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

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Διαδικτυακή Εφαρμογή Android για εύρεση και δήλωση αδέσποτων ζώων

Της φοιτήτριας

Καρυπίδου Κυριακής

Αρ. Μητρώου: 134078

Επιβλέπων καθηγητής

Κεραμόπουλος Ευκλείδης

Θεσσαλονίκη 2018

ΠΡΟΛΟΓΟΣ

Το Internet έχει διεισδύσει στην καθημερινή μας ζωή και σιγά σιγά επηρεάζει κάθε ανθρώπινη δραστηριότητα. Τα smart phones και το διαδίκτυο μεταβάλλουν τον τρόπο με τον οποίο επικοινωνούμε και ολοκληρώνουμε καθημερινές ασχολίες/υποχρεώσεις. Στο προσκήνιο έχουν εμφανιστεί ιστοσελίδες δήλωσης χαμένων ζώων και τρόποι εύρεσης μέσα από κοινωνικά δίκτυα χωρίς όμως να εμπλέκουν τις κινητές συσκευές και τις δυνατότητες που αυτές φέρουν. Αξιοποιώντας λοιπόν τις δυνατότητες του διαδικτύου και των έξυπνων κινητών συσκευών που πλέον έχει η πλειοψηφία των ανθρώπων, ο εντοπισμός ενός ζώου μέσω αυτής της εφαρμογής διευκολύνεται σε μεγάλο βαθμό.

Σκοπός της πτυχιακής είναι να συνδυάσει τις τεχνολογίες που βρίσκονται στη ζωή μας για να εξελίξει τον τρόπο με τον οποίο γίνεται η δήλωση αλλά και ο εντοπισμός χαμένων ζώων μέχρι στιγμής.

ΠΕΡΙΛΗΨΗ

Σκοπός της πτυχιακής εργασίας είναι η δημιουργία μιας εφαρμογής η οποία θα αξιοποιήσει τις δυνατότητες ενός Android smartphone προκειμένου να εξελίξει τον ήδη υπάρχον τρόπο εντοπισμού αδέσποτων ζώων. Χρησιμοποιώντας την τελική εφαρμογή ένας χρήστης θα μπορεί να κάνει μία δημοσίευση για ένα αδέσποτο ζώο που εντόπισε, αναφέροντας ορισμένες πληροφορίες γι' αυτό. Οι πληροφορίες αφορούν διάφορα φυσικά χαρακτηριστικά του και μια περιγραφή που θα περιέχει συνοπτικά σε τι κατάσταση είναι το ζώο ή κάτι που θεωρείται αξιοσημείωτο. Επιπλέον, θα προσδιορίζεται αν έχει κάποιο πρόβλημα και θα επιλέγεται η περιοχή που εντοπίστηκε. Κατά την διάρκεια της δημοσίευσης θα υπολογίζεται αυτόματα το γεωγραφικό μήκος και πλάτος του σημείου εντοπισμού του ζώου. Έτσι, ένας άλλος χρήστης που μπορεί να το έχει χάσει μπορεί να χρησιμοποιήσει την εφαρμογή για να βρει άμα έχει εντοπιστεί κάποιο ζώο στην περιοχή του με αυτά τα χαρακτηριστικά. Έπειτα, οι δύο χρήστες μπορούν να έρθουν σε επαφή μοιράζοντας τα στοιχεία τους.

Πρώτα, θα γίνει εισαγωγή στην πλατφόρμα Android όπου θα επεξηγηθούν κάποια βασικά θέματα που αφορούν την καλύτερη υποστήριξη του κώδικα σε διαφορετικές εκδόσεις Android.

Στη συνέχεια, θα αναφερθούν οι βιβλιοθήκες που με βοήθησαν να ολοκληρώσω την υλοποίηση.

Επιπλέον θα αναφερθούμε εκτενώς στο μοντέλο MVP και σε πραγματικά δείγματα κώδικα όπως ακριβώς εφαρμόστηκε το μοντέλο στην εφαρμογή.

Έπειτα, θα γίνει πλήρως κατανοητό τι προσφέρει το Firebase, πως λειτουργεί, και θα εξηγήσουμε πως μπορούμε να γράψουμε και να διαβάσουμε δεδομένα από τη βάση.

Τέλος, θα περιγράψουμε τις τεχνολογίες με τις οποίες μπορεί να υπολογιστεί η τοποθεσία σε μια συσκευή Android εξηγώντας τον τρόπο λειτουργίας τους. Θα γίνει επεξήγηση του κώδικα που αναπτύχθηκε, χρησιμοποιώντας έναν συνδυασμό διαφόρων τεχνολογιών για καλύτερα αποτελέσματα.

Η πτυχιακή θα ολοκληρωθεί αναφέροντας ορισμένα συμπεράσματα που προέκυψαν κατά την εκπόνηση της καθώς και πιθανές μελλοντικές επεκτάσεις της εφαρμογής.

ABSTRACT

The aim of this thesis is to develop an application which will use the capabilities of an Android device to advance the current way of locating stray dogs. By using the final application, a user will be able to make a post about a stray dog that he located by declaring its characteristics and the area he found it. During the publish the latitude and longitude of the current location will be computed. Therefore, another user who might have lost it might use the application to look for the dog by specifying the area it got lost and come in contact by sharing their information within the app.

At first, there will be a short introduction to the Android platform where some basic aspects will be explained about supporting many different Android versions in a better way.

Subsequently, the libraries that helped me in the completion of the implementation will be outlined.

Additionally, we will refer extensively to the MVP pattern and to real code samples I used in my application.

Afterwards, it will be fully understood what Firebase offers, how it works and we will explain how data can be read and get written to the database.

Finally, there will be a description of how the location can be tracked in an Android device explaining how each different technology works. The application's development will be described using a combination of these technologies for better results.

The thesis will be finished by referring to some conclusions that arise during the development and possible future extensions that can be made.

ΠΕΡΙΕΧΟΜΕΝΑ

Ευρετήριο Πινάκων

Ευρετήριο Εικόνων

Ευρετήριο Διαγραμμάτων

Ευρετήριο Κώδικα

ΕΙΣΑΓΩΓΗ

Ως πτυχιακή εργασία ανέλαβα τη σχεδίαση και ανάπτυξη μιας Android εφαρμογής η οποία διευκολύνει τον εντοπισμό αδέσποτων σκυλιών. Ο κύριος σκοπός είναι να αξιοποιηθούν οι δυνατότητες των κινητών τηλεφώνων καθώς και το γεγονός ότι πλέον ο καθένας έχει smartphone.

Οι χρήστες μπορούν να δουν τις καταχωρήσεις άλλων χρηστών όπου κάθε μία περιλαμβάνει τα χαρακτηριστικά ενός σκυλιού, την περιγραφή του, τον νομό και την περιοχή που βρέθηκε και μια φωτογραφία του ζώου. Επιπλέον, χάρη στη λειτουργία του αυτόματου εντοπισμού της τοποθεσίας του χρήστη κατά τη διάρκεια της δημοσίευσης, καταγράφεται το γεωγραφικό πλάτος και μήκος στη βάση δεδομένων με όλα τα υπόλοιπα χαρακτηριστικά ενός ζώου. Επομένως, ένας άλλος χρήστης που πιθανόν να έχει χάσει τον σκύλο του μπορεί να αναζητήσει τις δημοσιεύσεις ανά περιοχή. Επιπλέον, πατώντας το σύμβολο της τοποθεσίας μπορεί να δει που ακριβώς εντοπίστηκε ο σκύλος στον χάρτη. Οι δύο αυτοί χρήστες μπορούν να έρθουν σε επαφή έπειτα από αίτημα σύνδεσης του ενός στον άλλον.

Ο κάθε χρήστης μπορεί να συμπληρώσει το προφίλ του μαζί με μία φωτογραφία, να δει τις καταχωρήσεις που έχει κάνει ο ίδιος καθώς και να τις διαγράψει από την εφαρμογή.

Επιπλέον, με τους χρήστες τους οποίους έχει συνδεθεί μπορεί να δει τα στοιχεία τους από ξεχωριστή καρτέλα.

Κατά την διάρκεια της υλοποίησης μεγάλη έμφαση δόθηκε στο design ώστε να ακολουθεί τις αρχές σχεδίασης του Material Design και να είναι φιλικό και ευχάριστο προς τον χρήστη.

Για την ανάπτυξη της εφαρμογής χρησιμοποιήθηκε το Android Studio. Για τη δημιουργία, διαχείριση της βάσης, παροχή αποθηκευτικού χώρου των φωτογραφιών των χρηστών, καθώς για τις υπηρεσίες αυθεντικοποίησης τους χρησιμοποιήθηκε το Firebase. Το Firebase είναι μια πλατφόρμα που συνεργάζεται με το Android Studio. Επίσης, το εικονίδιο της εφαρμογής καθώς και οι εικόνες στην εγγραφή και σύνδεση χρήστη σχεδιάστηκαν με το Adobe Photoshop CC.

ΚΕΦΑΛΑΙΟ 1
ΕΙΣΑΓΩΓΗ ΣΤΟ ΛΕΙΤΟΥΡΓΙΚΟ ΣΥΣΤΗΜΑ ANDROID

ΕΙΣΑΓΩΓΗ
Σε αυτό το κεφάλαιο θα αναφερθούμε στο λειτουργικό σύστημα Android κάνοντας μια εισαγωγή στην πλατφόρμα Android καθώς και στην εμφάνιση και εξέλιξη του σε βάθος χρόνου. Στη συνέχεια, θα δούμε τον λόγο που η εφαρμογή αναπτύχθηκε σε Android και τα πλεονεκτήματα αυτής της επιλογής. Επιπλέον, θα γίνει αναφορά στις εκδόσεις Android γενικά αλλά και πιο ειδικά όσον αφορά το εύρος των εκδόσεων που υποστηρίζει η εφαρμογή μας.

1.1 ΠΕΡΙΓΡΑΦΗ ΤΟΥ ANDROID
Το Android είναι ένα λειτουργικό σύστημα βασισμένο στο Linux, που αναπτύχθηκε από την Google για συσκευές όπως smartphones, tablets και πολλές άλλες διαφορετικές συσκευές από ποικίλους κατασκευαστές. Το Android πέρα από λειτουργικό είναι και μια πλατφόρμα προγραμματισμού, που περιέχει εργαλεία ανάπτυξης για συγγραφή κώδικα προορισμένο για χρήστες του Android. Η πλατφόρμα αυτή λοιπόν, δημιουργήθηκε από μια οργάνωση γνωστή ως Open Handset Alliance (OHA) που σκοπός τους ήταν να χτίσουν μια καλύτερη συσκευή κινητής τηλεφωνίας. Στην πραγματικότητα όμως, το Android δεν προοριζόταν εξαρχής για κινητές συσκευές όπως θα δούμε στη συνέχεια.

1.2 ΙΣΤΟΡΙΚΗ ΑΝΑΔΡΟΜΗ
Η πρώτη προσέγγιση για το Android γεννήθηκε από την εταιρεία Android Inc που ιδρύθηκε στην Καλιφόρνια από τους Rich Miner, Nick Sears, Chris White και Andy Rubin. Πρόσφατα, έγινε γνωστό ότι αρχικά το Android δεν προοριζόταν για λειτουργικό σύστημα για κινητά αλλά ο αρχικός του στόχος ήταν να βελτιώσει τα λειτουργικά συστήματα των ψηφιακών φωτογραφικών μηχανών. Στη συνέχεια, τα σχέδια άλλαξαν λόγω της μείωσης αγοράς ψηφιακών φωτογραφικών μηχανών και ακριβώς η ίδια πλατφόρμα με το ακριβώς ίδιο λειτουργικό σύστημα έγινε το σημερινό Android για κινητά.

Το 2005, η αρχική εταιρεία εξαγοράστηκε από την Google αλλά ορισμένα μέλη παρέμειναν για να συνεχίσουν την ανάπτυξη του λειτουργικού συστήματος. Τότε, αποφασίστηκε να χρησιμοποιηθεί το Linux ως βάση και αυτό σήμαινε ότι το Android θα μπορούσε να προσφερθεί δωρεάν σε τρίτους κατασκευαστές κινητών τηλεφώνων.

Το 2007, η Google εργαζόταν κρυφά για την ανάπτυξη του Android και προς το τέλος τους έτους άρχισε σιγά σιγά να αποκαλύπτει τα σχέδια της για την καταπολέμηση της Apple και άλλων εταιρειών. Εκείνη την περίοδο ξεκίνησε να χρησιμοποιεί και το όνομα Open Handset Alliance το οποίο περιλάμβανε κατασκευαστές τηλεφώνων όπως HTC και Motorola, κατασκευαστές chip όπως Qualcomm και Texas Instruments και φορείς όπως η T-Mobile.

Τον Νοέμβριο του 2007 η εταιρεία λάνσαρε την beta έκδοση Android 1.0 που προοριζόταν για προγραμματιστές και το 2008 ανακοινώθηκε το πρώτο Android smartphone με όνομα T-Mobile G1. Βγήκε στην αγορά τον Οκτώβριο στα μέσα της ίδια χρονιάς. Άξιο προσοχής είναι ότι όλες οι εκδόσεις Android έχουν ονόματα γλυκών αλλά η πρώτη έκδοση 1.0 είναι η μόνη που δεν ακολούθησε αντίστοιχο στυλ.

Ο ερχομός του Nexus One τον Ιανουάριο του 2010 ήταν από τις πρώτες επιτυχίες των συσκευών που ενδυνάμωσαν την ανάπτυξη του Android και αύξησαν την αντιπαλότητα μεταξύ της Apple και της Google. Μέχρι το τέλος της χρονιάς το Android είχε εδραιωθεί ως λειτουργικό σύστημα που δεν μπορούσε πλέον να περάσει απαρατήρητο.

Η εταιρεία ξεκίνησε με μία συσκευή και σήμερα οι συσκευές Android είναι πλέον διαθέσιμες σχεδόν σε όλες τις αγορές του πλανήτη και όχι μόνο για κινητά τηλέφωνα.

1.3 ΓΙΑΤΙ ΕΠΕΛΕΞΑ ΤΟ ANDROID

Το Android είναι η πιο ευρέως διαδεδομένη πλατφόρμα κινητών συσκευών με εκατομμύρια χρήστες σε περισσότερες από 190 χώρες ανά τον κόσμο. Έχει παρουσιάσει συντριπτική άνοδο στην κορυφή μέσα στη σύντομη διάρκεια ζωής του και κατέχει τα πρωτεία σε αριθμό ενεργών χρηστών στην αγορά των smartphone.Το μερίδιο του στην αγορά κυμαίνεται γύρω στο 80% σε παγκόσμιο επίπεδο και συνεχίζει να αναπτύσσεται και να προσελκύει εκατομμύρια χρήστες καθημερινά. Μια τέτοια ταχεία ανάπτυξη έχει βοηθήσει το Android να φτάσει στο εντυπωσιακό ορόσημο του ενός δισεκατομμυρίου συσκευών που έχουν ενεργοποιηθεί μέχρι στιγμής. Κάθε προγραμματιστής που θέλει να μεγιστοποιήσει τους πιθανούς τελικούς χρήστες του ανεξαρτήτως γλώσσας ή χώρας καλό θα ήταν επιλέξει το Android. Τα πλεονεκτήματα του Android είναι τα εξής:

- Ανοιχτό λογισμικό
- Ευελιξία
- Χαμηλό κόστος δημοσίευσης της εφαρμογής

Το λειτουργικό σύστημα του Android είναι σχεδιασμένο να είναι δωρεάν και ανοιχτού κώδικα. Αυτό το καθιστά ιδιαίτερα ευέλικτο λόγω του ότι οποιοσδήποτε προγραμματιστής μπορεί να προσαρμόσει διάφορα χαρακτηριστικά και λειτουργίες του Android. Ουσιαστικά το γεγονός ότι είναι ανοιχτού λογισμικού σημαίνει ότι προσφέρει ευελιξία.

Αν ένας προγραμματιστής ενδιαφέρεται να δημοσιεύσει την εφαρμογή του, χρειάζεται να πληρώσει τα τέλη εγγραφής αφού πρώτα κάνει έναν λογαριασμό ως developer. Το ποσό είναι 20 ευρώ και είναι εφάπαξ αντίθετα με το App store του λειτουργικού iOS που το ποσό είναι 100 ευρώ και πρέπει να πληρώνεις κάθε χρόνο.

Όπως φαίνεται λοιπόν, επιλέγοντας το Android όχι μόνο θα προσεγγίσεις πολύ μεγαλύτερο κοινό αλλά θα επενδύσεις και λιγότερα χρήματα.

1.4 ΟΙ ΕΚΔΟΣΕΙΣ ANDROID

Όπως αναφέρθηκε και προηγουμένως, οι εκδόσεις του Android έχουν κωδική ονομασία κάποιο γλυκό ενώ ταυτόχρονα τα αρχικά των εκδόσεων τους είναι σε αλφαβητική σειρά. Συν τοις άλλοις, σε κάθε έκδοση Android ανατίθεται ένας μοναδικός αριθμός που είναι το αναγνωριστικό του και ονομάζεται επίπεδο API. Συνεπώς, κάθε έκδοση Android αντιστοιχεί σε ένα επίπεδο API. Λόγω του ότι οι χρήστες Android δεν χρησιμοποιούν μόνο καινούργιες εκδόσεις αλλά και παλαιότερες, οι εφαρμογές πρέπει να σχεδιάζονται με τέτοιο τρόπο ώστε να λειτουργούν σε διάφορα επίπεδα API. Η κάθε συσκευή Android λειτουργεί χρησιμοποιώντας ένα επίπεδο API που αναγνωρίζει επακριβώς την έκδοση της συλλογής API που θα κληθεί να χρησιμοποιήσει η εφαρμογή. Αυτό είναι εφικτό λαμβάνοντας υπόψιν τον συνδυασμό από στοιχεία του manifest, άδειες και άλλα στοιχεία στα οποία βασίζεται η λειτουργία της εφαρμογής. Το σύστημα των API καθορίζει αν μία εφαρμογή είναι συμβατή με μια συσκευή Android προτού γίνει η εγκατάσταση σε αυτή. Όταν αναπτύσσουμε μία εφαρμογή θα πρέπει ως προγραμματιστές να συμπεριλάβουμε τις εξής πληροφορίες:

- Το target επίπεδο API στο οποίο η εφαρμογή είναι ικανή να εκτελεστεί στο 100% μιας και εδώ βάζουμε συνήθως το API που έχουμε χρησιμοποιήσει για την δοκιμή της εφαρμογής

- Το ελάχιστο επίπεδο API το οποίο πρέπει να έχει μια συσκευή προκειμένου να μπορεί να τρέξει η εφαρμογή επιτυχώς

Επομένως, σε περίπτωση που το επίπεδο API μιας συσκευής είναι χαμηλότερο από το ελάχιστο επίπεδο API που τέθηκε στην εφαρμογή τότε η ίδια η συσκευή θα αποτρέψει τον χρήστη από το κατέβασμα της εφαρμογής. Σε περίπτωση που βάλουμε ως target API κάποιο API μικρότερο από αυτό που απαιτούν οι βιβλιοθήκες μας, αν κληθεί η αντίστοιχη βιβλιοθήκη τότε θα προκληθεί σφάλμα μεταγλώττισης. Ορισμένες εκδόσεις Android απαιτούν διαφορετικό κώδικα για την χρήση ορισμένων κλάσεων επομένως μπορούμε να περιλαμβάνουμε ελέγχους κατά την διάρκεια της εκτέλεσης. Έτσι, αναλόγως με την έκδοση του Android που χρησιμοποιείται εκτελείται ο κατάλληλος κώδικας. Αυτοί οι έλεγχοι είναι γνωστοί ως Runtime Checks και καθιστούν μια εφαρμογή πλήρως λειτουργική σε διαφορετικές εκδόσεις Android. Θεωρούνται αναπόσπαστο κομμάτι σε περίπτωση που θέλουμε η εφαρμογή μας να υποστηρίζει πλήρως τις αναβαθμίσεις στις εκδόσεις Android.

1.5 ΥΠΟΣΤΗΡΙΖΟΜΕΝΑ ΕΠΙΠΕΔΑ API

Λόγω των απαιτήσεων των βιβλιοθηκών που χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής η εφαρμογή υποστηρίζει ως ελάχιστο επίπεδο API το 14 και ως μέγιστο το 24. Αυτό συμβαίνει διότι αρκετές από τις βιβλιοθήκες που χρησιμοποιήθηκαν απαιτούσαν ως ελάχιστο API το 14 προκειμένου να

λειτουργήσουν. Τα dependencies των βιβλιοθηκών αυτών είναι τα ακόλουθα όπως θα δούμε και αναλυτικότερα στο επόμενο κεφάλαιο:

- 'com.google.firebase:firebase-storage:11.0.2'
- 'com.google.firebase:firebase-auth:11.0.2'
- 'com.google.firebase:firebase-database:11.0.2'
- com.google.android.gms:play-services:11.0.2
- 'com.google.android.gms:play-services-location:11.0.2'
- 'com.google.android.gms:play-services-maps:11.0.2'
- 'com.intuit.sdp:sdp-android:1.0.3'
- 'com.squareup.picasso:picasso:2.5.2'
- 'com.github.bumptech.glide:glide:3.7.0'

ΕΠΙΛΟΓΟΣ

Στο κεφάλαιο αυτό, εξηγήσαμε τι είναι το Android αναφέροντας σημαντικά χαρακτηριστικά του. Τα χαρακτηριστικά που αναφέρθηκαν αποτέλεσαν τον λόγο της επιλογής του ως την πλατφόρμα προγραμματισμού αυτής της πτυχιακής. Είδαμε πως το Android ξεκίνησε με δειλά βήματα και έφτασε σήμερα να είναι ένας κολοσσός στην κινητή τηλεφωνία. Επιπλέον, αναφέραμε τον τρόπο με τον οποίο μπορεί να γίνει καλύτερη η υποστήριξη των εκδόσεων Android και τέλος τις εκδόσεις που υποστηρίζει η εφαρμογή μας και γιατί.

ΚΕΦΑΛΑΙΟ 2
ΒΙΒΛΙΟΘΗΚΕΣ ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΑΝ

ΕΙΣΑΓΩΓΗ
Σε αυτό το κεφάλαιο θα αναφερθούν όλες οι βιβλιοθήκες που χρησιμοποιήθηκαν καθώς και η χρησιμότητα και ο ρόλος τους στην εφαρμογή. Αρχικά, θα επεξηγηθούν οι βοηθητικές βιβλιοθήκες που παρέχει το ίδιο το Android και στη συνέχεια οι υπόλοιπες από τρίτους ανεξάρτητους παρόχους.

2.1 ΒΟΗΘΗΤΙΚΕΣ ΒΙΒΛΙΟΘΗΚΕΣ
Όταν προγραμματίζεις μια εφαρμογή σε μία καινούργια έκδοση Android επιθυμείς να μπορεί να εκτελεστεί και να «τρέξει» επιτυχώς και σε παλαιότερες εκδόσεις. Αυτό είναι εφικτό εφόσον προσθέσεις κώδικα που εξαλείφει το χάσμα ασυμβατότητας. Γι' αυτόν ακριβώς τον λόγο δημιουργήθηκαν οι βοηθητικές βιβλιοθήκες Android (Android Support Libraries). Αναλυτικότερα, μια βοηθητική βιβλιοθήκη είναι ένα σύνολο από βιβλιοθήκες με κώδικα που παρέχουν εκδόσεις από framework APIs με συμβατότητα προς τα πίσω. Η κάθε μία από αυτές είναι συμβατή προς τα πίσω μέχρι ένα συγκεκριμένο επίπεδο API.

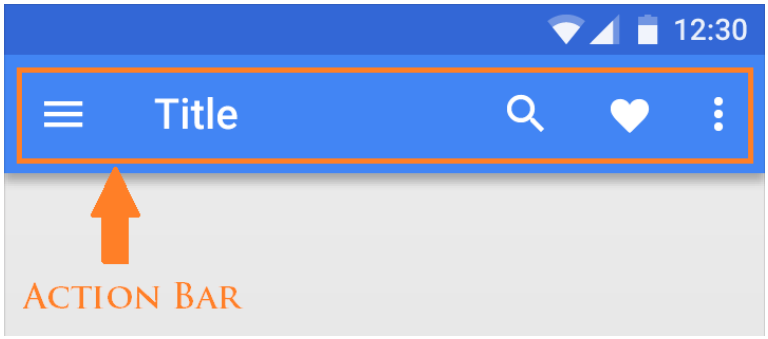
Η ενσωμάτωση αυτών των βιβλιοθηκών στο project θεωρείται μία πολύ καλή πρακτική που συνιστάται, ανάλογα με το εύρος των εκδόσεων που στοχεύει να καλύψει και το target API της εφαρμογής. Υπάρχουν διάφορες βιβλιοθήκες που μπορούν να συμπεριληφθούν όπου η κάθε μία υποστηρίζει ένα συγκεκριμένο εύρος εκδόσεων και χαρακτηριστικών. Παρακάτω μπορούμε να δούμε μία σύντομη περιγραφή αυτών των βιβλιοθηκών και το API που υποστηρίζουν.

Πίνακας 1 "Support Libraries"

Support library version	Features
v4 Support Library	Αυτή η βιβλιοθήκη είναι σχεδιασμένη να λειτουργεί για Android 1.6 (API level 4) και πάνω.
v7 Support Library	Περιέχει πολλές βιβλιοθήκες σχεδιασμένες για Android 2.1 (API level 7) και πάνω.
v8 Support Library	Αυτή η βιβλιοθήκη είναι σχεδιασμένη για Android (API level 8) και πάνω.
v13 Support Library	Αυτή η βιβλιοθήκη είναι σχεδιασμένη για Android 3.2 (API level 13) και πάνω.

Η βιβλιοθήκη v4 είναι η πρώτη βιβλιοθήκη που δημιουργήθηκε και υποστηρίζει υλοποιήσεις για σημαντικές κλάσεις όπως η Fragment -που θα αναλυθεί περαιτέρω σε επόμενο κεφάλαιο- και ο Loader.

Η βιβλιοθήκη v7 -που στον κώδικα αναφέρεται ως v7-appcompat παρέχει υποστήριξη για υλοποιήσεις του ActionBar και του Toolbar που είναι κύριο στοιχείο στην εφαρμογή μας. Αυτή η βιβλιοθήκη χρειάζεται την v4 αλλά δεν την περιέχει. Επομένως, οποιοδήποτε χαρακτηριστικό εξαρτάται από την v7-appcompat εξαρτάται και από την v4.



Εικόνα 1 “Action Bar”

Πέρα από αυτές τις δύο, στην εφαρμογή συμπεριέλαβα και ορισμένες βοηθητικές βιβλιοθήκες για την δημιουργία των γραφικών, μία από αυτές είναι η v7-recyclerview. Αυτή η βιβλιοθήκη παρέχει το στοιχείο RecyclerView το οποίο χρησιμοποιείται κατά κόρον στην εφαρμογή και είναι υπεύθυνο για την τοποθέτηση και εμφάνιση μεγάλου αριθμό δεδομένων μέσα σε αντικείμενα γραφικών. Επιπλέον, χρησιμοποιήθηκε και η v7-cardview που παρέχει το στοιχείο CardView ενεργοποιώντας την χρήση στοιχείων που μοιάζουν με κάρτες.

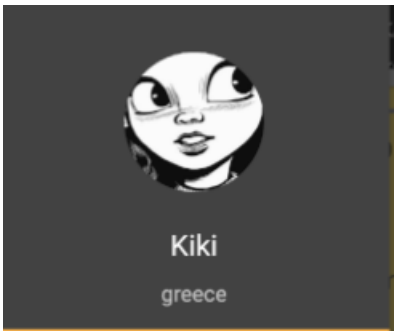
2.2 ΧΡΗΣΙΜΕΣ ΒΙΒΛΙΟΘΗΚΕΣ ΑΛΛΩΝ ΠΑΡΟΧΩΝ

Πέρα από τις βοηθητικές βιβλιοθήκες που μας παρέχει η ίδια η Google για την δυνατότητα υποστήριξης καινούργιων δυνατοτήτων και στοιχείων σε παλαιότερες εκδόσεις Android, υπάρχουν και άλλες βιβλιοθήκες που ήταν αναγκαίες προκειμένου να έχουμε το επιθυμητό αποτέλεσμα. Η πιο χρήσιμη βιβλιοθήκη στην εφαρμογή μας ήταν η βιβλιοθήκη του Firebase που είναι μια υπηρεσία backend για κινητά που παρέχει χρήσιμα και αναγκαία εργαλεία και υλοποιήσεις για να χτιστεί μια πλήρης εφαρμογή για κινητά. Για να υλοποιήσω το κομμάτι του Firebase λοιπόν, χρειάστηκε να εισάγω τα παρακάτω dependencies στο Grable build script. Το κάθε dependency παρέχει την κατάλληλη υπηρεσία όπως βλέπουμε στον πίνακα παρακάτω. Στη συνέχεια, θα δούμε το Firebase αποκλειστικά σε ολόκληρη ενότητα.

```
compile 'com.google.firebase:firebase-storage:11.0.2'
compile 'com.google.firebase:firebase-auth:11.0.2'
compile 'com.google.firebase:firebase-core:11.0.2'
compile 'com.google.firebase:firebase-database:11.0.2'
```

Εικόνα 2 “Dependencies του Firebase”

Στην εφαρμογή σε αρκετά σημεία χρησιμοποιείται το στοιχείο CircleImageView που βρίσκεται στην εικόνα 3. Θα μπορούσαμε να χρησιμοποιήσουμε κώδικα για να κόψουμε την εικόνα και να την διαμορφώσουμε ώστε να έχει αυτό το σχήμα προγραμματιστικά. Η πλατφόρμα Android δεν μας παρέχει ατόφιο το στοιχείο αυτό αλλά ένα άλλο στοιχείο το ImageView που δεν στρογγυλεύει τις εικόνες. Επομένως ή θα διαμορφώσουμε μόνοι μας την εικόνα ή θα χρησιμοποιήσουμε κάποια βιβλιοθήκη.



Εικόνα 3 “CircleImageView”

Πίνακας 2 “Υπηρεσίες του Firebase”

Υπηρεσία	Gradle dependency
Cloud Storage	com.google.firebase:firebase-storage:11.0.2
Authentication	com.google.firebase:firebase-auth:11.0.2
Firebase Core	com.google.firebase:firebase-core:11.0.2
Realtime Database	com.google.firebase:firebase-database:11.0.1

Προσθέτοντας το dependency 'de.hdodenhof:circleimageview:2.1.0' έχουμε στη διάθεση μας ένα στρογγυλοποιημένο ImageView. Δύο ακόμα βιβλιοθήκες σχετικές με γραφικά είναι η Glide και η Picasso. Αυτές οι δύο βιβλιοθήκες είναι οι πιο πολυχρησιμοποιημένες βιβλιοθήκες στον κόσμο των εφαρμογών Android. Παρόλο που και οι δύο χρησιμοποιούνται για τον ίδιο σκοπό, ο τρόπος με τον οποίο κατεβάζουν τις εικόνες από το διαδίκτυο, διαχειρίζονται τη μνήμη cache και φορτώνουν τις εικόνες είναι λίγο διαφορετικός. Η Glide χρησιμοποιεί λιγότερη μνήμη διότι αφού κατεβάσει την εικόνα από ένα URL την αλλάζει μέγεθος σύμφωνα με το μέγεθος που απαιτεί το CircleImageView και την αποθηκεύει στη μνήμη cache.

Όσον αφορά τα γραφικά χρησιμοποιήθηκε μία ακόμα βιβλιοθήκη που σκοπός της είναι να παρέχει μια άλλη μονάδα μέτρησης του ύψους και του πλάτους. Η μονάδα μέτρησης αυτή ονομάζεται sdp και έτσι ονομάζεται και η βιβλιοθήκη. Η πρωταρχική μονάδα μέτρησης τα DP(density independent pixels), είναι ο αριθμός των pixel που αναπαριστάνονται σε μία μονάδα dp και αυξάνεται όσο αυξάνεται η ανάλυση της

οθόνης. Η βιβλιοθήκη παρέχει μια καινούργια μονάδα μεγέθους, το sdp (scalable dp). Έτσι, τα γραφικά μεγαλώνουν ή μικραίνουν αναλόγως με το μέγεθος της οθόνης. Αυτό γίνεται προκειμένου να μην υπάρχουν περιπτώσεις όπου τα γραφικά που σχεδιάστηκαν σε μία μικρή οθόνη κινητού παραμείνουν σε μικρό μέγεθος όταν χρησιμοποιηθούν σε μεγαλύτερες οθόνες.

ΕΠΙΛΟΓΟΣ

Στο κεφάλαιο αυτό, είδαμε ποιες βιβλιοθήκες χρησιμοποιήθηκαν στο Android project και επεξηγήθηκε εν συντομία η χρήση της κάθε μίας. Οι βιβλιοθήκες αφορούσαν και τις βοηθητικές αλλά και τις βιβλιοθήκες τρίτων.

ΚΕΦΑΛΑΙΟ 3
ΑΝΑΛΥΣΗ ΤΟΥ MATERIAL DESIGN ΤΗΣ ΕΦΑΡΜΟΓΗΣ

ΕΙΣΑΓΩΓΗ
Σε αυτό το κεφάλαιο θα αναφερθούμε περιληπτικά στο Material design και έπειτα θα δούμε αναλυτικά τα συστατικά μέρη της εφαρμογής που απαρτίζουν το γραφικό περιβάλλον. Συγκεκριμένα, θα δούμε τα xml αρχεία και το περιεχόμενο τους.

3.1 ΛΙΓΑ ΛΟΓΙΑ ΓΙΑ ΤΟ MATERIAL DESIGN
Το Material design είναι ένα σύνολο σχεδιαστικών κανόνων που επιτρέπουν στα γραφικά να έχουν κίνηση και μεταβάσεις οι οποίες ανταποκρίνονται ποικιλοτρόπως ανάλογα τις ενέργειες των χρηστών. Επιπλέον, προσφέρει διάφορα εφέ όπως εφέ βάθους και σκίαση των αντικειμένων. Το Material έχει φυσικές επιφάνειες και άκρα και οι σκιές δίνουν νόημα σε αυτό που αγγίζεις. Το design αυτό μπορεί να χρησιμοποιηθεί από την έκδοση Android 2.1 και πάνω μέσω της βιβλιοθήκης v7 appcompat και χρησιμοποιείται από την πλειοψηφία των συσκευών Android που φτιάχτηκαν από το 2009 και μετά. Η εφαρμογή των κανόνων του Material design για διεπαφές χρήστη σε web εφαρμογές ονομάζεται polymer paper elements. Οι τρεις βασικές αρχές του είναι οι εξής:

1. Ρεαλιστικά γραφικά: Το design στηρίζεται στην πραγματικότητα όντας εμπνευσμένο από τη μελέτη του χαρτιού και του μελανιού. Χρησιμοποιώντας τις επιφάνειες, τις άκρες και τις βασικές αρχές του φωτός και της κίνησης γίνεται πιο αντιληπτός στον χρήστη ο τρόπος με τον οποίο τα αντικείμενα αλληλοεπιδρούν, κινούνται και σχετίζονται μεταξύ τους.

2. Έντονα γραφικά με σκοπό: Τα πλέγματα, ο χώρος, τα μεγέθη των γραφικών και οι εικόνες καθοδηγούν όλο το σχέδιο. Οι επιλογές των χρωμάτων είναι τολμηρές και σκόπιμες.

3. Η κίνηση δίνει νόημα: Η κίνηση είναι βασικό χαρακτηριστικό του Material design και πρέπει να γίνεται με τέτοιο τρόπο που να τραβά την προσοχή του χρήστη περιλαμβάνοντας απλές μεταβάσεις.

3.2 ΣΥΣΤΑΤΙΚΑ ΤΟΥ MATERIAL DESIGN ΣΤΗΝ ΕΦΑΡΜΟΓΗ ΜΑΣ

Για να δημιουργήσουμε γραφικά στο Android θα πρέπει να δηλώσουμε το κάθε στοιχείο ξεχωριστά σε ένα xml αρχείο. Ένας άλλος τρόπος να δημιουργήσουμε γραφικά είναι με δυναμικό τρόπο γράφοντας κώδικα μέσα σε κάποιο Activity ή Fragment. Αρκετά συχνά, χρησιμοποιούνται και οι δύο τρόποι παράλληλα. Το xml αρχείο με τα περισσότερα συστατικά είναι το xml activity_main και ακολουθεί τις αρχές του Material design. Είναι το xml που καθορίζει τα στοιχεία που είναι σταθερά σχεδόν σε όλη την εφαρμογή ανεξαρτήτως αν ο χρήστης μεταβεί σε κάποιο άλλο Activity ή Fragment. Η κύρια διάταξη της εφαρμογής activity_main είναι ο Κώδικας 1.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.DrawerLayout

xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:clickable="true"
    android:focusableInTouchMode="true"
    android:orientation="vertical">

    <include
        layout="@layout/app_bar_profile"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        />

    <android.support.design.widget.NavigationView
        android:id="@+id/nav_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        app:itemTextColor="@color/colorNavText"
        app:itemIconTint="@color/colorPrimaryDark"
        app:headerLayout="@layout/nav_header_profile"
        app:menu="@menu/activity_profile_drawer2"
        android:background="@color/colorShop" />

</android.support.v4.widget.DrawerLayout>
```

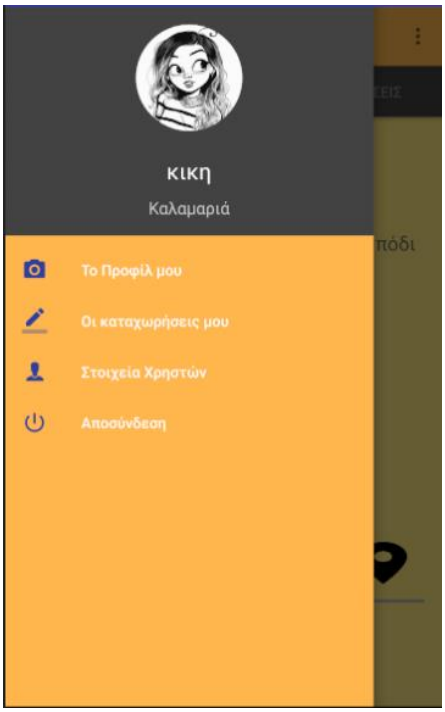
Κώδικας 1 "Η κύρια διάταξη της εφαρμογής"

Αρχικά στον Κώδικα 1, δηλώνουμε το Layout που θα χρησιμοποιήσουμε, επιλέγουμε το DrawerLayout. Είναι ένα container υψηλού επιπέδου στο οποίο τοποθετούμε περιεχόμενο το οποίο είναι διαδραστικό και ο χρήστης μπορεί να το εμφανίσει αν το τραβήξει από δεξιά ή από αριστερά ή αν πατήσει ένα κουμπί (Toggle Button) πάνω στο Action bar. Το DrawerLayout είναι μονόδρομος εφόσον θέλουμε να δημιουργήσουμε ένα Navigation Drawer. Στην εικόνα 3 και 5 βλέπουμε το Navigation Drawer της εφαρμογής μας.

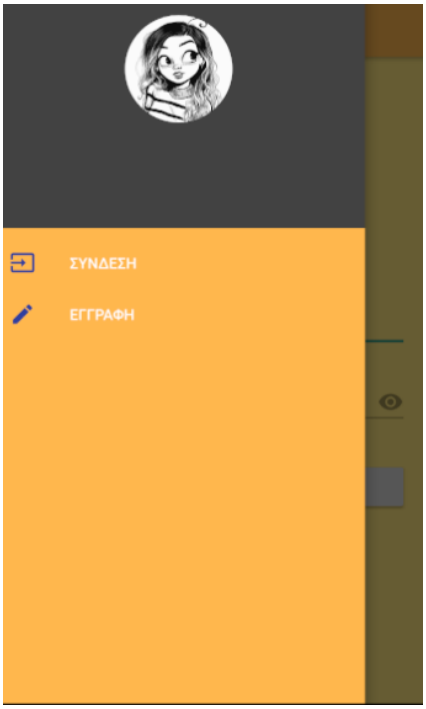
Προκειμένου να δημιουργήσουμε ένα Navigation Drawer θα πρέπει να εισάγουμε στο Build Gradle script τα παρακάτω dependencies:

```
compile 'com.android.support:appcompat-v7:24.2.1'
compile 'com.android.support:design:24.2.1'
```

Κώδικας 2 “Dependencies για το Navigation Drawer”



Εικόνα 4 “Το Navigation Drawer στους συνδεδεμένους Χρήστες”



Εικόνα 5 “Το Navigation Drawer στους μη συνδεδεμένους χρήστες”

Με το include συμπεριλαμβάνουμε το xml app_bar_profile το οποίο βλέπουμε στον Κώδικα 3. Με τη βοήθεια ενός CoordinatorLayout τα διάφορα συστατικά μέρη το οποία έχει ως «παιδιά» συνεργάζονται μεταξύ τους προκειμένου να υλοποιήσουν εντυπωσιακά animations. Χρησιμοποιείται κατά κόρον στο Material design λόγω του ότι τα animation και οι μεταβάσεις που δημιουργεί καθιστούν την πλοήγηση του χρήστη καλύτερη. Στον Κώδικα 3 βλέπουμε ότι υπάρχει ένα Toolbar με την ιδιότητα layout_scrollFlags. Σε αυτή την ιδιότητα έχουμε βάλει το flag “scroll” που σημαίνει ότι αναγνωρίζει πότε ο χρήστης θα κάνει scroll. Το flag αυτό λειτουργεί πάντα σε συνδυασμό με άλλες τιμές προκειμένου να δημιουργηθεί ένα εφέ κίνησης. Η τιμή “enterAlways” ορίζει ότι με κάθε κίνηση swipe του χρήστη από πάνω προς τα κάτω το Toolbar θα εμφανιστεί. Με την επιλογή “snap” καθορίζουμε τι θα γίνει στην περίπτωση που το Toolbar ή όποιο άλλο συστατικό μέρος έχουμε ορίσει, εξαφανιστεί μερικώς, δηλαδή είναι εμφανές αλλά δεν έχει το 100% του αρχικού του

μεγέθους. Αναλυτικότερα, σημαίνει ότι σε περίπτωση που ο χρήστης τραβήξει το αντικείμενο και το μειώσει λιγότερο από το 50% του μεγέθους του τότε επιστρέφει ξανά στο κανονικό του μέγεθος. Βάλαμε flags και στο Toolbar αλλά και στο TabLayout γιατί θέλουμε να κρύβονται και τα δύο αντικείμενα όταν ο χρήστης βλέπει τις δημοσιεύσεις ώστε να μην πιάνουν χώρο στο Activity.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
     android:layout_height="match_parent"
     android:layout_width="match_parent"
     android:orientation="vertical"
     android:fitsSystemWindows="true">

<android.support.design.widget.AppBarLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:elevation="@dimen/_8sdp"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="@color/colorShop"
        app:popupTheme="@style/Widget.AppCompat.ActionBar"
        xmlns:android="http://schemas.android.com/apk/res/android"
        app:layout_scrollFlags="scroll|enterAlways|snap">
    </android.support.v7.widget.Toolbar>

    <android.support.design.widget.TabLayout
        android:id="@+id/tabs"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/toolbar"
        app:tabSelectedTextColor="@color/colorWhite"
        app:tabTextColor="@color/wallet_hint_foreground_holo_dark"
        android:background="@color/colorShop2"
        app:layout_scrollFlags="scroll|enterAlways"/>

</android.support.design.widget.AppBarLayout>


    <android.support.v4.view.ViewPager
        android:id="@+id/container"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto" />

    <include layout="@layout/content_posts"/>


</android.support.design.widget.CoordinatorLayout>
```

Κώδικας 3 “Η διάταξη του Toolbar της εφαρμογής”

Άξιο προσοχής είναι ότι για να κάνουμε τη διεπαφή της εφαρμογής να ακολουθεί τις αρχές του Material design θα πρέπει ο Navigation Drawer να εμφανίζεται μπροστά από τη βασική μπάρα της εφαρμογής μας. Για να γίνει αυτό, θα πρέπει να θέσουμε ότι το Toolbar είναι η βασική μας μπάρα. Αυτό δεν είναι κάτι που γίνεται μόνο του μέσω του xml αλλά το ορίζουμε στο MainActivity. Πρώτα απ' όλα, θα πρέπει το activity να κληρονομεί από το AppCompatActivity όπως βλέπουμε στον κώδικα παρακάτω.

```
public class MainActivity extends AppCompatActivity
```

Κώδικας 4 "Τι κληρονομεί το MainActivity"

Στη συνέχεια στον Κώδικα 5, αφού αρχικοποιήσουμε το Toolbar καλούμε τη μέθοδο setSupportActionBar() για να ορίσουμε ότι αυτή θα είναι η βασική μπάρα της εφαρμογής μας. Το Toolbar πολύ συχνά καλείται και ActionBar.

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
setSupportActionBar(toolbar);
```

Κώδικας 5 "Πως ορίζεται το Toolbar"

Για να μπορέσει να ενεργοποιηθεί ο Navigation Drawer και να εμφανίσει το μενού θα πρέπει ο χρήστης να πατήσει στο κουμπί που βλέπουμε στην εικόνα 6, αυτό το κουμπί βρίσκεται πάνω στο ActionBar και υλοποιείται με τον Κώδικα 6.

```
DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer);
ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
        this, drawer, toolbar, R.string.navigation_drawer_open,
R.string.navigation_drawer_close
);
drawer.setDrawerListener(toggle);
toggle.syncState();
```

Κώδικας 6 "Ενεργοποίηση των events του Navigation Drawer"

Εικόνα 6 “Toggle Button”

Έπειτα το συστατικό μέρος ViewPager χρησιμοποιείται προκειμένου να ενσωματώσουμε fragments σε διαφορετικές σελίδες στις οποίες έχουμε πρόσβαση μέσω των Tabs. Ο χρήστης πατώντας τα Tabs μπορεί να μεταβεί σε διαφορετικό Fragment. Επιπροσθέτως, η μετάβαση αυτή μπορεί να συμβεί με σύρσιμο της οθόνης δεξιά ή αριστερά. Η διαδικασία για να γίνει είναι η εξής:

1. Δημιουργούμε ένα αντικείμενο SectionsPagerAdapter το οποίο είναι υπεύθυνο για την παροχή των fragment σε κάθε διαφορετική σελίδα.
2. Αρχικοποιούμε τον ViewPager που δηλώσαμε στο xml
3. Τοποθετούμε τις σελίδες (Fragments) εντός του ViewPager καλώντας τη μέθοδο setupViewPager (βλέπουμε τον κώδικα παρακάτω)
4. Αρχικοποιούμε το TabLayout που είχαμε στο xml και τον τροφοδοτούμε με τον ViewPager που ρυθμίσαμε καλώντας την προκαθορισμένη μέθοδο setupWithViewPager
5. Με την μέθοδο setSelectedTabIndicatorColor μπορούμε να θέσουμε το χρώμα που θέλουμε να έχουν οι τίτλοι των tabs

```
mSectionsPagerAdapter = new SectionsPagerAdapter(getSupportFragmentMan-
ager());

// Set up the ViewPager with the sections adapter.
mViewPager = (ViewPager) findViewById(R.id.container);
setupViewPager(mViewPager);


TabLayout tabLayout = (TabLayout) findViewById(R.id.tabs);
tabLayout.setupWithViewPager(mViewPager);
tabLayout.setSelectedTabIndicatorColor(Color.parseColor("#FFFFFF"));
```

Κώδικας 7 “Ενσωμάτωση των Fragments στο ViewPager και στα Tabs”

```
private void setupViewPager(ViewPager viewpager)
{
    mSectionsPagerAdapter.addFragment(new posts(), "ΑΓΝΟΟΥΜΕΝΑ ΖΩΑ");
    mSectionsPagerAdapter.addFragment(new top_posts(),
    "ΤΟΡ ΚΑΤΑΧΩΡΗΣΕΙΣ");
    viewpager.setAdapter(mSectionsPagerAdapter);

}
```

Κώδικας 8 “Η μέθοδος setupViewPager()”

Το layout content_posts στον Κώδικα 3 είναι ένα FrameLayout στο οποίο θα φορτώσουμε τα fragments που βρίσκονται μέσα στο Navigation Drawer. Τα δύο Fragments που τοποθετήσαμε στο TabLayout θα φορτωθούν εντός του ViewPager οπότε δεν θα μπουν στο ίδιο layout με όλα τα υπόλοιπα.

Το Navigation view που είναι και το τελευταίο View εντός του xml activity_main, είναι κομμάτι του Navigation Drawer μιας και μέσα σε αυτό βάζουμε οτιδήποτε περιέχεται πάνω του. Για να προσδιορίσουμε τις επιλογές του menu που θα εμπεριέχει ο Navigation Drawer βάζουμε στην ιδιότητα app:menu το layout που έχουμε φτιάξει. Τα μενού είναι ξεχωριστά αρχεία xml τα οποία εναλλάσσουμε προγραμματιστικά αναλόγως αν είναι συνδεδεμένος κάποιος χρήστης ή όχι. Στην περίπτωση που είναι, φορτώνεται το μενού activity_profile_drawer1 όπου το κάθε item είναι μία ξεχωριστή επιλογή. Στην περίπτωση που ένας χρήστης δεν είναι συνδεδεμένος βλέπει μόνο τις επιλογές Εγγραφή και Σύνδεση.

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

    <group android:checkableBehavior="single">
        <item
            android:id="@+id/nav_profile"
            android:icon="@drawable/ic_camera_alt_black_24dp"
            android:title="Το Προφίλ μου" />
        <item
            android:id="@+id/nav_found_pets"
            android:icon="@drawable/ic_border_color_black_24dp"
            android:title="@string/menu_entries" />


        <item
            android:id="@+id/nav_settings"
            android:icon="@mipmap/ic_contacts"
            android:title="Στοιχεία Χρηστών" />

        <item
            android:id="@+id/nav_logout"
            android:icon="@drawable/ic_power_settings_new_black_24dp"
            android:title="@string/menu_logout" />


    </group>

</menu>
```

Κώδικας 9 “Το xml του μενού ενός συνδεδεμένου χρήστη”

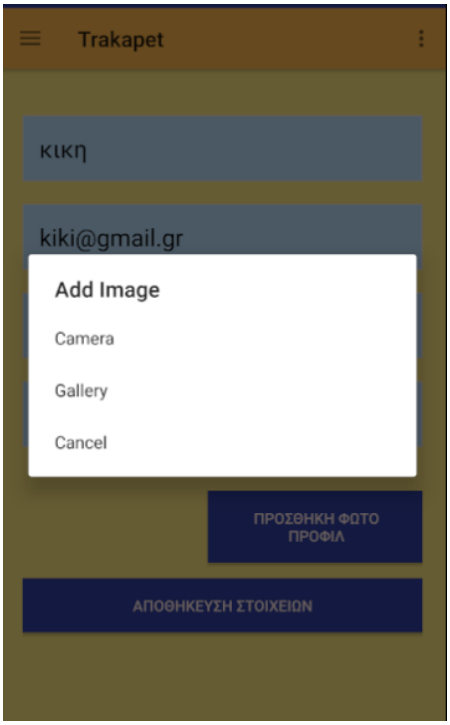
```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">

        <item
            android:id="@+id/nav_login"
            android:icon="@drawable/ic_input_black_24dp"
            android:title="@string/login" />

        <item
            android:id="@+id/nav_signup"
            android:icon="@drawable/ic_create_black_24dp"
            android:title="@string/signup" />

    </group>
</menu>
```

Κώδικας 10 “Το xml του μενού ενός αποσυνδεδεμένου χρήστη”

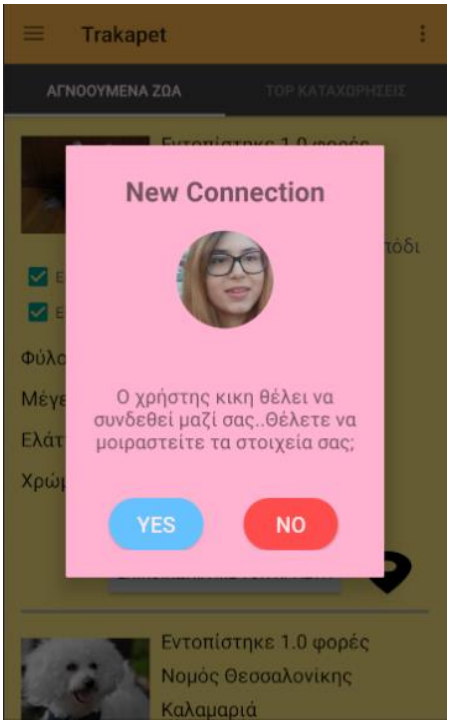
Επιπλέον, ορίσαμε και μία ακόμη ιδιότητα στον Navigation Drawer στον Κώδικα 1 που αφορά το πάνω μέρος του Drawer. Στην ιδιότητα λοιπόν headerLayout βάλαμε ένα layout που είναι ένα xml που θα περιέχει την φωτογραφία προφίλ του χρήστη, το όνομα και την περιοχή που μένει. Στην περίπτωση που ο χρήστης δεν είναι συνδεδεμένος ή δεν έχει συμπληρώσει ακόμα τα στοιχεία του προφίλ του αυτά είναι κενά.

Στην εφαρμογή χρησιμοποιούμε το στοιχείο CardView για να εμφανίσουμε ένα Custom Dialog στον χρήστη. Κάποιες φορές στην εφαρμογή μας χρειάζεται να ζητήσουμε από τον χρήστη να απαντήσει σε μια ερώτηση ή να του ζητήσουμε να κάνει κάτι και περιμένουμε από αυτόν μια απάντηση. Άλλες φορές μπορεί απλά να θέλουμε να τον προειδοποιήσουμε για κάποια ενέργεια του. Για όλες αυτές τις περιπτώσεις συνήθως χρησιμοποιούμε ένα αντικείμενο Alert Dialog. Αν θέλουμε να είναι ένα απλό Alert Dialog μπορούμε να το δημιουργήσουμε προγραμματιστικά χωρίς τη βοήθεια κάποιου xml -όπως αυτό που χρησιμοποιούμε σε ένα σημείο της εφαρμογής μας στην εικόνα 7-. Το Alert που δημιουργήσαμε χρησιμοποιώντας το CardView είναι στην εικόνα 8 και έτσι υποστηρίζει και το Material Design μιας και το CardView αποτελεί κύριο συστατικό του.

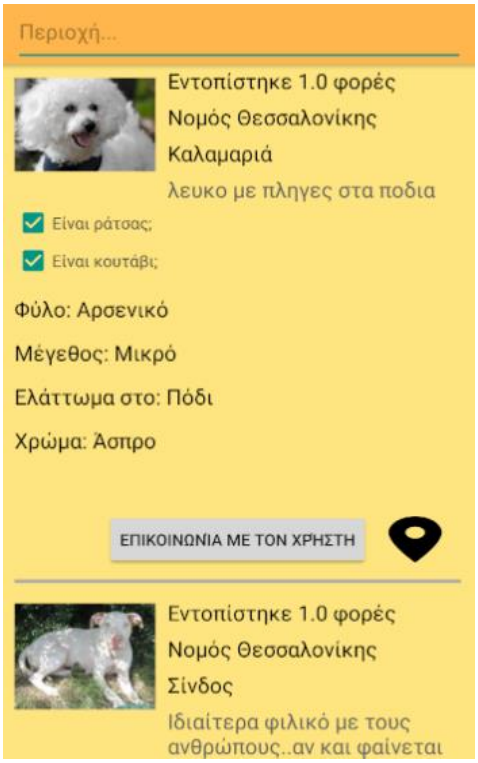
Πολλές λειτουργίες της εφαρμογής χρησιμοποιούν το αντικείμενο RecyclerView το οποίο είναι μια αναβάθμιση του ListView που χρησιμοποιούνταν πριν εμφανιστεί το Material Design. Ο RecyclerView είναι ένας υποδοχέας μεγάλου όγκου δεδομένων που αντικατοπτρίζεται σε ξεχωριστά σύνολα από Views. Με το αντικείμενο αυτό τα δεδομένα μπορούν να ανακυκλώνονται και ο χρήστης έχει τη δυνατότητα να πηγαίνει είτε προς τα κάτω είτε προς τα πάνω και να βλέπει περισσότερες δημοσιεύσεις. Τα κύρια fragments που χρησιμοποιούνε RecyclerView αντικείμενα είναι τα δύο Fragment posts και top_posts. Φορτώνουν τις δημοσιεύσεις των χρηστών μέσα σε RecyclerView αντικείμενα αφού τα έχουν κατεβάσει πρώτα από τη βάση Firebase. Στις εικόνες 9 και 10 βλέπουμε πως είναι το τελικό αποτέλεσμα του RecyclerView της εφαρμογής μας.



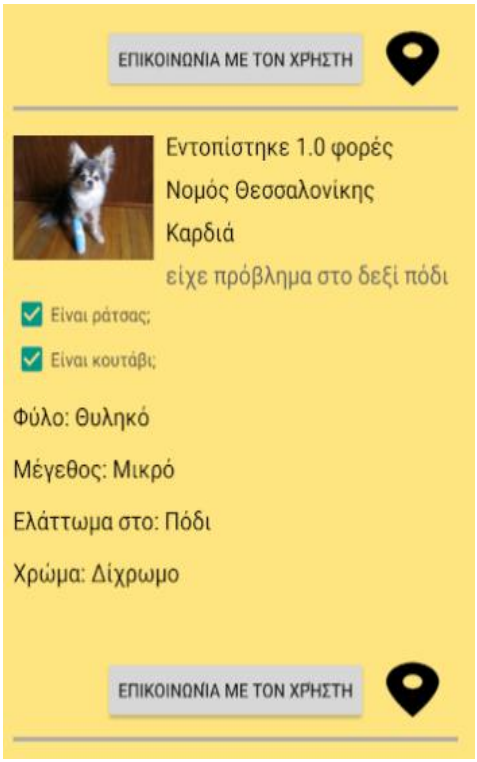
Εικόνα 7 “Alert Dialog για προσθήκη εικόνας”



Εικόνα 8 “Custom Alert Dialog για διαμοιρασμό στοιχείων των χρηστών”



Εικόνα 9 “Fragment posts με δυνατότητα αναζήτησης”



Εικόνα 10 “Παράδειγμα δημοσίευσης μετά την εισαγωγή αντικειμένων στο RecyclerView”

ΕΠΙΛΟΓΟΣ

Σε αυτό το κεφάλαιο αποκτήσαμε μια πρώτη επαφή με το Material design του Android περιγράφοντας τις βασικές αρχές του. Έπειτα, είδαμε ορισμένες από τις βασικές διεπαφές χρήστη που υλοποιήθηκαν χρησιμοποιώντας το Material design. Παράλληλα, αναλύσαμε τα αντικείμενα που χρησιμοποιήσαμε και τις διατάξεις που περιέχονται στο βασικό xml του MainActivity.

ΚΕΦΑΛΑΙΟ 4
ΤΟ ΠΡΟΤΥΠΟ ΤΗΣ ΕΦΑΡΜΟΓΗΣ MVP (MODEL VIEW PRESENTER)

ΕΙΣΑΓΩΓΗ
Σε αυτό το κεφάλαιο θα εξηγήσουμε τι είναι το μοντέλο MVP και ακολούθως θα δούμε τον τρόπο που εφαρμόστηκε στις κυριότερες λειτουργίες της εφαρμογής παραθέτοντας και εξηγώντας τον αντίστοιχο κώδικα.

4.1 ΤΟ ΜΟΝΤΕΛΟ MVP
Το μοντέλο MVP προέκυψε από το γνωστό μοντέλο MVC και το τελευταίο διάστημα φαίνεται να χρησιμοποιείται όλο και περισσότερο στην ανάπτυξη εφαρμογών Android. Επιτρέπει τον διαχωρισμό του επιπέδου παρουσίασης από την λογική της εφαρμογής. Με αυτό τον τρόπο, οτιδήποτε σχετίζεται με το πως λειτουργεί η διεπαφή διαχωρίζεται από τον τρόπο αναπαράστασης του. Αν οι κανόνες του MVP τηρηθούν κατά την διάρκεια της ανάπτυξης μιας εφαρμογής τότε ιδανικά θα πρέπει η ίδια λογική που χρησιμοποιήθηκε να μπορεί να εκτελεσθεί και σε διαφορετικά αντικείμενα.

4.2 MVP ΚΑΙ ANDROID
Αναπτύσσοντας μια εφαρμογή σε Android προκύπτει το εξής πρόβλημα, τα Activities είναι στενά συνδεδεμένα αν όχι πλήρως και με τις διεπαφές χρήστη αλλά και με τους μηχανισμούς πρόσβασης δεδομένων (μεθόδους, κλάσεις, βάσεις δεδομένων). Για να μπορεί μια εφαρμογή να επεκταθεί στο μέλλον καθώς και να συντηρηθεί χρειάζεται διαχωρισμός των επιπέδων. Αν θελήσουμε να κάνουμε κάποια αλλαγή στην εφαρμογή και αντί να πάρουμε τα δεδομένα από μία βάση δεδομένων τα πάρουμε από μια web υπηρεσία θα πρέπει να ξαναφτιάξουμε ολόκληρο το Activity, γεγονός ιδιαίτερα χρονοβόρο. Ακολουθώντας λοιπόν τους κανόνες του MVP, διαχωρίζουμε την εφαρμογή σε τουλάχιστον τρία διαφορετικά επίπεδα ανεξάρτητα το ένα από το άλλο. Με αυτό τον τρόπο, είναι πιο εύκολο και το testing της εφαρμογής και έτσι διαχωρίζεται το κομμάτι του front-end με του back end.

4.2.1 ΕΦΑΡΜΟΖΟΝΤΑΣ ΤΟ MVP ΣΤΟ ANDROID
Παρόλο που το MVP έχει κανόνες που πρέπει να ακολουθούνται υπάρχουν διάφοροι τρόποι για να εφαρμοστεί στον κώδικα μιας εφαρμογής. Ο καθένας μπορεί να προσαρμόσει την λογική του MVP στις ανάγκες του και στον τρόπο που τον βολεύει. Το μοντέλο ποικίλλει αναλόγως με το πόσες ευθύνες θα μεταβιβάσουμε στην κλάση Presenter. Παραδείγματος χάριν, «είναι το κομμάτι του View υπεύθυνο για την ενεργοποίηση ή απενεργοποίηση ενός κουμπιού ή πρέπει να γίνει στον Presenter»; Απορίες τέτοιου τύπου λύνονται αναλόγως με την υλοποίηση που θέλουμε να επιτύχουμε. Επομένως, δεν υπάρχει τυποποιημένος τρόπος ο οποίος μπορείς να ακολουθηθεί τυφλά.

4.2.1.1 Ο PRESENTER

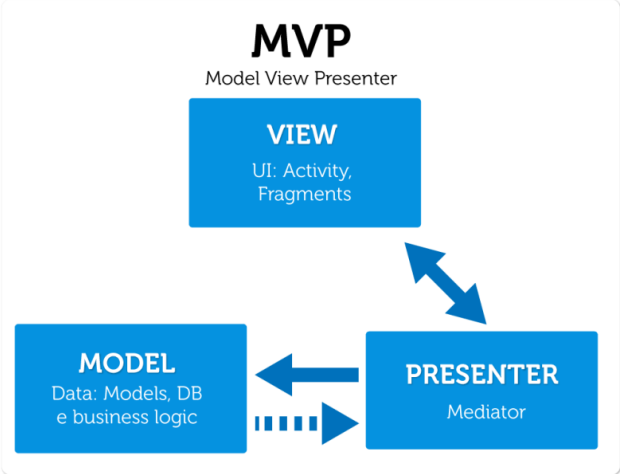
Ο presenter είναι σχεδιασμένος να ενεργεί ως ο ενδιάμεσος ανάμεσα στο View και στο Model. Ανακτά δεδομένα από το μοντέλο και τα επιστρέφει μορφοποιημένα στο View. Σε αντίθεση με το MVC αποφασίζει τι συμβαίνει όταν αλληλοεπιδράμε με το επίπεδο View.

4.2.1.2 ΤΟ VIEW

Το view συνήθως υλοποιείται από το Activity (ή το Fragment, γενικά οτιδήποτε επεκτείνει την κλάση View) περιέχει μια αναφορά στον presenter προκειμένου να μπορεί να τον καλεί και να τον χρησιμοποιεί. Ο presenter είναι υπεύθυνος να παρέχει την δυνατότητα στο View για τη δημιουργία ενός αντικειμένου του. Το μόνο πράγμα που θα κάνει το View είναι να καλεί κάποια μέθοδο του Presenter κάθε φορά που προκύπτει μια ενέργεια στην διεπαφή του χρήστη.

4.2.1.3 ΤΟ MODEL

Σε μία εφαρμογή που ακολουθεί μια καλή αρχιτεκτονική διαχωρισμένη σε επίπεδα το model είναι υπεύθυνο μόνο για να έχουμε πρόσβαση στο επίπεδο domain ή στην διαχείρισης δεδομένων από κάποια βάση. Θα μπορούσαμε να πούμε ότι είναι ο πάροχος των δεδομένων που θα εμφανίσουμε στο View.



Εικόνα 11 “Η αλληλεπίδραση των επιπέδων του MVP”

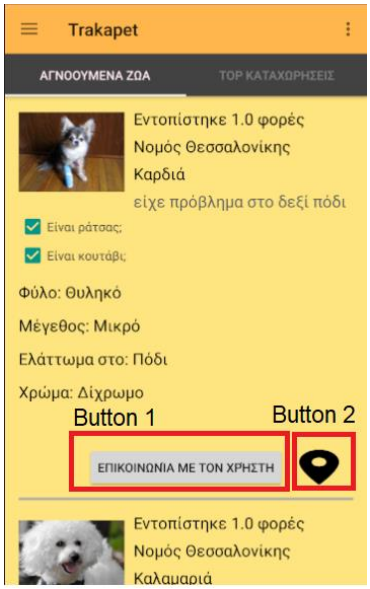
4.3 ΥΛΟΠΟΙΗΣΗ ΤΟΥ MVP ΣΤΗΝ ΕΦΑΡΜΟΓΗ ΜΟΥ

Θα δούμε πως εφαρμόστηκε το MVP στα εξής Fragments:

- posts
- new_post

4.3.1 ΑΝΑΠΤΥΞΗ ΤΟΥ FRAGMENT POSTS

Το fragment posts είναι υπεύθυνο για την πρόσβαση στα δεδομένα του Firebase, την τοποθέτηση τους στο RecyclerView και την διαχείριση των κλικ των δύο κουμπιών όπως βλέπουμε στην εικόνα 12. Αυτές είναι οι λειτουργίες πολύ συνοπτικά και ας δούμε πως διαχωρίστηκαν σε Model, View και Presenter.



Εικόνα 12 "Διαθέσιμα κουμπιά σε κάθε δημοσίευση"

Αρχικά, δημιουργούμε ένα Interface με όνομα PostContract οποίο συμπεριλαμβάνει άλλα δύο interface. Το ένα interface είναι για το View και το άλλο για τον Presenter. Ο λόγος που δημιουργούμε interface για το Fragment είναι για να αποδεσμευτούμε από την εξάρτηση με τις πραγματικές κλάσεις. Έτσι ο Presenter δεν θα βασίζεται σε συγκεκριμένο Fragment αλλά στο PostContract interface και μπορεί να χρησιμοποιηθεί και σε άλλα fragments ή activities χωρίς να χρειάζονται αλλαγές στον Presenter. Είναι συνηθισμένη πρακτική στο MVP να ονομάζουμε τα Interfaces Contracts λόγω του ότι παρέχουν τις υπογραφές (το όνομα μιας μεθόδου σε συνδυασμό με τις παραμέτρους της) των υπόλοιπων επιπέδων View και Presenter και όπως λέει το όνομα ουσιαστικά είναι ένα συμβόλαιο μεταξύ τους.

```
public interface PostContract {

    interface View {


        void addPosts(List<Animal> posts);

        void refreshPosts(List<Animal> posts);

        void show_location(double lati, double longi);

        void showProgress();

        void hideProgress();

        void showDetails(String user_id);

        void showErrorToast(String s);
    }

    interface Presenter {


        void get_coordinates(int position);

        void loadData(final boolean isRefresh);

        void loadFeeds(boolean isRefresh);

        void getUserId(int position);

        void loadDataDescending(boolean b);

        void loadConnectedPosts(boolean isRefresh);

        void loadSpecificData(String perioxh, boolean b);
    }

}
```

Κώδικας 11 “Το interface των δημοσιεύσεων”

Το Fragment posts είναι το view στο οποίο θα πρέπει να κάνουμε implement το interface View εντός του Interface PostContract προκειμένου να υλοποιήσουμε τις μεθόδους που θα χρειαστεί να κληθούν εντός του View. Οι μέθοδοι που μπαίνουν στο View είναι αυτές οι οποίες θα κληθούν μέσα από τον Presenter. Επιπλέον, μέσα στο fragment κάνουμε μια αναφορά στον PostPresenter για να μπορούμε να τον καλέσουμε αργότερα. Το RecyclerView το αναφέραμε και σε προηγούμενο κεφάλαιο ως συστατικό μέρος του Material design και σε συνδυασμό με τον MyAdapter θα γεμίσουμε το container που είχε το αντικείμενο ViewPager που είδαμε στο βασικό xml activity_main στον Κώδικα 1. Το SwipeRefreshLayout είναι ένα Widget που αντιλαμβάνεται πότε ο χρήστης σέρνει το δάχτυλο του (κίνηση swipe) φτάνοντας στο τέλος του Recyclerview με σκοπό να κάνει refresh. Έτσι γίνεται έλεγχος για το αν έχουν προστεθεί καινούργιες δημοσιεύσεις αφού έχει ήδη ανοίξει την εφαρμογή.

```
public class posts extends Fragment implements PostContract.View, My-
Adapter.ItemClickCallback{

    PostPresenter presenter;
    private RecyclerView recyclerview;
    private SwipeRefreshLayout mSwipeRefreshLayout;
    private MyAdapter adapter;
```

Κώδικας 12 "Βασικές μεταβλητές του Fragment posts"

Μέσα στο Viewpager θα ενσωματώσουμε το layout που χρειάζεται το Fragment posts. Στο layout στον Κώδικα 13 υπάρχει ένα ακόμα Toolbar το οποίο βρίσκεται κάτω από το βασικό και αυτό γιατί δεν θέλουμε τα γραφικά να είναι πολλά και να κουράζουν τον χρήστη.

Αυτό το toolbar χρησιμοποιείται μόνο στη μία σελίδα του ViewPager και αυτό γιατί θέλουμε ο χρήστης να μπορεί να αναζητήσει δημοσιεύσεις από εξαφανισμένα ζώα με βάση την πόλη που επιθυμεί. Αυτό το χαρακτηριστικό είναι μόνο για την καρτέλα ΑΓΝΟΟΥΜΕΝΑ ΖΩΑ και όχι και για τις TOP ΚΑΤΑΧΩΡΗΣΕΙΣ λόγω του ότι στις TOP καταχωρήσεις θέλουμε να φαίνονται οι δημοσιεύσεις με τις περισσότερες φορές εντοπισμού γενικά. Έτσι λοιπόν, αυτό το Fragment χρησιμοποιεί δικό του layout γιατί υπάρχουν συγκεκριμένες απαιτήσεις στην υλοποίηση. Στην εικόνα 13 βλέπουμε πως φαίνεται το Toolbar με την αναζήτηση αφού εξαφανιστεί το βασικό Toolbar. Αν ο χρήστης πραγματοποιήσει μια αναζήτηση μπορούμε να δούμε ότι τα αποτελέσματα φιλτράρονται με βάση την περιοχή, όπως βλέπουμε στην εικόνα 15.

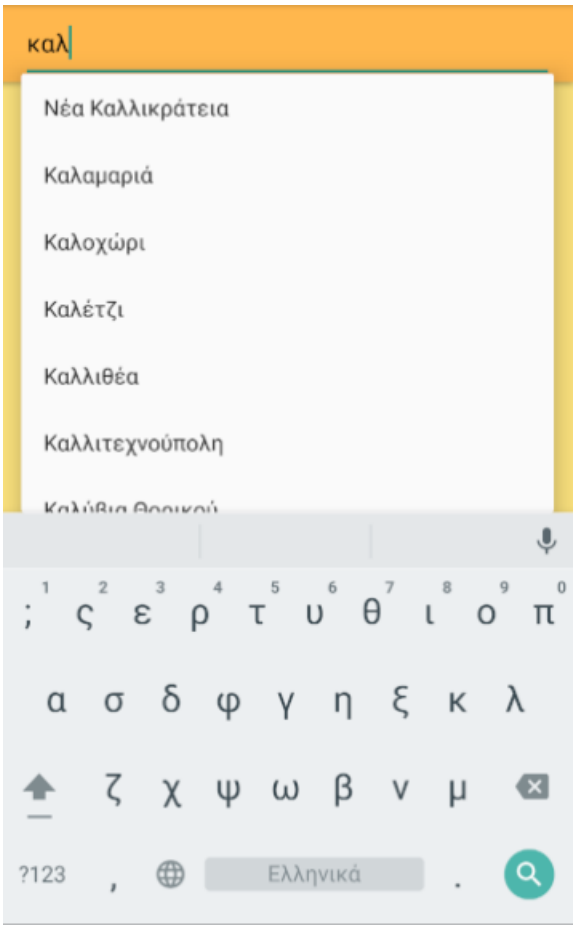
Το AutoCompleteTextView είναι ένα View το οποίο εμφανίζει προτάσεις στον χρήστη καθώς αυτός πληκτρολογεί όπως βλέπουμε στην εικόνα 15. Η λίστα των προτάσεων που βγαίνει είναι ένα drop down menu από το οποίο ο χρήστης μπορεί να επιλέξει και να αντικαταστήσει το περιεχόμενο της μπάρας αναζήτησης.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/sub_fragment2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#FFE57F"
    android:elevation="@dimen/_10sdp"
    tools:context="fragments.posts">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="@color/colorShop">

            <AutoCompleteTextView
                android:id="@+id/search_bar"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
                android:layout_marginLeft="@dimen/_10sdp"
                android:layout_marginRight="@dimen/_10sdp"
                android:ems="10"
                android:hint="Περιοχή..."
                android:imeOptions="actionSearch"
                android:inputType="textAutoComplete|textAutoCorrect"
                android:textColor="@android:color/black"/>

        </android.support.v7.widget.Toolbar>
    </android.support.design.widget.AppBarLayout>


    <android.support.v4.widget.SwipeRefreshLayout
        android:id="@+id/swiperefreshlayout"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <android.support.v7.widget.RecyclerView
            android:id="@+id/recyclerview"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:clipToPadding="false"
            android:paddingTop="@dimen/_80sdp" />

    </android.support.v4.widget.SwipeRefreshLayout>
</FrameLayout>
```

Κώδικας 13 “Το Layout του Fragment posts”

Εικόνα 13 “Εξαφάνιση του Toolbar”

Εικόνα 14 “Εμφάνιση του Toolbar”



Εικόνα 15 “Αναζήτηση περιοχής μέσω του AutoCompleteTextView”

Πάμε να δούμε λοιπόν πρώτα πως γεμίζει το RecyclerView με δημοσιεύσεις χρησιμοποιώντας το μοντέλο MVP και έπειτα πως εμφανίζονται οι δημοσιεύσεις με εξατομικευμένη αναζήτηση. Το κομμάτι του Firebase θα αναλυθεί στο επόμενο κεφάλαιο οπότε τώρα θα εστιάσουμε στο πως γίνεται η επικοινωνία ανάμεσα στις κλάσεις του MVP και όχι σε κώδικα σχετικό με το Firebase.

```
if(currentUser != null) {
    showProgress();
    init();
    presenter = new PostPresenter(this);
    presenter.loadData(false);
    loadData();
}
```

Κώδικας 14 “Βασικές κλήσεις μεθόδων του Fragment posts”

Αφού λοιπόν γίνει πρώτα έλεγχος μέσω μεθόδων του Firebase για το αν ο χρήστης είναι συνδεδεμένος στον Κώδικα 14 προχωράμε στην κλήση της μεθόδου showProgress() η οποία ενημερώνει τον χρήστη ότι τα δεδομένα φορτώνουν χρησιμοποιώντας ένα αντικείμενο ProgressDialog(). Όταν φορτωθούν τα δεδομένα θα καλέσουμε την αντίστοιχη μέθοδο για το κλείσιμο του ProgressDialog.

```
@Override
public void showProgress() {

    progress = new ProgressDialog(getActivity());
    progress.setTitle("Loading");
    progress.setMessage("Syncing");
    progress.setCancelable(false);
    progress.show();

    mSwipeRefreshLayout.setVisibility(View.INVISIBLE);
}

@Override
public void hideProgress() {

    progress.dismiss();

    mSwipeRefreshLayout.setVisibility(View.VISIBLE);
}
```

Κώδικας 15 “Βασικές μέθοδοι του ProgressDialog()”

Η μέθοδος Init() είναι υπεύθυνη για την αρχικοποίηση του RecyclerView και του Adapter.

```
private void init() {

    Context context = getContext();
    recyclerview = (RecyclerView)v.findViewById(R.id.recyclerview);
    recyclerview.setLayoutManager(new LinearLayoutManager(context));
    recyclerview.setHasFixedSize(true);
    adapter = new MyAdapter(new ArrayList<Animal>(),context);
    recyclerview.setAdapter(adapter);
    adapter.setItemClickCallback(this);
}
```

Κώδικας 16 “Αρχικοποίηση του RecyclerView και του Adapter”

Η κλήση της μεθόδου setLayoutManager() στον Κώδικα 16 είναι απαραίτητη μιας και δεν γίνεται να λειτουργήσει ο RecyclerView χωρίς να έχει τεθεί πρώτα ένας LayoutManager. Το αντικείμενο αυτό είναι υπεύθυνο για την μέτρηση και τοποθέτηση των item view (η κάθε δημοσίευση είναι και ένα item view) μέσα σε ένα RecyclerView καθώς και για να εντοπίζει πότε θα ανακυκλώσει το κάθε item view που δεν είναι ορατό στον χρήστη. Επιπλέον, χρειάζεται να γνωρίζει αν το μέγεθος του δηλαδή το ύψος και το πλάτος του, εξαρτάται από το περιεχόμενο του adapter προκειμένου να γνωρίζει αν θα πρέπει να αλλάζει μέγεθος κάθε φορά που ένα item προστίθεται ή αφαιρείται από τον adapter. Σε περίπτωση που το μέγεθος του RecylerView δεν εξαρτάται από το περιεχόμενο του Adapter και το γνωρίζουμε αυτό εκ των προτέρων βάζουμε την τιμή true. Αυτό το γνωρίζουμε διότι τα Views που χρησιμοποιούμε στα layout είναι σταθερά και δεν πρόκειται ξαφνικά να αλλάξουμε κάποιο ανώτερο View προγραμματιστικά. Η επιλογή αυτή βελτιστοποιεί τον ίδιο τον RecyclerView μιας και με τους συνεχόμενους ελέγχους έπειτα από κάθε προσθαφαίρεση item η διαδικασία είναι περισσότερο χρονοβόρα και χρησιμοποιεί περισσότερη μνήμη. Αρχικοποιούμε έπειτα τον Adapter με ένα άδειο Arraylist και μόλις παρθούν τα δεδομένα από το Firebase θα το γεμίσουμε. Η μέθοδος setItemClickCallback είναι για να μπορούν να γίνουν αντιληπτά τα κλικ του χρήστη στα δύο κουμπιά που υπάρχουν στην κάθε δημοσίευση.

```
public class MyAdapter extends RecyclerView.Adapter<MyAdapter.Animal-
ViewHolder> {

// o adapter tha prepei na eidopoiei ton presenter otan thelei na
kanei kapoia douleia
//kai kathe item tou collection tha prepei na to diaxeirizetai ws
ksexwristo MVP view


    private ArrayList<Animal> animals;
    private Context mContext;

    private ItemClickCallback itemClickCallback;

    public interface ItemClickCallback {
        void onItemClick(View v, int position);
    }

    public void setItemClickCallback(final ItemClickCallback item-
ClickCallback) {
        this.itemClickCallback = itemClickCallback;
    }


    //private AnimalsListPresenter presenter;

    public MyAdapter(ArrayList<Animal> animals, Context c) {
        this.animals = animals;
        mContext = c;
    }

    @Override
    public AnimalViewHolder onCreateViewHolder(ViewGroup parent, int
viewType) {
        //to layout tou kathe zwou ginetai set edw kai efarmozetai se
kathe antikeimeno tupou Animal
        View v = LayoutInflater.from(parent.getContext()).in-
flate(R.layout.pet_box,parent,false);
        return new AnimalViewHolder(v);
    }

    @Override
    public void onBindViewHolder(AnimalViewHolder holder, int posi-
tion) {

        Animal animal = animals.get(position);

        holder.setPetsImage(animal.getPhoto_url());
        holder.setTimes_found(animal.getFores_entopismou());
        holder.setNomos(animal.getCity());
        holder.setPerioxi(animal.getRegion());
        holder.setDescription(animal.getDescription());
        holder.setIf_ratsas(animal.getIs_breed());
        holder.setIf_small(animal.getIs_puppy());
        holder.setGender(animal.getGender());
        holder.setMegethos(animal.getSize());
        holder.setFlaws(animal.getFlaw());
        holder.setColor(animal.getColor());
    }
```

Κώδικας 17 “Η κλάση MyAdapter”

```
@Override
public int getItemCount() {
    return animals.size();
}

public void setPosts(ArrayList<Animal> posts) {
    this.animals = posts;
}

public void clearData() {
    animals.clear();
}

public void addAllData(List<Animal> posts) {
    this.animals.addAll(posts);
}
    }
```

Κώδικας 18 “Μέθοδοι του MyAdapter”

```
public class AnimalViewHolder extends RecyclerView.ViewHolder im-
plements View.OnClickListener{

    View mView;
    ImageView petsImage;
    TextView times_found;
    TextView nomos;
    TextView perioxi;
    TextView description;
    CheckBox if_ratsas;
    CheckBox if_small;
    Button contact_user;
    Button show_location;
    TextView gender;
    TextView size;
    TextView flaws;
    TextView color;

    public AnimalViewHolder(View itemView) {
    super(itemView);
    mView = itemView;
    petsImage = (ImageView)itemView.findViewById(R.id.pet_image);
        times_found = (TextView) itemView.findView-
ById(R.id.times_found);
        nomos = (TextView)itemView.findViewById(R.id.nomos);
        perioxi = (TextView)itemView.findViewById(R.id.perioxi);
        description = (TextView)itemView.findView-
ById(R.id.pet_description);
        if_ratsas = (CheckBox)itemView.findViewById(R.id.check-
box_breed);
        if_small = (CheckBox)itemView.findViewById(R.id.check-
box_small);

        gender = (TextView)itemView.findView-
ById(R.id.textView_fulo);
        size = (TextView) itemView.findView-
ById(R.id.textView_megethos);
        flaws = (TextView)itemView.findView-
ById(R.id.textView_flaws);
        color = (TextView) itemView.findView-
ById(R.id.textView_color);

        //auta ta duo koumpia tha exoun listeners
        contact_user = (Button) itemView.findViewById(R.id.con-
tact_pets_user);
        show_location = (Button)itemView.findView-
ById(R.id.show_location);

        contact_user.setOnClickListener(this);
        show_location.setOnClickListener(this);
```

Κώδικας 19 “Η κλάση AnimalViewHolder”

Η δουλειά του Adapter στον Κώδικα 17 είναι να δένει τα δεδομένα σε ξεχωριστά Views μέσα στον RecyclerView. Για να μπορεί να χρησιμοποιήσει το RecyclerView πρέπει να επεκτείνουμε την κλάση RecyclerView.Adapter. Ο Adapter ακολουθεί το πρότυπο σχεδιασμού ViewHolder που σημαίνει ότι μπορούμε να καθορίσουμε μία κλάση που να εκφράζει κάποια οντότητα, στην περίπτωση μας την κλάση Animal που χρησιμοποιείται σχεδόν σε κάθε κλάση της εφαρμογής μας. Στο διάγραμμα 1 μπορούμε να δούμε τι περιέχει αυτή η κλάση παρατηρώντας το διάγραμμα κλάσης.

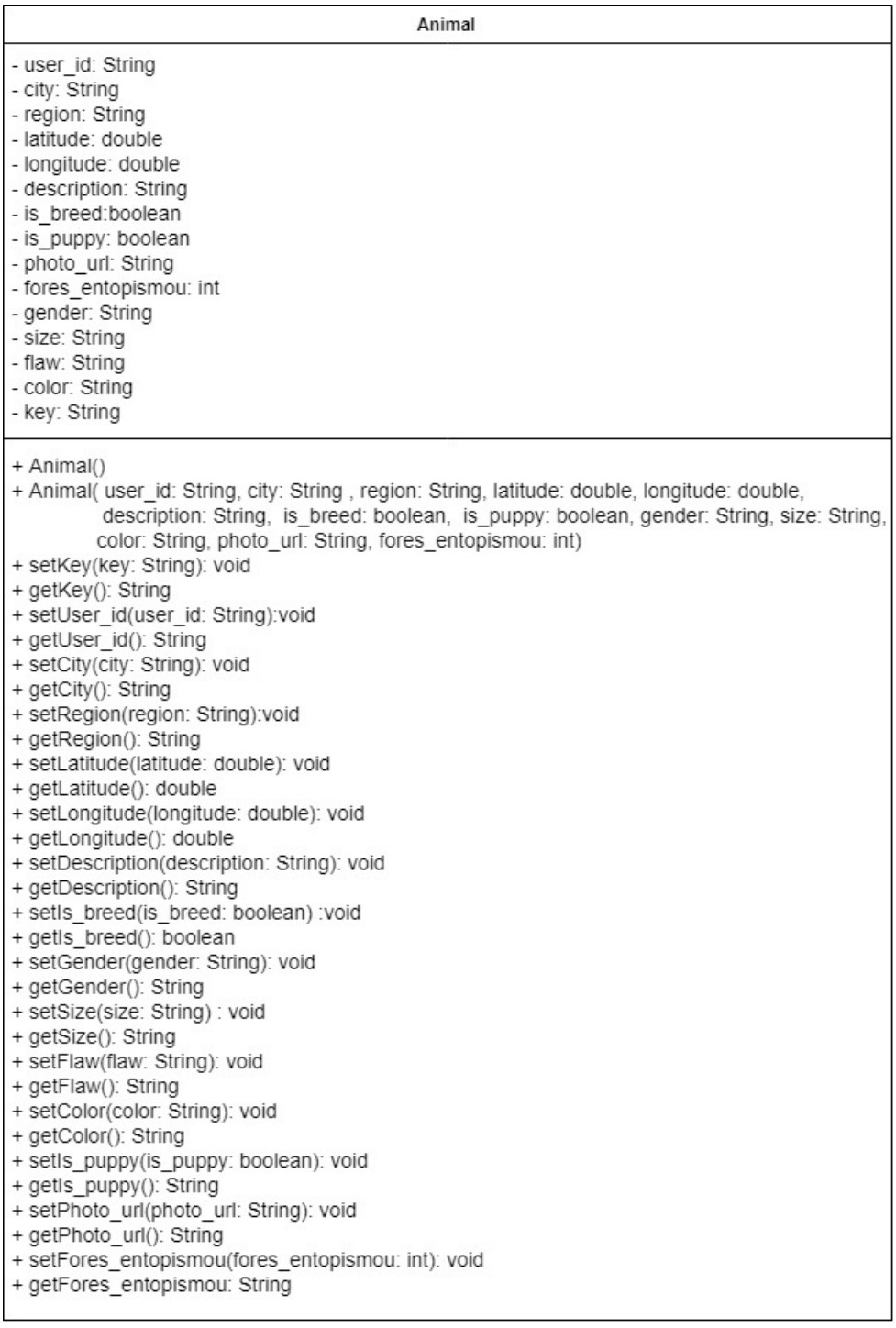
Στη μέθοδο onCreateViewHolder στον Κώδικα 17 καθορίζουμε το layout που θα χρησιμοποιήσει το κάθε στοιχείο του RecyclerView. Όπως υποδεικνύει και το όνομα της, αυτή η μέθοδος καλείται όταν ο ViewHolder αρχικοποιείται. Αφού καθοριστεί η διάταξη του κάθε στοιχείου θα πρέπει να καθορίσουμε τα περιεχόμενα τους. Αυτό συμβαίνει στη μέθοδο onBindViewHolder. Σε αυτήν τη μέθοδο λοιπόν παίρνονται ένα ένα τα στοιχεία και χρησιμοποιώντας τη μεταβλητή θέσης παίρνουμε τα στοιχεία από το ArrayList τύπου Animal. Με την ίδια σειρά που βρίσκονται στο ArrayList τοποθετούνται και στον RecyclerView. Στον δομητή του AnimalViewHolder αρχικοποιούμε τα Views που ανήκουν στα στοιχεία του RecyclerView για να μπορούμε να τοποθετήσουμε το περιεχόμενο μέσω του onBindViewHolder.

Αφού γίνουν λοιπόν όλες οι απαραίτητες διαδικασίες στον Adapter και αρχικοποιηθεί τότε τοποθετείται πάνω στον RecyclerView με την μέθοδο setAdapter().

Ακολούθως, αρχικοποιούμε τον Post Presenter βάζοντας το keyword this που δηλώνει το View στο οποίο βρισκόμαστε το οποίο είναι το Fragment. Ο δομητής του PostPresenter παίρνει ως παράμετρο το Fragment αρχικοποιώντας τη μεταβλητή mview προκειμένου να μπορούμε να καλέσουμε μέσα από τον Presenter τις μεθόδους που βρίσκονται στο Fragment. Επιπλέον, αρχικοποιούμε και δύο Arraylist τα οποία θα χρειαστούμε στη συνέχεια.

```
public PostPresenter(PostContract.View mView) {
    this.mView = mView;
    animals = new ArrayList<>();
    ids = new ArrayList<>();
}
```

Κώδικας 20 “Ο δομητής του PostPresenter”



Διάγραμμα 1 “Διάγραμμα Κλάσης Animal”

Έπειτα καλούμε τη μέθοδο loadData() του presenter δίνοντας την παράμετρο false. Η loadData και γενικά οι περισσότερες μέθοδοι εντός του Presenter χρησιμοποιούν την βάση δεδομένων Firebase προκειμένου να γεμίσουν τα Arrays που αρχικοποιήθηκαν χρησιμοποιώντας το Model Animal. Μόλις γεμίσει το ArrayList με αντικείμενα τύπου Animal θα κληθεί η μέθοδος loadFeed() του Κώδικα 21 η οποία αναλόγως με το αν η Boolean μεταβλητή isRefresh είναι true ή false εκτελεί διαφορετική μέθοδο στο Fragment. Όπως προαναφέραμε, στον Presenter θα πρέπει να υπάρχει η λογική της εφαρμογής οπότε μέσα σε αυτόν ελέγχεται η τιμή true ή false και καλείται αναλόγως η μέθοδος. Η μεταβλητή isRefresh χρειάζεται για να καθορίσει αν ο χρήστης έχει κάνει swipe στην οθόνη ή όχι. Αν έχει κάνει θα πρέπει ο adapter να κάνει Refresh. Ας δούμε αναλυτικά τι συμβαίνει με αναφορά στον κώδικα. Στην περίπτωση λοιπόν που η μεταβλητή είναι true θα κληθεί η μέθοδος refreshPosts(animals) διαφορετικά η μέθοδος addPosts(animals).

```
@Override
public void loadFeeds(boolean isRefresh) {

    mView.hideProgress();

    if (isRefresh)
        mView.refreshPosts(animals);
    else mView.addPosts(animals);
}
```

Κώδικας 21 “Η μέθοδος loadFeeds() του PostPresenter”

Η μέθοδος refreshPosts θα καθαρίσει όλα τα δεδομένα από τον adapter και θα βάλει αυτά που μόλις φόρτωσε οπότε αν τυχόν έχουν μπει καινούργιες δημοσιεύσεις θα εμφανιστούν.

```
public void refreshPosts(List<Animal> posts)
{
    adapter.clearData();
    adapter.addAllData(posts);
    adapter.notifyDataSetChanged();
    mSwipeRefreshLayout.setRefreshing(false);

}
```

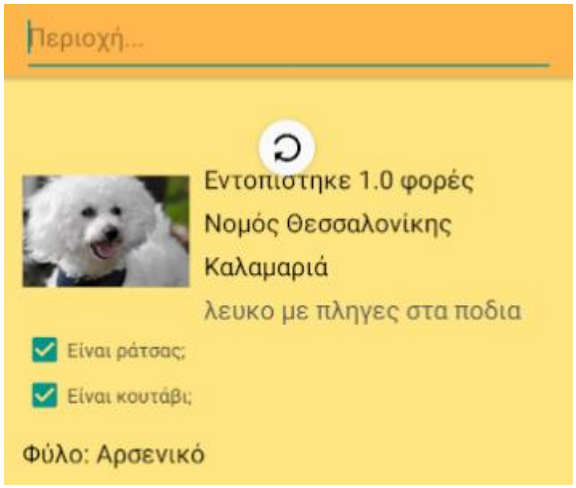
Κώδικας 22 “Η μέθοδος refreshPosts() του PostPresenter”

Αφού ανανεωθεί ο adapter με την μέθοδο notifyDataSetChanged θέτουμε την ιδιότητα setRefreshing του SwipeRefreshLayout ως false διότι με το που ενεργοποιείται ο RefreshListener με την κίνηση swipe εμφανίζεται το σύμβολο του Refresh που βλέπουμε στην εικόνα 16 για να ειδοποιηθεί ο χρήστης ότι το Refresh πραγματοποιείται.

Από την άλλη, η μέθοδος addPosts() τρέχει την πρώτη φορά που φορτώνεται ο Adapter που είναι ήδη κενός και δεν χρειάζεται να καθαρίσει τα προηγούμενα.

```
public void addPosts(List<Animal> posts)
{
    System.out.println("ADDPOSTS" + posts.size());
    adapter.addAllData(posts);
    adapter.notifyDataSetChanged();
}
```

Κώδικας 23 “Η μέθοδος addPosts() του PostPresenter”



Εικόνα 16 “SwipeRefreshLayout”

Στη συνέχεια θα επεξηγηθεί ο τρόπος που υλοποιείται η μπάρα αναζήτησης και η φόρτωση των δεδομένων έπειτα από φιλτράρισμα αλλά και πως ξαναγεμίζει ο RecyclerView μετά τη διαγραφή της περιοχής.

Αρχικά, να αναφέρουμε ότι οι περιοχές είναι αποθηκευμένες στο strings.xml σε ένα string-array.

```
<string-array name="perioxes">
    <item>Άγιος Νικόλαος Χαλκιδικής</item>
    <item>Αρναία</item>
    <item>Βεργιά</item>
    <item>Γαλάτιστα</item>
    <item>Γερακινή</item>
    <item>Δάφνη Αγίου Όρους</item>
    <item>Ιερισσός</item>
    <item>Καρυές</item>
    <item>Κασσάνδρα</item>
    <item>Κρήμνη</item>
    <item>Μεγάλη Παναγιά</item>
    <item>Νέα Ηράκλεια</item>
    <item>Νέα Καλλικράτεια</item>
    <item>Νέα Μουδανιά</item>
    <item>Νέα Πλάγια</item>
    <item>Νέα Τρίγλια</item>
    <item>Νέος Μαρμαράς</item>
    <item>Νικήτη</item>
    <item>Ολυμπιάδα</item>
    <item>Ορμύλια</item>
    <item>Πευκοχώρι</item>
    <item>Πολύγυρος</item>
    <item>Ποτίδαια</item>
    <item>Συκιά</item>
.....
```

Κώδικας 24 “Περιεχόμενο του strings.xml”

Πρώτα παίρνουμε τις περιοχές από το xml και τις τοποθετούμε σε ένα ArrayList τύπου String το οποίο θα το χρησιμοποιήσουμε για να αρχικοποιήσουμε έναν ArrayAdapter. Ο ArrayAdapter ουσιαστικά κάνει την ίδια δουλειά με τον MyAdapter που εξηγήσαμε προηγουμένως. Δημιουργεί views δηλαδή παίρνοντας το κάθε item από το list που του δώσαμε.

```
list = new ArrayList<>();

perioxes = getResources().getStringArray(R.array.perioxes);

for(String perioxi:perioxes)
     list.add(perioxi);

search_adapter = new ArrayAdapter<String>(getContext(), android.R.lay-
out.simple_list_item_1,list);
final AutoCompleteTextView textView =  (AutoCompleteTextView) v.findView-
ById(R.id.search_bar);
textView.setAdapter(search_adapter);
```

Κώδικας 25 “Γέμισμα του AutoCompletetextView”

Εφαρμόζουμε λοιπόν τον ArrayAdapter στο AutoCompleteTextView και έτσι με το που ξεκινάει ο χρήστης να πληκτρολογεί εμφανίζεται η λίστα με τις περιοχές. Για να φορτωθούν όμως οι δημοσιεύσεις έπειτα από επιλογή μιας περιοχής δημιουργήσαμε έναν OnItemClickListener ο οποίος με το που αντιληφθεί το κλικ του χρήστη πάνω στη λίστα θα αποθηκεύσει την περιοχή σε μία μεταβλητή. Την μεταβλητή μπορούμε εύκολα να τη βρούμε διαβάζοντας από τον Adapter το item που βρίσκεται στην θέση στην οποία έκανε κλικ ο χρήστης, μιας και η σειρά των στοιχείων του TextView είναι ίδια με την σειρά των στοιχείων του Adapter.

Στη συνέχεια, καλούμε μία μέθοδο του Presenter η οποία θα φιλτράρει τα δεδομένα της βάσης και θα πάρει μόνο αυτά με την συγκεκριμένη περιοχή. Έπειτα, θα διαγράψει τα προηγούμενα και θα βάλει τα φιλτραρισμένα αντικείμενα. Γι' αυτόν τον λόγο δίνουμε την μεταβλητή true ώστε ο Presenter να καλέσει την μέθοδο refreshPosts().

```
textView.setOnItemClickListener(new AdapterView.OnItemClickListener() {

    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position,
long id) {
        perioxh = (String) parent.getItemAtPosition(position);

        //here we send the area selected to the presenter to load only the
relative posts
        presenter.loadSpecificData(perioxh, true);
    }
});
```

Κώδικας 26 "Ο ItemClickListener του AutoCompleteTextView"

```
//an h perioxh pou epelekse o xrhsths einai idia me to post tote tha to em-
fanisoume
if (perioxh.equals(region)) {
    Animal match = new Animal(usersid, city, region, latitude, longitude,
description,
            breed, small, gender, size, flaws, color, url, fores_entopis-
mou);
    match.setKey(k);
    animals.add(match);
}
```

Κώδικας 27 "Πως επιλέγονται τα αντικείμενα ανά περιοχή"

Η μέθοδος loadSpecificData() που θα δούμε ολοκληρωμένο το περιεχόμενο της στο κομμάτι του Firebase, παίρνει τα δεδομένα από το Firebase και αφού αρχικοποιηθούν οι μεταβλητές που χρειαζόμαστε ελέγχει αν η περιοχή που πατήθηκε ισούται με την αντίστοιχη μεταβλητή της περιοχής. Αν όντως ισούται τότε δημιουργούμε ένα αντικείμενο Animal και το αποθηκεύουμε στο ArrayList τύπου Animals.

Για την υλοποίηση της φόρτωσης των δεδομένων δημιουργούμε έναν listener ο οποίος ενεργοποιείται με κάθε αλλαγή του κειμένου. Όπως βλέπουμε υπάρχουν τρεις διαφορετικές μέθοδοι. Η μέθοδος beforeTextChanged() καλείται πριν αλλάξει η λέξη, η μέθοδος onTextChanged() κατά τη διάρκεια της αλλαγής και η μέθοδος afterTextChanged() αφού έχει γίνει η αλλαγή. Η μεταβλητή count μετράει τους χαρακτήρες της λέξης που πληκτρολογείται ή τοποθετήθηκε μέσα στο TextView, αν ξεκινήσουμε και διαγράφουμε δεν προστίθεται κάποιο γράμμα στη λέξη επομένως η μεταβλητή θα έχει την τιμή 0. Η μεταβλητή before περιέχει των χαρακτήρων το οποίο υπόκειται αλλαγή, δηλαδή αν έχουμε γράψει «Θεσσαλ» και πατήσουμε το «Θεσσαλονίκη» από τη λίστα τότε το before θα έχει την τιμή 7 και το count θα έχει την τιμή 11. Αξιοποιώντας λοιπόν τον τρόπο λειτουργίας της μεθόδου αυτής, μπορούμε να εντοπίσουμε πότε διαγράφεται η λέξη. Αν ο χρήστης ξεκινήσει να διαγράφει τότε μιας και ένας χαρακτήρας είναι αυτός που υπόκειται σε αλλαγή το

```
textView.addTextChangedListener(new TextWatcher() {

    @Override
    public void beforeTextChanged(CharSequence s, int start, int count,
int after) {
    }

    @Override
    public void onTextChanged(CharSequence s, int start, int before, int
count) {

        String p = (textView.getEditableText().toString());

        if (before > count) {
            if(p.equals(""))
                isBackspaceClicked = true;
        } else {
            isBackspaceClicked = false;
        }
    }

    @Override
    public void afterTextChanged(Editable s) {


        if (!isBackspaceClicked) {
            // Your current code
        } else {
            presenter.loadData(true);
        }
    }
});
```

Κώδικας 28 "Το αντικείμενο TextWatcher"

before θα έχει την τιμή 1 και αφού δεν γράφονται χαρακτήρες η μεταβλητή count θα είναι μηδέν. Επομένως αν το before είναι μεγαλύτερο του count πραγματοποιείται διαγραφή.

Λόγω όμως του ότι θέλουμε να διαγραφεί ολοκληρωτικά η λέξη φορτώνουμε τις δημοσιεύσεις μόνο όταν η περιοχή ισούται με το κενό.

Μόλις λοιπόν η λέξη γίνει κενή αλλάζουμε μια μεταβλητή σε true και κάνουμε τις απαραίτητες ενέργειες. Οι δημοσιεύσεις θέλουμε να φορτωθούν αφού έχει ολοκληρωθεί η αλλαγή οπότε καλούμε την loadData() εντός της μεθόδου afterTextChanged().

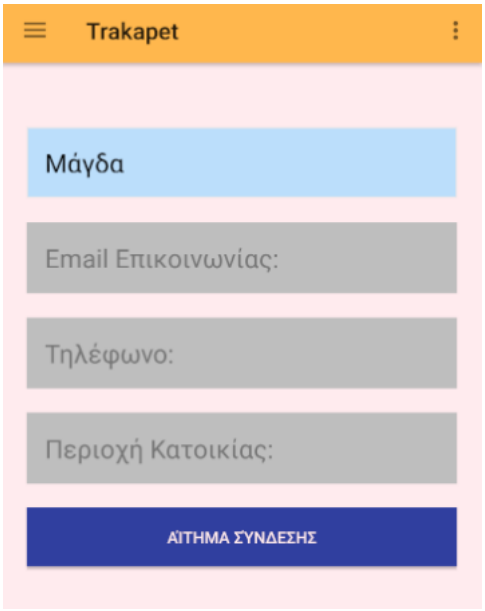
Υπάρχει όμως και μια άλλη περίπτωση που πρέπει να ληφθεί υπόψιν, ο χρήστης μπορεί να κάνει refresh ενώ έχει δηλωμένη συγκεκριμένη περιοχή. Εδώ απλά χρειάζεται ένας έλεγχος της μεταβλητής isBackSpaceClicked εντός του OnRefreshListener. Αν είναι true όπως ήδη αναφέρθηκε, σημαίνει ότι η λέξη έχει διαγραφεί επομένως θα καλέσουμε την μέθοδο loadData() για να φορτώσουν όλες οι περιοχές. Από την άλλη αν είναι false σημαίνει ότι υπάρχει δηλωμένη περιοχή οπότε θα φορτώσει η loadSpecificData(). Αν θέλουμε να ακολουθούμε το MVP πιο αυστηρά μπορούμε όλες αυτούς τους ελέγχους να τους τοποθετήσουμε μέσα στον Presenter.

Όσον αφορά την ανάδραση του χρήστη με τα δύο κουμπιά που βρίσκονται σε κάθε item, την υλοποίηση αυτών την αναλαμβάνουν διαφορετικά fragments αλλά το fragment posts υλοποιεί ένα σημαντικό κομμάτι αυτών. Είναι υπεύθυνο να στέλνει όλες τις απαραίτητες πληροφορίες για να φιλτραριστούν τα αποτελέσματα μετέπειτα.

```
@Override
public void onItemClick(View v, int position) {

    switch(v.getId())
    {
        case R.id.contact_pets_user:
        presenter.getUserId(position);
            break;


        case R.id.show_location:
        presenter.get_coordinates(position);
            break;
    }
}
```

Κώδικας 29 "Αναγνώριση των κλικ των δύο βασικών κουμπιών στις δημοσιεύσεις"

Οι listeners όπως έγινε κατανοητό δημιουργούνται και ρυθμίζονται εντός του MyAdapter αλλά υλοποιούνται μέσα στο Fragment. Μόλις λοιπόν πατήσει ο χρήστης το κουμπί της επικοινωνίας θα κληθεί η μέθοδος getUserId(position) όπου παίρνει το position ενός στοιχείου από τον Adapter. Η θέση του στοιχείου είναι ίδια με την θέση που έχει στο ArrayList επομένως παίρνουμε το αντικείμενο τύπου Animal και μετά το user_id του χρήστη που είναι συνδεδεμένος. Με το που πατηθεί η επικοινωνία θα φορτωθούν τα στοιχεία του χρήστη αλλά πρώτα απαιτούμε να έχει συμπληρώσει ο ίδιος συνδεδεμένος χρήστης το προφίλ του. Επομένως, θα γίνει πρώτα έλεγχος γι' αυτό το κομμάτι το οποίο αφορά το Firebase. Αναλυτικότερα θα το δούμε στο επόμενο κεφάλαιο. Εάν έχει συμπληρωθεί το προφίλ του τότε θα φορτωθεί η μέθοδος showDetails(user_id) του Κώδικα 30 που βρίσκεται μέσα στο Fragment, διαφορετικά θα φορτωθεί η μέθοδος showErrorToast("Το SEND A REQUEST please fill your profile info") για να ειδοποιηθεί ο χρήστης για το τι πρέπει να κάνει. Η λογική αυτή εφαρμόστηκε λόγω του ότι ο χρήστης θα πρέπει να είναι συνδεδεμένος με τον χρήστη που θέλει να επικοινωνήσει προκειμένου να δει τα στοιχεία του. Για να συνδεθεί θα πρέπει να στείλει ένα αίτημα σε αυτόν όπως βλέπουμε στην εικόνα 17. Αν δεν είναι συνδεδεμένος με τον άλλο χρήστη θα μπορεί να δει μόνο το όνομα του, αν όμως είναι θα του εμφανίσει όλα τα στοιχεία. Η showDetails(user_id) θα τοποθετήσει το user_id σε ένα αντικείμενο Bundle και θα το στείλει στο Fragment Clicked_User όπου εκεί θα γίνει ο έλεγχος για το αν οι δυο χρήστες είναι συνδεδεμένοι ή όχι.



Εικόνα 17 "Clicked_User fragment σε περίπτωση που οι χρήστες δεν είναι συνδεδεμένοι μεταξύ τους"

Εικόνα 18 "Clicked_User fragment σε περίπτωση που οι χρήστες είναι συνδεδεμένοι μεταξύ τους"

Το Bundle χρησιμοποιείται πολύ συχνά για να στείλουμε δεδομένα σε άλλα Activities ή Fragments. Παρέχει μεθόδους putType() και getType() για να αποθηκεύει και να ανακτά δεδομένα.

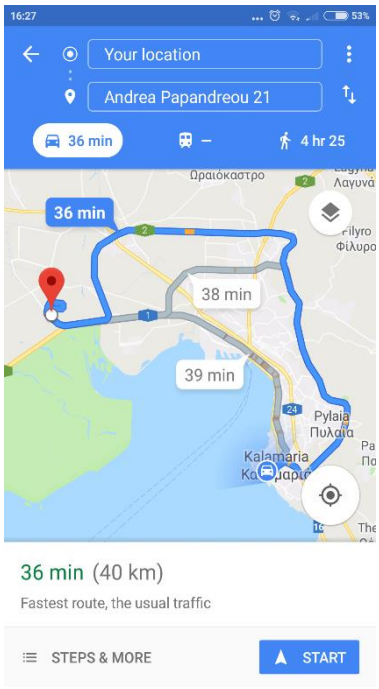
```
public void showDetails(String userID) {

     Bundle bundle=new Bundle();
     bundle.putString("user_id", userID);

     TabLayout tabs = (TabLayout)getActivity().findViewById(R.id.tabs);
     tabs.setVisibility(View.GONE);

    FrameLayout layout = (FrameLayout)getActivity().findView-
ById(R.id.main_frame);
    layout.setVisibility(View.VISIBLE);

     Fragment clicked_profile = new Clicked_User();
     clicked_profile.setArguments(bundle);
     FragmentManager manager = getActivity().getSupportFragmentManager();
     FragmentTransaction fragmentransaction = manager.beginTransaction();
         fragmentransaction.replace(R.id.main_frame, clicked_profile);
         fragmentransaction.addToBackStack(null);
         fragmentransaction.commit();
```

Κώδικας 30 "Η μέθοδος φόρτωσης δεδομένων του προφίλ του χρήστη"

Όσον αφορά το δεύτερο κουμπί που βρίσκεται στο Fragment posts και περιέχεται σε κάθε δημοσίευση (Εικόνα 12), με το που το πατήσει ο χρήστης θα ανοίξει η εφαρμογή google maps με τις συντεταγμένες που αποθηκεύτηκαν όταν πραγματοποιήθηκε η δημοσίευση. Οι συντεταγμένες βρίσκονται σε κάθε αντικείμενο τύπου Animal άρα αυτό που κάνουμε είναι να χρησιμοποιούμε το position για να τις πάρουμε από το αντικείμενο μέσω των μεθόδων getter (Κώδικας 31). Έπειτα, αφού επικοινωνήσει ο Presenter με το Model και πάρει τις δύο πληροφορίες που χρειαζόμαστε καλεί την μέθοδο show_location(lat, longi) για να μας εμφανίσει το σημείο όπου εντοπίστηκε το αδέσποτο ζώο όπως στην εικόνα 19.

```
@Override
public void get_coordinates(int position) {
    Animal animal = animals.get(position);
    double lat = animal.getLatitude();
    double longi = animal.getLongitude();
    mView.show_location(lat, longi);
}
```

Κώδικας 31 "Η μέθοδος φόρτωσης των συντεταγμένων του Presenter"



Εικόνα 19 “Εμφάνιση γεωγραφικού μήκους και πλάτους του ζώου στον χάρτη”

Το σημείο μόλις φορτώνεται μπαίνει αυτόματα ως destination από το σημείο που βρισκόμαστε. Η λογική πίσω από αυτό είναι αν κάποιος αναγνωρίσει ότι το ζώο που χάθηκε είναι δικό του να ξέρει βλέποντας στον χάρτη τουλάχιστον σε ποια περιοχή μπορεί να συχνάζει ή από που πέρασε. Ο κώδικας είναι ο εξής:

```
@Override
public void show_location(double lati, double longi) {

    String uri = "http://maps.google.com/maps?daddr=" + lati + "," +
longi + "(Dog was found here)" ;
    Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(uri));
    intent.setPackage("com.google.android.apps.maps");
    startActivity(intent);
}
```

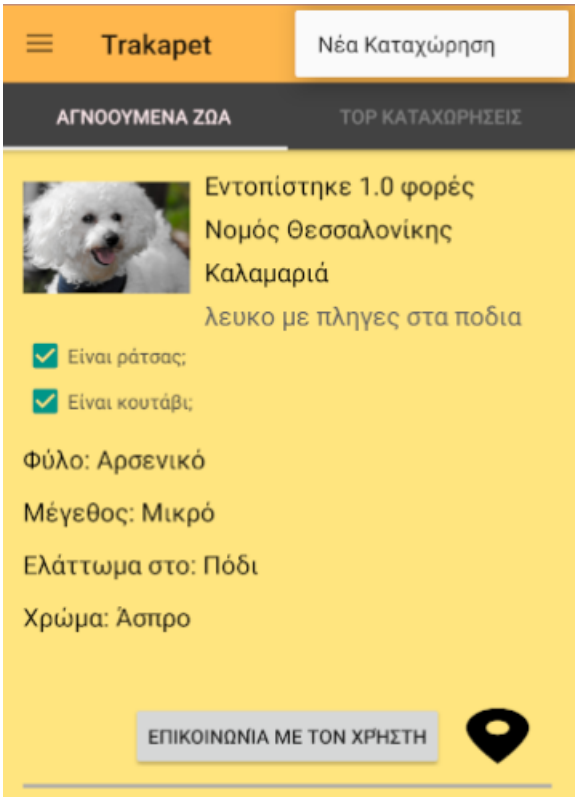
Κώδικας 32 “Φόρτωση του σημείου εντοπισμού ενός ζώου στον χάρτη”

Στο Android μπορούμε μέσα από μια εφαρμογή να συνδεθούμε σε μία άλλη, αυτή η σύνδεση γίνεται με το αντικείμενο Intent. Τα Intents είναι ασύγχρονα μηνύματα που επιτρέπουν μία εφαρμογή να χρησιμοποιήσει τα συστατικά μέρη μιας άλλης.

Intents χρησιμοποιούμε και σε άλλα σημεία στην εφαρμογή μας προκειμένου να ανοίξει ο χρήστης φακέλους μέσα από την εφαρμογή και να επιλέξει μία φωτογραφία ή για να ανοίξει η κάμερα. Υπάρχουν δύο τύποι Intent, ο explicit και ο implicit. Ο πρώτος προσδιορίζει με σαφήνεια το στοιχείο που θα κληθεί από το σύστημα Android χρησιμοποιώντας την κλάση java που θα χρησιμοποιηθεί ως αναγνωριστικό. Ο τελευταίος όμως που είναι και αυτός που χρησιμοποιήσαμε στην παραπάνω μέθοδο προσδιορίζει την ενέργεια που θα πραγματοποιηθεί και προαιρετικά τα δεδομένα που θα σταλθούν. Η ενέργεια που έχουμε προσδιορίζει είναι το “Intent.ACTION_VIEW” το οποίο σημαίνει ότι το Android θα ανοίξει μια ιστοσελίδα. Επειδή όμως το URI που θα χρησιμοποιήσουμε είναι μορφοποιημένο θα φορτώσει αυτόματα το google maps.

4.3.2 ΑΝΑΠΤΥΞΗ ΤΟΥ FRAGMENT NEW_POST

Αυτό το fragment είναι υπεύθυνο για την δημιουργία καινούργιας δημοσίευσης και ο χρήστης μπορεί να μεταβεί εκεί πατώντας στο σύμβολο με τις τρεις τελείες πάνω δεξιά (Εικόνα 18). Με το που πατάει ο χρήστης το σύμβολο αυτό βγαίνει ένα μικρό μενού (εικόνα 20). Αυτό το μενού είναι γνωστό ως μενού επιλογών και συναντάται συχνά στο Android.



Εικόνα 20 “Δημιουργία Νέας Καταχώρησης”

Με το που πατηθεί η επιλογή νέα καταχώρηση θα φορτωθεί το Fragment new_post μέσα στο FrameLayout που βρίσκεται στο xml του βασικού Activity. Ας δούμε πρώτα

τι περιλαμβάνει το xml αυτό και στη συνέχεια θα εξηγήσουμε τον κώδικα αναφέροντας πως ενσωματώθηκε το MVP.

Όπως και προηγουμένως, δημιουργούμε ένα interface που περιέχει δύο Interface, ένα για το View που είναι το new_post fragment και ένα για τον Presenter. Η διαδικασία είναι η ίδια με πριν, πρώτα δηλώνουμε τον Presenter και στην συνέχεια τον αρχικοποιούμε.

```
//anafora ston presenter auths ths klashs
private newPostPresenter myPresenter;
```

Κώδικας 33 “Δήλωση του newPostPresenter”

```
myPresenter = new newPostPresenter(this);
```

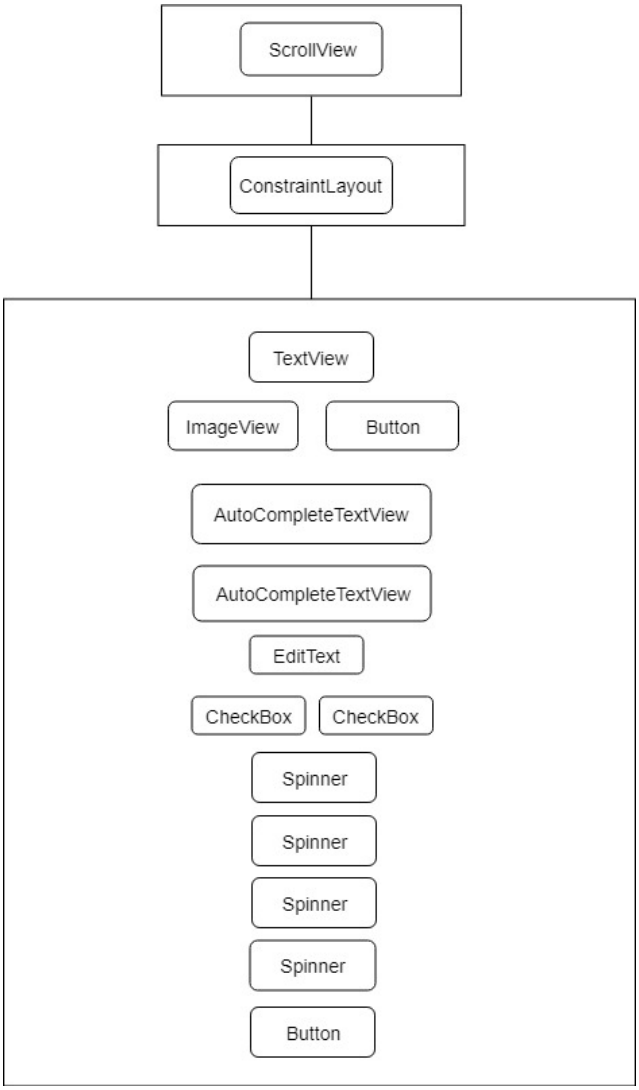
Κώδικας 34 “Αρχικοποίηση του newPostPresenter”

Έπειτα, κάνουμε implement το View του newPostContract.

```
public class new_post extends Fragment implements newPostContract.View
```

Κώδικας 35 “Τι υλοποιεί το Fragment new_post”

Το xml αυτού του fragment είναι ιδιαίτερα μεγάλο οπότε ας δούμε διαγραμματικά τι περιέχει παρακάτω στο Διάγραμμα 2.



Διάγραμμα 2 “Διάγραμμα του fragment new_post”

Το ScrollView είναι ένα layout το οποίο επιτρέπει στα υπόλοιπα views που βρίσκονται μέσα του να μπορούν να κυλιστούν. Επομένως αν δεν χωράνε όλα τα views στην οθόνη ο χρήστης μπορεί να κυλίσει προς τα κάτω για να δει τα υπόλοιπα. Όταν η εφαρμογή έχει περιεχόμενο που μπορεί να πιάνει χώρο παραπάνω από ότι το ύψος της οθόνης της συσκευής τότε το ScrollView είναι απαραίτητο.

Το ConstraintLayout μας επιτρέπει να στοιχίσουμε τα views που εμπεριέχονται χρησιμοποιώντας περιορισμούς για να καθορίσουμε τις σχέσεις που βασίζονται στη θέση μεταξύ τους. Ένας περιορισμός δηλαδή αναπαριστά την σχέση ή στοίχιση με ένα διαφορετικό view ή layout.

Εικόνα 21, 22 “Το layout της καταχώρησης ενός ζώου”

Σε αυτό το fragment ο χρήστης θα πρέπει να δηλώσει τα χαρακτηριστικά του ζώου που μόλις εντόπισε, με το που πατηθεί το κουμπί «Προσθήκη» θα υπολογιστούν οι συντεταγμένες γεωγραφικού μήκους και πλάτους.

Με το ανοίγει ο χρήστης αυτό το fragment θα του ζητηθεί άδεια για χρήση της κάμερας και άδεια για εγγραφή και πρόσβαση των αρχείων της συσκευής του. Αυτό είναι απαραίτητο στην περίπτωση που θέλει να δημοσιεύσει φωτογραφία του ζώου διότι απαιτείται άδεια από τον χρήστη προκειμένου να χρησιμοποιηθεί η εφαρμογή της κάμερας. Επιπλέον, οι υπόλοιπες δύο άδειες χρειάζονται γιατί μόλις τραβήξει ο χρήστης μια φωτογραφία θα πρέπει να αποθηκευτεί (write) στην συσκευή του και να διαβαστεί (read) για να μπορεί να φορτωθεί και στο ImageView. Πριν το επίπεδο API 23 η απαίτηση να ζητείται άδεια από τους χρήστες προκειμένου να χρησιμοποιηθούν λειτουργίες της συσκευής δεν υπήρχε. Τώρα οι προγραμματιστές είναι υποχρεωμένοι να κάνουν διάφορους ελέγχους προκειμένου να επιβεβαιώσουν ότι έχουν πάρει τις άδειες. Διαφορετικά, η εφαρμογή θα κολλήσει διότι θεωρείται παραβίαση. Σε περίπτωση που ο χρήστης δεν μας επιτρέψει να χρησιμοποιήσουμε την κάμερα και την πρόσβαση στην συσκευή θα πρέπει να του απαγορεύσουμε τη δυνατότητα να προσθέσει μία φωτογραφία. Γι’ αυτόν ακριβώς τον λόγο το κουμπί «Προσθήκη εικόνας» είναι ρυθμισμένο στον xml ως disabled. Αυτό σημαίνει ότι δεν ανταποκρίνεται στις ενέργειες του χρήστη επομένως δεν μπορεί να χρησιμοποιήσει

τις λειτουργίες που θα του παρείχαμε αν μας έδινε τις απαραίτητες άδειες. Μόλις δοθεί η άδεια γίνεται enabled. Όλες οι άδειες που θα ζητηθούν από τον χρήστη θα πρέπει να συμπεριληφθούν στο αρχείο Manifest.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.CAMERA" />
 <uses-permission android:name="android.permission.FLASHLIGHT" />
 <uses-feature android:name="android.hardware.camera" android:re-
quired="true" />
 <uses-feature android:name="android.hardware.camera.flash"
     android:required="false" />
 <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
/>
 <uses-permission android:name="android.permission.MANAGE_DOCUMENTS" />
 <uses-permission android:name="android.permission.WRITE_EXTERNAL_STOR-
AGE"/>
 <uses-permission android:name="android.permission.MEDIA_EXTERNAL_STOR-
AGE"/>

 <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
/>
 <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
/>
 <uses-permission android:name="android.permission.ACCESS_COARSE_LOCA-
TION" />
 <uses-feature android:name="android.hardware.location.gps"/>
 <uses-feature android:name="android.hardware.location.network"/>
```

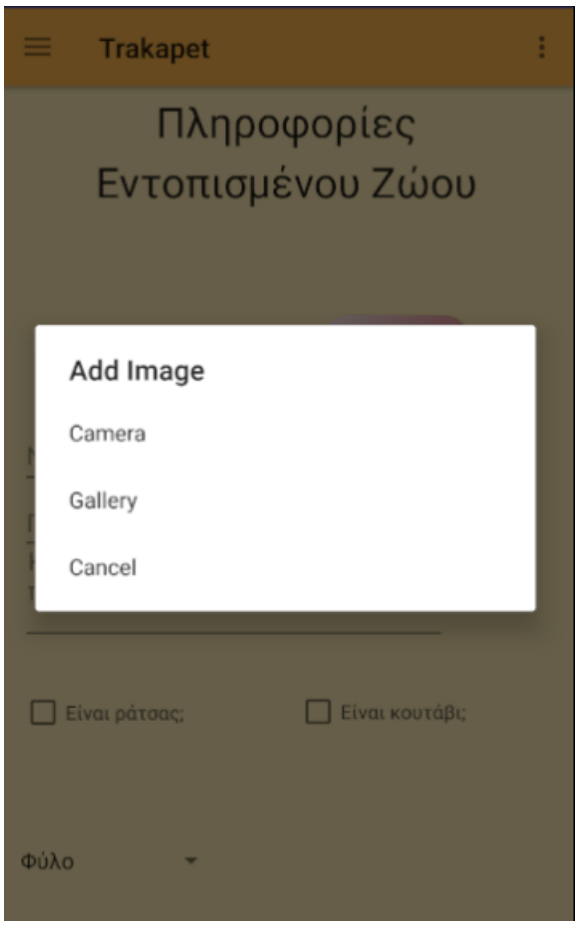
Κώδικας 36 “Οι απαραίτητες άδειες στο αρχείο Manifest”

Σημαντικό κομμάτι της εφαρμογής είναι η λήψη φωτογραφιών και τοποθέτηση σε ένα ImageView ή CircleImageView. Η υλοποίηση που περιλαμβάνεται σε αυτό το fragment είναι κοινή και για το ProfileFragment. Με το που πατηθεί λοιπόν το κουμπί καλείται η μέθοδος SelectImage().

```
btnCamera.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        SelectImage();
    }
});
```

Κώδικας 37 “Ο ClickListener του κουμπιού φόρτωσης φωτογραφιών”

Δημιουργούμε προγραμματιστικά ένα AlertDialog το οποίο παρέχει ορισμένες επιλογές στον χρήστη όπως βλέπουμε στην εικόνα 23. Αναλόγως με την επιλογή που θα επιλέξει θα δημιουργηθεί και το ανάλογο αντικείμενο Intent το οποίο όπως αναφέραμε προηγουμένως εκτελεί λειτουργίες από άλλες εφαρμογές που βρίσκονται στο Android. Στην περίπτωση που επιλεχθεί η κάμερα θα χρησιμοποιήσουμε το Intent “MediaStore.ACTION_IMAGE_CAPTURE” το οποίο δηλώνει ότι θέλουμε να χρησιμοποιήσουμε την εφαρμογή της κάμερας. Με την μέθοδο startActivityForResult() ανοίγουμε ένα activity το οποίο θα χρησιμοποιήσει την εφαρμογή που δηλώθηκε στο Intent. Πριν όμως χρησιμοποιήσουμε την κάμερα θα πρέπει να δημιουργήσουμε το Uri του αρχείου. Αν θέλουμε να παρέχουμε υποστήριξη και σε συσκευές με επίπεδο API από 24 και πάνω θα πρέπει να προσθέσουμε και μια διαφορετική υλοποίηση διαφορετικά προκύπτει ένα Exception “FileUriExposedException”. Αυτό προκύπτει σε περίπτωση που εκθέσουμε το Uri έξω από την εφαρμογή μας. Αυτό συμβαίνει διότι ουσιαστικά ανοίγουμε μια άλλη εφαρμογή, την εφαρμογή της κάμερας. Για να προληφθεί το Exception θα πρέπει να προσδιορίσουμε έναν FileProvider στο manifest (Κώδικας 38) όπου θα αναφέρουμε τους φακέλους που θα μοιραστούμε. Αυτό που κάνει δηλαδή ο Provider είναι να επιτρέπει τον διαμοιρασμό των αρχείων μεταξύ των εφαρμογών.



Εικόνα 23 “Επιλογές προσθήκης φωτογραφίας του ζώου πριν τη δημοσίευση”

```
<provider
    android:name="android.support.v4.content.FileProvider"
    android:authorities="${applicationId}.provider"
    android:exported="false"
    android:grantUriPermissions="true">
    <meta-data
        android:name="android.support.FILE_PROVIDER_PATHS"
        android:resource="@xml/provider_paths" />

</provider>
```

Κώδικας 38 “Προσδιορισμός του FileProvider στο αρχείο manifest”

```
<?xml version="1.0" encoding="utf-8"?>
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <external-path name="external_files" path="." />
</paths>
```

Κώδικας 39 “Προσδιορισμός του μονοπατιού του FileProvider”

Στον Κώδικα 41 για να πάρουμε το Uri του αρχείου καλείται η μέθοδος getFile(int type) του newPostPresenter στην οποία δημιουργούμε το αρχείο μέσα στον φάκελο "Pet_photos". Σε περίπτωση που ο φάκελος δεν υπάρχει τον δημιουργούμε ενώ κάθε φορά δημιουργούμε ένα αρχείο με μοναδικό όνομα μιας και ενσωματώνουμε στην ονομασία του την ημερομηνία και την ακριβή ώρα που δημιουργήθηκε το αρχείο. Τις μεθόδους που απαιτούν λογική τις έχουμε μέσα στον Presenter.

Αν επιλεχθεί το Gallery τότε το Intent (Κώδικας 40) θα πρέπει να του ανοίξει τους φακέλους με τις φωτογραφίες για να επιλέξει κάποια από εκεί. Με την μέθοδο setType() προσδιορίζουμε τον τύπο των δεδομένων που θα επιλεχθεί. Γενικά ό,τι προστίθεται ως παράμετρος στη μέθοδο είναι τύπου MIME και πιο συγκεκριμένα μπορούμε να είναι εικόνες, βίντεο ήχο, αρχεία κειμένου κ.ά. Μπορεί να προσδιοριστεί ακόμα πιο ειδικά βάζοντας τον τύπο του αρχείου αλλά αυτό που επιθυμούμε εμείς από τον χρήστη είναι να φορτώσει μια εικόνα οποιουδήποτε τύπου οπότε βάζουμε απλά τον τύπο δεδομένων που είναι εικόνα και το αστεράκι που υποδεικνύει ότι μπορεί να είναι ένας οποιοσδήποτε τύπος εικόνας. Θέτοντας ως Action το ACTION_GET_CONTENT ανοίγει μια εφαρμογή η οποία παρέχει μια διεπαφή χρήστη οπού μπορούμε να επιλέξουμε αρχεία.

```
private void SelectImage() {

    final CharSequence[] items = {"Camera", "Gallery", "Cancel"};

  AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
  builder.setTitle("Add Image");
  builder.setItems(items, new DialogInterface.OnClickListener() {
  @Override
   public void onClick(DialogInterface dialogInterface, int i) {


    if (items[i].equals("Camera")) {

                //anoigoume to application gia tin camera
    if (Build.VERSION.SDK_INT >= 24)
    file_uri = FileProvider.getUriForFile(getContext(),
                BuildConfig.APPLICATION_ID + ".provider",
myPresenter.getFile(1));
    else file_uri = myPresenter.getOutputMediaFileUri(1);
    Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
                intent.putExtra(MediaStore.EXTRA_OUTPUT, file_uri);
                startActivityForResult(intent, REQUEST_CAMERA);

            } else if (items[i].equals("Gallery")) {
                Intent intent = new Intent();
                intent.setType("image/*");
                intent.setAction(Intent.ACTION_GET_CONTENT);
                startActivityForResult(Intent.createChooser(intent, "Se-
lect File"), SELECT_FILE);

            } else if (items[i].equals("Cancel")) {

                dialogInterface.dismiss();
            }

        }

    });
    builder.show();

}
```

Κώδικας 40 “Η μέθοδος επιλογής ή λήψης εικόνας”

Σε περίπτωση που δεν είναι προκαθορισμένη επιλογή θα βγουν διάφορες σχετικές εφαρμογές για να επιλέξει ο χρήστης αυτή που τον βολεύει. Όπως παρατηρούμε στον Κώδικα 42 χρησιμοποιούμε τη μέθοδο startActivityForResult() η οποία ξεκινάει καινούργιο Activity και περιμένει να λάβει δεδομένα και μετά να επιστρέψει στο δικό μας Activity. Στην περίπτωση που τελικά ο χρήστης αποφασίσει ότι δεν θέλει να ανεβάσει κάποια φωτογραφία απλά θα πατήσει Cancel και το AlertDialog θα κλείσει. Η μεταβλητή REQUEST_CAMERA και SELECT_FILE είναι τύπου Integer και είναι δύο αριθμοί οι οποίοι θα μας βοηθήσουν να εντοπίσουμε τι ενέργεια πραγματοποιήθηκε μέσα το Activity ώστε να τρέξουμε τις κατάλληλες εντολές.

```
public Uri getOutputMediaFileUri(int type) {

    return Uri.fromFile(getFile(type));
}

//return image
public File getFile(int type) {

    File folder = new File(Environment
            .getExternalStoragePublicDirectory(Environment.DIREC-
TORY_PICTURES),
            IMAGE_DIRECTORY_NAME);

    if (!folder.exists()) {
        if (!folder.mkdirs()) {
            Log.d(IMAGE_DIRECTORY_NAME, "Oops! Failed create "
                    + IMAGE_DIRECTORY_NAME + " directory");
            return null;
        }
    }

    // Create a media file name
    String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss",
            Locale.getDefault()).format(new Date());
    File mediaFile;
    if (type == 1) {
        mediaFile = new File(folder.getPath() + File.separator
                + "IMG_" + timeStamp + ".jpg");
    } else {
        return null;
    }
    return mediaFile;

}
```

Κώδικας 41 “Η μέθοδος φόρτωσης του μονοπατιού ενός αρχείου και δημιουργίας αρχείων φωτογραφιών”

Όταν ο χρήστης ολοκληρώσει τις ενέργειες του στο Activity και επιστρέψει τότε το σύστημα καλεί την μέθοδο onActivityResult(). Ο αριθμός-αναγνωριστικό θα σταλθεί σε αυτήν. Η μέθοδος onActivityResult() περιέχει τρεις παραμέτρους, το request code που εξηγήσαμε, το resultCode που είτε έχει την τιμή RESULT_OK αν ολοκληρώθηκαν όλες οι διεργασίες στο Activity με επιτυχία είτε την τιμή RESULT_CANCELED αν ο χρήστης ακύρωσε τη διαδικασία ή κάτι πήγε στραβά. Τέλος, έχει ένα αντικείμενο Intent το οποίο περιέχει τα δεδομένα από το Activity δηλαδή στην περίπτωση μας το αρχείο που επιλέχθηκε.

Εάν η φωτογραφία είναι από λήψη με την κάμερα τότε καλείται η μέθοδος previewCapturedImage() η οποία το μόνο που κάνει είναι να φορτώνει το Uri στο ImageView χρησιμοποιώντας τη βιβλιοθήκη Glide.

```
public void onActivityResult(int requestCode, int resultCode, Intent
data) {
    super.onActivityResult(requestCode, resultCode, data);


    if (resultCode != RESULT_CANCELED) {

        if (requestCode == REQUEST_CAMERA) {

            previewCapturedImage();
        } else if (requestCode == SELECT_FILE && data != null) {

            Uri selectImage = data.getData();
            uni_uri = selectImage;
            try {
        bitmap = handleSamplingAndRotationBitmap(getContext(), selec-
tImage);
            } catch (IOException e) {
                e.printStackTrace();
            }
            ivImage.setImageBitmap(bitmap);
```

Κώδικας 42 “Η μέθοδος onActivityResult”

Αν όμως επιλέχθηκε ένα αρχείο από φάκελο τα πράγματα είναι λίγο πιο περίπλοκα. Αρχικά θα πάρουμε το Uri της φωτογραφίας που επιλέχθηκε και στη συνέχεια θα το στείλουμε στη μέθοδο handleSamplingAndRotation() όπου θα συμπιέσουμε την εικόνα προκειμένου να έχει μικρότερη μνήμη και να μην την επιβαρύνει (Κώδικας 43). Με την μέθοδο decodeStream() μειώνουμε το μέγεθος της φωτογραφίας στη μνήμη και όχι στον δίσκο, δηλαδή δεν την αλλάζουμε μέγεθος.

Άξιο προσοχής είναι το γεγονός ότι οι φωτογραφίες στον αποθηκευτικό χώρο της συσκευής είναι συμπιεσμένες αλλά όταν φορτώνονται στη μνήμη δεν είναι. Επομένως, μια καλή τεχνική είναι να μειώνουμε το μέγεθος μνήμης της προκειμένου να μην υπερβούμε την μέγιστη μνήμη του συστήματος Android.

```
Bitmap handleSamplingAndRotationBitmap(Context context, Uri selectedI-
mage)
        throws IOException {
    int MAX_HEIGHT = 1024;
    int MAX_WIDTH = 1024;

    // First decode with inJustDecodeBounds=true to check dimensions
    final BitmapFactory.Options options = new BitmapFactory.Options();
    options.inJustDecodeBounds = true;
    InputStream imageStream = context.getContentResolver().openIn-
putStream(selectedImage);
    BitmapFactory.decodeStream(imageStream, null, options);
    imageStream.close();

    // Calculate inSampleSize
    options.inSampleSize = myPresenter.calculateInSampleSize(options,
MAX_WIDTH, MAX_HEIGHT);

    // Decode bitmap with inSampleSize set
    options.inJustDecodeBounds = false;
    imageStream = context.getContentResolver().openInputStream(selectedI-
mage);
    Bitmap img = BitmapFactory.decodeStream(imageStream, null, options);

    img = rotateImageIfRequired(img, selectedImage, context);
    return img;
}
```

Κώδικας 43 “Η μέθοδος μείωσης αρχείου εικόνας στη μνήμη”

Αφού λοιπόν συμπληρώσει ο χρήστης τα στοιχεία του αδέσποτου ζώου τότε θα πάει να πατήσει το κουμπί «Προσθήκη». Με το που πατηθεί ο Presenter θα επικοινωνήσει με το Firebase και αφού πρώτα βεβαιωθεί ότι ο χρήστης έχει συμπληρωμένα τα στοιχεία του προφίλ του θα εισάγει την καινούργια δημοσίευση.

```
btnSumbmit.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        myPresenter.check_ProfileInfo();
        }
});
```

Κώδικας 44 “Ο ClickListener του κουμπιού προσθήκης δημοσίευσης”

ΕΠΙΛΟΓΟΣ

Σε αυτό το κεφάλαιο αναλύθηκε το MVP πρώτα σε γενικό επίπεδο εξηγώντας το κάθε επίπεδο του ξεχωριστά και εν συνεχεία μελετήσαμε πως ενσωματώθηκε σε ορισμένα βασικά κομμάτια της εφαρμογής. Παράλληλα εξηγήθηκε και ο κώδικας σε αρκετά σημεία κάνοντας έτσι πιο κατανοητή τη ροή της εφαρμογής.

ΚΕΦΑΛΑΙΟ 5
Η ΒΑΣΗ ΔΕΔΟΜΕΝΩΝ FIREBASE

ΕΙΣΑΓΩΓΗ
Σε αυτό το κεφάλαιο θα γίνει κατανοητό τι υπηρεσίες προσφέρει το Firebase με τα dependencies που χρησιμοποιήσαμε και θα αναλυθεί το περιεχόμενο ορισμένων σημαντικών μεθόδων της εφαρμογής για να καταλάβουμε πως λειτουργεί. Επιπλέον, θα δούμε τη δομή της εφαρμογής σε json document ή/και δέντρο-διαγράμματα για κάθε περίπτωση χρήστη ώστε να καταλάβουμε καλύτερα και τη δομή της βάσης καθώς και να γίνει φανερό σε πόσο μεγάλο βαθμό χρησιμοποιείται το Firebase στην εφαρμογή.

5.1 ΛΙΓΑ ΛΟΓΙΑ ΓΙΑ ΤΟ FIREBASE
Το Firebase είναι μια πλατφόρμα ανάπτυξης εφαρμογών για κινητές και web εφαρμογές η οποία δημιουργήθηκε από την Firebase το 2011 και μετέπειτα αγοράστηκε από την Google το 2014. Είναι μια βάση NoSQL και πραγματικού χρόνου και παρέχει ένα backend ως υπηρεσία. Η υπηρεσία παρέχει στους προγραμματιστές ένα API που επιτρέπει τον συγχρονισμό των δεδομένων μεταξύ των πελατών και την διατήρηση των δεδομένων στο cloud. Οι υπηρεσίες που προσφέρει κάνουν τη ζωή ενός προγραμματιστή πιο εύκολη.

5.2 ΥΠΗΡΕΣΙΕΣ ΤΟΥ FIREBASE
Στην εφαρμογή μου για να μπορέσω να υλοποιήσω το κομμάτι της εγγραφής και της αυθεντικοποίησης του χρήστη χρησιμοποίησα τις backend υπηρεσίες, SDKs και έτοιμες βιβλιοθήκες που παρέχει το Firebase. Παρέχει διάφορους τρόπους αυθεντικοποίησης, αυθεντικοποίηση με χρήση κωδικών, email, αριθμών τηλεφώνων καθώς και λογαριασμών σε τρίτους παρόχους όπως Google, Facebook, Twitter κ.ά. Υπάρχουν δύο λύσεις προκειμένου να υλοποιήσουμε την αυθεντικοποίηση αναλόγως με τους τρόπους αυθεντικοποίησης που θέλουμε να έχει η εφαρμογή μας. Η πρώτη λύση ονομάζεται FirebaseUI Auth και μας προσφέρει μια ολοκληρωμένη λύση αυθεντικοποίησης με drop-in γραφικά και πολύ εύκολη διαχείριση της ροής των επιλογών. Η δεύτερη λύση είναι αυτή που επέλεξα να υλοποιήσω και ονομάζεται αυθεντικοποίηση Firebase SDK. Μπορούμε να συμπεριλάβουμε όσους τρόπους αυθεντικοποίησης θέλουμε και είναι μια πιο «χειροκίνητη» λύση.

Μία ακόμα πολύ χρήσιμη υπηρεσία είναι το Firebase Core που παρέχει στατιστικά στοιχεία χρήσης της εφαρμογής. Είναι γνωστό και ως Google Analytics και είναι ιδιαίτερα χρήσιμο στην περίπτωση που η εφαρμογή μας είναι διαθέσιμη στην αγορά και χρειάζεται να δούμε την ενασχόληση των χρηστών με την εφαρμογή. Παρακολουθώντας τα στατιστικά μπορούμε να καταλάβουμε πως συμπεριφέρονται οι χρήστες και αυτό μας επιτρέπει να πάρουμε καλύτερες αποφάσεις όσον αφορά το marketing της εφαρμογής.

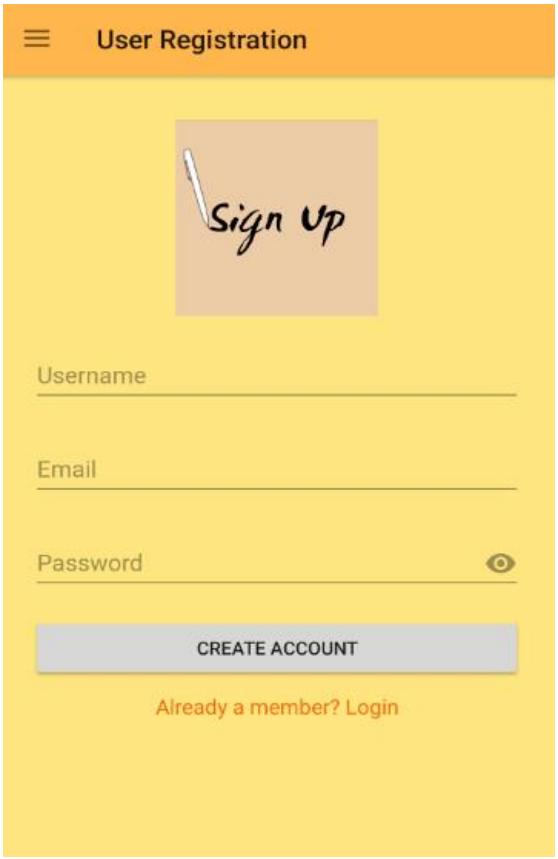
Στην εφαρμογή χρησιμοποιούμε σε δύο fragments το ανέβασμα φωτογραφιών οπότε η υπηρεσία της cloud αποθήκευσης που παρέχει το Firebase ήτανε ιδιαίτερης σημασίας για την εφαρμογή. Έχουμε τη δυνατότητα να αποθηκεύσουμε εικόνες, αρχεία ήχου, βίντεο καθώς και άλλο περιεχόμενο που δημιουργεί ο χρήστης.

5.3 ΛΕΙΤΟΥΡΓΙΕΣ ΤΗΣ ΕΦΑΡΜΟΓΗΣ ΒΑΣΙΣΜΕΝΕΣ ΣΤΟ FIREBASE

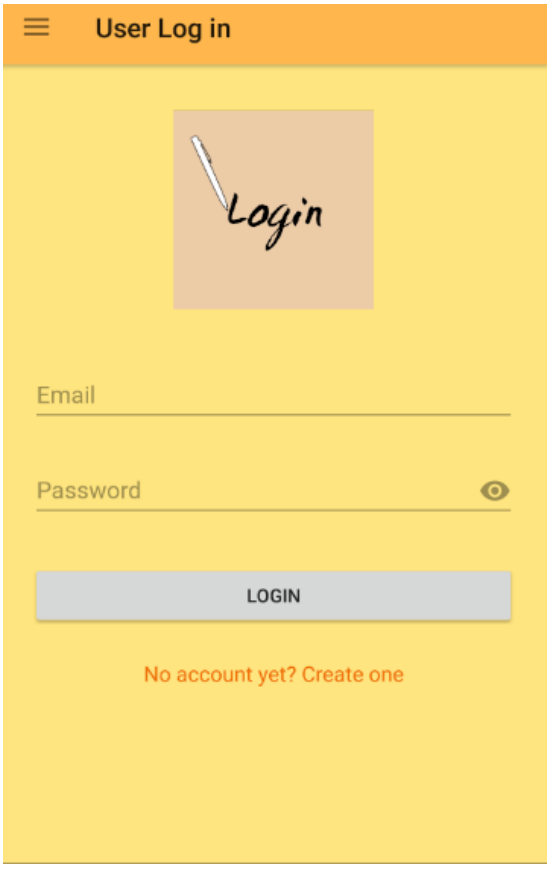
Για την καλύτερη κατανόηση του κώδικα θα επεξηγηθούν ορισμένα κομμάτια κώδικα της εφαρμογής τα οποία χρησιμοποιούν το Firebase.

5.3.1 ΕΓΓΡΑΦΗ ΤΟΥ ΧΡΗΣΤΗ ΣΤΗΝ ΕΦΑΡΜΟΓΗ

Για να μπορέσει κάποιος να δηλώσει κάποιο αδέσποτο ή χαμένο ζώο που εντόπισε ή να έρθει σε επαφή με κάποιον άλλο χρήστη θα πρέπει να έχει λογαριασμό. Για να δημιουργήσει λογαριασμό αρκεί να συμπληρώσει τα στοιχεία του και να πατήσει “Create Account”.



Εικόνα 24 “Fragment Εγγραφής Νέου Χρήστη”



Εικόνα 25 “Fragment Σύνδεσης Χρήστη”

```
_signupButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //initialize Firebase
        presenter.initFirebase();
        presenter.signup(get_Name(),get_Email(),get_Password());
    }
});
```

Κώδικας 45 "Ο ClickListener του κουμπιού εγγραφής χρήστη"

Στην initFirebase() θα αρχικοποιήσουμε δύο μεταβλητές που δηλώθηκαν νωρίτερα μέσα στον Presenter.

```
private DatabaseReference mDatabaseReference;
private FirebaseAuth mAuth;
```

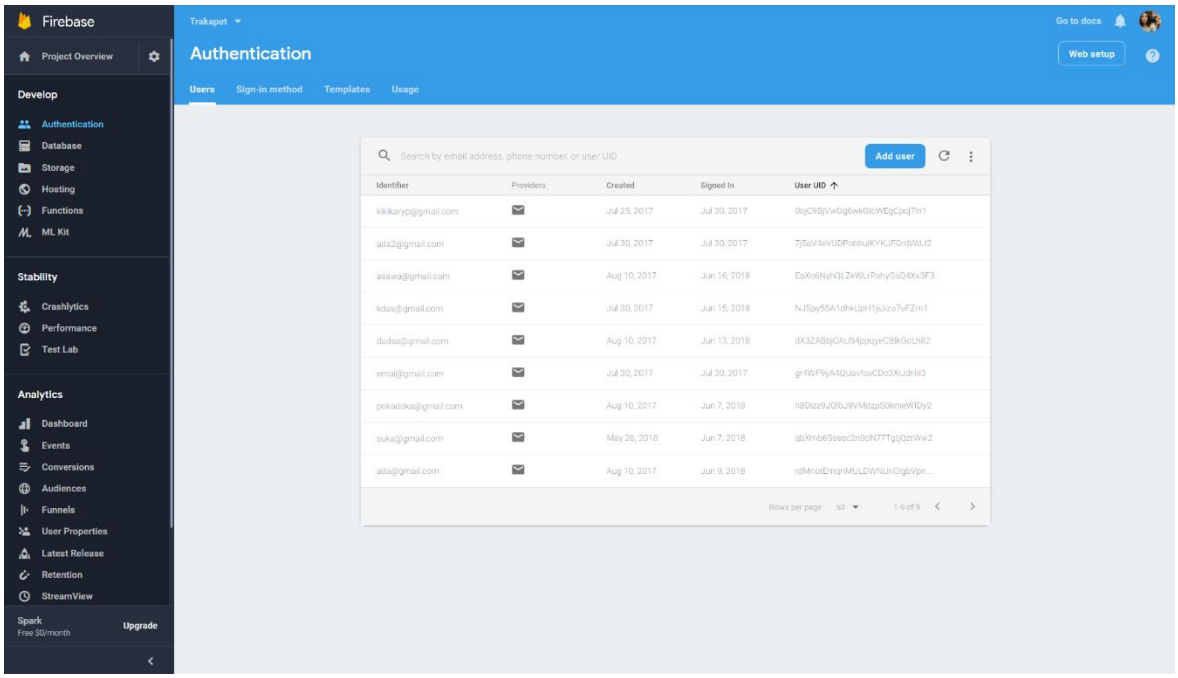
Κώδικας 46 "Βασικές μεταβλητές του Firebase"

Η πρώτη μεταβλητή αναπαριστά μία συγκεκριμένη τοποθεσία μέσα στην βάση μας και μπορεί να χρησιμοποιηθεί είτε για ανάγνωση είτε για εγγραφή δεδομένων. Είναι το ξεκίνημα για κάθε επικοινωνία με τη βάση, μιας και σε όλες τις μεθόδους που χρησιμοποιούν το Firebase είτε διαβάζουμε δεδομένα είτε γράφουμε καινούργια. Η δεύτερη μεταβλητή είναι το εναρκτήριο βήμα για να πραγματοποιηθούν όλες οι διαδικασίες αυθεντικοποίησης.

```
@Override
public void initFirebase() {

    //Get Firebase auth instance
    mAuth = FirebaseAuth.getInstance();
    mDatabaseReference =   FirebaseDatabase.getInstance().getRefer-
ence().child("users");
}
```

Κώδικας 47 "Η μέθοδος αρχικοποίησης βασικών μεταβλητών του Firebase"

Χρησιμοποιώντας την μέθοδο createUserWithEmailAndPassWord() του (Κώδικας 48) που παρέχει το Firebase μπορούμε να δημιουργήσουμε έναν καινούργιο χρήστη δίνοντας το email και τον κωδικό του. Το όνομα το ζητάμε για άλλες λειτουργίες τις εφαρμογής και δεν χρειάζεται υποχρεωτικά γι' αυτό και δεν το περνάμε ως παράμετρο στην μέθοδο του Firebase. Ο κάθε χρήστης που κάνει εγγραφή έχει ένα μοναδικό αναγνωριστικό. Οι εγγεγραμμένοι χρήστες εμφανίζονται στο παράθυρο Authentication του Firebase μαζί με τα mail τους καθώς και τι είδους εγγραφή κάνανε, πότε συνδέθηκαν τελευταία φορά και πότε δημιούργησαν τον λογαριασμό τους όπως βλέπουμε στην εικόνα 26.



Εικόνα 26 "Διεπαφή αυθεντικοποίησης Χρηστών του Firebase"

Εάν η εγγραφή πραγματοποιηθεί με επιτυχία θα εισάγουμε στο σχήμα της βάσης μας τα στοιχεία του χρήστη. Με την μέθοδο getUid() παίρνουμε το αναγνωριστικό του χρήστη και το εισάγουμε στη βάση χρησιμοποιώντας την αναφορά που ορίσαμε πριν. Αν η αναφορά δεν υπάρχει τότε δημιουργείται αυτόματα. Έτσι λοιπόν, τοποθετούμε στον κόμβο users το id του χρήστη και έπειτα το id του χρήστη θα έχει ως παιδιά το όνομα, το mail, και τον κωδικό του χρήστη. 'Ετσι λοιπόν, το σχήμα της βάσης μέχρι στιγμής έχει την δομή της εικόνας 27.

```
Task<AuthResult> task = mAuth.createUserWithEmailAndPassword(email,pass-
word);
task.addOnCompleteListener(new OnCompleteListener<AuthResult>() {
    @Override
    public void onComplete(@NonNull Task<AuthResult> task) {
       mView.show_Toast("createUserWithEmail:onComplete" + task.isSuccess-
ful());
        mView.hide_progressDialog();

        if (task.isSuccessful()) {
            String user_id = mAuth.getCurrentUser().getUid();
            DatabaseReference curr_db_user =  mDatabaseRefer-
ence.child(user_id);
            curr_db_user.child("name").setValue(_name);
            curr_db_user.child("email").setValue(_email);
            curr_db_user.child("password").setValue(_password);
            mView.show_Toast("Authentication succeeded." + task.getExcep-
tion());
            mView.dismiss_progressDialog();
            mView.onSignupSuccess();
        } else {
            mView.onSignupFailed();
        }
    }
});
```

Κώδικας 48 “Η built-in μέθοδος εγγραφής χρήστη στο Firebase”

Εικόνα 27 “JSON document της δομής ενός εγγεγραμμένου χρήστη στη βάση”

5.3.2 ΣΥΝΔΕΣΗ ΤΟΥ ΧΡΗΣΤΗ ΣΤΗΝ ΕΦΑΡΜΟΓΗ

Η σύνδεση του χρήστη γίνεται με μία built-in μέθοδο, την signInWithEmailAndPassword(). Το να αποθηκεύσουμε τον κωδικό του χρήστη δεν είναι απαραίτητο γιατί η υλοποίηση της σύνδεσης υπάρχει ήδη.

```
public void sign_in(String email, String password) {

    mAuth = FirebaseAuth.getInstance();
    mAuth.signInWithEmailAndPassword(email, password)
            .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    mView.hide_progressDialog();

                    if (task.isSuccessful()) {
                        mView.dismiss_progressDialog();
                        mView.update_UI();
                    } else {
                        mView.onLoginFailed();
                    }

                }
            });
}
```

Κώδικας 49 "Η μέθοδος σύνδεσης του χρήστη στο Firebase"

Ο λόγος που αποθήκευσα τον κωδικό στη βάση είναι για πρακτικούς λόγους και δεν θα το κάναμε ποτέ σε μία εφαρμογή η οποία διατίθεται στην αγορά. Αν η σύνδεση γίνει με επιτυχία του φορτώνεται το MainActivity και ο χρήστης μπορεί να έχει πρόσβαση σε κάθε λειτουργία της εφαρμογής.

5.3.3 ΚΑΤΑΧΩΡΗΣΗ ΜΙΑΣ ΝΕΑΣ ΔΗΜΟΣΙΕΥΣΗΣ ΣΤΗ ΒΑΣΗ

Το κομμάτι αυτό ανήκει στο Fragment new_post που εξηγήσαμε στο προηγούμενο κεφάλαιο, τώρα όμως θα αναφερθούμε στο κομμάτι του Firebase. Είχαμε αναφέρει ότι γίνεται έλεγχος για το αν ο χρήστης που κάνει τη δημοσίευση έχει συμπληρώσει το προφίλ του. Ας δούμε λοιπόν πως γίνεται αυτός ο έλεγχος.

Αν ο χρήστης έχει αποθηκεύσει τις πληροφορίες του προφίλ του τότε θα πρέπει να έχει ένα ακόμα παιδί-κόμβο μέσα στο id του το οποίο ονομάζεται contact_info (Κώδικας 50). Η ίδια υλοποίηση εφαρμόζεται και για την περίπτωση που ένας χρήστης πάει να επικοινωνήσει με κάποιον άλλον διότι ο έλεγχος αφορά και πάλι τη συμπλήρωση του προφίλ του, προκειμένου να μπορεί να στείλει αίτημα σε άλλον χρήστη. Αυτός ο κόμβος εφόσον έχει συμπληρωθεί το προφίλ του μπορεί να έχει μέγιστο άλλα 5 παιδιά και ελάχιστο 2 αναλόγως με το πόσα στοιχεία έχει συμπληρώσει. Το ελάχιστο είναι 2 γιατί όταν πάει να συμπληρώσει τα στοιχεία του δύο από τα στοιχεία EditText είναι υποχρεωτικό να συμπληρωθούν. Έστω λοιπόν ότι έχουν συμπληρωθεί όλα τα πεδία, το JSON document είναι όπως στην εικόνα 28.

```
@Override
public void check_ProfileInfo() {
    //to uid tou ekastote xrhsth
    final String current_user_id = mAuth.getCurrentUser().getUid();

    databaseReference = FirebaseDatabase.getInstance().
            getReference("users").
            child(current_user_id).
            child("contact_info");

    databaseReference.addValueEventListener(new com.google.firebase.data-
base.ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            if (!(dataSnapshot.exists())) {
                mView.showErrorToast("To publish a post please fill your
profile info :) ");
            } else {
                mView.submit();
            }
        }
        @Override
        public void onCancelled(DatabaseError databaseError) {

        }
    });
}
```

Κώδικας 50 “Η μέθοδος ελέγχου προφίλ του χρήστη ελέγχοντας τη βάση”


```
{
  "users": {
    "EaXo6NyhQLZkWLrPahyGsQ4Xx3F3": {
      "contact_info": {
        "area": "Καλαμαριά",
        "email": "kiki@gmail.gr",
        "image_url": "https://firebasestorage.googleapis.com/v0/b/trakapet-20c23.appspot.com/o
          /Profile_photos%2FEaXo6NyhQLZkWLrPahyGsQ4Xx3F3?alt=media&token=bd1f9022-6fa5-425e-b043
          -b1ab0f145ae2",
        "name": "κικη",
        "phone_n": "2310452123"
      },
      "email": "asawa@gmail.com",
      "name": "papep",
      "password": "asawada"
    }
  }
}
```


Εικόνα 28 “JSON document μετά από συμπλήρωση του προφίλ ενός χρήστη”

Όσον αφορά την εισαγωγή μιας δημοσίευσης στη βάση, η λογική που ακολουθείται είναι η ίδια μιας και παίρνουμε πρώτα την αναφορά στην ρίζα της βάσης που είναι το users και μετά στο κόμβο-παιδί που ονομάζεται “pets” θα γίνει εισαγωγή της δημοσίευσης. Στον κόμβο pets εισέρχονται όλες οι δημοσιεύσεις και αν ο χρήστης θέλει να δει τις δικές του καταχωρήσεις φιλτράρονται τα αποτελέσματα με βάση το user_id του χρήστη.

Δημιουργούμε πρώτα ένα αντικείμενο και εισάγουμε όλα τα στοιχεία που εισήχθησαν από τον χρήστη μέσω των setter μεθόδων. Η καινούργια δημοσίευση θα μπει σε ένα παιδί του pets που θα έχει το όνομα pet μαζί με ένα μοναδικό τυχαίο αναγνωριστικό. Το αντικείμενο Animal θα το εισάγουμε μετά στον καινούργιο κόμβο.Το Firebase καθιστά πολύ εύκολη την ανάγνωση των POJOs (plain old java objects) με τις ικανότητες σειριοποίησης που παρέχει. Προκειμένου να εισάγονται τα αντικείμενο με τα ονόματα κόμβων που επιθυμούμε θα πρέπει τα ονόματα των μεταβλητών που έχουμε ορίσει στην κλάση Animal να είναι τα ονόματα των παιδιών που θα έχει το κάθε παιδί του κόμβου pet. Επιπλέον, πρέπει να είναι ένας έγκυρος τύπος δεδομένων JSON. Οι τύποι μπορούν να είναι όπως αυτοί στον πίνακα 3. Επιπλέον, θα πρέπει να έχουμε δημιουργήσει έναν άδειο δομητή και μεθόδους setter και getter.

```
//Get Firebase auth instance
mAuth = FirebaseAuth.getInstance();

mDatabaseReference = FirebaseDatabase.getInstance().getRefer-
ence().child("users");

String user_id = mAuth.getCurrentUser().getUid();

//prosthiki polis sti vash
DatabaseReference curr_db_user = mDatabaseReference.child("pets");
uniqueId = UUID.randomUUID().toString();
ref = curr_db_user.child("pet" + uniqueId + "");

//Theloume me ena write na eisagoume ta dedomena kai oxi me pollapla
final Animal animal = new Animal();
animal.setUser_id(user_id);
animal.setCity(poli);
animal.setRegion(perioxi);
animal.setLatitude(latitude);
animal.setLongitude(longitude);
animal.setDescription(perigrafi);
animal.setIs_breed(if_breed);
animal.setIs_puppy(if_small);
animal.setFores_entopismou(1);
animal.setGender(fulo);
animal.setSize(size);
animal.setFlaw(flaws);
animal.setColor(color);
```

Κώδικας 51 “Δημιουργία αντικειμένου Animal με όλα τα στοιχεία που έδωσε ο χρήστης”

```
if (uni_uri != null) {
StorageReference filepath = mStorage.child("Photos").child(uniqueId);

filepath.putFile(uni_uri).addOnSuccessListener(new OnSuccessListener<Up-
loadTask.TaskSnapshot>() {
@Override
 public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {

ImageUpload imageUpload = new ImageUpload(taskSnapshot.getDown-
loadUrl().toString());
            Log.d("" + imageUpload, "url");
            animal.setPhoto_url(taskSnapshot.getDownloadUrl().toString());

            ref.setValue(animal);

            hideProgress();
            //edw prepei na kleisei to fragment
            posts p_frag = new posts();
            FragmentManager manager =    getActivity().getSupportFragment-
Manager();
manager.beginTransaction()
                    .replace(R.id.main_frame, p_frag, p_frag.getTag())
                    .commit();

        }
    });

} else {
    Toast.makeText(getContext(), "Sorry..you can't post without a pic-
ture", Toast.LENGTH_LONG).show();
    hideProgress();
    //edw prepei na kleisei to fragment
    posts p_frag = new posts();
    FragmentManager manager = getActivity().getSupportFragmentManager();
    manager.beginTransaction()
            .replace(R.id.main_frame, p_frag, p_frag.getTag())
            .commit();

}
```

Κώδικας 52 “Ανέβασμα φωτογραφίας στο Firebase και εισαγωγή δημοσίευσης”

Πίνακας 3 “Τύποι Δεδομένων JSON”

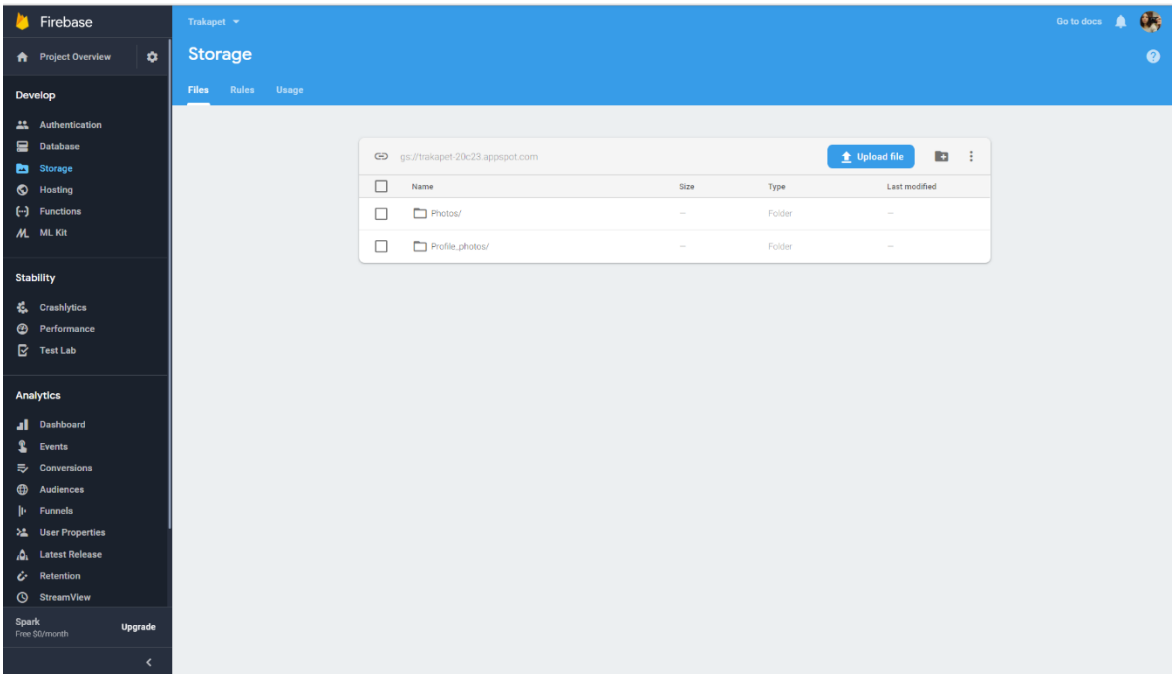
S. No.	Type & Description
1	**Number** double- precision floating-point format in JavaScript
2	**String** double-quoted Unicode with backslash escaping
3	**Boolean** true or false
4	**Array** an ordered sequence of values
5	**Value** it can be a string, a number, true or false, null etc
6	**Object** an unordered collection of key:value pairs
7	**Whitespace** can be used between any pair of tokens
8	**null** empty

Ο κώδικας για τη μεταφόρτωση της φωτογραφίας στον αποθηκευτικό χώρο του Firebase γίνεται αφού πρώτα δημιουργήσουμε ένα StorageReference δηλαδή ένα αντικείμενο που να έχει αναφορά για την τοποθεσία στην οποία θα ανεβάσουμε το αρχείο. Στον Κώδικα 53 παρατίθεται η αρχικοποίηση του mStorage.

```
//to root tou storage sto firebase
mStorage = FirebaseStorage.getInstance().getReference();
```

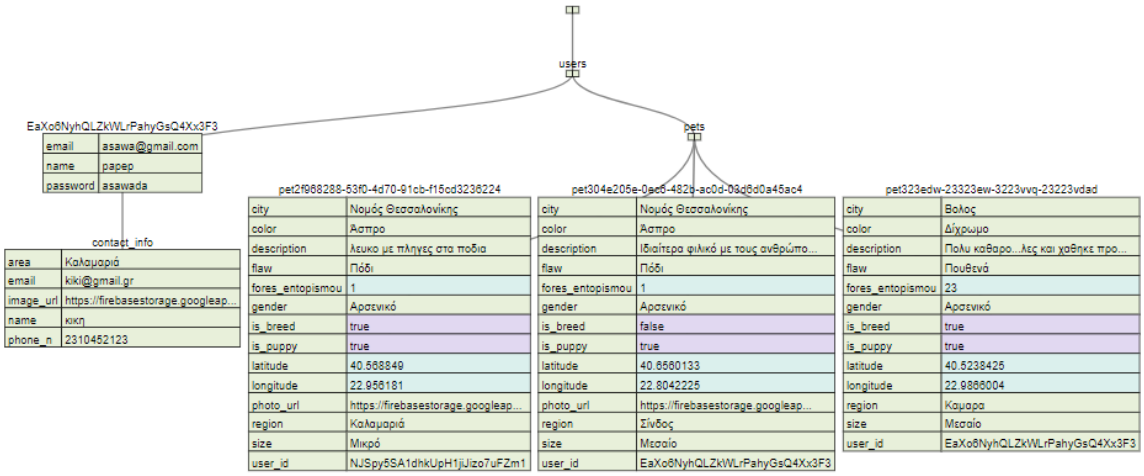
Κώδικας 53 “Αρχικοποίηση της αναφοράς στο Storage του Firebase”

Το αρχείο εικόνας θα βρίσκεται στον φάκελο Photos και η κάθε εικόνα θα έχει ένα μοναδικό id σε σχέση με τις υπόλοιπες εικόνες. Η μεταφόρτωση επομένως πραγματοποιείται με την κλήση της μεθόδου putFile() χρησιμοποιώντας την αναφορά για να γνωρίζει που ακριβώς θα το μεταφορτώσει. Η ίδια λογική ακολουθείται και όταν μεταφορτώνεται η φωτογραφία του χρήστη χρησιμοποιώντας το ProfileFragment. Στην εικόνα 29 μπορούμε να δούμε πως είναι η διεπαφή του Storage στο Firebase.



Εικόνα 29 “Διεπαφή του αποθηκευτικού χώρου στο Firebase”

Το JSON document με καταχωρημένες διάφορες δημοσιεύσεις και περισσότερους δημιουργημένους χρήστες είναι αρκετά μεγάλο και είναι πιο κατανοητό αν το δούμε σε μορφή διαγράμματος δέντρου όπως βλέπουμε στο διάγραμμα 3.



Διάγραμμα 3 “Δέντρο-διάγραμμα του JSON document έπειτα από δημοσιεύσεις καταχωρήσεων ζώων”

5.3.4 ΑΠΟΣΤΟΛΗ ΑΙΤΗΜΑΤΟΣ ΔΙΑΜΟΙΡΑΣΜΟΥ ΣΤΟΙΧΕΙΩΝ

Προκειμένου ένας χρήστης να μπορέσει να δει τα στοιχεία ενός άλλου σε μία δημοσίευση θα πρέπει ένας από τους δύο να έχει στείλει αίτημα διαμοιρασμού στοιχείων και αυτός που θα το λάβει να πατήσει αποδοχή. Για να δούμε ποιοι καινούργιοι κόμβοι προστίθενται σε περίπτωση που σταλθεί ένα αίτημα.

Ο listener ValueEventListener χρησιμοποιείται κατά κόρον όταν θέλουμε να πάρουμε τιμές από το Firebase μιας και με την μέθοδο ανάκλησης σε events onDataChange() μπορεί να διαβάσει το στιγμιότυπο των περιεχομένων του μονοπατιού που δόθηκε. Το στιγμιότυπο περιέχει και τους κόμβους παιδιά. Αν δεν υπάρχουν δεδομένα στο snapshot θα επιστραφεί η τιμή null. Με το datasnapshot.exists() βλέπουμε αν υπάρχει το μονοπάτι που δώσαμε. Σε περίπτωση που υπάρχει, αρχικοποιούμε δύο μεταβλητές με το όνομα και το url της εικόνας προφίλ που έχει καταχωρήσει ο χρήστης. Έπειτα, θα κληθεί μία άλλη μέθοδος η οποία θα στείλει αυτές τις δύο πληροφορίες στον χρήστη

```
@Override
public void sendDetails(final String user_id) {

    //to uid tou ekastote xrhsth
    String current_user_id = mAuth.getCurrentUser().getUid();

    databaseReference = FirebaseDatabase.getInstance().getReference("us-
ers").child(current_user_id).child("contact_info");
    databaseReference.addValueEventListener(new com.google.firebase.data-
base.ValueEventListener() {

        //oi plhrofories pou xreiazomaste apo ton ekastote xrhsth einai
        //to onoma kai to url tou
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {

            if(dataSnapshot.exists()) {
             name = (String) dataSnapshot.child("name").getValue();
             url = (String) dataSnapshot.child("image_url").getValue();
                sendDataToUser(user_id,name, url);
            }else{
                mview.NoProfileExists();
            }
        }

        @Override
        public void onCancelled(DatabaseError databaseError) {
        }
    });
}
```

Κώδικας 54 “Η μέθοδος ανάγνωσης στοιχείων του προφίλ του χρήστη μέσω του Firebase”

Αφού αρχικοποιήσουμε μία αναφορά στο contact_info του χρήστη στον οποίο θέλουμε να στείλουμε το αίτημα (Κώδικας 55), μέσα στο κόμβο alerts τοποθετούμε έναν ακόμα κόμβο που ονομάζεται user_id συν το id του χρήστη. Μέσα στον κόμβο αυτό θέτουμε ως παιδιά του, το όνομα του χρήστη που ζήτησε το αίτημα και την εικόνα του. Το JSON document έχει τότε την μορφή της Εικόνας 30.

```
public void sendDataToUser(String user_id, String name, String url) {

    //to uid tou ekastote xrhsth
String current_user_id = mAuth.getCurrentUser().getUid();
databaseReference = FirebaseDatabase.getInstance().getReference("users")
.child(user_id).child("contact_info");

DatabaseReference other_user = databaseRefence.
                                    child("alerts").
                                    child("user_id" + current_user_id);

    UserProfile other = new UserProfile();
    other.setName(name);
    other.setImage_url(url);
    other_user.setValue(other);
    mview.show_Toast("Request was sent succesfully");

}
```

Κώδικας 55 “Η μέθοδος αποστολής στοιχείων σε άλλον χρήστη μέσω του Firebase”

.

Στον δεύτερο χρήστη εισάγεται ένας καινούργιος κόμβος ο οποίος θα διαγραφεί μόνο όταν ο χρήστης ενημερωθεί μέσω ενός custom dialog που είδαμε σε προηγούμενο κεφάλαιο και πατήσει ναι ή όχι στο να συνδεθεί με τον χρήστη. Εάν συνδεθεί με τον χρήστη τότε θα προκύψει ένας καινούργιος κόμβος με όνομα connected_with και στους δύο χρήστες, που θα περιλαμβάνει το id του ενός στου άλλου. Έτσι, το JSON document θα έχει τώρα τη μορφή της εικόνας 31. Επομένως, τώρα οι δύο χρήστες μπορούν να δουν ο ένας τα στοιχεία του άλλου και να έρθουν σε επικοινωνία σε περίπτωση που ο ένας γνωρίζει κάτι για το ζώο που δημοσιεύτηκε.

```
{
  "users": {
    "EaXo6NyhQLZkWLrPahyGsQ4Xx3F3": {
      "contact_info": {
        "area": "Καλαμαριά",
        "email": "kiki@gmail.gr",
        "image_url": "https://firebasestorage.googleapis.com/v0/b/trakapet-20c23.appspot.com/o
          /Profile_photos%2FEaXo6NyhQLZkWLrPahyGsQ4Xx3F3?alt=media&token=bd1f9022-6fa5-425e-b043
          -b1ab0f145ae2",
        "name": "κικη",
        "phone_n": "2310452123"
      },
      "email": "asawa@gmail.com",
      "name": "papep",
      "password": "asawada"
    },
    "dX3ZABbjCAUS4ppqyeCBIkGoLh82": {
      "contact_info": {
        "alerts": {
          "user_idEaXo6NyhQLZkWLrPahyGsQ4Xx3F3": {
            "image_url": "https://firebasestorage.googleapis.com/v0/b/trakapet-20c23.appspot.com
              /o/Profile_photos%2FEaXo6NyhQLZkWLrPahyGsQ4Xx3F3?alt=media&token=bd1f9022-6fa5-425e
              -b043-b1ab0f145ae2",
            "name": "κικη"
          }
        },
        "area": "thessaloniki",
        "email": "maranth@gmail.com",
        "image_url": "https://firebasestorage.googleapis.com/v0/b/trakapet-20c23.appspot.com/o
          /Profile_photos%2FdX3ZABbjCAUS4ppqyeCBIkGoLh82?alt=media&token=d776964b-f92c-497e-aac1
          -1f03befcad05",
        "name": "Μαριάνθη",
        "phone_n": "6396532552"
      },
      "email": "dadsa@gmail.com",
      "name": "fafda",
      "password": "adawed"
    }
  }
}
```


Εικόνα 30 “JSON document έπειτα από αποστολή αιτήματος σύνδεσης”

```
{
  "users": {
    "EaXo6NyhQLZkWLrPahyGsQ4Xx3F3": {
      "contact_info": {
        "area": "Καλαμαριά",
        "connected_with": {
          "user_iddX3ZABbjCAUS4ppqyeCBIkGoLh82": "dX3ZABbjCAUS4ppqyeCBIkGoLh82"
        },
        "email": "kiki@gmail.gr",
        "image_url": "https://firebasestorage.googleapis.com/v0/b/trakapet-20c23.appspot.com/o
          /Profile_photos%2FEaXo6NyhQLZkWLrPahyGsQ4Xx3F3?alt=media&token=bd1f9022-6fa5-425e-b043
          -b1ab0f145ae2",
        "name": "κικη",
        "phone_n": "2310452123"
      },
      "email": "asawa@gmail.com",
      "name": "papep",
      "password": "asawada"
    },
    "dX3ZABbjCAUS4ppqyeCBIkGoLh82": {
      "contact_info": {
        "area": "thessaloniki",
        "connected_with": {
          "user_idEaXo6NyhQLZkWLrPahyGsQ4Xx3F3": "EaXo6NyhQLZkWLrPahyGsQ4Xx3F3"
        },
        "email": "maranth@gmail.com",
        "image_url": "https://firebasestorage.googleapis.com/v0/b/trakapet-20c23.appspot.com/o
          /Profile_photos%2FdX3ZABbjCAUS4ppqyeCBIkGoLh82?alt=media&token=d776964b-f92c-497e-aac1
          -1f03befcad05",
        "name": "Μαριάνθη",
        "phone_n": "6396532552"
      },
      "email": "dadsa@gmail.com",
      "name": "fafda",
      "password": "adawed"
    }
  }
}
```


Εικόνα 31 “JSON document έπειτα από επιτυχημένη σύνδεση των δύο χρηστών”

5.3.5 ΦΟΡΤΩΣΗ ΟΛΩΝ ΤΩΝ ΔΗΜΟΣΙΕΥΣΕΩΝ

Για να πάρουμε όλα τα δημοσιευμένα εντοπισμένα ζώα θα πρέπει να πάμε στο μονοπάτι users/pets. Μιας και θέλουμε να διαβάσουμε τα ζώα ένα ένα αρχικοποιούμε ένα στιγμιότυπο Iterable<DataSnapshot> το οποίο περιέχει όλα τα παιδιά που βρίσκονται κάτω από το pets. Στη συνέχεια, διαβάζουμε τον κάθε κόμβο του pets παίρνοντας τα στοιχεία που χρειαζόμαστε. Ο κάθε κόμβος γίνεται ένα ξεχωριστό αντικείμενο μέσα σε λίστα. Έπειτα, αυτό το ArrayList τύπου animals θα φορτωθεί στον Adapter που βρίσκεται μέσα στο RecyclerView.

```
public void loadData(final boolean isRefresh) {

databaseReference = FirebaseDatabase.getInstance().getReference("users");
databaseReference.addValueEventListener(new com.google.firebase.data-
base.ValueEventListener() {
@Override
 public void onDataChange(DataSnapshot dataSnapshot) {
 Iterable<DataSnapshot> pets = dataSnapshot.child("pets").getChildren();
 animals.clear();

for (DataSnapshot anims : pets) {

String usersid = (String) anims.child("user_id").getValue();
String region = (String) anims.child("region").getValue();
String city = (String) anims.child("city").getValue();
String description = (String) anims.child("description").getValue();
int fores_entopismou = (anims.child("fores_entopismou").getValue(Inte-
ger.class));
Double latitude = ((Double) anims.child("latitude").getValue());
Double longitude = (Double) anims.child("longitude").getValue();
boolean breed = (boolean) anims.child("is_breed").getValue();
boolean small = (boolean) anims.child("is_puppy").getValue();
String url = (String) anims.child("photo_url").getValue();
String gender = (String) anims.child("gender").getValue();
String size = (String) anims.child("size").getValue();
String flaws = (String) anims.child("flaw").getValue();
String color = (String) anims.child("color").getValue();

Animal match = new Animal(usersid, city, region, latitude, longitude, de-
scription, breed, small, gender, size, flaws, color, url, fores_entopismou);
animals.add(match);
}
```

Κώδικας 56 “Η μέθοδος ανάγνωσης όλων των δημοσιεύσεων από το Firebase”

ΕΠΙΛΟΓΟΣ

Σε αυτό το κεφάλαιο εξηγήσαμε εν συντομία τι είναι το Firebase και αναφέραμε τις υπηρεσίες που παρέχει και αξιοποιήσαμε στην εφαρμογή. Έπειτα, εξηγήσαμε αναλυτικά τα βασικά κομμάτια κώδικα της εφαρμογής που χρησιμοποιούν το Firebase κατά κόρον και έγινε καλύτερα κατανοητή η δομή της βάσης παρέχοντας τα JSON document και JSON δέντρο διαγράμματα.

ΚΕΦΑΛΑΙΟ 6
ΧΡΗΣΗ GPS ΚΑΙ WIFI/DATA ΣΤΟ ANDROID

ΕΙΣΑΓΩΓΗ
Στην εφαρμογή με το που πατήσει ο χρήστης το κουμπί της προσθήκης μιας δημοσίευσης θα υπολογιστεί το γεωγραφικό μήκος και πλάτος και θα αποθηκευτεί στη βάση. Για να γίνει αυτό χρησιμοποιούμε τεχνολογίες που έχουν όλα τα smartphones. Θα αναλυθούν οι τεχνολογίες που υπολογίζουν την τοποθεσία ενός χρήστη καθώς και πως το έγινε η υλοποίηση αυτής της λειτουργίας.

6.1 ΥΠΟΛΟΓΙΣΜΟΣ ΤΗΣ ΤΟΠΟΘΕΣΙΑΣ ΣΤΟ ANDROID
Ο υπολογισμός της τοποθεσίας μπορεί να γίνει με δύο τρόπους αλλά ο καθένας έχει πλεονεκτήματα και μειονεκτήματα. Ένας τρόπος είναι με τη χρήση του WIFI ή των δεδομένων και ένας άλλος τρόπος με τη χρήση του GPS. Λόγω του ότι η εφαρμογή μας προϋποθέτει σύνδεση στο διαδίκτυο μπορούμε να χρησιμοποιήσουμε και τους δύο τρόπους και να επιλέξουμε αυτόν που έχει την καλύτερη ακρίβεια. Επίσης, αν για κάποιο λόγο δεν μπορεί να υπολογιστεί με τον έναν από τους δύο τρόπους τότε θα επιλεχθεί ο άλλος. Προγραμματιστικά όμως, υπάρχουν διάφοροι τρόποι υπολογισμού άλλοι λιγότερο και άλλοι περισσότερο αποδοτικοί. Πριν ξεκινήσει ο υπολογισμός της τοποθεσίας θα πρέπει να γίνει έλεγχος για το αν έχουν δοθεί τα απαραίτητα permissions. Σε περίπτωση που ο χρήστης δεν μας δώσει ορισμένες άδειες δεν μπορούμε να συνεχίσουμε.

6.1.1 ΑΝΑΚΤΗΣΗ ΤΗΣ ΤΟΠΟΘΕΣΙΑΣ ΑΠΟ ΤΗ ΜΝΗΜΗ
Οι περισσότερες συσκευές Android έχουν την ικανότητα να διατηρούν την τελευταία τοποθεσία στην οποία βρισκόντουσαν οπότε μία επιλογή είναι να χρησιμοποιηθεί αυτή. Το πρόβλημα με αυτή τη λύση όμως είναι ότι θέλουμε η τοποθεσία να είναι πρόσφατη αλλιώς δεν υπάρχει λόγος ύπαρξης αυτής της λειτουργίας. Παρ' όλα αυτά μπορούμε να τη χρησιμοποιήσουμε και αφού ελέγξουμε πριν πόση ώρα αποθηκεύτηκε να δράσουμε ανάλογα. Αν είναι προ λίγων λεπτών τότε μπορεί πράγματι να τη χρησιμοποιήσουμε γιατί η λογική εντοπισμού στην εφαρμογή είναι για να μπορεί ο πιθανός ιδιοκτήτης του να δει που συχνάζει το ζώο. Αυτή η επιλογή παρέχεται και από το GPS αλλά και από το δίκτυο οπότε μπορούν να γίνουν ξεχωριστοί έλεγχοι.

6.1.2 ΥΠΟΛΟΓΙΣΜΟΣ ΤΟΠΟΘΕΣΙΑΣ ΣΕ ΠΡΑΓΜΑΤΙΚΟ ΧΡΟΝΟ
Όταν ο χρήστης ενεργοποιεί το GPS δεν γίνεται να επιστρέψει κατευθείαν δεδομένα αλλά χρειάζεται λίγο χρόνο. Αυτό είναι κάτι που δυσκολεύει ιδιαίτερα την υλοποίηση διότι θα πρέπει ή να εξαναγκάζουμε το χρήστη να το ανοίγει πιο νωρίς ή να πρέπει να περιμένει. Ο χρόνος αναμονής μεταβάλλεται από συσκευή σε συσκευή. Αυτός ο υπολογισμός μπορεί να γίνει και χρησιμοποιώντας το Δίκτυο κινητής τηλεφωνίας ή το WIFI.

6.1.2.1 NETWORK PROVIDER VS GPS PROVIDER

Ο Network provider θα χρησιμοποιηθεί μόλις ζητήσουμε την τοποθεσία χρησιμοποιώντας το διαδίκτυο ή το δίκτυο κινητής τηλεφωνίας και όχι το GPS. Στην περίπτωση που χρησιμοποιηθεί το δίκτυο κινητής τηλεφωνίας η τοποθεσία της συσκευής του χρήστη υπολογίζεται λαμβάνοντας υπόψιν την απόσταση μεταξύ των πύργων και της θέσης του χρήστη. Έτσι, υπολογίζεται ευκολότερα η τοποθεσία. Συχνά χρησιμοποιείται για τον προσδιορισμό της τοποθεσίας μέσα σε κτήρια, αντίθετα με το GPS. Τα είδη των permission που απαιτείται να έχουμε συμπεριλάβει στο manifest προκειμένου να χρησιμοποιηθεί ο Network Provider είναι τα εξής:

```
ACCESS_COARSE_LOCATION

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

ACCESS_FINE_LOCATION

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

Κώδικας 57 "Οι απαραίτητες άδειες για τη χρήση του Network Provider"

Ο GPS provider από την άλλη προσδιορίζει την τοποθεσία των χρηστών χρησιμοποιώντας δορυφόρους. Ο λήπτης GPS του smartphone λαμβάνει σήματα από τους δορυφόρους, τα επεξεργάζεται και η ακριβής τοποθεσία καθορίζεται. Χρειάζεται περισσότερο χρόνο αλλά όπως είπαμε δουλεύει ικανοποιητικά όταν είσαι έξω. Αυτός ο provider χρειάζεται μόνο το permission:

```
ACCESS_FINE_LOCATION

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

Κώδικας 58 " Οι απαραίτητες άδειες για τη χρήση του GPS Provider"

6.2 Η ΥΛΟΠΟΙΗΣΗ ΓΙΑ ΤΟΝ ΕΝΤΟΠΙΣΜΟ ΤΗΣ ΤΟΠΟΘΕΣΙΑΣ

Στην υλοποίηση υπάρχει συνδυασμός τεχνολογιών και μεθόδων προκειμένου να υπάρχει μεγαλύτερη ευελιξία. Τα permissions που απαιτούνται για την συγκεκριμένη υλοποίηση είναι τα εξής:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-feature android:name="android.hardware.location.gps"/>
<uses-feature android:name="android.hardware.location.network"/>
```

Κώδικας 59 “Οι απαραίτητες άδειες για τον εντοπισμό της τοποθεσίας”

Το permission ACCESS_NETWORK_STATE είναι για να μπορούμε να έχουμε πρόσβαση στις πληροφορίες του δικτύου.

Το στιγμιότυπο από το κάθε Fragment σχετίζεται με το στιγμιότυπο του Activity στο οποίο φορτώνεται. Στην προκειμένη περίπτωση, το Fragment new_post σχετίζεται με το ActivityMain. Η μέθοδος onAttach() είναι υπεύθυνη για να παρέχει μια αναφορά στο Activity.

Το αντικείμενο LocationManager, μας παρέχει πρόσβαση στις υπηρεσίες τοποθεσίας του συστήματος. Αυτές οι υπηρεσίες επιτρέπουν στις εφαρμογές να έχουνε ενημερώσεις ανά τακτά χρονικά διαστήματα σχετικά την τοποθεσία της συσκευής.

```
@Override
public void onAttach(Context context) {
super.onAttach(context);

locationManager = (LocationManager)context.getSystemService(Context.LOCA-
TION_SERVICE);
}
```

Κώδικας 60 “Αντικείμενο LocationManager”

Η μέθοδος configureLocation() στον Κώδικα 61 ελέγχει αν είναι ενεργοποιημένο το GPS και παίρνει την τελευταία τοποθεσία που έχει αποθηκευτεί. Σε περίπτωση που δεν είναι ανοιχτό το GPS θα βγει ένα Dialog το οποίο θα μας παραπέμψει στις ρυθμίσεις τοποθεσίας προκειμένου να το ανοίξουμε. Επειδή η τοποθεσία μπορεί να υπολογιστεί και από τα Data ή το WIFI δίνουμε την επιλογή στον χρήστη αν θέλει να μην το ενεργοποιήσει. Επομένως, στην περίπτωση που πατήσει Cancel θα προχωρήσει στην επόμενη μέθοδο την configureLocation2() που υπολογίζει την τοποθεσία από τα Data ή το WIFI.

```
private void configureLocation() {


    if (gps_enabled) {
       if (ActivityCompat.checkSelfPermission(getActivity(),
Manifest.permission.ACCESS_FINE_LOCATION) !=   PackageManager.PERMIS-
SION_GRANTED && ActivityCompat.checkSelfPermission(getActivity(), Mani-
fest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMIS-
SION_GRANTED) {
            return;
        }
gps_loc =    locationManager.getLastKnownLocation(LocationManager.GPS_PRO-
VIDER);
        configureLocation2();
    } else {
        final String prov = LocationManager.GPS_PROVIDER;
        AlertDialog.Builder dialog = new AlertDialog.Builder(getActiv-
ity());
dialog.setTitle("Enable Location")
.setMessage("Your Locations Settings is set to 'Off'.\nPlease Enable Loca-
tion to " +
                        "use this app if your outside")
.setPositiveButton("Location Settings", new DialogInterface.OnClick-
Listener() {
 @Override
 public void onClick(DialogInterface paramDialogInterface, int paramInt) {
 Intent intent = new Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
                        startActivityForResult(intent, 10);

                    }
                })
 .setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
 @Override
 public void onClick(DialogInterface paramDialogInterface, int paramInt) {

                    configureLocation2();
                    }


                });
        dialog.show();
    }
```

Κώδικας 61 “Η μέθοδος configureLocation”

```
private void configureLocation2(){

    if (network_enabled) {
        if (ActivityCompat.checkSelfPermission(getActivity(), Mani-
fest.permission.ACCESS_FINE_LOCATION) !=
                PackageManager.PERMISSION_GRANTED && Activity-
Compat.checkSelfPermission(getActivity(),
                Manifest.permission.ACCESS_COARSE_LOCATION) != Pack-
ageManager.PERMISSION_GRANTED) {
            return;
        }
        net_loc = locationManager.getLastKnownLocation(locationMan-
ager.NETWORK_PROVIDER);
        locate_pet();
    } else {
        if (confirmWiFiAvailable(getActivity()) == false || confirmAir-
planeModeOff(getActivity()) == false) {
            final String prov = LocationManager.GPS_PROVIDER;
            final AlertDialog.Builder dialog = new AlertDia-
log.Builder(getActivity());
            dialog.setTitle("Enable your wifi")
 .setMessage("Your WIFI or DATA is not enabled.\nPlease Enable it to " +
                        "use this app")
.setPositiveButton("Settings", new DialogInterface.OnClickListener() {

 @Override
 public void onClick(DialogInterface paramDialogInterface, int paramInt) {
 Intent intent = new Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
                        startActivityForResult(intent, 11);
                        Log.d("Provider Disabled", prov);
                    }
                })
.setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
 @Override
public void onClick(DialogInterface paramDialogInterface, int paramInt) {
                    }
                });
            dialog.show();
        }
    }
}
```

Κώδικας 62 “Η μέθοδος configureLocation2”

Η μέθοδος ConfigureLocation2() ελέγχει αν τα Data ή το WIFI είναι ενεργοποιημένα και παίρνει την τελευταία τιμή που πιθανόν να υπάρχει αποθηκευμένη. Εμφανίζεται και το κατάλληλο μήνυμα σε περίπτωση που το WIFI είναι απενεργοποιημένο. Εφόσον λοιπόν είναι ένα από τα δύο ανοιχτό θα κληθεί η επόμενη μέθοδος η locate_pet().

```
public void locate_pet() {

    if (gps_enabled) {
         if (gps_loc != null && gps_loc.getTime() < Calendar.get-
Instance().getTimeInMillis() - 2 * 60 * 1000) {
            if (ContextCompat.checkSelfPermission(getContext(), Mani-
fest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED
&& ContextCompat.checkSelfPermission(getContext(), Manifest.permission.AC-
CESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
                return;
            }
            locationManager.requestLocationUpdates(LocationMan-
ager.GPS_PROVIDER, 0, 0, this);
} else if (gps_loc == null) {
            locationManager.requestLocationUpdates(LocationMan-
ager.GPS_PROVIDER, 0, 0, this);
        }
    }

    if (network_enabled) {
if (net_loc != null && net_loc.getTime() < Calendar.getInstance().get-
TimeInMillis() - 2 * 60 * 1000) {
locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,
0, 0, this);
        } else if (net_loc == null) {
locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,
0, 0, this);
        }else {
            findBest();
        }
    }
}
```

Κώδικας 63 “Η μέθοδος που ενεργοποιεί τους Location Listeners”

Όσον αφορά τον Κώδικα 63, σε περίπτωση που υπήρχε αποθηκευμένη κάποια τοποθεσία έπειτα από προηγούμενη χρήση του GPS γίνεται έλεγχος για το αν η τοποθεσία αυτή καταγράφηκε πρόσφατα. Αν δεν καταγράφηκε πρόσφατα, καλούμε τον listener να μας δώσει μια καινούργια, το ίδιο συμβαίνει και αν τυχόν ήτανε null. Η ίδια λογική ισχύει και για τον network provider.

Ο listener υπολογίζει την τοποθεσία στην οποία βρίσκεται η συσκευή μας. Μπορούμε να τον βάλουμε να λαμβάνει ενημερώσεις για την τοποθεσία ανά συγκεκριμένο χρόνο ή μόλις “αισθανθεί” αλλαγή τοποθεσίας βάζοντας ένα ελάχιστο όριο μέτρων. Εμείς θέλουμε να κληθεί κατευθείαν επομένως βάλαμε μηδενικές τιμές σε αυτές τις παραμέτρους. Η δουλειά του είναι να στέλνει συνεχόμενα την τοποθεσία μέχρι να τον σταματήσουμε. Αυτό είναι εφικτό καλώντας την μέθοδο removeUpdates().

Ο listener σύμφωνα με τον Κώδικα 63, καλείται πάντα να υπολογίσει πρώτα την τοποθεσία με το GPS. Φυσικά αυτό προϋποθέτει να είναι ενεργοποιημένο αλλιώς απλά δεν θα χρησιμοποιηθεί καθόλου αυτή η τιμή. Αν υπολογίσει την τοποθεσία με το GPS θέλουμε έπειτα να ξαναχρησιμοποιηθεί άλλη μια φορά για να την υπολογίσει και με τα Data ή το WIFI. Επομένως σταματάμε να δεχόμαστε ενημερώσεις τοποθεσίας μόλις κληθεί ο listener χρησιμοποιώντας τον NETWORK_PROVIDER. Σε περίπτωση που η τοποθεσία που υπολογίστηκε ήτανε πρόσφατη και δεν κληθεί ο listener, θα κληθεί αμέσως η μέθοδος findBest(). Διαφορετικά η μέθοδος findBest() θα κληθεί εντός του listener μόλις υπολογίσει την τοποθεσία με τον NETWORK_PROVIDER.

```
public void findBest(){

    if (gps_loc != null && net_loc != null) {
        //smaller the number more accurate result will
        if (gps_loc.getAccuracy() > net_loc.getAccuracy())
            finalLoc = net_loc;
        else
            finalLoc = gps_loc;

    } else {

        if (gps_loc != null) {
            finalLoc = gps_loc;

        } else if (net_loc != null) {
            finalLoc = net_loc;
        } else {
            finalLoc = null;
        }
    }

    if (finalLoc != null) {
        mlatitude = finalLoc.getLatitude();
        mlongitude = finalLoc.getLongitude();
        System.out.println("LAT " + mlatitude + "LONG" + mlongitude);
        submitData();
    } else {
        mlatitude = 0.1;
        mlongitude = 0.1;
        showErrorToast("Specific location on maps couldn't be speci-
fied");
        submitData();
    }
```

Κώδικας 64 “Η μέθοδος υπολογισμού της καλύτερης τιμής της τοποθεσίας”

Η μέθοδος findBest() είναι για να αποθηκεύσουμε το γεωγραφικό πλάτος και μήκος σε μεταβλητές και να μπορέσουμε να τις εισάγουμε στη βάση. Βλέποντας τον Κώδικα 64 καταλαβαίνουμε ότι υπάρχει έλεγχος για την κάθε περίπτωση που μπορεί να προκύψει. Αν έχουμε τιμή και από τον GPS_PROVIDER αλλά και από τον NETWORK_PROVIDER τότε θα χρησιμοποιήσουμε αυτή με τη μεγαλύτερη ακρίβεια. Διαφορετικά, σημαίνει ότι μία από τις δύο είναι null οπότε θα χρησιμοποιηθεί αυτή που δεν είναι. Αν πάλι δεν έχει εντοπιστεί η τοποθεσία με κανέναν provider θα εισαχθούν οι τιμές 0.1 που συμβολίζουν για εμάς ότι δεν εντοπίστηκε τοποθεσία. Αν βάλουμε μηδενικές τιμές τότε το Firebase θα βγάζει Error κατά το διάβασμα των τιμών οπότε αυτό είναι μια εναλλακτική λύση.

```
@Override
public void onLocationChanged(Location location) {
    if (location != null) {
        if (location.getProvider().matches("network")) {
            net_loc = location;
            locationManager.removeUpdates(this);
            findBest();
        } else {
            gps_loc = location;
            if(!network_enabled){
                findBest();
            }
        }

    }
}
```

Κώδικας 65 “Η μέθοδος υπολογισμού της τοποθεσίας του Location Listener”

Αυτός ο Listener θα κληθεί με το που το καλέσουμε τη μέθοδο requestLocationUpdates().

ΕΠΙΛΟΓΟΣ

Παρουσιάστηκαν οι τεχνολογίες και οι τρόποι με τους οποίους μπορούμε να εντοπίσουμε την τοποθεσία μιας συσκευής παραθέτοντας την υλοποίηση που χρησιμοποιήθηκε στην εφαρμογή.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Ολοκληρώνοντας αυτή την πτυχιακή εργασία ασχοληθήκαμε σε μεγάλο βαθμό με τις κύριες λειτουργίες της εφαρμογής διαχωρίζοντας τες σε διαφορετικά κεφάλαια. Ένα πολύ μεγάλο μέρος της εφαρμογής χρησιμοποιεί το Firebase γι' αυτό και η αναφορά σε αυτό υπήρξε σε αρκετά σημεία.

Όταν πρωτοξεκίνησε η ανάπτυξη της εφαρμογής δεν χρησιμοποιούνταν κάποιο μοντέλο με αποτέλεσμα ο κώδικας να γίνει αρκετά μεγάλος και δύσκολα συντηρήσιμος και επεκτάσιμος. Περίπου στα μισά της εφαρμογής ξεκίνησε η μετατροπή όλων των κλάσεων ώστε να υποστηρίζεται το MVP και πραγματικά η διαφορά στην ταχύτητα της υλοποίησης από εκεί και μετέπειτα ήταν μεγάλη. Οποιαδήποτε εφαρμογή καλό είναι να χρησιμοποιεί ένα μοντέλο ώστε να είναι πιο εύκολο να θυμάται ο ίδιος ο προγραμματιστής καλύτερα τι έχει κάνει μιας όλα είναι πιο οργανωμένα και ξεκάθαρα.

Όσον αφορά τις μελλοντικές επεκτάσεις, θα ήταν πολύ χρήσιμο να έμπαινε ένας μηχανισμός ο οποίος μπορεί να αντιληφθεί πότε μια δημοσίευση αναφέρεται στο ίδιο ζώο με ένα άλλο ώστε να μην προστίθενται και άλλες δημοσιεύσεις για το ίδιο και απλά οι φορές εντοπισμού να αυξάνονται.

ΒΙΒΛΙΟΓΡΑΦΙΑ

https://www.androidcentral.com/android-history

https://www.androidauthority.com/why-developers-choose-android-285774/

http://www.kritikalsolutions.com/blog/top-5-reasons-choose-android-platform-app-development/

https://docs.microsoft.com/en-us/xamarin/android/app-fundamentals/android-api-levels?tabs=vswin

https://riggaroo.co.za/constraintlayout-guidelines-barriers-chains-groups/

https://code.tutsplus.com/el/tutorials/getting-started-with-recyclerview-and-cardview-on-android--cms-23465

https://developer.android.com/training/animation/screen-slide

https://developer.android.com/reference/android/widget/AutoCompleteTextView

https://www.bignerdranch.com/blog/implementing-swipe-to-refresh/

https://infinum.co/the-capsized-eight/share-files-using-fileprovider

http://www.vogella.com/tutorials/AndroidArchitecture/article.html

https://medium.com/@jintin/mvp-architecture-in-android-1f98a74e7e1b

https://github.com/nikiizvorski/Firebase-MVP/blob/master/app/src/main/java/com/niki/mvpexample/app/Login/LoginPresenterImpl.java

http://priocept.com/2017/10/02/implementing-mvp-on-android/

https://code.tutsplus.com/tutorials/how-to-adopt-model-view-presenter-on-android--cms-26206

https://firebase.google.com/docs/android/setup

https://code.tutsplus.com/tutorials/image-upload-to-firebase-in-android-application--cms-29934

https://www.learnhowtoprogram.com/android/data-persistence/firebase-reading-data-and-event-listeners