την πρόοδο(πχ με ένα `ProgressBar`) με απροσδιόριστο όμως τον ακριβή χρόνο εκτέλεσης της.
4. `onPostExecute`: Εκτελείτε στο UI νήμα μετά την ολοκλήρωση της `doInBackground`.

4.2.2 Αναφορά προόδου και ακύρωση εργασιών

Για να αναφέρουμε την πρόοδο της εργασίας μας χρησιμοποιούμε την μέθοδο `publishProgress(Progress...)` περνώντας σαν παραμέτρους αντικείμενα του τύπου

Progress που έχουμε δημιουργήσει κατά τη διάρκεια της εργασίας, μέσα δηλαδή στην doInBackground, μετά την οποία θα κληθεί η onProgressUpdate() στο UI νήμα.

Μια εργασία μπορεί να ακυρωθεί ανά πάσα στιγμή, με την κλήση της cancel. Η κλήση αυτής της μεθόδου θα προκαλέσει επόμενες κλήσεις προς την isCancelled() να επιστρέφουν αληθές. Μετά την επίκληση αυτής της μεθόδου, αντί της onPostExecute() θα εκτελείτε η onCancelled(), μετά την ολοκλήρωση της εργασίας μας. Για να εξασφαλιστεί ότι μια εργασία έχει ακυρωθεί το συντομότερο δυνατό, θα πρέπει να ελέγχουμε αν έχει ακυρωθεί ελέγχοντας την τιμή που επιστρέφει η isCancelled() σε τακτά χρονικά διαστήματα μέσα στην doInBackround(), σε ένα loop για παράδειγμα.

4.2.3 Κανόνες και σειρά εκτέλεσης εργασιών

Το αντικείμενο της εργασίας και η εκτέλεση του(μέθοδος execute) πρέπει να γίνει στο UI νήμα. Δεν πρέπει να γίνονται απευθείας κλήσεις στις μεθόδους των 4 σταδίων. Κάθε εργασία μπορεί να εκτελεστεί μόνο μια φορά, δεύτερη κλήση της execute θα πετάξει σφάλμα.

Όταν εισήχθηκε η AsyncTask οι εργασίες εκτελούνταν διαδοχικά σε ένα μοναδικό νήμα παρασκηνίου. Στην έκδοση 1.6 αυτό άλλαξε σε μια ομάδα νημάτων που επιτρέπει πολλαπλές εργασίες να λειτουργούν παράλληλα. Ξεκινώντας με την 3.0 οι εργασίες εκτελούνται ξανά σε ένα ενιαίο νήμα για να αποφεύγονται κοινά σφάλματα που προκαλούνται από την παράλληλη εκτέλεση. Παρόλα αυτά μπορούμε να επιτύχουμε πραγματικά παράλληλη εκτέλεση χρησιμοποιώντας την μέθοδο executeOnExecutor ορίζοντας σαν παράμετρο το AsyncTask.THREAD_POOL_EXECUTOR.

4.2.4 Μειονεκτήματα

Παρά τις πολλές δυνατότητες της και τις ευκολίες που μας παρέχει η AsyncTask δεν χειρίζεται αυτόματα τις αλλαγές στη διαμόρφωση του συστήματος. Θα πρέπει δηλαδή να χειριστούμε μόνοι μας την αναδημιουργία των Activities. Αυτό μπορούμε να το επιτύχουμε είτε ακυρώνοντας την εργασία και ξαναρχίζοντας την μόλις ξαναδημιουργηθεί το Activity είτε με τη χρήση των Fragments. Υπάρχει επίσης η μέθοδος onRetainNonConfigurationInstance η οποία μπορεί να βοηθήσει στη διατήρηση της εργασίας έχει όμως πλέον καταργηθεί από την έκδοση 3.0.

4.2.5 Παράδειγμα χρήσης

Υλοποίηση ενός AsyncTask για μια χρονοβόρα εργασία. Δεν χρησιμοποιούμε κάποιο παράδειγμα από την εφαρμογή ώστε να δείξουμε όλες τις λειτουργίες τις κλάσης ταυτόχρονα. Για λόγους συντομίας θεωρούμε ότι οι μέθοδοι πραγματοποιούν κάποια ουσιαστική εργασία. Οι generic τύποι είναι String για τις παραμέτρους και τύπο επιστροφής και Integer για την πρόοδο.

```
class CustomTask extends AsyncTask<Integer, Integer, String> {
```

Εμφανίζουμε την μπάρα προόδου πριν την έναρξη του task.

```
protected void onPreExecute() {
      showProgress(true);
}
```

Η εργασία μας χρησιμοποιώντας τις παραμέτρους που περνάμε κατά την εκκίνηση της(varargs, μεταβλητού πλήθους παράμετροι) .

```
protected String doInBackground(Integer... params) {
```

Εξομοιώνουμε μια επαναλαμβανόμενη διαδικασία αναφέροντας την πρόοδο σε κάθε επανάληψη και ελέγχοντας αν έχει ακυρωθεί το task. Σε πιθανή ακύρωση μπορούμε να διακόψουμε το βρόχο ώστε το υπόλοιπο task να ολοκληρωθεί, θα δούμε παρακάτω πως θα χειριστούμε το αποτέλεσμα.

```
        for (int i = 0 ; i < 100 ; i++) {
                doStuff(params[0], params[1]);
                publishProgress(i);

                if (isCancelled()) break; // return null or result
        }

        return "result";
}
```

Η μέθοδος που θα τρέξει στο UI νήμα ενημερώνοντας στη συγκεκριμένη περίπτωση την μπάρα προόδου που εμφανίσαμε στην αρχή.

```
        protected void onProgressUpdate(Integer... progress) {
                setProgressPercent(progress[0]);
        }
```

Μετά την ολοκλήρωση του task σταματάμε να δείχνουμε την πρόοδο και εμφανίζουμε το αποτέλεσμα.

```
        protected void onPostExecute(String result) {
                showProgress(false);
                showResult(result);
        }
```

Σε περίπτωση που ακυρώσουμε την εργασία μας μπορούμε αν θέλουμε να επεξεργαστούμε το (όποιο)τελικό αποτέλεσμα έχουμε. Εξ ορισμού η μέθοδος απορρίπτει οποιοδήποτε αποτέλεσμα.

```
        protected void onCancelled(String result) {
                showProgress(false);
                storeResult(result);
        }
}
```

Δημιουργία ενός `CustomTask`

```
CustomTask task = new CustomTask();
```

Κανονική(εξ ορισμού) εκτέλεση(Σειριακή / παράλληλη ανάλογα με την έκδοση)

```
task.execute();
```

Παράλληλη εκτέλεση σε εκδόσεις μεγαλύτερες της 3.0

```
task.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR);
```

Αίτηση ακύρωσης από τον κώδικα πελάτη, η Boolean παράμετρος καθορίζει αν θα διακοπεί η εργασία ή αν θα ακυρωθεί αφού όμως ολοκληρωθεί πρώτα, πχ για να αποθηκεύσουμε το τελικό αποτέλεσμα.

```
task.cancel(true);
```

4.3 Κλάση AsyncTaskLoader

Την λύση στο πρόβλημα του αυτόματου χειρισμού των αλλαγών αλλά και να απλοποιήσει περαιτέρω την παράλληλη εκτέλεση εργασιών ήρθε με την έκδοση 3.0 να δώσει η κλάση `AsyncTaskLoader`. Κύριος σκοπός της κλάσης είναι η φόρτωση δεδομένων στο παρασκήνιο και για αυτό θα δούμε πως δεν έχει όλες τις δυνατότητες της `AsyncTask` αλλά στις

περισσότερες περιπτώσεις καλύπτει επαρκώς τις ανάγκες μας. Έχουν επίσης τη δυνατότητα να «παρακολουθούν» την πηγή των δεδομένων και να παραδίδουν νέα αποτελέσματα όταν αλλάζει το περιεχόμενο. Παρόλο που εισήχθηκε στην έκδοση 3.0 η κλάση είναι διαθέσιμη και για παλιότερες εκδόσεις μέσω της βιβλιοθήκης υποστήριξης υπάρχουν όμως σημαντικές διαφορές τις οποίες και θα δούμε. Όπως φαίνεται και από το όνομα η κλάση εσωτερικά χρησιμοποιεί την `AsyncTask`.

4.3.1 Ορισμός της κλάσης

Η `AsyncTaskLoader(android.content.AsyncTaskLoader<D>)` είναι όπως και η `AsyncTask` μια abstract generic κλάση. Για την επιτυχή χρήση της όμως θα πρέπει να χρησιμοποιήσουμε δυο ακόμα πράγματα:

- `AsyncTaskLoader`: Πρέπει να επεκτείνουμε αυτή τη κλάση υλοποιώντας την μέθοδο `loadInBackground` και ορίζοντας τον generic τύπο επιστροφής.
- `LoaderManager`: Μια κλάση η οποία συσχετίζεται με κάθε Activity ή Fragment, στα οποία αντιστοιχεί ένας μόνο LoaderManager. Αυτή βοηθάει στην διαχείριση των αλλαγών και στη διατήρηση της εργασίας μας.
- `LoaderManager.LoaderCallbacks`: Ένα interface το οποίο πρέπει να υλοποιήσει το Activity ή Fragment από το οποίο εκκινούμε την εργασία για να αλληλεπιδράσουν επιτυχώς με τον `LoaderManager`.

Επειδή το στιγμιότυπο του Loader δεν καταστρέφεται κατά την αναδημιουργία των Activities / Fragments μπορεί να διατηρήσει τα δεδομένα.

4.3.2 Παράδειγμα χρήσης

1. Υλοποιούμε την κλάση μας με τύπο επιστροφής μια λίστα ακεραίων.

```
class CustomLoader extends AsyncTaskLoader<ArrayList<Integer>> {
```

Διατηρούμε το αποτέλεσμα του task σαν πεδίο σε περίπτωση που ζητηθεί ξανά, πχ μετά από αλλαγή στον προσανατολισμό.

```
	private ArrayList<Integer> list;

	public CustomLoader(Context context) {
		super(context);
	}
```

Η μέθοδος που πρέπει να υλοποιήσουμε και περιέχει την εργασία που θέλουμε να πραγματοποιήσουμε. Ελέγχουμε αρχικά αν έχουμε ήδη το αποτέλεσμα και το επιστρέφουμε απευθείας αλλιώς συνεχίζουμε στον υπολογισμό.

```
	public ArrayList<Integer> loadInBackground() {
		if (list != null) return list;

		list = loadList();

		return list;
	}
}
```

2. Ο `LoaderManager` ενός Activity διαχειρίζεται ένα ή περισσότερα στιγμιότυπα `Loader`. Η δημιουργία ενός `Loader` γίνεται μέσω της παρακάτω κλήσης μεθόδου. Οι τρείς παράμετροι είναι:

a. Το ID, ένας ακέραιος, με το οποίο θα συσχετιστεί ο Loader μας
b. Προαιρετικά, παραμέτρους τις οποίες θέλουμε να περάσουμε μέσα σε ένα `Bundle`. Εδώ null.
c. Το αντικείμενο που υλοποιεί το `LoaderManager.LoaderCallbacks` interface δηλαδή το `Activity` μας.

```
getLoaderManager().initLoader(0, null, this);
```

3. Υλοποιούμε το interface `LoaderManager.LoaderCallbacks` στο Activity μας με generic τύπο τον ίδιο που χρησιμοποιήσαμε και για το `AsyncTaskLoader`, μια λίστα ακεραίων. Θα πρέπει να υλοποιήσουμε 3 μεθόδους για τη διαχείριση του Loader μας, όλες εκτελούνται στο UI νήμα.

```
public class CustomActivity extends Activity implements
LoaderManager.LoaderCallbacks<ArrayList<Integer>> {
```

Η μέθοδος για την δημιουργία του `Loader`, για κάθε ID που θα περάσουμε μπορούμε να δημιουργήσουμε διαφορετικό Loader.

```
public Loader<ArrayList<Integer>> onCreateLoader(int id, Bundle args) {
      return new CustomLoader(this);
}
```

Εκτελείτε όταν επανεκκινείται ο Loader, εδώ αφαιρούμε τις αναφορές προς τα δεδομένα του Loader.

```
public void onLoaderReset(Loader<ArrayList<Integer>> loader) {
      list = null;
}
```

Καλείτε όταν ο Loader μας έχει τελειώσει το φόρτωμα των δεδομένων. Εδώ παρουσιάζουμε τα δεδομένα στο Activity μας ορίζοντας τον `Adapter` του `ListView` μας.

```
public void onLoadFinished(Loader<ArrayList<Integer>> loader,
ArrayList<Integer> data) {

      if (loader.getId() == 0) {
      list = data;

      if (list.isEmpty()){
            setEmptyText("no records");
      } else {
            setListAdapter(new ArrayAdapter<Integer>(
                              getActivity(),
                              android.R.layout.simple_list_item_1,
                              list));
      }
}
```

4.4 Υπηρεσίες

Σε αντίθεση με τις δραστηριότητες οι οποίες προβάλουν ένα γραφικό περιβάλλον οι υπηρεσίες(`Services`) τρέχουν «αόρατα» εκτελώντας εργασίες χωρίς αλληλεπίδραση με τον χρήστη. Ένα συστατικό μπορεί να εκκινήσει μια υπηρεσία και αυτή θα συνεχίσει να εκτελείτε ακόμα και αν ο χρήστης ξεκινήσει μια άλλη εφαρμογή, αυτό καθιστά τις υπηρεσίες κατάλληλες για χρονοβόρες ή συνεχιζόμενες εργασίες.

Υπάρχουν δύο κλάσεις που μπορούμε να επεκτείνουμε για να δημιουργήσουμε μια υπηρεσία:

- `Service`: Αυτή είναι η βασική κλάση για όλες τις υπηρεσίες. Όταν επεκτείνουμε αυτή την κλάση, είναι σημαντικό να δημιουργήσουμε ένα νέο νήμα για να κάνει όλες τις εργασίες της υπηρεσίας, επειδή η υπηρεσία χρησιμοποιεί το βασικό νήμα της εφαρμογής εξ ορισμού κάτι το οποίο θα μπορούσε να επιβραδύνει την εκτέλεση οποιασδήποτε δραστηριότητας βρίσκεται σε λειτουργία.
- `IntentService`: Αυτή είναι μια υποκλάση της `Service` η οποία χρησιμοποιεί ένα νήμα παρασκηνίου για να χειριστεί όλα τα αιτήματα, ένα ένα κάθε φορά. Αυτή είναι η καλύτερη επιλογή αν δεν χρειάζεται η υπηρεσία να χειριστεί πολλαπλές αιτήσεις ταυτόχρονα. Το μόνο που χρειάζεται να κάνετε είναι να υλοποιήσουμε την `onHandleIntent()` η οποία λαμβάνει την πρόθεση για κάθε νέα αίτηση.

Οι βασικές μέθοδοι μιας υπηρεσία τις οποίες θα δούμε επιγραμματικά:

- `onStartCommand()`: Το σύστημα καλεί αυτή τη μέθοδο όταν ένα άλλο συστατικό, πχ μια δραστηριότητα, κάνει μια αίτηση να ξεκινήσει η υπηρεσία καλώντας την `startService()`.
- `onBind()`: Το σύστημα καλεί αυτή τη μέθοδο όταν ένα άλλο συστατικό θέλει να συνδεθεί με την υπηρεσία καλώντας την `bindService()` .
- `onCreate()`: Καλείτε όταν δημιουργηθεί για πρώτη φορά η υπηρεσία, για να εκτελέσει διαδικασίες προετοιμασίας, πριν κληθεί η `onStartCommand()` ή `onBind()`. Εάν η υπηρεσία είναι ήδη σε λειτουργία αυτή η μέθοδος δεν καλείτε .
- `onDestroy()`: Το σύστημα καλεί αυτή τη μέθοδο όταν δεν χρησιμοποιείται πλέον η υπηρεσία και καταστρέφεται.

Εάν ένα συστατικό ξεκινά μια υπηρεσία καλώντας την `startService()` η οποία οδηγεί σε μια κλήση στην `onStartCommand()` η υπηρεσία εξακολουθεί να τρέχει μέχρι να σταματήσει μόνη της με την `stopSelf()` ή ένα συστατικό τη σταματήσει καλώντας την `stopService()`.

4.4.1 Δημιουργία μιας υπηρεσίας

Για τις ανάγκες της εφαρμογής χρησιμοποιούμε μόνο την `IntentService` η οποία μας καλύπτει πλήρως καθώς:

- Δημιουργεί ένα νήμα παρασκηνίου και εκτελεί όλες τις προθέσεις που παραδίδεται στην `onStartCommand()` ξεχωριστά από κύριο νήμα της εφαρμογής μας.
- Δημιουργεί μια ουρά εργασιών περνώντας τις προθέσεις στην `onHandleIntent()` της υπηρεσίας μας σειριακά.
- Σταματά την υπηρεσία μόλις ολοκληρωθούν όλα τα αιτήματα.

Για να δημιουργήσουμε λοιπόν μια υπηρεσία χρειάζεται αρχικά να την δηλώσουμε στο manifest όπως με τα υπόλοιπα συστατικά της εφαρμογής μας.

```
<application
      ...
      <service
            android:name="com.nkyrim.itapp.services.BulletinService"
            android:icon="@drawable/ic_launcher"
            android:label="BulletinService" >
      </service>
      ...
```

```
</application>
```

Το μόνο που μένει είναι να επεκτείνουμε την κλάση `IntentService` και να υλοποιήσουμε το δομητή ορίζοντας το όνομα της υπηρεσία και την μέθοδο `onHandleIntent()` η οποία πραγματοποιεί και την ουσιαστική εργασία.

```
public class DownloadService extends IntentService {
	public final static String URL = "com.nkyrim.itapp.services.URL";

	public DownloadService () {
		super("DownloadService");
	}

	@Override
	protected void onHandleIntent(Intent intent) {
		downloadFile();

		postNotification();
	}
}
```

Στην δραστηριότητα(ή το `Fragment`) για να εκκινήσουμε την υπηρεσία

```
Intent intent = new Intent(ITapp.getAppContext(), DownloadService.class);
intent.putExtra(DownloadService.URL, urlString);
context.startService(intent);
```

Κεφάλαιο 5: Ανάκτηση και αποθήκευση δεδομένων

Κάθε εφαρμογή η οποία λαμβάνει και επεξεργάζεται δεδομένα θα πρέπει να χρησιμοποιεί ένα συγκεκριμένο τρόπο ανάκτησης τους καθώς και να μπορεί να αποθηκεύει τα δεδομένα αυτά για χρήση αργότερα. Για την εφαρμογή μας χρησιμοποιούμε ένα HTTP πελάτη και τις διάφορες μεθόδους αποθήκευσης που παρέχει το σύστημα.

5.1 Λήψη δεδομένων μέσω HTTP

Όπως οι περισσότερες Android εφαρμογές που συνδέονται στο δίκτυο χρησιμοποιούν το πρωτόκολλο HTTP για την αποστολή και λήψη δεδομένων έτσι και η εφαρμογή μας βασίζεται σε αυτό για την υλοποίηση των λειτουργιών της.

Το Android περιλαμβάνει δύο πελάτες HTTP: Τις κλάσεις `HttpURLConnection` και `ApacheHTTPClient`. Και οι δύο έχουν υποστήριξη για HTTPS, streaming αποστολές και λήψεις, δυνατότητα ρύθμισης χρονικών ορίων, IPv6 και connection pooling. Για την εφαρμογή μας χρησιμοποιήθηκε η `HttpURLConnection` καθώς η χρήση της προτείνεται για εκδόσεις μετά την 2.3. Είναι ένας ελαφρύς γενικού σκοπού HTTP πελάτης ο οποίος καλύπτει τις περισσότερες ανάγκες. Έχει ένα πολύ βασικό αλλά ευέλικτο API και η υλοποίηση του βελτιώνεται συνεχώς.

5.1.1 Δημιουργία HTTP πελάτη της εφαρμογής

Για την εφαρμογή μας κατασκευάστηκε ένα πακέτο κλάσεων για την κάλυψη όλων των πιθανών HTTP αναγκών, πχ αιτήσεις GET, POST, ώστε να είναι εύκολη η χρήση του καθώς οι περισσότερες λειτουργίες βασίζονται στη λήψη δεδομένων μέσω HTTP.

Οι κλάσεις που υλοποιήθηκαν:

- `HttpClient`: Η κύρια κλάση που πραγματοποιεί τις GET και POST αιτήσεις.
- `RequestOption`: Ο ορισμός με τις ιδιότητες της αίτησης, URL, παραμέτρους, κλπ.
- `RequestResult`: Το αποτέλεσμα της αίτησης το οποίο περιέχει τον κωδικό της απάντησης, το μήνυμα και το ουσιαστικό αποτέλεσμα της αίτησης σε ένα πίνακα byte ώστε να είναι εύκολη η μετατροπή πχ σε String αν πρόκειται για ιστοσελίδα που θα αναλύσουμε ή αποθήκευση σε αρχείο.
- `PersistentCookieStore` και `CoocieDbHelper`: Βοηθητικές κλάσεις για την αποθήκευση των cookies σε περιπτώσεις επανεκκίνησης της εφαρμογής.

Οι υλοποιήσεις των 3 βασικών κλάσεων:

HttpClient

```
public class HttpClient {
	private final static String TAG = "HttpClient";

	// General client options
	private final String USER_AGENT = "HttpClient";
	private final int TIMEOUT = 20 * 1000;

	// Custom error codes
	public final static int ERROR_TIMEOUT = 601;


	public HttpClient() {
		CookieHandler.setDefault(new CookieManager());
	}


	public HttpClient(CookieStore store) {
		CookieHandler.setDefault(new CookieManager(store,
			CookiePolicy.ACCEPT_ORIGINAL_SERVER));
	}

	// ================ GET Request methods ==================================
	public RequestResult get(URL url) {
		return get(url, null);
	}

	public RequestResult get(URL url, RequestOptions options) {
		HttpURLConnection conn;
		InputStream in = null;
		RequestResult result = new RequestResult();

		try {
			conn = (HttpURLConnection) url.openConnection();
			conn.setConnectTimeout(TIMEOUT);
			conn.setReadTimeout(TIMEOUT);
			conn.setRequestProperty("User-Agent", USER_AGENT);

			if (options != null) {
				for (Entry<String, String> e : options.getHeaders()) {
					conn.setRequestProperty(e.getKey(), e.getValue());
				}
			}

			result.setCode(conn.getResponseCode());
			result.setMessage("Http status message: "
					+ conn.getResponseMessage());
			result.setResponseHeaders(conn.getHeaderFields());

			in = new BufferedInputStream(conn.getInputStream());
			result.setResponse(readBytes(in));
			result.setSuccess(true);

			conn.disconnect();
		} catch (SocketTimeoutException ex) {
			result = new RequestResult(ERROR_TIMEOUT, "Connection Timeout");
			Log.e(TAG, Log.getStackTraceString(ex));
		} catch (IOException ex) {
```

```
                result = new RequestResult(1);
                if (ex.getMessage() != null) result.setMessage(ex.getMessage());
                Log.e(TAG, Log.getStackTraceString(ex));
        } finally {
                try {
                        if (in != null) in.close();
                } catch (IOException ex) {
                        Log.e(TAG, Log.getStackTraceString(ex));
                }
        }

        return result;
}

public CookieStore getCookieStore() {
        return ((CookieManager) CookieHandler.getDefault()).getCookieStore();
}

// ================ POST Request methods ================================
public RequestResult post(URL url) {
        return post(url, null);
}

public RequestResult post(URL url, RequestOptions options) {
        HttpURLConnection conn;
        OutputStream out;
        InputStream in = null;
        RequestResult result = new RequestResult();

        try {
                conn = (HttpURLConnection) url.openConnection();
                conn.setConnectTimeout(TIMEOUT);
                conn.setReadTimeout(TIMEOUT);
                conn.setRequestProperty("User-Agent", USER_AGENT);
                conn.setDoOutput(true);
                conn.setUseCaches(false);

                if (options != null) {
                        for (Entry<String, String> e : options.getHeaders()) {
                                conn.setRequestProperty(e.getKey(), e.getValue());
                        }
                }

                out = new BufferedOutputStream(conn.getOutputStream());
                out.write(options.getParams().getBytes());
                out.close();

                result.setCode(conn.getResponseCode());
                result.setMessage("Http status message: "
                                + conn.getResponseMessage());
                result.setResponseHeaders(conn.getHeaderFields());

                in = new BufferedInputStream(conn.getInputStream());
                result.setResponse(readBytes(in));
                result.setSuccess(true);

                conn.disconnect();
        } catch (SocketTimeoutException ex) {
                result = new RequestResult(ERROR_TIMEOUT, "Connection Timeout");
                Log.e(TAG, Log.getStackTraceString(ex));
        } catch (IOException ex) {
                result = new RequestResult(1);
                if (ex.getMessage() != null) result.setMessage(ex.getMessage());
                Log.e(TAG, Log.getStackTraceString(ex));
        } finally {
                try {
                        if (in != null) in.close();
                } catch (IOException ex) {
                        Log.e(TAG, Log.getStackTraceString(ex));
                }
        }

        return result;
}

//====================== Private methods ======================================
private byte[] readBytes(InputStream in) throws IOException {
```

```
                ByteArrayOutputStream baos = new ByteArrayOutputStream(1024 * 1024);
                BufferedInputStream bin = new BufferedInputStream(in);
                byte[] buffer = new byte[8 * 1024];

                int read = 0;
                while ((read = bin.read(buffer)) != -1) {
                        baos.write(buffer, 0, read);
                }

                return baos.toByteArray();
        }
}
```

RequestOption

```
public class RequestOptions {
        private String encoding;
        private HashMap<String, String> headers;
        private HashMap<String, String> params;

        public RequestOptions() {
                this("utf-8");
        }

        public RequestOptions(String encoding) {
                this.encoding = encoding;
                this.headers = new HashMap<String, String>(5);
                this.params = new HashMap<String, String>(5);
        }

        public void addCookie(HttpCookie cookie) {
                String cookies = headers.get("Cookie");
                if (cookies != null) cookies = cookies + ";" + cookie.toString();
                else cookies = cookie.toString();

                headers.put("Cookie", cookies);
        }

        public void addHeader(String name, String value) {
                headers.put(name, value);
        }

        public void addParam(String name, String value) {
                params.put(name, value);
        }

        public Set<Entry<String, String>> getHeaders() {
                return headers.entrySet();
        }

        public String getParams() {
                StringBuilder result = new StringBuilder();

                for (Entry<String, String> e : params.entrySet()) {

                        // if there are already some params added
                        if (result.length() > 0) {
                                result.append("&");
                        }

                        result.append(e.getKey());
                        result.append("=");
                        try {
                                result.append(URLEncoder.encode(e.getValue(), encoding));
                        } catch (UnsupportedEncodingException ex) {
                                throw new IllegalArgumentException(ex);
                        }
                }

                return result.toString();
        }
}
```

RequestResult

```
public class RequestResult {
        private boolean success;
```

```
        private int code;
        private String message;
        private byte[] response;
        private Map<String, List<String>> responceHeaders;

        public RequestResult() {}

        public RequestResult(int code) {
                this.code = code;
        }

        public RequestResult(int code, boolean success) {
                this.code = code;
                this.success = success;
        }

        public RequestResult(int code, String message) {
                this.code = code;
                this.message = message;
        }

        public int getCode() {
                return code;
        }

        public String getMessage() {
                return message;
        }

        public Map<String, List<String>> getResponseHeaders() {
                return responceHeaders;
        }

        public void setResponseHeaders(Map<String, List<String>> responceHeaders) {
                this.responceHeaders = responceHeaders;
        }

        public byte[] getResponse() {
                return response;
        }

        public boolean isSuccess() {
                return success;
        }

        public void setCode(int code) {
                this.code = code;
        }

        public void setMessage(String message) {
                this.message = message;
        }

        public void setResponse(byte[] response) {
                this.response = response;
        }

        public void setSuccess(boolean success) {
                this.success = success;
        }
}
```

5.1.2 Χρήση του πελάτη HTTP

Όλες οι HTTP αιτήσεις γίνονται με τη χρήση του HttpClient. Θα δούμε ένα παράδειγμα χρήσης του για την διαδικάσία σύνδεσης στις υπηρεσίες της Ύδρας.

```
public final static HttpClient client;

public static RequestResult loginHydra(String username, String password) {
        RequestOptions options = new RequestOptions();
        options.addParam("am", username);
        options.addParam("pass", password);
        options.addParam("login", "Login");
```

```
        RequestResult result = null;

        try {
                // 1. Handshake - Get the session cookie
                result = client.get(Constants.URLS.HYDRA_BASE);

                // if HTTP error occurred, return result with message
                if (result.getCode() >= 400) return result;

                // 2. Post request - Login
                result = client.post(Constants.URLS.HYDRA_LOGIN, options);

                // 3. On Gingerbread get the index page manually
                if (Build.VERSION.SDK_INT < Build.VERSION_CODES.HONEYCOMB) {
                        if (result.getCode() == 302) {
                                result = client.get(Constants.URLS.HYDRA_LOGIN);
                        }
                }

                if (!result.isSuccess()) return result;

                        Document doc = Jsoup.parse(new String(
                                                result.getResponse(),
                                                Constants.ENCODING_HYDRA),
                                                Constants.URLS.HYDRA_BASE.toString());
                        Element error = doc.select("div.veh-msg-error").first();
                        Element accName = doc.select("div.txt").last();
                        result.setResponse(null);

                        if (error != null) {
                                // page presents an error
                                result.setMessage(error.text());
                                result.setSuccess(false);
                        } else if (accName == null) {
                                // no apparent error but no verified login(unknown
                                // error), leave the original HTTP response message
                                result.setSuccess(false);
                                result.setMessage(Util.getString(
                                                        R.string.error_occured,
                                                        result.getMessage()));
                        } else {
                                // successful login
                                result.setSuccess(true);
                                result.setMessage(accName.text());
                                setLastHydra();
                        }
                } catch (IOException ex) {
                        Log.d(TAG, ex.toString());
                }

        return result;
}
```

5.2 Εξαγωγή Δεδομένων

Εξαγωγή δεδομένων είναι μια τεχνική κατά την οποία ένα πρόγραμμα υπολογιστή αντλεί δεδομένα από μια αναγνώσιμη από τον άνθρωπο έξοδο προερχόμενη από άλλο πρόγραμμα. Συνήθως, η μεταφορά δεδομένων μεταξύ των προγραμμάτων επιτυγχάνεται με τη χρήση δομών δεδομένων κατάλληλων για την αυτοματοποιημένη επεξεργασία από υπολογιστές, όχι ανθρώπους. Τέτοιες μορφές ανταλλαγών και πρωτόκολλα είναι αυστηρά δομημένα, καλά τεκμηριωμένα, εύκολα να αναλυθούν, και παραμένουν σαφή αλλά οι μεταδόσεις αυτές δεν είναι αναγνώσιμες από τον άνθρωπο. Έτσι, το βασικό στοιχείο που διακρίνει την εξαγωγή δεδομένων

από την τυπική ανάλυσης είναι ότι η έξοδος προοριζόταν για εμφάνιση στον τελικό χρήστη και όχι ως είσοδος σε ένα άλλο πρόγραμμα και ως εκ τούτου συνήθως δεν είναι ούτε τεκμηριωμένες ούτε δομημένες για εύκολη ανάλυση.

Οι ιστοσελίδες έχουν κατασκευαστεί χρησιμοποιώντας mark up γλώσσες που βασίζονται σε κείμενο (HTML και XHTML) και συχνά περιέχουν πληθώρα χρήσιμων δεδομένων σε μορφή κειμένου. Ωστόσο, οι ιστοσελίδες σχεδιάζονται για με στόχο την βέλτιστη εμφάνιση για ανθρώπους και όχι για ευκολία στην αυτοματοποιημένη χρήση τους. Για αυτό το λόγο έχουν δημιουργηθεί εργαλεία που εξάγουν το περιεχόμενο ιστοσελίδων Αυτή η αυξανόμενη διαδικασία της εξόρυξης δεδομένων από το διαδίκτυο(web data extraction) αναφέρεται ως web harvesting ή web scrapping .

5.2.1 Εξόρυξης δεδομένων από το διαδίκτυο

Web scrapping είναι η διαδικασία της αυτόματης συλλογής πληροφοριών από το Παγκόσμιο Ιστό. Προσομοιώνουν ουσιαστικά ένα άτομο που βλέπει μια ιστοσελίδα με ένα πρόγραμμα περιήγησης και εξάγουν μόνο την απαραίτητη πληροφορία σύμφωνα με τους κανόνες που έχουμε ορίσει. Υπάρχουν διάφορες τεχνικές οι οποίες χρησιμοποιούνται ανάλογα με το βαθμό της αυτοματοποίησης που προσφέρουν.

- Κανονικές εκφράσεις και ανάλυση κειμένου: Μια πολύ απλή τεχνική η οποία δέχεται τα δεδομένα σαν απλό κείμενο και με τη χρήση κανονικών εκφράσεων(regular expressions) εξάγει τα δεδομένα που ταιριάζουν με τις επιλογές μας.
- Αλγόριθμοι εξόρυξης δεδομένων: Ιστοσελίδες οι οποίες παρουσιάζουν ένα μεγάλο αριθμό δεδομένων ο οποίος βρίσκεται αποθηκευμένος σε κάποια δομημένη πηγή(πχ σχεσιακή βάση δεδομένων) χρησιμοποιούν κοινά πρότυπα(templates) πχ HTML πίνακες. Στην εξόρυξη δεδομένων προγράμματα που ανιχνεύουν τέτοια πρότυπα και εξάγουν την πληροφορία σε σχεσιακή μορφή λέγονται Wrappers.
- Ανάλυση DOM: Με τη χρήση ενός ολοκληρωμένου προγράμματος περιήγησης(πχ Chrome, Firefox) μπορούμε να ανακτήσουμε το δυναμικό περιεχόμενο το οποίο παράγεται.
- Ανάλυση HTML: Ανάλυση του HTML κώδικα με γλώσσες ερωτημάτων όπως οι Xquery/XPath.

Τις περισσότερες φορές όμως χρησιμοποιούμε ένα συνδυασμό αυτών των τεχνικών για να αντλήσουμε την πληροφορία που θέλουμε.

5.2.2 Βιβλιοθήκη ανάλυσης HTML - Jsoup(HTML Parser)

Για τη εφαρμογή χρησιμοποιήθηκε η βιβλιοθήκη JSoup(http://jsoup.org/) στην έκδοση 1.7.3. Είναι μια βιβλιοθήκη ανοιχτού κώδικα σε Java συμβατή με το Android η οποία μας παρέχει την δυνατότητα να εξάγουμε και να διαχειριστούμε δεδομένα από HTML κάνοντας χρήση του DOM, CSS selectors καθώς και άλλων μεθόδων.

Υλοποιεί την προδιαγραφή HTML5 και αναλύει τον HTML κώδικα στην ίδια DOM ιεραρχία όπως οι σύγχρονοι browsers.

- Μπορεί να αναλύσει HTML από μια διεύθυνση URL, αρχείο, ή Java String.
- Εξάγει τα δεδομένα, διασχίζοντας την DOM ιεραρχία ή χρησιμοποιώντας CSS επιλογείς.

- Χειρίζεται τα στοιχεία HTML, τα χαρακτηριστικά, και το κείμενο που περιέχουν.
- Μπορεί να «καθαρίσει» το περιεχόμενο για να αποτρέψει τις επιθέσεις XSS
- Τέλος μπορεί να εξάγει έγκυρο HTML από μια ιστοσελίδα με μη έγκυρη δομή δημιουργώντας μια λογική DOM ιεραρχία.

5.2.2.1 Είσοδος - Ανάλυση HTML κώδικα

Η βασική κλάση σε αντικείμενο της οποίας μετατρέπουμε τον HTML κώδικα, ουσιαστικά δηλαδή κείμενο συνήθως ένα αντικείμενο String, είναι η κλάση Document. Η βοηθητική μέθοδος της κλάσης Jsoup χρησιμοποιείτε για την ανάλυση(parsing) της εισόδου.

```
String htmlString = getHtml();

Document doc = Jsoup.parse(htmlString);
```

Ο αναλυτής θα προσπαθήσει να δημιουργήσει μια όσο το δυνατόν πιο «καθαρή» και έγκυρη ιεραρχία από το παρεχόμενο HTML, πχ αναγνωρίζοντας tags τα οποία δεν έχουν κλείσει ή έχουν παραλειφθεί εντελώς. Το αντικείμενο Document που δημιουργείτε αποτελείται κυρίως από αντικείμενα των κλάσεων `Element` και `TextNode`. Όλες αυτές οι κλάσεις έχουν σαν βασική κλάση την Node. Η βιβλιοθήκη παρέχει και τη δυνατότητα δημιουργίας ενός Document από ένα File ή ένα URL αντικείμενο.

5.2.2.2 Επιλογή τμημάτων

Παρέχονται δύο τρόποι για να επιλέξουμε τα τμήματα που μας ενδιαφέρουν από το αντικείμενο Document.

1. Με χρήση μεθόδων που θυμίζουν μεθόδους του DOM πχ `getElementById()`
2. Με τη χρήση CSS selectors(CSS επιλογείς)

Μέθοδοι DOM

Οι μέθοδοι της υπέρ-κλάσης Node, `getElementById` και `getElementsByTag` λειτουργούν όπως οι αντίστοιχες μέθοδοι DOM.

```
Element content = doc.getElementById("content");

Elements links = content.getElementsByTag("a");
```

Επιλογείς CSS

Οι CSS επιλογείς είναι πρότυπα(patterns) το οποία χρησιμοποιούνται κατά την ανάπτυξη ιστοσελίδων για την επιλογή των HTML στοιχείων, πχ παράγραφοι, εικόνες, για εφαρμογή style ή τροποποίηση του. Ομοίως μπορούμε να επιλέξουμε τα στοιχεία που επιθυμούμε με τη χρήση της μεθόδου `select`().

Εδώ επιλέγουμε όλα τα στοιχεία τα οποία έχουν την ιδιότητα “onmouseover”.

```
Elements bulletins = doc.select("[onmouseover]");
```

Μπορούμε επίσης να χρησιμοποιήσουμε πολλαπλούς επιλογείς.

```
Elements grades =doc.select("td.groupHeader, tr[height][bgcolor=#fafafa]");
```

Καθώς επίσης και να ορίσουμε ένα «μονοπάτι» προς το στοιχείο που θέλουμε.

```
Elements requests = doc.select("table#tablemain>tbody>tr>td>table>tbody>tr");
```

5.2.2.3 Εξαγωγή Δεδομένων

Αφού επιλέξουμε τα στοιχεία που θέλουμε πρέπει να εξάγουμε την πληροφορία. Τα αντικείμενα της κλάσης `Element` παρέχουν τις απαραίτητες μεθόδους για πρόσβαση στα δεδομένα τους.

Το παρακάτω επιστρέφει το κείμενο που περιέχεται στο συγκεκριμένο `Element`.

```
Element firstGrade = grades.get(0);
String grade = firstGrade.text();
```

Εδώ επιλέγουμε την τιμή που έχει η ιδιότητα “onmouseover”.

```
Element desc = bulletins.get(5);
String text = desc.attr("onmouseover");
```

Τέλος, πολλές φορές θα χρειαστεί να χρησιμοποιήσουμε μεθόδους χειρισμού συμβολοσειρών(όπως οι `replace`() και `substring`() της `String`) και ταίριασμα κανονικών εκφράσεων για να απομονώσουμε το τμήμα της πληροφορίας που επιθυμούμε.

5.3 Αποθήκευση δεδομένων

Μετά την ανάκτηση των δεδομένων μας συνήθως υπάρχει η ανάγκη για αποθήκευση τους ώστε να είναι διαθέσιμα για μετέπειτα επεξεργασία. Όμως και τα ίδια δεδομένα της εφαρμογής, όπως οι ρυθμίσεις και οι προτιμήσεις του χρήστη θα πρέπει να αποθηκεύονται.

Το Android μας παρέχει 3 βασικές μεθόδους αποθήκευσης δεδομένων:

- Αποθήκευση ζευγών κλειδιού-τιμής απλών τύπων δεδομένων σε ένα κοινόχρηστο αρχείο (Shared Preferences).
- Αποθήκευση διαφόρων αρχείων στο σύστημα αρχείων του Android.
- Χρήση βάσεων δεδομένων που διαχειρίζεται η SQLite.

5.3.1 Χρησιμοποιώντας τα Shared Preferences

Η κλάση `SharedPreferences` παρέχει ένα γενικό πλαίσιο που μας επιτρέπει να αποθηκεύσουμε και να ανακτήσουμε ζευγάρια κλειδιού-τιμής των βασικών τύπων δεδομένων όπως `boolean`, `int`, `Strings`, κλπ. Αυτά τα δεδομένα θα παραμένουν μεταξύ των συνεδριών των χρηστών (ακόμα και αν η εφαρμογή επανεκκινήσει). Αν και έχει το όνομα «προτιμήσεις» μπορούμε να αποθηκεύσουμε οποιαδήποτε δεδομένα θέλουμε, όχι μόνο τις προτιμήσεις του χρήστη.

Για να πάρουμε ένα αντικείμενο `SharedPreferences` για την εφαρμογή μας χρησιμοποιούμε μία από τις δύο μεθόδους:

- `getSharedPreferences`(): Χρησιμοποιείτε όταν χρειαζόμαστε πολλαπλά αρχεία προτιμήσεων που προσδιορίζονται με βάση το όνομα, το οποίο ορίζουμε με την πρώτη παράμετρο και τον τύπο πρόσβασης με τη δεύτερη, αν και τύποι πρόσβασης περάν του ιδιωτικού έχουν καταργηθεί .
- `getPreferences`(): Χρησιμοποιείτε όταν χρειαζόμαστε μόνο ένα αρχείο προτιμήσεων για την τρέχουσα δραστηριότητα. Επειδή αυτό θα είναι το μόνο αρχείο προτιμήσεων για την δραστηριότητα δεν δίνουμε όνομα.

Για την εφαρμογή μας τα SharedPreferences χρησιμοποιήθηκαν για την αποθήκευση των ρυθμίσεων της εφαρμογής.

```
public final class Settings {
```

Χρησιμοποιούμε το Application Context για να αποκτήσουμε πρόσβαση στο αρχείο μας με όνομα "app_settings" καθώς και ιδιωτικό τύπο πρόσβασης.

```
  private static SharedPreferences getPrefs() {
    SharedPreferences prefs = ITapp.getAppContext().getSharedPreferences(
                                    app_settings",
                                    Context.MODE_PRIVATE);
    return prefs;
  }
```

Μέθοδοι ανάκτησης των δεδομένων, Key είναι ένα Enum που χρισιμοποιούμε για τα ονόματα.

```
  public static boolean getBoolean(Key key, boolean defValue) {
      return getPrefs().getBoolean(key.name(), defValue);
  }

  public static int getInt(Key key, int defValue) {
      return getPrefs().getInt(key.name(), defValue);
  }
...
...
```

Για να αποθηκεύσουμε τα δεδομένα μας πρέπει πρώτα να χρησιμοποιήσουμε ένα αντικείμενο Editor καλώντας την edit() της SharedPreferences.

```
  public static void setBoolean(Key key, boolean value) {
      SharedPreferences.Editor editor = getPrefs().edit();
      editor.putBoolean(key.name(), value).commit();
  }

  public static void setInt(Key key, int value) {
      getPrefs().edit().putInt(key.name(), value).commit();
  }
...
...
}
```

5.3.2 Χρησιμοποιώντας αρχεία

Η χρήση των αρχείων στο Android δεν διαφέρει από τον τρόπο που χειριζόμαστε τα αρχεία στη Java με την χρήση Streams. Οι διαφοροποιήσεις έχουν να κάνουν κυρίως με το «που» θα αποθηκεύσουμε τα αρχεία μας, στην εσωτερική ή εξωτερική μνήμη, καθώς και το αν άλλες εφαρμογές αλλά και ο χρήστης θα έχουν πρόσβαση σε αυτά.

5.3.2.1Χρήση εσωτερικής μνήμης

Ο εσωτερικός χώρος αποθήκευσης είναι η καλύτερη επιλογή όταν θέλουμε να είμαστε σίγουροι ότι ούτε ο χρήστης ούτε άλλες εφαρμογές μπορούν να έχουν πρόσβαση στα αρχεία μας.

- Είναι πάντα στη διάθεσή μας.
- Τα αρχεία που αποθηκεύονται εδώ είναι προσβάσιμα εξ ορισμού μόνο από την εφαρμογή μας.
- Όταν ο χρήστης απεγκαταστήσει την εφαρμογή μας το σύστημα αφαιρεί όλα τα αρχεία της από την εσωτερική μνήμη.

Χρησιμοποιούμε δύο μεθόδους για να πάρουμε ένα αντικείμενο `File` με τον κατάλληλο κατάλογο.

- `getFilesDir`(): Επιστρέφει ένα αντικείμενο `File` που εκπροσωπεί τον προσωπικό εσωτερικό κατάλογο της εφαρμογή μας.
- `getCacheDir`(): Επιστρέφει αρχείου ένα αντικείμενο `File` που εκπροσωπεί τον εσωτερικό κατάλογο για τα προσωρινά αρχεία.

5.3.2.2 Χρήση εξωτερικής μνήμης

Ο εξωτερικός χώρος αποθήκευσης είναι το καλύτερο μέρος για τα αρχεία που δεν απαιτούν περιορισμούς πρόσβασης και για τα που θέλουμε να μοιραστούμε με άλλες εφαρμογές ή να επιτρέψουμε στο χρήστη να έχει πρόσβαση.

- Δεν είναι πάντα στη διάθεσή μας επειδή ο χρήστης μπορεί να την αφαιρέσει από την συσκευή.
- Τα αρχεία που αποθηκεύονται εδώ μπορεί να διαβαστούν χωρίς να μπορούμε να το ελέγξουμε.
- Όταν ο χρήστης απεγκαταστήσει την εφαρμογή μας το σύστημα αφαιρεί τα αρχεία της εφαρμογής από εδώ μόνο εάν τα αποθηκεύσουμε στον προσωπικό εξωτερικό κατάλογο της εφαρμογής.

Οι μέθοδοι που χρησιμοποιούμε:

- `getExternalStorageState`(): Για τον έλεγχο της κατάστασης της μνήμης.
- `getExternalFilesDir`(): Επιστρέφει ένα αντικείμενο `File` που εκπροσωπεί τον εξωτερικό κατάλογο της εφαρμογή μας.
- `getExternalStoragePublicDirectory`(): Περνώντας της την κατάλληλη παράμετρο επιστρέφει τον ανάλογο κατάλογο, πχ `DIRECTORY_MUSIC` ή `DIRECTORY_PICTURES`.

Παράδειγμα ανάκτησης του καταλόγου λήψεων της συσκευής.

```
File dirFile = Environment.getExternalStoragePublicDirectory(
                                Environment.DIRECTORY_DOWNLOADS);
```

5.3.3 Χρήση της SQL βάσης δεδομένων

Η αποθήκευση δεδομένων σε μια βάση δεδομένων είναι ιδανική για επαναλαμβανόμενους ή δομημένους τύπους δεδομένων όπως είναι πχ τα στοιχεία επικοινωνίας. Το Android δεν υποστηρίζει απ' ευθείας σύνδεση σε εξωτερικές βάσεις δεδομένων παρέχει όμως μια εσωτερική βάση βασισμένη στο SQLite στην οποία έχουμε άμεση πρόσβαση.

Το μόνο που χρειάζεται είναι να καθορίσουμε τις δηλώσεις SQL για τη δημιουργία και την ενημέρωση της βάσης δεδομένων. Στη συνέχεια η βάση διαχειρίζεται αυτόματα για μας από την πλατφόρμα Android.

Η SQLite υποστηρίζει τους τύπους δεδομένων TEXT(παρόμοια με `String` στη Java), INTEGER (παρόμοιες με το `long` στη Java) και REAL (παρόμοια με `double` στη Java). Όλοι οι άλλοι τύποι θα πρέπει να μετατραπούν σε ένα από τα πεδία πριν αποθηκευτούν στη βάση δεδομένων. Η ίδια

η βάση δεν επικυρώνει αν οι τύποι στις στήλες είναι από τον καθορισμένο τύπο, πχ μπορεί να γράψουμε έναν ακέραιο σε μια στήλη String και το αντίστροφο.

5.3.3.1 Δημιουργία της βάσης δεδομένων με SQLiteOpenHelper

Για να δημιουργήσουμε μια βάση δεδομένων επεκτείνουμε την κλάση `SQLiteOpenHelper`. Στον δομητή της κλάσης μας καλούμε τον δομητή της `SQLiteOpenHelper` καθορίζοντας το όνομα και την τρέχουσα έκδοση της βάσης.

Θα πρέπει να υπερβούμε τις ακόλουθες μεθόδους για να δημιουργήσουμε και να μπορούμε να ενημερώσουμε τη βάση μας.

- `onCreate`(): Καλείται από το σύστημα εάν η βάση δεδομένων είναι προσβάσιμη αλλά δεν έχει ακόμη δημιουργηθεί.
- `onUpgrade`( ): Καλείτε εάν η έκδοση της βάσης δεδομένων αυξάνεται στον κώδικα της εφαρμογής μας. Αυτή η μέθοδος επιτρέπει να ενημερώσουμε το υπάρχον σχήμα της βάσης δεδομένων ή να διαγράψουμε την υπάρχουσα βάση δεδομένων και να την ξαναδημιουργήσουμε μέσω της μεθόδου `onCreate`().

Και οι δύο μέθοδοι λαμβάνουν στο αντικείμενο `SQLiteDatabase` ως παράμετρο το οποίο είναι η Java αναπαράσταση της βάσης. Η κλάση `SQLiteOpenHelper` παρέχει τη `getReadableDatabase()` και `getWriteableDatabase()` μεθόδους για να αποκτήσουμε πρόσβαση στο `SQLiteDatabase` αντικείμενο για ανάγνωση ή εγγραφή.

Οι πίνακες της βάσης θα πρέπει να χρησιμοποιούν το `_id` αναγνωριστικό για το πρωτεύον κλειδί του πίνακα, πολλές λειτουργίες του Android βασίζονται σε αυτό το πρότυπο.

5.3.3.2 Διαχείριση της βάσης δεδομένων

Η `SQLiteDatabase` είναι η βασική κλάση για να διαχειριστούμε μια βάση δεδομένων SQLite και παρέχει τις μεθόδους για να ανοίξουμε, να κάνουμε ερωτήματα, να τροποποιήσουμε και κλείσουμε τη βάση δεδομένων.

Για την εισαγωγή, ενημέρωση και διαγραφή χρησιμοποιούμε τις μεθόδους `insert`(), `update`() σε συνδυασμό με ένα αντικείμενο `ContentValues` στο οποίο προσθέτουμε ζεύγη στήλης/τιμής και `delete`() όπου ορίζουμε τις παραμέτρους της “Where” πρότασης. Επιπροσθέτως μπορούμε να χρησιμοποιήσουμε την μέθοδο `execSQL()` η οποία εκτελεί μια SQL εντολή απ’ ευθείας.

Για τα ερωτήματα υπάρχουν δύο μέθοδοι. Η `rawQuery`() που δέχεται άμεσα ένα SQL Select εντολή ως είσοδο και η `query`() που παρέχει μια δομημένη διασύνδεση για τον καθορισμό των παραμέτρων του SQL ερωτήματος. Και οι δύο μέθοδοι επιστρέφουν ένα αντικείμενο `Cursor`. Αυτό αντιπροσωπεύει το αποτέλεσμα ενός ερωτήματος και καταδεικνύει μια γραμμή στο αποτέλεσμα του ερωτήματος. Με αυτό τον τρόπο το Android μπορεί να διαχειριστεί τα αποτελέσματα του ερωτήματος αποτελεσματικά καθώς δεν χρειάζεται να φορτώσει όλα τα δεδομένα στη μνήμη.

Ο κώδικας για την αποθήκευση και ανάκτηση των τελευταίων ανακοινώσεων.

```
public class BulletinDbHelper extends SQLiteOpenHelper {
      private static final String DB_NAME = "bulletins.db";
```

```
private static final int DB_VERSION = 1;

private static final String TABLE_BULLETIN = "bulletin";
private static final String COL_ID = "_id";
private static final String COL_TITLE = "title";
private static final String COL_TEXT = "text";
private static final String COL_AUTHOR = "author";
private static final String COL_BOARD = "board";
private static final String COL_DATE = "date";
private static final String COL_ATTACH = "attach";

public BulletinDbHelper(Context context) {
	super(context, DB_NAME, null, DB_VERSION);
}

@Override
public void onCreate(SQLiteDatabase db) {
	String create = "create table " + TABLE_BULLETIN
			+ " (" + COL_ID
			+ " integer primary key autoincrement, "
			+ COL_TITLE + " text, "    + COL_TEXT + " text, "
			+ COL_AUTHOR + " text, " + COL_BOARD + " text, "
			+ COL_DATE + " text, " + COL_ATTACH + " text);";

	db.execSQL(create);
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int
                                                    newVersion) {
	// Drop older table if existed
	db.execSQL("drop table if exists " + TABLE_BULLETIN);

	// Create tables again
	onCreate(db);
}

public ArrayList<Bulletin> getBulletins() {
	ArrayList<Bulletin> list = new ArrayList<Bulletin>(10);
	Bulletin bulletin = null;

	// Open db
	SQLiteDatabase db = this.getWritableDatabase();

	// Select query, select all no parameters
	Cursor cursor = db.query(TABLE_BULLETIN,
				null, null, null, null, null, null);

	// Get the records
	cursor.moveToFirst();
	while (!cursor.isAfterLast()) {
		bulletin = new Bulletin(
			cursor.getString(1), cursor.getString(2),
			cursor.getString(3), cursor.getString(4),
			cursor.getString(5), cursor.getString(6));

		list.add(bulletin);
		cursor.moveToNext();
	}

	// close db
	db.close();

	return list;
}

public Bulletin getLatest() {
	Bulletin bulletin = null;
```

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

		// Select query
		Cursor cursor = db.rawQuery("SELECT * FROM " + TABLE_BULLETIN
				+ " WHERE " + COL_ID
				+ "= (SELECT MIN(" + COL_ID + ") FROM "
				+ TABLE_BULLETIN + ");", null);

		// Get the records
		cursor.moveToFirst();
		if (!cursor.isAfterLast()) {
			bulletin = new Bulletin(
				cursor.getString(1),		cursor.getString(2),
				cursor.getString(3), cursor.getString(4),
				cursor.getString(5), cursor.getString(6));
		}

		db.close();

		return bulletin;
	}

	public void insert(List<Bulletin> list) {
		SQLiteDatabase db = this.getWritableDatabase();

		db.delete(TABLE_BULLETIN, null, null);

		ContentValues[] values = new ContentValues[20];
		for (int i = 0; i < 20; i++) {
			values[i] = new ContentValues();
			values[i].put(COL_TITLE, list.get(i).getTitle());
			values[i].put(COL_TEXT, list.get(i).getText());
			values[i].put(COL_AUTHOR, list.get(i).getAuthor());
			values[i].put(COL_BOARD, list.get(i).getBoard());
			values[i].put(COL_DATE, list.get(i).getDate());
			values[i].put(COL_ATTACH, list.get(i).getAttach());
		}
		for (int i = 0; i < 20; i++) {
			db.insert(TABLE_BULLETIN, null, values[i]);
		}

		db.close();
	}

}
```

5.3.4 Content Providers

Όπως είχαμε δει στην αρχή ένας πάροχος περιεχομένου (`ContentProvider`) διαχειρίζεται ένα κοινό σύνολο δεδομένων μιας εφαρμογής το οποίο άλλες εφαρμογές μπορούν να ανακτήσουν ή ακόμα και να τροποποιήσουν εάν ο πάροχος το επιτρέπει. Χρησιμοποιείτε κυρίως δηλαδή όταν θέλουμε να παρέχουμε έναν ελεγχόμενο, δομημένο και αυστηρά ορισμένο τρόπο πρόσβασης στα δεδομένα τις εφαρμογής μας.

Στην εφαρμογή μας δεν δημιουργούμε κάποιο νέο πάροχο χρησιμοποιούμε όμως τον πάροχο Επαφών(Contacts Provider) του συστήματος για να προσθέσουμε επαφές προσωπικού στις επαφές μας.

```
private void addContact() {

String firstName = member.getFirstName();
String lastName = member.getLastName();
String[] workNumbers = member.getTel().split(",");
String email = member.getEmail();
```

```
String webpage = member.getWebpage();
```

Δημιουργούμε πρώτα μια λίστα στην οποία θα προσθέσουμε τις «εργασίες» που θα πραγματοποιήσουμε, τις προσθήκες στην περίπτωση μας.

```
ArrayList<ContentProviderOperation> ops = new ArrayList<ContentProviderOperation>();

ops.add(ContentProviderOperation.newInsert(ContactsContract.RawContacts.CONTENT_URI)
        .withValue(ContactsContract.RawContacts.ACCOUNT_TYPE, null)
        .withValue(ContactsContract.RawContacts.ACCOUNT_NAME, null)
        .build());
```

Προσθέτουμε ένα - ένα τα πεδία.

```
if (firstName != null) {
  ops.add(ContentProviderOperation.newInsert(ContactsContract.Data.CONTENT_URI)
        .withValueBackReference(ContactsContract.Data.RAW_CONTACT_ID, 0)
        .withValue(ContactsContract.Data.MIMETYPE,
                ContactsContract.CommonDataKinds.StructuredName.CONTENT_ITEM_TYPE)
        .withValue(ContactsContract.CommonDataKinds.StructuredName.DISPLAY_NAME,
                firstName + " " + lastName).build());
}

if (workNumbers.length > 0) {
  for (String wn : workNumbers) {
        ops.add(ContentProviderOperation.newInsert(ContactsContract.Data.CONTENT_URI)
          .withValueBackReference(ContactsContract.Data.RAW_CONTACT_ID, 0)
          .withValue(ContactsContract.Data.MIMETYPE,
                     ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE)
          .withValue(ContactsContract.CommonDataKinds.Phone.NUMBER, wn)
          .withValue(ContactsContract.CommonDataKinds.Phone.TYPE,
                     ContactsContract.CommonDataKinds.Phone.TYPE_WORK).build());
  }
}

...
...
```

Εκτελούμε συγκεντρωτικά τις εργασίες που ορίσαμε παραπάνω χρησιμοποιώντας τον `ContentResolver` που παρέχεται από τη δραστηριότητας μας.

```
        try {
          getActivity().getContentResolver().applyBatch(ContactsContract.AUTHORITY, ops);
          Toast.makeText(getActivity(), R.string.add_success, Toast.LENGTH_SHORT).show();
        } catch (Exception e) {
          e.printStackTrace();
          Toast.makeText(getActivity(), e.getMessage(), Toast.LENGTH_SHORT).show();
        }

}
```

Κεφάλαιο 6: Υλοποίηση της Εφαρμογής

Σε αυτό το κεφάλαιο θα παρουσιάσουμε τις λειτουργίες της εφαρμογής και θα περιγράψουμε τον τρόπο με τον οποίο αυτές υλοποιούνται χρησιμοποιώντας τις μεθόδους και τεχνικές που είδαμε παραπάνω συμπληρώνοντας όπου είναι απαραίτητο κάποιες πιο συγκεκριμένες περιπτώσεις.

6.1 Λειτουργίες Εφαρμογής

Η εφαρμογής προσφέρει πρόσβαση στις διάφορες υπηρεσίες των εξυπηρετητών του τμήματος Ύδρα(hydra) και Πυθία(pithia) καθώς και σε πρόσθετες υπηρεσίες και πληροφορίες. Οι λειτουργίες που υλοποιήθηκαν είναι οι εξής:

- Υπηρεσίες Ύδρας
    - Εμφάνιση των ανακοινώσεων του τμήματος και καθηγητών, με δυνατότητα:

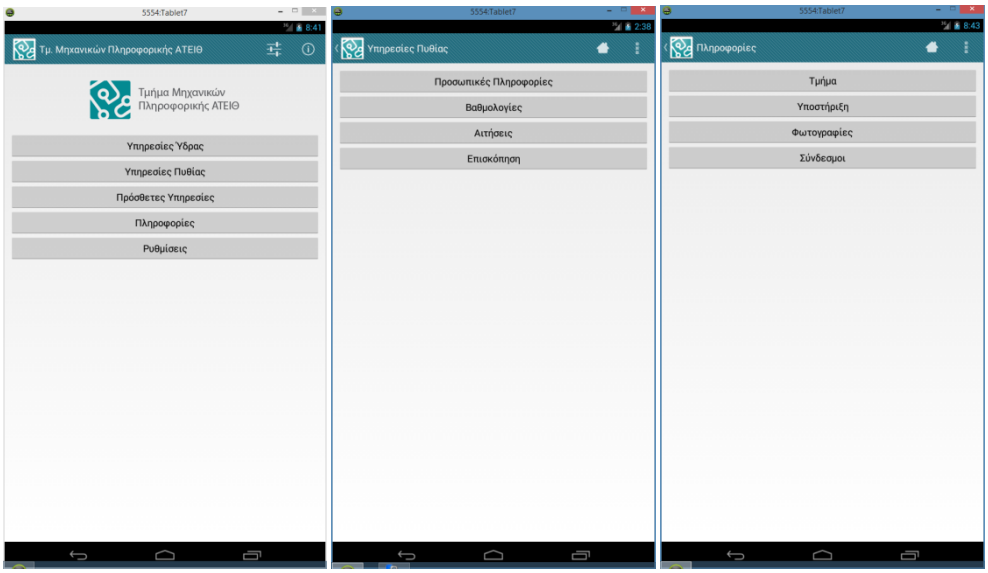
        - Εμφάνισης συγκεκριμένων κατηγοριών.
        - Λήψης συνημμένων αρχείων.
        - Κοινοποίησης των ανακοινώσεων.
    - Εμφάνιση των πτυχιακών, με δυνατότητα:
        - Φιλτραρίσματος με βάση τη διαθεσιμότητα και την κατηγορία(tags).
        - Λήψη του εγγράφου της δήλωσης.
    - Εμφάνιση του προσωπικού της σχολής, με δυνατότητα:
        - Αναζήτησης κατά όνομα.
        - Προσθήκης στις επαφές του χρήστη.
    - Δυνατότητα για έλεγχο νέων ανακοινώσεων και ειδοποίηση του χρήστη.
    - Εφαρμογή αρχικής οθόνης(App Widget) για εμφάνιση των πιο πρόσφατων ανακοινώσεων.
- Υπηρεσίες Πυθίας
    - Εμφάνιση των προσωπικών πληροφοριών του φοιτητή, γενικές πληροφορίες, εξάμηνο, κλπ.
    - Εμφάνιση της βαθμολογίας του φοιτητή οργανωμένη κατά εξάμηνα.
    - Εμφάνιση και λήψη των ολοκληρωμένων αιτήσεων, πχ βεβαιώσεις εγγραφών/δηλώσεων.
- Πρόσθετες Υπηρεσίες
    - Εμφάνιση των χρόνων άφιξης της γραμμής 52 μέσω της mobile ιστοσελίδας του ΟΑΣΘ για την γραμμή 52 ΑΤΕΙΘ – Ν.Σ. Σταθμός.
    - Χάρτης σχολής, με εμφάνιση των σημείων ενδιαφέροντος όπως κτίρια σχολών, γραμματείες, κλπ. Δυνατότητα πλοήγησης με εμφάνιση διαδρομής προς το ΑΤΕΙΘ.
    - Εμφάνιση των εισερχόμενων μηνυμάτων(INBOX) στην υπηρεσία e-mail της σχολής.
- Πληροφορίες
    - Εμφάνιση πληροφοριών τμήματος, σπουδών, υποστήριξης.
    - Εμφάνιση slideshow φωτογραφιών του τμήματος.
    - Εμφάνιση διαφόρων χρήσιμων συνδέσμων.

Παρακάτω παρουσιάζουμε τις δραστηριότητες που υλοποιούν αυτές τις λειτουργίες και χρησιμοποιούν τις κλάσεις και τεχνικές τις οποίες παρουσιάσαμε στα προηγούμενα κεφάλαια.

6.2 Δραστηριότητες Πλοήγησης

Οι παρακάτω δραστηριότητες δεν εκτελούν κάποια διαδικασία πέρα από το να ομαδοποιήσουν τις διάφορες λειτουργίες της εφαρμογής. Όλες οι δραστηριότητες παρέχουν ένα action item για επιστροφή στην αρχική δραστηριότητα και για τις πληροφορίες της εφαρμογής.

- `MainActivity`
- `HydraActivity`
- `PithiaActivity`
- `InfoActivity`
- `AdditionalServicesActivity`



Εικόνα 29: Εμφάνιση των δραστηριοτήτων πλοήγησης

6.3 Ρυθμίσεις Εφαρμογής

Για την δημιουργία της δραστηριότητας για τον έλεγχο των ρυθμίσεων της εφαρμογής χρησιμοποιήθηκε το μέρος του Preference API με το οποίο κατασκευάζουμε μια διεπαφή που είναι κοινή μεταξύ διαφόρων εφαρμογών του Android. Αντί για τη χρήση στοιχείων `View` η κατασκευή του UI γίνεται με τη χρήση υποκλάσεων της κλάσης `Preference` τις οποίες δηλώνουμε σε ένα XML αρχείο. Κάθε `Preference` αντικείμενο αποτελεί το δομικό στοιχείο για μια ρύθμιση. Το Android μας παρέχει αρκετές υποκλάσεις της `Preference` τις οποίες μπορούμε να χρησιμοποιήσουμε όπως οι `ListPreference`, `CheckBoxPreference`, κα. Αυτές αναλαμβάνουν την αποθήκευση των ρυθμίσεων στο αρχείο ρυθμίσεων. Στην εφαρμογή μας όμως θα χρησιμοποιήσουμε στις περισσότερες περιπτώσεις κανονικές δραστηριότητες καθώς αυτές θα πρέπει να επαναχρησιμοποιηθούν και πέρα από την δραστηριότητα των ρυθμίσεων, πχ η δραστηριότητα για τη σύνδεση στα διάφορα συστήματα που εμφανίζεται και στην αρχική οθόνη μας.

6.3.1 Δημιουργία ρυθμίσεων

Το πρώτο βήμα είναι να ορίσουμε το XML αρχείο μας στον κατάλογο των πόρων της εφαρμογής μας, `res/xml/preferences.xml`. Για κάθε στοιχείο μπορούμε να ορίσουμε τον τίτλο του, τις αρχικές τιμές, πιθανή εξάρτηση από άλλη ρύθμιση καθώς και το όνομα του κλειδιού με το οποίο θα αποθηκεύεται. Μπορούμε επίσης να τα βάλουμε σε κατηγορίες για καλύτερη οργάνωση(`PreferenceCategory`).

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android" >
    <PreferenceCategory android:title="@string/pref_accounts" >
        <Preference
            android:key="hydra"
            android:summary="@string/pref_hydra_msg"
            android:title="@string/pref_hydra" />
        <Preference
            android:key="pithia"
            android:summary="@string/pref_pithia_msg"
            android:title="@string/pref_pithia" />
        <Preference
            android:key="mail"
            android:summary="@string/pref_mail_msg"
```

```
            android:title="@string/pref_email" />
    </PreferenceCategory>
    <PreferenceCategory android:title="@string/pref_notifications" >
        <CheckBoxPreference
            android:key="NOTIFICATIONS_ACTIVATION"
            android:summary="@string/pref_notifications_activate_msg"
            android:title="@string/pref_notifications_activate" />

        <ListPreference
            android:defaultValue="3600000"
            android:dependency="NOTIFICATIONS_ACTIVATION"
            android:dialogTitle="@string/pref_interval"
            android:entries="@array/intervals"
            android:entryValues="@array/intervalValues"
            android:key="NOTIFICATIONS_INTERVAL"
            android:summary="@string/pref_interval_msg"
            android:title="@string/pref_interval" />
    </PreferenceCategory>
    <PreferenceCategory android:title="@string/pref_other" >
        <Preference
            android:key="downloads"
            android:summary="@string/pref_download_dir_msg"
            android:title="@string/pref_download_dir" />

</PreferenceScreen>
```

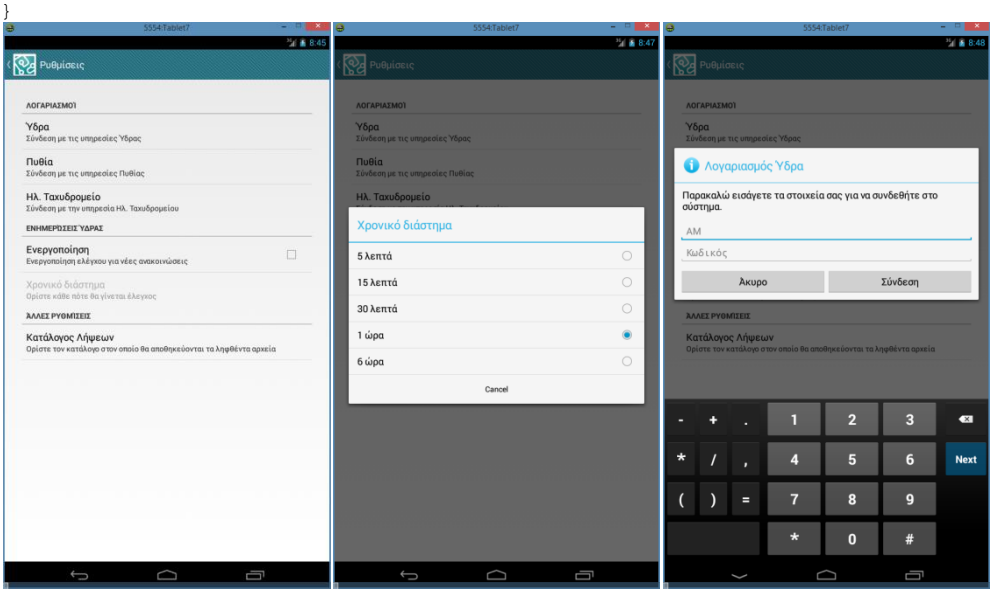
Στο κώδικα μας τώρα αφού φορτώσουμε το αρχείο που δημιουργήσαμε καλώντας την `addPreferencesFromResource()` παίρνουμε τις αναφορές προς τα αντικείμενα των ρυθμίσεων χρησιμοποιώντας την `findPreference()` ώστε να υπερβούμε την μέθοδο `setOnPreferenceClickListener()` και να ορίσουμε τις δικές μας δραστηριότητες οι οποίες θα αναλάβουν τον ορισμό και διαχείριση των ρυθμίσεων.

```
public class PreferencesActivity extends PreferenceActivity {
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    addPreferencesFromResource(R.xml.preferences);

    Preference hydra = findPreference("hydra");
    Preference pithia = findPreference("pithia");
    Preference mail = findPreference("mail");
    ...
    ...

    hydra.setOnPreferenceClickListener(new OnPreferenceClickListener() {
       @Override
       public boolean onPreferenceClick(Preference p) {
         Intent intent = new Intent(
              PreferencesActivity.this,
              LoginActivity.class);
         startActivity(intent);
         return true;
       }
       ...
       ...

});
```

Για τις ανάγκες της εφαρμογής μας δημιουργήσαμε επίσης την κλάση `Settings` ώστε να είναι το κεντρικό σημείο από όπου θα ελέγχουμε τις ρυθμίσεις μας καθώς όπως είπαμε χρησιμοποιούμε κοινές δραστηριότητες για κάποιες από αυτές. Προσοχή πρέπει να δοθεί στην χρήση της μεθόδου `PreferenceManager.getDefaultSharedPreferences` που επιστρέφει το προεπιλεγμένο αρχείο ρυθμίσεων το οποίο χρησιμοποιεί και η

`PreferenceActivity`. Η κλάση `Key` είναι ένα `Enum` όπου ορίζουμε τα κλειδιά για την αποθήκευση.

```
public final class Settings {

        private static SharedPreferences getPrefs() {
                return PreferenceManager.getDefaultSharedPreferences(ITapp.getAppContext());
        }

        public static boolean getBoolean(Key key, boolean defValue) {
                return getPrefs().getBoolean(key.name(), defValue);
        }

        public static int getInt(Key key, int defValue) {
                return getPrefs().getInt(key.name(), defValue);
        }
                ...
                ...
        public static void setBoolean(Key key, boolean value) {
                SharedPreferences.Editor editor = getPrefs().edit();
                editor.putBoolean(key.name(), value).commit();
        }

        public static void setInt(Key key, int value) {
                getPrefs().edit().putInt(key.name(), value).commit();
        }
        ...
        ...
        public static void setupFirstRun() {
                if (getBoolean(Key.IS_FIRST_RUN, true)) {
                        File downdir = Environment.getExternalStoragePublicDirectory(
                                                        Environment.DIRECTORY_DOWNLOADS);
                        setString(Key.DOWNLOAD_DIR, downdir.getAbsolutePath());
                        setBoolean(Key.IS_FIRST_RUN, false);
                }
        }
}
```



Εικόνα 30: Διάφορες επιλογές της δραστηριότητας ρυθμίσεων

Τέλος να πούμε ότι από την έκδοση 3.0 το Preference API έχει ανανεωθεί με την εισαγωγή των `PreferenceFragmnets`, αυτά όμως δεν είναι διαθέσιμα σε παλιότερες εκδόσεις ούτε μέσω της βιβλιοθήκης υποστήριξης και για αυτό δεν τα χρησιμοποιούμε.

6.4 Λειτουργίες υπηρεσιών Ύδρας

Για την υλοποίηση των υπηρεσιών της Ύδρας αναπτύχθηκαν τρείς δραστηριότητες, μία υπηρεσία και ένα app widget:

- `BulletinsActivity`: Εμφανίζει της ανακοινώσεις καθηγητών και τμήματος.
- `StaffActivity`: Εμφανίζει το προσωπικό της σχολής.
- `ThesisActivity`: Εμφανίζει τις πτυχιακές εργασίες.
- `BulletinService`: Ελέγχει για νέες ενημερώσεις και εμφανίζει ειδοποίηση με τον αριθμό των νέων ενημερώσεων. Εμφανίζονται επίσης και οι τίτλοι τους σε μια «μεγάλη προβολή» στις εκδόσεις 4.1+.
- Ένα πολύ βασικό app widget το οποίο ενημερώνεται με τους τίτλους των 6 τελευταίων ενημερώσεων κάθε μισή ώρα ή όταν επιλέξει ο χρήστης.

6.4.1 Δραστηριότητα ανακοινώσεων

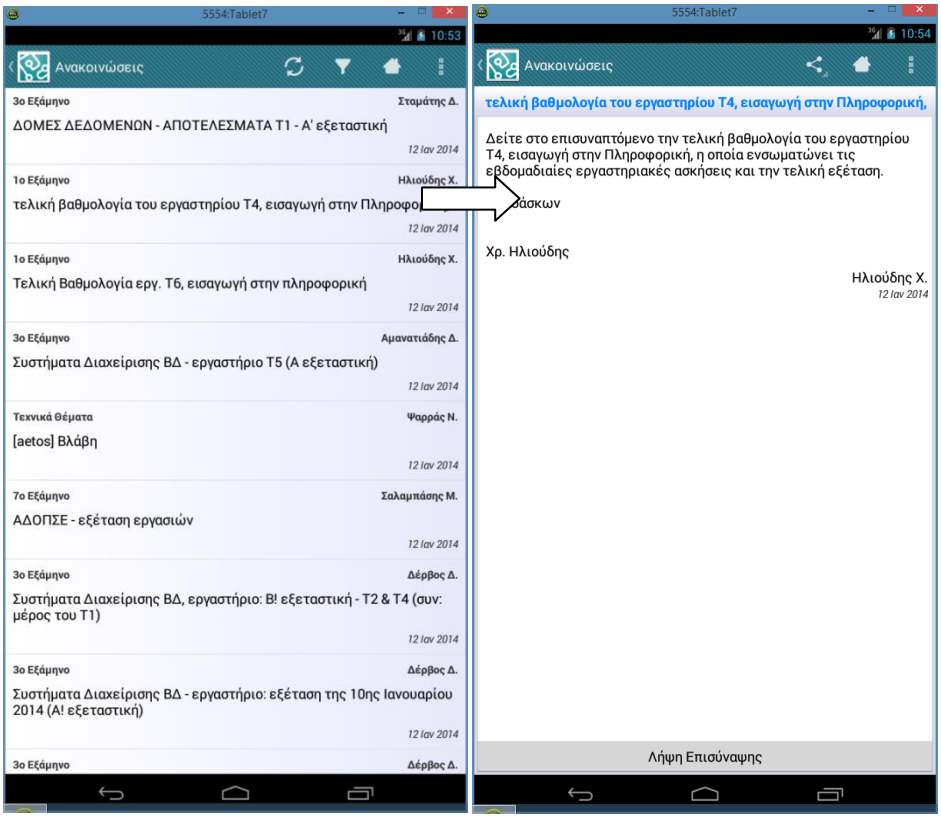
Η δραστηριότητα των ανακοινώσεων αποτελείτε από δύο `Fragments` τα `BulletinListFragment` για την προβολή της λίστας των ανακοινώσεων και `BulletinDetailFragment` για την προβολή της επιλεγμένης από τον χρήστη ανακοίνωσης.

Η ActionBar παρέχει action items για φιλτράρισμα κατά κατηγορία και ανανέωση των αποτελεσμάτων καθώς και `ShareActionProvider` για την δυνατότητα κοινοποίησης των ανακοινώσεων με εγκατεστημένες εφαρμογές που χειρίζονται την αποστολή μηνυμάτων.

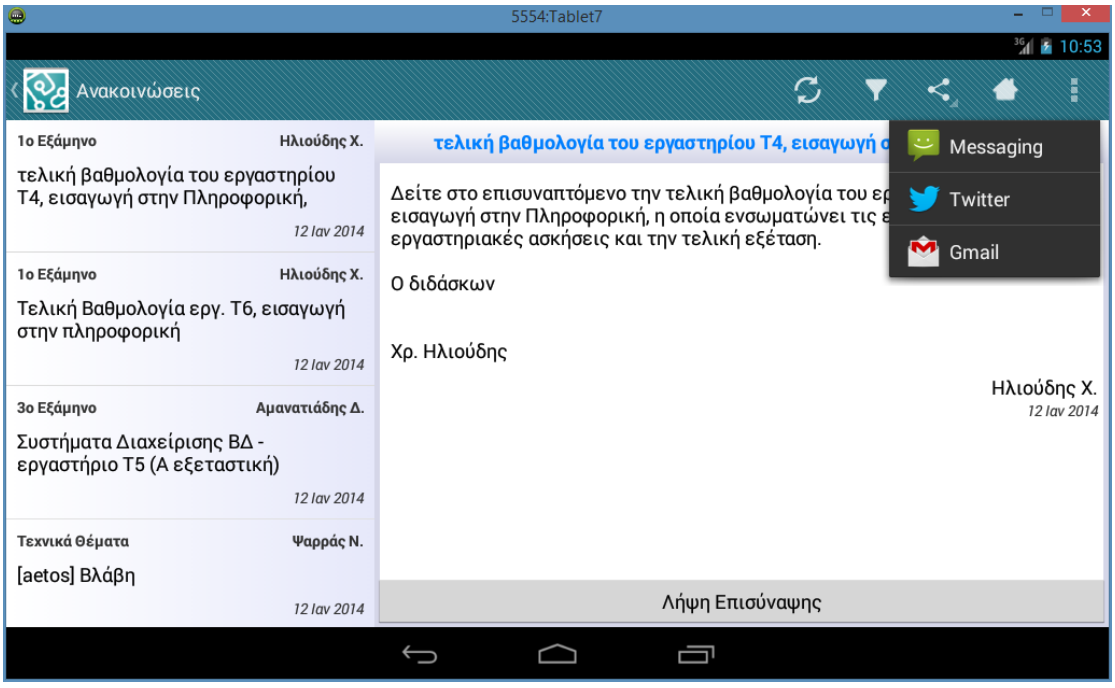
Το γέμισμα και φιλτράρισμα της λίστας, ένα αντικείμενο `ListView`, γίνεται με την χρήση ενός `ListAdapter` αντικειμένου.

```
public class BulletinListAdapter extends ArrayAdapter<Bulletin> {
   private List<Bulletin> originalList;
   private List<Bulletin> bulletinList;
   private Filter filter;
   private final Object lock = new Object();

   public BulletinListAdapter(Context context, int resource, List<Bulletin> list) {
      super(context, resource, list);
      this.bulletinList = list;
      originalList = new ArrayList<Bulletin>(bulletinList);
   }

   @Override
   public int getCount() {
      return bulletinList.size();
   }

   @Override
   public Filter getFilter() {
      if (filter == null) filter = new ArrayFilter();

      return filter;
   }

   @Override
   public Bulletin getItem(int position) {
      return (bulletinList.size() <= position) ? null : bulletinList.get(position);
   }

   @Override
   public View getView(int position, View convertView, ViewGroup parent) {
      View v = convertView;
      Bulletin p = getItem(position);

      if (p != null) {
         // Inflate new view if there is no reusable convertView
        if (v == null) {
           v = LayoutInflater.from(getContext()).inflate(R.layout.row_bulletin_list, null);
        }

        TextView t1 = (TextView) v.findViewById(R.id.etAuthor);
        TextView t2 = (TextView) v.findViewById(R.id.etTitle);
           TextView t3 = (TextView) v.findViewById(R.id.etBoard);
        TextView t4 = (TextView) v.findViewById(R.id.etDate);
```

```
            t1.setText(p.getAuthor());
            t2.setText(p.getTitle());
             t3.setText(p.getBoard());
            t4.setText(p.getDate());
         }

         return v;
      }

      @Override
      public boolean isEmpty() {
         return bulletinList.isEmpty();
      }

      private class ArrayFilter extends Filter {
         @Override
         protected FilterResults performFiltering(CharSequence filter) {
            String[] filters = filter.toString().split(":");
            FilterResults results = new FilterResults();
            List<Bulletin> originalValues;

            synchronized (lock) {
               originalValues = new ArrayList<Bulletin>(originalList);
            }

            if (filter == null || filter.length() == 0) {
               results.values = originalValues;
               results.count = originalValues.size();
            } else {
               final List<Bulletin> newValues = new ArrayList<Bulletin>();

               for (Bulletin b : originalValues) {
                  for (String f : filters) {
                     if (b.getBoard().equals(f)) {
                        newValues.add(b);
                     }
                  }
               }

               results.values = newValues;
               results.count = newValues.size();
            }
            return results;
         }

         @SuppressWarnings("unchecked")
         @Override
         protected void publishResults(CharSequence constraint, FilterResults results) {
            bulletinList = (List<Bulletin>) results.values;
            if (results.count > 0) {
               notifyDataSetChanged();
            } else {
               notifyDataSetInvalidated();
            }
         }
      }
   }
```



Εικόνα 31: Εμφάνιση σε διάταξη ενός παραθύρου(1 pane)



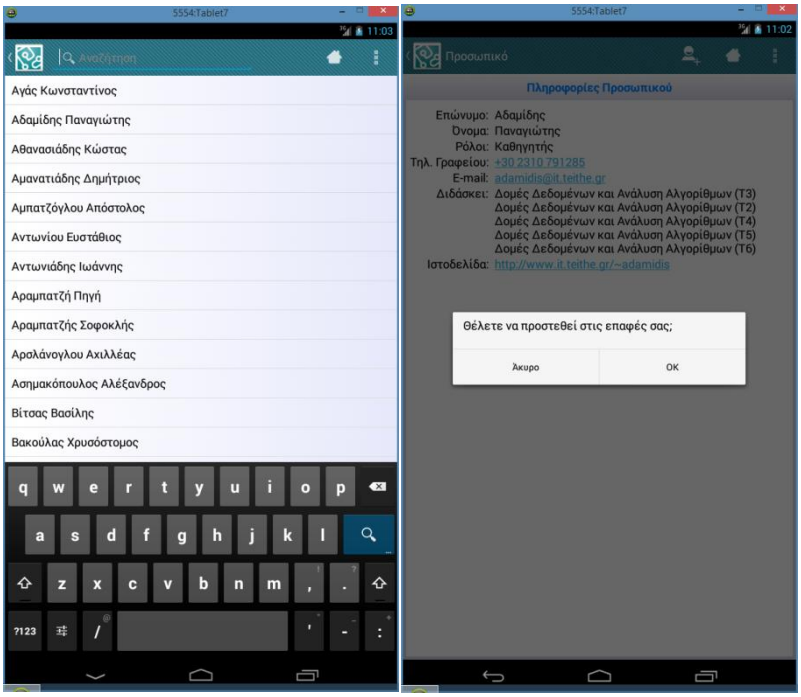
Εικόνα 32: Εμφάνιση σε διάταξη δύο παραθύρων(2-pane)

6.4.2 Δραστηριότητα προσωπικού

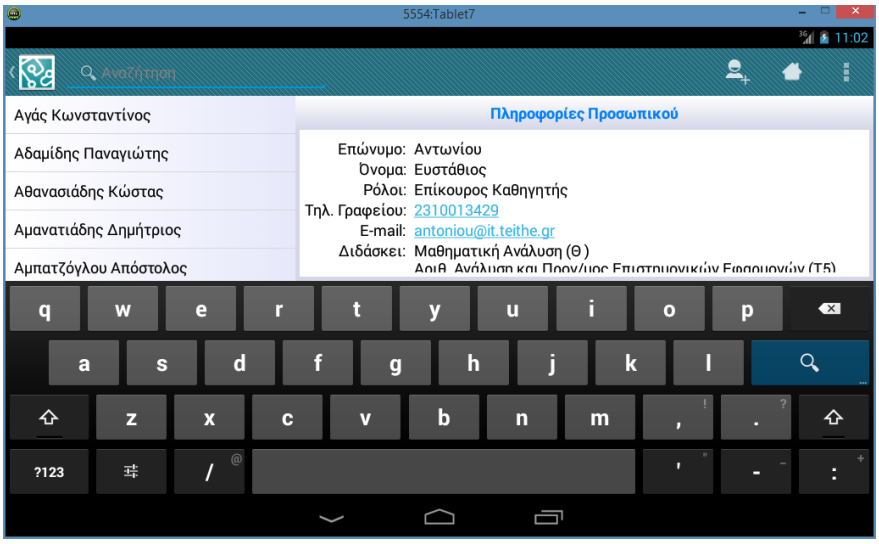
Η διάταξη της δραστηριότητας του προσωπικού δεν διαφέρει ιδιαίτερα από αυτήν των ανακοινώσεων καθώς και εδώ έχουμε χρήση δύο Fragments, ένα για την λίστα και ένα για την προβολή των λεπτομερειών.

Η ActionBar εδώ χρησιμοποιεί ένα action view και συγκεκριμένα ένα SearchView για την αναζήτηση στη λίστα του προσωπικού. Επίσης υπάρχει ένα action item με το οποίο προσθέτουμε το μέλος του προσωπικού στις επαφές μας. Αυτό επιτυγχάνεται με την χρήση του Contacts ContentProvider του συστήματος τη χρήση του οποίου είδαμε παραπάνω.

Η εμφάνιση της δραστηριότητας σε διάταξη 1 και 2 παραθύρων:



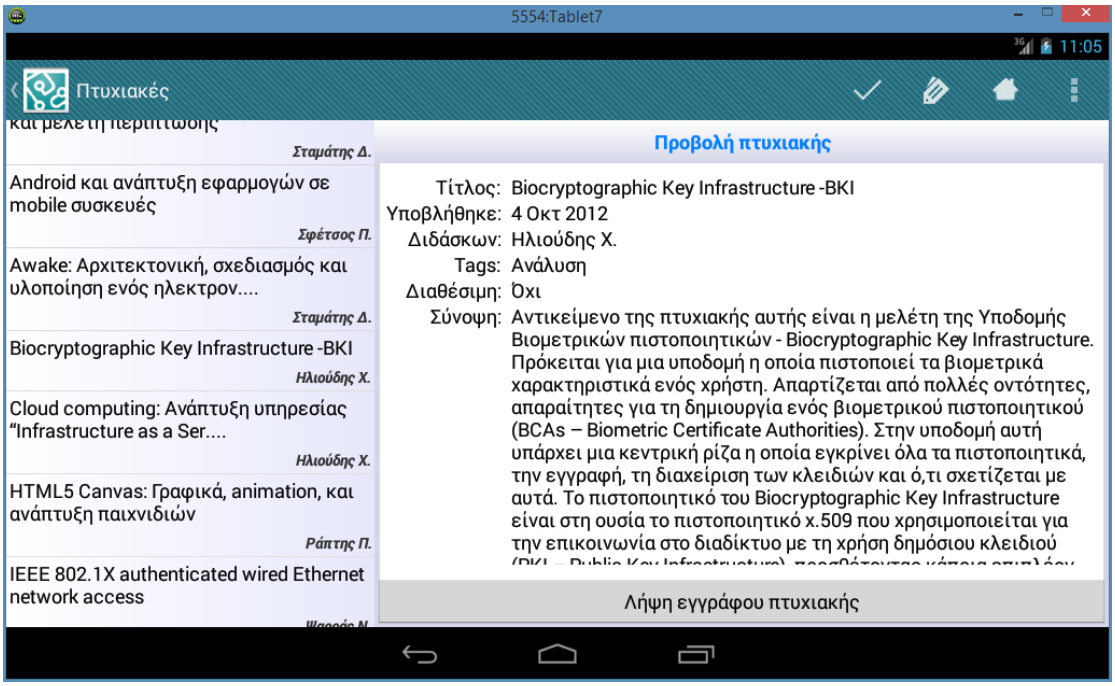
Εικόνα 33: Εμφάνιση σε διάταξη ενός παραθύρου(1-pane)



Εικόνα 34Εμφάνιση σε διάταξη δύο παραθύρου(2-pane)

6.4.3 Δραστηριότητα πτυχιακών εργασιών

Όπως και η προηγούμενες χρησιμοποιεί μια master-detail διάταξη με τη χρήση δύο Fragments. Η ActionBar παρέχει δυο αντικείμενα για φιλτράρισμα των πτυχιακών με βάση την διαθεσιμότητα του και της κατηγορίες στις οποίες ανήκει(tags).



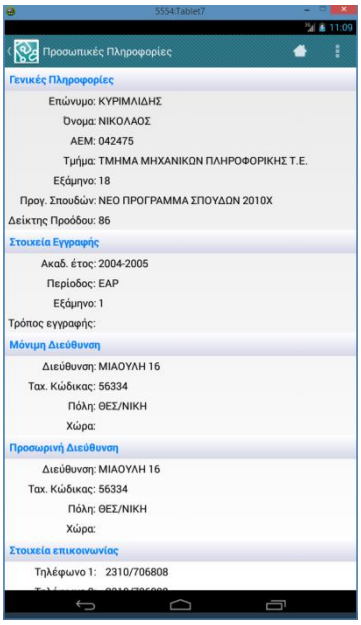
6.5 Λειτουργίες υπηρεσιών Πυθίας

Για τις υπηρεσίες της Πυθίας παρέχονται 3 ξεχωριστές δραστηριότητες υλοποιημένες όμως με `Fragments` παρά την απλότητα τους και μια δραστηριότητα επισκόπησης που ανάλογα με την διαμόρφωση της συσκευής παρουσιάζει το περιεχόμενο σε διαφορετική διάταξη.

- `PersonalInfoActivity`: Οι προσωπικές πληροφορίες του φοιτητή.
- `GradesActivity`: Οι βαθμολογίες του φοιτητή.
- `PithiaRequestActivity`: Οι ολοκληρωμένες αιτήσει για τις οποίες ο φοιτητής μπορεί να κατεβάσει το pdf.
- `PithiaOverviewActivity`: Μια δραστηριότητα επισκόπησης η οποία εμφανίζει τις πληροφορίες των προηγουμένων ταυτόχρονα.

6.5.1 Δραστηριότητα προσωπικών πληροφοριών

Μια απλή δραστηριότητα με ένα μόνο Fragment για την εμφάνιση των πληροφοριών. Κατά την φόρτωση ορίζουμε τα στοιχεία κειμένου(`TextView`) που προϋπάρχουν στη διάταξη μας καλώντας την `setText`() κάθε στοιχείου.



Εικόνα 35: Προσωπικές πληροφορίες

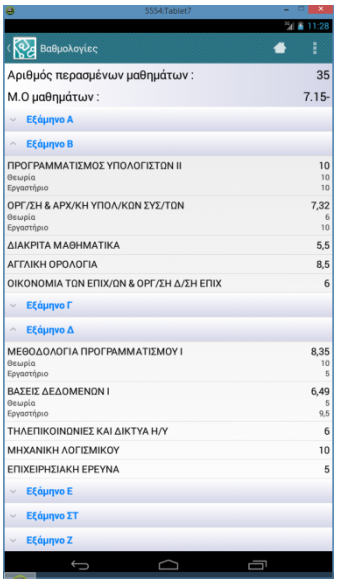
6.5.2 Δραστηριότητα βαθμολογιών

Ομοίως με την προηγούμενη ένα `Fragment` εδώ όμως τα δεδομένα είναι πιο περίπλοκα και χρησιμοποιούμε ένα `ExpandableListAdapter` μια ειδική κατηγορία `ListAdapters` για να «γεμίσουμε» την «λίστα από λίστες» των μαθημάτων. Κάθε μάθημα ανήκει σε ένα εξάμηνο και όλα μαζί εμφανίζονται ομαδοποιημένα.

Ο κώδικας του `ExpandableListAdapter`.

```
public class GradeListAdapter extends BaseExpandableListAdapter {
   private Context context;
   private List<GradeList> list;

   public GradeListAdapter(Context context, List<GradeList> gradeList) {
      this.context = context;
      this.list = gradeList;
   }

   @Override
   public Object getChild(int groupPosition, int childPosition) {
      return list.get(groupPosition).get(childPosition);
   }

   @Override
   public long getChildId(int groupPosition, int childPosition) {
      return childPosition;
   }

   @Override
   public int getChildrenCount(int groupPosition) {
      return list.get(groupPosition).size();
   }

   @Override
   public View getChildView(int groupPosition,
               int childPosition, boolean isLastChild,
               View convertView, ViewGroup parent) {
      View v = convertView;

      if (v == null) {
                  v = LayoutInflater.from(context).inflate(R.layout.row_grade_list, null);
      }

      TextView t1 = (TextView) v.findViewById(R.id.tvCourse);
```

```
        TextView t2 = (TextView) v.findViewById(R.id.tvAVG);
        TextView t3 = (TextView) v.findViewById(R.id.tvTheoryGrade);
        TextView t4 = (TextView) v.findViewById(R.id.tvLabGrade);
        TextView t5 = (TextView) v.findViewById(R.id.tvTheory);
        TextView t6 = (TextView) v.findViewById(R.id.tvLab);

        Grade g = list.get(groupPosition).get(childPosition);

        t1.setText(g.getCourse());
        t2.setText(g.getAvgGrade());
        if (g.getTheoryGrade() != null) {
           t3.setText(g.getTheoryGrade());
           t4.setText(g.getLabGrade());
           t3.setVisibility(View.VISIBLE);
           t4.setVisibility(View.VISIBLE);
           t5.setVisibility(View.VISIBLE);
           t6.setVisibility(View.VISIBLE);
        } else {
           t3.setVisibility(View.GONE);
           t4.setVisibility(View.GONE);
           t5.setVisibility(View.GONE);
           t6.setVisibility(View.GONE);
        }

        return v;
    }

    @Override
    public Object getGroup(int groupPosition) {
        return list.get(groupPosition);
    }

    @Override
    public int getGroupCount() {
        return list.size();
    }

    @Override
    public long getGroupId(int groupPosition) {
        return groupPosition;
    }

    @Override
    public View getGroupView(
                int groupPosition, boolean isExpanded,
                View convertView, ViewGroup parent) {
        View v = convertView;

        if (v == null) {
                  v = LayoutInflater.from(context).inflate(R.layout.row_heading_list, null);
        }

        TextView t1 = (TextView) v.findViewById(R.id.text1);
        t1.setText(list.get(groupPosition).getSemester());

        return v;
    }

    @Override
    public boolean hasStableIds() {
        return true;
    }

    @Override
    public boolean isChildSelectable(int arg0, int arg1) {
        return false;
    }
}
```



Εικόνα : Βαθμολογία φοιτητή

6.5.3 Δραστηριότητα αιτήσεων

Μια διάταξη δραστηριότητας με ένα `ListFragment`, ένας τύπος `Fragment` που διευκολύνει την εμφάνιση δεδομένων σε λίστα παρέχοντας βοηθητικές μεθόδους και χωρίς την ανάγκη να δηλώσουμε το αρχείο διάταξης του.

Εικόνα 36: Λίστα αιτήσεων

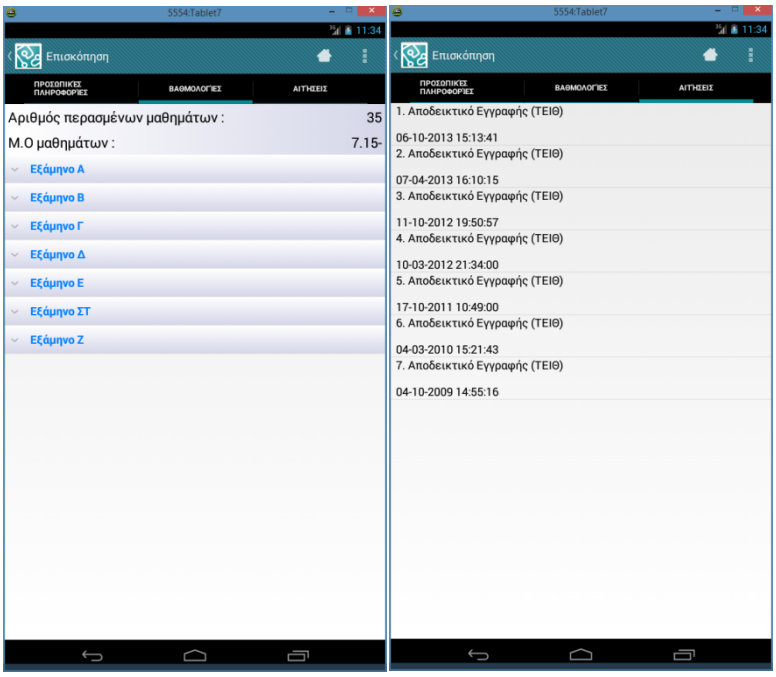
6.5.4 Δραστηριότητα επισκόπησης Πυθίας

Παρουσιάζει όλα μαζί τα Fragments που αποτελούσαν τις προηγούμενες δραστηριότητες. Υλοποιήθηκε για να δείξει ακριβώς τις δυνατότητες που μας παρέχουν τα Fragments όχι μόνο να παρέχουμε εύκολα διαφορετικές διατάξεις για την ίδια δραστηριότητα αλλά και να επαναχρησιμοποιούμε τις λειτουργίες που παρέχουν σε διαφορετικές.

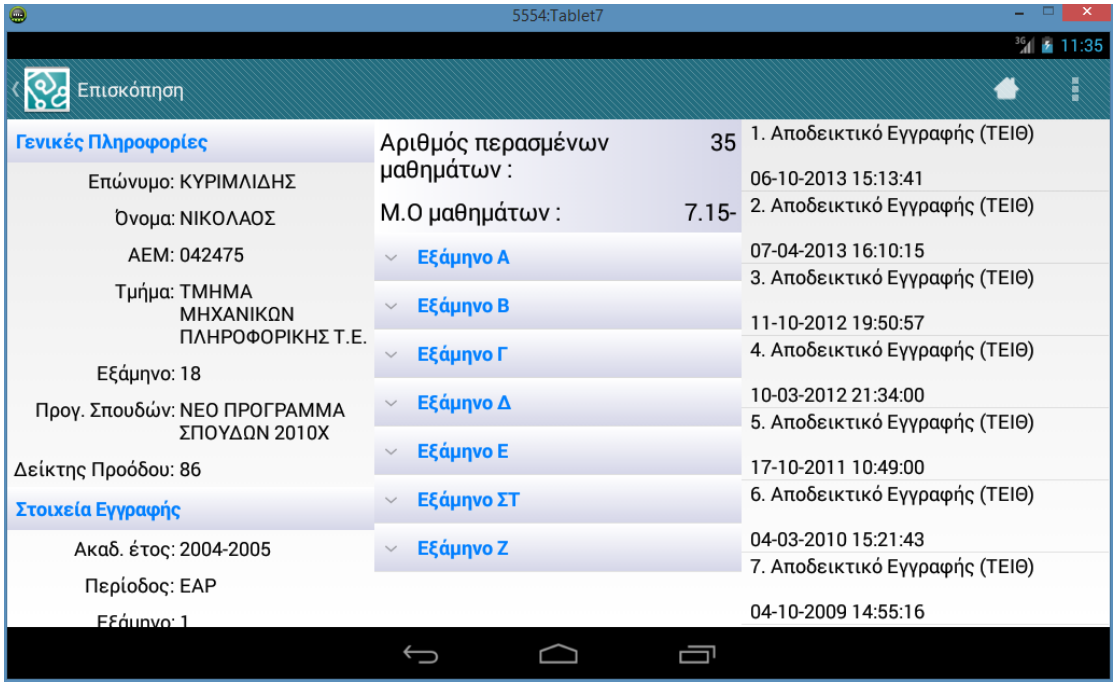
Η διάταξη επιτυγχάνεται με τη χρήση των ActionBar tabs και των συνδυασμό `ViewPager` και `FragmentStatePagerAdapter` που είδαμε στην παράγραφο για την ActionBar.

Εδώ η δυνατότητα πολλαπλών διατάξεων δεν επιτυγχάνεται με την χρήση πολλαπλών αρχείων αλλά με υπέρβαση της μεθόδου του `PagerAdapter` που επιστρέφει το πλάτος των «σελίδων»

επιτρέποντας την εμφάνιση, και εναλλαγή, πολλαπλών σελίδων ταυτόχρονα. Εμφανίζεται δηλαδή μια διάταξη 3 παραθύρων καθώς η δραστηριότητα εμφανίζει όλες τις σελίδες ταυτόχρονα.



Εικόνα 37: Διάταξη με χρήση tabs



Εικόνα 38: Διάταξη τριών παραθύρων(3-pane)

6.6 Λειτουργίες πρόσθετων υπηρεσιών

Εδώ περιλαμβάνονται διάφορες υπηρεσίες που δεν ανήκουν σε κάποια από τις άλλες δύο κατηγορίες.

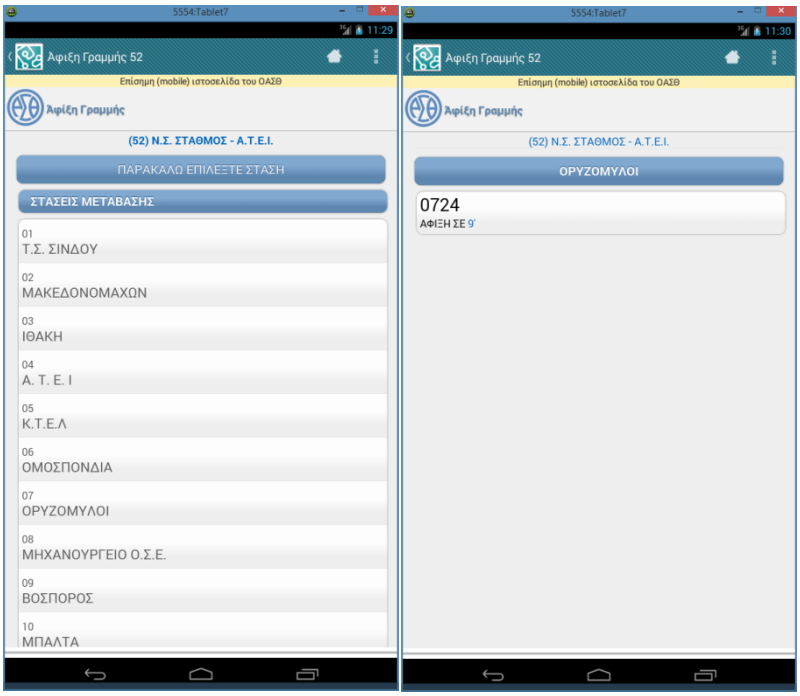
- `BusLineActivity`: Προβολή των χρόνων άφιξης της γραμμής 52.

- `MapActivity`: Ο χάρτης της σχολής και πλοήγηση προς αυτήν.
- `MailActivity`: Προβολή των εισερχομένων e-mail.

6.6.1 Δραστηριότητα γραμμής ΑΤΕΙΘ

Μια απλή δραστηριότητα που περιέχει μόνο ένα στοιχείο `WebView`, ένα ελαφρύ ενσωματωμένο στην εφαρμογή μας web browser, ο οποίος εμφανίζει την mobile ιστοσελίδα του ΟΑΣΘ για την γραμμή 52. Το `WebView` χρειάζεται κάποιες ρυθμίσεις για να χειριστεί την JavaScript της ιστοσελίδας και χειριζόμαστε το πλήκτρο back ώστε να επιτρέψουμε στον χρήστη να επιστρέψει σε προηγούμενη σελίδα και να μην «βγει» από την δραστηριότητα. Κατά την διάρκεια της φόρτωσης εμφανίζεται «ένα χαρακτηριστικό παραθύρου», ένα progress bar στην περίπτωση μας, για την ενημέρωση του χρήστη.

```
public class BusLineActivity extends BaseActivity {
   private WebView web;

   @SuppressLint("SetJavaScriptEnabled")
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      supportRequestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
      setContentView(R.layout.activity_bus_line);
      setSupportProgressBarIndeterminateVisibility(true);
      getSupportActionBar().setDisplayHomeAsUpEnabled(true);

      web = (WebView) findViewById(R.id.web);
      web.getSettings().setJavaScriptEnabled(true);
      WebViewClient c = new WebViewClient() {
         @Override
         public void onPageFinished(WebView view, String url) {
            setSupportProgressBarIndeterminateVisibility(false);
         }
      };
      web.setWebViewClient(c);
      web.loadUrl(Constants.URLS.OASTH.toString());
   }

   @Override
   public boolean onKeyDown(int keyCode, KeyEvent event) {
      if (event.getAction() == KeyEvent.ACTION_DOWN) {
         if (keyCode == KeyEvent.KEYCODE_BACK) {
            if (web.canGoBack() == true) web.goBack();
            else finish();
         }

         return true;
      }
      return super.onKeyDown(keyCode, event);
   }
}
```



Εικόνα 39: Άφιξη γραμμής 52

6.6.2 Δραστηριότητα χάρτη ΑΤΕΙΘ

Εδώ χρησιμοποιούμε το Google Maps API v2 του Android και την web υπηρεσία Directions API της Google για να εμφανίσουμε τον χάρτη της σχολής και να υπολογίζουμε την διαδρομή προς το ΑΤΕΙΘ από το σημείο στο οποίο βρίσκεται ο χρήστης κάνοντας χρήση της θέσης του βάση του GPS.

6.6.2.1 Google Map Api

Μία από τις γνωστότερες υπηρεσίες της Google είναι η υπηρεσία Maps και όπως είναι λογικό την βρίσκουμε και στο Android. Για να έχουμε πρόσβαση στο Google Maps API στην εφαρμογή μας θα πρέπει πρώτα να εισάγουμε την βιβλιοθήκη Google Play Services η οποία μας δίνει πρόσβαση σε διάφορες υπηρεσίες της Google και όχι μόνο στους χάρτες.

Το Google Maps Android API v2 επιτρέπει να εμφανίσουμε ένα χάρτη του Google στην εφαρμογή μας. Οι χάρτες αυτοί έχουν την ίδια εμφάνιση όπως οι χάρτες που βλέπουμε στην επίσημη εφαρμογή και το API εκθέτει πολλά από τα ίδια χαρακτηριστικά. Η βασική κλάση όταν εργαζόμαστε με ένα αντικείμενο χάρτη είναι η κλάση `GoogleMap`. Αυτή η κλάση μοντελοποιεί το αντικείμενο του χάρτη μέσα στην εφαρμογή μας.

Ένα `GoogleMap` αντικείμενο χειρίζεται τις παρακάτω ενέργειες αυτόματα:

- Σύνδεση με την υπηρεσία Google Maps.
- Λήψη χάρτη.
- Εμφάνιση του χάρτη στην οθόνη της συσκευής.
- Εμφάνιση διάφορων στοιχείων ελέγχου, πχ ζουμ, καθώς και τον χειρισμό τους.

Μέσα στο UI μας ένας χάρτης θα παριστάτε είτε ως ένα αντικείμενο `MapFragment` ή `MapView` το οποίο θα μας παρέχει πρόσβαση στο `GoogleMap` αντικείμενο. Στην εφαρμογή μας

θα χρησιμοποιήσουμε το `MapFragment` από την βιβλιοθήκη υποστήριξης συγκεκριμένα δηλαδή το `SupportMapFragment`.

Προσθέτοντας ένα χάρτη στην εφαρμογή μας

Καταρχήν θα πρέπει να αποκτήσουμε ένα κλειδί για να μπορούμε να χρησιμοποιήσουμε την συγκεκριμένη υπηρεσία στην εφαρμογή μας από το Google Developer Console και να το εισάγουμε στο manifest μαζί με τα απαραίτητα δικαιώματα.

Τώρα θα δημιουργήσουμε το αρχείο διάταξης για την δραστηριότητα μας. Εδώ βλέπουμε το αρχείο για τη διάταξη 2 παραθύρων, το δεύτερο παράθυρο θα εμφανίζει τις οδηγίες πλοήγησης που θα δούμε παρακάτω. Όπως βλέπουμε δεν έχει καμία διαφορά από τον ορισμό των απλών Fragments. Το `MapFragment` είναι η κλάση η οποία επεκτείνει το `SupportMapFragment`.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:baselineAligned="false"
    android:orientation="horizontal" >

    <fragment
        android:id="@+id/fragMap"
        android:name="com.nkyrim.itapp.ui.fragments.MapFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1" />

    <FrameLayout
        android:id="@+id/fragDirCon"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_weight="2"
        android:visibility="gone" >

        <fragment
            android:id="@+id/fragDir"
            android:name="com.nkyrim.itapp.ui.fragments.DirectionsFragment"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />
    </FrameLayout>

</LinearLayout>
```

Εδώ βλέπουμε μέρος της υλοποίησης της κλάσης μας. Κατά την εκκίνηση αρχικοποιούμε το χάρτη μας ορίζοντας το είδος του, την θέση του, το επίπεδο του ζουμ. Εμφανίζουμε επίσης τις τοποθεσίες και την διαδρομή αν είχε επιλεχτεί πρωτύτερα από τον χρήστη, εάν δηλαδή η δραστηριότητα έχει ξαναδημιουργηθεί.

```
public class MapFragment extends SupportMapFragment {
  @Override
  public void onActivityCreated(Bundle savedInstanceState) {
   super.onActivityCreated(savedInstanceState);
   map = getMap();
   map.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
   map.moveCamera(CameraUpdateFactory.newLatLng(
                      new LatLng(40.656, 22.804)));
   map.moveCamera(CameraUpdateFactory.zoomTo(16.5F));
   map.setMyLocationEnabled(true);

   if (savedInstanceState != null) {
     checkedItem = savedInstanceState.getInt("markers");
     route = (Route) unbundleParcelable("route", savedInstanceState);
```

```
      showMarkers(checkedItem);
      if (route != null) {
        PolylineOptions op = new PolylineOptions();
        op.addAll(route.getPoints()).width(3).color(Color.RED);
           map.addPolyline(op);
    }
   }
...
...
  private void showMarkers(int markers) {
   map.clear();

   switch (markers) {
   case ALL:
      showDeps();
      showSecs();
      showOther();
      break;
   case DEPARTMENTS:
      showDeps();
      break;
   case SECRETARIES:
      showSecs();
      break;
   case OTHER:
      showOther();
      break;
    }
   }
...

}
```

Για να προσθέσουμε τώρα τις ετικέτες με τις τοποθεσίες το χάρτη δημιουργούμε αντικείμενα της κλάσης `MarkerOptions` με τις ιδιότητες που θέλουμε, τίτλος, εικονίδιο, τοποθεσία, και τα προσθέτουμε στο χάρτη μας.

```
private void showOther() {
  String[] array = getResources().getStringArray(R.array.other);

  BitmapDescriptor bd = BitmapDescriptorFactory
                        .defaultMarker(BitmapDescriptorFactory.HUE_AZURE);

  map.addMarker(new MarkerOptions()
       .icon(bd)
       .position(new LatLng(40.65885499081485, 22.80369697794211))
       .title(array[0]));
  map.addMarker(new MarkerOptions()
       .icon(bd)
       .position(new LatLng(40.65923063121845, 22.80359420984222))
       .title(array[1]));
  map.addMarker(new MarkerOptions()
       .icon(bd)
       .position(new LatLng(40.65558223168065, 22.80401688849685))
       .title(array[2]));
  ...
  ...
}
```

6.6.2.3 Google Directions API

Το Google Directions API είναι μια υπηρεσία ιστού που υπολογίζει και επιστρέφει την διαδρομή μεταξύ των τοποθεσιών χρησιμοποιώντας ένα αίτημα HTTP. Μπορούμε να αναζητήσουμε τις οδηγίες για διάφορους τρόπους μεταφοράς όπως περπάτημα ή οδήγηση. Οι οδηγίες μπορούν να

καθορίζουν την προέλευση, τους προορισμούς και σημεία είτε ως συμβολοσειρές κειμένου, π.χ. «Μιαούλη Κορδελιό ,Ελλάδα» ή ως συντεταγμένες γεωγραφικού πλάτους/μήκους.

Πραγματοποίηση αιτήσεων

Για να κάνουμε μια αίτηση στην υπηρεσία ακολουθούμε την παρακάτω φόρμα στην HTTP αίτηση μας:

```
http://maps.googleapis.com/maps/api/directions/output?parameters
```

Το αποτέλεσμα μπορεί να επιστραφεί σε δύο μορφές αναλόγως με το τι θα έχουμε επιλέξει(η τιμή που θα θέσουμε στη θέση του **output**):

- json: (συνιστάται) δίνει έξοδο σε JavaScript Object Notation (JSON).
- xml: δίνει έξοδο ως XML.

Παράμετροι αίτησης

Ορισμένες παράμετροι απαιτούνται ενώ άλλες είναι προαιρετικές. Όπως είναι το στάνταρ στα URLs όλες οι παράμετροι διαχωρίζονται χρησιμοποιώντας το εμπορικό και (&). Ο κατάλογος των παραμέτρων και οι πιθανές τιμές τους δίνονται παρακάτω.

Απαιτούμενες παράμετροι

- origin: Η διεύθυνση ή τιμή γεωγραφικού πλάτους/μήκους της προέλευσης από την οποία θέλουμε να υπολογίσουμε τις κατευθύνσεις.
- destination: Η διεύθυνση ή τιμή γεωγραφικού πλάτους/μήκους του προορισμού προς τον οποίο θέλουμε να υπολογίσουμε τις κατευθύνσεις.
- sensor: Υποδεικνύει εάν το αίτημα προέρχεται από μια συσκευή με αισθητήρα θέσης. Αυτή η τιμή πρέπει να είναι true ή false.

Προαιρετικές παράμετροι

Δεν θα αναλύσουμε όλες τις παραμέτρους θα αναφερθούμε μόνο σε όσες χρησιμοποιούνται στην εφαρμογή μας:

- units: Οι μονάδες μέτρησης που θα χρησιμοποιηθούν για την εμφάνιση των αποτελεσμάτων.
- language: Η γλώσσα του κειμένου που θα χρησιμοποιηθεί για την εμφάνιση των αποτελεσμάτων.

Επιγραμματικά οι υπόλοιπες παράμετροι: mode, waypoints, alternatives, avoid, region, departure_time, arrival_time.

Αποτέλεσμα αίτησης

Ανεξαρτήτως του είδους της εξόδου που θα επιλέξουμε το αποτέλεσμα της αίτησης θα αποτελείται από τα ίδια στοιχεία με μικρές μόνο διαφορές. Για την εφαρμογή μας χρησιμοποιήσαμε έξοδο σε JSON.

Υπάρχουν δύο βασικά στοιχεία ρίζας:

- `status`: Περιλαμβάνει μεταδεδομένα για την γενικό αποτέλεσμα της αίτησης, πχ OK αν το αποτέλεσμα είναι έγκυρο.

- `routes`: Περιλαμβάνει ένα πίνακα από διαδρομές από την αφετηρία στον προορισμό.

Τα στοιχεία του στοιχείου routes περιλαμβάνουν τις πληροφορίες για την διαδρομή μας. Παρ' όλο που περιλαμβάνονται πολλές πληροφορίες θα αναλύσουμε ιεραρχικά μόνο αυτές που χρησιμοποιούνται στην εφαρμογή:

- routes[]: Το αποτέλεσμα μας θα περιλαμβάνει μόνο μία διαδρομή.
  - legs[]: Ένα τμήμα της διαδρομής. Θα είναι μόνο ένα καθώς δεν ορίσαμε waypoints.
    - duration: Η διάρκεια της διαδρομής.
    - distance: Η απόσταση της διαδρομής.
    - origin: Η αφετηρία της διαδρομής(διεύθυνση).
    - destination: Ο προορισμός της διαδρομής(διεύθυνση).
    - steps[]: Κομμάτια του «ποδιού», αυτές είναι ουσιαστικά οι οδηγίες.
  - polyline_overview: Συντεταγμένες των σημείων για την εμφάνιση της διαδρομής στο χάρτη.

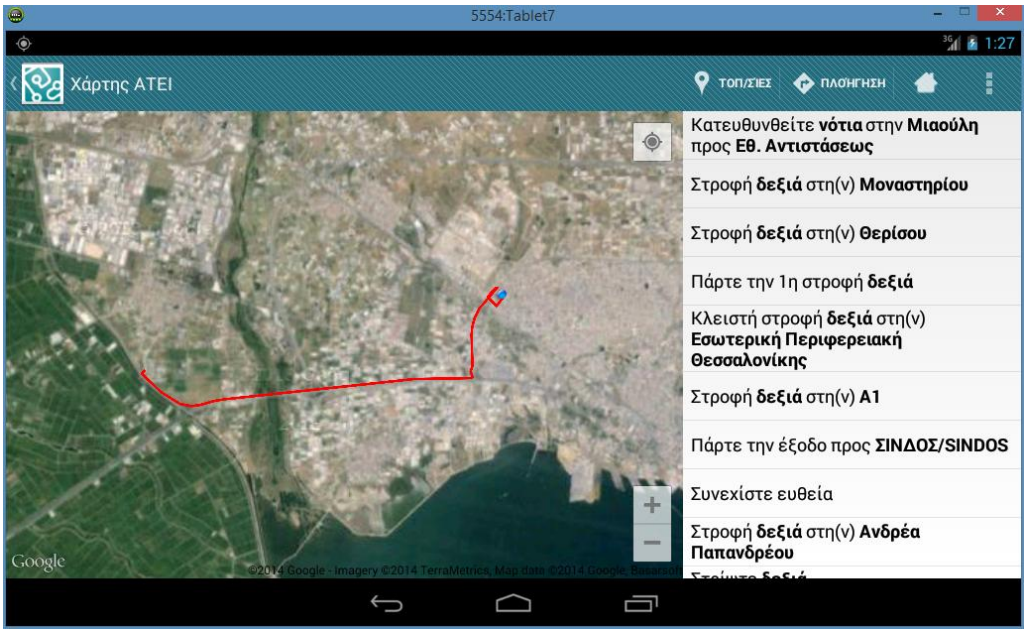
Αυτός είναι ο κώδικας του `AsyncTask` που αναλαμβάνει την λήψη των δεδομένων και της μεθόδου `createRoute`() που δημιουργεί ένα αντικείμενο τύπου `Route` το όποιο και θα χρησιμοποιήσουμε στην εφαρμογή μας να εμφανίσουμε τις οδηγίες και τη διαδρομή στο χάρτη.

```
public class DirectionsTask extends AsyncTask<Void, Void, Route>() {
   @Override
   protected Route doInBackground(Void... params) {
      String request = "https://maps.googleapis.com/maps/api/directions/json?origin="
                + String.valueOf(loc.getLatitude()) + ","
      + String.valueOf(loc.getLongitude()) + "&destination="
                + String.valueOf(tei.latitude) + "," + String.valueOf(tei.longitude)
      + "&sensor=true&language=el&units=metric";

      JSONObject responce = ItHttpClient.getDirections(request);

      return Route.createRoute(responce);
   }

   @Override
   protected void onPostExecute(Route result) {
      if (result == null) {
         Toast.makeText(getActivity(),
                                 getString(R.string.no_location),
                                 Toast.LENGTH_LONG).show();
         return;
      }

      route = result;

      PolylineOptions op = new PolylineOptions();
      op.addAll(route.getPoints()).width(3).color(Color.RED);
      map.addPolyline(op);

      ShowDirections(route);
   }
}
```

```
public static Route createRoute(JSONObject gmJSON) {
   String distance, duration, origin, destination;
   ArrayList<String> directions = new ArrayList<String>();
   ArrayList<LatLng> points;

   try {
      String status = gmJSON.getString("status");
      if (!status.equals("OK")) return null;
      // we get only one route and one leg since we don't use waypoints
      JSONObject route = gmJSON.getJSONArray("routes").getJSONObject(0);
      JSONObject leg = route.getJSONArray("legs").getJSONObject(0);
      // get the steps
      duration = leg.getJSONObject("duration").getString("text");
      distance = leg.getJSONObject("distance").getString("text");
      origin = leg.getString("start_address");
      destination = leg.getString("end_address");
```

```
        JSONArray steps = leg.getJSONArray("steps");
        directions = new ArrayList<String>(steps.length());
        for (int i = 0; i < steps.length(); i++) {
           String s=steps.getJSONObject(i).getString("html_instructions");
           directions.add(s);
        }

        JSONObject polyline = route.getJSONObject("overview_polyline");
        points = decodePoly(polyline.getString("points"));

     } catch (JSONException e) {
        return null;
     }

     return new Route(distance, duration, origin, destination, directions, points);
}
```

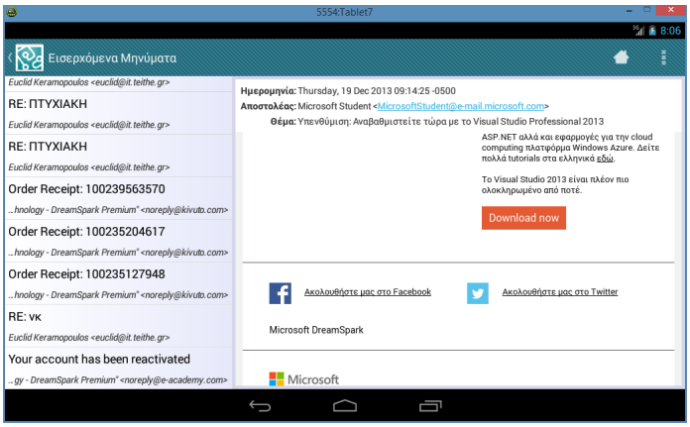
Εικόνα 40: Εμφάνιση σημείων ενδιαφέροντος



Εικόνα 41: Διαδρομή και οδηγίες

6.6.3 Δραστηριότητα εισερχομένων e-mail

Για την υλοποίηση της δραστηριότητας χρησιμοποιήθηκε μια μεταφορά(port) της JAVA βιβλιοθήκης Java Mail στο Android. Παρόλο που δεν υποστηρίζονται όλες οι λειτουργίες της αρχικής βιβλιοθήκης υπερκαλύπτουν τις ανάγκες μας. Η διάταξη υλοποιείται και εδώ με την χρήση δύο Fragments, ένα για την εμφάνιση της λίστας και ένα για την προβολή του μηνύματος.



6.7 Λειτουργίες πληροφοριών

Οι τέσσερις δραστηριότητες που παρέχουν στατικές πληροφορίες υλοποιούνται με τον ίδιο τρόπο χρησιμοποιώντας στο αρχείο διάταξης μας το `ViewPager` σε συνδυασμό με ένα `FragmentPagerAdapter` που θα διαχειρίζεται τα `Fragments` που εμφανίζουν το περιεχόμενο και θα περιέχει ένα `PagerTitleStrip` για να εμφανίζει τον τίτλο κάθε `Fragment`.

- `LinksActivity`
- `SupportActivity`
- `DepartmnetActivity`
- `PhotosActivity`

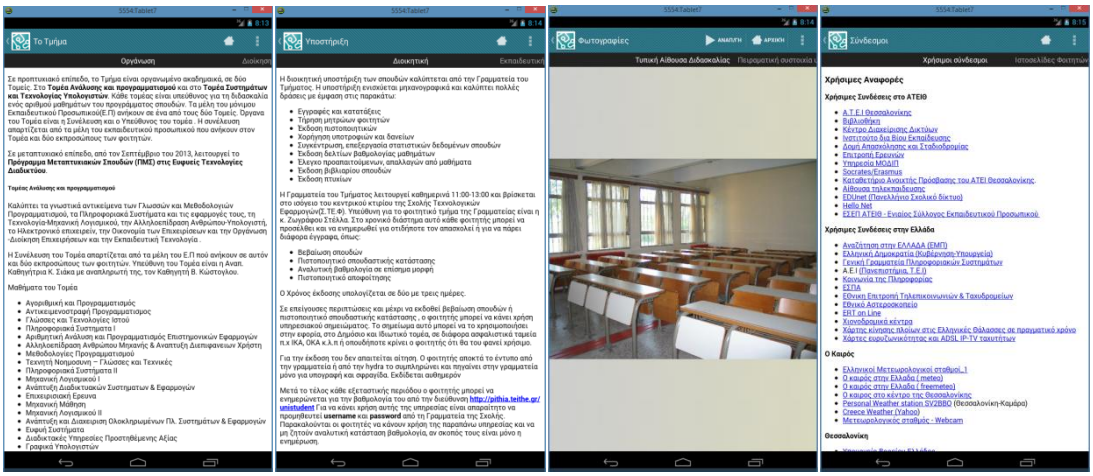
Το αρχείο διάταξης για όλες της δραστηριότητες(`res/layout/activity_viewgager`)

```
<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"
    android:background="@drawable/bg3" >

   <android.support.v4.view.PagerTabStrip
        android:id="@+id/pager_tab_strip"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="top"
        android:background="#000"
        android:padding="10dp"
        android:textColor="#fff" />

</android.support.v4.view.ViewPager>
```

Η κλάση της δραστηριότητας για τις πληροφορίες του τμήματος(`DepartmentActivity`), οι υπόλοιπες υλοποιούνται με παρόμοιο τρόπο.

```
public class DepartmentActivity extends BaseActivity {
   private ContentPagerAdapter pagerAdapter;
```

```
    private ViewPager viewPager;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_viewpager);
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);

        // Create the adapter that will return a fragment for each content view
        pagerAdapter = new ContentPagerAdapter(getSupportFragmentManager());

        // Set up the ViewPager, attaching the adapter
        viewPager = (ViewPager) findViewById(R.id.pager);
        viewPager.setAdapter(pagerAdapter);
    }

    private class ContentPagerAdapter extends FragmentPagerAdapter {
        private String[] filenames =
            getResources().getStringArray(R.array.dep_filenames);
        private String[] titles =
            getResources().getStringArray(R.array.dep_titles);

        public ContentPagerAdapter(FragmentManager fm) {
            super(fm);
        }

        @Override
        public int getCount() {
            return 3;
        }

        @Override
        public Fragment getItem(int i) {
            Fragment fragment = new RawWebViewFragment();
            Bundle args = new Bundle();
            args.putString("resource", filenames[i]);
            fragment.setArguments(args);
            return fragment;
        }

        @Override
        public CharSequence getPageTitle(int i) {
            return titles[i];
        }
    }
}
```



Εικόνα 42: Εμφάνιση δραστηριοτήτων πληροφοριών

Συμπεράσματα

Η χρήση της γλώσσας προγραμματισμού Java, μιας εύκολης στη χρήση αλλά ταυτόχρονα πολύ ισχυρής γλώσσας, διευκόλυνε την ανάπτυξη της εφαρμογής. Αυτό σε συνδυασμό με το πολύ καλό περιβάλλον ανάπτυξης το οποίο περιλαμβάνει όλα τα απαραίτητα εργαλεία κάνει την όλη διαδικασία ακόμα πιο προσιτή.

Η ενοποίηση του γραφικού περιβάλλοντος των εφαρμογών για smartphones και tablets στις νεότερες εκδόσεις επιτρέπει την βέλτιστη αξιοποίηση των διαμορφώσεων διαφορετικών συσκευών καθώς στις περισσότερες περιπτώσεις το σύστημα αναλαμβάνει αυτόματα την επιλογή των κατάλληλων ιδιοτήτων.

Παρόλα αυτά η ανάγκη για υποστήριξη παλιότερων εκδόσεων οι οποίες διατηρούν ακόμα ένα μεγάλο μερίδιο της αγοράς δυσχεράνει την ανάπτυξη. Η εισαγωγή νέων χαρακτηριστικών στις καινούργιες εκδόσεις καλύπτεται με τη χρήση των επίσημων βιβλιοθηκών υποστήριξης αλλά όχι σε όλες τις περιπτώσεις. Αλλαγές στις υλοποιήσεις των υπαρχόντων APIs, αν και κάτι απαραίτητο για την βελτίωση του συστήματος, αναγκάζει σε διαφορετικό τρόπο χρήσης τους ανάλογα με την έκδοση.

Η εφαρμογή παρέχει πρόσβαση σε αρκετές από τις βασικότερες υπηρεσίες του τμήματος παρόλα αυτά υπάρχουν αρκετές ακόμα που μπορούν να προστεθούν όπως και να δημιουργηθούν νέες για την περαιτέρω βελτίωση της. Μερικές από αυτές θα μπορούσαν να είναι:

- Προσθήκη προβολής προγράμματος του τρέχοντος εξαμήνου η οποία θα μπορεί να προσαρμόζεται με βάση τα μαθήματα που έχει επιλέξει ο φοιτητής.
- Δυνατότητα αναζήτησης στον κατάλογο της βιβλιοθήκης.
- Χρήση της FTP υπηρεσίας που παρέχεται από το τμήμα για αποστολή και λήψη αρχείων.
- Η λειτουργία των e-mail μπορεί να επεκταθεί προσθέτοντας τις δυνατότητες λήψης των συνημμένων και αποστολής νέου μηνύματος.
- Εμφάνιση ελεύθερων εργαστηρίων με βάση το αιθουσιολόγιο του εξαμήνου.

Βιβλιογραφία

Frank Feinbube, Embedded OS - Android, 17 November 2011

Gilles Printemps, Deep Inside Android…, Zurich - OpenExpo 2008

Reto Meier (2012) Professional Android™ 4 Application Development

http://www.vogella.com/tutorials/android.html

http://en.wikipedia.org/wiki/Android_(operating_system)

http://en.wikipedia.org/wiki/Android_version_history

http://en.wikipedia.org/wiki/Web_scraping

http://socialcompare.com/en/comparison/android-versions-comparison

http://developer.android.com/about/versions/index.html

http://developer.android.com/develop/index.html

https://developers.google.com/maps/documentation/android/

https://developers.google.com/maps/documentation/directions/

https://source.android.com/devices/tech/security/index.html