ΤΕΧΝΟΛΟΓΙΚΟ
ΕΚΠΑΙΔΕΥΤΙΚΟ
ΙΔΡΥΜΑ
ΔΥΤΙΚΗΣ ΕΛΛΑΔΑΣ
ΣΧΟΛΗ ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΟΙΚΟΝΟΜΙΑΣ
ΤΜΗΜΑ ΔΙΟΙΚΗΣΗΣ ΕΠΙΧΕΙΡΗΣΕΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Ανάπτυξη εφαρμογής σε λειτουργικό σύστημα Android για την εύρεση λιστών μουσικής με βάση τις τοπικες καιρικές συνθήκες»

Συγγραφέας:
Λαζάρου Συμεών

Επιβλέπων:
Στάμος Κώστας

Πάτρα, 2019

Περιεχόμενα

Περίληψη

Ο σκοπός της παρούσας πτυχιακής είναι υλοποίηση εφαρμογής σε λειτουργικό σύστημα Android. Επίσης, η ανάδειξη της επικοινωνίας μεταξύ εφαρμογών και εξωτερικών πακέτων δεδομένων. Αναλυτικότερα, πρόκειται για ένα Music Player που εκτός της αναπαραγωγής μουσικής μέσα απο την βιβλιοθήκη του τηλεφώνου, επικοινωνεί με ένα εξωτερικό API και λαμβάνει τις καιρικές συνθήκες που επικρατούν στην περιοχή που βρίσκεται ο χρήστης και του προτείνει λίστες αναπαραγωγής αναλόγως τις συνθήκες. Το πρώτο κεφάλαιο της εργασίας γίνεται μια εισαγωγή στον προγραμματισμό σε Android και αναλύονται βασικά στοιχεία. Στο δέυτερο κεφάλαιο γίνεται μια αναφορά στο περιβάλλον ανάπτυξης της εφαρμογής, το Android Studio. Στην συνέχεια, αναλύονται κάποια δομικά στοιχεία που χρησιμοποιήθηκαν εκτενώς για την υλοποίηση της εφαρμογής. Τέλος στο 4ο και τελευταίο κεφάλαιο γίνεται ανάλυση του κώδικα της εφαρμογής και η παρουσίαση της μέσω εικόνων.

Κεφάλαιο 1

Εισαγωγή στο Προγραμματισμό σε Android

1.1 Λειτουργικό σύστημα Android

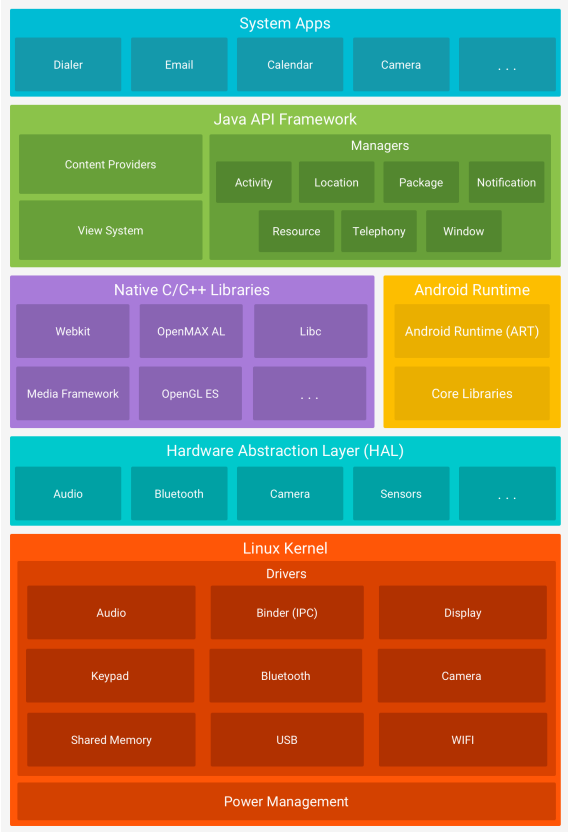
Το Android ειναι ενα λειτουργικό σύστημα για κινητά αναπτυγμένο απο την google βασισμένο πάνω σε μία παραλλαγή του πυρήνα των linux και άλλων ανοιχτών πηγών προγραμμάτων και είναι σχεδιασμένο κυρίως για συσκευές αφής όπως τα smartphones και τα tablets, ενώ τα τελευταία χρόνια έχει επεκταθεί και σε τεχνολογικά προϊόντα όπως η τηλεοράση(Android Tv), τα ρολόγια(smartwatch) και άλλα.

Το android ξεκίνησε αρχικά από μια εταιρία λογισμικού που ονομαζόταν ”Android Inc” με σκοπό την ανάπτυξη του ως ενα λειτουργικό που θα υποστηρίζει ενα μεγάλο φάσμα συσκευών και θα απευθύνεται σε όσους περισσότερους χρήστες γίνεται. Για την υλοποίηση αυτών των στόχων βασίστηκαν σε ευρέως γνωστά εργαλεία και συγκεκριμένα στο πυρήνα του linux και στην γλώσσα προγραμματισμού java.

Το 2005 η Google εξαγόρασε την Android Inc. με απώτερο σκοπό την εισοδό της στην αγορά των κινητών. Απο τότε το λειτουργικό σύστημα έχει υποστεί μεγάλες αλλαγές μέσα από της ανανεώσεις τον εκδόσεων. Αν και το Android λογίζεται ως ενα λογισμικό ανοιχτου κώδικα (γνωστό και ως Android Open Source Project)(AOSP) η κύρια υποστήριξη προέρχεται απο την Google. Απο το 2011 το android έχει το μεγαλύτερο μερίδιο πωλήσεων στην αγόρα τον smartphone, ενω απο το 2013 και σε αύτη τον tablet.

1.2 Αρχιτεκτονική Android

Η αρχιτεκτονική του Android αποτελείται απο μια στοίβα εργαλείων οπου το κάθε επίπεδο καλείται να αντιμετωπίση διαφορετικές ανάγκες, να επιλύσει συγκεκριμένα προβλήματα και να παρέχει υπηρεσίες στο ανώτερο επίπεδο.



Σχήμα 1.1: Στοίβα των εργαλείων του Λειτουργικού Android

Αναλυτικότερα:

Linux Kernel

Πρόκειται για την βάση της πλατφόρμας Android. Χρησιμοποιώντας τον πυρήνα τον linux το Android μπορεί να εκμεταλευτεί τα πλεονεκτήματα της ασφάλειας που παρέχονται και επιτρέπει στους κατασκευαστές να αναπτύξουν οδηγούς υλικού(hardware drivers) για έναν ευρέως γνωστό πυρήνα.

Hardware Abstraction Layer (HAL)

Το επίπεδο αυτό ονομάζεται Επίπεδο Αφαίρεσης Υλικού (Hardware Abstration Layer-HAL) και μαζί με τις βασικές βιβλιοθήκες λογισμικού(πχ. κάμερα) παρέχουν συγκεκριμένες διεπαφές που εκθέτουν της δυνατότητες της συσκευής στο ανώτερο επίπεδο ”Java API Framework”. Όταν το πλαίσιο του API ζητά πρόσβαση στο υλικό της συσκευής, τότε το Android φορτώνει την κατάλληλη βιβλιοθήκη για το συγκεκριμένο κομμάτι της συσκευής.

Android Runtime

Η ART(Android Runtime) δημιουργήθηκε ώστε να τρέχει πολλαπλές εικονικές μηχανές σε χαμηλής μνήμης συσκευές εκτελώντας DEX αρχεία, μια bytecode μορφή ειδίκα σχεδιασμένη για Android οπου είναι βελτιωμένα για ελάχιστα ίχνη μνήμης. Κάποια απο τα βασικά χαρακτηριστικά των ART είναι:

- Πρόωρη και πανω στην ώρα compilation
- Βελτιστοποίηση συλλογής σκουπιδιών
- Καλύτερη υποστήριξη debugging, εμπεριέχοντας λεπτομερής διαγνώσείς εξαιρέσεων και crash αναφορές.

Native C/C++ Libraries

Πολλά δομικά στοιχεία και υπηρεσίες του Android όπως το ART και το HAL, είναι φτιαγμένα σε χαμηλότερες γλώσσες που απαιτούν χαμηλές βιβλιοθήκες γραμμένες σε C και C++. Η πλατφόρμα του Android παρέχει το java Framework Api ώστε να αποκαλύψη την λειτουργικότητα απο αυτές της βιβλιοθήκες στις εφαρμογές.

Java API Framework

Το ολοκληρωμένο σετ χαρακτηριστικών του λειτουργικου Android είναι διαθέσιμο μέσα από APIs τα οποία είναι γραμμένα στην γλώσσα προγραμματισμού java. Αυτά τα APIs σχηματίζουν τα θεμέλια που χρειάζονται για την δημιουργία εφαρμογών και απλοποιούν την επαναχρησιμοποίηση του πυρήνα, των δομικών στοιχείων του συστήματος και των υπηρεσιών, όπου εμπεριέχουν τα εξής:

- Ενα πλούσιο και επεκτάσιμο οπτικό σύστημα που χρησιμοποίεται για την δημιουργία του UI

- Ένας διαχειριστής πόρων που επιτρέπει σε όλες τις εφαρμογές να εμφανίζουν δικές τους ειδοποιήσεις στην στήλη της κατάστασης

- Έναν διαχειριστή δραστηριοτήτων που διαχειρίζεται τον κύκλο ζωής των εφαρμογών και παρέχει μια συνηθισμένη πλοήγηση.

- Παροχοί περιεχομένων που επιτρέπουν στις εφαρμογές την είσοδο σε δεδομένα απο άλλες εφαρμογές, όπως την εφαρμογή των Επαφών, ή να μοιράζονται τα δικά τους δεδομένα.

System Apps

Το Android έρχεται μαζί με ενα σετ απο βασικές εφαρμογές για mail, Sms μηνύματα, ημερολόγιο, Πρόγραμμα περιήγησης στο διαδίκτυο, Επαφές και άλλα. Οι εφαρμογές του συστήματος δεν λειτουργούν μόνο ως εφαρμογές για τους χρήστες αλλά και για να παρέχουν την δυνατότητα πρόσβασης σε αυτές στους προγραμματιστές μέσω των δικών τους εφαρμογών.

1.3 Εφαρμογές στο λειτουργικο Android

1.3.1 Βασικές Αρχές

Οι εφαρμογές android μπορούν να γραφτουν χρησιμοποιώντας τις γλώσσες προγραμματισμού Kotlin,Java και C++. Τα εργαλεία SDK Android συνθέτουν τον κώδικα, μαζί με οποιαδήποτε δεδομένα και αρχεία πόρων σε ένα πακέτο APK, ένα πακέτο Android, το οποίο είναι αρχείο αρχειοθέτησης με κατάληξη .apk. Ένα αρχείο APK περιέχει όλα τα περιεχόμενα μιας εφαρμογής Android και είναι το αρχείο που χρησιμοποιούν οι συσκευές Android για την εγκατάσταση της εφαρμογής.

Κάθε εφαρμογή Android ζει σε δικό της sandbox ασφάλειας, και προστατεύεται από τις ακόλουθες δυνατότητες ασφαλείας Android:

- Το λειτουργικό σύστημα Android είναι ένα σύστημα Linux για πολλούς χρήστες, στο οποίο κάθε εφαρμογή είναι διαφορετικός χρήστης.
- Από προεπιλογή, το σύστημα εκχωρεί σε κάθε εφαρμογή ένα μοναδικό αναγνωριστικό χρήστη Linux (το αναγνωριστικό χρησιμοποιείται μόνο από το σύστημα και είναι άγνωστο στην εφαρμογή). Το σύστημα ορίζει δικαιώματα για όλα τα αρχεία μιας εφαρμογής, έτσι ώστε μόνο τα αναγνωριστικά του χρήστη που έχουν εκχωρηθεί σε αυτήν την εφαρμογή να έχουν πρόσβαση σε αυτά.
- Κάθε διαδικασία έχει τη δική της εικονική μηχανή (VM), οπότε ο κώδικας μιας εφαρμογής εκτελείται μεμονωμένα από άλλες εφαρμογές.
- Από προεπιλογή, κάθε εφαρμογή τρέχει στη δική της διαδικασία Linux. Το σύστημα Android ξεκινά τη διαδικασία όταν χρειάζεται να εκτελεστεί κάποιο από τα στοιχεία της εφαρμογής και στη συνέχεια κλείνει τη διαδικασία όταν δεν είναι πλέον απαραίτητο ή όταν το σύστημα πρέπει να ανακτήσει τη μνήμη για άλλες εφαρμογές.

Το σύστημα Android εφαρμόζει την αρχή του λιγότερου προνομίου. Δηλαδή, κάθε εφαρμογή, από προεπιλογή, έχει πρόσβαση μόνο στα συστατικά που χρειάζεται για να κάνει τη δουλειά της και όχι περισσότερο. Αυτό δημιουργεί ένα πολύ ασφαλές περιβάλλον στο οποίο μια εφαρμογή δεν μπορεί να έχει πρόσβαση σε τμήματα του συστήματος για τα οποία δεν του έχει δοθεί άδεια. Ωστόσο, υπάρχουν τρόποι για μια εφαρμογή να μοιράζεται δεδομένα με άλλες εφαρμογές και για μια εφαρμογή να έχει πρόσβαση σε υπηρεσίες συστήματος:

- Είναι δυνατόν να κανονίσετε δύο εφαρμογές να μοιράζονται το ίδιο αναγνωριστικό χρήστη Linux, οπότε μπορούν να έχουν πρόσβαση στα αρχεία του άλλου. Για τη διατήρηση των πόρων του συστήματος, οι εφαρμογές με το ίδιο αναγνωριστικό χρήστη μπορούν επίσης να κανονίσουν να εκτελούνται στην ίδια διαδικασία Linux και να μοιράζονται το ίδιο VM. Οι εφαρμογές πρέπει επίσης να υπογράφονται με το ίδιο πιστοποιητικό.

- Μια εφαρμογή μπορεί να ζητήσει άδεια πρόσβασης στα δεδομένα της συσκευής, όπως οι επαφές του χρήστη, τα μηνύματα SMS, το αποθηκευτικό χώρο (κάρτα SD), η κάμερα και το Bluetooth. Ο χρήστης πρέπει να χορηγήσει ρητά αυτά τα δικαιώματα.

1.3.2 Στοιχεία εφαρμογής

Τα στοιχεία της εφαρμογής είναι τα βασικά δομικά στοιχεία μιας εφαρμογής Android. Κάθε στοιχείο είναι ένα σημείο εισόδου μέσω του οποίου το σύστημα ή ένας χρήστης μπορεί να εισάγει την εφαρμογή σας. Ορισμένα εξαρτήματα εξαρτώνται από άλλους.

Υπάρχουν τέσσερις διαφορετικοί τύποι στοιχείων της εφαρμογής:

- Δραστηριότητες (Activities)
- Υπηρεσίες (Services)
- Δέκτες εκπομπής (Broadcast receivers)
- Παροχείς περιεχομένου (Content providers)

Κάθε τύπος εξυπηρετεί έναν ξεχωριστό σκοπό και έχει έναν ξεχωριστό κύκλο ζωής ο οποίος καθορίζει τον τρόπο με τον οποίο το στοιχείο δημιουργείται και καταστρέφεται. Οι ακόλουθες ενότητες περιγράφουν τους τέσσερις τύπους στοιχείων της εφαρμογής.

Activities

Μια δραστηριότητα είναι το σημείο εισόδου για την αλληλεπίδραση με τον χρήστη. Αντιπροσωπεύει μια ενιαία οθόνη με μια διεπαφή χρήστη. Για παράδειγμα, μια εφαρμογή ηλεκτρονικού ταχυδρομείου μπορεί να έχει μια δραστηριότητα που εμφανίζει μια λίστα με νέα μηνύματα ηλεκτρονικού ταχυδρομείου, μια άλλη δραστηριότητα για τη σύνταξη ενός μηνύματος ηλεκτρονικού ταχυδρομείου και μια άλλη δραστηριότητα για την ανάγνωση μηνυμάτων ηλεκτρονικού ταχυδρομείου. Αν και οι δραστηριότητες συνεργάζονται για να σχηματίσουν μια συνολική εμπειρία χρήστη στην εφαρμογή email, κάθε μία είναι ανεξάρτητη από τις άλλες. Ως εκ τούτου, μια διαφορετική εφαρμογή μπορεί να ξεκινήσει οποιαδήποτε από αυτές τις δραστηριότητες αν το επιτρέπει η εφαρμογή ηλεκτρονικού ταχυδρομείου. Για παράδειγμα, μια εφαρμογή κάμερας μπορεί να ξεκινήσει τη δραστηριότητα στην εφαρμογή ηλεκτρονικού ταχυδρομείου που συνθέτει νέα μηνύματα ηλεκτρονικού ταχυδρομείου για να επιτρέψει στο χρήστη να μοιραστεί μια φωτογραφία. Μια δραστηριότητα διευκολύνει τις ακόλουθες αλληλεπιδράσεις κλειδιών μεταξύ συστήματος και εφαρμογής:

- Παρακολουθήστε για το τι ενδιαφέρεται επί του παρόντος ο χρήστης (τι εμφανίζεται στην οθόνη) για να διασφαλίσετε ότι το σύστημα εξακολουθεί να εκτελεί τη διαδικασία που φιλοξενεί τη δραστηριότητα.

- Γνωρίζοντας ότι οι προηγουμένως χρησιμοποιούμενες διαδικασίες περιέχουν πράγματα που ο χρήστης μπορεί να επιστρέψει στις (σταματημένες δραστηριότητες) και έτσι να δώσει μεγαλύτερη προτεραιότητα στη διατήρηση αυτών των διαδικασιών.
- Βοηθώντας το χειριστή της εφαρμογής να σκοτώσει τη διαδικασία, ώστε ο χρήστης να μπορέσει να επιστρέψει σε δραστηριότητες με την προηγούμενη κατάσταση που έχει αποκατασταθεί.
- Παρέχοντας έναν τρόπο για τις εφαρμογές να υλοποιούν ροές χρηστών μεταξύ τους και για το σύστημα να συντονίζει αυτές τις ροές. (Το πιο κλασικό παράδειγμα εδώ είναι το μερίδιο.)

Services

Μια υπηρεσία είναι ένα σημείο εισόδου γενικού σκοπού για τη διατήρηση μιας εφαρμογής που εκτελείται στο παρασκήνιο για κάθε είδους λόγο. Πρόκειται για ένα στοιχείο που εκτελείται στο παρασκήνιο για εκτέλεση εκτεταμένων εργασιών ή για εκτέλεση εργασίας για απομακρυσμένες διεργασίες. Μια υπηρεσία δεν παρέχει διεπαφή με τον χρήστη. Για παράδειγμα, μια υπηρεσία μπορεί να αναπαράγει μουσική στο παρασκήνιο ενώ ο χρήστης βρίσκεται σε διαφορετική εφαρμογή ή μπορεί να ανακτήσει δεδομένα μέσω του δικτύου χωρίς να εμποδίζει την αλληλεπίδραση των χρηστών με μια δραστηριότητα. Ένα άλλο στοιχείο, όπως μια δραστηριότητα, μπορεί να ξεκινήσει την υπηρεσία και να την αφήσει να τρέξει ή να συνδεθεί σε αυτήν για να αλληλεπιδράσει με αυτήν.

Υπάρχουν δύο πολύ ξεχωριστές υπηρεσίες σημασιολογίας που λένε στο σύστημα πώς να διαχειριστεί μια εφαρμογή: Οι υπηρεσίες που ξεκίνησαν λένε στο σύστημα να τις κρατήσει σε λειτουργία μέχρι να ολοκληρωθεί η εργασία τους. Αυτό θα μπορούσε να είναι ο συγχρονισμός ορισμένων δεδομένων στο παρασκήνιο ή η αναπαραγωγή μουσικής ακόμα και μετά την έξοδο από την εφαρμογή από τον χρήστη. Η συγχρονισμός δεδομένων στο παρασκήνιο ή η αναπαραγωγή μουσικής αντιπροσωπεύουν επίσης δύο διαφορετικούς τύπους υπηρεσιών που ξεκινούν και οι οποίες τροποποιούν τον τρόπο χειρισμού του συστήματος:

- Η αναπαραγωγή μουσικής είναι κάτι που ο χρήστης γνωρίζει άμεσα, οπότε η εφαρμογή λέει στο σύστημα αυτό, ότι θέλει να είναι νεά στο προσκήνιο, χρησιμοποιώντας μια ειδοποίηση για να το πει στο χρήστη. Σε αυτή την περίπτωση το σύστημα ξέρει ότι θα πρέπει να προσπαθήσει πραγματικά σκληρά για να διατηρήσει τη διαδικασία της υπηρεσίας αυτής, επειδή ο χρήστης θα είναι δυσαρεστημένος εάν απομακρυνθεί.
- Μια τακτική υπηρεσία παρασκηνίου δεν είναι κάτι που ο χρήστης γνωρίζει άμεσα ως τρέχουσα, οπότε το σύστημα έχει περισσότερη ελευθερία στη διαχείριση της διαδικασίας του. Μπορεί να επιτρέψει τη θανάτωσή του (και στη συνέχεια την επανεκκίνηση της υπηρεσίας κάποια στιγμή αργότερα) εάν χρειάζεται RAM για πράγματα που έχουν άμεση ανησυχία για τον χρήστη.

Οι δεσμευμένες υπηρεσίες εκτελούνται επειδή κάποια άλλη εφαρμογή (ή το σύστημα) έχει δηλώσει ότι θέλει να κάνει χρήση της υπηρεσίας. Αυτή είναι βασικά η υπηρεσία που παρέχει ένα API σε μια άλλη διαδικασία. Το σύστημα ξέρει λοιπόν ότι υπάρχει μια εξάρτηση μεταξύ αυτών των διαδικασιών, οπότε αν η διαδικασία Α είναι συνδεδεμένη με μια υπηρεσία στη διαδικασία Β, γνωρίζει ότι πρέπει να κρατήσει τη διαδικασία Β (και την υπηρεσία της) για να τρέξει για το Α. Επιπλέον, αν η διαδικασία Α είναι κάτι που ο χρήστης νοιάζεται,τότε αντιμετωπίζει και τη διαδικασία Β ως κάτι που και ο χρήστης νοιάζεται. Λόγω της ευελιξίας τους, οι υπηρεσίες έχουν αποδειχθεί ότι είναι ένα πολύ χρήσιμο δομικό στοιχείο για κάθε είδους έννοιες συστήματος υψηλότερου επιπέδου. Οι ζωντανές ταπετσαρίες, οι ακροατές ειδοποίησης, οι προφυλακτήρες οθόνης, οι μέθοδοι εισόδου, οι υπηρεσίες προσβασιμότητας και πολλές άλλες βασικές λειτουργίες του συστήματος είναι όλες χτισμένες ως υπηρεσίες που υλοποιούν οι εφαρμογές και το σύστημα συνδέεται όταν πρέπει να εκτελούνται.

Broadcast receivers

Ένας δέκτης εκπομπής είναι ένα στοιχείο το οποίο επιτρέπει στο σύστημα να εκπέμπει συμβάντα στην εφαρμογή εκτός μιας κανονικής ροής του χρήστη, επιτρέποντας στην εφαρμογή να ανταποκρίνεται σε ανακοινώσεις εκπομπής σε όλο το σύστημα. Επειδή οι δέκτες εκπομπής είναι μια άλλη καλά καθορισμένη είσοδος στην εφαρμογή, το σύστημα μπορεί να παραδώσει εκπομπές ακόμα και σε εφαρμογές που δεν εκτελούνται αυτήν τη στιγμή. Έτσι, για παράδειγμα, μια εφαρμογή μπορεί να προγραμματίσει ένα συναγερμό για να δημοσιεύσει μια ειδοποίηση για να πει στο χρήστη για ένα επερχόμενο συμβάν και με την παράδοση αυτού του συναγερμού σε έναν Broadcast Receiver της εφαρμογής, δεν υπάρχει ανάγκη η εφαρμογή να παραμείνει σε λειτουργία μέχρι να σβήσει ο συναγερμός. Πολλές εκπομπές προέρχονται από το σύστημα - για παράδειγμα, μια εκπομπή που αναγγέλλει ότι η οθόνη έχει απενεργοποιηθεί, η μπαταρία είναι χαμηλή ή έχει τραβηχτεί μια φωτογραφία.

Οι εφαρμογές μπορούν επίσης να εκκινήσουν εκπομπές - για παράδειγμα, για να επιτρέψουν σε άλλες εφαρμογές να γνωρίζουν ότι ορισμένα δεδομένα έχουν ληφθεί στη συσκευή και είναι διαθέσιμα για χρήση από αυτούς. Αν και οι δέκτες εκπομπής δεν εμφανίζουν διεπαφή χρήστη (user interface), ενδέχεται να δημιουργήσουν μια ειδοποίηση γραμμής κατάστασης για την ειδοποίηση του χρήστη όταν συμβεί κάποιο γεγονός εκπομπής. Συνηθέστερα, όμως, ένας δέκτης εκπομπής είναι απλώς μια πύλη προς άλλα εξαρτήματα και προορίζεται να κάνει πολύ ελάχιστη ποσότητα εργασίας.

Content Providers

Ένας Content Provider (πάροχος περιεχομένου) διαχειρίζεται ένα κοινό σύνολο δεδομένων της εφαρμογής που μπορείτε να αποθηκεύσετε στο σύστημα αρχείων, σε μια βάση δεδομένων SQLite, στον ιστό ή σε οποιαδήποτε άλλη μόνιμη τοποθεσία αποθήκευσης στην οποία μπορεί να έχει πρόσβαση η εφαρμογή σας. Μέσω των Content Provider, άλλες εφαρμογές μπορούν να ζητήσουν ή να τροποποιήσουν τα δεδομένα, αν το επιτρέπει ο πάροχος περιεχομένου. Για παράδειγμα, το σύστημα Android παρέχει έναν παροχέα περιεχομένου που διαχειρίζεται τα στοιχεία επικοινωνίας του χρήστη. Ως εκ τούτου, οποιαδήποτε εφαρμογή με τα κατάλληλα δικαιώματα μπορεί να ζητήσει από τον πάροχο περιεχομένου, όπως ContactsContract.Data, να διαβάσει και να γράψει πληροφορίες σχετικά με ένα συγκεκριμένο άτομο. Είναι δελεαστικό να σκεφτόμαστε έναν πάροχο περιεχομένου ως μια αφαίρεση σε μια βάση δεδομένων, επειδή υπάρχουν πολλά API και υποστήριξη που ενσωματώνονται σε αυτά για αυτή την κοινή υπόθεση. Ωστόσο, έχουν διαφορετικό βασικό σκοπό από την άποψη του σχεδιασμού του συστήματος. Στο σύστημα, ένας πάροχος περιεχομένου είναι ένα σημείο εισόδου σε μια εφαρμογή για τη δημοσίευση ονομάτων δεδομένων που προσδιορίζονται από ένα σχήμα URI. Έτσι, μια εφαρμογή μπορεί να αποφασίσει πώς θέλει να αντιστοιχίσει τα δεδομένα που περιέχει σε ένα χώρο ονομάτων URI, παραδίδοντας τα URI σε άλλες οντότητες, οι οποίες με τη σειρά τους μπορούν να τις χρησιμοποιήσουν για να έχουν πρόσβαση στα δεδομένα. Υπάρχουν μερικά συγκεκριμένα πράγματα που επιτρέπει στο σύστημα να κάνει τη διαχείριση μιας εφαρμογής:

- Η αντιστοίχιση ενός URI δεν απαιτεί την παραμονή της εφαρμογής, έτσι ώστε τα URI να μπορούν να διατηρηθούν μετά την έξοδο των εφαρμογών τους. Το σύστημα χρειάζεται μόνο να βεβαιωθεί ότι εξακολουθεί να εκτελείται μια εφαρμογή που κατέχει όταν πρέπει να ανακτήσει τα δεδομένα της εφαρμογής από το αντίστοιχο URI.

- Αυτά τα URI παρέχουν επίσης ένα σημαντικό μοντέλο ασφαλείας. Για παράδειγμα, μια εφαρμογή μπορεί να τοποθετήσει το URI για μια εικόνα που έχει στο πρόχειρο, αλλά να αφήσει τον παροχέα περιεχομένου κλειδωμένο, ώστε να μην μπορούν να έχουν πρόσβαση σε άλλες εφαρμογές. Όταν μια δεύτερη εφαρμογή προσπαθεί να έχει πρόσβαση σε αυτό το URI, το σύστημα μπορεί να επιτρέψει σε αυτήν την εφαρμογή να έχει πρόσβαση στα δεδομένα μέσω προσωρινής επιχορήγησης άδειας URI, ώστε να έχει πρόσβαση στα δεδομένα μόνο πίσω από αυτό το URI, αλλά τίποτα άλλο στη δεύτερη εφαρμογή

1.3.3 Αρχείο Manifest

Πριν το σύστημα Android μπορέσει να ξεκινήσει ένα στοιχείο της εφαρμογής, το σύστημα πρέπει να γνωρίζει ότι υπάρχει το στοιχείο διαβάζοντας το αρχείο δήλωσης της εφαρμογής AndroidManifest.xml. Η εφαρμογή σας πρέπει να δηλώσει όλα τα συστατικά της σε αυτό το αρχείο, το οποίο πρέπει να βρίσκεται στη ρίζα του καταλόγου έργου εφαρμογής.

Το Manifest κάνει πολλά πράγματα πέρα από τη δήλωση των στοιχείων της εφαρμογής, όπως τα εξής:

- Προσδιορίζει τα δικαιώματα χρήστη που απαιτεί η εφαρμογή, όπως πρόσβαση στο Internet ή πρόσβαση ανάγνωσης στις επαφές του χρήστη.
- Δηλώνει το ελάχιστο επίπεδο API που απαιτείται από την εφαρμογή, με βάση τα API που χρησιμοποιεί η εφαρμογή.
- Δηλώνει τις λειτουργίες υλικού και λογισμικού που χρησιμοποιούνται ή απαιτούνται από την εφαρμογή, όπως κάμερα, υπηρεσίες Bluetooth ή οθόνη πολλαπλών τόνων.
- Δηλώνει τις βιβλιοθήκες API, οι οποίες πρέπει να συνδέονται με την εφαρμογή (εκτός από τα API πλαισίου Android), όπως η βιβλιοθήκη Google Maps.

1.3.4 Οροι Εφαρμογής

Μια εφαρμογή Android αποτελείται από κάτι παραπάνω από απλό κώδικα - απαιτεί πόρους ξεχωριστούς από τον πηγαίο κώδικα, όπως εικόνες, αρχεία ήχου και οτιδήποτε σχετίζεται με την οπτική παρουσίαση της εφαρμογής. Για παράδειγμα, μπορούμε να ορίσουμε κινούμενα σχέδια, μενού, στυλ, χρώματα και τη διάταξη των διεπαφών χρήστη δραστηριότητας με αρχεία XML. Η χρήση των πόρων καθιστά εύκολη την ενημέρωση διαφόρων χαρακτηριστικών των εφαρμογων χωρίς τροποποίηση του κώδικα. Παρέχοντας σύνολα εναλλακτικών πόρων, επιτυγχάνεται η βελτιστοποιήση των εφαρμογων για μια ποικιλία διαμορφώσεων συσκευών, όπως διαφορετικές γλώσσες και μεγέθη οθόνης.

Για κάθε πόρο που συμπεριλαμβάνετε στο έργο Android, τα εργαλεία δημιουργίας του SDK ορίζουν ένα μοναδικό αναγνωριστικό ακέραιου αριθμού, το οποίο μπορείτε να χρησιμοποιήσετε για να αναφερθείτε στον πόρο από τον κώδικα της εφαρμογής σας ή από άλλους πόρους που ορίζονται στη XML. Για παράδειγμα, εάν κάποια εφαρμογή περιέχει ένα αρχείο εικόνας που ονομάζεται logo.png (αποθηκευμένο στον κατάλογο res / drawable /), τα εργαλεία SDK δημιουργούν ένα αναγνωριστικό πόρου με όνομα R.drawable.logo. Αυτό το αναγνωριστικό αντιστοιχεί σε έναν ακέραιο αριθμό συγκεκριμένης εφαρμογής, τον οποίο μπορείτε να χρησιμοποιήσετε για να αναφερθείτε στην εικόνα και να την τοποθετήσετε στη διεπαφή χρήστη.

Μια από τις πιο σημαντικές πτυχές της παροχής πόρων ξεχωριστά από τον πηγαίο κώδικα, είναι η δυνατότητα παροχής εναλλακτικών πόρων για

διαφορετικές διαμορφώσεις συσκευών. Για παράδειγμα, ορίζοντας τις συμβολοσειρές UI σε XML, μπορούμε να μεταφράσουμε τις συμβολοσειρές σε άλλες γλώσσες και να αποθηκεύσουμε αυτές τις συμβολοσειρές σε ξεχωριστά αρχεία. Στη συνέχεια, το Android εφαρμόζει τις κατάλληλες συμβολοσειρές γλώσσας στο UI με βάση ένα προκριματικό γλώσσας που προσθέτετε στο όνομα του καταλόγου πόρων (όπως res / values-fr / για τιμές γαλλικών συμβολοσειρών) και τη ρύθμιση γλώσσας του χρήστη.

Το Android υποστηρίζει πολλά διαφορετικά προκριματικά για τους εναλλακτικούς πόρους σας. Ο προσδιοριστής είναι μια σύντομη συμβολοσειρά που συμπεριλαμβάνετε στο όνομα των καταλόγων πόρων σας, για να ορίσουμε τη διαμόρφωση της συσκευής για την οποία πρέπει να χρησιμοποιηθούν οι πόροι αυτοί.

Κεφάλαιο 2

Περιβάλλον ανάπτυξης εφαρμογής

2.0.1 Android Studio

Το Android Studio είναι ένα ισχυρό και εξελιγμένο περιβάλλον ανάπτυξης, σχεδιασμένο με ο συγκεκριμένος σκοπός της ανάπτυξης, δοκιμών και συσκευασίας εφαρμογών Android. Μπορεί να είναι μαζί με το Android SDK, ως ένα ενιαίο πακέτο, αλλά στην πραγματικότητα είναι μια συλλογή εργαλείων και συστατικών, πολλά από τα οποία εγκαθίστανται και ενημερώνονται ανεξάρτητα το ένα από το άλλο.

Το Android Studio δεν είναι ο μόνος τρόπος για την ανάπτυξη εφαρμογων Android, υπάρχουν και άλλα IDE, όπως το Eclipse και το NetBeans, και είναι ακόμη δυνατή η ανάπτυξη μιας πλήρους εφαρμογής που δεν χρησιμοποιεί τίποτα περισσότερο από το Σημειωματάριο και τη γραμμή εντολών, αν και αυτή η τελευταία μέθοδος θα ήταν πολύ αργή και δυσκίνητη.

Χτισμένο για ένα σκοπό, το Android Studio έχει προσελκύσει όλο και περισσότερους plugins τρίτων κατασκευαστών που παρέχουν μια μεγάλη ποικιλία πολύτιμων λειτουργιών, οι οποίες δεν είναι διαθέσιμες απευθείας μέσω του IDE. Αυτά περιέχουν πρόσθετα για την επιταχύνση της κατασκευής, την διόρθωση ενος έργου μέσω Wi-Fi και πολλά άλλα.

Τα κύρια προτερήματα του Android σε σχέση με άλλα IDE είναι:

- **Ανάπτυξη UI**: Η πιο σημαντική διαφορά μεταξύ Studio και άλλων IDE είναι ο επεξεργαστής διαμόρφωσης, ο οποίος είναι πολύ ανώτερος από οποιονδήποτε από τους αντιπάλους του, προσφέροντας κείμενο, και προβολές σχεδιαγράμματος, και το σημαντικότερο, εργαλεία διάταξης περιορισμού για κάθε δραστηριότητα ή τεμάχιο, εύκολο στη χρήση. Ο επεξεργαστής διάταξης παρέχει επίσης πολλά εργαλεία που δεν είναι διαθέσιμα αλλού, όπως μια εκτεταμένη λειτουργία προεπισκόπησης για την προβολή των σχεδιαγραμμάτων σε ένα πλήθος συσκευών και απλοί στη χρήση εκδότες θεμάτων και μεταφράσεων.

- **Δομή του έργου**: Παρόλο που η υποκείμενη δομή καταλόγου παραμένει η ίδια, ο τρόπος με τον οποίο το Android Studio οργανώνει κάθε έργο διαφέρει σημαντικά από αυτό προκατόχους. Αντί να χρησιμοποιεί χώρους εργασίας όπως στο Eclipse, το Studio απασχολεί μονάδες που μπορούν πιο εύκολα να επεξεργαστούν μαζί χωρίς να χρειάζεται να αλλάξουν χώρους εργασίας.

- **Ολοκλήρωση κώδικα και refactoring**: Ο τρόπος που το Android Studio με έξυπνο τρόπο συμπληρώνει τον κωδικό ενώ πληκτρολογείτε, το καθιστά απίστευτα βολικό. Προβλέπει τακτικά τι είστε έτοιμοι να πληκτρολογήσετε και συχνά μια ολόκληρη γραμμή κώδικα μπορεί να εισαχθεί χωρίς άλλα από δύο ή τρεις πληκτρολογήσεις. Και ο επαναληπτικός έλεγχος είναι ευκολότερος και πιο εκτεταμένος από εναλλακτικούς IDE, όπως το Eclipse και το NetBeans. Σχεδόν οτιδήποτε μπορεί να είναι μετονομαζόμενη, από τοπικές μεταβλητές σε ολόκληρα πακέτα.

- **Εξομοίωση**: Το στούντιο είναι εξοπλισμένο με ένα ευέλικτο πρόγραμμα επεξεργασίας εικονικών συσκευών, το οποίο επιτρέπει προγραμματιστές να δημιουργήσουν εξομοιωτές συσκευών για να μοντελοποιήσουν οποιοδήποτε αριθμό πραγματικού κόσμου συσκευές. Αυτοί οι εξομοιωτές είναι εξαιρετικά προσαρμόσιμοι, τόσο όσον αφορά τον παράγοντα μορφής και τις παραμέτρους υλικού και οι εικονικές συσκευές μπορούν να μεταφορτωθούν από πολλούς κατασκευαστές. Οι χρήστες άλλων IDE θα είναι ήδη εξοικειωμένοι με τα αρχεία AVD του Android, αν και σίγουρα θα εκτιμήσουν τα χαρακτηριστικά προεπισκόπησης που υπάρχουν στην καρτέλα του Design.

- **Δημιουργία εργαλείων**: Το Android Studio χρησιμοποιεί το σύστημα κατασκευής Gradle, το οποίο εκτελεί οι ίδιες λειτουργίες με το σύστημα Apache Ant που θα έχουν πολλοί προγραμματιστές Java οικείος με. Παρέχει, ωστόσο, πολύ μεγαλύτερη ευελιξία και επιτρέπει custom builds, επιτρέποντας στους προγραμματιστές να δημιουργούν APK που μπορούν να μεταφορτωθούν TestFlight, ή για να παράγετε εύκολα εκδόσεις επίδειξης μιας εφαρμογής. Είναι επίσης το Gradle σύστημα που επιτρέπει την αρθρωτή φύση που συζητήθηκε προηγουμένως. Αντί για καθένα βιβλιοθήκη ή ένα SDK τρίτων κατασκευαστών που συντάσσεται ως αρχείο JAR, το Studio δημιουργεί το καθένα από αυτά αυτά χρησιμοποιώντας το Gradle.

Το Android Studio παρέχει επίσης μια καταπληκτική συσκευή εξοικονόμησης χρόνου με τη μορφή Instant Run. Αυτή η λειτουργία χτίζει έξυπνα μόνο το τμήμα ενός έργου που έχει επεξεργαστεί, αυτό σημαίνει ότι οι προγραμματιστές μπορούν να δοκιμάσουν μικρές αλλαγές στον κώδικα χωρίς να χρειαστεί να περιμένουν την πλήρη κατασκευή τους για κάθε δοκιμή. Αυτή η λειτουργία μπορεί να μειώσει τους χρόνους αναμονής από λεπτά σε σχεδόν μηδέν.

2.0.2 Android SDK

Το Android SDK είναι μια τεράστια συλλογή API, που αποτελείται από τάξεις Java και διεπαφές, οργανωμένη σε σύνθετες αλλά λογικές ιεραρχίες μαζί με άλλες υπηρεσίες κοινής ωφέλειας, όπως το USB οδηγούς και επιταχυντές υλικού. Το SDK και τα στοιχεία του ενημερώνονται πολύ πιο συχνά από το ίδιο το λειτουργικό σύστημα, που οι χρήστες πρέπει να αγνοούν ευχαρίστως. Το SDK ελέγχεται με το SDK Manager, το οποίο είναι προσβάσιμο μέσω της κύριας γραμμής εργαλείων. Υπάρχει επίσης ένας αυτόνομος Διαχειριστής SDK, ο οποίος μπορεί να εκτελεστεί χωρίς το Android Studio.

Εικονικές Συσκευές

Υπάρχουν τόσες πολλές συσκευές Android διαθέσιμες στην αγορά που θα ήταν ένα Αδύνατο να ελέγξουμε λεπτομερώς τις εφαρμογές μας σε πολλές πραγματικές συσκευές. Για το λόγο αυτό ότι το σύστημα μας επιτρέπει να δημιουργούμε συσκευές προσομοίωσης χρησιμοποιώντας τον εικονικό διαχειριστή συσκευών.

Ο AVD Manager μας επιτρέπει να δημιουργούμε από την αρχή τόσο τα προφίλ μορφής όσο και τα προφίλ υλικού και να παρέχει αρκετές έτοιμες εικονικές συσκευές και εικόνες συστημάτων που μπορούμε να κατεβάσουμε από τις ιστοσελίδες διαφόρων κατασκευαστών. Οι εξομοιωτές Android μπορούν να είναι αρκετά αργή, ακόμη και σε πολύ ισχυρές μηχανές, και αυτό είναι αναμενώμενο, καθώς η δημιουργία μιας πλήρως λειτουργικής εικονικής συσκευής είναι ένα εξαιρετικά περίπλοκο έργο.Οι συσκευές πραγματικού κόσμου ανταποκρίνονται φυσικά πολύ πιο γρήγορα από κάθε εξομοιωτή και όταν πρόκειται για κάποια δοκιμη βασικών λειτουργιών, χρησιμοποιώντας τις δικές μας συσκευές θα έχουμε ταχύτερα αποτελέσματα. Αυτή η προσέγγιση είναι αρκετά καλή για τη δοκιμή των θεμελιωδών στοιχείων μιας εφαρμογής, αλλά παρέχει εξαιρετικά λίγα εως καθόλου, στοιχεία σχετικά με το πώς ακριβώς θα δούμε τις εφαρμογές μας σε μεγάλη ποικιλία μεγεθών, σχημάτων και πυκνοτήτων οθόνης που μπορεί να έχουν οι συσκευές Android. Η χρήση πραγματικών συσκευών είναι ένας γρήγορος τρόπος για να δοκιμάσετε τη λογική εφαρμογών, αλλά για την ανάπτύξη εφαρμογών για συγκεκριμένα μοντέλα ή ακόμα για όλων τον μέγεθών και σχημάτων θα απαιτούν αναπόφευκτα τη δημιουργία εικονικών συσκευών.

Κεφάλαιο 3

Δομικά στοιχεία εφαρμογής Climetune

3.1 Activity

Ένα από τα βασικά στοιχεία των εφαρμογών Android είναι το Activity. Αναπαριστά μία οθόνη με στοιχεία ελέγχου μέσω των οποίων ο χρήστης αλληλεπιδρά με την εφαρμογή. Για να επιτευχθεί αυτό, ο προγραμματιστής συνδέει ένα σύνολο από resources από τα οποία αποτελείται η οθόνη (layouts, buttons, εικόνες, κλπ) με ένα αντικείμενο το οποίο κληρονομεί μία Activity κλάση. Αυτό του επιτρέπει να γράψει κώδικα ο οποίος θα ενεργοποιηθεί όταν προκύψουν συγκεκριμένα συμβάντα, το μοντέλο αυτο ονομάζεται event-driven.

3.1.1 Κύκλος ζωής Activity

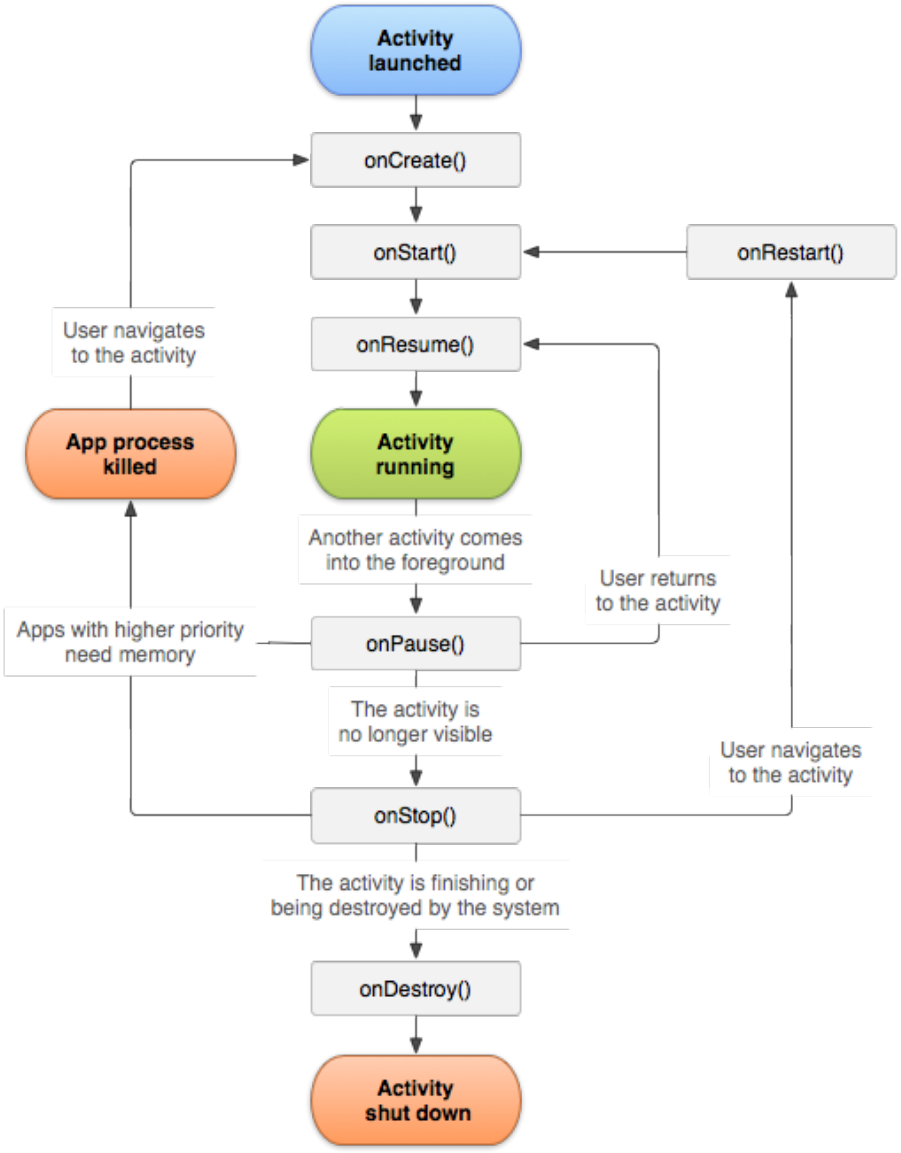
Το σύστημα διαχειρίζεται τα Activities σε μία στοίβα. Το πάνω Activity είναι το μόνο που εμφανίζεται στην οθόνη. Αυτό έχει ως συνέπεια ότι ένα Activity μπορεί να βρίσκεται σε μία από τις 4 καταστάσεις:

- **Running/Active**: Ένα Activity είναι στο προσκήνιο (foreground).
- **Paused**: Ένα Activity δεν είναι στο προσκήνιο αλλά είναι ακόμα ορατό, είτε γιατί το Activity που είναι στο προσκήνιο δεν καταλαμβάνει όλη την οθόνη, είτε γιατί είναι διαφανές.
- **Stopped**: Ένα Activity δεν είναι πρώτο στη στοίβα και το τρέχον Activity το καλύπτει πλήρως.
- **Restarted/Restored**: Ένα Activity το οποίο επανήλθε στην κατάσταση running ενώ βρισκόταν στην κατάσταση paused ή στη stopped.

Τα Activities που είναι σε κατάσταση paused ή stopped έχουν προτεραιότητα για να τερματιστούν στην περίπτωση που το σύστημα δεν έχει αρκετούς πόρους. Αυτό γίνεται είτε αυτόματα, είτε ζητώντας από το Activity να τερματίσει. Το δέυτερο κριτήριο για τον τερματισμό των εφαρμογών αποτελεί η εμπειρία χρήσης, τα Activities σε κατάσταση running έχουν τη ελάχιστες πιθανότητες να τερματιστούν από το σύστημα, αυτά σε κατάσταση paused την

αμέσως περισσότερες ενώ τα stopped της μεγαλύτερες όλων. Ακόμα πάντως και από τα τελευταία, το σύστημα θα διαλέξει πρώτα εκείνα τα Activities τα οποία βρίσκονται σε κατάσταση stopped τον περισσότερο χρόνο. Είναι επόμενο, ότι θα πρέπει ο προγραμματιστής να ελέγχει και να διαχειρίζεται αυτές τις καταστάσεις γράφοντας κώδικα και υλοποιώντας τα κατάλληλα callbacks:

- **OnCreate**: Καλείται όταν το Activity δημιουργείται πρώτη φορά. Εκεί αρχικοποιούνται μεταβλητές που διατηρούνται καθόλη τη διάρκεια ζωής του Activity, ”γεμίζουν” τα layouts και δημιουργούνται τα Views που πρέπει να διαχειριστούν. Είναι η μόνη μέθοδος που υλοποιείται από όλα τα Activities.
- **OnRestart**: Καλείται κατά το πέρασμα του Activity από την κατάσταση stopped στην κατάσταση running. Ακολουθείται από την OnStart.
- **OnStart**: Καλείται όταν το Activity γίνεται ορατό στο χρήστη. Ακολουθείται από την OnResume.
- **OnResume**: Καλείται όταν το Activity είναι σε θέση να αλληλεπιδράσει με το χρήστη.
- **OnPause**: Καλείται όταν ένα νέο Activity ξεκινά να ανεβαίνει πρώτο στη στοίβα (running).
- **OnStop**: Καλείται όταν το Activity δεν είναι πλέον ορατό στο χρήστη.
- **onDestroy**: Καλείται όταν το Activity τερματίζεται, είτε γιατί καλείται η μέθοδος finish() μέσα από τον κώδικά του είτε γιατί το σύστημα το ενημερώνει ότι πρόκειται να τερματιστεί.



Σχήμα 3.1: Κυκλος ζωής Activity

3.1.2 Layout

Το Activity παρέχει μεθόδους για να μετατρέπουμε ένα XML κείμενο που ακολουθεί κατάλληλους κανόνες και περιγράφει ένα User Interface (UI), σε ένα αντικείμενο τύπου View ή ViewGroup το οποίο μπορεί να προβληθεί στην οθόνη ενός συστήματος που τρέχει Android. Το XML αυτό περιγράφει ένα layout και είναι ένας από τους πόρους της εφαρμογής. Το πλεονέκτημα της δήλωσης του UI σε XML είναι ότι μας επιτρέπει να διαχωρίσουμε τα επίπεδα παρουσίασης και λογικής της εφαρμογής. Αυτό με τη σειρά του σημαίνει ότι αλλαγές στο UI της εφαρμογής δεν επηρεάζουν τον κώδικα. Επίσης, είναι εφικτή η δημιουργία layouts σε χρόνο εκτέλεσης, χωρίς τη χρήση XML αλλά προγραμματιστικά, απλά δημιουργώντας νέα Views ή ViewGroups. Η αδυναμία αυτής της λύσης είναι ότι πρέπει να οριστούν όλες οι παράμετροι του View και του ViewGroup καθώς και η συμπεριφορά τους προγραμματιστικά, κάτι που είναι χρονοβόρο και δύσκολο.

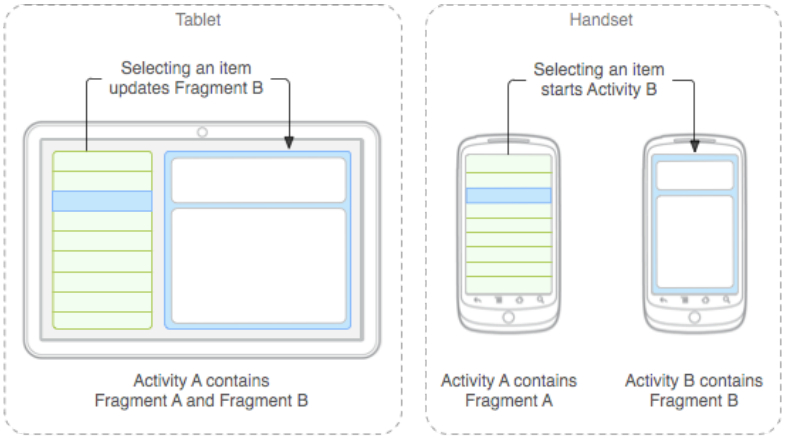
Η γλώσσα XML για το σχεδιασμό του UI είναι άμεσα συνδεδεμένη με τις κλάσεις και τις μεθόδους του View και των υποκλάσεών του, όπως το Button, EditText, κλπ. Σχεδόν πάντα, τα elements αντιστοιχούν σε ονόματα κλάσεων (π.χ. Button) ενώ τα attributes σε ονόματα μεθόδων (π.χ. για το text υπάρχει η μέθοδος setText). Αυτό δημιουργεί ένα άτυπο κανόνα που διευκολύνει τον προγραμματιστή να αναγνωρίζει τη σχέση μεταξύ ενός XML και του αντικειμένου που χειρίζεται στον κώδικα. Πέρα από αυτό, υπάρχουν μερικοί ακόμα βασικοί κανόνες για την δόμησει του XML: όπως ο ορισμός του root element, των attributes και μίας υποκατηγορίας των attributes, τα layout parameters. Τα οποία θα δούμε παρακάτω. Για να περιγράψουμε το UI με την XML θα πρέπει να ορίσουμε ένα root element το οποίο θα πρέπει πάντα να αντιστοιχεί σε ένα αντικείμενο τύπου View ή ViewGroup. Μέσα στο root element μπορούμε να ορίσουμε όσα layouts ή widgets επιθυμούμε. Η συνήθης πρακτική είναι να ορίζεται για root element ένα layout. Τα πιο συνηθισμένα layouts είναι τα:

- **Linear Layout**: Ορίζει ότι η στοίχιση των περιεχομένων του θα είναι σε μία γραμμή ή μία στήλη.
- **Relative Layout**: Ορίζει ότι η στοίχιση των περιεχομένων του θα είναι σχετική, δηλαδή επιτρέπει να ορίσουμε τη θέση του ενός σε σχέση με το άλλο.
- **Web View**: Προσαρμοσμένο στην προβολή ιστοσελίδων (HTML, CSS, κλπ).
- **List View**: Ορίζει ότι η στοίχιση των περιεχομένων του θα είναι σε μία λίστα, το ένα κάτω από το άλλο. Τα στοιχεία της λίστας μπορούν να έχουν μία συμπεριφορά και να αλληλεπιδρούν με το χρήστη με callbacks.
- **Grid View**: Ορίζει ότι η στοίχιση των περιεχομένων του θα είναι σε πλέγμα. Όπως και με το List View τα στοιχεία του πλέγματος μπορούν να έχουν μία συμπεριφορά και να αλληλεπιδρούν με το χρήστη με callbacks.

Όλα τα XML elements έχουν τα δικά τους attributes που σχετίζονται με την κλάση που αναπαριστούν αλλά και κάποια που τα κληρονομούν από τις μητρικές τους κλάσεις. Υπάρχουν και κάποια attributes που σχετίζονται με την τοποθέτηση των στοιχείων που αναπαριστούν τα elements στην οθόνη. Αυτά ξεκινούν με το “layout” και δύο από αυτά θεωρούνται υποχρεωτικά: τα layoutwidth και layoutheight. Αυτά επιτρέπουν τη δήλωση του ύψους και του πλάτους που θα καταλαμβάνουν τα στοιχεία σε κάποια μετρική όπως: dp, sp, pt κ.α. Ωστόσο, αντί να δηλώνουμε επακριβώς το μέγεθος των στοιχείων έχουμε την δυνατότητα να χρησιμοποιήσουμε δύο συγκεκριμένες δηλώσεις: matchparent και wrapcontent. Η πρώτη σημαίνει ότι το στοιχείο θα πρέπει να καταλαμβάνει το πλήρες μέγεθος του μητρικού του στοιχείου ενώ η δεύτερη ότι το μέγεθος του στοιχείου θα πρέπει να είναι τέτοιο ώστε να χωράει πλήρως το περιεχόμενό του (π.χ. ενα κείμενο μέσα σε ένα κουμπί).

3.1.3 Fragment

Παρόλο που τα Views φαίνεται να επαρκούν για να υλοποιηθεί όποια συμπεριφορά θέλουμε σε ένα UI, στην πραγματικότητα υπάρχουν περιπτώσεις που δεν είναι η ιδανική λύση. Αυτό ισχύει κυρίως στην περίπτωση που θέλουμε να φτιάξουμε επαναχρησιμοποιήσιμα UI στοιχεία τα οποία προσαρμόζονται αυτόματα σε διάφορα μεγέθη οθονών, αναλύσεις κλπ.

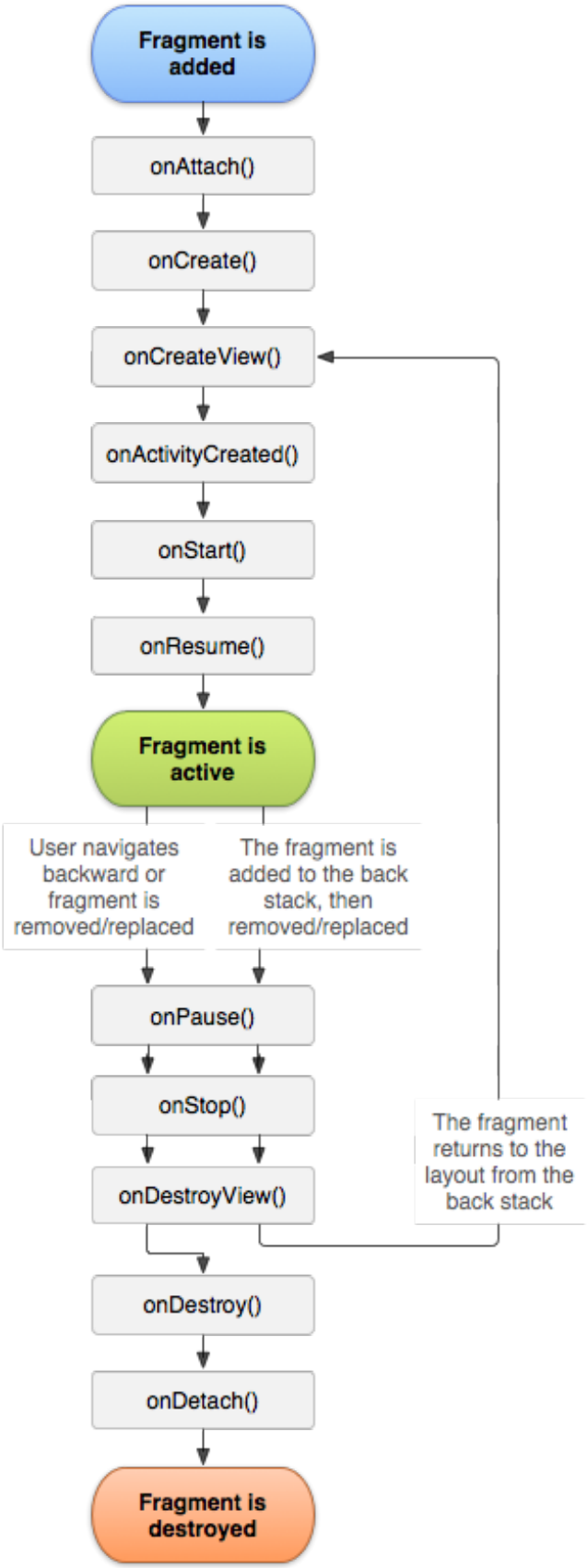


Σχήμα 3.2: Παράδειγμα Fragment

Τα Fragments είναι τμήματα ενός μεγαλύτερου Activity τα οποία ορίζουν ένα τμήμα του UI το οποίο έχει νόημα να διαχωριστεί από το υπόλοιπο π.χ. γιατί έχει διαφορετική συμπεριφορά. Η αλληλεπίδραση με τα Fragments γίνεται μέσω του FragmentManager, ο οποίος μπορεί να γίνει instantiate μέσω της μεθόδου getFragmentManager() που υλοποιούν και το Activity και το Fragment. Τέλος, τα Fragments έχουν το δικό τους Layout, το Fragment Layout.

3.1.4 Κύκλος ζωής Fragment

Ενώ το Fragment έχει φαινομενικά το δικό του κύκλο ζωής μέσα στο Activity, στην πραγματικότητα είναι στενά συνδεδεμένο με αυτό. Αν, για παράδειγμα, το Activity τερματιστεί, το ίδιο θα κάνει και το Fragment που εμπεριέχεται.



Σχήμα 3.3: Κύκλος ζωής Fragment

3.2 Intents

Τα intents είναι μια διαδικασία κλήσης μεθόδων με σκοπό την αρχικοποίηση βασικών στοιχείων μέσα από το event driven μοντέλο. Μία διαδικασία δηλαδή όπως ένα πάτημα ενός κουμπιού στην οθόνη οδηγεί στο instantiation μίας κλάσης -π.χ. τύπου Activity- και την κλήση της αντίστοιχης μεθόδου (την onCreate για την περίπτωση του Activity). Ή πώς είναι δυνατόν ένα αντικείμενο να ζητά από το σύστημα να αρχικοποιήσει ένα δεύτερο αντικείμενο, όπως για παράδειγμα για την εναλλαγή Activities και UIs μέσα στην ίδια ή σε διαφοτερικές εφαρμογές. Ο μηχανισμός αυτός υλοποιείται μέσω των αντικειμένων της κλάσης Intent. Τα αντικείμενα αυτά αντιπροσωπεύουν μηνύματα που αποστέλλονται από μία εφαρμογή στο λειτουργικό το οποίο με τη σειρά του αντιδρά αρχικοποιώντας κάποια σχετική κλάση και προωθώντας το αντικείμενο Intent στο νέο αντικείμενο. Η λογική αυτή επιτρέπει μία ιδιότητα που λέγεται “late runtime binding”, δηλαδή τη σύνδεση του κώδικα διαφορετικών εφαρμογών κατά τη διάρκεια της εκτέλεσης του προγράμματος. Η ιδιότητα αυτή με τη σειρά της επιτρέπει στα στοιχεία του Android να είναι ανεξάρτητα και να συνδέονται με χαλαρές και μη προδιαγεγραμμένες (στον κώδικα) συνδέσεις (loosely coupled components). Τα στοιχεία του Android επομένως προγραμματίζονται με ενσωματωμένη την ιδιότητα να αλληλεπιδρούν χωρίς να ξέρουν αν και ποιο άλλο στοιχείο θέλει να αλληλεπιδράσει με αυτά. Πρακτικά τα Intents χρησιμοποιούνται στις περιπτώσεις κατά τις οποίες ένα στοιχείο μίας εφαρμογής θέλει να ζητήσει από το λειτουργικό σύστημα μία συγκεκριμένη λειτουργικότητα. Το Android θα αναζητήσει τη λειτουργικότητα στο σύνολο των εγκατεστημένων εφαρμογών και θα αρχικοποιήσει το κατάλληλο αντικείμενο στην κατάλληλη εφαρμογή. Το αντικείμενο αυτό μπορεί να είναι ένα Activity, ένα Service ή ένας BroadcastReceiver. Υπάρχουν δύο τρόποι να χρησιμοποιηθούν τα Intents: άμεσα (explicit) και έμμεσα (implicit). Παρακάτω εξηγούνται και οι δύο κατηγορίες.

3.2.1 Άμεσα Intents

Η λογική της χρήσης των άμεσων Intents είναι ότι κάποιο στοιχείο “A” ζητάει από το σύστημα να αρχικοποιήσει ένα δεύτερο στοιχείο “B” το οποίο το “ονοματίζει”, δηλαδή το στοχεύει άμεσα. Για να γίνει αυτό θα πρέπει να δηλωθεί στο Intent το όνομα της εφαρμογής ή του Context αν είναι γνωστό καθώς και της συγκεκριμένης κλάσης του στοιχείου που πρέπει να αρχικοποιηθεί. Αυτό μπορεί να γίνει με τις μεθόδους setClass(Context, Class), setClassName(Context, String), setClassName(String, String) ή με μία εκ των υλοποιήσεων του constructor: Intent(Context packageContext, Class cls) και Intent(String action, Uri uri, Context packageContext, Class cls). Intent i = new Intent();

3.2.2 Έμμεσα Intents

Η έμμεση κλήση αντικειμένων με Intents εδράζει στη λογική των loosely coupled components. Τα στοιχεία των εφαρμογών δε γνωρίζουν τίποτα το

ένα για το άλλο και πολύ συχνά χρειάζονται να ζητήσουν από το σύστημα να τους βρει το κατάλληλο στοιχείο για να εκτελέσει μία επιθυμητή ενέργεια. Ένα παράδειγμα είναι ένα HTTP url που στέλνεται μέσα στο κείμενο ενός μηνύματος. Αν ο παραλήπτης πατήσει το link τότε το σύστημα θα ανοίξει κάποιο browser (φυλλομετρητή) για να εξυπηρετήσει τη συγκεκριμένη ενέργεια, δηλαδή να προβάλλει τη σελίδα που αντιστοιχεί στο url. Η εφαρμογή διαχείρισης/ανάγνωσης των μηνυμάτων δε χρειάζεται να γνωρίζει κάτι για το φυλλομετρητή, ούτε προφανώς να υλοποιεί έναν. Αρκεί να ενημερώσει το λειτουργικό σύστημα ότι χρειάζεται κάποιο στοιχείο να μπορεί να προβάλει το resource που αντιστοιχεί στο URL και το λειτουργικό αναλαμβάνει να βρει εκείνο το στοιχείο και να το αρχικοποιήσει. Επομένως, τα Intents σε αυτή την περίπτωση χρησιμοποιούνται ώστε να δηλωθούν δύο πράγματα: μία ενέργεια και τα δεδομένα πάνω στα οποία θα εκτελεστεί η ενέργεια (action και data). Η ενέργεια μπορεί να είναι η προβολή μίας ιστοσελίδας (ACTION_VIEW) όπως στο προαναφερθέν παράδειγμα, η προβολή μίας τοποθεσίας στο χάρτη ή μίας επαφής. Ακόμα μπορεί να είναι η επεξεργασία (ACTION_EDIT) ενός μηνύματος, η δημιουργία και αποστολή (ACTION_SEND) ενός email και η κλήση (ACTION_DIAL) ενός τηλεφωνικού αριθμού, κλπ. Επομένως η ενέργεια συνοδεύεται κατά κανόνα από τα δεδομένα, όπως το URL, η γεωγραφική τοποθεσία, η επαφή, το μήνυμα κλπ. Η αναπαράσταση των δεδομένων γίνεται με τη χρήση URIs τα οποία αποτελούνται από τα εξής στοιχεία: scheme, host, port, path.

- Scheme: Το scheme δείχνει την κατηγορία του resource. Για παράδειγμα “http” και “ftp” για web resources ενώ για δεδομένα (αρχεία, βάσεις δεδομένων) στο Android υπάρχει το “content”. Για γεωγραφικές τοποθεσίες υπάρχει το “geo”.
- host: Το host μπορεί να είναι το hostname στην περίπτωση ιστοσελίδων (π.χ.google.com) ή μπορεί να είναι το όνομα της εφαρμογής (π.χ. package.climetune).
- Port: Το port χρησιμοποιείται κυρίως για web resources και αναφέρεται στο port στο οποίο δέχεται requests ο server.
- Path: Το κομμάτι που έπεται στο URI και δείχνει πιο συγκεκριμένα ένα resource μέσα στο σύνολο των διαθέσιμων resources.

Υπάρχουν φυσικά και περιπτώσεις όπου τα δεδομένα δεν υπάρχουν εκ των προτέρων και πρέπει να δημιουργηθούν μέσα στο Intent όπως στη δημιουργία και αποστολή (ACTION_SEND) ενός email. Πέρα όμως από τα βασικά τμήματα του Intent, δηλαδή την ενέργεια και τα δεδομένα, τα Intents διατηρούν και μερικά δευτερεύοντα στοιχεία και τα οποία αποσκοπούν στο να κάνουν ακόμα περισσότερο συγκεκριμένα τα κριτήρια αναζήτησης του νέου στοιχείου που θα υλοποιήσει την ενέργεια στα δεδομένα. Αυτά είναι το category, type, component και extras. Τα δύο τελευταία έχουν ήδη εξηγηθεί και πλέον πρέπει να είναι σαφές πώς κάνουν πολύ συγκεκριμένα τα κριτήρια επιλογής στοιχείου για αρχικοποίηση από το σύστημα. Το category δίνει επιπλέον πληροφορία σχετικά με την ενέργεια που είναι να εκτελεστεί. Για παράδειγμα το CATEGORY_LAUNCHER υποδεικνύει ότι το στοιχείο που θα αρχικοποιηθεί θα πρέπει να είναι εκείνο που ξεκινάει με το πάτημα του εικονιδίου της

εφαρμογής. Το CATEGORY_DEFAULT δείχνει ότι το στοιχείο που πρέπει να αρχικοποιηθεί θα πρέπει να έχει δηλωθεί σε αυτή την κατηγορία (DEFAULT). Το MIME type είναι ένα αναγνωριστικό που βοηθά το σύστημα να καταλάβει τον τύπο των δεδομένων και έτσι να περιορίσει ακόμα περισσότερο τα κριτήρια αναζήτησης του κατάλληλου στοιχείου (π.χ. text/plain ή το image/*). Επομένως η πραγματική δομή του Intent είναι η ακόλουθη:

- action
- data
- category
- type
- component
- extras

Μπορούμε πλέον με τη χρήση των κατάλληλων μεθόδων να ορίσουμε τα κριτήρια με βάση τα οποία το Android θα αρχικοποιήσει ένα στοιχείο για να καλύψει τις ανάγκες μας. Παραδείγματα τέτοιων μεθόδων είναι:

- setAction(String)
- setData(Uri data)
- setType(String) ή συνδυαστικά setDataAndType(Uri, String)
  addCategory(String)

  ή μπορούμε να χρησιμοποιήσουμε απ'ευθείας κάποιον constructor του Intent για να ορίσουμε τουλάχιστον το action και τα data:
- Intent(String action, Uri uri)

3.3 Υπηρεσίες Τοποθεσίας

Στο επίπεδο του Android Framework υπάρχουν διάφορες υπηρεσίες τα οποία παρέχονται από το σύστημα (system services), όπως το Filesystem, το GUI, η υπηρεσία κλήσεων και το Task management. Εκεί υπάρχουν και τα location services, ένα σύνολο από υπηρεσίες οι οποίες έχουν σκοπό να παρέχουν στις εφαρμογές πληροφορία σχετικά με τη θέση της συσκευής είτε στιγμιαία είτε τακτικά ρυθμίζοντας ταυτόχρονα θέματα που αφορύν στην κατανάλωση ενέργειας, ενδεχόμενο κόστος υπηρεσίας, ποιότητας πληροφορίας στίγματος και παροχής επιπλέον στοιχείων κλπ. Σε χαμηλότερο επίπεδο, οι υπηρεσίες αυτές διαχειρίζονται οδηγούς (drivers) για την επικοινωνία με αισθητήρες όπως δέκτες GNSS2 (συνήθως GPS), αλλά και πρωτόκολλα τα οποία επιτρέπουν την παροχή πληροφορίας τοποθεσίας, όπως μέσω των υπηρεσιών των WiFi routers ή του δικτύου κινητής τηλεφωνίας. Οι επιλογές αυτές ονομάζονται «πάροχοι» (providers). Υπάρχουν δύο υλοποιήσεις για τη διαχείριση των τοπικών δεδομένων μέσω του Android. Η πρώτη είναι μέσω του κοινού Android API και η δεύτερη μέσω του Google Location Services API, το οποίο είναι κομμάτι των Google Play Services.

3.3.1 Πακέτο Android.location

Το Android SDK δίνει στις εφαρμογές πρόσβαση στις υπηρεσίες τοποθεσίας που υποστηρίζονται από τη συσκευή μέσω του πακέτου android.location. Η λογική είναι ότι θα πρέπει να δημιουργηθεί ένα αντικείμενο τύπου Location Manager το οποίο δίνει μεθόδους για πρόσβαση σε χωρικά δεδομένα που παρέχει η συσκευή.

Εν γένει τα system services δεν τα αρχικοποιούμε με κάποιο constructor αλλά ζητάμε από το σύστημα να το κάνει μέσω της μεθόδου: getSystemService (String name) της κλάσης Context. Αυτή δέχεται το όνομα της υπηρεσίας που επιθυμεί ο προγραμματιστής ως είσοδο (String name) και επιστρέφει ένα γενικό αντικείμενο (Object) το οποίο επιτρέπει στον προγραμματιστή να το διαχειριστεί το service. Το όνομα του system service μπορεί κανείς να το βρει ως σταθερά στο Context. Π.χ. WINDOW-SERVICE, ACTIVITY-SERVICE, LOCATION-SERVICE, κλπ. Ακριβώς επειδή η getSystemService είναι κοινή για όλα τα system services ο προγραμματιστής οφείλει να κάνει το επιστρεφόμενο Object typecast σε μία σχετική κλάση. Στην περίπτωση των υπηρεσιών τοποθεσίας, η κλάση αυτή είναι η LocationManager. Μία από τις κεντρικές λειτουργικότητες του LocationManager είναι ο έλεγχος και η διαχείριση των παρόχων χωρικών δεδομένων. Έτσι, ο προγραμματιστής μπορεί να πάρει μία λίστα με όλους τους γνωστούς πάροχους με τη μέθοδο List<String> getAllProviders() ή να ζητήσει το πάροχο ο οποίος ικανοποιεί με τον καλύτερο δυνατό τρόπο ένα σύνολο από κριτήρια με τη μέθοδο: String getBest Provider(Criteria criteria, boolean enabledOnly). Τα κριτήρια ορίζονται μέσω ενός αντικειμένου Criteria το οποίο είναι ένα Bundle όπου ορίζουμε τα παρακάτω στοιχεία:

- Ακρίβεια μέτρησης τοποθεσίας (Accuracy)
- Ακρίβεια μέτρησης κατεύθυνσης (BearingAccuracy)
- Ακρίβεια μέτρησης γεωγραφικού μήκους και πλάτους (HorizontalAccuracy)
- Ακρίβεια μέτρησης ταχύτητας (SpeedAccuracy)
- Ακρίβεια μέτρησης υψόμετρου (VerticalAccuracy)
- Απαραίτητη μέτρηση υψόμετρου (AltitudeRequired)
- Αν επιτρέπεται η χρήση πάροχων με κόστος (CostAllowed)
- Ανάγκες σε ενέργεια (PowerRequirement)
- Απαραίτητη μέτρηση κατεύθυνσης (BearingRequired)
- Απαραίτητη μέτρηση ταχύτητας (SpeedRequired)

Για κάθε ένα από αυτά υπάρχουν accessors και mutators και μπορούν να πάρουν μία από τις ακόλουθες τιμές οι οποίες είναι static στην κλάση Criteria:

- ACCURACY_COARSE

- ACCURACY_FINE
- ACCURACY_HIGH
- ACCURACY_LOW
- ACCURACY_MEDIUM
- NO_REQUIREMENT
- POWER_HIGH
- POWER_LOW
- POWER_MEDIUM

Τα αποτελέσματα των δύο μεθόδων αυτών μπορούν να χρησιμοποιηθούν για να αντλήσουμε την τελευταία γνωστή θέση της συσκευής με τη μέθοδο: Location getLastKnownLocation(String provider). Η μέθοδος αυτή επιστρέφει ένα αντικείμενο τύπου Location το οποίο περιέχει όλα τα χωρικά δεδομένα τα οποία υποστηρίζει ο επιλεγμένος provider.

Το όνομα της μεθόδου getLastKnownLocation δείχνει ότι οι providers δε λειτουργούν διαρκώς αλλά αντλούν χωρικά δεδομένα διακοπτόμενα. Ο λόγος που συμβαίνει αυτό είναι γιατί σε διαφορετική περίπτωση θα κατανάλωναν μεγάλα ποσά ενέργειες όπως επίσης και γιατί δεν είναι πάντα σε θέση να υπολογίσουν τα χωρικά δεδομένα (π.χ. ασθενές σήμα GPS).

Η δήλωση του Listener και ο ορισμός των «νέων» δεδομένων γίνεται μέσα από τη μέθοδο του LocationManager: requestLocationUpdates. Η μέθοδος αυτή ζητάει ως ορίσματα τον provider, το Listener και δύο ακόμα μεταβλητές: μία για τον ελάχιστο χρόνο (σε milliseconds) και μία για το ελάχιστο διάστημα (σε μέτρα) μέσα στα οποία τα δεδομένα θα έχουν αλλάξει ουσιαστικά. Οι τελευταίες δύο συνθήκες ορίζουν ανεξάρτητα (αρκεί να ισχύει η μία ή η άλλη) κάθε πότε ο provider θα υπολογίζει τα νέα δεδομένα, σώζοντας επομένως ενέργεια. Ο Listener ο οποίος υλοποιεί τον event handler είναι τύπου LocationListener. Ένα αντικείμενο που κληρονομεί ένα τέτοιο Interface, πρέπει να υλοποιεί 4 μεθόδους:

- onLocationChanged(Location location), που καλείται όταν υπάρχει νέο location από το provider
- abstract void onProviderDisabled(String provider), που καλείται όταν ο provider απενεργοποιείται από το χρήστη.
- abstract void onProviderEnabled(String provider), που καλείται όταν ο provider ενεργοποιείται από το χρήστη
- abstract void onStatusChanged(String provider, int status, Bundle extras), που καλείται όταν αλλάζει ο provider δεν μπορεί να φέρει το location (π.χ. δεν έχει GPS σήμα) ή όταν γίνεται διαθέσιμος μετά από μία περίοδο κατά την οποία ήταν μη διαθέσιμος

3.4 Services

3.4.1 Διεργασίες

Η εκτέλεση κάθε Android εφαρμογής ανατίθεται στη δική της διεργασία και όλα τα συστατικά των εφαρμογών τρέχουν επάνω σε αυτή τη διεργασία σε ένα μοναδικό “main” thread. Είναι εφικτό να δηλωθεί από τον προγραμματιστή ότι συγκεκριμένα συστατικά της εφαρμογής θα εκτελεστούν σε διαφορετικές διεργασίες και έτσι να υλοποιηθεί ένα πολυνηματικό μοντέλο. Για να επιτευχθεί αυτό, αρκεί να χρησιμοποιηθεί η δήλωση “android:process” μέσα στο manifest με ένα όνομα που επιθυμεί ο προγραμματιστής. Μέσω αυτής της διαδικασίας, είναι δυνατόν να διαμοιράζονται τα συστατικά διαφορετικών εφαρμογών κοινές διεργασίες, με σκοπό την από κοινού πρόσβαση σε πόρους.

Κυκλος ζωης διεργασιών

Όπως έχει επανειλημμένως ειπωθεί, το λειτουργικό Android σταματάει την εκτέλεση των παλαιότερων διεργασιών προκειμένου να αποδώσει πόρους (κυρίως μνήμη) σε νέες ή πιο σημαντικές διεργασίες. Το τελευταίο υποδεικνύει ότι υπάρχει μία ιεράρχηση των διεργασιών σε σχέση με τη σημαντικότητά τους. Πράγματι, υπάρχουν 4 κατηγορίες διεργασιών οι οποίες διαφοροποιούνται με βάση την πιθανότητα το λειτουργικό να τις τερματίσει, αν χρειαστεί. Η ιεράρχηση αυτή ανάλογα με την κατηγορία είναι η ακόλουθη (με αντίστροφη ταξινόμηση με βάση τη σημαντικότητα):

- **Foreground process**: Πρόκειται για διεργασίες οι οποίες είναι απαραίτητες για τις λειτουργίες που επιτελεί ο χρήστης, τη στιγμή που τις επιτελεί. Επομένως, αυτές οι διεργασίες είναι λίγες, καθώς μπορεί γρήγορα να μεταπέσουν κατηγορία κατά τη διάρκεια της χρήσης τους από το χρήστη. Υπό τις απαραίτητες συνθήκες μπορούν να είναι διεργασίες που εξυπηρετούν Activities, Services ή BroadcastReceivers. Αν το σύστημα φτάσει στο σημείο να τερματίζει foreground processes, αυτό σημαίνει ότι έχει φτάσει σε τόσο κρίσιμο σημείο από άποψη resources, που δεν μπορεί να διατηρήσει το UI responsive.

- **Visible process**: Πρόκειται για διεργασίες τις οποίες ο χρήστης γνωρίζει, ακόμα και αν δεν τις χρησιμοποιεί τη συγκεκριμένη στιγμή, και επομένως, ο τερματισμός τους θα έχει αρνητικές συνέπειες στην εμπειρία χρήσης. Υπό συνθήκες μπορούν να είναι είτε διεργασίες για Activities είτε για Services.

- **Service process**: Πρόκειται για διεργασίες που εξυπηρετούν Services τα οποία έχουν αρχικοποιηθεί αλλά δεν κάνουν κάτι απαραίτητα, παρά μόνο περιμένουν να τους ζητηθεί να κάνουν μία δουλειά, όπως να κατεβάσουν ένα αρχείο από το Internet. Οι διεργασίες αυτές δεν είναι ορατές από το χρήστη, ωστόσο είναι σημαντικές για αυτόν, και επομένως, το σύστημα θα προσπαθεί πάντα να διατηρήσει αυτές τις διεργασίες μέχρι οι πόροι να μην είναι αρκετοί για τα foreground and visible

processes. Ακόμα, Services τα οποία εκτελούνται για πολλή ώρα, το σύστημα τα υποβιβάζει στις επόμενες κατηγορίες. Αυτό επιτρέπει να αποφευχθούν περιπτώσεις όπου η μακροχρόνια εκτέλεση services οδηγεί σε κλιμάκωση προβλημάτων, όπως διαρροή μνήμης κλπ.

- **Cached process**: Πρόκειται για διεργασίες οι οποίες δεν είναι απαραίτητες, πιθανόν να εξυπηρετούν συστατικά τα οποία έχουν ξεχαστεί από το χρήστη, επομένως ο τερματισμός τους δεν θα επηρεάσει την εμπειρία χρήσης. Συνήθως πρόκειται για διεργασίες που εξυπηρετούν Activities στο τέλος της Activity stack. Για τα Activities αυτά διατηρείται μία εσωτερική ιεράρχηση που βασίζεται στο LRU (Least Recently Used) πρωτόκολλο, επομένως έχουν προτεραιότητα να τερματιστούν διεργασίες που έχουν χρησιμοποιηθεί λιγότερο μέσα σε ένα χρονικό διάστημα. Σε αυτή την κατηγορία ανήκουν και τα empty processes, τα οποία είναι διεργασίες οι οποίες έχουν αρχικοποιηθεί (καταναλώνουν πόρους), αλλά δεν κάνουν κάτι αν δε δημιουργηθούν οι κατάλληλες συνθήκες. Αυτά χρησιμοποιούνται σχεδόν αποκλειστικά για τη μείωση του χρόνου αρχικοποίησης των εφαρμογών που εξυπηρετούν.

3.4.2 Νήματα

Το μονο-νηματικό μοντέλο του Android αφήνει μόνο ένα “main” thread να τρέχει την εφαρμογή, όπως ειπώθηκε. Επομένως, το κεντρικό αυτό thread αναλαμβάνει να διαχειριστεί και το UI και επειδή αυτή είναι η πιο σημαντική λειτουργία του (προστασία της εμπειρίας χρήσης), αποκαλείται και “UI thread”. Επομένως, λειτουργίες οι οποίες μπορεί να απαιτούν πολύ χρόνο να εκτελεστούν (συνήθως IOs) καθυστερούν το UI thread.

Εργασίες που κοστίζουν πολύ χρονικά, π.χ. είναι απαιτητικές όσον αφορά τους υπολογισμούς ή υλοποιούν πολλές λειτουργίες I/O, δεν πρέπει να τοποθετούνται στο UI, αλλά σε δευτερεύοντα νήματα (worker threads), για τα οποία πρέπει να φροντίσει ο προγραμματιστής.

3.4.3 Υπηρεσίες

Οι υπηρεσίες είναι συστατικά εφαρμογών στο Android τα οποία εκτελούν χρονοβόρες λειτουργίες και δεν παρέχουν UI. Διαχωρίζοντας τις υπηρεσίες που υλοποιούν οι προγραμματιστές (custom) με αυτές του συστήματος,υπάρχουν 2 βασικοί τύποι custom υπηρεσιών:

- Started:
  ✓Ένα συστατικό εφαρμογής (component, π.χ. Activity) ξεκινάει την υπηρεσία καλώντας τη startService( ).
  ✓Τρέχει ακόμα και αν η αρχική εφαρμογή έχει καταστραφεί.
  ✓Πρέπει να τερματιστεί μόνο του.

- Bound:
  ✓Ένα συστατικό εφαρμογής συνδέεται στην υπηρεσία καλώντας τη bindService( ).

✓Παρέχει ένα interface το οποίο επιτρέπει το component να αλληλεπιδράσει με την υπηρεσία.
✓Επιτρέπει την επικοινωνία μεταξύ διεργασιών μέσω IPC (inter-process communication).
✓Η υπηρεσία δεν καταστρέφεται όταν τα συστατικά αποσυνδεθούν από αυτή.

3.5 Json

Στον Υπολογιστή, το JavaScript Object Notation ή το JSON είναι μια μορφή αρχείου ανοιχτού προτύπου που χρησιμοποιεί κείμενο αναγνώσιμο από άνθρωπο για τη μετάδοση αντικειμένων δεδομένων που αποτελούνται από ζεύγη χαρακτηριστικών-τιμών και πίνακες τύπους δεδομένων (ή οποιαδήποτε άλλη σειριοποιήσιμη τιμή). Πρόκειται για μια πολύ κοινή μορφή δεδομένων που χρησιμοποιείται για την ασύγχρονη επικοινωνία προγράμματος περιήγησης-διακομιστή, συμπεριλαμβανομένης της αντικατάστασης της XML σε ορισμένα συστήματα τύπου AJAX.

Το JSON είναι μια μορφή δεδομένων ανεξάρτητη από τη γλώσσα. Προέρχεται από τη JavaScript, αλλά από το 2017 πολλές γλώσσες προγραμματισμού περιλαμβάνουν κώδικα για τη δημιουργία και την ανάλυση δεδομένων JSON. Ο επίσημος τύπος μέσου διαδικτύου για το JSON είναι η εφαρμογή / json. Τα ονόματα αρχείων JSON χρησιμοποιούν την επέκταση .json.

Κεφάλαιο 4

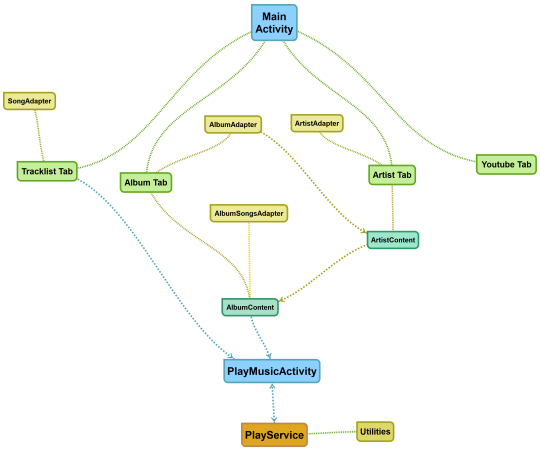
Εφαρμογή ClimeTune

4.1 Εισαγωγή

Το climetune είναι ενα Music Player με την ιδιαιτερότητα της αύτοματης εύρεσης λιστών αναπαραγωγής αναλόγος τον καιρικών συνθηκών που επικρατούν στην περιοχή που βρίσκεται ο χρήστης. Η λίστες παρέχονται και αναπαράγωνται από το youtube όπου είναι ενσωματωμένο στην εφαρμογή. Η έυρεση του καιρού επιτυγχάνετε μεσα απο την επικοινωνία της εφαρμογής με ενα weather API που επιστρέφει στην εφαρμογή ενα json αρχείο με τις καιρικές συνθήκες που αναλογούν στην τοποθεσία του χρήστη. Σκοπός της εφαρμογής είναι η ανάδειξη της επικοινωνίας μεταξύ αυτής με εξωτερικά Apis και πακέτα δεδομένων.

4.2 Αρχιτεκτονική Εφαρμογής

Η εφαρμογή αποτελείται από μια κύρια δραστηριότητα (Main Activity) που διαχειρίζεται 4 Fragments και η απεικόνιση των τριων απο αυτά γίνεται με την βοήθεια τριών κλάσεων τύπου adapter. Στο fragment των αλμπουμς οτάν ο χρήστης επιλέξει κάποιο άλμπουμ, τότε 1 νέο activity(AlbumContent) μαζί με 1 adapter δημιουργεί και απεικονίζει την λίστα των τραγούδιων που συγκαταλέγονται στο άλμπουμ που επίλεξε ο χρήστης. Αντίστοιχα στο fragment των καλλιτεχνών, όταν ο χρήστης επιλέξει κάποιον καλλιτέχνη μεταφέρεται σε ένα καινούργιο activity (ArtistContent) οπου με την βοήθεια ενός adapter απεικονίζονται τα άλμπουμ του καλλιτέχνη στην οθόνη και μετέπειτα η επιλογή του αλμπουμ καταλήγει στο activity AlbumContent. Όταν ο χρήστης επιλέξει κάποιο κομμάτι για αναπαραγωγή τότε ανοίγει ένα νέο activity (PlayMusicActivity) οπου ο χρήστης μπορεί να διαχειριστή το τραγούδι(παυση, αναπαραγωγή, επόμενο, κτλ).Τέλος, για την αναπαραγωγή μουσικής το PlaymusicActivity συνδέεται με ένα service το οποίο αναλαμβάνει και ελέγχει το κύκλο ζωής τον τραγουδιών.



Σχήμα 4.1: Διαγραμμα ροής

4.3 Ανάλυση περιεχομένων εφαρμογής

4.3.1 Main Activity

Πρόκειται για την κύρια δραστηριότητα της εφαρμογής, η οποία δημιουργεί και διαχειρίζεται ένα πλαίσιο με 4 καρτέλες με την βοήθεια της κλάσης SectionPagerAdapter.

- Η 1η καρτέλα είναι αύτη που περιέχει ολα τα αρχεία μουσικής και τα εμφανίζει σε μια λίστα
- Η 2η καρτέλα περιέχει όλα τα αλμπουμ των καλλιτεχνών
- Η 3η καρτέλα περιέχει ολους τους καλλιτέχνες
- Η 4η και τελευταία καρτέλα περιέχει ενα ενσωματομένο player του youtube.

```
public class SectionsPagerAdapter extends FragmentPagerAdapter {

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

        @Override
        public Fragment getItem(int position) {
            switch (position) {
                case 0:
                    TracklistTab tracklistTab = new TracklistTab();
                    return tracklistTab;
                case 1:
                    AlbumTab albumTab = new AlbumTab();
                    return albumTab;
                case 2:
                    ArtistTab artistTab = new ArtistTab();
```

```
                return artistTab;
            case 3:
                YouTubeFragment youTubeFragment =
                new YouTubeFragment();
                return youTubeFragment;
            default:
                return null;
        }
    }
```

Ο παραπάνω κώδικας δημιουργεί τις καρτέλες και επιστρέφει αυτή που αντιστοιχεί στη κάθε θέση.

Main_Activity.xml

Πρόκειται για το αρχείο που αναλαμβάνει την δημιουργία του layout της κύριας δραστηριότητας, το οποίο αποτελείται απο ενα toolbar που περιέχει το όνομα της εφαρμογής, το tab layout με της 4 καρτέλες της εφαρμογής, αλλά και ένα footer που εμφανίζει το τραγούδι που παίζει, μαζι με ένα κουμπί pause/play.

Σχήμα 4.2: Το Main Activity στο android studio

4.3.2 Καρτέλα Τραγουδιών

Αρχείο Tracklist Tab

Το αρχείο Tracklist Tab είναι ένα fragment που αποτελεί την πρώτη καρτέλα του Main Activity. Εδώ δημιουργείται μια λίστα με ολά τα αρχεία ήχου που περιέχονται στο android phone. Για την καταχώρησει των αρχείων στη λίστα χρησιμοποιείται μια μέθοδο με το όνομα getSongs().

```
public void getSongs() {
        musicResolver = getActivity().getContentResolver();
        Uri musicUri = MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;
        Cursor musicCursor =
        musicResolver.query(musicUri, null ,null, null, null);
        songArrayList = new ArrayList<Song>();
```

Στον παραπάνω κώδικα, δημιουργείται ενα αντικείμενο της κλάσης content Resolver, το οποιο το ονομάζεται musicResolver και επιτρέπει στην εφαρμογή την πρόσβαση στα περιεχόμενα του μοντέλου (κινητου). Έπειτα δημιουργείται το αντικειμενο musicUri της κλάσης Uri στο οποίο υποδεικνύειται να διαβάσει τα αρχεία ήχου που βρίσκονται στην συσκευη. Με την βοήθεια ενος κέρσορα (musicCursor) η εφαρμογή μπορεί να διαβάσει τα αρχεία ήχου. Τέλος δημιουργείται ενα ArrayList που περιέχει αντικείμενα της κλάσης song.

```
if (musicCursor != null && musicCursor.moveToFirst()) {

    int IdColumn =
    musicCursor.getColumnIndex(MediaStore.Audio.Media._ID);
    int TitleColumn =
    musicCursor.getColumnIndex(MediaStore.Audio.Media.TITLE);
    int ArtistColumn =
    musicCursor.getColumnIndex(MediaStore.Audio.Media.ARTIST);
    int AlbumColumn =
    musicCursor.getColumnIndex(MediaStore.Audio.Media.ALBUM);
    int DurationColumn=
    musicCursor.getColumnIndex(MediaStore.Audio.Media.DURATION);
    int AlbumIdColumn =
    musicCursor.getColumnIndex(MediaStore.Audio.Media.ALBUM_ID);
```

Ο παραπάνω κώδικας είναι η συνέχεια της μεθόδου getSongs() και δείχνει την διαδικάσια με την οποία ο κερσορας διαβάζει την βάση δεδομένων της συσκευής και περνάει τις τιμές του κάθε πεδίου σε μια μεταβλητή η οποία χρησιμοποιείται στον παρακάτω κωδικα.

```
do {
        Long thisId = musicCursor.getLong(IdColumn);
        int thisAlbumId = musicCursor.getInt(AlbumIdColumn);
        String thisTitle = musicCursor.getString(TitleColumn);
        String thisArtist = musicCursor.getString(ArtistColumn);
        String thisAlbum = musicCursor.getString(AlbumColumn);
        String RawDuration = musicCursor.getString(DurationColumn);

        int Duration = (Integer.parseInt(RawDuration) / 1000);
        int mins = Duration / 60;
        Duration = Duration % 60;
        String FinalDuration = String.format("%02d:%02d", mins, Duration);
        if (Duration > 0.1) {

            songArrayList.add
            (new Song(thisId, thisTitle, thisArtist,thisAlbum,
            FinalDuration, thisAlbumId));
        }
    }
    while (musicCursor.moveToNext());
}musicCursor.close();
```

Όπως φαίνεται στον κώδικα καταχωρούνται οι τιμές σε νεές μεταβλητές οι οποίες αντιστοιχουν με αυτές του αντικειμένου song. Τέλος πρίν γίνει η πρόσθεση του αρχείου στην λίστα γίνεται έλεγχος αμα είναι μικρότερο απο 1 δευτερόλεπτο με σκοπό την αποφυγή των default ringtones της συσκευής.

Σε περίπτωση επιλογής κάποιου τραγουδιού απο τον χρήστη τότε ενεργοποίεται ένα intent και στέλνει την λίστα με ολα τα τραγούδια αλλα και την θέση του επιλεγμένου τραγουδιου στο playMusicActivity, όπως φαίνεται στον παρακατω κωδικα.

```
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
    public void onItemClick(AdapterView<?> parent, View view,
    int position, long id) {

    Intent startmusicactivity =
    new Intent(getContext(),PlayMusicActiviy.class);
    startmusicactivity.putExtra("SongList",songArrayList);
    startmusicactivity.putExtra("pos",position);
    startActivity(startmusicactivity);

            }

        });
```

Για την απεικόνιση των τραγουδιών χρησιμοποιείται η κλάση songAdapter που είναι επεκταση της κλάσης BaseAdapter, η οποία αναλύεται παρακάτω.

```
final SongAdapter songAdt = new SongAdapter(songArrayList, getActivity());
```

SongAdapter

Με την βοήθεια της κλάσης songAdapter γίνεται η σύνδεση μεταξύ του αρχείου song.xml με το fragment TracklistTab και έτσι ορίζεται η εμφάνιση για το κάθε τραγούδι στην λίστα.

```
    public View getView(int position, View convertView, ViewGroup parent) {
        LinearLayout songLay = (LinearLayout)songinf.inflate
                (R.layout.song, parent, false);

        //pass title and artist views
        TextView songView =
        (TextView)songLay.findViewById(R.id.song_title);
        TextView artistView =
        (TextView)songLay.findViewById(R.id.song_artist);
        TextView durationView =
        (TextView)songLay.findViewById(R.id.song_duration);

        //get song using position
        Song currSong = songs.get(position);

        //get title and artist strings
        songView.setText(currSong.getTitle());
        artistView.setText(currSong.getArtist());
        durationView.setText(currSong.getDuration());
```

Στον παραπάνω κώδικα δημιουργούνται 3 μεταβλητές τυπου TextView και έχουν ως όρισμα τα αντίστοιχα TextView που βρίσκονται στο αρχείο song.xml. Με την βοήθεια τον setter θέτουμε τα χαρακτηριστικά του κάθε τραγουδιού.

Song

Η κλάση song επιτρέπει την δημιουργία αντικειμένων αυτου του τύπου. Παρακάτω είναι ο κώδικας κατασκευής(constructor) και εμφανίζονται οι παράμετροι που χρειάζονται για την δημιουργία ενος αντικειμένου song. Το αρχείο περιέχει getters and setters για κάθε παράμετρο του αντικειμένου.

```
public Song(long id, String title, String artist, String album,
String duration, int albumId) {
    this.id = id;
    this.title = title;
    this.artist = artist;
    this.album = album;
    this.duration = duration;
    this.albumId = albumId;
}
```

song.xml

Σε αυτό το αρχείο ορίζουμε την απεικόνιση του song, το οποίο περιέχει 2 Linear layouts, το αρχικό και ένα εμφωλευμένο και μέσα σε αυτά 3 TextViews :

- Το `song_title` που είναι το τίτλος του τραγουδιου που βρίσκεται μέσα στην αρχικη γραμμική διάταξη(Linear Layout).
- Το `song_artist` για την απεικόνιση του καλλιτέχνη οπου βρίσκεται μέσα στην εμφωλευμένη γραμμική διάταξη ωστε να βρίσκεται κάτω απο τον τίτλο.
- και τέλος το `song_duration` για την απεικόνιση της διάρκειας οπου επίσης βρίσκεται μέσα στην εμφωλευμένη γραμμική διάταξη.

Σχήμα 4.3: Απεικόνιση αντικειμένου Song

Αρχείο TracklistTab.xml

Περιέχει ενα listView ωστε να γίνεται η απεικόνιση όλων των τραγουδιών σε μορφή λίστας.

Σχήμα 4.4: Τελική απεικόνιση του Tracklist Tab σε συσκευή android

4.3.3 Καρτέλα Αλμπουμ

Αρχείο Albums

Ως πρώτο βήμα για την λειτουργία της καρτέλας αλμπουμ δημιουργείται η κλάση Albums που παρέχει αντικείμενα αυτού του τύπου. Παρακάτω βρίσκεται ο κώδικας κατασκευής, οπου παρατηρείται τι παράμετρους χρειάζεται για την δημιουργία ενος Album.

```
public Albums(String albumName, String albumArt, String artistname,
    int albumId) {

       AlbumName = albumName;
       AlbumArt = albumArt;
```

```
        Albumartist = artistname;
        Albumid = albumId;

    }
```

Αρχείο AlbumTab

Πρόκειται επίσης για ένα fragment και αποτελεί την 2η καρτέλα. Η λογική του κώδικα είναι εφάμιλλη με τον αντίστοιχο του πρωτου μας fragment TracklistTab, μόνο που σε αυτή την περίπτωση ψάχνει την συσκεύη για αλμπουμ και δημιουργούνται αντικείμενα τύπου Albums και ύστερα τα προσθέτονται σε ένα ArrayList. Για την οπτική εμφάνιση του κάθε album ατομικά δημιουργείται μια νεα κλάση τύπου baseAdapter, οπου ονομάζεται AlbumAdapter.

```
public void getAlbums() {

 AlbumResolver = getActivity().getContentResolver();
 Uri albumArtUri = MediaStore.Audio.Albums.EXTERNAL_CONTENT_URI;
 Cursor AlbumCursor =
 AlbumResolver.query(albumArtUri, null, null, null, null);
 AlbumArrayList = new ArrayList<Albums>();

if (AlbumCursor != null && AlbumCursor.moveToFirst()) {

    int AlbumColumn =
    AlbumCursor.getColumnIndex(MediaStore.Audio.Media.ALBUM);
    int AlbumArtColumn =
    AlbumCursor.getColumnIndex(MediaStore.Audio.Albums.ALBUM_ART);
    int AlbumId =
    AlbumCursor.getColumnIndex(MediaStore.Audio.Albums._ID);
    int AlbumContent =
    AlbumCursor.getColumnIndex(MediaStore.Audio.Albums.ARTIST);

    do {

        String thisAlbumTitle = AlbumCursor.getString(AlbumColumn);
        String thisAlbumArt = AlbumCursor.getString(AlbumArtColumn);
        int thisAlbumId = AlbumCursor.getInt(AlbumId);
        String thisArtist = AlbumCursor.getString(AlbumContent);

        AlbumArrayList.add(new Albums(thisAlbumTitle, thisAlbumArt,
        thisArtist, thisAlbumId));

        } while (AlbumCursor.moveToNext());
    }
    AlbumCursor.close();
}
```

Ακόμα, οταν ο χρήστης επιλέξει κάποιο άλμπουμ, η εφαρμογή μέσω ένος intent το οποίο ανοίγει ένα νέο activity που ονομάζεται albumContent, μεταφέρει την λίστα με τα τραγούδια του album. Για την εύρεση των τραγουδιών

δημιουργείται μια μέθοδος και ονομάζεται MatchSongstoAlbum(int AlbumId), η οποία είναι παρόμοια με την μέθοδο getSongs() του TracklistTab, με την διαφορά οτι δέχεται σαν παράμετρο την ταυτότητα του album, οπως επίσης και ένας if κόμβος που ελέγχει αν το albumId που δέχεται σαν παράμετρο είναι ίδιο με αυτό που βρίσκεται στον κέρσορα, σε περίπτωση που είναι προσθέτει το τραγούδι στο αλμπουμ.

```
if (thisAlbumId == Albumid) {
  if (Duration > 0.1) {

      AlbumSongs.add(new Song(thisId, thisTitle, thisArtist,
      thisAlbum, FinalDuration, thisAlbumId));

                  }
              }
```

Τέλος, σε αντίθεση με την 1η καρτέλα που εμφανίζεται σε μορφή ListView, εδώ εμφανίζεται σε μορφή gridView.

Αρχείο AlbumAdapter

Όπως και στο SongAdapter ο constructor δέχεται 2 παράμετρους μία λίστα που περιέχει albums και context της εφαρμογής.

```
public AlbumAdapter(ArrayList<Albums> theAlbums, Context context) {

       albumList = theAlbums;
       albuminf = LayoutInflater.from(context);

  }
```

Η διαφορά σε σχέση με τον SongAdapter πέρα απο την απεικόνιση είναι οτι εδω χρησιμοποιείται η κλάση viewHolder για την βελτιστοποίηση της χρήσης της μνήμης.

```
static class ViewHolder{
      ImageView coverView;
      TextView albumTitle;

 }

  @Override
  public View getView(int position, View convertView, ViewGroup parent) {

      final ViewHolder albumDisplayHolder;
      View AlbumLay = convertView;

      if(AlbumLay==null) {
```

```
            AlbumLay = (RelativeLayout) albuminf.inflate
                    (R.layout.album, parent, false);

            albumDisplayHolder = new ViewHolder();
            albumDisplayHolder.coverView =
            (ImageView) AlbumLay.findViewById(R.id.image);
            albumDisplayHolder.albumTitle =
            (TextView) AlbumLay.findViewById(R.id.Album_Name);

            AlbumLay.setTag(albumDisplayHolder);

        }
        else{
                albumDisplayHolder = (ViewHolder) AlbumLay.getTag();
        }
        Albums currAlbum = albumList.get(position);
        final String AlbumArt =  currAlbum.getAlbumArt();

        albumDisplayHolder.albumTitle.setText(currAlbum.getAlbumName());
        tf=ResourcesCompat.getFont(
        (albumDisplayHolder.albumTitle).getContext(), R.font.liquidrom);
        albumDisplayHolder.albumTitle.setTypeface(tf);
```

Απο τον παραπάνω κώδικα παρατηρείται οτι για την απεικόνιση χρησιμοποιούνται 2 views, ένα ImageView και ένα TextView οπου πρόκειται για το εξώφυλλο και τον τίτλο του άλμπουμ αντίστοιχα. Αφού περάστεί το όνομα του αλμπουμ χρησιμοποιώντας την setText μέθοδο και επιλέγεται η επιθυμητή γραμματοσειρά μέσω της getfont() μένει η εισάγωγή του εξώφυλλου στο ImageView, αυτό επιτυνγχάνεται με τον παρακάτω κώδικα οπου έιναι η συνέχεια του προηγούμενου.

```
  new Thread(new Runnable() {
    @Override
    public void run() {
        albumDisplayHolder.coverView.post(new Runnable() {
        @Override
        public void run() {
            if(AlbumArt!=null) {
                Bitmap bm = BitmapFactory.decodeFile(AlbumArt);
                albumDisplayHolder.coverView.setImageBitmap(bm);
            }else {
                albumDisplayHolder.coverView.setImageResource
                (R.drawable.generic);
            }
        }
    });
  }
}).start();
```

Πρόκειται για ένα νήμα που ελέγχει άμα το album έχει εξώφυλλο, σε περίπτωση που διαθέτει τότε το μετατρέπει το αρχείο σε μορφή bitmap και ύστερα το περναέι στο ImageView του αλμπουμ. Στην δεύτερη περίπτωση στη θέση του imageView μπαίνει μια εικόνα πού προέρχεται από τους πόρους της εφαρμογής.

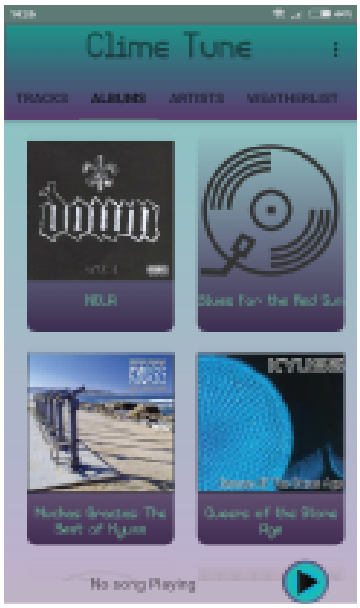
Album.xml

Το αρχείο Album.xml είναι αυτό που περιέχει τον κώδικα για την απεικόνιση του κάθε αλπμουμ ατομικά. Όπως αναφέρθηκε και παραπάνω για τον σχεδιασμό του άλμπουμ χρησιμοποιείται ένα Image View και ένα TextView τα οποία αρχικοποιούνται σε αυτό το αρχείο.

Σχήμα 4.5: Απεικόνιση album με εξώφυλλο και χωρίς

AlbumTab.xml

Περιέχει μόνο ένα gridView για την αναπαράσταση των αλμπουμ σε πλέγμα.



Σχήμα 4.6: Καρτέλα αλμπουμ

AlbumContent

Πρόκειται για την δραστηριότητα που περιέχει τα τραγούδια του επιλεγμένου άλμπουμ. Λαμβάνει την λίστα μέσω του ενός intent όπως αναφέρθηκε και παραπάνω.

```
Intent getUIlist = getIntent();
AlbumSongsList =  getUIlist.getParcelableArrayListExtra("albumSongs");
String AlbumName = getUIlist.getStringExtra("AlbumName");
```

Η απεικόνιση είναι σε μορφή λίστας, ενώ το κάθε τραγούδι ατομικά εμφανίζεται όπως και στην καρτέλα tracklistTab. Οταν ο χρήστης επιλέξει κάποιο κομμάτι τότε μέσω ενός intent ξεκινάει την PlayMusicActivity δραστηριότητα και στέλνει την λίστα με τραγούδια του αλμπουμ.

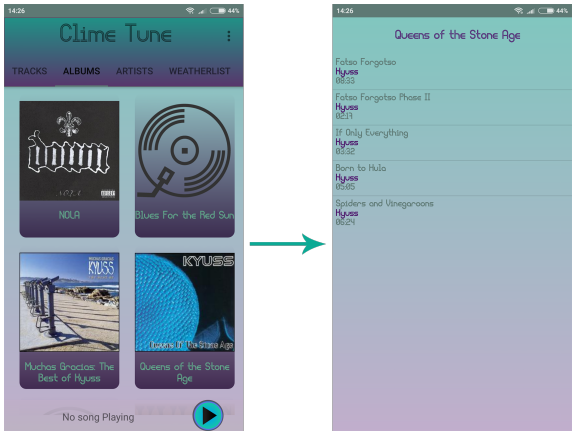
```
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
      @Override
      public void onItemClick(AdapterView<?> parent, View view,
      int position, long id) {

          Intent startmusicactivity =
      new Intent(getApplicationContext(),PlayMusicActiviy.class);

          startmusicactivity.putExtra("SongList",AlbumSongsList);
          startmusicactivity.putExtra("pos",position);
          startActivity(startmusicactivity);
 });
```

AlbumSongsAdapter & album_songs.xml

Για την αναπαράσταση κάθε αρχείου ατομικά χρησιμοποιείται η παραπάνω κλάση που είναι και αυτη επέκταση της baseAdapter. Η κλάση, όπως και η AlbumAdapter, χρησιμοποιεί ένα viewHolder που αποτελεί 3 στοιχεία, τα οποία είναι τα ίδια και με το songAdapter(τίτλος,όνομα καλλιτέχνη,διάρκεια τραγουδιού). Το τελικό αποτέλεσμα αναπαράσταστης είναι το ίδιο με αυτό του songAdapter.Τέλος το αρχείο album_songs.xml περιέχει τα 3 textViews(Album_song_title , Album_song_artist,Album_song_duration) που αναφέραμε και παραπάνω.

album_list.xml

Περιέχει ενα TextView εμφωλευμένο σέ ένα toolbar που παίρνει το όνομα του άλμπουμ που επιλέχθηκε. Όπως και επίσης ενα listView για την απεικόνιση των τραγουδιών σε μορφή λίστας.



Σχήμα 4.7: Απεικόνιση περιεχομένων αλμπουμ

4.3.4 Καρτέλα Artist

Artist

Όπως και στην καρτελά album, το πρώτο βήμα είναι η δημιουργία της κλάσης artist. Ο κώδικας κατασκευής είναι ο παρακάτω, συνεπώς για την δημιουργία ενός αντικειμένου artist χρειαζόμαστε 3 παραμέτρους, το όνομα του καλλιτέχνη, το αριθμό των άλμπουμ , και τον αριθμό των τραγουδιών.

```
    public Artist(String nameofArtist ,String noAlbums,String noTracks) {

NameofArtist = nameofArtist;
NoAlbums = noAlbums;
NoTracks = noTracks;

    }
```

ArtistTab

Είναι ένα fragment που αποτελεί την τρίτη καρτέλα που περιέχει το Main Activity. Ο κώδικας είναι παρόμοιος με αυτόν του albumTab αφού και εδώ χρησιμοποιούνται μεθόδους όπως getAlbums() καί MatchSongstoAlbum(int AlbumId) μόνο που σε αυτή την περίπτωση είναι getArtist() και MatchAlbums toArtists(final String artistName).

```
private void MatchAlbumstoArtist(final String artistName) {

    ContentResolver ArtistAlbumResolver =
    getActivity().getContentResolver();
    Uri ArtistAlbumUri =
    MediaStore.Audio.Albums.EXTERNAL_CONTENT_URI;
    Cursor AlbumArtistCursot =
    ArtistAlbumResolver.query(ArtistAlbumUri, null, null, null, null);
    ArtistAlbums = new ArrayList<Albums>();


    if (AlbumArtistCursot != null && AlbumArtistCursot.moveToFirst()) {

        int AlbumColumn =
        AlbumArtistCursot.getColumnIndex(MediaStore.Audio.Media.ALBUM);
        int AlbumArtColumn =
        AlbumArtistCursot.getColumnIndex(MediaStore.Audio.Albums.ALBUM_ART);
        int AlbumId =
        AlbumArtistCursot.getColumnIndex(MediaStore.Audio.Albums._ID);
        int AlbumContent =
        AlbumArtistCursot.getColumnIndex(MediaStore.Audio.Albums.ARTIST);

        do {

            String thisAlbumTitle =
            AlbumArtistCursot.getString(AlbumColumn);
            String thisAlbumArt =
            AlbumArtistCursot.getString(AlbumArtColumn);
            int thisAlbumId =
            AlbumArtistCursot.getInt(AlbumId);
            String thisArtist =
            AlbumArtistCursot.getString(AlbumContent);

            if (thisArtist.equals(artistName)) {

                ArtistAlbums.add(new Albums(thisAlbumTitle,
                thisAlbumArt, thisArtist, thisAlbumId));
            }

        } while (AlbumArtistCursot.moveToNext());


    }AlbumArtistCursot.close();
```

Η μεθοδος MatchAlbumstoArtist(final String artistName) δημιουργεί την λίστα των albums που έχει ένας καλλιτέχνης, η οποία χρησιμοποιείται στην μέθοδο setOnItemClickListener και στέλνει την λίστα μέσω ενος intent σε ένα activity που ονομάζεται ArtistContent όπως επίσης και το όνομα του καλλιτέχνη. Για την εμφάνιση των artist στην 3η καρτέλα χρησιμοποιείται ενα ακόμα adapter και ονομάζεται ArtistAdapter.

ArtistAdapter & artists.xml

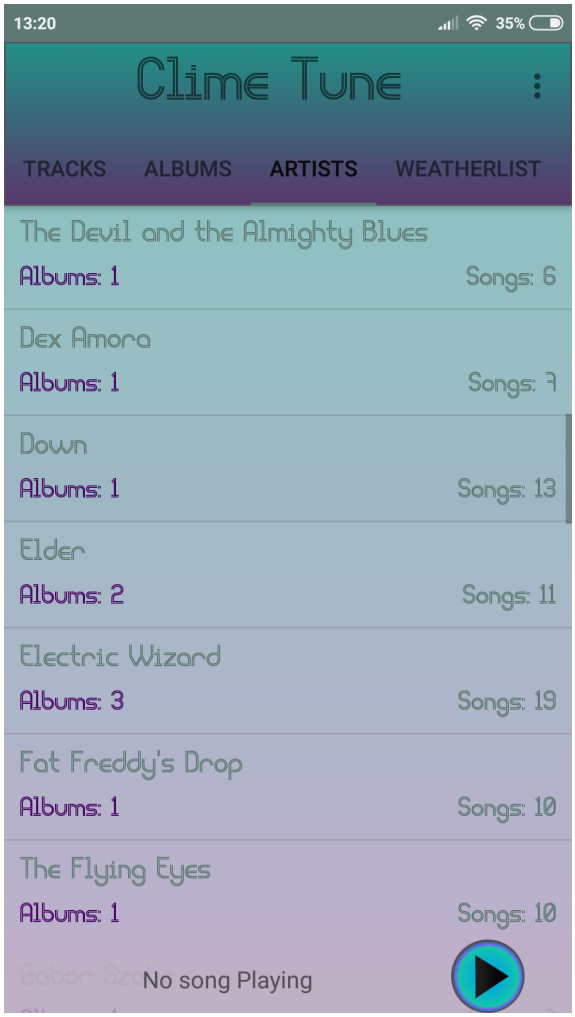
Η λογική είναι ακριβώς ίδια σε σχέση με τους προηγούμενους adapter, σε αυτή την περίπτωση, για την απεικόνιση των Artist χρησιμοποιούνται 3 textView που βρίσκονται στο αρχείο artist.xml πρόκειται για τα :

- ArtistName, οπου εμφανίζεται το όνομα του καλλιτέχνη.
- AlbumsOfArtist, το οποίο θα απεικονίζει τον αριθμό των αλμπουμ του καλλιτέχνη
- NoTracks, που πρόκειται για τον αριθμό των τραγουδιών του καλλιτέχνη.

Σχήμα 4.8: Απεικόνιση Artist

ArtistTab.xml

Περιέχει ενα listView για την απεικόνιση των Artists σε μορφή λίστας.

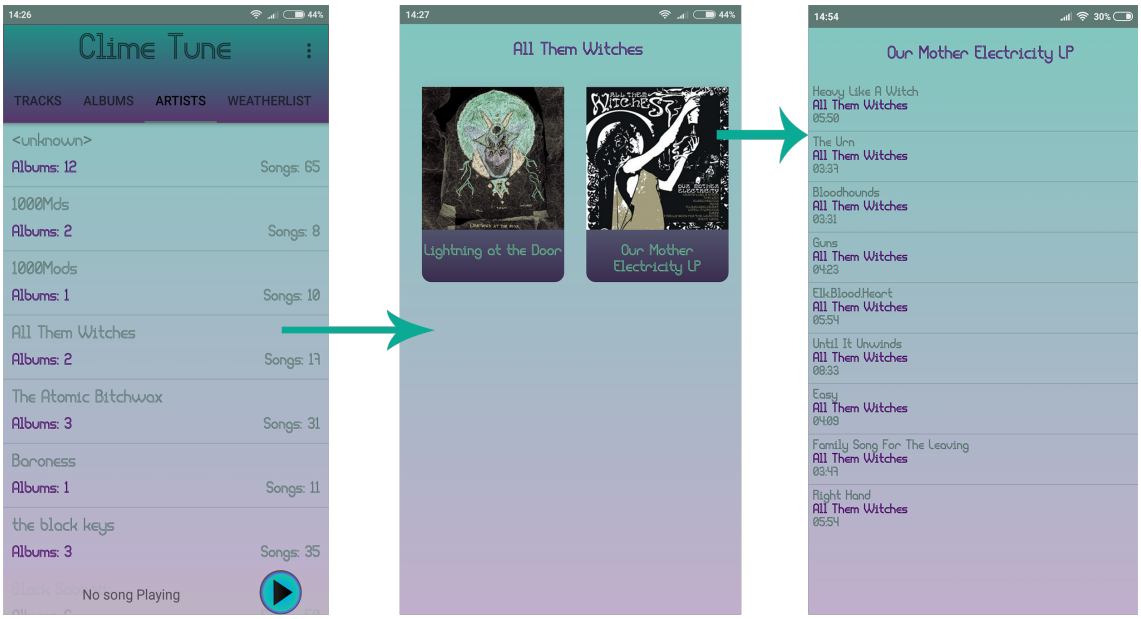


Σχήμα 4.9: Απεικόνιση ArtistTab

ArtistContent & artist_list

Η δραστηριότητα ArtistContent περιέχει την συλλογή των αλμπούμ του καλλιτέχνη που επιλέχθηκε απο το albumTab. Εμφανίζει τα albums σε μορφή πλέγματος(gridView) και στην επικεφαλίδα εμφανίζεται και το όνομα του καλλιτέχνη. Η κλάση για την απεικόνιση των album μεμονωμένα χρησιμοποιεί ως adapter τον AlbumAdapter που αναφέρθηκε παραπάνω, αφού και σε αυτή την περίπτωση γίνεται διαχείριση αντικειμένων τυπου album. Η κλάση αυτη περιέχει μια μέθοδος που ονομάζεται MatchSongstoArtistAlbum (final int Albumid) και παίρνει ως παράμετρο την ταυτότητα του album, και δημιουργεί μία νέα λίστα με τα τραγούδια του album όπως και η MatchSongsto Album(final int Albumid) μέθοδος που περιέχεται στην κλάση albumTab που έχει αναφερθεί. Οσον αφορά την setOnItemClickListener μέθοδος, στέλνει, μέσω ένος intent την λίστα των τραγουδιών που βρίσκεται στο album

και το όνομα του αλμπουμ, στην albumContent κλάση που αναλύθηκε παραπάνω απο κει και πέρα ακολουθέι την διαδικασία που έχει ήδη ειπωθεί.

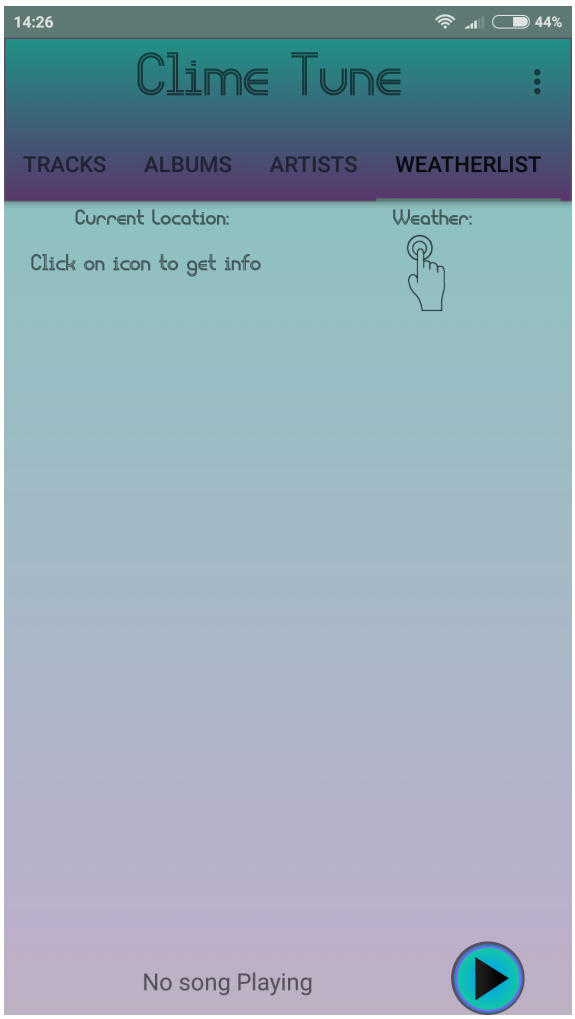


Σχήμα 4.10: Απο το ArtistTab(αριστερά) στο ArtistContent(μέση) και τέλος στο albumContent(δεξια)

4.3.5 Καρτέλα WeatherList

YouTubeFragment

Πρόκειται για την τελευταια καρτέλα της εφαρμογής η οποία, ουσιαστικά, εκτελεί 3 κύριες διεργασίες οι οποίες αναλύονται παρακάτω. Όταν ο χρήστης βρεθεί στην 4η καρτέλα θα αντικρίσει το παρακάτω Interface :



Σχήμα 4.11: Αρχική επιφάνεια καρτέλας WeatherList

Όπως παρατηρείται η οθόνη περιέχει 2 πληροφορίες για τον χρήστη, ενα textView το οποίο παροτρύνει τον χρήστη να πατήσει πάνω στην 2η πληροφορία που πρόκειται για ενα imageView και κατα αυτόν τον τρόπο να δωθεί η εντολή ωστε η εφαρμογή να μπορει να αντλήσει τις πληροφορίες του καιρού και να παρέχει στο χρήστη μια λίστα μουσικής.
Όταν ο χρήστης πατήσει το εικόνιδιο τότε η εφαρμογή επιχειρεί αρχικά να βρεί την τοποθεσία του χρήστη με την βοήθεια της μεθόδου getLocation().

```
public void getLocation() {

    LocationManager locationManager = (LocationManager)
    getActivity().getSystemService(Context.LOCATION_SERVICE);

    Location loc  = locationManager.getLastKnownLocation
    (LocationManager.NETWORK_PROVIDER);

    if(loc!=null) {
        Longitude = loc.getLongitude();
        Latitude = loc.getLatitude();
     }
```

Με την βοήθεια του LocationManager που παρέχεται απο το πακέτο του android.location η εφαρμογή μπορεί να εντοπίσει την τοποθεσία και να αποθηκεύσει την συντεταγμένες της τοποθεσιας στις μεταβλητές Longitude και Latitude. Αφού γίνει η ανάκτηση των δύο μεταβλητών, το επόμενο βήμα είναι η μέθοδος getJSON(Latitude,Longtitude) η οποία αποσκοπεί στην άντληση πληροφοριών σχετικά με τις καιρικές συνθήκες που επικρατούν στις συντεταγμένες που δώθηκαν σαν παράμετροι. Η παραπάνω διαδικασία βρίσκεται μέσα σε μία AsyncTask<void,void,void>() ώστε να εκτελείται στο υπόβαθρο.

```
new AsyncTask<Void, Void, Void>() {

@Override
protected void onPreExecute() {
  super.onPreExecute();

}

@Override
protected Void doInBackground(Void... params) {
try {
URL url = new
URL("http://api.openweathermap.org/data/2.5/weather?lat=" + lat
+ "&lon="
+lon+"&units=metric"+"&APPID=cea81f0fd6d5074bfc05d32898e32e1b");

HttpURLConnection connection =
(HttpURLConnection) url.openConnection();

BufferedReader reader =
new BufferedReader(new InputStreamReader(connection.getInputStream()));

StringBuffer json = new StringBuffer(1024);
String tmp = "";

while ((tmp = reader.readLine()) != null)
         json.append(tmp).append("\n");
```

```
            reader.close();

            data = new JSONObject(json.toString());

            if (data.getInt("cod") != 200) {
              System.out.println("Cancelled");
              return null;
            }
```

Ο παραπάνω κώδιακας επικοινωνεί με το api της σελίδας OpenWeatherMap η οποία παρέχει όλες τις καιρικές πληροφορίες που χρειάζεται η εφαρμογή. Το επόμενο βήμα είναι η χρησιμοποίηση αυτων των πληροφοριών και η εφαρμογή τους στο UI, για την επίτευξη αυτών δημιουργήθηκε η μέθοδος render Weather (JSONObject jsonObject) που οπως φαίνεται δέχεται ενα αντικείμενο τυπου JSONObject για παράμετρο.

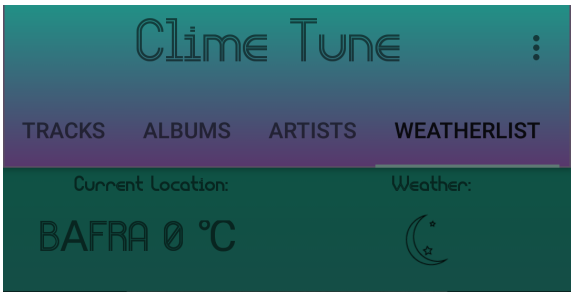
```
JSONObject main = jsonObject.getJSONObject("main");
       tobeConvert = main.getDouble("temp");
       int temperature  = tobeConvert.intValue();
       String celcius = "␣□␣";
       CurrCityText.setText(jsonObject.getString("name").
       toUpperCase()+ "␣" +temperature +celcius);
```

Ο παραπάνω κώδικας χρησιμοποίει το JsonObject που δώθηκε σαν παράμετρο και δημιουργεί ένα Text με το όνομα της περιοχής και την θερμοκρασια(σε βαθμους κελσιου) που βρίσκεται ο χρήστης.

```
JSONObject weatherCondition =
jsonObject.getJSONArray("weather").getJSONObject(0);
CurrentWeather = weatherCondition.getString("id");

 if (CurrentWeather.equals("800")) { // Clear
   if(currTime>=Sunrise && currTime<Sunset)
       CurrWeather.setImageResource(R.drawable.sun);
   else{
       CurrWeather.setImageResource(R.drawable.nightclear);
```

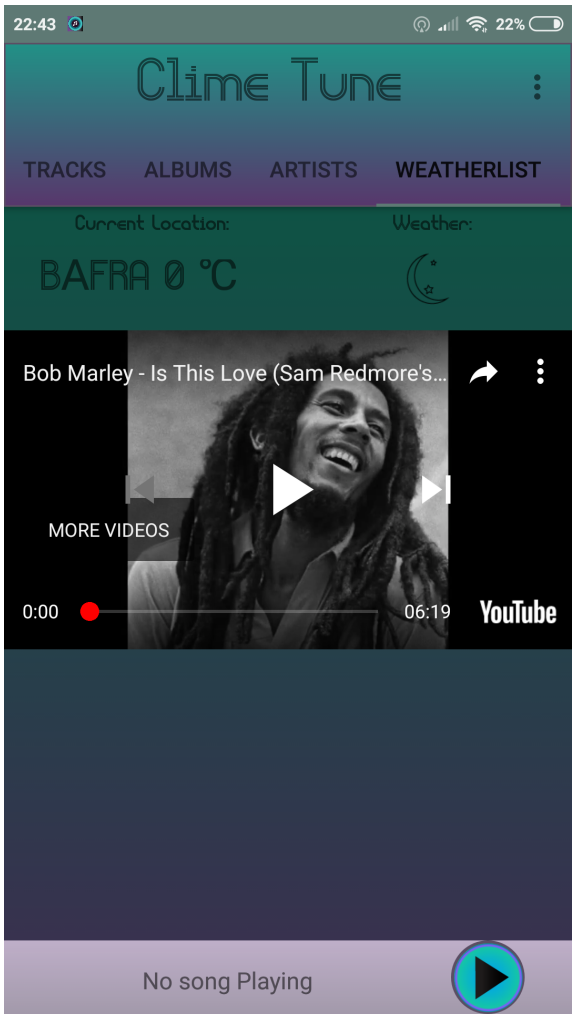
Επίσης, αποθηκεύει την ταυτότητα του καιρού στην μεταβλητή ”Current Weather και μέσω ορισμένων if συνθηκών, ελέγχει σε ποία κατηγορία ανήκει και θέτει τα ανάλογα εικονίδια στο UI.

Τέλος, το τελευταίο βήμα είναι η δημιουργία του εμφωλευμένου player στην 4η καρτέλα, ο παρακάτω κώδικας αρχικοποιεί το player, αλλάζει το background και τέλος καλεί την μέθοδο getPlaylist(YouTubePlayer YPlayer) η οποία προσθέτει στο player την κατάλληλη λίστα, οπου ορίζεται μέσω if συνθηκών.



Σχήμα 4.12: Αριστερά βλέπουμε την περιοχή και την θερμοκρασία ενώ στα δεξιά βρίσκεται το εικονίδιο του καιρού

```
view.setBackground(getResources().getDrawable
(R.drawable.youtube_background));
youTubePlayerFragment.initialize(String.valueOf(YoutubeDeveloperKey),
new YouTubePlayer.OnInitializedListener() {

  @Override
  public void onInitializationSuccess(YouTubePlayer.Provider provider,
  YouTubePlayer player, boolean wasRestored) {

  if (!wasRestored) {
       canfoundlocation = true;
       fragment_added = true;
       Player = player;
       Player.setShowFullscreenButton(false);
       getPlaylist(Player);
      }
 }
```



Σχήμα 4.13: Η καρτέλα WeatherList αφού έχει αρχικοποιηθεί ο player

4.3.6 Αναπαραγωγή Μουσικής

PlayMusic Activity

Πρόκειται για το Activity που διαχειρίζεται το UI για την αναπαραγωγή της μουσικής, όταν ο χρήστης πατήσει σε ένα τραγούδι τότε το activity αυτο δέχεται την θέση του τραγουδιου και την λίστα που ανήκει το τραγούδι.

```
Intent UIinfo = getIntent();
actionis = UIinfo.getAction();

if(actionis=="com.app.sl.tabbedapplication.action.RESTART"){

    songPosition = UIinfo.getIntExtra("PosSong",0);


    if(playSrv.mediaPlayer.isPlaying()) {
```

```
            pausePlay.setImageResource(R.drawable.pausebutton);

        }else{
            pausePlay.setImageResource(R.drawable.playbutton);
        }
    }else {
        songPosition = UIinfo.getIntExtra("pos", 0);
    }
    songList = UIinfo.getParcelableArrayListExtra("SongList");
```

Ο παραπάνω κώδικας μας δείχνει πως η δραστηριότητα δέχεται το intent, όπως επίσης ελέγχει άμα το τραγούδι βρίσκεται ήδη σε αναπαραγωγή ή άμα θα αρχικοποιηθεί τώρα. Ύστερα το activity μπαίνει στο πρώτο στάδιο του κύκλου ζωής την onStart() στην οποία προσπαθεί να χτίσει ένα connection με την υπηρεσία PlayService της εφαρμογής που διαχειρίζεται στο υπόβαθρο την αναπαραγωγή της μουσικής.

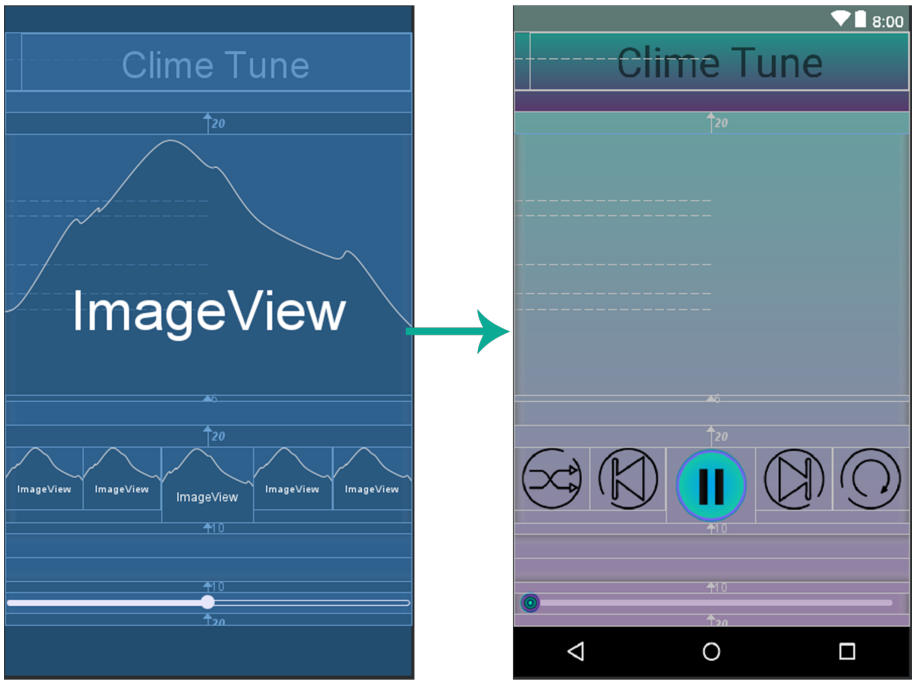
```
@Override
protected void onStart() {
  super.onStart();
  if (PlayMusic == null) {

      Intent PlayMusic =
      new Intent(getApplicationContext(), PlayService.class);

      bindService(PlayMusic, musicConnection, Context.BIND_AUTO_CREATE);
      startService(PlayMusic);

      }
}
```

Στην δημιουργία του musicConnection όπως παρατηρείται και στον παρακάτω κώδικα, χρησιμοποιείται ένα αντικείμενο του PlayService το ”playSrv” και με την βοήθεια του μπορεί και έχει πρόσβαση στης βοηθητικές μεθόδους της υπηρεσίας PlayService ”setAction”,”setSongCurrentDuration”,”setSeekbar” κτλ. με αυτόν τον τρόπο στέλνει πληροφορίες σχετικά με το UI, την λίστα της μουσικής κ.α.

```
private PlayService playSrv;

private ServiceConnection musicConnection = new ServiceConnection() {

 @Override
 public void onServiceConnected(ComponentName name, IBinder service) {
     binder = (PlayService.MusicBinder) service;
     playSrv = binder.getService();
     if(actionis == "com.app.sl.tabbedapplication.action.RESTART"){
         ACTION="com.app.sl.tabbedapplication.action.RESTART";
         playSrv.setAction(ACTION);
     }else{
         if (musicBound) {
           playSrv.setAction(ACTION);
         }
         if (!musicBound) {
             playSrv.setAction(ACTION);
         }
     }

     playSrv.setSongCurrentDurationLabel(songCurrentDurationLabel);
     playSrv.setSeekbar(songProgressBar);
     playSrv.setView(title_info, album_info, artist_info,
     songTotalDurationLabel, cover_info);
     playSrv.setBoolean(isRepeat, isShuffle, isNextPrevButton);
     playSrv.giveList(songList);
     playSrv.setSongPosition(songPosition);

     musicBound = true;

     }
```

play_activity.xml

Ο σχεδιασμός του PlayMusicActivity αποτελείται κυρίως απο image και text Views ενώ περιέχει και ένα seekBar που επιτρέπει στον χρήστη την αναζήτηση συγκεκριμένου χρονικού σημείου στην διάρκεια του τραγουδιού. Ακολουθεί ένα screenshot με τον σχεδιασμό του play_activity.xml



Σχήμα 4.14: Design Tab αρχείου play_activity.xml στο android studio

PlayService

Πρόκειται για την υπηρεσία που αναλαμβάνει την αναπαραγωγή μουσικής αλλα και τον χειρισμό του κύκλου ζωής, όσο υπάρχει κάποιο κομμάτι στο player, η υπηρεσία μένει ζωντανή και λειτουργεί στο υπόβαθρο.

```
public void onCreate(){
    super.onCreate();

    mediaPlayer = new MediaPlayer();
    initMusicPlayer();
    getAlbumArt();
    }
```

Στον παραπάνω κώδικα βλέπουμε οτι δημιουργείται ενα αντικείμενο τύπου Media Player και ύστερα γίνεται η κλήση αρχικά στην μέθοδο initMusic Player() και έπειτα στην getAlbumArt(). Η μέθοδος initMusicPlayer ελέγχει αν το mediaPlayer είναι άδειο και σε περίπτωση που είναι το αρχικοποιεί, ύστερα χρησιμοποιεί ορισμένες μεθόδους της κλάσης MediaPlayer που ”ξυπνουν” το mediaplayer.

```
public void initMusicPlayer(){

    if(mediaPlayer==null) {
            mediaPlayer = new MediaPlayer();
    }

    mediaPlayer.reset();
    mediaPlayer.setWakeMode(getApplicationContext(),
    PowerManager.PARTIAL_WAKE_LOCK);
    mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
    mediaPlayer.setOnPreparedListener(this);
    mediaPlayer.setOnCompletionListener(this);
    mediaPlayer.setOnErrorListener(this);
    }
```

Η μέθοδος getAlbumArt δημιουργεί μια λίστα με όλα τα εξώφυλλα των αλμπουμ, η μέθοδος αυτή είναι εφάμμιλη με της getSongs(), getAlbums() κτλ. Επίσης δημιουργείται μία νέα κλάση η οποία ονομάζεται AlbumArt και έτσι υπάρχει η δυνατότητα δημιουργίας αντικειμένων τέτοιου τύπου.

```
public void getAlbumArt() {
    ArtResolve = getApplicationContext().getContentResolver();
    Uri albumArtUri = MediaStore.Audio.Albums.EXTERNAL_CONTENT_URI;
    Cursor AlbumCursor = ArtResolve.query
    (albumArtUri, null, null, null, null);
    AlbumArtList = new ArrayList<>();


    if (AlbumCursor != null && AlbumCursor.moveToFirst()) {

        int AlbumArtColumn =
        AlbumCursor.getColumnIndex(MediaStore.Audio.Albums.ALBUM_ART);
        int AlbumId =
        AlbumCursor.getColumnIndex(MediaStore.Audio.Albums._ID);



        do {

            String thisAlbumArt = AlbumCursor.getString(AlbumArtColumn);
```

```
            int thisAlbumId = AlbumCursor.getInt(AlbumId);

            AlbumArtList.add(new AlbumArt(thisAlbumArt, thisAlbumId));


        } while (AlbumCursor.moveToNext());
    }
    AlbumCursor.close();
}
```

Όταν αρχικοποιείται η playService όπως έχει αναφερθει δέχεται κάποιες παραμέτρους απο την δραστηριότητα playMusicActivity, παρακάτω αναδυκνύεται πώς η υπηρεσία δέχεται και χρησιμοποιεί τις πληροφορίες που δέχεται.

```
public void setAction(String PlayAction){
        action = PlayAction;
    }
```

Στην παραπάνω μέθοδο η υπηρεσία δέχεται την δράση η οποία θέτει αν πρόκειται για πρώτω φορά που αρχικοποιείται το player ή αν το player είναι ήδη ενεργό και ο χρήστης έχει επιλέξει κάποιο νεό τραγούδι και η τελευταία περίπτωση άμα έγινε η επανεκκίνηση της PlayMusicActivity.

```
public void setView(TextView title , TextView album , TextView artist ,
    TextView duration, ImageView cover){
        title_display = title;
        album_display= album;
        artist_display = artist;
        songTotalDurationLabel = duration;
        cover_display = cover;
    }
```

Η μέθοδος setView δέχεται 5 παραμέτρους, πρόκειται για της θέσεις που θα περιέχουν τίτλο,το όνομα του αλπουμ, του καλλιτέχνη , την διάρκεια και τέλος το εξώφυλλο. Με αυτόν τον τρόπο γίνεται η σύνδεση μεταξύ του UI και της υπηρεσίας καί έτσι μπορούν να περαστούν οι κατάλληλες τιμές στις παραμετρους.

```
public void setBoolean(boolean onRepeat,boolean onShuffle,
        boolean onNextPrev){

    isRepeat = onRepeat;
    isShuffle = onShuffle;
    isNextPrevButton =  onNextPrev;
    }
```

Σε αυτή την μέθοδο οι παράμετροι που δέχεται η υπηρεσία σχετίζονται με

την λειτουργία των κουμπιών, δηλαδή αμα ο χρήστης έχει ενεργοποιήσει την επανάληψη τραγουδιού, την τυχαία αναπαραγωγή ή πάτησε το επόμενο - προηγούμενο τραγούδι ή απλα διάλεξε κάποιο τραγούδι απο την λίστα.

```
public void setSongPosition(int pos){

    positionOfSong = pos;
    if(isNextPrevButton){

        songDisplay(positionOfSong);
    }
    if(!isRepeat && !isShuffle) {
        songDisplay(positionOfSong);
    }
}
```

Η μέθοδος setSongPosition δέχεται την θέση του τραγουδιού και ελέγχει με 2 συνθήκες if άμα ο χρήστης έχει ενεργοποιήσει κάποιο κουμπί ή όχι, ανεξαρτήτως του αποτελέσματος σε αυτή την μέθοδο καλείται η μέθοδος songDisplay που θα αναλύσουμε πιο κάτω.

```
public void setSeekbar(SeekBar seekbar){
    ServiceSongProgressBar = seekbar;
    }
```

Στην μέθοδο setSeekbar περνάει το seekBar στην υπηρεσία.

```
public void setSongCurrentDurationLabel(TextView currDuration){

    ServiceSongCurrentDurationLabel = currDuration;

}
```

Η μέθοθος setSongCurrentDurationLabel δέχεται ενα TextView το οποίο θα απεικονίζει την τωρινή διάρκεια του τραγουδιού.

```
public void giveList(ArrayList songlist){
    SongList = songlist;
    }
```

Τέλος στην μέθοδο giveList περνάμε την λίστα τραγουδιών στην υπηρεσία.

```
public void songDisplay(int Position) {

    long SongId = SongList.get(Position).getId();
    SongTitle = SongList.get(Position).getTitle();
    SongArtist = SongList.get(Position).getArtist();
```

```
    SongAlbum = SongList.get(Position).getAlbum();
    SongDuration = SongList.get(Position).getDuration();
    int SongAlbumId = SongList.get(Position).getAlbumId();

    tf = ResourcesCompat.getFont(getApplicationContext(), R.font.liquidrom);
    album_display.setText(SongAlbum);
    album_display.setTypeface(tf);
    title_display.setText(SongTitle);
    title_display.setTypeface(tf);
    songTotalDurationLabel.setText(SongDuration);
    songTotalDurationLabel.setTypeface(tf);
    artist_display.setText(SongArtist);
    artist_display.setTypeface(tf);

    song = new Song(SongId, SongTitle, SongArtist, SongAlbum,
    SongDuration, SongAlbumId);

    findAlbumArt(SongAlbumId);
    playSong(song);

    }
```

Η επόμενη μέθοδος που ακολουθεί είναι η songDisplay(int Position) που δέχεται σαν παράμετρο την θέση του τραγουδιού. Στην μέθοδο songDisplay η υπηρεσία χρησιμοποιεί ορισμένους getters ώστε να εξάγει τα στοιχεία του τραγουδιού και έπειτα με κάποια setters περνάει τις τιμές στα TextViews. Τέλος δημιουργεί ένα αντικείμενο τύπου song, αρχικοποιεί την μέθοδο findAlbum Art(int Album_id) όπως επίσης και την μέθοδο playSong(Song song).

```
 public void  findAlbumArt(int Album_id) {

    for(AlbumArt art:AlbumArtList) {
        if (art.getAlbumArtId() == Album_id) {
            final String RawAlbumArt = art.getAlbumArt();
                new Thread(new Runnable() {
                @Override
                    public void run() {
                    cover_display.post(new Runnable() {
                    @Override
                    public void run() {
                        if (RawAlbumArt != null) {
                            Bitmap artwork = BitmapFactory.decodeFile
                            (RawAlbumArt);
                            cover_display.setImageBitmap(artwork);
                        } else {
                            cover_display.setImageResource
                            (R.drawable.generic);
                            }
                        }
                    });
```

```
                }
            }).start();
        }
    }
}
```

Η μέθοδος findAlbumArt περιέχει ένα νήμα που ελέγχει άμα το τραγούδι περιέχει κάποιο cover η άμα θα τοποθετήσει ένα γενικό εικονίδιο στην θέση του imageView.

```
public void playSong(Song song){

    long songID = song.getId();
    Uri trackUri = ContentUris.withAppendedId
    (MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,songID);

    if (ACTION == action) {
        try {
            mediaPlayer.reset();

            mediaPlayer.setDataSource(getApplicationContext(),trackUri);

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

        mediaPlayer.prepareAsync();

        ServiceSongProgressBar.setProgress(0);
        ServiceSongProgressBar.setMax(100);
    }
    if(PLAYBACKACTION==action){
        mediaPlayer.stop();
        mediaPlayer.reset();
        mediaPlayer.release();
        mediaPlayer = null;

        initMusicPlayer();

        try {
            mediaPlayer.setDataSource(getApplicationContext(),trackUri);

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

        mediaPlayer.prepareAsync();
        ServiceSongProgressBar.setProgress(0);
        ServiceSongProgressBar.setMax(100);
```

```
}
if(REOPEN==action){
        action=PLAYBACKACTION;
        }
```

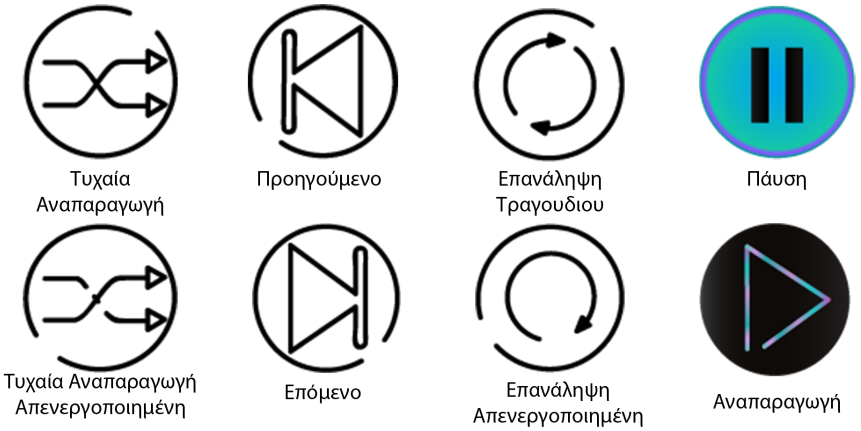
Η μέθοδος playSong είναι ο τελικός προορισμός που αναλαμβάνει την διαχείριση του κύκλο ζωής του mediaPlayer και αναλογα με το action που δέχεται, εκτελεί συγκεκριμένες εντολές.

Σχήμα 4.15: Αναπαραγωγή τραγουδιού στο PlayMusicActivity

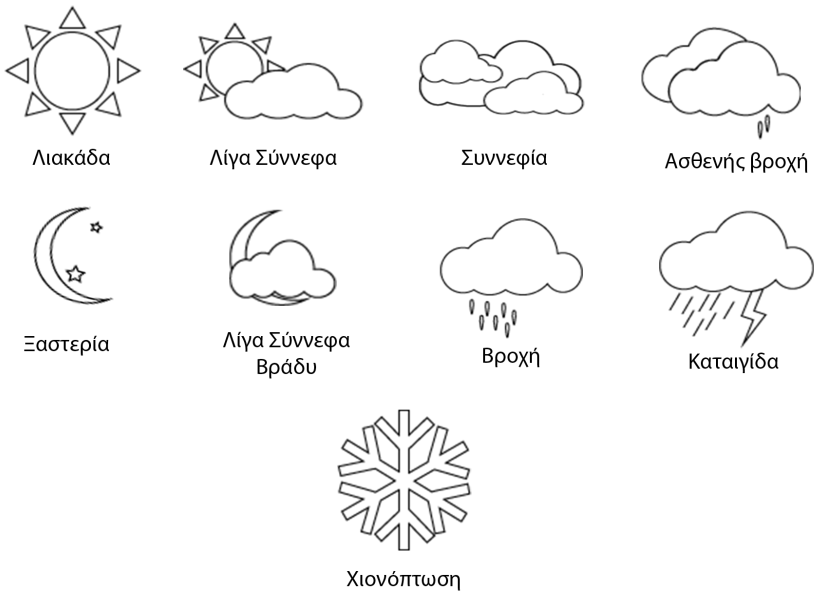
4.3.7 Συμπληρωματικα Υπόλοιπα στοιχεία εφαρμογής

Εικονίδια

Τα εικονίδια της εφαρμογής έχουν σχεδιαστεί στο πρόγραμμα Adobe Illustrator. Η θέση τους βρίσκεται στον φάκελο res/drawable.



Σχήμα 4.16: Εικονίδια του Player



Σχήμα 4.17: Εικονίδια καιρού

Backgrounds

Τα backgrounds είναι αρχεία xml τα οποία ορίζουν τα χρώματα του φόντου. Τα backgrounds βρίσκονται και αύτα στο φάκελο drawable και είναι τα:

- main_background.xml: Το φόντο του Main Activity και των fragments του activity.
- playactivity_background.xml: Το φόντο του PlayMusicActivity.
- album_background.xml: Πρόκειται για το φόντο του περιγράμματος του κάθε Album.
- header_background.xml: Το φόντο της επικεφαλίδας της εφαρμογής
- footer_background.xml: Το φόντο της υποσελίδας που βρίσκεται στο MainActivity
- youtube_background.xml: Το φόντο του τελευταίου fragment.

Σχήμα 4.18: Το φόντο του Main Activity(Αριστερά) και του Youtube Fragment (Δεξιά)

Ο σκοπός της παρούσας πτυχιακής είναι υλοποίηση εφαρμογής σε λειτουργικό σύστημα Android. Επίσης, η ανάδειξη της επικοινωνίας μεταξύ εφαρμογών και εξωτερικών πακέτων δεδομένων. Αναλυτικότερα, πρόκειται για ένα Music Player που εκτός της αναπαραγωγής μουσικής μέσα απο την βιβλιοθήκη του τηλεφώνου, επικοινωνεί με ένα εξωτερικό API και λαμβάνει τις καιρικές συνθήκες που επικρατούν στην περιοχή που βρίσκεται ο

χρήστης και του προτείνει λίστες αναπαραγωγής αναλόγως τις συνθήκες. Το πρώτο κεφάλαιο της εργασίας γίνεται μια εισαγωγή στον προγραμματισμό σε Android και αναλύονται βασικά στοιχεία. Στο δέυτερο κεφάλαιο γίνεται μια αναφορά στο περιβάλλον ανάπτυξης της εφαρμογής, το Android Studio. Στην συνέχεια, αναλύονται κάποια δομικά στοιχεία που χρησιμοποιήθηκαν εκτενώς για την υλοποίηση της εφαρμογής. Τέλος στο 4ο και τελευταίο κεφάλαιο γίνεται ανάλυση του κώδικα της εφαρμογής και η παρουσίαση της μέσω εικόνων.

Βιβλιογραφία

[1] Rick Boyer and Kyle Mew, **Android Application Development Cookbook**, Packt, 2nd edition, March 2016.

[2] Kyle Mew, **Mastering Android Studio 3**,Packt,August 2017.

[3] Android Developers, **Android Developers Documentation**, Available : https://developer.android.com/docs/