ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Μελέτη και υλοποίηση συστήματος ποτίσματος με τεχνολογίες του Διαδικτύου των Πραγμάτων

Του φοιτητή

Δήμου Κασαπούδη

Αρ. Μητρώου: 134125

Επιβλέπων - Μέλος ΔΕΠ

Δρ. Περικλής Χατζημίσιος

Καθηγητής

Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων

Ιούνιος 2021

Θεσσαλονίκη

ΠΕΡΙΛΗΨΗ

Όπως οι περισσότεροι γνωρίζουμε, το νερό, αυτό το πολύτιμο αγαθό της ζωής κοντεύει να εκλείψει. Γι' αυτόν το λόγο πρέπει να πραγματοποιηθούν ενέργειες οι οποίες θα περιορίσουν όσο το δυνατόν περισσότερο την άσκοπη σπατάλη του. Στην παρούσα πτυχιακή εργασία θα αναφέρουμε σύντομα το πρόβλημα της άσκοπης σπατάλης νερού, τις ενέργειες που έγιναν όταν παρατηρήθηκε, καθώς και λύσεις που εφαρμόστηκαν, για να αντιμετωπιστεί αυτή η κατάσταση. Επίσης, θα αναφερθούμε στο τι είναι το Διαδίκτυο των Πραγμάτων (Internet of Things - IoT), πώς λειτουργεί, την ασφάλεια που παρέχει μέχρι στιγμής και θα εξηγήσουμε τον τρόπο με τον οποίο συμβάλει έτσι, ώστε να υλοποιηθεί η έξυπνη γεωργία. Ακόμα, θα γίνει αναφορά στην έξυπνη γεωργία, τους τομείς με τους οποίους είναι διασυνδεδεμένη, καθώς και κάποια από τα συστήματα που παρέχει μέχρι στιγμής τα οποία είναι διαθέσιμα στην αγορά. Παρακάτω, θα περιγράψουμε τις τεχνολογίες οι οποίες συνέβαλαν στο να ολοκληρωθούν τα κομμάτια κώδικα στον server, όπως και τα κομμάτια κώδικα της Android εφαρμογής. Στη συνέχεια, γίνεται αναλυτική περιγραφή του πρακτικού μέρους της πτυχιακής εργασίας τόσο στο server side κομμάτι όσο και στο κομμάτι της εφαρμογής για κινητά. Έπειτα, καταγράφουμε τα συμπεράσματα που βγάλαμε από τη μελέτη και την υλοποίηση της καθώς και τι προτείνουμε σε κάποιον που θα θελήσει να ασχοληθεί με παρόμοιο θέμα. Τέλος, αναφέρουμε κάποιες μελλοντικές επεκτάσεις οι οποίες θα μπορούσαν να εφαρμοστούν στο σύστημα που δημιουργήσαμε.

ABSTRACT

It is widely known, that water, one of the most important things in life, is about to disappear. That is the reason why actions must be taken to reduce its waste as much as possible. In the current B.Sc. thesis we will briefly mention the problem of water waste, the actions that were taken when it was observed and certain solutions that were applied to combat this situation. We will also talk about what the Internet of Things (IoT) is, how it works, the safety it provides so far and explain how it contributes to smart agriculture. Moreover, we will talk about smart agriculture, the sectors with which it is interconnected, as well as some of the systems it provides so far that are available in the market. In addition, we will describe the technologies that helped complete the code snippets on the server, as well as the code snippets of the android application. Also, a detailed description of the practical part of the dissertation is given in both the server-side part and the part of the mobile application. To conclude, we record the results we drew from the study and its implementation and the suggestions to someone who will want to deal with a similar issue. Finally, we mention some future extensions that could be implemented in the system we create.

ΕΥΧΑΡΙΣΤΙΕΣ

Αρχικά, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου Δρ. Περικλή Χατζημίσιο για την πολύτιμη βοήθεια, τις εύστοχες παρατηρήσεις και τις χρήσιμες συμβουλές που μου παρείχε. Επίσης, ευχαριστώ τον Δρ. Ευκλείδη Κεραμόπουλο για τις ιδέες που μου έδωσε για να εμπλουτίσω το περιεχόμενο του πρακτικού κομματιού της πτυχιακής. Ακόμη, θα ήθελα να ευχαριστήσω όλους τους καθηγητές και καθηγήτριες της σχολής για όλες τις γνώσεις και τα εφόδιο που μου δώσανε καθ' όλη τη διάρκεια των σπουδών μου. Στη συνέχεια, ευχαριστώ τους γονείς μου που χωρίς αυτούς δεν θα έφτανα εδώ που έφτασα. Έπειτα, ευχαριστώ την μέλλουσα γυναίκα μου Αντιγόνη Αμοιράδου για όλη την στήριξη και υποστήριξη της, καθώς και για την αρχική φιλολογική επιμέλεια της πτυχιακής εργασίας. Ακόμη, θέλω να ευχαριστήσω την συμφοιτήτρια και καλή μου φίλη Ραφαηλία Παναγιώτα Χριστουδούλου, για την επιμέλεια της μετάφραση της περίληψης (abstract) στην αγγλική γλώσσα. Τέλος, θα ήθελα να ευχαριστήσω τον εφηβικό μου φίλο Ανδρέα Κ. Χατζηδήμου, για την ολική επιμέλεια του κειμένου της πτυχιακής.

ΠΕΡΙΕΧΟΜΕΝΑ

Ευρετήριο εικόνων

ΚΕΦΑΛΑΙΟ 1 – Εισαγωγή

1.1 – Εισαγωγή

Τα προβλήματα που σχετίζονται με το νερό υπάρχουν σε παγκόσμιο επίπεδο. Το νερό αποτελεί «πηγή ζωής για τον πλανήτη και τους κατοίκους του», είναι «το υλικό της ζωής». Χωρίς αυτό δεν υπάρχει πλανήτης, δεν υπάρχει ζωή, δεν υπάρχει άνθρωπος. Το νερό είναι ένας πεπερασμένος φυσικός πόρος που η έλλειψη του αποτελεί στις μέρες μας μεγάλη απειλή για τη γη. Η έλλειψη αυτή οφείλεται στο γεγονός ότι ο πληθυσμός της γης έχει αυξηθεί, που αυτό συνεπάγεται ότι έχουν αυξηθεί και οι ανάγκες γι' αυτό το πολύτιμο αγαθό της ζωής. Για τον λόγο αυτό θα πρέπει όλοι μας σε ατομικό και συλλογικό επίπεδο να ευαισθητοποιηθούμε έτσι, ώστε να μην γίνεται άσκοπη σπατάλη του.

Το πρόβλημα της σπατάλης νερού δεν διαδραματίζεται τα τελευταία χρόνια αλλά παρατηρήθηκε δεκαετίες πριν. Από την πρώτη στιγμή όμως, που διαπιστώθηκε το πρόβλημα, έγιναν ενέργειες ενημέρωσης και ευαισθητοποίησης των πολιτών με σκοπό να αλλάξουν συνήθειες (όσο αφορά στη χρήση του νερού) όπως και ενέργειες εξοικονόμησης σε παγκόσμιο επίπεδο οι οποίες και χρηματοδοτούνταν είτε από τους αρμόδιους δήμους και τις τοπικές κυβερνήσεις είτε από ευρωπαϊκά προγράμματα (για χώρες της ευρωπαϊκής ένωσης).

Κάποιες από τις ενέργειες που έγιναν ήταν: αλλαγή των παλιών σωλήνων από τους οποίους υπήρχε μεγάλη διαρροή, αλλαγή στα παλαιού τύπου καζανάκια τα οποία σπαταλούσαν πολύ νερό με νέα τα οποία έκαναν σημαντική εξοικονόμηση, δημιουργήθηκαν βρύσες, ντουζιέρες κ.α. τα οποία σχεδιάστηκαν για εξοικονόμηση νερού, έγιναν κατάλληλες προσθήκες σε παλιές συσκευές, προωθήθηκαν οικιακές συσκευές (πλυντήρια πιάτων και ρούχων) αποτελεσματικής χρήσης νερού κ.α. Όλες οι παραπάνω ενέργειες περιόρισαν κατά πολύ τα λίτρα κατανάλωσης νερού αλλά όπως είπαμε και παραπάνω, λόγω του ότι ο πληθυσμός της γης αυξήθηκε, οι παραπάνω ενέργειες δεν αρκούν πλέον. Αυτό που θα μπορούσε να γίνει είναι να δημιουργηθούν συστήματα τα οποία θα πραγματοποιούν αυτόματους και απομακρυσμένους ελέγχους.

Το πιο κύριο σύστημα που πρέπει να δημιουργηθεί είναι αυτό το οποίο θα πραγματοποιεί ελέγχους στους σωλήνες που μεταφέρουν το νερό για τυχόν διαρροές πριν υπάρξει μεγαλύτερη διαρροή και έτσι «χαθούν» άσκοπα πολλά λίτρα νερού. Ένα άλλο σύστημα που είναι καλό να δημιουργηθεί είναι αυτό που θα ελέγχει την υγρασία του χώματος πριν αρχίσει το πότισμα του χωραφιού, καθώς και αν θα υπάρξει βροχή το επόμενο χρονικό διάστημα. Αυτά που αναφέραμε καθώς και κάποια άλλα παρόμοια συστήματα θα περιορίσουν σε σημαντικό βαθμό

την άσκοπη σπατάλη του νερού. Στον αγώνα της μείωσης σπατάλης νερού μπορεί να συμβάλει η επιστήμη της πληροφορικής και συγκεκριμένα ο κλάδος του διαδικτύου των πραγμάτων (Internet of Things - IoT). Η υλοποίηση των συστημάτων που αναφέραμε παραπάνω μπορεί να είναι εφικτή με τη χρήση τεχνολογιών IoT. Συγκεκριμένα, όσο αφορά το πρώτο σύστημα, θα μπορούσαν να τοποθετηθούν αισθητήρες στους σωλήνες είτε να δημιουργηθούν σωλήνες με ενσωματωμένους αισθητήρες οι οποίοι θα επικοινωνούν με ένα κεντρικό σύστημα στέλνοντας πληροφορίες για την κατάσταση που επικρατεί (στους σωλήνες) έτσι, ώστε, όταν υπάρξει η παραμικρή διαρροή, να ειδοποιηθούν οι αρμόδιες αρχές για να στείλουν όσο το δυνατόν πιο άμεσα συνεργείο αποκατάστασης. Εκτός του ότι θα προκληθεί μεγαλύτερη ζημιά στους σωλήνες αν δεν γίνουν άμεσα ενέργειες, πολύ πιθανό θα πάνε χαμένα πολλά λίτρα νερού μέχρι να γίνει αντιληπτή. Επίσης, αυτοί οι αισθητήρες θα μπορούσαν να ελέγχουν την υγεία των σωλήνων και να την παρουσιάζουν στους υπεύθυνους έτσι, ώστε όταν ένας σωλήνας έχει διαβρωθεί (που αυτό σημαίνει ότι σε μικρό χρονικό διάστημα θα υπάρξει διαρροή) να το γνωρίζουν και να προβούν σε ενέργειες είτε συντήρησης του είτε αλλαγής του, για να μην προκύψει η παραμικρή διαρροή και χαθεί το πόσιμο νερό. Το δεύτερο σύστημα που αναφέραμε το οποίο είναι και το αντικείμενο που θα αναπτύξουμε παρακάτω (σε μικρότερη κλίμακα βέβαια) έχει να κάνει με την εξοικονόμηση νερού στον κλάδο της γεωργίας, ο οποίος καταναλώνει το 75% της συνολικής κατανάλωσης νερού παγκοσμίως. Η φιλοσοφία του συστήματος αυτού είναι να τοποθετηθούν αισθητήρες σε όλο το χωράφι οι οποίοι όταν έρθει η ώρα του ποτίσματος να στέλνουν τις πληροφορίες της υγρασίας που άντλησαν από το χώμα στο κεντρικό σύστημα το οποίο με τη σειρά του θα διενεργήσει ελέγχους. Ο πρώτος έλεγχος που θα γίνεται θα είναι για το αν στο επόμενο διάστημα υπάρχει πιθανότητα βροχής. Αν ναι, να μην δώσει εντολή για πότισμα, αν όχι, τότε να προχωρήσει στον επόμενο έλεγχο ο οποίος θα ελέγξει το ποσοστό της υγρασίας του χώματος και είτε να ξεκινήσει συνολικά το πότισμα είτε μεμονωμένα στην περιοχή του χωραφιού που η υγρασία είναι κάτω από ένα ποσοστό το οποίο έχει οριστεί προηγουμένως.

1.2 – Στόχοι και σκοποί της πτυχιακής εργασίας

Ο κύριος στόχος της παρούσας πτυχιακής εργασίας είναι να περιοριστεί όσο το δυνατόν περισσότερο η άσκοπη σπατάλη του νερού του οποίου και η παραμικρή σταγόνα είναι πολύτιμη. Εφόσον ο αγροτικός τομέας καταναλώνει τη μεγαλύτερη ποσότητα νερού παγκοσμίως και πολλές φορές άσκοπα, γι’ αυτό το λόγο σκεφτήκαμε να υλοποιήσουμε την συγκεκριμένη εργασία. Η εργασία αυτή δημιουργεί ένα σύστημα το οποίο πριν ξεκινήσει το πότισμα ελέγχει καιρικά δεδομένα για να «δει» αν θα υπάρξει βροχή το επόμενο χρονικό διάστημα καθώς

και την υγρασία του χώματος. Σε περίπτωση που θα βρέξει ή που η υγρασία του χώματος δεν είναι κάτω από ένα όριο που έχει οριστεί προηγουμένως, δεν ανοίγει την παροχή νερού. Έτσι, με αυτό το τρόπο ξοδεύεται νερό μόνο όταν είναι απαραίτητο. Ένας άλλος στόχος της πτυχιακής είναι να κάνει γνωστή την έξυπνη γεωργία, τις τεχνικές της όπως και τα συστήματα της στον Ελλαδικό χώρο, στον οποίο δεν είναι ακόμα τόσο γνωστά τα πλεονεκτήματα της όπως και οι διευκολύνσεις που προσφέρει στους γεωργούς που την χρησιμοποιούν.

1.3 – Δομή της πτυχιακής εργασίας

Στο δεύτερο κεφάλαιο θα μιλήσουμε για τη λειτουργία του διαδικτύου των πραγμάτων (IoT), την ασφάλεια που παρέχει, καθώς και πώς θα βοηθήσει στην ανάπτυξη της «Έξυπνης γεωργίας». Στη συνέχεια του ίδιου κεφαλαίου θα πούμε τι εννοούμε με τον όρο «Έξυπνη γεωργία» καθώς και τους τρεις τομείς τεχνολογιών με τους οποίους είναι διασυνδεδεμένη (πληροφοριακά συστήματα διοίκησης, γεωργία ακριβείας και γεωργικοί αυτοματισμοί, καθώς και ρομποτική). Στο τέλος του κεφαλαίου, αναφέρουμε τις νέες γεωργικές τεχνολογίες οι οποίες θα δημιουργηθούν με την βοήθεια του IoT και θα στελεχώσουν στο μέλλον την «Έξυπνη γεωργία».

Στο τρίτο κεφάλαιο θα παρουσιάσουμε τις τεχνολογίες οι οποίες χρησιμοποιήθηκαν τόσο στο server side κομμάτι της πτυχιακής εργασίας όσο και στη δημιουργία της android εφαρμογής, για να είναι εφικτή η υλοποίηση του πρακτικού μέρους της παρούσας εργασίας.

Το τέταρτο κεφάλαιο χωρίζεται σε δύο μέρη. Στο πρώτο μέρος θα αναλύσουμε τα scripts που τρέχουν στον server, κάτω υπό ποιες προϋποθέσεις γίνεται η εκτέλεση τους καθώς και το κάθε πότε εκτελούνται. Και στο δεύτερο μέρος θα αναλύσουμε την android εφαρμογή που δημιουργήσαμε η οποία αλληλεπιδρά με τον server. Εκτός από την αλληλεπίδραση με τον server, αυτή η εφαρμογή παρέχει και άλλες υπηρεσίες όπως: Ο Καιρός τώρα, Πρόγνωση Καιρού, Αγροτικές Ειδήσεις, Ειδοποιήσεις εργασιών, Ατζέντα και Σημειωματάριο.

Στο πέμπτο και τελευταίο κεφάλαιο με το οποίο ολοκληρώνεται η παρούσα πτυχιακή εργασία, γίνεται ανασκόπηση στα όσο έχουν αναφερθεί στα προηγούμενα κεφάλαια, παραθέτονται συμπεράσματα και προτάσεις σε όσους θέλουν να δημιουργήσουν παρόμοιες εφαρμογές και μη. Τέλος, γίνεται αναφορά σε μελλοντικές επεκτάσεις της εφαρμογής οι οποίες θα μπορούσαν να εφαρμοστούν.

ΚΕΦΑΛΑΙΟ 2 – ΔΙΑΔΙΚΤΥΟ ΤΩΝ ΠΡΑΓΜΑΤΩΝ (IoT) & ΕΞΥΠΝΗ ΓΕΩΡΓΙΑ

2.1 – Εισαγωγή

Σε αυτό το κεφάλαιο θα μιλήσουμε για το διαδίκτυο των πραγμάτων, πώς προέκυψε, πώς λειτουργεί, καθώς και κάποιες εφαρμογές του στην κοινωνία. Στη συνέχεια, θα αναφέρουμε τι είναι η έξυπνη γεωργία και πώς συμβάλει το διαδίκτυο των πραγμάτων στην υλοποίηση της, καθώς και τις τεχνολογίες οι οποίες συνεργάζονται έτσι, ώστε να είναι εφικτή. Επίσης, θα πούμε λίγα λόγια για τους τρεις τομείς οι οποίοι συνδέονται στενά με την έξυπνη γεωργία. Οι τομείς αυτοί είναι: πληροφοριακά συστήματα διοίκησης, γεωργία ακριβείας και γεωργικοί αυτοματισμοί καθώς και ρομποτική. Τέλος, παρακάτω θα μιλήσουμε για τις νέες τεχνολογίες της έξυπνης γεωργίας. Οι οποίες είναι: αυτόνομη και ρομποτική εργασία, τρακτέρ χωρίς οδηγό, σπορά και φύτευση, αυτόματη άρδευση, έλεγχος ζιζανίων και συντήρηση καλλιεργειών, συγκομιδή και τα drones.

2.2 – Διαδίκτυο των πραγμάτων

Το Διαδίκτυο των Πραγμάτων ή Ίντερνετ των Πραγμάτων (Internet of Things - IoT) αποτελεί το δίκτυο επικοινωνίας πληθώρας συσκευών. Όπως, οικιακών συσκευών, αυτοκινήτων, καθώς και κάθε αντικειμένου που ενσωματώνει ηλεκτρονικά μέσα, λογισμικό, αισθητήρες και συνδεσιμότητα σε δίκτυο, ώστε να επιτρέπεται η σύνδεση και η ανταλλαγή δεδομένων. Απλούστερα, η φιλοσοφία του IoT είναι η σύνδεση όλων των ηλεκτρονικών συσκευών μεταξύ τους (τοπικό δίκτυο) ή και ακόμη η σύνδεση στο διαδίκτυο (παγκόσμιο ιστό) [1].

2.2.1 – Λίγη ιστορία

Ο όρος Διαδίκτυο των Πραγμάτων επινοήθηκε στα τέλη της δεκαετίας του 1990 από τον επιχειρηματία Kevin Ashton. Ο Ashton, ο οποίος είναι ένας από τους ιδρυτές του Auto-ID Center στο MIT, ήταν μέρος μιας ομάδας που ανακάλυψε τον τρόπο να συνδέσει τα αντικείμενα με το διαδίκτυο μέσω μιας ετικέτας RFID. Έχει δηλώσει ότι χρησιμοποίησε πρώτη φορά τη φράση Internet of Things σε μια παρουσίαση που έκανε το 1999 – και ο όρος αυτός έχει καθιερωθεί από τότε [2].

2.2.2 – Η έννοια «Things»

Η έννοια "Things" (πράγματα) δεν είναι αυστηρά συνδεδεμένη με ορισμένα προϊόντα. Αναφέρεται σε μία ευρεία ποικιλία συσκευών εντελώς διαφορετικών μεταξύ τους, όπως για παράδειγμα κάμερες, κλιματιστικά, φώτα, συστήματα ασφαλείας, smartwatches, ακόμα και αυτοκίνητα με περίπλοκους αισθητήρες οι οποίοι εντοπίζουν αντικείμενα στην πορεία τους. Όλα τα προαναφερθέντα είναι μερικά από τα πολλά προϊόντα τεχνολογίας. Βασικό χαρακτηριστικό όλων είναι η σύνδεση μεταξύ τους με απώτερο σκοπό την δυνατότητα του χρήστη να τα ελέγχει από έναν υπολογιστή ή κινητό [1].

Εικόνα 1 – Διαδίκτυο των πραγμάτων. Σύνδεση από παντού

2.2.3 – Πώς λειτουργεί;

Οι συσκευές και τα αντικείμενα με ενσωματωμένους αισθητήρες συνδέονται με μια πλατφόρμα, η οποία περιλαμβάνει δεδομένα από τις διάφορες συσκευές και εφαρμόζει επεξεργασία και διαμοιρασμό των πληροφοριών σε εφαρμογές οι οποίες έχουν δημιουργηθεί για την αντιμετώπιση συγκεκριμένων αναγκών. Οι συσκευές και τα αντικείμενα αυτά λόγω της συνεχόμενης λειτουργίας τους παράγουν μεγάλο όγκο δεδομένων. Για να πραγματοποιηθεί η επεξεργασία αυτού του μεγάλου όγκου δεδομένων χρησιμοποιούνται οι τεχνολογίες analyticks. Στην παραδοσιακή ανάλυση, τα δεδομένα αποθηκεύονται και μετά αναλύονται. Ωστόσο, στην περίπτωση των δεδομένων συνεχούς ροής (streaming data) όπως αυτά του IoT, τα μοντέλα και οι αλγόριθμοι είναι αυτοί που αποθηκεύονται και τα δεδομένα περνούν μέσα από αυτά για ανάλυση. Όταν ολοκληρωθεί η ανάλυση των εισαχθέντων δεδομένων, οι αλγόριθμοι βγάζουν κάποιο συμπέρασμα το οποίο το διαθέτουν στις συσκευές οι οποίες είναι συνδεδεμένες στην πλατφόρμα, για να το χρησιμοποιήσουν [3].

2.2.4 – Ασφάλεια

Κάθε συσκευή που μεταφέρει πληροφορίες στο δημόσιο διαδίκτυο κινδυνεύει άμεσα ή έμμεσα από κλοπή πληροφοριών. Ήδη έχουν σημειωθεί περιπτώσεις εφαρμογών όπου «χάκερ» κατάφεραν να υποκλέψουν πληροφορίες κατά τη μετάδοσή τους, μεταξύ συσκευών. Προς το παρόν, δεν υπάρχουν διεθνή πρότυπα ασφαλείας στο διαδίκτυο των πραγμάτων για πολλούς λόγους με τον σημαντικότερο να είναι πως η τεχνολογία αυτή βρίσκεται σε εμβρυακή μορφή ακόμα. Έως ότου καθιερωθούν διεθνή πρότυπα, κάθε κατασκευαστής θα ενσωματώνει δικούς του τρόπους κρυπτογράφησης άλλοτε με επιτυχία, άλλοτε χωρίς αποτέλεσμα. Αυτός είναι και ο λόγος που προς το παρόν οι εφαρμογές του διαδικτύου των πραγμάτων επικεντρώνονται κυρίως σε εφαρμογές τηλεμετρίας, καθώς σε μια τέτοια εφαρμογή η διαρροή δεδομένων δεν προκαλεί μεγάλο κίνδυνο. Καθώς όμως οι έξυπνες συσκευές εισέρχονται όλο και περισσότερο στη καθημερινή μας ζωή και συλλέγουν δεδομένα για τη καθημερινότητά μας, υπάρχει ενδεχόμενος κίνδυνος διαρροής είτε πολύ προσωπικών δεδομένων (συστήματα παρακολούθησης, πληροφορίες υγείας, στίγμα GPS, κ.α.) είτε λιγότερο προσωπικών (έξυπνοι λαμπτήρες, εξοικονόμηση νερού, εξοικονόμηση ενέργειας, κ.α.) [4].

2.3 – Έξυπνη γεωργία

Στην προηγούμενη ενότητα μιλήσαμε για το τι είναι το διαδίκτυο των πραγμάτων, πώς λειτουργεί καθώς και κάποιες εφαρμογές του. Σε αυτήν την ενότητα θα μιλήσουμε για την συμβολή του διαδικτύου των πραγμάτων στην έξυπνη γεωργία.

2.3.1 – Τι είναι η έξυπνη γεωργία;

Η Έξυπνη Γεωργία αντιπροσωπεύει την εφαρμογή των σύγχρονων Τεχνολογιών Πληροφορίας και Επικοινωνιών (ΤΠΕ) στην γεωργία, που οδηγεί σε αυτό που μπορεί να ονομαστεί ως τρίτη Πράσινη Επανάσταση. Μετά τις επαναστάσεις αναπαραγωγής των φυτών και της γενετικής, αυτή η τρίτη Πράσινη Επανάσταση αρχίζει και επιβάλλεται στο γεωργικό κόσμο με βάση τη συνδυασμένη εφαρμογή των λύσεων ΤΠΕ όπως ο εξοπλισμός ακριβείας, το Διαδίκτυο των Πραγμάτων (Internet of Things – IoT), οι αισθητήρες και ενεργοποιητές, τα συστήματα γεω-εντοπισμού, τα Μεγάλα Δεδομένα (Big Data), τα μη επανδρωμένα εναέρια οχήματα (Unmanned Aerial Vehicles – UAV), η ρομποτική, κλπ. Η Έξυπνη Γεωργία έχει πραγματική δυνατότητα να δώσει πιο μεγάλη και βιώσιμη γεωργική παραγωγή, η οποία θα βασίζεται σε μια προσέγγιση πιο ακριβούς και αποδοτικής χρήσης των πόρων. Ωστόσο, ενώ στις ΗΠΑ το 20-80% των γεωργών χρησιμοποιούν κάποιο είδος Τεχνολογίας Έξυπνης Γεωργίας, στην Ευρώπη μόνο το 0-24% των γεωργών τις χρησιμοποιεί. Από την πλευρά του γεωργού, η Έξυπνη Γεωργία θα πρέπει να παρέχει στο γεωργό προστιθέμενη αξία με τη μορφή της καλύτερης λήψης αποφάσεων ή της πιο αποτελεσματικής λειτουργίας και διαχείρισης της εκμετάλλευσής του. Με αυτή την έννοια, η Έξυπνη Γεωργία συνδέεται στενά με τρεις διασυνδεδεμένους τομείς τεχνολογιών:

- **Πληροφοριακά Συστήματα Διοίκησης:** Προγραμματισμένα συστήματα για τη συλλογή, επεξεργασία, αποθήκευση και διάδοση των δεδομένων με τη μορφή που απαιτείται για την εκτέλεση των εργασιών και λειτουργιών μιας γεωργικής επιχείρησης.
- **Γεωργία Ακριβείας:** Υποστηρίζει τους αγρότες με πιο αειφόρες πρακτικές. Δηλαδή πρακτικές παραγωγής ενός αγαθού, με τέτοιο τρόπο, ώστε να μην μειώνεται, αλλά να βελτιώνεται η παραγωγική ικανότητα του τόπου και ταυτόχρονα να μην επηρεάζεται το περιβάλλον. Η γεωργία ακριβείας έχει στόχο την επίτευξη περισσότερων αποτελεσμάτων στις γεωργικές δραστηριότητες με την παράλληλη χρήση λιγότερων πόρων, όπως νερό, ενέργεια, λιπάσματα και φυτοφάρμακα. Παρέχει δηλαδή την δυνατότητα για μια πιο μεγάλη και βιώσιμη γεωργική παραγωγή, η οποία θα βασίζεται σε μια προσέγγιση πιο ακριβούς και αποδοτικής χρήσης των πόρων. Χρησιμοποιεί κυρίως την τεχνολογία, τις πληροφορίες, τα δεδομένα από

δορυφόρους και το διαδίκτυο. Στόχος δεν είναι μόνο να διευκολυνθούν οι αγρότες, αλλά και να αυξηθεί η απόδοση και η ποιότητα με την αποτελεσματικότερη δυνατή χρήση των πόρων. Ως πρόσθετο πλεονέκτημα, βελτιώνουν επίσης την ποιότητα ζωής των εργαζομένων στη γεωργία, μειώνοντας τη βαριά εργασία και τα κουραστικά καθήκοντα.

- **Γεωργικοί αυτοματισμοί και ρομποτική:** Η διαδικασία της εφαρμογής της ρομποτικής, του αυτόματου ελέγχου και των τεχνικών τεχνητής νοημοσύνης σε όλα τα επίπεδα της γεωργικής παραγωγής, συμπεριλαμβανομένων των farmbots και των farmdrones.

Οι εφαρμογές Έξυπνης Γεωργίας δεν στοχεύουν μόνο σε μεγάλες γεωργικές εκμεταλλεύσεις, αλλά θα μπορούσαν να δράσουν για την τόνωση της οικογενειακής γεωργίας. Η Έξυπνη Γεωργία μπορεί επίσης να παρέχει μεγάλα οφέλη σχετικά με το περιβάλλον, μέσω της αποτελεσματικότερης χρήσης του νερού, ή της βελτιστοποίησης των γεωργικών πρακτικών [5].

2.4 – Νέες τεχνολογίες γεωργίας

Συμπερασματικά, ο πληθυσμός της γης συνεχώς αυξάνεται (7,5 δισεκατομμύρια). Λαμβάνοντας υπόψιν τις προβλέψεις των Ηνωμένων Εθνών, ότι θα φθάσουμε τα 9,7 δισεκατομμύρια μέχρι το 2050, ένας πληθυσμός αυτού του μεγέθους έχει πολλές προκλήσεις. Ο Οργανισμός Τροφίμων και Γεωργίας του ΟΗΕ προβλέπει ότι πρέπει να ενισχύσουμε την παγκόσμια παραγωγή τροφίμων κατά 70% τις επόμενες δεκαετίες προκειμένου να τροφοδοτήσουμε τον αναμενόμενο πληθυσμό του 2050. Για να μπορέσει να αντιμετωπιστεί η αύξηση του πληθυσμού, καθώς και η κλιματική αλλαγή, θα πρέπει να χρησιμοποιηθούν νέες τεχνολογίες οι οποίες θα βοηθήσουν στην άρδευση, την φυτοπροστασία, καθώς και τη συγκομιδή. Μερικές από τις νέες τεχνολογίες θα τις παρουσιάσουμε παρακάτω [5].

2.4.1 – Αυτόνομη και ρομποτική εργασία

Η αντικατάσταση της ανθρώπινης εργασίας με την αυτοματοποιημένη είναι μια αυξανόμενη τάση σε πολλές βιομηχανίες και η γεωργία δεν αποτελεί εξαίρεση. Οι περισσότερες πτυχές της γεωργίας είναι εξαιρετικά έντονες, με μεγάλο μέρος της εργασίας να αποτελείται από επαναλαμβανόμενα και τυποποιημένα καθήκοντα - μια ιδανική θέση για τη ρομποτική και τον αυτοματισμό. Βλέπουμε ήδη γεωργικά ρομπότ - ή AgBots - που αρχίζουν να εμφανίζονται σε αγροκτήματα και εκτελούν καθήκοντα που κυμαίνονται από φύτευση και άρδευση έως συγκομιδή και διαλογή. Τελικά, αυτό το νέο κύμα έξυπνου εξοπλισμού θα καταστήσει δυνατή την

παραγωγή περισσότερων και υψηλότερης ποιότητας τροφίμων με λιγότερο ανθρώπινο δυναμικό [6].

Εικόνα 2 – Αυτόνομη ρομποτική εργασία

2.4.2 – Τρακτέρ χωρίς οδηγό

Τα τρακτέρ είναι η καρδιά της αγροτικής εκμετάλλευσης τα οποία χρησιμοποιούνται για πολλές διαφορετικές εργασίες, ανάλογα με το είδος της εκμετάλλευσης και τη διαμόρφωση του βοηθητικού εξοπλισμού της. Καθώς προωθούνται οι αυτόνομες τεχνολογίες οδήγησης, τα τρακτέρ αναμένεται να καταστούν μερικές από τις πρώτες μηχανές που πρέπει να μετατραπούν σε αυτόνομες μηχανές.

Σε πρώιμα στάδια, θα χρειαστεί ακόμα ανθρώπινη προσπάθεια για τη δημιουργία τοπογραφικών χαρτών χωραφιού και ορίων, για τον προγραμματισμό των καλύτερων διαδρομών μέσα στο χωράφι χρησιμοποιώντας λογισμικό προγραμματισμού διαδρομών και για τη λήψη άλλων λειτουργικών συνθηκών. Οι άνθρωποι θα εξακολουθούν να απαιτούνται για τακτική επισκευή και συντήρηση.

Ωστόσο, οι αυτόνομοι ελκυστήρες θα γίνουν πιο ικανοί και αυτοσυντηρούμενοι με την πάροδο του χρόνου, ειδικά με τη συμπερίληψη πρόσθετων μηχανών και συστημάτων μηχανικής όρασης, GPS πλοήγησης, διασύνδεσης IoT για την απομακρυσμένη παρακολούθηση και λειτουργία και ραντάρ και LiDAR για ανίχνευση και αποφυγή αντικειμένων. Όλες αυτές οι τεχνολογικές εξελίξεις θα μειώσουν σημαντικά την ανάγκη των ανθρώπων να ελέγχουν ενεργά αυτά τα μηχανήματα.

Απώτερος σκοπός είναι αυτοί οι ελκυστήρες να χρησιμοποιούν στο μέλλον «μεγάλα δεδομένα» όπως π.χ. δεδομένα σε πραγματικό χρόνο, καλύτερη χρήση των ιδανικών συνθηκών, ανεξάρτητα από την ανθρώπινη εισροή και ανεξάρτητα από την ώρα της ημέρας [7].

Εικόνα 3 – Τρακτέρ χωρίς οδηγό

2.4.3 – Σπορά και φύτευση

Η σπορά ήταν κάποτε μια επίπονη χειρωνακτική διαδικασία. Η σύγχρονη γεωργία βελτιώθηκε σε αυτό με μηχανές σποράς, οι οποίες μπορούν να καλύψουν περισσότερο έδαφος πολύ πιο γρήγορα από τον άνθρωπο. Ωστόσο, αυτές συχνά χρησιμοποιούν μια μέθοδο διασκορπισμού που μπορεί να είναι ανακριβής και σπάταλη όταν οι σπόροι βρίσκονται εκτός της βέλτιστης τοποθεσίας. Η αποτελεσματική σπορά απαιτεί έλεγχο πάνω σε δύο μεταβλητές: φύτευση σπόρων στο σωστό βάθος και φυτά σε κατάλληλη απόσταση ώστε να επιτρέπεται η βέλτιστη ανάπτυξη.

Ο εξοπλισμός σποράς ακριβείας έχει σχεδιαστεί, για να μεγιστοποιεί τις μεταβλητές αυτές κάθε φορά. Ο συνδυασμός δεδομένων γεωμετρίας και αισθητήρων που αναλύουν την ποιότητα του εδάφους, την πυκνότητα, την υγρασία και τα επίπεδα θρεπτικών ουσιών απαιτεί πολλές εικασίες από τη διαδικασία σποράς.

Οι σπόροι έχουν την καλύτερη πιθανότητα να φυτρώσουν και να αναπτυχθούν και η συνολική καλλιέργεια θα έχει μεγαλύτερη συγκομιδή.

Καθώς η γεωργία μετακινείται στο μέλλον, οι υπάρχοντες σπαρτικές μηχανές ακριβείας θα έρθουν σε επαφή με αυτόνομους ελκυστήρες και συστήματα με δυνατότητα IoT που θα τροφοδοτούν τις πληροφορίες πίσω στον αγρότη. Ένα ολόκληρο χωράφι θα μπορούσε να φυτευτεί με αυτό τον τρόπο, με μόνο έναν άνθρωπο να παρακολουθεί τη διαδικασία μέσω μιας τροφοδοσίας βίντεο ή ενός πίνακα ψηφιακού ελέγχου σε έναν υπολογιστή ή tablet, ενώ πολλές μηχανές κυλούν στο χωράφι ταυτόχρονα [8].

Εικόνα 4 – Σπορά και φύτευση

2.4.4 – Αυτόματη άρδευση

Η στάγδην άρδευση είναι ήδη μια επικρατούσα μέθοδος άρδευσης που επιτρέπει στους αγρότες να ελέγχουν πότε και πόσο νερό λαμβάνουν οι καλλιέργειες τους. Συνδυάζοντας αυτά τα συστήματα SDI με ολοένα και πιο εξελιγμένους αισθητήρες με δυνατότητα IoT, ώστε να παρακολουθούν συνεχώς τα επίπεδα υγρασίας και την υγεία των φυτών, οι αγρότες θα μπορούν να παρεμβαίνουν μόνο όταν είναι απαραίτητο, επιτρέποντας έτσι το σύστημα να λειτουργεί αυτόνομα.

Ενώ τα συστήματα SDI δεν είναι ακριβώς ρομποτικά, θα μπορούσαν να λειτουργήσουν εντελώς αυτόνομα σε ένα έξυπνο αγρόκτημα, βασιζόμενα σε δεδομένα από αισθητήρες που υφίστανται γύρω από τα χωράφια, για να εκτελούν άρδευση ανάλογα με τις εκάστοτε αρδευτικές ανάγκες [9].

Εικόνα 5 – Αυτόματο πότισμα

2.4.5 – Έλεγχος ζιζανίων και συντήρηση καλλιεργειών

Ο έλεγχος των ζιζανίων και των παρασίτων είναι κρίσιμες πτυχές της συντήρησης των καλλιεργειών και είναι αυτές οι αγροτικές εργασίες που είναι ιδανικές για αυτόνομα ρομπότ.

Έχει κατασκευαστεί, παραδείγματος χάριν, ένα ρομπότ περίπου στο μέγεθος ενός αυτοκινήτου και μπορεί να πλοηγηθεί αυτόνομα μέσα από ένα χωράφι χρησιμοποιώντας βίντεο, LiDAR και δορυφορικό GPS. Οι προγραμματιστές του χρησιμοποιούν τη μηχανική μάθηση, για να διδάξουν σε αυτό να εντοπίσει τα ζιζάνια πριν τα αφαιρέσει. Με την προηγμένη μηχανική μάθηση ή ακόμα και την τεχνητή νοημοσύνη (AI) που ενσωματώνεται στο μέλλον, μηχανήματα όπως αυτό θα μπορούσαν να αντικαταστήσουν πλήρως την ανάγκη για ανθρώπους να παρακολουθούν καλλιέργειες ή να κάνουν την παραπάνω διαδικασία χειροκίνητα.

Ένα άλλο αντίστοιχο ρομπότ, που έχει κατασκευασθεί, λειτουργεί λίγο διαφορετικά. Ο καλλιεργητής τους ρυμουλκείται πίσω από έναν ελκυστήρα και είναι εφοδιασμένος με συστήματα απεικόνισης που μπορούν να αναγνωρίσουν μια φθορίζουσα χρωστική ουσία την οποία καλύπτουν οι σπόροι όταν φυτεύονται και η οποία μεταφέρεται στα νεαρά φυτά καθώς αυτά φυτρώνουν και αρχίζουν να αναπτύσσονται. Ο καλλιεργητής στη συνέχεια κόβει τα μη λαμπερά ζιζάνια.

Οι παραπάνω ρομποτικές μηχανές είναι σχεδιασμένες τόσο για τον έλεγχο των ζιζανίων όσο και για το εντοπισμό εχθρών και ασθενειών, χρησιμοποιώντας τους κατάλληλους κάθε φορά αισθητήρες [9].

Εικόνα 6 – Έλεγχος ζιζανίων και συντήρηση καλλιεργειών

2.4.6 – Συγκομιδή

Υπάρχει ένα πλήθος μηχανών οι οποίες είναι σε θέση να συγκομίζουν μηχανικά τους καρπούς που είναι έτοιμοι για συγκομιδή.

Η ανάπτυξη τεχνολογίας ικανής για ευαίσθητες εργασίες συγκομιδής, όπως η συλλογή καρπών από δέντρα ή λαχανικά όπως οι ντομάτες, είναι όπου τα υψηλής τεχνολογίας αγροκτήματα θα διαπρέψουν πραγματικά. Οι μηχανικοί εργάζονται για να δημιουργήσουν τα σωστά ρομποτικά εξαρτήματα για αυτά τα εξελιγμένα καθήκοντα, όπως το ρομπότ συλλογής τομάτας της Panasonic, το οποίο ενσωματώνει εκλεπτυσμένες κάμερες και αλγόριθμους για τον προσδιορισμό του χρώματος, του σχήματος και της θέσης της ντομάτας για τον προσδιορισμό της ωριμότητάς της [7].

Εικόνα 7 – Συγκομιδή

2.4.7 – Μείωση της εργασίας, αύξηση της παραγωγής και της αποδοτικότητας

Η βασική ιδέα της ενσωμάτωσης της αυτόνομης ρομποτικής στη γεωργία παραμένει ο στόχος της μείωσης της εξάρτησης από τη χειρωνακτική εργασία, ενώ παράλληλα η επίτευξη της αύξησης της παραγωγής, της απόδοσης και της ποιότητας του προϊόντος.

Ο αγρότης του μέλλοντος θα ξοδέψει το χρόνο του εκτελώντας καθήκοντα όπως η επισκευή μηχανημάτων, η σωστή χρήση ρομποτικών μηχανών, η ανάλυση δεδομένων και ο αποτελεσματικός σχεδιασμός των αγροτικών επιχειρήσεων.

Όπως σημειώνεται με όλα αυτά τα γεωργικά ρομπότ, η ύπαρξη μιας σωστής βάσης αισθητήρων και IoT ενσωματωμένων στην υποδομή της γεωργικής εκμετάλλευσης είναι απαραίτητη. Το κλειδί για μια πραγματικά "έξυπνη" εκμετάλλευση βασίζεται στην ικανότητα όλων των μηχανών και των αισθητήρων να επικοινωνούν μεταξύ τους και με τον αγρότη, ακόμη και όταν αυτοί λειτουργούν αυτόνομα [8].

Εικόνα 8 – Διασύνδεση μηχανών και αισθητήρων

2.4.8 – Drones

Τα drones (μη επανδρωμένα αεροσκάφη) μπορούν να προσφέρουν μεγάλη βοήθεια στον αγρότη. Όπως η δυνατότητα να επιβλέπει την εκμετάλλευση του από ψηλά. Αυτό επιτυγχάνεται με τη δυνατότητα των drones να φέρουν κάμερα. Πολλές φορές, οι κάμερες που φέρουν εκτός από τη τυπική φωτογραφική απεικόνιση έχουν τη δυνατότητα της υπέρυθρης, υπεριώδους, καθώς και της υπερφυσικής απεικόνισης. Επίσης, μερικές απ' αυτές τις κάμερες που φέρουν μπορούν να γράψουν και βίντεο. Όλοι αυτοί οι διαφορετικοί τύποι απεικόνισης επιτρέπουν στους αγρότες να συλλέγουν λεπτομερή δεδομένα, ενισχύοντας τις δυνατότητές τους για την παρακολούθηση της υγείας των καλλιεργειών, την αξιολόγηση της ποιότητας του εδάφους και τον προγραμματισμό των τοποθεσιών φύτευσης για τη βελτιστοποίηση των πόρων και της χρήσης γης. Η ικανότητά για τακτική εκτέλεση αυτών των επιτόπιων ερευνών, βελτιώνει τον προγραμματισμό των σχεδίων φύτευσης σπόρων, καθώς και της άρδευσης.

Εικόνα 9 – Drone για απεικόνιση και φύτευση

Επίσης, τα drones θα μπορούσαν να χρησιμοποιηθούν για να εκτοξεύουν προς φύτευση το σπόρο στην κατάλληλη θέση και σημείο του χωραφιού. Με το IoT και το λογισμικό για αυτόνομη λειτουργία, ένας στόλος αεροσκαφών θα μπορούσε να ολοκληρώσει εξαιρετικά ακριβή φύτευση στις ιδανικές συνθήκες για την ανάπτυξη κάθε καλλιέργειας. Ακόμα, υπάρχουν και άλλα drones τα οποία είναι κατάλληλα κατασκευασμένα για ψεκασμούς καλλιεργειών. Χρησιμοποιώντας ένα συνδυασμό GPS, μέτρησης με λέιζερ και τοποθέτηση υπερήχων, τα αεροσκάφη ψεκασμού καλλιεργειών μπορούν εύκολα να προσαρμοστούν σε υψόμετρο και θέση, προσαρμοζόμενα όσον αφορά μεταβλητές όπως η ταχύτητα του ανέμου, η τοπογραφία και η γεωγραφία. Αυτό επιτρέπει στα αεροσκάφη να εκτελούν πιο αποτελεσματικά τις εργασίες ψεκασμού καλλιεργειών και με μεγαλύτερη ακρίβεια. Μια άλλη χρήσιμη εργασία που μπορεί να κάνει ένα drone είναι η εξ αποστάσεως παρακολούθηση και ανάλυση αγρών και καλλιεργειών. Ας φανταστούμε τα οφέλη από τη χρήση ενός μικρού στόλου αεροσκαφών αντί μιας ομάδας εργαζομένων που ξοδεύουν ώρες στα πόδια τους ή σε ένα όχημα που κινείται κατά μήκος των καλλιεργειών, για να ελέγξει οπτικά τις συνθήκες καλλιέργειας.

Δεδομένου ότι τα αεροσκάφη για γεωργική χρήση είναι ακόμη νωρίς στην εξέλιξή τους, υπάρχουν μερικά μειονεκτήματα. Οι διαδρομές και οι χρόνοι πτήσεων δεν είναι τόσο ισχυροί όσο θα χρειάζονταν πολλά αγροκτήματα. Στη παρούσα φάση, η μέγιστη διάρκεια διαδρομής αγγίζει τη μία ώρα οπότε και το drone θα πρέπει να επαναφορτιστεί. Τέλος, το κόστος τους αποτελεί ακόμη ένα θέμα, το οποίο στη πορεία λογικά θα προσπεραστεί [10].

Εικόνα 10 – Drone για παρακολούθηση σε πραγματικό χρόνο

2.5 – Το συνδεδεμένο χωράφι

Η έξυπνη φάρμα θα έχει ενσωματωμένους αισθητήρες σε κάθε στάδιο των καλλιεργητικών διαδικασιών και σε κάθε εξοπλισμό. Οι αισθητήρες που είναι εγκατεστημένοι σε όλα τα χωράφια θα συλλέγουν δεδομένα σχετικά με το επίπεδο φωτισμού, τις συνθήκες του εδάφους, την άρδευση, την ποιότητα του αέρα και τον καιρό. Αυτά τα δεδομένα θα επιστρέφουν στον αγρότη, ή απευθείας σε εξειδικευμένα γεωργικά ρομπότ στο χωράφι. Ομάδες απ' αυτά τα ρομπότ θα διασχίζουν τα χωράφια και θα εργάζονται αυτόνομα για να ανταποκριθούν στις ανάγκες των καλλιεργειών και θα εκτελούν λειτουργίες ζιζανιοκτονίας, άρδευσης, κλαδέματος και συγκομιδής, οι οποίες και θα καθοδηγούνται από τη δική τους συλλογή αισθητήρων, πλοήγησης και δεδομένων των καλλιεργειών.

Εικόνα 11 – Διάγραμμα συνδεδεμένου χωραφιού

Drones θα περιηγούνται στον ουρανό και θα συλλέγουν στοιχεία και δεδομένα όσον αφορά στην υγεία των φυτών και τις εδαφολογικές συνθήκες, ή θα δημιουργούν χάρτες που θα καθοδηγούν τα ρομπότ και θα βοηθούν τους αγρότες να σχεδιάσουν τα επόμενα βήματα για τη γεωργική τους εκμετάλλευση. Όλα αυτά θα συμβάλουν στη δημιουργία μεγαλύτερης καλλιέργειας φυτών και στην αύξηση της διαθεσιμότητας και της ποιότητας των τροφίμων. Η συνεργασία όλων αυτών θα δίνει τη δυνατότητα στους αγρότες, να ελέγχουν μέσω του κινητού τους για παράδειγμα αν τα φυτά είναι υγιή ή χρειάζονται προσοχή, αν κάποιο χωράφι χρειάζεται νερό, κτλ. και να λαμβάνουν τεκμηριωμένες απαντήσεις [8].

Εικόνα 12 – Έλεγχος αγροκτήματος απομακρυσμένα

Επίλογος

Καινοτόμα και αυτόνομα γεωργικά ρομπότ είναι σίγουρα χρήσιμα μηχανήματα, αλλά αυτό που θα κάνει πραγματικά έξυπνο το μελλοντικό αγρόκτημα, είναι αυτό που φέρνει όλη αυτή την τεχνολογία μαζί: το **Διαδίκτυο των πραγμάτων (IoT)**. Το IoT έχει γίνει κομμάτι ενός συνόλου όρων για την ιδέα της ύπαρξης ηλεκτρονικών υπολογιστών, μηχανών, εξοπλισμού και συσκευών παντός τύπου, ανταλλαγής δεδομένων και επικοινωνίας με τρόπους που τους επιτρέπουν να λειτουργούν ως το λεγόμενο «έξυπνο σύστημα». Βλέπουμε ήδη τεχνολογίες IoT που χρησιμοποιούνται με πολλούς τρόπους, όπως έξυπνα σπίτια και ψηφιακοί βοηθοί, έξυπνα εργοστάσια και έξυπνες ιατρικές συσκευές. Ωραία είναι τα θεωρητικά, αλλά ας πούμε και λίγο πιο τεχνικά θέματα.

ΚΕΦΑΛΑΙΟ 3 – ΤΕΧΝΟΛΟΓΙΕΣ ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΑΝ

3.1 – Εισαγωγή

Σε αυτό το κεφάλαιο θα περιγράψουμε τις τεχνολογίες που χρησιμοποιήσαμε για να ολοκληρώσουμε τις υπηρεσίες οι οποίες στελεχώνουν την android εφαρμογή μας, με όσο το δυνατόν πιο απλό και κατανοητό τρόπο, χωρίς να εμβαθύνουμε πολύ σε τεχνικές λεπτομέρειες έτσι, ώστε οι αναγνώστες να πάρουν τη βασική γνώσει περί των τεχνολογιών και στη συνέχεια αν επιθυμούν οι ίδιοι να κάνουν περαιτέρω έρευνα.

3.2 – Debian linux server

Το Debian, αποτέλεσμα του Debian Project, είναι μια δημοφιλής διανομή Linux, ελεύθερο λογισμικό που αναπτύσσεται μέσω της συνεργασίας εθελοντών από όλο τον κόσμο. Βασίζεται στον πυρήνα linux και στην ομάδα βασικών εργαλείων του εγχειρήματος GNU.

Το Debian είναι γνωστό για την αφοσίωσή του στη φιλοσοφία του Unix και του ελεύθερου λογισμικού. Είναι επίσης γνωστό για το πλήθος επιλογών και δυνατοτήτων που προσφέρει. Η τρέχουσα έκδοση περιλαμβάνει πάνω από 29.000 πακέτα λογισμικού για δώδεκα αρχιτεκτονικές υπολογιστών που το φάσμα τους κυμαίνεται από αρχιτεκτονική ARM, που διαθέτουν συνήθως τα ενσωματωμένα συστήματα και αρχιτεκτονική κεντρικού υπολογιστή IBM s390 μέχρι τις πιο κοινές αρχιτεκτονικές x86 και PowerPC που υπάρχουν στους σύγχρονους προσωπικούς υπολογιστές.

Είναι επίσης πολύ γνωστό για το σύστημα διαχείρισης πακέτων του και για το APT (Advanced Packaging Tool, προηγμένο εργαλείο πακέτων) που διαθέτει· συγκεκριμένα, για τις αυστηρές πολιτικές που υιοθετεί ως προς την ποιότητα των πακέτων και των εκδόσεων του και την ανοιχτή διαδικασία ανάπτυξης και ελέγχου που υιοθετεί. Αυτές οι πρακτικές κάνουν πιο εύκολες τις αναβαθμίσεις και την εγκατάσταση ή αφαίρεση πακέτων. Το Debian υποστηρίζεται από δωρεές που γίνονται μέσω οργανισμών που προωθούν το ελεύθερο λογισμικό. Δεν υποστηρίζεται από κάποια εταιρία, αλλά από το Debian Project και τον οργανισμό Software in the Public Interest.

Το Debian ως σχέδιο εργασίας άρχισε το 1993 από τον Ίαν Μέρντοκ, φοιτητή τότε του πανεπιστημίου Purdue, όταν έγραψε το Μανιφέστο Debian το οποίο καλούσε για τη δημιουργία μιας διανομής linux η οποία θα αναπτύσσονταν με τρόπο

ανοιχτό στο πνεύμα του GNU/Linux. Διάλεξε το όνομα συνδυάζοντας το όνομα της τότε φιλενάδας του Ντέμπρα (**Deb**ra) με το δικό του (**Ian**).

Επίσης, χωρίζεται σε τρεις διαφορετικές εκδόσεις:

- **Stable (σταθερή)**: αυτή είναι η τελευταία επίσημη έκδοση. Τα προγράμματα της έκδοσης σπανίως ανανεώνονται (εκτός από επείγουσες διορθώσεις ασφάλειας).
- **Testing (υπό δοκιμή)**: η έκδοση αυτή περιλαμβάνει όλα τα πακέτα (προγράμματα) που δεν θεωρούνται ακόμα αρκετά σταθερά και χρειάζονται ακόμα έλεγχο.
- **Unstable ή sid (still in development)**: περιλαμβάνει τα πακέτα που είναι υπό ανάπτυξη. Η έκδοση αυτή πρέπει να θεωρείται ως πολύ ασταθής και να χρησιμοποιείται από έμπειρους χρήστες.
- **Experimental (πειραματική)**: Η έκδοση αυτή χρησιμοποιείται μόνο για την προετοιμασία των πακέτων που θα προστεθούν στην unstable [11].

3.3 – Python

Η Python είναι διερμηνευόμενη (interprited), γενικού σκοπού (general-purpose) και υψηλού επιπέδου γλώσσα προγραμματισμού. Ανήκει στις γλώσσες προστακτικού προγραμματισμού (Imperative programming) και υποστηρίζει τόσο το διαδικαστικό (procedural programming) όσο και το αντικειμενοστρεφές (object-oriented programming) προγραμματιστικό υπόδειγμα (programming paradigm). Είναι δυναμική γλώσσα προγραμματισμού (dynamically typed) και υποστηρίζει συλλογή απορριμμάτων (garbage collection ή GC).

Δημιουργήθηκε από τον Ολλανδό Γκίντο βαν Ρόσσουμ (Guido van Rossum) το 1990 και κυκλοφόρησε για πρώτη φορά το 1991.

Ο κύριος στόχος της είναι η αναγνωσιμότητα του κώδικά και η ευκολία χρήσης της. Το συντακτικό της επιτρέπει στους προγραμματιστές να εκφράσουν έννοιες σε λιγότερες γραμμές κώδικα από ό,τι θα ήταν δυνατόν σε γλώσσες όπως η C++ ή η Java. Διακρίνεται λόγω του ότι έχει πολλές βιβλιοθήκες που διευκολύνουν ιδιαίτερα αρκετές συνηθισμένες εργασίες και για την ταχύτητα εκμάθησης της [12].

Οι διερμηνευτές της Python είναι διαθέσιμοι για εγκατάσταση σε πολλά λειτουργικά συστήματα (Linux, Windows, Macintosh, Playstation), επιτρέποντας στην Python την εκτέλεση κώδικα σε ευρεία γκάμα συστημάτων χρησιμοποιώντας εργαλεία τρίτων, όπως το Py2exe ή το Pyinstaller [13]. Ο κώδικας της Python μπορεί να

πακεταριστεί σε αυτόνομα εκτελέσιμα προγράμματα για μερικά από τα πιο δημοφιλή λειτουργικά συστήματα, επιτρέποντας τη διανομή του βασισμένου σε Python λογισμικού για χρήση σε αυτά τα περιβάλλοντα χωρίς να απαιτείται εγκατάσταση του διερμηνευτή της Python [12].

Η Python αναπτύσσεται ως ανοιχτό λογισμικό (open source) και η διαχείρισή της γίνεται από τον μη κερδοσκοπικό οργανισμό Python Software Foundation. Ο κώδικας διανέμεται με την άδεια Python Software Foundation License η οποία είναι συμβατή με την GPL [13]. Το όνομα της γλώσσας προέρχεται από την ομάδα των Άγγλων κωμικών Μόντυ Πάιθον και δεν έχει καμιά σχέση με το φίδι πύθωνα, παρότι το λογότυπό της παραπέμπει σε κάτι τέτοιο [12].

3.4 – OpenWeatherMap

Το OpenWeatherMap είναι μια ηλεκτρονική υπηρεσία που παρέχει δεδομένα καιρού, συμπεριλαμβανομένων των τρεχόντων καιρικών συνθηκών, των προβλέψεων και των ιστορικών δεδομένων στους προγραμματιστές των υπηρεσιών διαδικτύου και των κινητών εφαρμογών. Ως πηγές δεδομένων χρησιμοποιεί υπηρεσίες μετεωρολογικής μετάδοσης, ακατέργαστα δεδομένα από μετεωρολογικούς σταθμούς, ακατέργαστα δεδομένα από σταθμούς ραντάρ και ανεπεξέργαστα δεδομένα από άλλους επίσημους μετεωρολογικούς σταθμούς. Όλα τα δεδομένα υποβάλλονται σε επεξεργασία από το OpenWeatherMap με τέτοιο τρόπο, ώστε να παρέχει όσο το δυνατόν ακριβή δεδομένα για τις καιρικές συνθήκες και τους χάρτες καιρού. Πέρα από αυτό, η υπηρεσία επικεντρώνεται στην κοινωνική πτυχή. Αυτό το επιτυγχάνει με τη σύνδεση ιδιόκτητων μετεωρολογικών σταθμών με την υπηρεσία έτσι, ώστε να επιτευχθεί μεγαλύτερη ακρίβεια των δεδομένων καιρού. Η ιδεολογία εμπνέεται από το OpenStreetMap και τη Wikipedia που κάνουν τις πληροφορίες ελεύθερες και διαθέσιμες για όλους. Χρησιμοποιεί το OpenStreetMap για την εμφάνιση χαρτών καιρού [14].

Για να μπορέσουμε όμως να χρησιμοποιήσουμε τις υπηρεσίες του OpenWeatherMap όσον αφορά τον καιρό τώρα ή την πρόγνωση του καιρού ανά τρεις ώρες για πέντε μέρες, θα πρέπει να κάνουμε λογαριασμό και να δημιουργήσουμε μέσω του λογαριασμού αυτού ένα «APPID» το οποίο θα το χρησιμοποιούμε όταν κάνουμε αιτήματα για να πάρουμε πληροφορίες για τον καιρό. Σαφώς το OpenWeatherMap παρέχει και άλλες δυνατότητες πρόγνωσης του καιρού όπως: ανά μία ώρα για τέσσερις μέρες, καθημερινή πρόγνωση για δεκαέξι μέρες, κτλ, αλλά αυτές οι δυνατότητες παρέχονται στις εκδόσεις επί πληρωμή και όχι στην free έκδοση που χρησιμοποιούμε εμείς [15].

3.5 – Java

Η Java είναι μια αντικειμενοστρεφής γλώσσα γενικού σκοπού που βασίζεται στην δομημένη σε κλάσεις λογική προγραμματισμού. Δημιουργήθηκε το 1995 από την Sun Microsystems ως μια πλατφόρμα για την ανάπτυξη λογισμικού σε μικροσυσκευές. Σκοπός της Java είναι να υπάρχει ανεξαρτησία του λειτουργικού συστήματος και της πλατφόρμας. Τα προγράμματα που είναι γραμμένα σε Java τρέχουνε ακριβώς το ίδιο σε Windows, Linux, Unix και Macintosh χωρίς να χρειαστεί να ξαναγίνει μεταγλώττιση (compiling) ή να αλλάξει ο πηγαίος κώδικας για κάθε διαφορετικό λειτουργικό σύστημα. Για να επιτευχθεί όμως αυτό χρειαζόταν κάποιος τρόπος έτσι, ώστε τα προγράμματα γραμμένα σε Java να μπορούν να είναι «κατανοητά» από κάθε υπολογιστή ανεξάρτητα του είδους επεξεργαστή (Intel x86, IBM, Sun SPARC, Motorola) αλλά και λειτουργικού συστήματος (Windows, Unix, Linux, BSD, MacOS). Η λύση δόθηκε με την ανάπτυξη της Εικονικής Μηχανής (Virtual Machine). Αφού γραφεί κάποιο πρόγραμμα σε Java, στη συνέχεια μεταγλωτίζεται μέσω του μεταγλωττιστή javac, ο οποίος παράγει έναν αριθμό από αρχεία «.class» (κώδικας byte ή bytecode). Ο κώδικας byte είναι η μορφή που παίρνει ο πηγαίος κώδικας της Java όταν μεταγλωττιστεί. Όταν πρόκειται να εκτελεστεί η εφαρμογή σε ένα μηχάνημα, το Java Virtual Machine (που πρέπει να είναι εγκατεστημένο σε αυτό) θα αναλάβει να διαβάσει τα αρχεία «.class». Στη συνέχεια τα μεταφράζει σε γλώσσα μηχανής που να υποστηρίζεται από το λειτουργικό σύστημα και τον επεξεργαστή έτσι, ώστε να εκτελεστεί. Αυτό πραγματοποιείται από την Εικονική Μηχανή (Virtual Machine). Οτιδήποτε θέλει να κάνει ο προγραμματιστής (ή ο χρήστης) γίνεται μέσω της εικονικής μηχανής. Αυτό βοηθάει στο να υπάρχει μεγαλύτερη ασφάλεια στο σύστημα, γιατί η εικονική μηχανή είναι υπεύθυνη για την επικοινωνία χρήστη - υπολογιστή. Ο προγραμματιστής δεν μπορεί να γράψει κώδικα ο οποίος θα έχει καταστροφικά αποτελέσματα για τον υπολογιστή, γιατί η εικονική μηχανή θα τον ανιχνεύσει και δε θα επιτρέψει να εκτελεστεί. Αυτό συνεπάγεται ότι ούτε ο χρήστης μπορεί να κατεβάσει «κακό» κώδικα από το διαδίκτυο και να τον εκτελέσει [16][17].

3.6 – Android Studio

Το Android Studio είναι ένα Ολοκληρωμένο Προγραμματιστικό Περιβάλλον (IDE). Η δουλειά του είναι να παρέχει τη διασύνδεση για τη δημιουργία εφαρμογών, καθώς και τον χειρισμό μεγάλου μέρους της περίπλοκης διαχείρισης αρχείων στο παρασκήνιο.

Η γλώσσα προγραμματισμού που χρησιμοποιούταν κατά κόρον για τη δημιουργία Android εφαρμογών ήταν η Java. Τώρα πλέον έχει μπει στο παιχνίδι και άλλη γλώσσα προγραμματισμού, η Kotlin. Το Android Studio είναι απλό στη χρήση του

και εύκολα μπορεί κανείς να γράψει, να επεξεργαστεί και να αποθηκεύσει τα projects του, καθώς και τα αρχεία που περιλαμβάνουν τα projects αυτά.

Ταυτόχρονα, το Android Studio παρέχει πρόσβαση στο Android SDK. Το Android SDK χρησιμοποιείται για την ομαλή λειτουργία της Java σε Android συσκευές, καθώς και για να επωφεληθεί (η java) από το εγγενές υλικό.

Την ίδια στιγμή, το Android Studio μας επιτρέπει να εκτελούμε τον κώδικα μας, είτε μέσω ενός εξομοιωτή, είτε μέσω ενός υλικού (του κινητού μας για παράδειγμα) συνδεδεμένου στον υπολογιστή μας. Επίσης, μπορούμε να εντοπίζουμε το πρόβλημα κατά την εκτέλεση του προγράμματος και να λαμβάνουμε ανατροφοδότηση που εξηγεί προβλήματα, διενέξεις κτλ. έτσι, ώστε να μπορούμε να επιλύσουμε πιο γρήγορα ότι πρόβλημα μας παρουσιαστεί.

Το Android Studio είναι ένα πολύ ισχυρό και χρήσιμο εργαλείο. Προσφέρει συμβουλές σε πραγματικό χρόνο όταν γράφουμε κώδικα και συχνά θα προτείνει τις απαραίτητες αλλαγές που μπορούν να διορθώσουν τα λάθη ή να κάνουν τον κώδικα μας πιο αποτελεσματικό. Εάν μια μεταβλητή δεν χρησιμοποιείται για παράδειγμα, θα επισημανθεί ως γκρι. Και αν ξεκινήσουμε να πληκτρολογούμε μια γραμμή κώδικα, το Android Studio θα προσκομίσει μια λίστα προτάσεων αυτόματης συμπλήρωσης, για να μας βοηθήσει να την ολοκληρώσουμε. Κάτι που είναι πολύ χρήσιμο σε περίπτωση που δεν θυμόμαστε τη σωστή σύνταξη ή απλά θέλουμε να εξοικονομήσουμε χρόνο [18].

3.7 – Picasso

Το Android Picasso είναι μια ισχυρή βιβλιοθήκη λήψης, προσωρινής αποθήκευσης και επεξεργασίας εικόνων το οποίο αναπτύχθηκε και συντηρείται από την Square Inc. Η βιβλιοθήκη αυτή είναι εξαιρετικά δημοφιλής, καθώς συχνά απαιτείται μόνο μια γραμμή κώδικα, για να επιτελεστεί κάθε μια από τις λειτουργίες της. Ακόμη, ένα άλλο θετικό της βιβλιοθήκης αυτής είναι ότι όλες οι λειτουργίες της γράφονται σε κώδικα με παρόμοιο τρόπο.

Οι λειτουργίες του Android Picasso είναι οι εξής:

- Αλλαγή μεγέθους και κλιμάκωση
- Περικοπή
- Περιστροφή και μετασχηματισμός
- Ρύθμιση εικόνων «Placeholder» και «Error»
- Ξεθώριασμα

- Χρήση της Cache μνήμης του δίσκου
- Προτεραιότητα αιτημάτων
- Υποστήριξη αιτημάτων ακύρωσης και παράλληλη λήψη

Με το Android Picasso μπορούμε να φορτώσουμε μια εικόνα από URL, από τα resources (drawable, mipmap) του android studio, καθώς και από αρχείο εικόνας.

Τέλος, για να μπορέσουμε να χρησιμοποιήσουμε την βιβλιοθήκη Picasso θα πρέπει να την προσθέσουμε στο project μας. Η προσθήκη στο project γίνεται με τον κώδικα που φαίνεται στο παρακάτω απόσπασμα κώδικα.

```
01. dependencies {
02.     implementation 'com.squareup.picasso:picasso:2.5.2'
03. }
```

Απόσπασμα κώδικα 3.7.1

Τον παραπάνω κώδικα τον εισάγουμε στο αρχείο **build.gradle** του **app** [19][20]**.**

3.8 – AsyncTask

Η AsyncTask είναι μια κλάση η οποία χρησιμοποιείται για την δημιουργία εργασιών στο παρασκήνιο, όπως για παράδειγμα απόκτηση δεδομένων από μια τοποθεσία του διαδικτύου (στο 4ο κεφάλαιο θα δούμε λεπτομέρειες). Για να το κάνει αυτό η κλάση αυτή, δημιουργεί ένα άλλο νήμα το οποίο είναι ανεξάρτητο από το κύριο νήμα της εφαρμογής (UI νήμα). Αυτό το κάνει έτσι, ώστε να μην απασχολείται το κύριο νήμα όσο διαρκεί η επικοινωνία με την διαδικτυακή τοποθεσία για την λήψη των απαιτούμενων δεδομένων. Όταν ολοκληρωθεί η λήψη των δεδομένων γίνεται ενημέρωση του κύριου νήματος (UI εφαρμογής) με τα δεδομένα που ήρθαν. Η επικοινωνία του κύριου νήματος με το νήμα που δημιουργήθηκε για τη λήψη δεδομένων γίνεται με την χρήση των μεθόδων onPreExecute, onProgressUpdate και onPostExecute της κλάσης AsyncTask τις οποίες θα τις αναφέρουμε παρακάτω. Επίσης, η AsyncTask περιέχει τρεις γενικούς τύπους:

- **Params**: Ο τύπος (String, int, long, Boolean, κτλ) των παραμέτρων που αποστέλλονται στην εργασία κατά την εκτέλεση.
- **Progress**: Περιέχει πληροφορίες σχετικά με την πρόοδο εκτέλεσης της εργασίας παρασκηνίου. Με την χρήση αυτής της μεταβλητής, καθώς και της μεθόδου onProgressUpdate (θα την δούμε παρακάτω) μπορούμε να

ενημερώσουμε το κύριο νήμα της εφαρμογής για το πόσο υπολείπεται ακόμη μέχρι την ολοκλήρωση της εκτέλεσης της εργασίας.

- **Result**: Περιέχει πληροφορίες σχετικά με τον τύπο (String, int, long, Boolean, κτλ) των αποτελεσμάτων που θα επιστραφούν μετά την ολοκλήρωση της εκτέλεσης της εργασίας.

Όταν εκτελείται μια ασύγχρονη εργασία (AsyncTask), η εργασία περνάει από τέσσερα βήματα:

- **onPreExecute()**: Αυτή η μέθοδος καλείται στο νήμα του UI πριν ξεκινήσει να εκτελείται η εργασία. Είναι υπεύθυνη να κάνει κάποιες αρχικοποιήσεις πριν ξεκινήσει η εκτέλεση της εργασίας, όπως για παράδειγμα να εμφανίσει μια γραμμή προόδου ή ένα ProgressDialog.

- **doInBackground(Params…)**: Αυτό το βήμα καλείται όταν ολοκληρωθεί η εκτέλεση της παραπάνω (onPreExecute) μεθόδου. Σε αυτό το βήμα διαβιβάζονται οι παράμετροι εκτέλεσης της ασύγχρονης εργασίας. Επίσης, σε αυτό το βήμα τοποθετούμε τον κώδικα της εργασίας η οποία θα εκτελεστεί στο παρασκήνιο. Επειδή όμως ενδέχεται η ολοκλήρωση της εκτέλεσης της εργασίας να αργήσει, δίνεται η δυνατότητα με την χρήση της συνάρτησης «publishProgress(Progress…)» να δημοσιεύεται η πρόοδος της εκτέλεσης της εργασίας στο UI της εφαρμογής. Η δημοσίευση αυτή γίνεται, όταν εκτελεστεί η μέθοδος «onProgressUpdate».

- **onProgressUpdate(Progress…)**: Το βήμα αυτό εκτελείται σε περίπτωση που από την μέθοδο «doInBackground» κληθεί η συνάρτηση «publishProgress». Η μέθοδος αυτή είναι υπεύθυνη να ενημερώσει το UI της εφαρμογής για την πρόοδο εκτέλεσης της εργασίας παρασκηνίου.

- **onPostExecute(Result)**: Το βήμα αυτό καλείται όταν ολοκληρωθεί η εκτέλεση της εργασίας παρασκηνίου. Σε αυτό το βήμα επιστρέφεται (γίνεται return) σαν παράμετρος το αποτέλεσμα από την εκτέλεση της μεθόδου doInBackground και μπορούμε να ενημερώσουμε το UI της εφαρμογής, καθώς και να απενεργοποιήσουμε για παράδειγμα το ProgressDialog που δημιουργήσαμε στη μέθοδο «onPreExecute» [19].

3.9 – RSS

Το ακρωνύμιο RSS, από τον αγγλικό όρο Rich Site Summary (Σύνοψη Πλουσίας Σελίδας), ο οποίος συχνά παραφράζεται ως Really Simple Syndication (Πολύ Απλή Διανομή), αναφέρεται σε μία προτυποποιημένη μέθοδο ανταλλαγής

ψηφιακού πληροφοριακού περιεχομένου διαμέσου του Διαδικτύου, στηριγμένη στην πρότυπη, καθιερωμένη και ευρέως υποστηριζόμενη γλώσσα σήμανσης XML. Ένας χρήστης του Διαδικτύου μπορεί έτσι να ενημερώνεται αυτομάτως για γεγονότα και νέα από όσες ιστοσελίδες υποστηρίζουν RSS, αρκεί να έχει εγγραφεί ο ίδιος συνδρομητής στην αντίστοιχη υπηρεσία της εκάστοτε ιστοσελίδας. Οι εν λόγω ενημερώσεις («ροές RSS», αγγλ: «RSS feeds») περιέχουν τα πλήρη δεδομένα, σύνοψη των δεδομένων, σχετικά μεταδεδομένα, ημερομηνία έκδοσης κλπ, ενώ αποστέλλονται αυτομάτως στον συνδρομητή μέσω Διαδικτύου.

Το πρότυπο RSS υπάρχει από το 1999, ωστόσο παλαιότερες, όχι τόσο επιτυχημένες τεχνολογίες παρόμοιου σκοπού (οι οποίες βέβαια δεν αξιοποιούσαν την καθιερωμένη σήμερα γλώσσα XML, καθώς αυτή δεν είχε εμφανιστεί πριν το 1998) κυκλοφορούσαν από τα μέσα της δεκαετίας του 1990. Περί το 2005, το RSS άρχισε να υποστηρίζεται ευρέως από τους δημοφιλέστερους πλοηγούς Web και γρήγορα καθιερώθηκε, ιδιαιτέρως λόγω της εξάπλωσης κατά την ίδια περίοδο των προσωπικών ιστολογίων με τακτική ανανέωση περιεχομένου. Οι ομάδες κατασκευής των εν λόγω πλοηγών είναι που επέλεξαν τότε ένα λογότυπο για την τεχνολογία RSS (το σημερινό ). Πολύ γρήγορα έκανε την εμφάνισή του και το ανταγωνιστικό ως προς το RSS, αλλά επίσης στηριγμένο στην XML, πρότυπο Atom.

Για να μπορέσει ο χρήστης να ενημερώνεται για γεγονότα και νέα από τις ιστοσελίδες που τον ενδιαφέρουν, θα πρέπει πρώτων οι ιστοσελίδες να υποστηρίζουν την τεχνολογία RSS και δεύτερων να έχουν κατάλληλο λογισμικό το οποίο να διαβάζει τις ροές RSS. Μια RSS ροή ακολουθεί το παρακάτω πρότυπο [21].

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0">
   <channel>
      <title>RSS Τίτλος</title>
      <description>Αυτό είναι ένα παράδειγμα RSS κειμένου</description>
      <link>http://www.example.com/main.html</link>
      <lastBuildDate>Thu, 12 Mar 2020 23:00:00 +0000</lastBuildDate>
      <pubDate>Thu, 12 Mar 2020 23:30:00 +0000</pubDate>
      <ttl>1800</ttl>
      <item>
         <title>Παράδειγμα καταχώρισης</title>
         <description>Ακολουθεί ένα κείμενο που περιέχει μια ενδιαφέρουσα περιγραφή.</description>
         <link>http://www.example.com/blog/post/1</link>
         <guid isPermaLink="false">7bd204c6-1655-4c27-aeee-53f933c5395f</guid>
         <pubDate>Thu, 12 Mar 2020 23:30:00 +0000</pubDate>
      </item>
   </channel>
</rss>
```

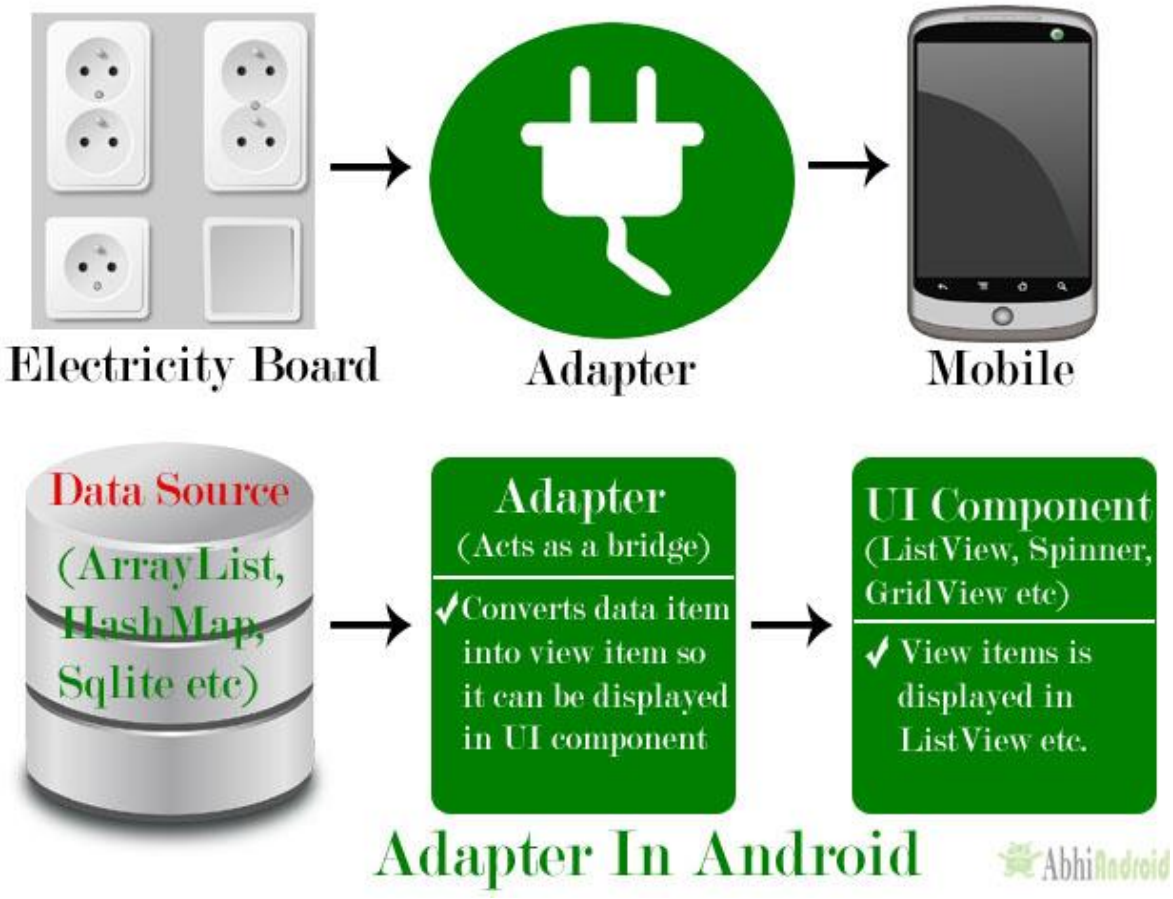
Εικόνα 13 – Πρότυπο RSS feed

Για να μπορέσει κάποιο λογισμικό να διαβάσει τις RSS ροές θα πρέπει να γραφτεί κώδικας ο οποίος να κάνει parse xml περιεχόμενο με βάση το παραπάνω πρότυπο. Έτσι κι εμείς, για να πάρουμε τις ειδήσεις από τα γεωργικά sites που

επιλέξαμε, με βάση το παραπάνω πρότυπο γράψαμε κώδικα στην android εφαρμογή ο οποίος παίρνει τα RSS feeds και μετά από κατάλληλη επεξεργασία τα τοποθετεί στην εφαρμογή, για να τα δει ο χρήστης. Περισσότερες λεπτομέρειες γι' αυτό θα δούμε στο κεφάλαιο 4 ενότητα 4.3.4.

3.10 – Adapter

Ο Adapter είναι η γέφυρα μεταξύ του UI της εφαρμογής και της πηγής δεδομένων, η οποία μας βοηθάει να γεμίσουμε με περιεχόμενο το UI. Τα δεδομένα μπορεί να προέρχονται είτε από μια βάση δεδομένων είτε από αρχείο είτε από το διαδίκτυο, κ.α. Επίσης ο Adapter, διατηρεί τα δεδομένα και τα αποστέλλει σε ένα adapter view. Στη συνέχεια, το view μπορεί να λάβει τα δεδομένα από το adapter view και να εμφανίσει τα δεδομένα σε διαφορετικά views όπως RecyclerView, ListView, GridView, Spinner, κτλ. Ακόμη, έχουμε τη δυνατότητα να δημιουργήσουμε περισσότερο προσαρμοσμένα views χρησιμοποιώντας το base adapter ή το custom adapters [22].



Εικόνα 14 – Εννοιολογικό διάγραμμα Adapter

3.11 – ListView

Το ListView είναι ένα ViewGroup το οποίο περιέχει μια λίστα με δυνατότητα κύλισης. Στη λίστα αυτή μπορούν να εισαχθούν στοιχεία (views) το ένα κάτω από το άλλο. Ο χρήστης, μπορεί να επιλέξει οποιοδήποτε στοιχείο απ' αυτήν την λίστα κάνοντας κλικ πάνω του. Το ListView ακόμα, είναι κατασκευασμένο με δυνατότητα κύλισης, οπότε δεν χρειάζεται να χρησιμοποιήσουμε το ScrollView ή οτιδήποτε άλλο με αυτό. Για να μπορέσουμε να χρησιμοποιήσουμε το ListView, θα πρέπει να τοποθετούμε στο UI ένα αντικείμενο ListView.

Το ListView χρησιμοποιείται ευρέως σε εφαρμογές Android. Ένα πολύ συνηθισμένο παράδειγμα του ListView είναι η λίστα των επαφών μας.

Για να τοποθετήσουμε δεδομένα σε ένα ListView χρησιμοποιούμε Adapter ο οποίος δημιουργείται επεκτείνοντας τον ArrayAdapter (απόσπασμα κώδικα 3.11.1).

```
01.  public class NoteAdapter extends ArrayAdapter<Note>{
```

Απόσπασμα κώδικα 3.11.1

Τα στοιχεία της λίστα εισάγονται στο ListView χρησιμοποιώντας έναν Adapter, ο οποίος τραβάει τα στοιχεία από μια πηγή δεδομένων, όπως ένα arraylist, array, βάση δεδομένων, κ.α. Στη συνέχεια, ο Adapter αυτός, μετατρέπει το κάθε στοιχείο της λίστας σε view και έπειτα το τοποθετεί στο ListView.

Το ListView είναι διαθέσιμο στη παλέτα εργαλείων του Android Studio. Από εκεί μπορούμε να το κάνουμε drag and drop στην οθόνη του εικονικού κινητού, για να το δημιουργήσουμε. Εναλλακτικά, μπορούμε να το δημιουργήσουμε γράφοντας κώδικα xml. Στο παρακάτω απόσπασμα κώδικα, βλέπουμε την κύρια μέθοδο του ArrayAdapter.

```
01.  public View getView(int position, View convertView, ViewGroup parent){
02.      ...
03.  }
```

Απόσπασμα κώδικα 3.11.2

Η μέθοδος «getView» χρησιμοποιείται για να επιστρέψει (έπειτα από κάποια επεξεργασία) ένα στοιχείο στο ListView [19].

3.12 – GridView

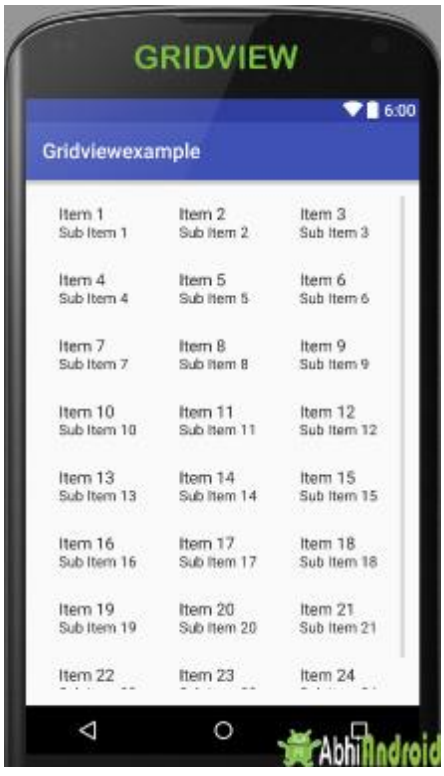
Στο android, το GridView είναι ένα ViewGroup το οποίο εμφανίζει τα items που εισάγονται σε αυτό σε ένα πλέγμα (grid) κύλισης δύο διαστάσεων (γραμμές και

στήλες).Τα στοιχεία (items) πλέγματος δεν είναι κατ' ανάγκη προκαθορισμένα, αλλά εισάγονται αυτόματα στο layout χρησιμοποιώντας ένα ListAdapter. Ο χρήστης μπορεί στη συνέχεια να επιλέξει οποιοδήποτε item από το πλέγμα κάνοντας κλικ πάνω σ' αυτό. Το GridView ακόμα, είναι κατασκευασμένο με δυνατότητα κύλισης, οπότε δεν χρειάζεται να χρησιμοποιήσουμε το ScrollView ή οτιδήποτε άλλο με αυτό. Για να μπορέσουμε να χρησιμοποιήσουμε το GridView, θα πρέπει να τοποθετούμε στο UI ένα αντικείμενο GridView.

Επίσης, το GridView χρησιμοποιείται ευρέως σε εφαρμογές Android. Ένα παράδειγμα χρήσης του είναι η συλλογή εικόνων του χρήστη, όπου εμφανίζονται οι εικόνες χρησιμοποιώντας το πλέγμα.

Ο adapter χρησιμοποιείται για το γέμισμα δεδομένων στο GridView. Τα items του πλέγματος εισάγονται αυτόματα σε ένα GridView χρησιμοποιώντας τον adapter αυτόν, ο οποίος τραβάει τα δεδομένα από μια πηγή, όπως ένα arraylist, array, βάση δεδομένων, κ.α.

Το GridView είναι διαθέσιμο στη παλέτα εργαλείων του Android Studio. Από εκεί μπορούμε να το κάνουμε drag and drop στην οθόνη του εικονικού κινητού για να το δημιουργήσουμε. Εναλλακτικά, μπορούμε να το δημιουργήσουμε γράφοντας κώδικα xml [19].



Εικόνα 15 – Διάταξη GridView

3.13 – RecyclerView

Η κλάση RecyclerView κάνει extends την κλάση ViewGroup και implements το ScrollingView interface. Το RecyclerView είναι μια βελτιωμένη έκδοση του ListView, καθώς και του GridView το οποίο χρησιμοποιείται για τη δημιουργία λίστας κύλισης στοιχείων βασισμένη σε μεγάλο όγκο δεδομένων. Στο μοντέλο RecyclerView, πολλά διαφορετικά στοιχεία συνεργάζονται για την εμφάνιση των δεδομένων. Για να μπορέσουμε να χρησιμοποιήσουμε το RecyclerView, θα πρέπει να τοποθετούμε στο UI ένα αντικείμενο RecyclerView. Το RecyclerView γεμίζει με views τα οποία παρέχονται από έναν layout manager. Έχουμε τη δυνατότητα να χρησιμοποιήσουμε τους βασικούς layout managers (όπως LinearLayoutManager ή GridLayoutManager) ή να δημιουργήσουμε δικό μας.

Τα views στη λίστα αντιπροσωπεύονται από αντικείμενα view holder. Αυτά τα αντικείμενα είναι instance μιας κλάσης η οποία επεκτείνει το RecyclerView.ViewHolder (απόσπασμα κώδικα 3.13.1).

```
01. public class MyViewHolder extends RecyclerView.ViewHolder{
```

Απόσπασμα κώδικα 3.13.1

Κάθε view holder είναι υπεύθυνο για την προβολή ενός μόνο view. Για παράδειγμα, εάν η λίστα μας εμφανίζει άρθρα από ένα ειδησεογραφικό site, κάθε view holder μπορεί να αντιπροσωπεύει ένα μόνο άρθρο. Το RecyclerView δημιουργεί μόνο όσα view holders χρειάζονται για την εμφάνιση κατά μήκος της οθόνης, καθώς και μερικά επιπλέον. Όταν ο χρήστης μετακινείται προς τα κάτω της λίστας, το RecyclerView παίρνει τα views τα οποία είναι εκτός της οθόνης για να συνδέσει σε αυτά τα καινούρια δεδομένα.

Τα αντικείμενα view holder διαχειρίζονται από έναν adapter, ο οποίος δημιουργείται επεκτείνοντας των RecyclerView.Adapter (απόσπασμα κώδικα 3.13.2).

```
01. public class RSSFeedAdapter extends RecyclerView.Adapter<RSSFeedAdapter.MyViewHolder>{
```

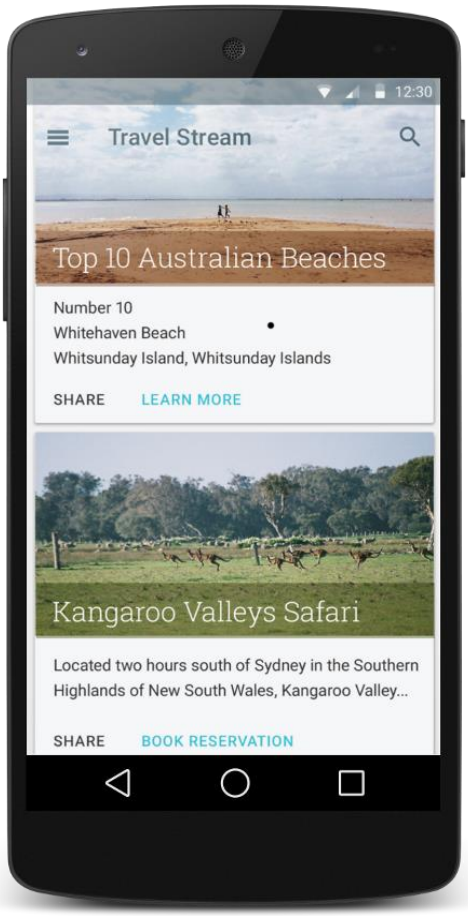
Απόσπασμα κώδικα 3.13.2

Ο adapter δημιουργεί view holders ανάλογα με τις ανάγκες και τα συνδέει με τα δεδομένα. Αυτό το κάνει εκχωρώντας το view holder σε μια θέση (position) και καλώντας τη μέθοδο «onBindViewHolder()» του adapter. Αυτή η μέθοδος χρησιμοποιεί τη θέση του view holder, για να καθορίσει ποιο είναι το περιεχόμενο του, ανάλογα με τη θέση του στη λίστα. Στο παρακάτω απόσπασμα κώδικα φαίνονται οι κύριες μέθοδοι του RecyclerView.Adapter.

```
@NonNull
@Override
public MyViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    ...
}
@Override
public void onBindViewHolder(@NonNull MyViewHolder holder, int position) {
    ...
}
@Override
public int getItemCount() {
    ...
}
```

Απόσπασμα κώδικα 3.13.3

Η μέθοδος «onCreateViewHolder» χρησιμοποιείται, για να δημιουργήσει ένα view holder. Στη συνέχεια, η μέθοδος «onBindViewHolder» είναι υπεύθυνη να τοποθετήσει δεδομένα στο view holder που δημιουργήθηκε ενώ η μέθοδος «getItemCount» επιστρέφει τον συνολικό αριθμό των στοιχείων τα οποία περιέχει ο adaptrer. Τέλος, η εμφάνιση των στοιχείων στο UI της εφαρμογής γίνεται με το CardView. Το CardView είναι ένα FrameLayout με στρογγυλωμένο background και σκιά (εικόνα 16).



Εικόνα 16 – Παράδειγμα CardView

Επίσης, εκτός από την εμφάνιση στοιχείων μέσα σε ένα RecyclerView, μπορεί να χρησιμοποιηθεί και για να εμφανίσει στοιχείων μέσα σε ένα ListView.

Τέλος, για να μπορέσουμε να χρησιμοποιήσουμε το RecyclerView και το CardView επειδή δεν προσφέρονται από προεπιλογή στο android studio θα πρέπει να τα προσθέσουμε στο project μας. Η προσθήκη στο project γίνεται με τον κώδικα που φαίνεται παρακάτω.

```
01. dependencies {
02.     implementation 'com.android.support:recyclerview-v7:26.1.0'
03.     implementation 'com.android.support:cardview-v7:28.0.0'
04. }
```

Απόσπασμα κώδικα 3.13.4

Τον παραπάνω κώδικα τον εισάγουμε στο αρχείο **build.gradle** του **app** [22]**.**

3.14 – AlarmManager

Το AlarmManager μας επιτρέπει να έχουμε πρόσβαση στο σύστημα συναγερμού (alarm) του Android. Το AlarmManager χρησιμοποιείται, για να ενεργοποιήσει ένα κομμάτι κώδικα σε μια συγκεκριμένη χρονική στιγμή. Αυτό το κάνει χρησιμοποιώντας την υπηρεσία συναγερμού του Android SDK το οποίο μπορεί να εκτελεστεί ανεξάρτητα από τον κύκλο ζωής της εφαρμογής (δηλαδή, ακόμα και αν η εφαρμογή έχει τερματιστεί). Για παράδειγμα, θα μπορούσαμε να χρησιμοποιήσουμε ένα alarm, για να ξεκινήσει μια λειτουργία η οποία θα διαρκέσει μεγάλο χρονικό διάστημα (σε σχέση με τις κανονικές λειτουργίες), όπως η εκκίνηση μιας υπηρεσίας μια φορά την ημέρα για να κατεβάσουμε μια πρόγνωση καιρού. Η εκκίνηση της υπηρεσίας μπορεί να γίνει είτε το κινητό λειτουργεί είτε δεν λειτουργεί («κοιμάται»). Κατά την εκτέλεση της υπηρεσίας, το σύστημα AlarmManager του Android διατηρεί κλειδαριά προστασίας από τη λειτουργία CPU, η οποία παρέχει εγγύηση να μην «κοιμηθεί» (η CPU), μέχρι να ολοκληρωθεί η εκτέλεση της υπηρεσίας αυτής.

Οι συναγερμοί χωρίζονται σε δύο κατηγορίες, στους μη επαναλαμβανόμενους και στους επαναλαμβανόμενους. Απ’ αυτές τις δύο κατηγορίες, οι επαναλαμβανόμενοι συναγερμοί χωρίζονται σε δύο τύπους. Ο ένας τύπος είναι **Elapsed Real Time** και ο άλλος **Real Time Clock (RTC)**. Ο πρώτος χρησιμοποιεί ως σημείο αναφοράς τον χρόνο από την εκκίνηση της συσκευής ενώ ο δεύτερος χρησιμοποιεί τον χρόνο UTC (ρολόι τοίχου). Αυτό σημαίνει ότι ο Elapsed Real Time είναι κατάλληλος για τη δημιουργία ενός συναγερμού με βάση το πέρασμα του χρόνου (για παράδειγμα, ένας συναγερμός που πυροδοτείται κάθε μισή ώρα), καθώς δεν επηρεάζεται από τη ζώνη ώρας/τοπική ρύθμιση. Αντίθετα, ο Real Time Clock είναι

πιο κατάλληλος για συναγερμούς οι οποίοι εξαρτώνται από την τρέχουσα τοπική ώρα (για παράδειγμα, ένας συναγερμός που πυροδοτείται στις 16:00).

Και οι δύο τύποι επαναλαμβανομένων συναγερμών έχουν μια έκδοση «wakeup», η οποία ξυπνάει την CPU της συσκευής, αν η οθόνη είναι απενεργοποιημένη. Αυτό εξασφαλίζει ότι ο συναγερμός θα πυροδοτηθεί την προγραμματισμένη ώρα. Πράγμα χρήσιμο, αν η εφαρμογή έχει εξάρτηση από το χρόνο. Σε περίπτωση που η εφαρμογή δεν εξαρτάται από το χρόνο, μπορούμε να χρησιμοποιήσουμε την έκδοση επαναλαμβανόμενου συναγερμού χωρίς το «wakeup». Έτσι, ο συναγερμός θα πυροδοτηθεί, όταν η συσκευή «ξυπνήσει».

Η επιλογή του τύπου συναγερμού είναι συχνά το πρώτο βήμα για τη δημιουργία συναγερμού. Μια περαιτέρω διάκριση είναι το πόσο ακριβής θα πρέπει να είναι ο συναγερμός:

- **setInexactRepeating**: Το setInexactRepeating προγραμματίζει έναν επαναλαμβανόμενο συναγερμό ο οποίος είναι ανακριβής στον χρόνο ενεργοποίησής του. Δηλαδή, εάν προγραμματίσουμε την ενεργοποίηση του συναγερμού στις 17:00, ίσως να μην ενεργοποιηθεί ακριβώς την ίδια στιγμή. Αυτοί οι συναγερμοί δεν καταναλώνουν πολλή ενέργεια, γιατί ρυθμίζουν τους χρόνους παράδοσης, για να πυροδοτήσουν ταυτόχρονα πολλούς συναγερμούς. Λόγω αυτής τους της ιδιότητας συνιστώνται σε περίπτωση που η εφαρμογή δεν έχει απαιτήσεις να ενεργοποιηθεί ο συναγερμός την ακριβή χρονική στιγμή που ορίστηκε. Μετά την ενεργοποίηση του, αυτός ο συναγερμός θα ξανά ενεργοποιηθεί έπειτα από την προγραμματισμένη ώρα (για παράδειγμα, ανά μία ώρα).

- **setRepeating**: Το setRepeating είναι ίδιο με το setInexactRepeating εκτός από το γεγονός ότι ο συναγερμός ενεργοποιείται ακριβώς την προγραμματισμένη ώρα. Το Android μας προτείνει να το χρησιμοποιήσουμε μόνο όταν αυτό είναι απαραίτητο, επειδή θέτει περιττές επιβαρύνσεις στο σύστημα, καθώς δεν θα μπορέσει να προσαρμόσει το χρόνο παράδοσης για τη δέσμη πολλών συναγερμών μαζί. Επίσης, μ' αυτόν τον συναγερμό μειώνεται και ο χρόνος ζωής της μπαταρίας. Όπως το setInexactRepeating, αυτός ο συναγερμός θα επαναληφθεί μετά από την προγραμματισμένη ώρα.

- **set**: Το set προγραμματίζει έναν συναγερμό ο οποίος θα εκτελεστεί μία φορά και περίπου στην προγραμματισμένη ώρα. Το λειτουργικό σύστημα, μπορεί να προσαρμόσει το χρόνο παράδοσης για αυτούς τους συναγερμούς.

- **setExact**: Το setExact είναι ίδιο με το set, εκτός από το γεγονός ότι στο setExact η εκτέλεση του συναγερμού θα γίνει ακριβώς την προγραμματισμένη ώρα. Επίσης, δεν επιτρέπεται στο λειτουργικό σύστημα να ρυθμίσει το χρόνο παράδοσης γι' αυτούς τους τύπους συναγερμών. Αυτός ο τύπος συναγερμού δεν πρέπει να χρησιμοποιείται, εκτός εάν είναι απαραίτητη η ενεργοποίηση ενός συναγερμού σε συγκεκριμένο χρόνο [22][23].

3.15 – JobService

Το JobService χρησιμοποιείται για δημιουργία και εκτέλεση εργασιών στο παρασκήνιο (background) ανεξάρτητα από το αν η εφαρμογή η οποία το δημιούργησε εκτελείται αυτή τη στιγμή ή όχι. Πιο συγκεκριμένα, το JobService είναι η βασική κλάση που χειρίζεται ασύγχρονα αιτήματα που είχαν προγραμματιστεί προηγουμένως. Για να χρησιμοποιήσουμε τις δυνατότητες της κλάσης JobService θα πρέπει να την κάνουμε extend. Αυτό συνεπάγεται με το γεγονός ότι πρέπει να κάνουμε override τις βασικές μεθόδους της κλάσης αυτής:

- **onStartJob (JobParameters)**: Η μέθοδος «onStartJob» καλείται από το σύστημα, όταν είναι η ώρα για την εκτέλεση της εργασίας. Αν η εργασία την οποία θέλουμε να εκτελέσουμε είναι σύντομή και απλή μπορούμε απευθείας να γράψουμε τον κώδικα της μέσα στη μέθοδο onStartJob και να επιστρέψουμε (return) «false» έτσι, ώστε να ενημερώσουμε το σύστημα ότι όλες οι εργασίες έχουν ολοκληρωθεί. Σε περίπτωση όμως που χρειάζεται να κάνουμε πιο περίπλοκη εργασία, όπως σύνδεση με το διαδίκτυο για να λάβουμε κάποια δεδομένα, θα πρέπει να ξεκινήσουμε ένα thread στο παρασκήνιο (background) και να επιστρέψουμε (return) «true» έτσι, ώστε το σύστημα να γνωρίζει ότι έχει ένα thread να τρέχει ακόμα και θα πρέπει να διατηρήσει για μεγαλύτερο χρονικό διάστημα το wakelock (κλειδαριά προστασίας για να μην «κοιμηθεί» η CPU).

- **onStopJob (JobParameters)**: Η μέθοδος «onStopJob» καλείται από το σύστημα αν η εργασία ακυρωθεί πριν τελειώσει. Αυτό συμβαίνει γενικά όταν οι συνθήκες εργασίας δεν πληρούνται πλέον, όπως όταν η συσκευή δεν φορτίζεται αυτή τη στιγμή ή αν το Wi-Fi δεν είναι διαθέσιμο. Επομένως, χρησιμοποιούμε τη μέθοδο αυτή για οποιονδήποτε έλεγχο ασφάλειας και καθαρισμό που μπορεί να χρειαστεί να κάνουμε, για να αντιμετωπίσουμε μια μισοτελειωμένη εργασία. Στη συνέχεια επιστρέφουμε «true» αν θέλουμε το σύστημα να επαναπρογραμματίσει την εργασία ή «false» αν δεν είναι απαραίτητο να γίνει αυτό.

Η υπηρεσία JobService εκτελεί κάθε εισερχόμενη εργασία σε handler που τρέχει στο κύριο thread της εφαρμογής. Αυτό σημαίνει ότι αν η εργασία που θέλουμε να γίνει θα χρειαστεί κάποιο χρονικό διάστημα για να ολοκληρωθεί θα πρέπει να δημιουργήσουμε είτε καινούριο thread είτε handler είτε AsyncTask. Αν δεν το κάνουμε αυτό, θα αποκλείσουμε τυχόν μελλοντικές επανακλήσεις από το JobManager, συγκεκριμένα onStopJob(android.app.job.JobParameters), το οποίο μας ενημερώνει ότι οι απαιτήσεις προγραμματισμού δεν πληρούνται πλέον [22].

3.16 – Service

Μια υπηρεσία (service) είναι ένα component της εφαρμογής το οποίο μπορεί να εκτελεί εκτεταμένες εργασίες στο παρασκήνιο και δεν παρέχει διεπαφή χρήστη (UI). Επίσης, ένα component της εφαρμογής μπορεί να ξεκινήσει μια υπηρεσία η οποία θα συνεχίζεται να εκτελείται στο παρασκήνιο ακόμα και αν ο χρήστης μεταβεί σε άλλη εφαρμογή. Ακόμα, ένα component μπορεί να κάνει bind μια υπηρεσία, για να αλληλεπιδράσει μαζί της και να πραγματοποιήσει ακόμη και επικοινωνία μεταξύ διεργασιών (InterProcess Communication, IPC). Για παράδειγμα, μια υπηρεσία μπορεί να χειριστεί συναλλαγές δικτύου, να παίξει μουσική, να εκτελέσει I / O αρχείων ή να αλληλεπιδράσει με έναν παροχέα περιεχομένου, όλα από το παρασκήνιο.

Παρακάτω θα μιλήσουμε για τους τρεις διαφορετικούς τύπους υπηρεσιών:

- **Foreground**: Μια υπηρεσία Foreground εκτελεί κάποια λειτουργία που είναι αξιοσημείωτη για τον χρήστη. Για παράδειγμα, μια εφαρμογή ήχου θα χρησιμοποιεί μια υπηρεσία Foreground για την αναπαραγωγή ενός ηχητικού κομματιού. Οι Foreground υπηρεσίες πρέπει να εμφανίζουν μια ειδοποίηση. Επίσης, συνεχίζουν να εκτελούνται ακόμα και όταν ο χρήστης δεν αλληλεπιδρά με την εφαρμογή.

- **Background**: Μια υπηρεσία Background εκτελεί μια ενέργεια που δεν παρατηρείται άμεσα από το χρήστη. Για παράδειγμα, μια εφαρμογή χρησιμοποιεί μια υπηρεσία, για να συμπιέσει τον αποθηκευτικό της χώρο.

- **Bound**: Μια υπηρεσία είναι δεσμευμένη (bound) όταν ένα component εφαρμογής δεσμεύεται (binds) σε αυτήν καλώντας την μέθοδο «bindService()». Μια δεσμευμένη υπηρεσία προσφέρει μια διεπαφή πελάτη - διακομιστή που επιτρέπει στα components να αλληλεπιδρούν με την υπηρεσία, να στέλνουν αιτήματα, να λαμβάνουν αποτελέσματα, ακόμα και να υλοποιούν επικοινωνία μεταξύ διεργασιών (InterProcess Communication, IPC). Μια δεσμευμένη υπηρεσία εκτελείται μόνο για όσο χρονικό διάστημα την έχει δεσμεύσει κάποιο component από άλλη

εφαρμογή. Πολλαπλά components μπορούν να συνδεθούν (bind) με μια υπηρεσία ταυτόχρονα, αλλά όταν όλα αυτά αποσυνδεθούν, η υπηρεσία θα καταστραφεί.

Για να δημιουργήσουμε μια υπηρεσία, θα πρέπει σε μία κλάση που θα δημιουργήσουμε να κάνουμε extend την κλάση «Service». Αφ' ότου γίνει αυτό, θα πρέπει να υπερφορτώσουμε τις βασικές μεθόδους της κλάσης αυτής:

- **onCreate()**: Το σύστημα κάνει χρήση αυτής της μεθόδου, για να εκτελέσει διαδικασίες αρχικής ρύθμισης και δημιουργίας της υπηρεσίας (πριν καλέσει αυτή τη μέθοδο το σύστημα καλεί είτε τη «onStartCommand()» είτε τη «onBind()»). Στην περίπτωση που η υπηρεσία εκτελείται ήδη, αυτή η μέθοδος δεν καλείται.

- **onBind()**: Το σύστημα επικαλείται αυτή τη μέθοδο καλώντας την bindService() όταν ένα άλλο component θέλει να δεσμεύσει την υπηρεσία (όπως για παράδειγμα, να εκτελέσει RPC (Remote Procedure Call)). Κατά την εφαρμογή αυτής της μεθόδου, θα πρέπει να δώσουμε μια διεπαφή την οποία οι πελάτες θα χρησιμοποιούν, για να επικοινωνήσουν με την υπηρεσία. Αυτό γίνεται επιστρέφοντας έναν IBinder. Ωστόσο, αν δεν θέλουμε να επιτρέψουμε να γίνει δέσμευση της υπηρεσίας, θα πρέπει να κάνουμε «return null» [24].

3.17 – CustomView

Το android προσφέρει ένα εξελιγμένο και ισχυρό componentized μοντέλο για τη δημιουργία του δικού μας UI. Αυτό το μοντέλο βασίζεται στις βασικές κλάσεις σχεδίασης: **View** και **ViewGroup**. Γενικά, η πλατφόρμα android περιλαμβάνει μια ποικιλία προκατασκευασμένων υποκλάσεων View και ViewGroup οι οποίες ονομάζονται widgets και layouts αντίστοιχα, τις οποίες μπορούμε να χρησιμοποιήσουμε, για να δημιουργήσουμε το δικό μας UI.

Μερικά από τα διαθέσιμα widgets που προσφέρει το android είναι: Button, TextView, EditText, ListView, CheckBox, RadioButton, Gallery, Spinner και τα πιο ειδικής χρήσης: AutoCompleteTextView, ImageSwitcher και TextSwitcher.

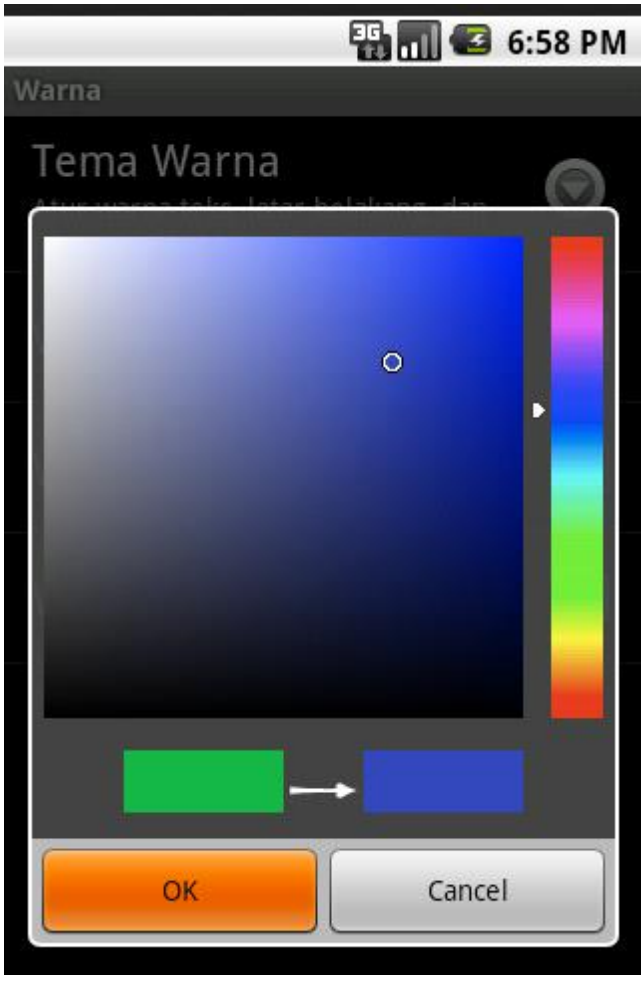
Μερικά από τα διαθέσιμα layouts που προσφέρει το android είναι: LinearLayout, FrameLayout, RelativeLayout, WebView και το GridView.

Αν κανένα από τα προεγκατεστημένα widgets ή layouts δεν ανταποκρίνεται στις ανάγκες μας, μπορούμε να δημιουργήσουμε τη δική μας υποκλάση View χρησιμοποιώντας τα προεγκατεστημένα widgets και layouts. Στην περίπτωση που μας αρκεί μόνο να κάνουμε μικρές προσαρμογές σε ένα υπάρχον widget ή layout,

μπορούμε απλά να το κάνουμε extend σε μία κλάση και να κάνουμε override τις μεθόδους του [25].

3.18 – Παλέτα χρωμάτων

Η παλέτα χρωμάτων που χρησιμοποιήσαμε στην εφαρμογή μας η οποία δίνει τη δυνατότητα στον χρήστη να επιλέξει αυτός τι χρώμα θα έχει το κείμενο στη σημείωση που θα δημιουργήσει (ενότητα 4.3.7) δεν ανήκει στα widgets που προσφέρει το android studio. Αυτή η παλέτα χρωμάτων δημιουργήθηκε από μια ομάδα προγραμματιστών η οποία χρησιμοποίησε τη βιβλιοθήκη «**AmbilWarna**». Παρ' όλου που δίνουν την παλέτα χρωμάτων να την χρησιμοποιήσουν και άλλοι προγραμματιστές, στην πραγματικότητα δεν είναι ανοιχτού κώδικα και έτσι δεν μας δίνουν πολλές πληροφορίες γι' αυτήν. Οι μόνες πληροφορίες που μας έχουν δώσει είναι μόνο όσον αφορά στο πώς θα την χρησιμοποιήσουμε στην εφαρμογή μας. Η παλέτα χρωμάτων φαίνεται στην εικόνα 17.



Εικόνα 17 – Παλέτα χρωμάτων

Παρακάτω βλέπουμε τον κώδικα ο οποίος χρησιμοποιείται για να εμφανιστεί η παλέτα χρωμάτων στο UI της εφαρμογής μας.

```
AmbilWarnaDialog dialog = new AmbilWarnaDialog(this, initialColor, new OnAmbilWarnaListener() {
    @Override
    public void onOk(AmbilWarnaDialog dialog, int color) {
        ...
    }
    @Override
    public void onCancel(AmbilWarnaDialog dialog) {
        ...
    }
dialog.show();
```

Απόσπασμα κώδικα 3.18.1

Ανάλογα με το αν πατήσει «OK» ή «Cancel» θα εκτελεστεί η αντίστοιχη μέθοδο του AmbilWarnaDialog (παραπάνω απόσπασμα κώδικα).

Τέλος, για να μπορέσουμε να χρησιμοποιήσουμε την παλέτα χρωμάτων θα πρέπει να την προσθέσουμε στο project μας. Η προσθήκη στο project γίνεται με τον κώδικα που φαίνεται στο παρακάτω απόσπασμα κώδικα.

```
dependencies {
    implementation 'com.github.yukuku:ambilwarna:2.0.1'
}
```

Απόσπασμα κώδικα 3.18.2

Τον παραπάνω κώδικα τον εισάγουμε στο αρχείο **build.gradle** του **app** [26].

Επίλογος

Στο κεφάλαιο που αναπτύξαμε παραπάνω, έγινε αναφορά στις τεχνολογίες οι οποίες συνέβαλαν στο να ολοκληρωθεί το πρακτικό κομμάτι της πτυχιακής εργασίας. Η παρουσίαση των τεχνολογιών έγινε όσο το δυνατόν πιο απλά και κατανοητά, μένοντας όσο το δυνατόν γινόταν στο θεωρητικό μέρος της κάθε τεχνολογίας. Στο κεφάλαιο που ακολουθεί, θα μιλήσουμε για πιο τεχνικά χαρακτηριστικά των τεχνολογιών που αναφέραμε παραπάνω.

ΚΕΦΑΛΑΙΟ 4 – ΥΛΟΠΟΙΗΣΗ

4.1 – Εισαγωγή

Σε αυτό το κεφάλαιο θα περιγράψουμε την υλοποίηση του πρακτικού κομματιού της παρούσας πτυχιακής εργασίας. Η υλοποίηση αυτή αποτελείται από δύο κομμάτια. Το **server side** και μια **android εφαρμογή**. Για το **server side** κομμάτι θα περιγράψουμε τα δύο scripts τα οποία τρέχουν στον server που χρησιμοποιούμε, τι πληροφορίες επεξεργάζονται και από πού τις αντλούν, καθώς και τι πληροφορία δημιουργείται η οποία θα χρησιμοποιηθεί από την **android εφαρμογή**. Για την **android εφαρμογή** θα περιγράψουμε το κύριο κομμάτι της, το οποίο παρουσιάζει στην συσκευή του χρήστη πληροφορίες για το αν πότισε ή αν θα ποτίσει, καθώς και την δυνατότητα του χρήστη να αλλάξει την απόφαση που λήφθηκε μετά την εκτέλεση των scripts στον server. Ακόμα, θα περιγράψουμε και τα άλλα κομμάτια της android εφαρμογής τα οποία είναι: **Ο Καιρός τώρα**, **Πρόγνωση Καιρού**, **Αγροτικές Ειδήσεις**, **Ειδοποιήσεις Εργασιών**, **Ατζέντα** και **Σημειωματάριο**.

4.2 – Server side

Το κυριότερο κομμάτι αυτής την πτυχιακής διαδραματίζεται στον **server**. Ο server είναι υπεύθυνος να επικοινωνήσει με το **openweathermap** (site που παρέχει δεδομένα καιρού) και να παραλάβει τα δεδομένα που θα του στείλει. Έπειτα ελέγχει την υγρασία του χώματος (στην παρούσα πτυχιακή δεν ασχοληθήκαμε με το hardware και έτσι την υγρασία του χώματος την παίρνουμε με χρήση ενός φανταστικού αισθητήρα). Τέλος, ελέγχει την απόφαση που πήρε ο χρήστης (αν πήρε) η οποία έρχεται μέσω **Http Post** από την android εφαρμογή. Και στη συνέχεια, ανάλογα με τα καιρικά δεδομένα, την υγρασία του χώματος και την επιλογή του χρήστη, παίρνει την απόφαση για το αν θα πραγματοποιηθεί πότισμα ή όχι. Για να κάνει όλη την παραπάνω διαδικασία που περιγράψαμε, χρησιμοποιεί δύο **python scripts**. Το «**potisma.py**» και το «**forecastBeforeTenMinutes.py**». Αυτά τα δύο scripts τρέχουν κάθε τρεις ώρες με διαφορά δέκα λεπτών (το «**forecastBeforeTenMinutes.py**» τρέχει δέκα λεπτά πριν τρέξει το «**potisma.py**»).

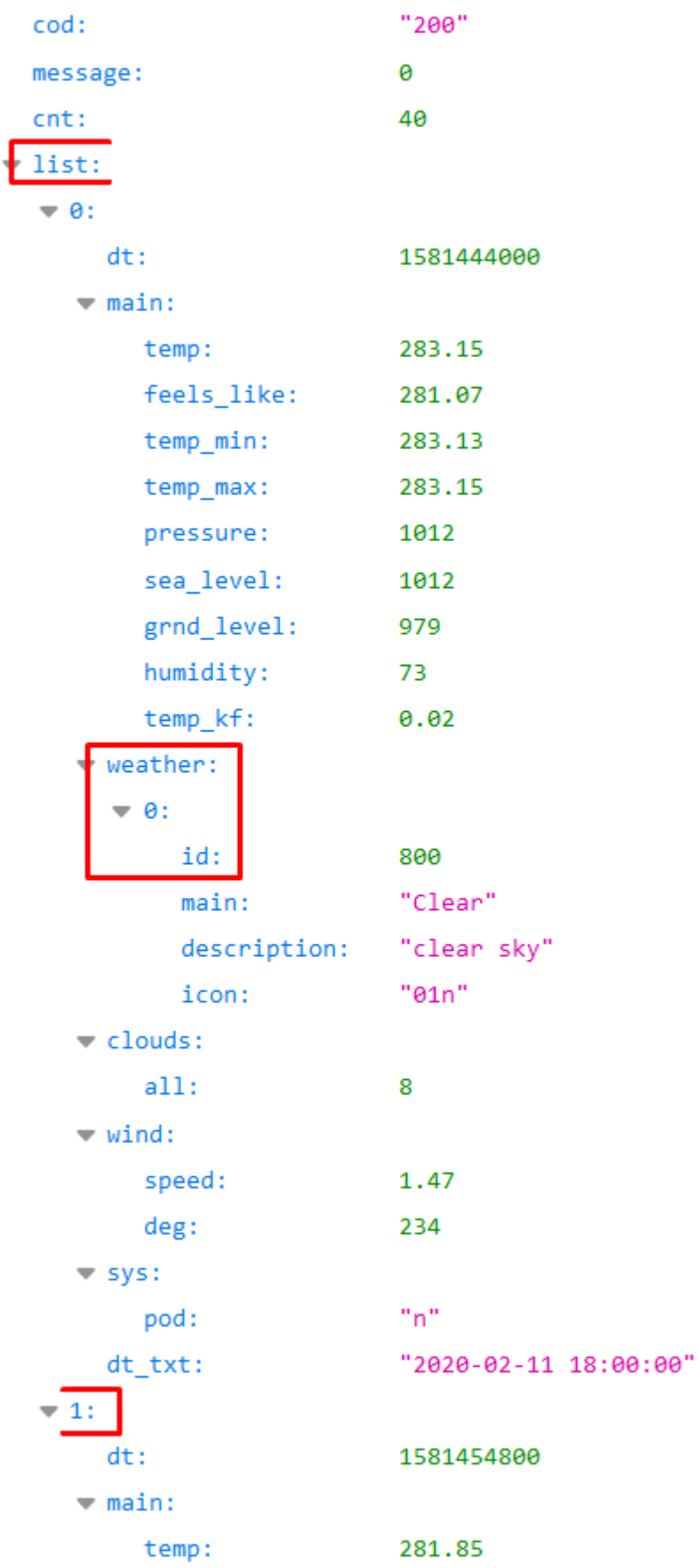
4.2.1 – Έλεγχος ποτίσματος (script: potisma.py)

Το script αυτό είναι υπεύθυνο, για να κάνει όλες τις απαραίτητες διαδικασίας (τις οποίες τις αναφέραμε στην εισαγωγή της ενότητας αυτής) για να ληφθεί η απόφαση για το αν θα δώσει εντολή να ποτίσει ή όχι (όπως αναφέρθηκε και στην εισαγωγή αυτής της ενότητας, σε αυτήν την πτυχιακή δεν ασχοληθήκαμε καθόλου με hardware, επομένως και το «πότισμα» που λέμε είναι φανταστικό). Πάμε τώρα να αναλύσουμε τον κώδικα του script αυτού. Αρχικά ορίζουμε το url στο οποίο θα κάνουμε το request για να πάρουμε τα δεδομένα καιρού (απόσπασμα κώδικα 4.2.1.1 γραμμή 1). Έπειτα, «ανοίγουμε» σύνδεση με το παραπάνω url και αποθηκεύουμε το response στην μεταβλητή «**respone**» (απόσπασμα κώδικα 4.2.1.1 γραμμή 2). Τα δεδομένα που μας ήρθαν από το παραπάνω url έχουν την μορφή **json** (γι' αυτό χρησιμοποιούμε την εντολή **json.loads**). Τα δεδομένα αυτά τα αποθηκεύουμε στην μεταβλητή **data** (απόσπασμα κώδικα 4.2.1.1 γραμμή 3).

```
01. url = "http://api.openweathermap.org/data/2.5/forecast?q=Giannitsa&APPID=e6523ae81ba7cbe19ebc05524dea0c78"
02. respone = urllib.urlopen(url)
03. data = json.loads(respone.read())
04.
05. weatherId = data["list"][1]["weather"][0]["id"]
06.
07. igrasia = randint(0,9)
08. informationPotisma = ""
09.
10. file = open('public_html/infoPotismaForUser.txt','r')
11. apofasiUser = file.read()
12.
13. if (apofasiUser != "minPotiseis"):
14.     if (weatherId >= 200 and weatherId <= 622) and igrasia < 5:
15.         informationPotisma = "0"
16.     else:
17.         informationPotisma = "1"
18. else:
19.     informationPotisma = "0"
20.
21. data = {}
22. data['infoPotisma'] = []
23. data['infoPotisma'].append({
24.         'info': informationPotisma
25. })
26. with open('public_html/dataInfoPotisma.json','w')as outfile:
27.         json.dump(data,outfile)
28.
29. file = open("public_html/dataForecast.json","w")
30. file.write("")
31. file.close()
32.
33. file = open("public_html/infoPotismaForUser.txt","w")
34. file.write("")
35. file.close()
```

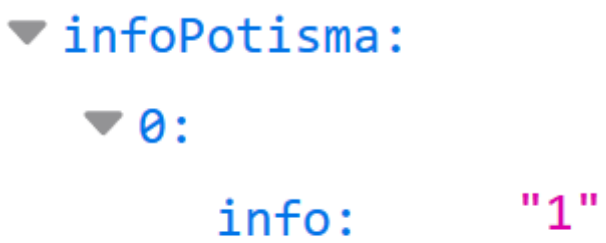
Απόσπασμα κώδικα 4.2.1.1

Τώρα η μεταβλητή «**data**» έχει όλο το json περιεχόμενο που μας επέστρεψε το **openweathermap** (εικόνα 18). Το οποίο περιέχει πολλές πληροφορίες για την πρόγνωση του καιρού στα «Γιαννιτσά» και όχι μόνο. Για τις ανάγκες αυτής της πτυχιακής το μόνο που χρειαζόμαστε απ' όλο αυτό το json αρχείο είναι το **["list"][1]["weather"][0]["id"]** (εικόνα 18 κόκκινα τετραγωνάκια) το περιεχόμενο του οποίου το αποθηκεύουμε στην μεταβλητή «**weatherId**» (απόσπασμα κώδικα 4.2.1.1 γραμμή 5).



Εικόνα 18 – Πρόγνωση του καιρού

Όπως έχει αναφερθεί παραπάνω, επειδή στην συγκεκριμένη πτυχιακή δεν ασχοληθήκαμε καθόλου με το hardware και θέλαμε να πάρουμε μια τιμή υγρασίας, για να μπορέσουμε να κάνουμε τους ελέγχους που απαιτούνται για τις ανάγκες της πτυχιακής σκεφτήκαμε η τιμή της υγρασίας να αποκτάται με τυχαίο τρόπο και έτσι βάλαμε την εντολή «**igrasia = randint(0,9)**» (απόσπασμα κώδικα 4.2.1.1 γραμμή 7). Στη συνέχεια ανοίγουμε το αρχείο στο οποίο αποθηκεύεται η επιλογή του χρήστη (η οποία στέλνεται μέσω της android εφαρμογής) απόσπασμα κώδικα 4.2.1.1 γραμμή 10), διαβάζουμε το αρχείο και αποθηκεύουμε το περιεχόμενο του αρχείου αυτού στην μεταβλητή «**apofasiUser**» (απόσπασμα κώδικα 4.2.1.1 γραμμή 11). Έπειτα, ελέγχουμε αν η τιμή της μεταβλητής «**apofasiUser**» είναι διάφορη του «**minPotiseis**». Στην περίπτωση που ισχύει αυτό, τότε η απόφαση για το αν θα πραγματοποιηθεί πότισμα (τιμή μεταβλητής «**1**») ή όχι (τιμή μεταβλητής «**0**») θα ληφθεί ανάλογα με το αν προβλέπεται βροχή ή όχι (στο site openweathermap τα id από 200 έως 622 δηλώνουν βροχή ή χιόνι) και από την υγρασία του χώματος (μικρότερη του πέντε) (απόσπασμα κώδικα 4.2.1.1 γραμμή 14). Αν δεν ισχύει αυτό, δηλαδή η τιμή της μεταβλητής «**apofasiUser**» είναι ίση με «**minPotiseis**» τότε δεν θα πραγματοποιηθεί πότισμα ακόμα και αν δεν προβλέπεται βροχή και η υγρασία του χώματος είναι μικρότερη του πέντε. Παρακάτω, δημιουργούμε json περιεχόμενο με την απόφαση που λήφθηκε (απόσπασμα κώδικα 4.2.1.1 γραμμές 21-25). Ένα παράδειγμα του json περιεχομένου που δημιουργήσαμε απεικονίζεται στην εικόνα 19.



Εικόνα 19 – Πληροφόρηση ότι θα πραγματοποιηθεί πότισμα

Στη συνέχεια, ανοίγουμε το αρχείο στο οποίο θα αποθηκεύσουμε το json περιεχόμενο και πραγματοποιούμε την αποθήκευση (απόσπασμα κώδικα 4.2.1.1 γραμμές 26-27). Έπειτα, εφόσον το script της πρόγνωσης εκτελείται δέκα λεπτά πριν τρέξει το script του ποτίσματος, μπορούμε μέσα σε αυτά τα δέκα λεπτά να δούμε αν θα πραγματοποιηθεί πότισμα ή όχι. Επομένως, όταν περάσουν αυτά τα δέκα λεπτά και τρέξει και το script του ποτίσματος, δεν υπάρχει λόγος το αρχείο που περιέχει την πρόγνωση να είναι γεμάτο, γιατί στην android εφαρμογή αν αυτό το αρχείο είναι γεμάτο, σε οποιαδήποτε στιγμή τρέξει η εφαρμογή (ακόμα και αν δεν είναι δέκα λεπτά πριν την εκτέλεση του script του ποτίσματος) θα λέει ότι πραγματοποιήθηκε πρόγνωση. Έτσι, σβήνουμε το περιεχόμενο του αρχείου

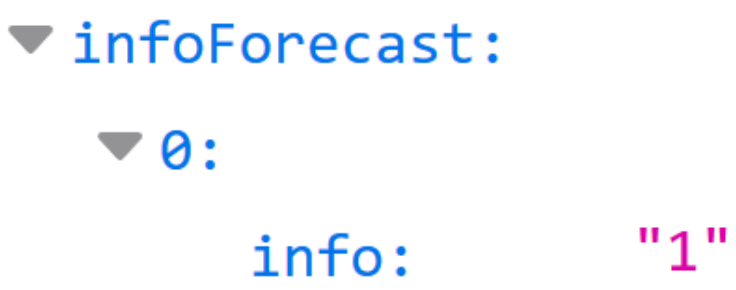
(απόσπασμα κώδικα 4.2.1.1 γραμμές 29-31) που περιλαμβάνει την πληροφορία της πρόγνωσης. Τέλος, σβήνουμε το περιεχόμενο του αρχείου που περιέχει την επιλογή του χρήστη (αποστάλθηκε από την android εφαρμογή) γιατί αν τρέξει το script (**potise.py**) την επόμενη φορά χωρίς να διαγράψουμε το περιεχόμενο του αρχείου «**infoPotismaForUser.txt**» τότε υπάρχει περίπτωση να τρέξει με δεδομένα που έδωσε ο χρήστης παλιότερα [12][27][28][29].

4.2.2 – Πρόγνωση (script: forecastBeforeTenMinutes.py)

Το script της πρόγνωσης τρέχει δέκα λεπτά πριν τρέξει το script του ποτίσματος. Αυτό το script το δημιουργήσαμε έτσι, ώστε πριν πραγματοποιηθεί πότισμα να γίνει πρόγνωση για το αν θα βρέξει ή όχι και να ενημερωθεί ο χρήστης για να αποφασίσει το τι θέλει να κάνει (να ποτίσει ή όχι). Την επιλογή «**Πότισε**» ή «**Μην ποτίσεις**» την στέλνει ο χρήστης στον server μέσω της android εφαρμογής που θα περιγράψουμε σε επόμενη ενότητα. Πάμε τώρα να εξηγήσουμε τον κώδικα αυτού του script. Αρχικά ορίζουμε το url στο οποίο θα κάνουμε το request για να πάρουμε τα δεδομένα καιρού (απόσπασμα κώδικα 4.2.2.1 γραμμή 1). Έπειτα, «ανοίγουμε» σύνδεση με το παραπάνω url και αποθηκεύουμε το response στην μεταβλητή «**respone**» (απόσπασμα κώδικα 4.2.2.1 γραμμή 2). Τα δεδομένα που μας ήρθαν από το παραπάνω url έχουν την μορφή **json** (γι' αυτό χρησιμοποιούμε την εντολή **json.loads**). Τα δεδομένα αυτά τα αποθηκεύουμε στην μεταβλητή **data** (απόσπασμα κώδικα 4.2.2.1 γραμμή 3). Τώρα η μεταβλητή «**data**» έχει όλο το json περιεχόμενο που μας επέστρεψε το **openweathermap** (εικόνα 18). Το οποίο περιέχει πολλές πληροφορίες για την πρόγνωση του καιρού στα «Γιαννιτσά» και όχι μόνο. Για τις ανάγκες αυτής της πτυχιακής το μόνο που χρειαζόμαστε απ' όλο αυτό το json αρχείο είναι το **["list"][1]["weather"][0]["id"]** (εικόνα 18 κόκκινα τετραγωνάκια) το περιεχόμενο του οποίου το αποθηκεύουμε στην μεταβλητή «**weatherId**» (απόσπασμα κώδικα 4.2.2.1 γραμμή 5). Έπειτα, ανάλογα με την τιμή της μεταβλητής «**weatherId**» θα αποφασιστεί αν προβλέπεται βροχή (**informationForecast = "0"**) ή όχι (**informationForecast = "1"**) (στο site openweathermap τα id από 200 έως 622 δηλώνουν βροχή ή χιόνι). Παρακάτω, δημιουργούμε json περιεχόμενο με την απόφαση που λήφθηκε (απόσπασμα κώδικα 4.2.2.1 γραμμές 12-16). Ένα παράδειγμα του json περιεχομένου που δημιουργήσαμε απεικονίζεται στην εικόνα 20. Τέλος, ανοίγουμε το αρχείο στο οποίο θα αποθηκεύσουμε το json περιεχόμενο αυτό και πραγματοποιούμε την αποθήκευση (απόσπασμα κώδικα 4.2.2.1 γραμμές 17-18) [12][27][28][29].

```
01. url = "http://api.openweathermap.org/data/2.5/forecast?q=Giannitsa&APPID=e6523ae81ba7cbe19ebc05524dea0c78"
02. respone = urllib.urlopen(url)
03. data = json.loads(respone.read())
04.
05. weatherId = data["list"][1]["weather"][0]["id"]
06.
07. if weatherId >= 200 and weatherId <= 622:
08.     informationForecast = "0"
09. else:
10.     informationForecast = "1"
11.
12. data = {}
13. data['infoForecast'] = []
14. data['infoForecast'].append({
15.         'info': informationForecast
16. })
17. with open('public_html/dataForecast.json','w')as outfile:
18.         json.dump(data,outfile)
```

Απόσπασμα κώδικα 4.2.2.1



Εικόνα 20 – Πληροφόρηση ότι δεν προβλέπεται βροχή

4.2.3 – Crontab linux

Όπως αναφέρθηκε προηγουμένους τα δύο παραπάνω scripts τα οποία βρίσκονται στον server εκτελούνται ανά τρεις ώρες. Με την διαφορά ότι το ένα από τα δύο (forecastBeforeTenMinutes) τρέχει δέκα λεπτά πριν τρέξει το άλλο. Για να τρέξει το καθένα από τα δύο την χρονική στιγμή που πρέπει, θα πρέπει να προσθέσουμε δύο εγγραφές (μία για το κάθε script) στο αρχείο του **crontab** στον server. Οι εγγραφές που θα προστεθούν φαίνονται παρακάτω.

```
0 */3 * * * python potisma.py
50 2,5,8,11,14,17,20,23 * * * python forecastBeforeTenMinutes.py
```

Το πρώτο ψηφίο (τα ψηφία χωρίζονται με το κενό) δηλώνει τα λεπτά, το δεύτερο τις ώρες, το τρίτο τις μέρες, το τέταρτο τους μήνες και το έκτο την ημέρα της εβδομάδας (σε περίπτωση που π.χ. θέλουμε να γίνεται μια ενέργεια μόνο Παρασκευή). Επίσης, όπου υπάρχει το αστεράκι «*» αυτό σημαίνει «κάθε». Λοιπόν, με βάση το crontab παραπάνω βλέπουμε ότι το script «potisma.py» τρέχει στο μηδέν λεπτό της ώρας κάθε τρεις ώρες (*/3). Το script

«forecastBeforeTenMinutes» τρέχει στο πεντηκοστό λεπτό της ώρας τις ώρες 2,5,8… κτλ κάθε μέρα [30] [31].

4.3 – Android εφαρμογή

Στην android εφαρμογή παρουσιάζονται οι αποφάσεις οι οποίες λήφθηκαν κατά την εκτέλεση των scripts στον server. Επίσης δίνεται η δυνατότητα στον χρήστη, να επέμβει και να αλλάξει την απόφαση που πάρθηκε στον server. Ακόμα, η εφαρμογή παρέχει και άλλες υπηρεσίες. Αυτές είναι η δυνατότητα να δει τι καιρό κάνει την συγκεκριμένη χρονική στιγμή σε μία πόλη της επιλογής του, καθώς και να δει πρόγνωση του καιρού για την συγκεκριμένη πόλη. Επίσης, μπορεί να δει αγροτικές ειδήσεις από κάποια ειδησεογραφικά sites. Μία άλλη υπηρεσία που παρέχει η εφαρμογή είναι η δυνατότητα ο χρήστης να ορίσει μια ειδοποίηση για μια εργασία η οποία πρέπει να γίνει μια συγκεκριμένη χρονική στιγμή. Επίσης, άλλη υπηρεσία που παρέχει η εφαρμογή είναι η δυνατότητα να ορίζει γεγονότα ο χρήστης στο custom ημερολόγιο της εφαρμογής. Και η τελευταία υπηρεσία που παρέχει η εφαρμογή είναι η δημιουργία σημείωσης.

4.3.1 – Λήψη πληροφοριών από τον server

Σε αυτήν την υποενότητα θα περιγράψουμε όλες τις διαδικασίες που λαμβάνουν χώρα στο κύριο μέρος της εφαρμογής, για να ληφθούν οι αποφάσεις που πάρθηκαν κατά την εκτέλεση των scripts στον server, καθώς και με ποιο τρόπο μπορεί ο χρήστης να μεταβάλλει τις αποφάσεις αυτές.

4.3.1.1 – Διαδικασίες που γίνονται

Όταν ξεκινήσει η εφαρμογή θα γίνει έλεγχος για το αν υπάρχει πρόσβαση στο διαδίκτυο (απόσπασμα κώδικα 4.3.1.1.1).

```
01. private boolean chechConnection(){
02.
03.         ConnectivityManager cm =
04.                 (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
05.
06.         NetworkInfo activeNetwork = cm.getActiveNetworkInfo();
07.         boolean isConnected = activeNetwork != null &&
08.                 activeNetwork.isConnectedOrConnecting();
09.
10.         return isConnected;
11. }
```

Απόσπασμα κώδικα 4.3.1.1.1

Για να μπορέσει όμως να εκτελεστεί ο κώδικας παραπάνω θα πρέπει να δώσουμε δικαιώματα χρήσης. Για να δώσουμε δικαιώματα χρήσης, θα πρέπει να προσθέσουμε στο Android Manifest το απόσπασμα κώδικα 4.3.1.1.2 [32].

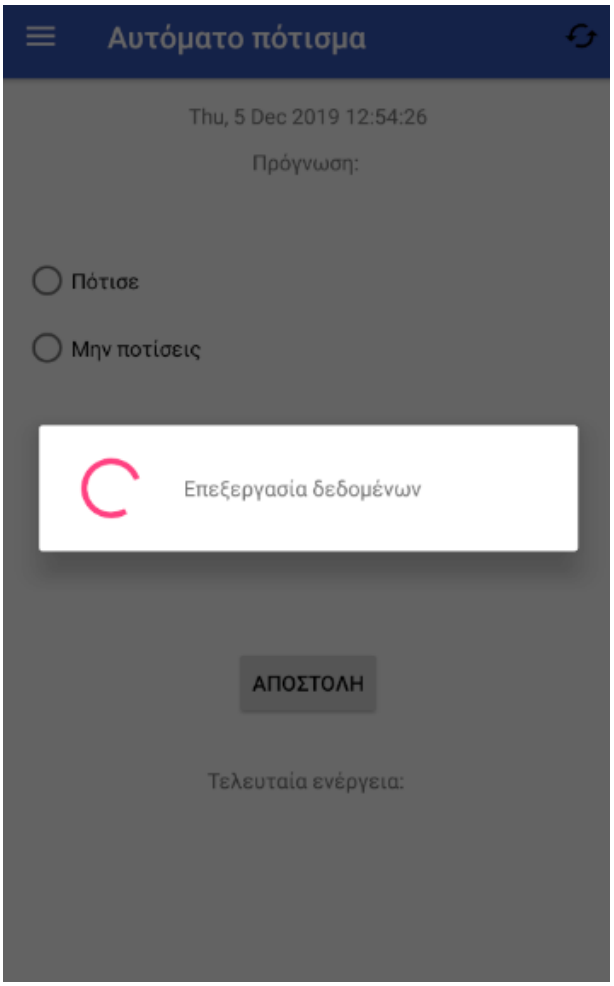
```
01.  <uses-permission android:name="android.permission.INTERNET" />
02.  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Απόσπασμα κώδικα 4.3.1.1.2

Στην περίπτωση που υπάρχει πρόσβαση στο διαδίκτυο τότε στέλνονται δύο **Http Request** στον **server**. Το ένα για να λάβει (η android εφαρμογή) δεδομένα για το «**αν θα ποτίσει**» (απόσπασμα κώδικα 4.3.1.1.3 γραμμή 10) την επόμενη φορά που θα τρέξει το script στον server και το άλλο για να λάβει δεδομένα για το «**αν πότισε**» (απόσπασμα κώδικα 4.3.1.1.3 γραμμή 11) την τελευταία φορά που έτρεξε το script. Περισσότερες λεπτομέρειες για το πώς πραγματοποιούνται τα παραπάνω θα αναφέρουμε παρακάτω. Ακόμα, βλέπουμε στο απόσπασμα κώδικα 4.3.1.1.3 γραμμή 8 ότι ενεργοποιούμε το κουμπί «Αποστολή». Αυτό το κάνουμε επειδή, σε περίπτωση που δεν απαντήσει ο server, γίνεται απενεργοποίηση του κουμπιού. Στη συνέχεια, αν εμείς πατήσουμε το κουμπί «Ανανέωση» (πάνω δεξιά της εφαρμογής) και τότε απαντήσει ο server, ενεργοποιείται ξανά το κουμπί. Επίσης, όσο διαρκεί η επικοινωνία της εφαρμογής με τον server (για να λάβει τις παραπάνω πληροφορίες) εμφανίζεται στην οθόνη της συσκευής ένα παράθυρο διαλόγου (απόσπασμα κώδικα 4.3.1.1.3 γραμμές 4-6) το οποίο φαίνεται στην εικόνα 21 [33][58]. Ακόμα, παρατηρούμε ότι την ασύγχρονη κλάση **ParseDataForecast** την εκτελούμε αμέσως ενώ την **ParseDataInfoPotisma** την βάζουμε σε μεταβλητή. Αυτό το κάνουμε έτσι ώστε να έχουμε την δυνατότητα να **ακυρώσουμε** την εκτέλεση της. Θα αναρωτιέται κανείς, γιατί να το κάνουμε αυτό. Ο λόγος που το κάνουμε αυτό είναι ότι στην περίπτωση που ο server δεν απαντάει (το οποίο το εντοπίζουμε όταν εκτελεστεί η κλάση **ParseDataForecast**) δεν υπάρχει λόγος να εκτελεστεί και η κλάση **ParseDataInfoPotisma**.

```
01.  private void diadikasiaEpikoinonias(){
02.          if (chechConnection()) {
03.
04.              dialogProcessData = new ProgressDialog(Main2Activity.this);
05.              dialogProcessData.setMessage("Επεξεργασία δεδομένων");
06.              dialogProcessData.show();
07.
08.              bApostoli.setEnabled(true);
09.
10.              new ParseDataForecast().execute();
11.              taskParseDataInfoPotisma = (ParseDataInfoPotisma) new ParseDataInfoPotisma().execute();
12.          }
```

Απόσπασμα κώδικα 4.3.1.1.3

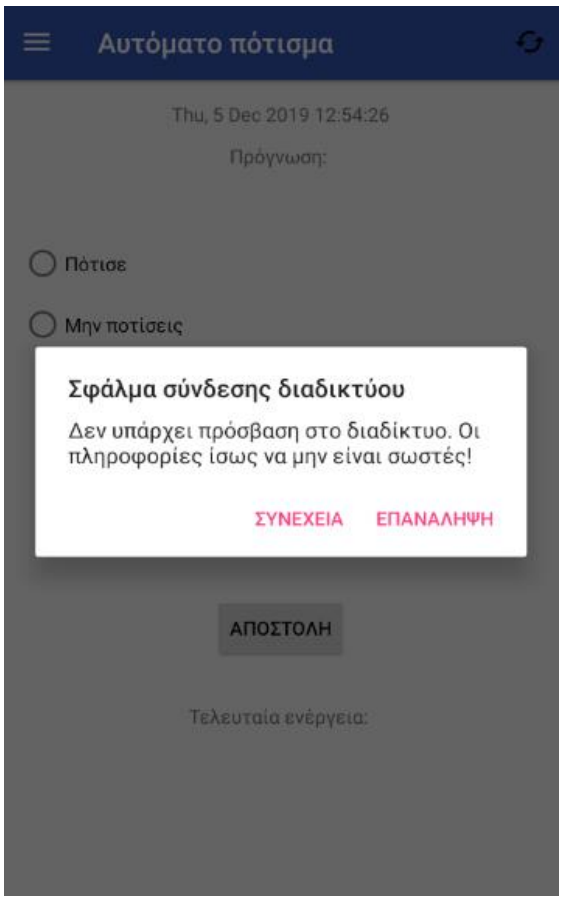


Εικόνα 21 – Progress Dialog

Στην περίπτωση που δεν υπάρχει πρόσβαση στο διαδίκτυο (απόσπασμα κώδικα 4.3.1.1.4) θα εμφανιστεί στην εφαρμογή το παράθυρο διαλόγου που φαίνεται στην εικόνα 22 [33][58]. Αν ο χρήστης επιλέξει «Επανάληψη» θα τρέξει ξανά η μέθοδος «diadikasiaEpikoinonias» (απόσπασμα κώδικα 4.3.1.1.3), για να ξανά ελέγξει αν υπάρχει σύνδεση στο διαδίκτυο. Αν επιλέξει «Συνέχεια» θα φορτώσει τις πληροφορίες για το «αν θα ποτίσει» και για το «αν πότισε» (την τελευταία φορά που έτρεξε το script στον server) από τα αρχεία στα οποία αποθηκεύτηκαν οι πληροφορίες αυτές, την τελευταία φορά που απόκτησε πρόσβαση στο διαδίκτυο η συσκευή του χρήστη. Επίσης, μια άλλη πληροφορία που αποθηκεύεται σε αρχείο είναι ποια χρονική στιγμή έγινε η ενημέρωση των παραπάνω αρχείων (δηλαδή πότε απέκτησε τελευταία φορά σύνδεση στο διαδίκτυο η συσκευή). Η μέθοδος «**getDataForFile**» [33][58] είναι υπεύθυνη να κάνει όλες τις απαραίτητες διαδικασίες για να παρθεί το περιεχόμενο το οποίο είναι αποθηκευμένο στα αρχεία **dataForecast**, **dataInfoPotisma** και **lastUpdateInfoPotisma**. Σε περίπτωση που από την στιγμή που θα εγκατασταθεί η εφαρμογή και τρέξει, δεν συνδεθεί η συσκευή στο διαδίκτυο θα εμφανίσει αντίστοιχο μήνυμα.

```
else {
            new AlertDialog.Builder(context)
                    .setTitle("Σφάλμα σύνδεσης διαδικτύου")
                    .setMessage("Δεν υπάρχει πρόσβαση στο διαδίκτυο. Οι πληροφορίες ίσως να μην είναι σωστές!")
                    .setPositiveButton("Επαναληψη", new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            diadikasiaEpikoinonias();
                        }
                    })
                    .setNegativeButton("Συνέχεια", new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            if(!getDataForFile("dataForecast").equals("")) {
                                TextView txtViewForecastInformationPotisma = findViewById(R.id.txtForecastInfoPotisma);
                                txtViewForecastInformationPotisma.setText(getDataForFile("dataForecast"));
                            }
                            if(!getDataForFile("dataInfoPotisma").equals("")) {
                                TextView txtViewInformationPotisma = findViewById(R.id.txtInfoPotisma);
                                txtViewInformationPotisma.setText(getDataForFile("dataInfoPotisma"));
                            }
                            if(!getDataForFile("lastUpdateInfoPotisma").equals("")) {
                                TextView txtViewLastUpdateInfoPotisma = findViewById(R.id.txtLastUpdateInfoPotisma);
                                txtViewLastUpdateInfoPotisma.setText(getDataForFile("lastUpdateInfoPotisma"));
                            }
                            else {
                                AlertDialog.Builder builder = new AlertDialog.Builder(context);
                                builder.setTitle("Ενημέρωση");
                                builder.setMessage("Επειδή χρησιμοποιείτε πρώτη φορά την εφαρμογή " +
                                        "θα πρέπει να ανοίξετε την σύνδεση στο διαδίκτυο");

                                builder.setPositiveButton("Ενταξει", null);

                                AlertDialog dialogCheckFiles = builder.create();
                                dialogCheckFiles.show();
                            }
                        }
                    })
                    .show();
}
```

Απόσπασμα κώδικα 4.3.1.1.4



Εικόνα 22 – Alert Dialog

Για να μπορέσει όμως να γίνει αποθήκευση και ανάγνωση αρχείων, θα πρέπει να δώσουμε άλλα δύο δικαιώματα χρήσης στο Android Manifest (απόσπασμα κώδικα 4.3.1.1.5).

```
01.  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
02.  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

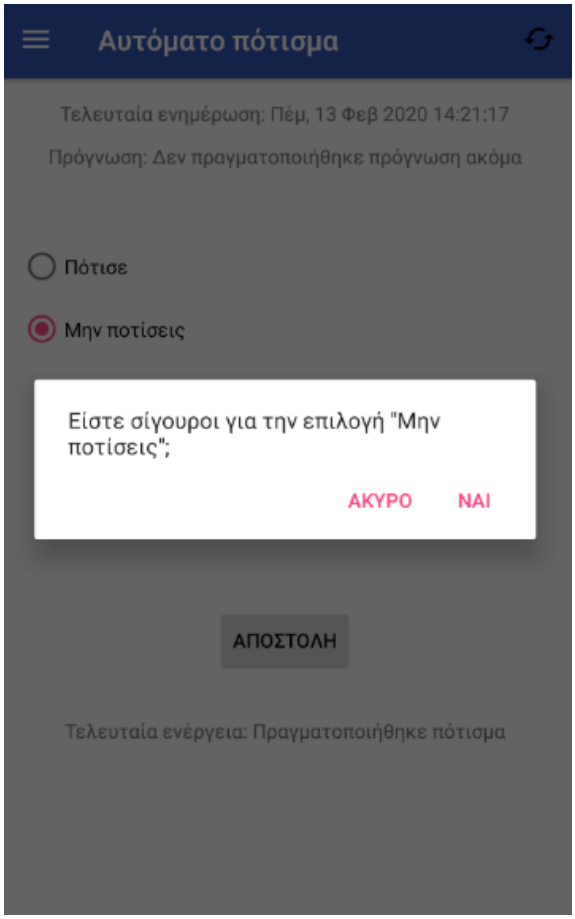
Απόσπασμα κώδικα 4.3.1.1.5

4.3.1.2 – Αλλαγή απόφασης από τον χρήστη

Στην εισαγωγή του κεφαλαίου αυτού, καθώς και στην αρχή της ενότητας αυτής, αναφέραμε την δυνατότητα που παρέχει η εφαρμογή να αλλάξει ο χρήστης την απόφαση που λήφθηκε κατά την εκτέλεση των scripts τα οποία τα αναφέραμε στην προηγούμενη ενότητα. Τώρα ήρθε η στιγμή να εξηγήσουμε με ποιο τρόπο γίνεται αυτή η αλλαγή της απόφασης από τον χρήστη. Αν όλα πάνε καλά και η επικοινωνία με τον server ολοκληρωθεί με επιτυχία τότε θα έχουμε το αποτέλεσμα που φαίνεται στην εικόνα 23.

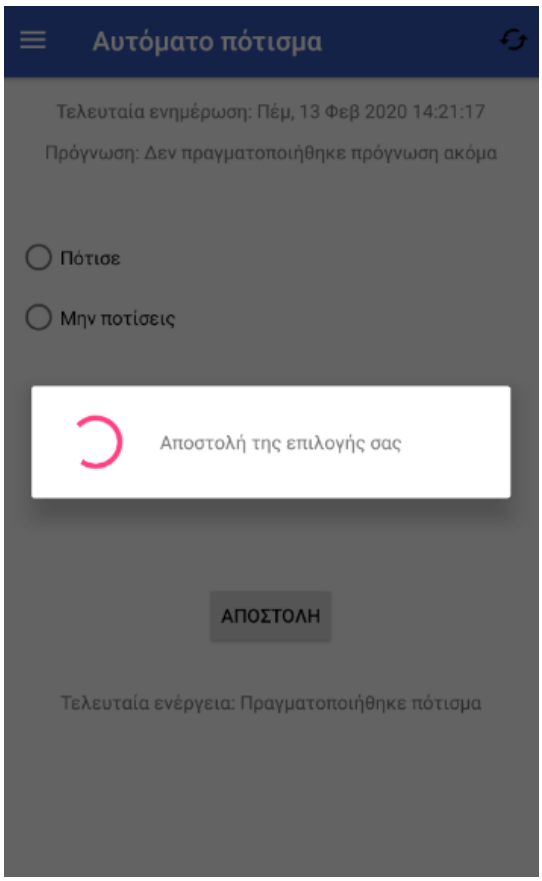
Εικόνα 23 – Επιτυχή λήψη πληροφοριών από τον server

Βλέπουμε ότι δεν πραγματοποιήθηκε πρόγνωση ακόμα (η πρόγνωση γίνεται δέκα λεπτά πριν τρέξει το script του ποτίσματος) και ότι την τελευταία φορά που έτρεξε το script του ποτίσματος πραγματοποιήθηκε πότισμα. Τώρα, όποια και αν είναι η απόφαση που λήφθηκε μετά την εκτέλεση των scripts στον server, ο χρήστης μπορεί να την μεταβάλει και να επιλέξει αυτός για το αν θα πραγματοποιηθεί πότισμα ή όχι. Όταν επιλέξει ανάμεσα στο «Πότισε» ή «Μην ποτίσεις» και πατήσει το κουμπί «ΑΠΟΣΤΟΛΗ», τότε αφ' ότου κάνει έλεγχο για το άμα υπάρχει πρόσβαση στο διαδίκτυο θα μας εμφανίσει το παράθυρο διαλόγου της εικόνας 24 [33][58].



Εικόνα 24 – Επιλογή χρήστη

Στην περίπτωση που πατήσουμε «ΝΑΙ», τότε θα εμφανιστεί το παράθυρο διαλόγου που φαίνεται στην εικόνα 25 [33][58].



Εικόνα 25 – Progress Dialog επιλογής χρήστη

Και όταν ολοκληρωθεί η αποστολή της επιλογής του χρήστη στον server, τότε θα εμφανιστεί το snackbar που φαίνεται στην εικόνα 26 [34].

Εικόνα 26 – Snackbar

Στην περίπτωση που πατήσει «Αποστολή» χωρίς να έχει επιλέξει κάποια από τις επιλογές «Πότισε», «Μην ποτίσεις» τότε θα του εμφανίσει ένα παράθυρο διαλόγου το οποίο θα τον ενημερώνει ότι δεν επέλεξε κάποια από τις παραπάνω επιλογές. Στην περίπτωση που δεν υπάρχει σύνδεση στο διαδίκτυο και επιλέξει μία από τις δύο επιλογές και πατήσει «Αποστολή», τότε θα εμφανιστεί ένα άλλο παράθυρο διαλόγου το οποίο θα τον ενημερώνει ότι δεν υπάρχει πρόσβαση στο διαδίκτυο.

4.3.1.3 – Parse των δεδομένων πρόγνωσης

Όταν τρέξει το script (στον server) για την πρόγνωση, αποθηκεύει την απόφαση που πήρε σε ένα **Json** αρχείο μέσα στον server. Για να πάρει το περιεχόμενο του αρχείου αυτού η εφαρμογή θα τρέξει το απόσπασμα κώδικα 4.3.1.3.1 [35].

```
01. @Override
02. protected String doInBackground(String... arg0) {
03. 
04.     try {
05. 
06.         HttpHandler sh = new HttpHandler();
07.         String jsonStr = sh.makeServiceCall(yourJsonStringUrl);
08.         JSONObject jsonObj = new JSONObject(jsonStr);
09.         JSONArray dataJsonArr = jsonObj.getJSONArray("infoForecast");
10.         JSONObject c = dataJsonArr.getJSONObject(0);
11. 
12.         // Storing each json item in variable
13.         informationPotisma = c.getString("info");
14. 
15.         if(informationPotisma.equals("0"))
16.             messageDisplay = "Πρόγνωση: Προβλέπετε βροχή. Δεν θα ποτίσω";
17.         else if(informationPotisma.equals("1"))
18.             messageDisplay = "Πρόγνωση: Δεν προβλέπετε βροχή. Θα ποτίσω";
```

Απόσπασμα κώδικα 4.3.1.3.1

Για να κάνουμε **Request** στον server, για να μας επιστρέψει το περιεχόμενο του json, δημιουργούμε ένα αντικείμενο **HttpHandler** και καλούμε την μέθοδο **makeServiceCall** βάζοντας ως παράμετρο το url στο οποίο είναι αποθηκευμένο το json αρχείο. Περισσότερες λεπτομέρειες για την κλάση **HttpHandler** και τις μεθόδους της θα πούμε σε επόμενη υποενότητα. Σε αυτό το σημείο να κάνουμε κάποιες παρατηρήσεις. Το script της πρόγνωσης τρέχει ανά τρεις ώρες. Για την ακρίβεια, τρέχει δέκα λεπτά πριν το script του ποτίσματος. Αυτό σημαίνει ότι αν ο χρήστης τρέξει την εφαρμογή οποιαδήποτε στιγμή πριν την στιγμή που έχει τρέξει το script της πρόγνωσης το αρχείο που θα φτάσει στην εφαρμογή θα είναι κενό. Το γεγονός αυτό θα δημιουργήσει εξαίρεση (**exception**). Για να αντιμετωπίσουμε το πρόβλημα αυτό, θα «πιάσουμε» την εξαίρεση όταν προκληθεί (απόσπασμα κώδικα 4.3.1.3.2) και θα την χειριστούμε με τέτοιο τρόπο έτσι ώστε να μας εξυπηρετεί. Πώς θα το κάνουμε αυτό; Το γεγονός ότι το αρχείο της πρόγνωσης είναι άδειο δηλώνει ότι η πρόβλεψη δεν έγινε ακόμα. Έτσι, θα βάλουμε να εμφανιστεί το μήνυμα: «**Πρόγνωση: Δεν πραγματοποιήθηκε πρόγνωση ακόμα**».

```
01. catch (JSONException e) {
02.     messageDisplay = "Πρόγνωση: Δεν πραγματοποιήθηκε πρόγνωση ακόμα";
03. }
```

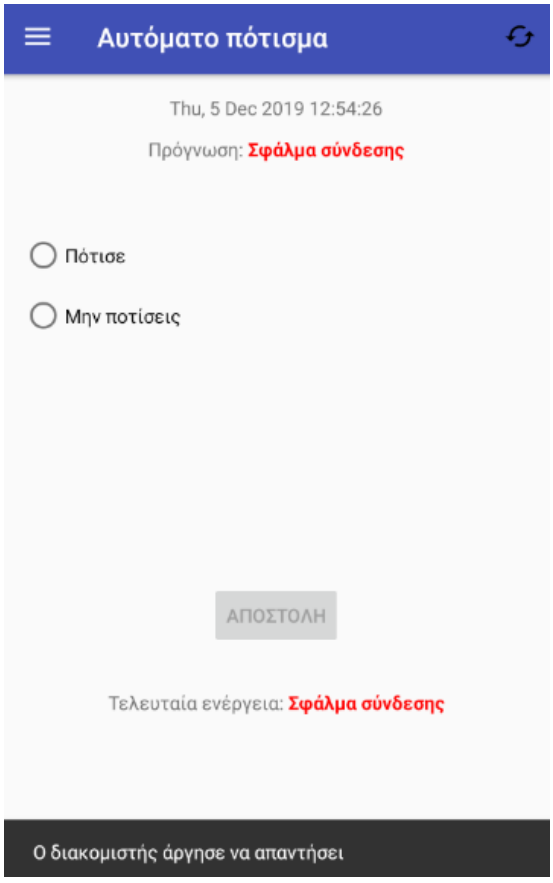
Απόσπασμα κώδικα 4.3.1.3.2

Στη περίπτωση που η εφαρμογή στείλει το **Http Request** και ο server αργήσει να απαντήσει θα προκληθεί πάλι εξαίρεση την οποία πρέπει να «πιάσουμε» και να αντιμετωπίσουμε. Για να μπορέσουμε όμως να αποκτήσουμε πρόσβαση στα **TextView** και στο **Button** τα οποία βρίσκονται στο κύριο thread του activity, θα χρησιμοποιήσουμε την εντολή «**runOnUiThread**» [36]. Στη συνέχεια, όταν

προκληθεί εξαίρεση αρχικά θα γίνει **dismiss** (απόσπασμα κώδικα 4.3.1.3.3) του παραθύρου διαλόγου (εικόνα 21). Έπειτα εμφανίζεται το μήνυμα «**Ο διακομιστής άργησε να απαντήσει**» [34] στο κάτω μέρος της οθόνης της συσκευής και στα δύο TextView «Σφάλμα σύνδεσης». Ακόμη, γίνεται απενεργοποίηση του κουμπιού «**Αποστολή**» (εικόνα 27) έτσι, ώστε να μην έχει την δυνατότητα ο χρήστης αφού δεν απαντάει ο server να του αποστείλει πληροφορία. Για τον ίδιο λόγο που απενεργοποιήσαμε το κουμπί, ακυρώνουμε και την εκτέλεση της ασύγχρονης κλάσης **ParseDataInfoPotisma**.

```
catch (NullPointerException e) {
                dialogProcessData.dismiss();

                runOnUiThread(new Runnable() {
                    public void run() {
                        Snackbar.make(drawer, "Ο διακομιστής άργησε να απαντήσει", Snackbar.LENGTH_LONG)
                                .setAction("Action", null).show();

                        bApostoli.setEnabled(false);

                        TextView txtViewForecastInformationPotisma = findViewById(R.id.txtForecastInfoPotisma);
                        txtViewForecastInformationPotisma.setText(Html.fromHtml("Πρόγνωση: <font color='red'><b>Σφάλμα σύνδεσης</b></font>"));

                        TextView txtViewInformationPotisma = findViewById(R.id.txtInfoPotisma);
                        txtViewInformationPotisma.setText(Html.fromHtml("Τελευταία ενέργεια: <font color='red'><b>Σφάλμα σύνδεσης</b></font>"));
                    }
                });
                taskParseDataInfoPotisma.cancel(false);
}
```

Απόσπασμα κώδικα 4.3.1.3.3



Εικόνα 27 – Ο διακομιστής δεν απαντάει

Τέλος, για να ενημερώσουμε το UI της εφαρμογής και να αποθηκεύσουμε σε αρχείο το αποτέλεσμα της πρόγνωσης χρησιμοποιούμε το απόσπασμα κώδικα 4.3.1.3.4 [33][36][58]. Η μέθοδος «**setDataForFile**» χρησιμοποιείται για να κάνει όλες τις απαραίτητες διαδικασίες, για να αποθηκευτεί η πρόγνωση στο αρχείο.

```
runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    if(messageDisplay != null) {
                        TextView txtViewForecastInformationPotisma = findViewById(R.id.txtForecastInfoPotisma);
                        txtViewForecastInformationPotisma.setText(messageDisplay);
                        setDataForFile("dataForecast", messageDisplay);
                    }
                }
});
```

Απόσπασμα κώδικα 4.3.1.3.4

4.3.1.4 – Parse των δεδομένων ποτίσματος

Όταν τρέξει το script (στον server) του ποτίσματος, αποθηκεύει την απόφαση που πήρε σε ένα **Json** αρχείο μέσα στον server. Για να πάρει το περιεχόμενο του αρχείου αυτού η εφαρμογή θα τρέξει το απόσπασμα κώδικα 4.3.1.4.1 [35].

```
@Override
protected String doInBackground(String... arg0) {

            try {

                HttpHandler sh = new HttpHandler();
                String jsonStr = sh.makeServiceCall(yourJsonStringUrl);
                JSONObject jsonObj = new JSONObject(jsonStr);
                JSONArray dataJsonArr = jsonObj.getJSONArray("infoPotisma");
                JSONObject c = dataJsonArr.getJSONObject(0);

                // Storing each json item in variable
                informationPotisma = c.getString("info");

                if(informationPotisma.equals("0"))
                    messageDisplay = "Τελευταία ενέργεια: Δεν πραγματοποιήθηκε πότισμα";
                else if(informationPotisma.equals("1"))
                    messageDisplay = "Τελευταία ενέργεια: Πραγματοποιήθηκε πότισμα";
```

Απόσπασμα κώδικα 4.3.1.4.1

Για να κάνουμε **Request** στον server για να μας επιστρέψει το περιεχόμενο του json, δημιουργούμε ένα αντικείμενο **HttpHandler** και καλούμε την μέθοδο **makeServiceCall** βάζοντας ως παράμετρο το url στο οποίο είναι αποθηκευμένο το json αρχείο. Περισσότερες λεπτομέρειες για την κλάση **HttpHandler** και τις μεθόδους της θα πούμε σε επόμενη υποενότητα. Το script του ποτίσματος τρέχει ανά τρεις ώρες. Σε αυτό το σημείο του κώδικα δεν χρειάζεται να «πιάσουμε» κάποιες εξαιρέσεις, όπως π.χ. τι θα γίνει σε περίπτωση που δεν απαντήσει ο server ή τι θα γίνει σε περίπτωση που το αρχείο που περιέχει πληροφορίες για το πότισμα είναι κενό. Ο λόγος είναι πρώτον, γιατί το αν απαντήσει ο server ή όχι το αντιμετωπίζουμε όταν εκτελεστεί ο κώδικας της πρόγνωσης (παραπάνω

υποενότητα) και δεύτερον, το αρχείο που περιέχει πληροφορίες για το πότισμα είναι πάντα γεμάτο. Τέλος, για να ενημερώσουμε το UI της εφαρμογής και να αποθηκεύσουμε σε αρχείο την πληροφορία για το πότισμα καθώς και την ημερομηνία τις τελευταίας ενημέρωσης χρησιμοποιούμε το απόσπασμα κώδικα 4.3.1.4.2 [33][36][58][62].

```
runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    TextView txtViewInformationPotisma = findViewById(R.id.txtInfoPotisma);
                    txtViewInformationPotisma.setText(messageDisplay);

                    SimpleDateFormat dateFormat = new SimpleDateFormat("EEE, d MMM yyyy HH:mm:ss");
                    Date dateTime = Calendar.getInstance().getTime();

                    TextView txtViewLastUpdateInfoPotisma = findViewById(R.id.txtLastUpdateInfoPotisma);
                    txtViewLastUpdateInfoPotisma.setText("Τελευταία ενημέρωση: "+dateFormat.format(dateTime));

                    setDataForFile("lastUpdateInfoPotisma", "Τελευταία ενημέρωση: "+dateFormat.format(dateTime));
                    setDataForFile("dataInfoPotisma", messageDisplay);
                }
});
```

Απόσπασμα κώδικα 4.3.1.4.2

4.3.1.5 – Μέθοδοι HttpHandler

Αυτή η κλάση είναι υπεύθυνη στο να αποστείλει ένα Http Request σε ένα url που θα της δοθεί (makeServiceCall) και να μετατρέψει το stream που θα δεχτεί από το Http Request που έκανε σε string (convertStreamToString).

4.3.1.5.1 – Μέθοδος makeServiceCall

Οι κλάσεις **ParseDataForecast** και **ParseDataInfoPotisma** για να πάρουν τα δεδομένα που χρειάζονται από τον server χρησιμοποιούν την μέθοδο **makeServiceCall** (απόσπασμα κώδικα 4.3.1.5.1.1) η οποία είναι υπεύθυνη να υλοποιήσει την επικοινωνία με τον server. Αυτό θα το κάνει στέλνοντας ένα **Http Request** στο **url** που δέχτηκε ως παράμετρο.

```
public String makeServiceCall(String reqUrl) {

            String response = null;

            try {
                URL url = new URL(reqUrl);
                HttpURLConnection conn = (HttpURLConnection) url.openConnection();
                conn.setRequestMethod("GET");

                InputStream in = new BufferedInputStream(conn.getInputStream());
                response = convertStreamToString(in);
            } catch (Exception e) {
                Log.e(TAG, "Exception makeServiceCall: " + e.getMessage());
                e.printStackTrace();
            }
            return response;
}
```

Απόσπασμα κώδικα 4.3.1.5.1.1

Μετά την ολοκλήρωση του Http Request, τα δεδομένα που έρχονται από τον server έχουν την μορφή **Stream**. Το περιεχόμενο του Stream αυτού το αποθηκεύουμε στην μεταβλητή **InputStream** «**in**» (απόσπασμα κώδικα 4.3.1.5.1.1). Την μετατροπή του **Stream** αυτού σε **String** θα την αναλάβει η μέθοδος «**convertStreamToString**». Λεπτομέρειες για την μέθοδο αυτή θα πούμε στην επόμενη υποενότητα [33][58].

4.3.1.5.2 – Μέθοδος convertStreamToString

Η μέθοδος **convertStreamToString** καλείται από την μέθοδο **makeServiceCall** με παράμετρο το stream που δέχτηκε από το request στον server. Η μέθοδος αυτή κάνει τις απαραίτητες ενέργειες έτσι, ώστε το stream που δέχτηκε, να το μετατρέψει σε string και να το επιστρέψει (απόσπασμα κώδικα 4.3.1.5.2.1).

```
01. private String convertStreamToString(InputStream is) {
02. 
03.             BufferedReader reader = new BufferedReader(new InputStreamReader(is));
04.             StringBuilder sb = new StringBuilder();
05. 
06.             String line;
07.             try {
08.                 while ((line = reader.readLine()) != null) {
09.                     sb.append(line).append('\n');
10.                 }
11.             } catch (IOException e) {
12.                 e.printStackTrace();
13.             } finally {
14.                 try {
15.                     is.close();
16.                 } catch (IOException e) {
17.                     e.printStackTrace();
18.                 }
19.             }
20.             return sb.toString();
21. }
```

Απόσπασμα κώδικα 4.3.1.5.2.1

4.3.1.6 – Αποστολή επιλογής χρήστη

Όταν ο χρήστης επιλέξει «**Πότισε**» ή «**Μην ποτίσεις**» και πατήσει το κουμπί «**Αποστολή**» πραγματοποιείται μια διαδικασία εμφάνισης μηνυμάτων και ερωτήσεων τα οποία τα αναφέραμε στην υποενότητα 4.3.1.2. Όταν ολοκληρωθεί η διαδικασία αυτή, τότε εκτελείται το απόσπασμα κώδικα 4.3.1.6.1 [37].

```
01. public class PostDataAsyncTask extends AsyncTask<String, String, String> {
02.         @Override
03.         protected String doInBackground(String... strings) {
04.             try {
05.                 String postReceiverUrl = "https://users.iee.ihu.gr/~it134125/index.php";
06.
07.                 HttpClient httpClient = new DefaultHttpClient();
08.                 HttpPost httpPost = new HttpPost(postReceiverUrl);
09.
10.                 String sendInfoPotismaForUser;
11.
12.                 if(getChoose.equals("Πότισε"))
13.                     sendInfoPotismaForUser = "potise";
14.                 else
15.                     sendInfoPotismaForUser = "minPotiseis";
16.
17.                 List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(0);
18.                 nameValuePairs.add(new BasicNameValuePair("infoPotismaForUser", sendInfoPotismaForUser));
19.
20.                 httpPost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
21.
22.                 HttpResponse response = httpClient.execute(httpPost);
23.                 HttpEntity resEntity = response.getEntity();
```

Απόσπασμα κώδικα 4.3.1.6.1

Το παραπάνω απόσπασμα κώδικα θα δημιουργήσει ένα HttpPost σύμφωνα με την επιλογή του χρήστη και θα το αποστείλει στο url που φαίνεται στην γραμμή 5. Όταν ο server αντιληφθεί το HttpPost, τότε θα αποθηκεύσει το περιεχόμενο που ήρθε στο αρχείο «**infoPotismaForUser.txt**» [38] (απόσπασμα κώδικα (από τον server) 4.3.1.6.2). Η αποστολή του HttpPost στον server γίνεται στην γραμμή 22. Και στην γραμμή 23 παίρνουμε την κατάσταση του response έτσι, ώστε να την χρησιμοποιήσουμε, για να ελέγξουμε αν πήγαν όλα καλά με την αποστολή της επιλογής του χρήστη στον server.

```
01. <?php
02.     if($_POST)
03.         file_put_contents("infoPotismaForUser.txt", $_POST['infoPotismaForUser']);
04. ?>
```

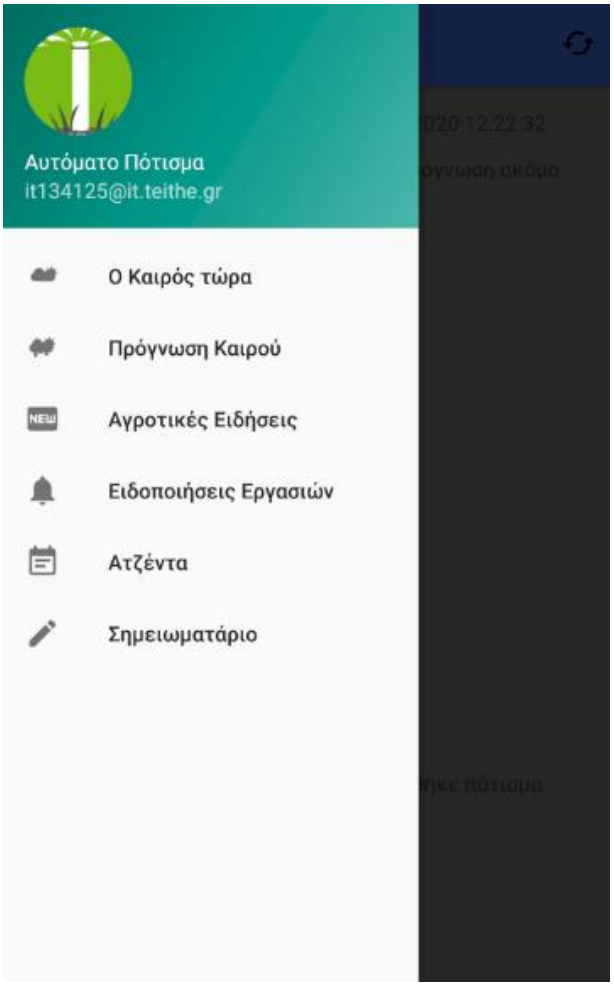
Απόσπασμα κώδικα 4.3.1.6.1

Σε περίπτωση που ο server αργήσει να απαντήσει ή προκύψει κάποιο σφάλμα κατά την διαδικασία της αποστολής αυτής στον server, θα εμφανιστεί κατάλληλο μήνυμα το οποία θα ενημερώνει τον χρήστη ότι προέκυψε κάποιο σφάλμα.

4.3.1.7 – Ανανέωση δεδομένων και μετάβαση στις άλλες υπηρεσίες της εφαρμογής

Μια ακόμα δυνατότητα που παρέχει η εφαρμογή είναι η ανανέωση των πληροφοριών τις πρόγνωσης και του ποτίσματος. Η ανανέωση αυτή πραγματοποιείτε πατώντας το κουμπί πάνω δεξιά (). Όταν πατηθεί το κουμπί αυτό, τότε τρέχει η μέθοδος «diadikasiaEpikoinonias» της οποίας τις ενέργειες τις αναλύσαμε σε προηγούμενη υποενότητα. Τέλος, για να μεταβούμε στις άλλες υπηρεσίες που παρέχει η εφαρμογή θα πατήσουμε το κουμπί πάνω αριστερά

(). Όταν πατηθεί αυτό το κουμπί εμφανίζεται το μενού που φαίνεται στην εικόνα 28. Απ' αυτό το μενού μπορεί ο χρήστης να περιηγηθεί στις υπόλοιπες υπηρεσίες που παρέχει η εφαρμογή.



Εικόνα 28 – Μενού εφαρμογής

4.3.1.8 – Ενημέρωση των δεδομένων της εφαρμογής

Μόλις εκκινήσει η εφαρμογή, δημιουργείται μια υπηρεσία η οποία είναι υπεύθυνη να ενημερώσει όλα τα αρχεία της εφαρμογής. Τα αρχεία που ενημερώνει ή δημιουργεί (σε περίπτωση που δεν υπάρχουν) είναι τα εξής: τα αρχεία της πρόγνωσης και του ποτίσματος από τον server, τα τέσσερα αρχεία από τα ειδησεογραφικά sites και στη συνέχεια όλα τα αρχεία του «Ο Καιρός τώρα» και «Πρόγνωση του καιρού». Τώρα ας αναφέρουμε μερικές λεπτομέρειες για την συγκεκριμένη υπηρεσία που δημιουργήσαμε (απόσπασμα κώδικα 4.3.1.8.1).

```
01. PersistableBundle persistableBundleEpilogiPolis = new PersistableBundle();
02. /* UpdateBackground => 0 --> runAllUpdates */
03. persistableBundleEpilogiPolis.putInt("epilogiMethodExecute", 0);
04.
05. ComponentName componentName = new ComponentName(context, UpdateBackground.class);
06. JobInfo info = new JobInfo.Builder(123, componentName)
07.     .setRequiredNetworkType(JobInfo.NETWORK_TYPE_ANY)
08.     .setPersisted(true)
09.     .setPeriodic(60 * 60 * 1000)
10.     .build();
11.
12. JobScheduler scheduler = (JobScheduler) getSystemService(JOB_SCHEDULER_SERVICE);
13. int resultCode = scheduler.schedule(info);
14. if (resultCode == JobScheduler.RESULT_SUCCESS) {
15.     Log.d(TAG, "Job scheduled");
16. } else {
17.     Log.d(TAG, "Job scheduling failed");
18. }
```

Απόσπασμα κώδικα 4.3.1.8.1

Στις πρώτες γραμμές του παραπάνω κώδικα, δημιουργούμε ένα αντικείμενο **PersistableBundle** [39] στο οποίο ορίζουμε κάποια **key values** στοιχεία. Μια εξήγηση για τα αντικείμενα «**bundle**» είναι ότι χρησιμοποιούνται, για να μεταφέρουν τιμές από κλάση σε κλάση. Εμείς, αυτό που μεταφέρουμε είναι ένα αναγνωριστικό για το ποια μέθοδος θα εκτελεστεί. Στη γραμμή 5 του παραπάνω αποσπάσματος ορίζουμε σε ποια κλάση υπάρχει ο κώδικας που θα εκτελεστεί όταν εκκινήσει η υπηρεσία. Παρακάτω, δηλώνουμε ότι για να μπορέσει να εκκινήσει η υπηρεσία θα πρέπει να υπάρχει σύνδεση στο διαδίκτυο (είτε μέσω wi-fi είτε μέσω δεδομένων κινητής τηλεφωνίας) (γραμμή 7). Επίσης, ορίζουμε ότι σε περίπτωση που γίνει επανεκκίνηση της συσκευής να ξαναδημιουργηθεί η υπηρεσία (γραμμή 8). Για να μπορέσει όμως να λειτουργήσει αυτό, θα πρέπει να δοθεί στο Android Manifest το δικαίωμα χρήσης που φαίνεται στο απόσπασμα κώδικα 4.3.1.8.2 [40][41].

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

Απόσπασμα κώδικα 4.3.1.8.2

Ακόμα, για να μπορέσει να τρέξει η υπηρεσία θα πρέπει στο Android Manifest να προστεθεί και ο κώδικας που φαίνεται παρακάτω [40].

```
01. <service
02.     android:name=".UpdateBackground"
03.     android:label="Word service"
04.     android:permission="android.permission.BIND_JOB_SERVICE" >
05. </service>
```

Απόσπασμα κώδικα 4.3.1.8.3

Όταν δημιουργηθεί η υπηρεσία και αρχίζει να εκτελείται, γίνεται μετάβαση στην κλάση UpdateBackground. Όμως, για να μπορέσουν να εκτελεστούν σωστά οι

ενέργειες της κλάσης αυτής θα πρέπει να περαστούν κάποια στοιχεία από την μία κλάση στην άλλη. Τα στοιχεία που θα χρειαστεί η κλάση UpdateBackground τα περάσαμε στο αντικείμενο PersistableBundle. Ακόμη, η υπηρεσία αυτή εκτελείται ανά μία ώρα (γραμμή 9). Τέλος, στην γραμμή 12 γίνεται η δημιουργία της υπηρεσίας. Οι βασικές μέθοδοι της κλάσης «UpdateBackground» φαίνονται στο παρακάτω απόσπασμα κώδικα [40].

```
01. public class UpdateBackground extends JobService {
02.     private JobParameters params;
03.     @Override
04.     public boolean onStartJob(JobParameters jobParameters) {
05.         params = jobParameters;
06.         if (params.getExtras().getInt("epilogiMethodExecute") == 0)
07.             runAllUpdates(jobParameters);
08.         else if (params.getExtras().getInt("epilogiMethodExecute") == 1)
09.             runSingleUpdate(jobParameters);
10.         return true;
11.     }
12.     @Override
13.     public boolean onStopJob(JobParameters jobParameters) {
14.         return true;
15.     }
```

Απόσπασμα κώδικα 4.3.1.8.4

Η κλάση «UpdateBackground» εφόσον κληρονομεί την κλάση «JobService» είναι υποχρεωμένη να υπερφορτώσει τις βασικές μεθόδους της. Οι μέθοδοι αυτοί είναι «onStartJob» και η «onStopJob». Η πρώτη εκτελείται όταν ξεκινήσει η υπηρεσία και η δεύτερη όταν σταματήσει. Στον παραπάνω κώδικα βλέπουμε ότι όταν ξεκινήσει η υπηρεσία γίνονται κάποιοι έλεγχοι για το ποια μέθοδος θα εκτελεστεί. Είτε η μέθοδος στη γραμμή 7 η οποία τρέχει, για να ενημερώσει όλα τα αρχεία της εφαρμογής είτε η μέθοδος στη γραμμή 9 η οποία εκτελείται, όταν στο activity «Ο Καιρός τώρα» ή στο «Πρόγνωση του καιρού» προστεθεί μια καινούρια πόλη ή γίνει ανανέωση πληροφοριών για μια πόλη. Το περιεχόμενο των δύο προαναφερθέντων μεθόδων δεν χρειάζεται να το παρουσιάσουμε γιατί στην ουσία είναι οι κώδικες που χρησιμοποιούν τα activities για να πάρουν πληροφορίες από το διαδίκτυο (είτε για το πότισμα είτε για τον καιρό είτε για τις ειδήσεις).

4.3.2 – Ο Καιρός τώρα

Αυτή η υπηρεσία που παρέχει η εφαρμογή έχει ως σκοπό να δώσει την δυνατότητα στον χρήστη να δει τι καιρό κάνει την συγκεκριμένη χρονική στιγμή σε μία πόλη την οποία θα την επιλέξει αυτός.

4.3.2.1 – Ενέργειες που διενεργούνται

Όταν εκτελεστεί το activity, από το αρχείο «onomataPoleon» παίρνουμε τις πόλεις οι οποίες είναι αποθηκευμένες εκεί και τις προσθέτουμε στο spinner το οποίο βρίσκεται στο πάνω μέρος της οθόνης. (απόσπασμα κώδικα 4.3.2.1.1) [33][58].

```
01.  FileInputStream fileInputStream = getApplicationContext().openFileInput(onomataPoleon);
02.  InputStreamReader inputStreamReader = new InputStreamReader(fileInputStream);
03.
04.  BufferedReader bufferedReader = new BufferedReader(inputStreamReader);
05.  while ((dataForFile = bufferedReader.readLine()) != null)
06.      listNameCityCountryCode.add(dataForFile);
07.  bufferedReader.close();
```

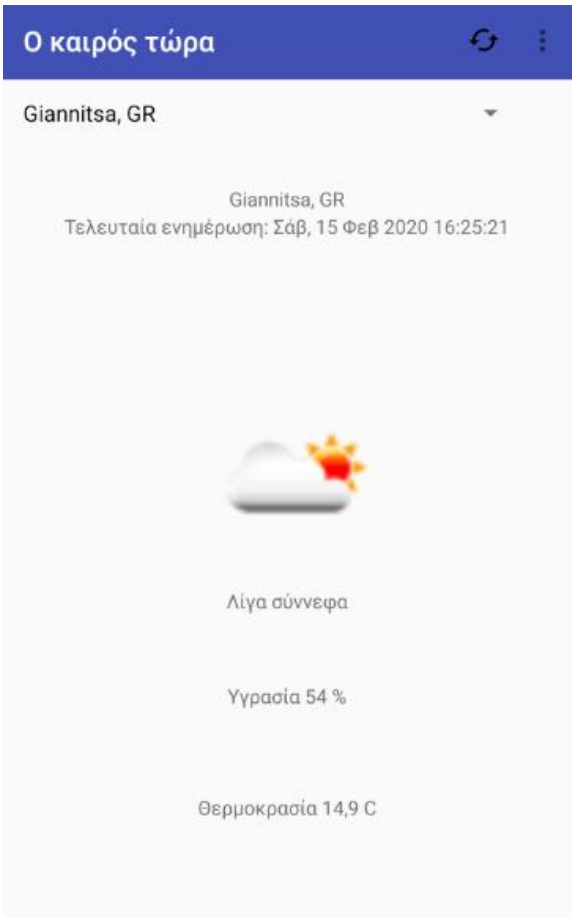
Απόσπασμα κώδικα 4.3.2.1.1

Όταν ολοκληρωθεί η εισαγωγή των πόλεων στο spinner, τότε παίρνουμε την πόλη η οποία είναι πρώτη και κάνουμε request στον server της OpenWeatherMap ο οποίος μας παρέχει πληροφορίες για τον καιρό (απόσπασμα κώδικα 4.3.2.1.2) [35].

```
01.  yourJsonStringUrl = "http://api.openweathermap.org/data/2.5/weather?q=" + epilogiPolis + "&APPID=e6523ae81ba7cbe19ebc05524dea0c78";
02.
03.  HttpHandler sh = new HttpHandler();
04.  jsonStr = sh.makeServiceCall(yourJsonStringUrl);
05.
06.  JSONObject jsonObj = new JSONObject(jsonStr);
07.  JSONObject jsonObjCodeCountry = jsonObj.getJSONObject("sys");
08.  JSONArray jsonArrWeather = jsonObj.getJSONArray("weather");
09.  JSONObject jsonObj0 = jsonArrWeather.getJSONObject(0);
10.  JSONObject jsonObjMain = jsonObj.getJSONObject("main");
11.
12.  city = jsonObj.getString("name");
13.  countryCode = jsonObjCodeCountry.getString("country");
14.  iconWeather = jsonObj0.getString("icon");
15.  igrasia = jsonObjMain.getString("humidity");
16.  thermokrasia = jsonObjMain.getString("temp");
```

Απόσπασμα κώδικα 4.3.2.1.2

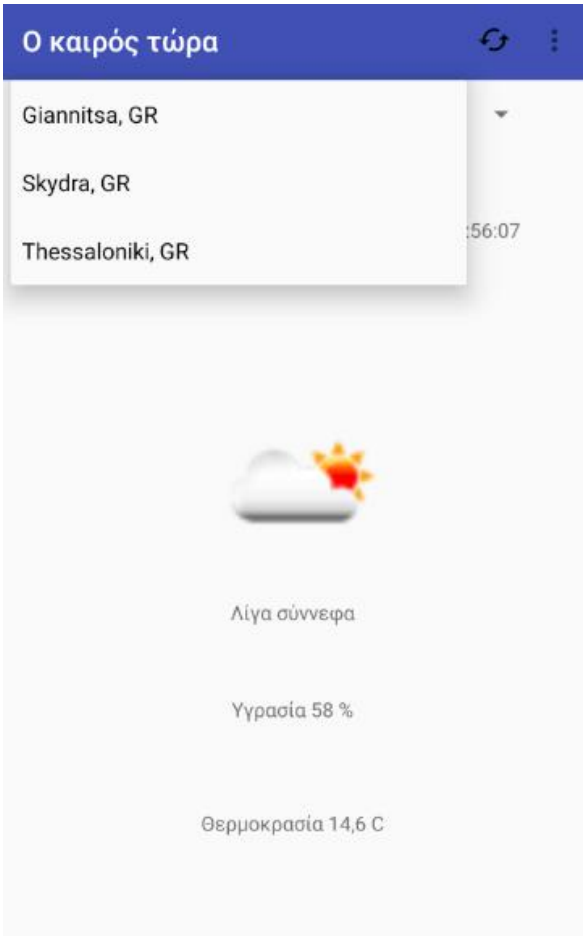
Η διαδικασία του request στον server είναι ίδια με αυτή που περιγράψαμε στην προηγούμενη ενότητα, το μόνο που αλλάζει είναι ο server, επομένως δεν υπάρχει λόγος να επαναλάβουμε την εξήγηση και σ' αυτήν την ενότητα. Όταν ολοκληρωθεί το request στον server, γίνεται ενημέρωση του UI της εφαρμογής και έχουμε το αποτέλεσμα που φαίνεται στην εικόνα 29.



Εικόνα 29 – UI ο καιρός τώρα

Επιπλέον, μαζί με την ενημέρωση του UI, αποθηκεύουμε σε αρχείο το αποτέλεσμα του request καθώς και πότε έγινε έτσι, ώστε όταν δεν είναι συνδεδεμένη η συσκευή στο διαδίκτυο να μπορεί ο χρήστης να το δει.

Στη συνέχεια, όταν πατήσουμε πάνω στο spinner θα μας εμφανίσει την λίστα με τις πόλεις που έχει αποθηκευμένες (εικόνα 30).



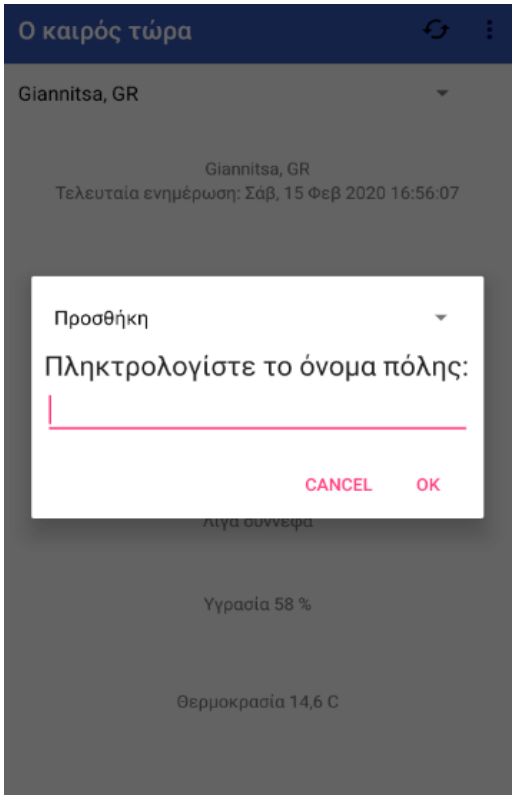
Εικόνα 30 – Λίστα του spinner

Επιλέγοντας μία πόλη ο χρήστης, γίνεται η διαδικασία που αναφέραμε στην αρχή της υποενότητας αυτής.

4.3.2.2 – Ανανέωση και ρυθμίσεις πόλης

Ο χρήστης, πατώντας το κουμπί της ανανέωσης (πάνω δεξιά δίπλα από τις τρεις κάθετες τελίτσες) ξανακάνει request στον server, για να του στείλει ξανά πληροφορίες για τον καιρό στην επιλεγμένη πόλη. Επίσης, έχει τη δυνατότητα να εισάγει το όνομα της πόλης της οποίας θέλει να δει τον καιρό από την επιλογή «Ρυθμίσεις πόλης» εικόνα 31.

Εικόνα 31 – Ρυθμίσεις πόλης



Εικόνα 32 – Layout inflater

Όταν πατήσει πάνω στην επιλογή «Ρυθμίσεις πόλης» θα εμφανιστεί το layout inflater που φαίνεται στην εικόνα 32.

Στο σημείο που αναβοσβήνει ο κέρσορας εκεί ο χρήστης μπορεί να εισάγει το όνομα πόλης για την οποία θέλει να μάθει πληροφορίες για τον καιρό και να πατήσει το κουμπί «OK» (απόσπασμα κώδικα 4.3.2.2.1). Η διαδικασία που ακολουθείται για να σταλθεί request με την πόλη που εισήγαγε ο χρήστης είναι ίδια με την διαδικασία που λαμβάνει χώρα όταν ανοίγει το activity. Επίσης, όταν ολοκληρωθεί το request και πάνε όλα καλά, θα εμφανιστεί ένα μήνυμα το οποίο θα ενημερώνει τον χρήστη ότι η «τάδε» πόλη προστέθηκε με επιτυχία και θα φορτώσει τις πληροφορίες της νέας πόλης στο UI. Σε περίπτωση που ο χρήστης εισάγει μια πόλη που δεν είναι καταχωρημένη στην βάση δεδομένων του openWeatherMap ή έχει κάποιο ορθογραφικό λάθος εμφανίζεται κατάλληλο μήνυμα που τον ενημερώνει. Επίσης, στην περίπτωση που ο χρήστης εισάγει μια πόλη η οποία υπάρχει ήδη, θα του εμφανιστεί αντίστοιχο μήνυμα [42].

```
01. for (String cityCountryCodeAdd: listNameCityCountryCode) {
02.     if (onomaPolisUserInput.equalsIgnoreCase(cityCountryCodeAdd.substring(
03.             0, cityCountryCodeAdd.indexOf(','))) {
04.         flagCityCountryCodeAdd = false;
05.         break;
06.     } else {
07.         flagCityCountryCodeAdd = true;
08.     }
09. }
10. if (flagCityCountryCodeAdd)
11.     diadikasiaEpikoinonias_AllagiOnomatos();
12. else
13.     Toast.makeText(CurrentWeatherActivity.this, "Η πόλη " + onomaPolisUserInput + " υπάρχει ήδη", Toast.LENGTH_LONG).show();
```

Απόσπασμα κώδικα 4.3.2.2.1

Ακόμη, μέσα από το layout inflater έχει την δυνατότητα ο χρήστης να διαγράψει μία πόλη. Για να εμφανιστεί το μενού της διαγραφής, πατάμε πάνω στο spinner της εικόνας 32 και επιλέγουμε «Διαγραφή». Τότε θα εμφανιστεί ένα layout inflater το οποίο θα έχει ένα spinner που θα περιέχει τις διαθέσιμες πόλεις. Μια σημείωση: στην λίστα των πόλεων, αν υπάρχει μόνο μία πόλη, δεν πραγματοποιείται η διαγραφή και εμφανίζεται ενημερωτικό μήνυμα στον χρήστη. Όταν οι προϋποθέσεις είναι ευνοϊκές και πραγματοποιηθεί η διαγραφή, εκτός από την λίστα των πόλεων, διαγράφεται και το αρχείο που περιέχει πληροφορίες για την συγκεκριμένη πόλη, καθώς και το αρχείο της ίδιας πόλης στο activity της πρόγνωσης του καιρού το οποίο θα το αναλύσουμε στην επόμενη υπό ενότητα (απόσπασμα κώδικα 4.3.2.2.2) [33][42][58].

```
if (spinnerEpilogiPolisGiaDiagrafi.getAdapter().getCount() == 1) {
    Toast.makeText(CurrentWeatherActivity.this, "Δεν μπορεί να πραγματοποιηθεί η διαγραφή γιατί υπάρχει μόνο μία πόλη", Toast.LENGTH_LONG).show();
    alertDialogEpilogiEnergeias.dismiss();
}
else {
    File inputFile = new File(getFilesDir(), onomataPoleon);
    File tempFile = new File(getFilesDir(), "tempFile");

    listNameCityCountryCode.clear();

    try {

        BufferedReader reader = new BufferedReader(new FileReader(inputFile));
        BufferedWriter writer = new BufferedWriter(new FileWriter(tempFile));

        String lineToRemove = onomaPolis[0];
        String currentLine;
        String trimmedLine;

        while ((currentLine = reader.readLine()) != null) {
            trimmedLine = currentLine.trim();
            if (trimmedLine.equals(lineToRemove))
                continue;
            writer.write(currentLine + System.getProperty("line.separator"));
            listNameCityCountryCode.add(currentLine);
        }
        writer.close();
        reader.close();
        tempFile.renameTo(inputFile);

        file = new File(getFilesDir(), lineToRemove.substring(
            0, lineToRemove.indexOf(',')));
        file.delete();
```

Απόσπασμα κώδικα 4.3.2.2.2

Τέλος, όταν ο χρήστης πατάει το κουμπί της ανανέωσης ή προσθέσει μια καινούρια πόλη, πυροδοτείται ένα job service το οποίο είναι υπεύθυνο είτε να ανανεώσει το αντίστοιχο αρχείο πόλης της πρόγνωσης του καιρού είτε να δημιουργήσει το αρχείο σε περίπτωση που δεν υπάρχει (απόσπασμα κώδικα 4.3.2.2.3) [39][40][41].

```
PersistableBundle persistableBundleEpilogiPolis = new PersistableBundle();
persistableBundleEpilogiPolis.putString("epilogiPolis", epilogiPolis);
/* UpdateBackground => 11 --> ParseDataForecastWeather */
persistableBundleEpilogiPolis.putInt("epilogiClassExecute", 11);
/* UpdateBackground => 1 --> runSingleUpdate */
persistableBundleEpilogiPolis.putInt("epilogiMethodExecute", 1);

ComponentName componentName = new ComponentName(context, UpdateBackground.class);
JobInfo info = new JobInfo.Builder(01234, componentName)
    .setRequiredNetworkType(JobInfo.NETWORK_TYPE_ANY)
    .setPersisted(true)
    .build();

JobScheduler scheduler = (JobScheduler) getSystemService(JOB_SCHEDULER_SERVICE);
int resultCode = scheduler.schedule(info);
if (resultCode == JobScheduler.RESULT_SUCCESS) {
    Log.d(TAG, "Job scheduled");
} else {
    Log.d(TAG, "Job scheduling failed");
}
```

Απόσπασμα κώδικα 4.3.2.2.3

Στις πρώτες γραμμές του παραπάνω κώδικα, δημιουργούμε ένα αντικείμενο **PersistableBundle** στο οποίο περνάμε το όνομα της πόλης για την οποία θέλουμε να κάνουμε το request, ένα αναγνωριστικό για το ποια κλάση («Ο Καιρός τώρα» ή «Πρόγνωση του καιρού», στον παραπάνω κώδικα «Πρόγνωση του καιρού») θα εκτελεστεί και αν εκτελεστεί η μέθοδος που ενημερώνει ή δημιουργεί μόνο το αρχείο της πρόγνωσης του καιρού ή η μέθοδος που ανανεώνει όλα τα αρχεία. Στον παραπάνω κώδικα θα εκτελεστεί μόνο η μέθοδος που ανανεώνει ή δημιουργεί το αρχείο της πρόγνωσης του καιρού. Επίσης, παρακάτω ορίζουμε ότι για να εκτελεστεί η μέθοδος, χρειάζεται σύνδεση στο διαδίκτυο και σε περίπτωση

που επανεκκινηθεί η συσκευή, η υπηρεσία να εκκινήσει ξανά. Τέλος, στην γραμμή 14 γίνεται η δημιουργία της υπηρεσίας.

Να επισημάνουμε σ' αυτό το σημείο ότι σε όποιες λειτουργίες χρειάζεται σύνδεση στο διαδίκτυο, υπάρχει έλεγχος και εμφάνιση μηνύματος σε περίπτωση που δεν υπάρχει σύνδεση. Απλά δεν κρίναμε απαραίτητο να το αναφέρουμε στα σημεία που χρειάζεται ή να προσθέσουμε τα αντίστοιχα ενημερωτικά μηνύματα, γιατί έχουμε δείξει παρόμοια στη προηγούμενη ενότητα.

4.3.3 – Πρόγνωση καιρού

Σε αυτήν την υπηρεσία που παρέχει η εφαρμογή, ο χρήστης μπορεί να εισάγει το όνομα της πόλης για την οποία επιθυμεί να μάθει πληροφορίες για την πρόγνωση του καιρού. Η υπηρεσία «Πρόγνωση καιρού» περιέχει δύο activities. Το ένα είναι «Ημερήσια πρόγνωση του καιρού» και το άλλο είναι «Καθημερινή πρόγνωση του καιρού».

4.3.3.1 – Ημερήσια πρόγνωση καιρού

Στην ημερήσια πρόγνωση του καιρού, εμφανίζουμε πληροφορίες για τον καιρό ανά τρεις ώρες για δέκα στιγμιότυπα. Αυτά τα δέκα στιγμιότυπα τα περνάμε σε ένα scrollView στο κάτω μέρος της οθόνης (εικόνα 33).

Εικόνα 33 – UI ημερήσιας πρόγνωσης καιρού

Τώρα, όσο για τις ενέργειες που λαμβάνουν χώρα όταν ανοίγει το activity, όπως η διαδικασία που εκτελείται, για να γεμίσει το spinner (στο πάνω μέρος της οθόνης) με τις πόλεις που είναι αποθηκευμένες, τι γίνεται όταν ο χρήστης πατήσει το κουμπί της ανανέωσης (πάνω δεξιά δίπλα από τις τρεις τελίτσες ) καθώς και τι εμφανίζεται όταν πατηθεί το κουμπί «Ρυθμίσεις πόλης» και τις υποεπιλογές του όπως για παράδειγμα την προσθήκη και την διαγραφή πόλης κτλ, δεν θα τις αναλύσουμε γιατί η ανάλυση τους έγινε εκτενέστερα στην προηγούμενη ενότητα. Αυτό που θα αναλύσουμε σε αυτήν την ενότητα είναι τις διαδικασίες που γίνονται, για να παρουσιαστούν τα δεδομένα της ημερήσιας πρόγνωσης στο UI της εφαρμογής. Όπως αναφέραμε και στην προηγούμενη ενότητα, για να αντλήσουμε πληροφορίες για τα δεδομένα καιρού παίρνουμε την επιλεγμένη πόλη από το spinner και κάνουμε request στον server της OpenWeatherMap. Αλλά υπάρχει μια μικρή διαφορά σε σχέση με το request που γίνεται στην προηγούμενη ενότητα. Στην προηγούμενη ενότητα στο link του request πριν το σημείο που τοποθετείται η πόλη, γράφει «weather» ενώ σε αυτήν την ενότητα γράφει «forecast». Η διαφορά υπάρχει γιατί στο πρώτο μας παρέχονται πληροφορίες για τον καιρό την συγκεκριμένη χρονική στιγμή ενώ στο δεύτερο μας παρέχεται πρόγνωση του καιρού. Όταν τοποθετηθεί η πόλη στο url πραγματοποιείται το request (απόσπασμα κώδικα 4.3.3.1.1) [35]. Δεν θα αναφέρουμε περισσότερες λεπτομέρειες για το πώς πραγματοποιείται το request γιατί το αναλύσαμε σε προηγούμενη ενότητα.

```
01. yourJsonStringUrl = "http://api.openweathermap.org/data/2.5/forecast?q=" + epilogiPolis + "&APPID=e6523ae81ba7cbe19ebc05524dea0c78";
02.
03. HttpHandler sh = new HttpHandler();
04. jsonStr = sh.makeServiceCall(yourJsonStringUrl);
05.
06. JSONObject jsonObj = new JSONObject(jsonStr);
07. JSONObject jsonObjCity = jsonObj.getJSONObject("city");
08. city = jsonObjCity.getString("name");
09. countryCode = jsonObjCity.getString("country");
10.
11. for (int i = 0; i <= 10; i++) {
12.     jsonArrList = jsonObj.getJSONArray("list");
13.     jsonObj0 = jsonArrList.getJSONObject(i);
14.     jsonObjMain0 = jsonObj0.getJSONObject("main");
15.     jsonArrWeather0 = jsonObj0.getJSONArray("weather");
16.     jsonObjWeather0 = jsonArrWeather0.getJSONObject(0);
17.
18.     dateText = jsonObj0.getString("dt_txt");
19.
20.     listIconWeather.add(jsonObjWeather0.getString("icon"));
21.     listIgrasia.add(jsonObjMain0.getString("humidity"));
22.     listThermokrasia.add(jsonObjMain0.getString("temp"));
23.     listDateDay.add(dateText.substring(8, 10));
24.     listDateMonth.add(dateText.substring(5, 7));
25.     listDateHour.add(dateText.substring(11, 13));
```

Απόσπασμα κώδικα 4.3.3.1.1

Αυτό που θα σχολιάσουμε στην συγκεκριμένη υποενότητα είναι ο τρόπος επεξεργασίας των δεδομένων που λαμβάνουμε μετά την ολοκλήρωση του request. Στον κώδικα παραπάνω βλέπουμε ότι υπάρχει ένα for το οποίο τρέχει έντεκα φορές. Αυτές τις έντεκα φορές που τρέχει, γεμίζει τις λίστες: του εικονιδίου του καιρού, της υγρασίας, της θερμοκρασίας, καθώς και της μέρας, του μήνα και της ώρας κάθε πρόγνωσης. Θα αναρωτιέται κανείς γιατί το for να τρέξει έντεκα φορές αφού στην αρχή της ενότητας λέμε ότι έχουμε δέκα στιγμιότυπα στο scrollView. Η

απάντηση είναι, γιατί εκτός του ότι γεμίζουμε το scrollView με δέκα στιγμιότυπα έχουμε να γεμίσουμε και τις πληροφορίες του UI πάνω απ' αυτό.

4.3.3.2 – Καθημερινή πρόγνωση καιρού

Για να ανοίξουμε το activity της καθημερινής πρόγνωσης του καιρού, θα πρέπει από το activity της ημερήσιας προβολής του καιρού να πατήσουμε στις τρεις τελίτσες πάνω δεξιά και από το μενού που θα μας εμφανίσει (εικόνα 34) να πατήσουμε το «Καθημερινά».

Εικόνα 34 – Μενού μετάβασης

Όταν ανοίξει το activity της καθημερινής πρόγνωσης του καιρού θα έχουμε το αποτέλεσμα που φαίνεται στην εικόνα 35. Η ιστοσελίδα OpenWeatherMap από την οποία αντλούμε τις πληροφορίες που χρειαζόμαστε για τον καιρό, στην free έκδοση την οποία χρησιμοποιούμε μας παρέχει δεδομένα πρόγνωσης καιρού μέχρι πέντε μέρες. Έτσι, σε αυτό το activity στο scrollView έχουμε πέντε στιγμιότυπα αντί για δέκα που είχαμε στην προηγούμενη υποενότητα.

Εικόνα 35 – UI καθημερινής πρόγνωσης καιρού

Οι ενέργειες που μπορεί να κάνει ο χρήστης σε αυτό το activity είναι ίδιες με τις ενέργειες που μπορεί να κάνει και στο activity της προηγούμενης υποενότητας. Δηλαδή, έχει την δυνατότητα να επιλέξει από το spinner για ποια πόλη θέλει να δει την πρόγνωση του καιρού, να κάνει ανανέωση των δεδομένων πρόγνωσης καιρού και αν δεν υπάρχει η πόλη για την οποία επιθυμεί να μάθει πληροφορίες για την πρόγνωση του καιρού σε αυτήν, έχει τη δυνατότητα να την προσθέσει από τις ρυθμίσεις πόλης. Επομένως, εφόσον τις παραπάνω ενέργειες τις εξηγήσαμε σε προηγούμενες ενότητες δεν θα επεκταθούμε περισσότερο σε αυτές. Αυτό που θα εξηγήσουμε σε αυτήν την υποενότητα είναι οι διαδικασίες που λαμβάνουν χώρα, για να παρουσιαστούν τα δεδομένα της καθημερινής πρόγνωσης του καιρού σε αυτό το activity (απόσπασμα κώδικα 4.3.3.2.1) [35].

```
01.  yourJsonStringUrl = "http://api.openweathermap.org/data/2.5/forecast?q=" + epilogiPolis + "&APPID=e6523ae81ba7cbe19ebc05524dea0c78";
02.
03.  HttpHandler sh = new HttpHandler();
04.  jsonStr = sh.makeServiceCall(yourJsonStringUrl);
05.
06.  JSONObject jsonObj = new JSONObject(jsonStr);
07.  JSONObject jsonObjCity = jsonObj.getJSONObject("city");
08.  city = jsonObjCity.getString("name");
09.  countryCode = jsonObjCity.getString("country");
10.
11.  int x = 0;
12.  for (int i = 0; i <= 5; i++) {
13.      jsonArrList = jsonObj.getJSONArray("list");
14.      if (x > 39)
15.          x = 39;
16.
17.      jsonObj0 = jsonArrList.getJSONObject(x);
18.      jsonObjMain0 = jsonObj0.getJSONObject("main");
19.      jsonArrWeather0 = jsonObj0.getJSONArray("weather");
20.      jsonObjWeather0 = jsonArrWeather0.getJSONObject(0);
21.
22.      dataText = jsonObj0.getString("dt_txt");
23.
24.      listIconWeather.add(jsonObjWeather0.getString("icon"));
25.      listIgrasia.add(jsonObjMain0.getString("humidity"));
26.      listThermokrasia.add(jsonObjMain0.getString("temp"));
27.      listDateDay.add(dataText.substring(8, 10));
28.      listDateMonth.add(dataText.substring(5, 7));
29.
30.      if (x == 0) {
31.          dateHour = dataText.substring(11, 13);
32.
33.          if (dateHour.equals("06"))
34.              x += 2;
35.          else if (dateHour.equals("09"))
36.              x += 1;
37.          else if (dateHour.equals("12"))
38.              x += 0;
39.          else if (dateHour.equals("15"))
40.              x += -1;
41.          else if (dateHour.equals("18"))
42.              x += -2;
43.          else if (dateHour.equals("21"))
44.              x += -3;
45.          else if (dateHour.equals("00"))
46.              x += -4;
47.          else if (dateHour.equals("03"))
48.              x += -5;
49.      }
50.      x += 8;
51.  }
```

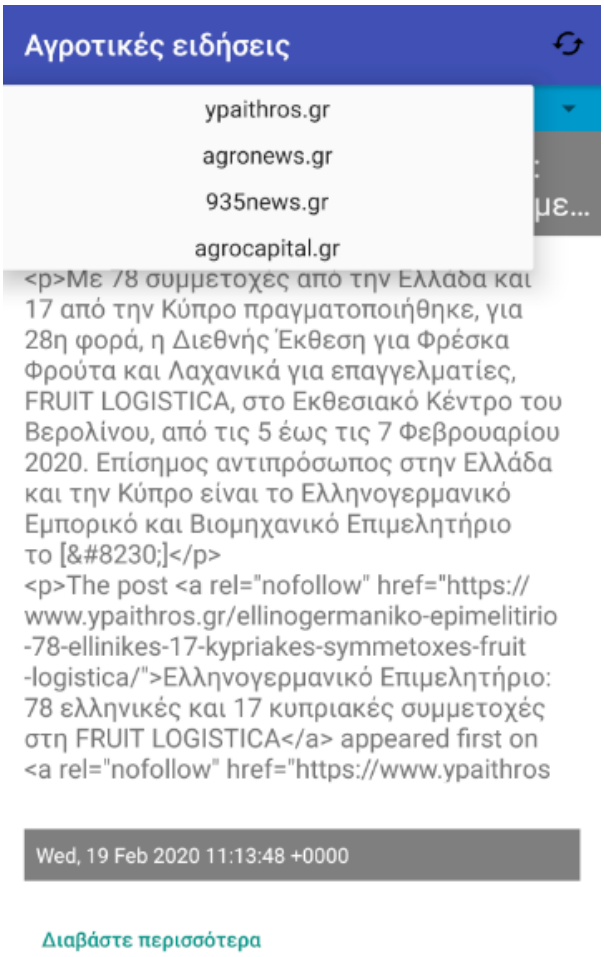
Απόσπασμα κώδικα 4.3.3.2.1

Λοιπόν, εφόσον ολοκληρωθεί το request στον server της OpenWeatherMap με την επιλεγμένη πόλη, το αποτέλεσμα που δεχτήκαμε θα αρχίσουμε να το επεξεργαζόμαστε έτσι, ώστε να το ετοιμάσουμε, για να το εμφανίσουμε στο UI του activity. Να σημειωθεί σε αυτό το σημείο ότι από το αρχείο της πρόγνωσης του καιρού για τις επόμενες μέρες, επιλέξαμε να εμφανίσουμε σαν στιγμιότυπο την πρόγνωση του καιρού στις δώδεκα η ώρα το μεσημέρι, επειδή θεωρούμε ότι είναι η ώρα που βρίσκεται στη μέση του εικοσιτετράωρου. Συνεχίζοντας, όπως είπαμε και παραπάνω, εφόσον η OpenWeatherMap στην free έκδοση της, μας επιτρέπει να δούμε πληροφορίες για την πρόγνωση του καιρού μέχρι πέντε μέρες γι' αυτό το for θα τρέξει έξι φορές (η σημερινή μέρα συν τις επόμενες πέντε). Την μεταβλητή «x» την χρησιμοποιούμε, για να περιηγηθούμε στα αποτελέσματα που μας ήρθαν μετά την ολοκλήρωση του request. Όταν η μεταβλητή έχει την τιμή μηδέν (πρώτο αποτέλεσμα) μπαίνει στο «if» της γραμμής 30. Αυτό γίνεται, γιατί σε περίπτωση που ο χρήστης κάνει request οποιαδήποτε ώρα εκτός τις δώδεκα το μεσημέρι, να ορίσουμε κατάλληλα το «x» έτσι, ώστε το επόμενο βήμα επανάληψης της for να πέσει στις δώδεκα η ώρα το μεσημέρι της επόμενης μέρας. Στο πρώτο βήμα επανάληψης της for θα γίνει κανονικά η ετοιμασία της εμφάνισης του αποτελέσματος στο UI πάνω από το scrollView. Βλέπουμε παρακάτω (γραμμή 50) ότι την «x» την αυξάνουμε συν οκτώ. Αυτό το κάνουμε επειδή από τις δώδεκα το

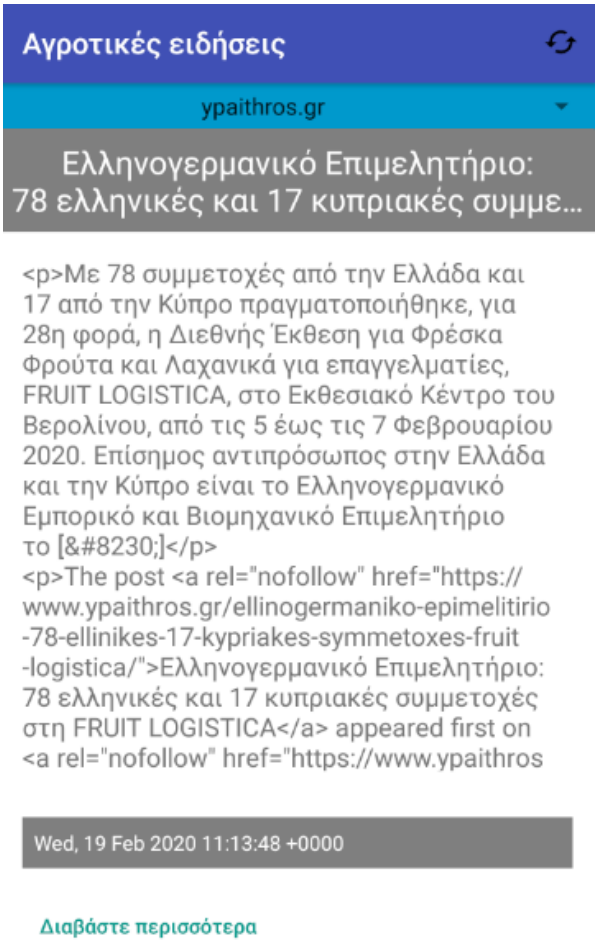
μεσημέρι της σημερινής μέρας μέχρι τις δώδεκα το μεσημέρι της αυριανής μέρας, μας χωρίζουν οκτώ στιγμιότυπα πρόγνωσης καιρού. Τέλος, το «if» της 14 γραμμής το βάλαμε για όταν η «x» ξεπεράσει το 39 να την κάνουμε ίση με 39 γιατί το τελευταίο στιγμιότυπο που μας παρέχει το αρχείο πρόγνωσης του καιρού είναι μέχρι τόσο.

4.3.4 – Αγροτικές ειδήσεις

Σε αυτό το activity ο χρήστης μπορεί να ενημερώνετε με τα τελευταία αγροτικά νέα. Η εφαρμογή του παρέχει την δυνατότητα να ενημερώνεται από τέσσερα διαφορετικά ενημερωτικά sites (εικόνα 36). Για να αντλήσουμε τα άρθρα απ' αυτά τα sites χρησιμοποιούμε την τεχνολογία RSS [21]. Αυτό συνεπάγεται ότι τα sites παρέχουν την συγκεκριμένη τεχνολογία. Όταν ολοκληρωθεί το request στην επιλεγμένη ιστοσελίδα και πάνε όλα καλά, θα έχουμε το αποτέλεσμα που φαίνεται στην εικόνα 37.



Εικόνα 36 – Ειδησεογραφικά sites



Εικόνα 37 – UI αγροτικές ειδήσεις

Σε αυτό το activity για την εμφάνιση των άρθρων συνεργάζονται άλλες δύο τεχνολογίες. Η card view και η recycler view [22] τις οποίες τις αναλύσαμε στο προηγούμενο κεφάλαιο.

4.3.4.1 – Request στον server

Όταν επιλεχθεί από τον χρήστη το site από το οποίο επιθυμεί να δει αγροτικές ειδήσεις, θα ελέγξουμε την πρόσβαση στο διαδίκτυο και αν υπάρχει πρόσβαση θα προετοιμάσουμε τις απαραίτητες διαδικασίες για την εκτέλεση του request (απόσπασμα κώδικα 4.3.4.1.1) [43].

```
01. try {
02.     DefaultHttpClient httpClient = new DefaultHttpClient();
03.     HttpGet httpGet = new HttpGet(url);
04.
05.     HttpResponse httpResponse = httpClient.execute(httpGet);
06.     HttpEntity httpEntity = httpResponse.getEntity();
07.     xml = EntityUtils.toString(httpEntity);
08.
09. } catch (UnknownHostException e) {
10.     Snackbar.make(relativeLayout, "Ο διακομιστής άργησε να απαντήσει", Snackbar.LENGTH_LONG)
11.         .setAction("Action", null).show();
12. }
```

Απόσπασμα κώδικα 4.3.4.1.1

Η διεύθυνση στην οποία θα πραγματοποιήσουμε το request είναι αποθηκευμένη στη μεταβλητή «url» και αν δεν προκύψει κάποιο σφάλμα το οποίο θα οδηγήσει σε εξαίρεση το αποτέλεσμα του request θα το πάρουμε στην μεταβλητή «xml». Αν αργήσει να απαντήσει ο server στον οποίο κάναμε το request θα προκύψει εξαίρεση και θα εμφανιστεί το κατάλληλο μήνυμα (γραμμή 10-11). Σε περίπτωση που δεν υπάρχει πρόσβαση στο διαδίκτυο, θα γίνει έλεγχος για το άμα υπάρχει αποθηκευμένο αρχείο με το επιλεγμένο site. Αν δεν υπάρχει, θα εμφανιστεί κατάλληλο μήνυμα. Αν υπάρχει, θα φορτώσει τις πληροφορίες μέσα από το αρχείο και θα ενημερώσει τον χρήστη ότι οι πληροφορίες πιθανόν δεν είναι σωστές.

4.3.4.2 – Parse δεδομένων

Εφόσον ολοκληρωθούν οι ενέργειες που περιγράψαμε στην προηγούμενη υποενότητα, τώρα είμαστε έτοιμοι να αναλύσουμε το αποτέλεσμα που προέκυψε (απόσπασμα κώδικα 4.3.4.2.1) [43].

```
01. try {
02.     Document doc = this.getDomElement(rss_feed_xml);
03.     NodeList nodeList = doc.getElementsByTagName(TAG_CHANNEL);
04.     Element e = (Element) nodeList.item(0);
05.
06.     NodeList items = e.getElementsByTagName(TAG_ITEM);
07.     for (int i = 0; i < items.getLength(); i++) {
08.         Element e1 = (Element) items.item(i);
09.
10.         String title = this.getValue(e1, TAG_TITLE);
11.         String link = this.getValue(e1, TAG_LINK);
12.         String description = this.getValue(e1, TAG_DESRIPTION);
13.         String pubdate = this.getValue(e1, TAG_PUB_DATE);
14.         String guid = this.getValue(e1, TAG_GUID);
15.
16.         RSSItem rssItem = new RSSItem(title, link, description, pubdate, guid);
17.         itemsList.add(rssItem);
18.     }
19. } catch (Exception e) {
20.     e.printStackTrace();
21. }
```

Απόσπασμα κώδικα 4.3.4.2.1

Για να μπορέσουμε να αποκτήσουμε πρόσβαση στο xml έγγραφο το οποίο παραλάβαμε από τον server πρέπει να χρησιμοποιήσουμε το **Document interface** επειδή εννοιολογικά είναι η ρίζα του δέντρου document και παρέχει την κύρια πρόσβαση στο έγγραφο (απόσπασμα κώδικα 4.3.4.2.1 γραμμή 2). Στη γραμμή 2 του παραπάνω κώδικα καλείται η μέθοδος «getDomElement» με βάση την οποία θα χρησιμοποιήσουμε το API του Document, για να διαμορφώσουμε κατάλληλα το xml που μας ήρθε, έτσι ώστε να μπορούμε να κάνουμε parse το περιεχόμενο του. Ο κώδικας της μεθόδου getDomElement φαίνεται παρακάτω [43].

```
01. Document doc = null;
02. DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
03. try {
04.     DocumentBuilder db = dbf.newDocumentBuilder();
05.     InputSource is = new InputSource();
06.     is.setCharacterStream(new StringReader(xml));
07.     doc = db.parse(is);
08. } catch (ParserConfigurationException e) {
09.     Log.e("Error: ", e.getMessage());
10.     return null;
11. } catch (SAXException e) {
12.     Log.e("Error: ", e.getMessage());
13.     return null;
14. } catch (IOException e) {
15.     Log.e("Error: ", e.getMessage());
16.     return null;
17. }
```

Απόσπασμα κώδικα 4.3.4.2.2

Όταν ολοκληρωθεί η εκτέλεση του αποσπάσματος κώδικα 4.3.4.2.2, συνεχίζεται η εκτέλεση του αποσπάσματος κώδικα 4.3.4.2.1. Στη συνέχεια του αποσπάσματος αυτού, παίρνουμε τα elements του xml με βάση το όνομα τους και προσπαθούμε να εισχωρήσουμε μέσα στο καθένα, για να πάρουμε το κείμενο που περιέχει (αποσπάσματα κώδικα 4.3.4.2.3, 4.3.4.2.4) [43] το οποίο θα το τοποθετήσουμε στις αντίστοιχες μεταβλητές (απόσπασμα κώδικα 4.3.4.2.1 γραμμές 10-14) έτσι ώστε να δημιουργήσουμε αντικείμενα RSSItem (απόσπασμα κώδικα 4.3.4.2.1 γραμμή 16) τα οποία θα τα τοποθετήσουμε σε λίστα (απόσπασμα κώδικα 4.3.4.2.1 γραμμή 17).

```
01. public String getValue(Element item, String str) {
02.     NodeList n = item.getElementsByTagName(str);
03.     return this.getElementValue(n.item(0));
04. }
```

Απόσπασμα κώδικα 4.3.4.2.3

```
01. public final String getElementValue(Node elem) {
02.     Node child;
03.     if (elem != null) {
04.         if (elem.hasChildNodes()) {
05.             for (child = elem.getFirstChild(); child != null; child = child
06.                 .getNextSibling()) {
07.                 if (child.getNodeType() == Node.TEXT_NODE || (child.getNodeType() == Node.CDATA_SECTION_NODE)) {
08.                     return child.getNodeValue();
09.                 }
10.             }
11.         }
12.     }
13.     return "";
14. }
```

Απόσπασμα κώδικα 4.3.4.2.4

Όταν ολοκληρωθεί η τοποθέτηση των αντικειμένων στην λίστα, τότε την κάνουμε return.

4.3.4.3 – Εμφάνιση δεδομένων

Για να μπορέσουμε να εμφανίσουμε τα δεδομένα από την λίστα η οποία επιστράφηκε στην προηγούμενη υποενότητα, θα την στείλουμε στην κλάση «RSSFeedAdapter». Για να γίνει αυτό θα πρέπει να δημιουργήσουμε έναν **adapter** στον οποίο θα τοποθετήσουμε την λίστα των αντικειμένων που επιστράφηκε. Έπειτα, θα κάνουμε κάποιες αρχικοποιήσεις για το recyclerView και θα ορίσουμε στον adapter του συγκεκριμένου recyclerView τον adapter που δημιουργήσαμε στην γραμμή 1 (απόσπασμα κώδικα 4.3.4.3.1) [44].

```
01. RSSFeedAdapter rssFeedAdapter = new RSSFeedAdapter((ArrayList <RSSItem>) rssItems, getApplicationContext());
02. recyclerView.setLayoutManager(new LinearLayoutManager(getApplicationContext()));
03. recyclerView.setAdapter(rssFeedAdapter);
```

Απόσπασμα κώδικα 4.3.4.3.1

Η κλάση RSSFeedAdapter περιέχει μια υποκλάση, την «MyViewHolder». Αυτή η υποκλάση κληρονομεί την «RecyclerView.ViewHolder». Έργο της είναι να αρχικοποιήσει τα πεδία τα οποία θα χρησιμοποιηθούν για την δημιουργία ενός view (όταν λέμε «view» εννοούμε άρθρο το οποίο περιέχει τίτλο, περιγραφή, ημερομηνία, κτλ) (απόσπασμα κώδικα 4.3.4.3.2) [44].

```
01. public MyViewHolder(View itemView) {
02.     super(itemView);
03.
04.     Title = (TextView) itemView.findViewById(R.id.title_text);
05.     Description = (TextView) itemView.findViewById(R.id.description_text);
06.     Date = (TextView) itemView.findViewById(R.id.date_text);
07.     BDiavasePerissotera = itemView.findViewById(R.id.bDiavasePerissotera);
08.     cardView = (CardView) itemView.findViewById(R.id.cardview);
09. }
```

Απόσπασμα κώδικα 4.3.4.3.2

Επίσης, κληρονομεί την «RecyclerView.Adapter», αυτό συνεπάγεται το γεγονός ότι πρέπει να υλοποιήσει τις βασικές μεθόδους της κλάσης που κληρονόμησε. Οι μέθοδοι αυτοί είναι: **onCreateViewHolder**, **onBindViewHolder**, **getItemCount**, τις οποίες τις εξηγήσαμε στο προηγούμενο κεφάλαιο.

Εφόσον ορίσαμε παραπάνω τον adapter, θα τρέξει η μέθοδος «onCreateViewHolder» (απόσπασμα κώδικα 4.3.4.3.3) [44].

```
01. @NonNull
02. @Override
03. public MyViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
04. 
05.     View myView = LayoutInflater.from(context).inflate(R.layout.rss_feed_row_news_items, parent, false);
06.     MyViewHolder viewHolder = new MyViewHolder(myView);
07. 
08.     return viewHolder;
09. }
```

Απόσπασμα κώδικα 4.3.4.3.3

Αυτή η μέθοδος αρχικά με βάση το layout θα δημιουργήσει τον σκελετό ενός view (γραμμή 5) και έπειτα θα δημιουργήσει ένα αντικείμενο τύπου «MyViewHolder» στο οποίο θα περάσει ως παράμετρο το view (γραμμή 6). Στη συνέχεια, με βάση αυτό το view θα πραγματοποιηθούν οι αρχικοποιήσεις των πεδίων του layout (τίτλος, περιγραφή, ημερομηνία, κτλ). Τέλος, θα χρησιμοποιηθεί η μέθοδος «onBindViewHolder», η οποία είναι υπεύθυνη να προσθέσει περιεχόμενο στον σκελετό view που δημιουργήσαμε προηγουμένως και κατ' επέκταση να εμφανίσει αυτό το περιεχόμενο στο UI της εφαρμογής (απόσπασμα κώδικα 4.3.4.3.4) [44][45].

```
01. @Override
02. public void onBindViewHolder(@NonNull MyViewHolder holder, int position) {
03.     YoYo.with(Techniques.FadeIn).playOn(holder.cardView);
04.     final RSSItem current = rssItmes.get(position);
05.     holder.Title.setText(current.getTitle());
06.     holder.Description.setText(current.getDescription());
07.     holder.Date.setText(current.getPubDate());
08.     holder.BDiavasePerissotera.setOnClickListener(new View.OnClickListener() {
09.         public void onClick(View v) {
10.             Intent in = new Intent(context, BrowserActivity.class);
11.             in.putExtra("url", current.getLink());
12.             context.startActivity(in);
13.         }
14.     });
15. }
```

Απόσπασμα κώδικα 4.3.4.3.4

4.3.4.4 – Άνοιγμα είδησης

Μια άλλη δυνατότητα που παρέχεται στον χρήστη είναι πατώντας το κουμπί «Διαβάστε περισσότερα» να ανοίγει το άρθρο, για να μπορεί να το διαβάσει

ολόκληρο. Αυτό για να γίνει, θα πρέπει να δημιουργηθεί ένας browser μέσα στην εφαρμογή ο οποίος θα είναι υπεύθυνος να ανοίξει το άρθρο. Στην προηγούμενη υποενότητα έγινε η αρχικοποίηση του κουμπιού αυτού (απόσπασμα κώδικα 4.3.4.3.2 γραμμή 7) σε κάθε view καθώς και ο ορισμός του url στο οποίο θα οδηγεί. Όταν πατήσει ο χρήστης πάνω στο κουμπί, τότε περνάμε το url του άρθρου σαν έξτρα (απόσπασμα κώδικα 4.3.4.3.4 γραμμή 11) στο Intent [45], το οποίο είναι υπεύθυνο να ανοίξει το activity του browser. Όταν ανοίξει το activity του browser, ελέγχουμε αν το url (που στάλθηκε σαν έξτρα από το απόσπασμα κώδικα 4.3.4.3.4) είναι άδειο. Στην περίπτωση που είναι, κλείνει το activity. Στην περίπτωση που δεν είναι, αρχικοποιούμε ένα web view. Στη συνέχεια, σε αυτό το web view θα ορίσουμε έναν web view client ο οποίος περιέχει τέσσερις μεθόδους (απόσπασμα κώδικα 4.3.4.4.1) [43].

```
01. webView.setWebViewClient(new WebViewClient() {
02.     @Override
03.     public void onPageStarted(WebView view, String url, Bitmap favicon) {
04.         super.onPageStarted(view, url, favicon);
05.     }
06.     @Override
07.     public boolean shouldOverrideUrlLoading(WebView view, String url) {
08.         webView.loadUrl(url);
09.         return false;
10.     }
11.     @Override
12.     public void onPageFinished(WebView view, String url) {
13.         super.onPageFinished(view, url);
14.         dialogProcessDataOpenSite.dismiss();
15.     }
16.     @Override
17.     public void onReceivedError(WebView view, WebResourceRequest request, WebResourceError error) {
18.         super.onReceivedError(view, request, error);
19.         invalidateOptionsMenu();
20.     }
21. });
```

Απόσπασμα κώδικα 4.3.4.4.1

Η μέθοδος «onPageStarted» ενημερώνει την εφαρμογή ότι μια σελίδα ξεκίνησε τη φόρτωση. Η μέθοδος «shouldOverrideUrlLoading» είναι υπεύθυνη να φορτώσει την σελίδα από το url. Η μέθοδος «onPageFinished» έχει αρμοδιότητα να ενημερώσει την εφαρμογή όταν ολοκληρωθεί η φόρτωση της σελίδας. Επίσης, όταν ολοκληρωθεί η φόρτωση της σελίδας κάνουμε dismiss το παράθυρο διαλόγου που εμφανίζεται (εικόνα 38).

Εικόνα 38 – Progress Dialog για το άνοιγμα της είδησης

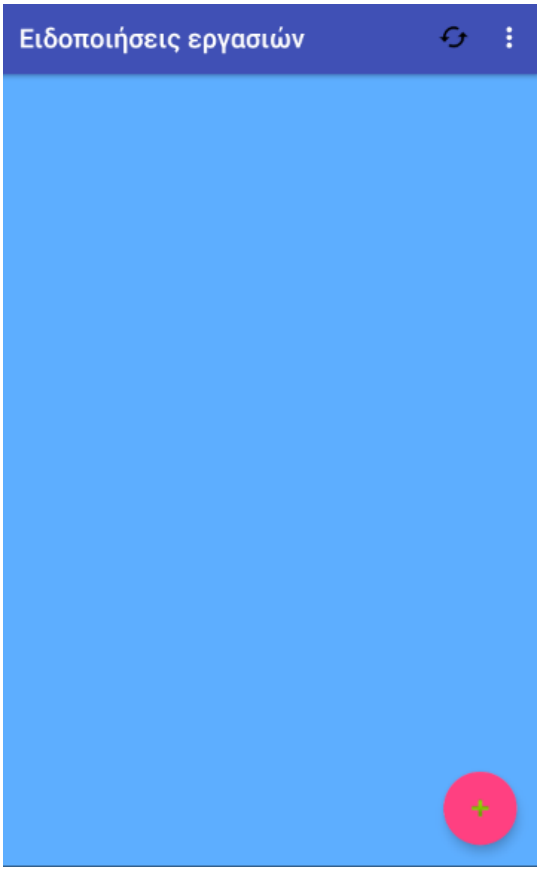
Τέλος, η μέθοδος «onReceivedError» χρησιμοποιείται για να δηλώσει σφάλμα φόρτωσης πόρων ιστού, το οποίο σημαίνει ότι υπάρχει κάποια αδυναμία σύνδεσης με τον διακομιστή.

4.3.5 – Ειδοποιήσεις εργασιών

Αυτή η υπηρεσία της εφαρμογής παρέχει την δυνατότητα στον χρήστη να δημιουργήσει μια εργασία σε συγκεκριμένη ημερομηνία και ώρα. Επίσης, έχει την δυνατότητα ο χρήστης να επιλέξει αν η εργασία που όρισε θέλει να επαναληφθεί μετά από κάποιο χρονικό διάστημα. Όταν φτάσει η ημερομηνία και η ώρα που όρισε ο χρήστης, εμφανίζεται ειδοποίηση στη συσκευή του.

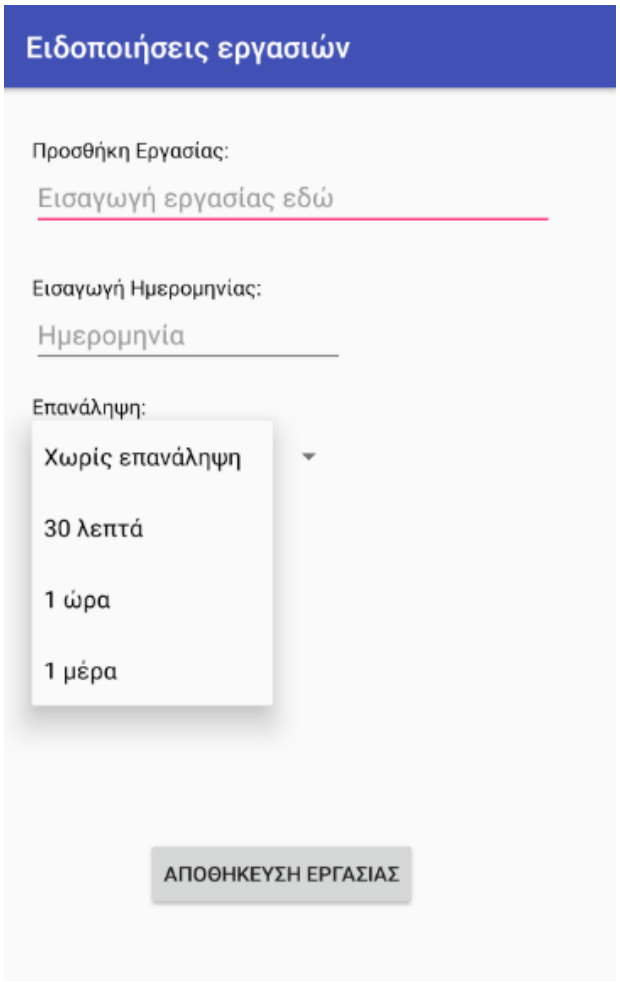
4.3.5.1 – Δημιουργία εργασιών

Για να δημιουργήσουμε μια εργασία θα πατήσουμε το κουμπί κάτω δεξιά που φαίνεται στην εικόνα 39.



Εικόνα 39 – UI ειδοποιήσεις εργασιών

Όταν πατηθεί αυτό το κουμπί ανοίγει ένα άλλο activity στο οποίο μπορούμε να τοποθετήσουμε το όνομα της εργασίας, την ημερομηνία, όπως και την ώρα ενεργοποίησης, καθώς και το αν επαναλαμβάνεται ή όχι (εικόνα 40).



Εικόνα 40 – Δημιουργία εργασίας

Στην παραπάνω εικόνα βλέπουμε ότι δεν υπάρχει πουθενά το πεδίο «Ώρα», αυτό συμβαίνει γιατί το κάναμε να εμφανίζεται αφ' ότου επιλέξει ημερομηνία ο χρήστης. Όταν πατήσει πάνω στο text view «Ημερομηνία» θα ανοίξει ένα παράθυρο το οποίο θα περιέχει ένα ημερολόγιο. Αντίστοιχα και όταν πατήσει πάνω στο πεδίο «Ώρα». Επιλέγοντας την ημερομηνία και την ώρα ο χρήστης και πατώντας «OK» εμφανίζεται η επιλεγμένη ημερομηνία και ώρα αντίστοιχα στα πεδία. Σε περίπτωση που επιλέξει ημερομηνία και ώρα η οποία έχει παρέλθει θα εμφανιστεί αντίστοιχο μήνυμα. Επίσης, αν δεν έχει συμπληρώσει κάποιο από τα πεδία («Προσθήκη Εργασία», «Εισαγωγή Ημερομηνίας», «Εισαγωγή Ώρας») θα εμφανιστεί αντίστοιχο μήνυμα. Αν η εισαγωγή των απαιτούμενων δεδομένων γίνει σωστά, τότε αποθηκεύουμε στην βάση δεδομένων το όνομα της εργασίας, την ημερομηνία και την ώρα ενεργοποίησης, την ένδειξη για το αν επαναλαμβάνεται και τέλος το requestCode (απόσπασμα κώδικα 4.3.5.1.1) [40][42][46][47][62].

```
int requestCode = (int) System.currentTimeMillis();
bundleJobData.putInt("jobRequestCode", requestCode);
database.addJob(editTxtErgasias.getText().toString(), dateFormat.format(calendar.getTime()), repeat, requestCode);

Intent intent = new Intent(context, JobScheduleMyReceiver.class);
intent.putExtras(bundleJobData);

pendingIntent = PendingIntent.getBroadcast(context, requestCode, intent, 0);
Toast.makeText(context, "Η εργασία αποθηκεύτικε με επιτυχία", Toast.LENGTH_LONG).show();
setAlarm(pendingIntent);
```

Απόσπασμα κώδικα 4.3.5.1.1

Στον παραπάνω κώδικα παρατηρούμε ότι στο requestCode του δίνουμε τιμή σύμφωνα με την ώρα του συστήματος σε millisecond. Αυτό το κάνουμε επειδή η συγκεκριμένη μεταβλητή θα χρησιμοποιηθεί σαν id για τη δημιουργία των εργασιών και θέλουμε να είναι μοναδική. Στη συνέχεια, μετά την αποθήκευση των δεδομένων στην βάση γίνεται η προετοιμασία για τη δημιουργία της εργασίας. Όταν ολοκληρωθεί η προετοιμασία για τη δημιουργία της εργασίας, τότε καλείται η μέθοδος «setAlarm» η οποία είναι η κατ' εξοχήν υπεύθυνη για τη δημιουργία της εργασίας (απόσπασμα κώδικα 4.3.5.1.2) [22][23].

```
AlarmManager alarmManager = (AlarmManager) context.getSystemService(ALARM_SERVICE);
switch (repeat) {
    case 0:
        alarmManager.setExact(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(), pendingIntent);
        break;
    case 1:
        alarmManager.setRepeating(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(), AlarmManager.INTERVAL_HALF_HOUR, pendingIntent);
        break;
    case 2:
        alarmManager.setRepeating(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(), AlarmManager.INTERVAL_HOUR, pendingIntent);
        break;
    case 3:
        alarmManager.setRepeating(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(), AlarmManager.INTERVAL_DAY, pendingIntent);
        break;
}
```

Απόσπασμα κώδικα 4.3.5.1.2

Σε αυτή τη μέθοδο γίνεται έλεγχος της μεταβλητής «repeat» και ανάλογα με το περιεχόμενο της θα δημιουργηθεί η εργασία (0→χωρίς επανάληψη, 1→ανά μισή ώρα, 2→ανά μία ώρα και 3→ανά μία μέρα). Έπειτα, γίνεται η μετάβαση στην κλάση «JobScheduleMyReceiver» η οποία θα δημιουργήσει την ενημέρωση που θα εμφανιστεί στη συσκευή του χρήστη όταν φτάσει η χρονική στιγμή ενεργοποίησης της εργασίας (απόσπασμα κώδικα 4.3.5.1.3) [48].

```
NotificationManagerCompat notificationManager = NotificationManagerCompat.from(context);
String valueBundleJobTitle = bundleJobData.getString("jobTitle");

Notification notification = new NotificationCompat.Builder(context, CHANNEL_ID)
    .setSmallIcon(R.drawable.ic_one)
    .setContentTitle(valueBundleJobTitle)
    .setPriority(NotificationCompat.PRIORITY_DEFAULT)
    .build();
notificationManager.notify(idNotification, notification);

updateJobAfterExecuteAlarm(context);
```

Απόσπασμα κώδικα 4.3.5.1.3

Αρχικά, για να δημιουργήσουμε την ενημέρωση, ορίζουμε το «CHANNEL_ID» που θα έχει το συγκεκριμένο Notification (στην περίπτωση μας αφού έχουμε ένα

Notification ότι περιεχόμενο και αν έχει η CHANNEL_ID δεν μας απασχολεί), το εικονίδιο, τον τίτλο τον οποίο τον παίρνουμε από το extra που μας ήρθε από την κλάση η οποία κάλεσε την JobScheduleMyReceiver, καθώς και την προτεραιότητα που θα έχει η ενημέρωση αυτή (στο παράδειγμα μας ορίσαμε την κανονική προτεραιότητα). Τέλος, στη γραμμή 9 γίνεται δημιουργία της ενημέρωσης. Εφόσον ολοκληρωθεί η δημιουργία της ενημέρωσης, τρέχουμε την μέθοδο στη γραμμή 11. Το περιεχόμενο της μεθόδου αυτής φαίνεται στο απόσπασμα κώδικα 4.3.5.1.4 [40][49][62].

```
01. if (repeat > 0) {
02.     Calendar calendar = Calendar.getInstance();
03.     calendar.setTime(convertStringToDate(job.get(1)));
04.     if (repeat == 1)
05.         calendar.add(Calendar.MINUTE, 30);
06.     else if (repeat == 2)
07.         calendar.add(Calendar.HOUR_OF_DAY, 1);
08.     else
09.         calendar.add(Calendar.DAY_OF_MONTH, 1);
10.     database.updateJob(id, job.get(0), convertDateToString(calendar.getTime()), Integer.parseInt(job.get(2)), Integer.parseInt(job.get(3)));
11. }
12. try {
13.     context.startService(new Intent(context, JobScheduleMyService.class));
14.     context.stopService(new Intent(context, JobScheduleMyService.class));
15. } catch (IllegalStateException ex) {
16.     ex.printStackTrace();
17. }
```

Απόσπασμα κώδικα 4.3.5.1.4

Η μέθοδος αυτή το πρώτο πράγμα που κάνει είναι να ελέγξει αν η συγκεκριμένη εργασία που ενεργοποιήθηκε επαναλαμβάνεται (γραμμές 4-9). Σε περίπτωση που επαναλαμβάνεται, θα κάνει update την ημερομηνία και την ώρα της συγκεκριμένης εργασίας στη βάση δεδομένων. Ο κώδικας της μεθόδου update φαίνεται στο απόσπασμα κώδικα 4.3.5.1.5 [50][51].

```
01. public void updateJob(int id, String title, String date, int repeat, int requestCode) {
02.     SQLiteDatabase db = this.getWritableDatabase();
03.
04.     String UPDATE_Job = "UPDATE " + Database.DATABASE_NAME + " " +
05.         "SET title='" + title + "', date='" + date + "', repeat='" + repeat + "', requestCode='" + requestCode + "' " +
06.         "WHERE _id=" + id;
07.
08.     db.execSQL(UPDATE_Job);
09.     db.close();
10. }
```

Απόσπασμα κώδικα 4.3.5.1.5

Στη συνέχεια θα ανανεώσει το UI της εφαρμογής με τη νέα χρονική στιγμή που θα εκτελεστεί η εργασία. Σε περίπτωση που η εργασία δεν επαναλαμβάνεται, τότε απλά θα λέει ότι έληξε. Η ανανέωση του UI πραγματοποιείται ξεκινώντας μια υπηρεσία (service)(απόσπασμα κώδικα 4.3.5.1.4 γραμμή 13). Η υπηρεσία αυτή κάνει την ανανέωση του UI εκτελώντας στην κλάση της υπηρεσίας το απόσπασμα κώδικα 4.3.5.1.6.

```
01. JobScheduleNotifications jobScheduleNotifications = JobScheduleNotifications.instance;
02. jobScheduleNotifications.loaderJobs();
```

Απόσπασμα κώδικα 4.3.5.1.6

Η μέθοδος «loaderJobs» χρησιμοποιείται, για να κάνει φόρτωση των εργασιών από την βάση δεδομένων και από λίστα (περισσότερες λεπτομέρειες θα πούμε στην επόμενη υποενότητα). Για να μπορέσουμε όμως να εκτελέσουμε την μέθοδο αυτή η οποία βρίσκεται στην κλάση JobScheduleNotifications, ορίσαμε ένα static instance της κλάσης αυτής έτσι, ώστε να μπορούμε να έχουμε πρόσβαση από την κλάση της υπηρεσίας.

4.3.5.2 – Φόρτωση εργασιών

Η φόρτωση των εργασιών γίνεται κατά κύριο λόγο από τη βάση δεδομένων. Στην μόνη περίπτωση που γίνεται φόρτωση των εργασιών από λίστα είναι όταν κάνουμε ταξινόμηση είτε με βάση τον τίτλο είτε με βάση την ημερομηνία (απόσπασμα κώδικα 4.3.5.2.1).

```
01.  List<Job> jobsFromDatabase;
02.
03.  if (jobs != null && flagTaxinomisi)
04.      jobsFromDatabase = jobs;
05.  else
06.      jobsFromDatabase = databaseQuery.getAllFutureJobs();
```

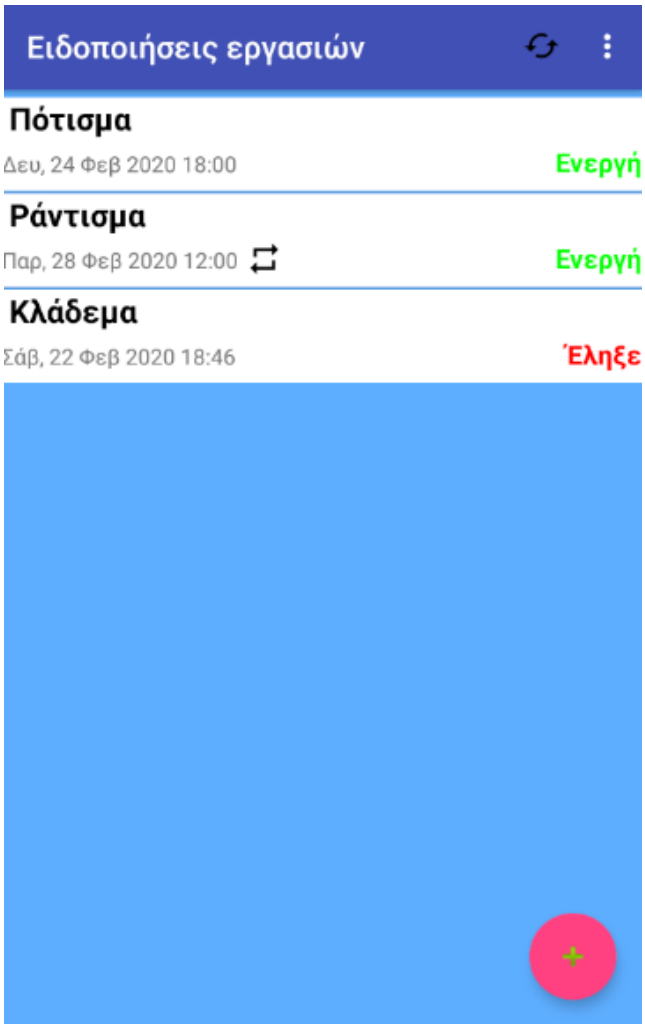
Απόσπασμα κώδικα 4.3.5.2.1

Η μέθοδος «getAllFutureJobs» η οποία είναι υπεύθυνη για την φόρτωση των εργασιών από τη βάση δεδομένων φαίνεται στο παρακάτω απόσπασμα κώδικα [40][51][52][54].

```
01.  public List<Job> getAllFutureJobs() {
02.      Calendar calendar = Calendar.getInstance();
03.      List<Job> jobs = new ArrayList<>();
04.      String query = "select * from " + Database.DATABASE_NAME;
05.      Cursor cursor = this.getDbConnection().rawQuery(query, null);
06.
07.      if (cursor.moveToFirst()) {
08.          do {
09.              int id = cursor.getInt(0);
10.              String title = cursor.getString(cursor.getColumnIndexOrThrow("title"));
11.              String date = cursor.getString(cursor.getColumnIndexOrThrow("date"));
12.              int repeat = cursor.getInt(cursor.getColumnIndexOrThrow("repeat"));
13.
14.              Date dateJob = convertStringToDate(date);
15.              calendar.setTime(dateJob);
16.
17.              jobs.add(new Job(id, title, dateJob, repeat));
18.          } while (cursor.moveToNext());
19.      }
20.      cursor.close();
21.      return jobs;
22.  }
```

Απόσπασμα κώδικα 4.3.5.2.2

Όταν ολοκληρωθεί με επιτυχία η φόρτωση των εργασιών από τη βάση δεδομένων, τότε δημιουργούμε έναν list adapter στον οποίο τοποθετούμε τη λίστα με τις εργασίες οι οποίες φορτώθηκαν, για να τις τοποθετήσει στο UI της εφαρμογής. Σε αυτό το σημείο δεν θα αναλύσουμε τις διαδικασίες που κάνει ο list adapter, για να παρουσιάσει τα δεδομένα επειδή στην προηγούμενη ενότητα (Αγροτικές ειδήσεις) αναλύσαμε τις διαδικασίες που κάνει ο recycler view adapter, για να παρουσιάσει τα δεδομένα. Οι ενέργειες που λαμβάνουν χώρα είτε στο ένα είτε στο άλλο πάνω κάτω είναι ίδιες. Αφού ολοκληρωθεί η τοποθέτηση των εργασιών στο UI, έχουμε το αποτέλεσμα που φαίνεται στην εικόνα 41.



Εικόνα 41 – Αποθηκευμένες εργασίες

Η εικόνα 41 μας δείχνει τρεις προγραμματισμένες εργασίες. Δύο ενεργές και μία που έληξε. Στην εργασία «Ράντισμα» παρατηρούμε ότι δίπλα από την ώρα που έχει προγραμματιστεί να εκτελεστεί υπάρχει το εικονίδιο . Αυτό το εικονίδιο μας δείχνει ότι η συγκεκριμένη εργασία επαναλαμβάνεται.

4.3.5.3 – Επεξεργασία εργασιών

Μια ακόμα δυνατότητα που παρέχει αυτό το activity είναι η επεξεργασία της εργασίας που δημιούργησε ο χρήστης. Αυτό μπορεί να γίνει πατώντας πάνω στην εργασία που θέλει να επεξεργαστεί. Όταν κάνει κλικ πάνω στην εργασία, τότε ανοίγει το activity το οποίο χρησιμοποιούμε, για να εισάγουμε νέα εργασία (απόσπασμα κώδικα 4.3.5.3.1) [47][53].

```
01. LinearLayout linearLayoutParent = (LinearLayout) view;
02. LinearLayout linearLayoutChild = (LinearLayout) linearLayoutParent.getChildAt(0);
03. TextView textViewID = (TextView) linearLayoutChild.getChildAt(1);
04.
05. Bundle dataBundleJobId = new Bundle();
06. dataBundleJobId.putInt("id", Integer.parseInt(textViewID.getText().toString()));
07.
08. Intent intent = new Intent(context, ViewInsertJob.class);
09. intent.putExtras(dataBundleJobId);
10. startActivity(intent);
11. finish();
```

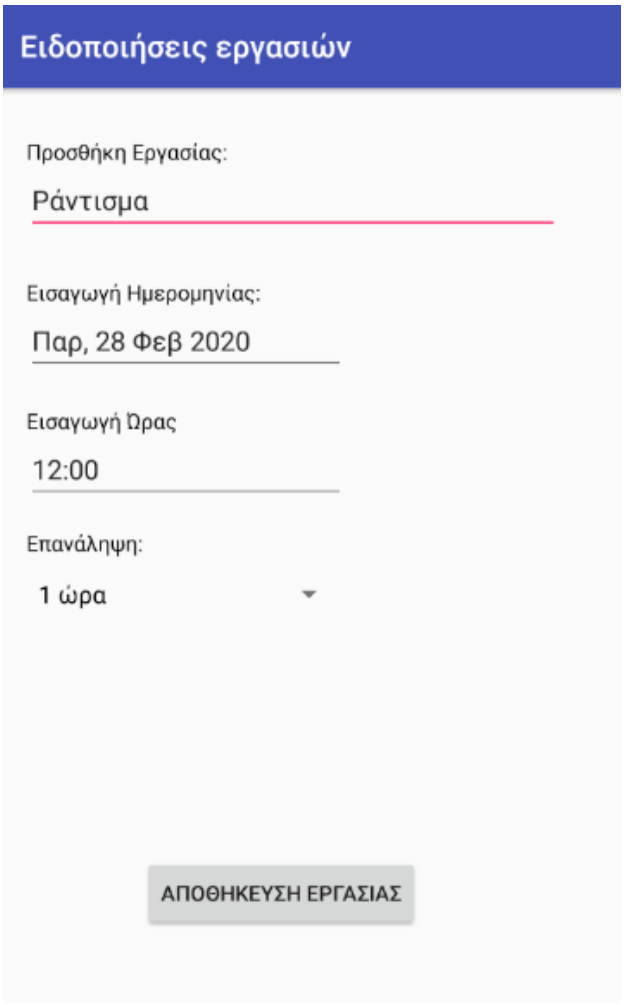
Απόσπασμα κώδικα 4.3.5.3.1

Πριν ανοίξουμε το activity, χρησιμοποιούμε τις γραμμές 1-3 του παραπάνω κώδικα για να πάρουμε το id της επιλεγμένης εργασίας έτσι ώστε να το στείλουμε σαν extra στο activity που θα ανοίξουμε. Αφ' ότου ανοίξει το activity θα τοποθετήσουμε τις τιμές του κάθε πεδίου της εργασίας στα πεδία του activity που άνοιξε (απόσπασμα κώδικα 4.3.5.3.2) [33][54][58][62].

```
01. valueBundleJobId = bundleJobId.getInt("id");
02.
03. List <String> contentJob = databaseQuery.selectAllFields(valueBundleJobId);
04. editTxtErgasias.setText(contentJob.get(0));
05.
06. Date date = convertStringToDate(contentJob.get(1));
07. calendar.setTime(date);
08. SimpleDateFormat dateFormat = new SimpleDateFormat("EEE, d MMM yyyy");
09. editTxtDate.setText(dateFormat.format(calendar.getTime()));
10.
11. txtViewTime.setVisibility(View.VISIBLE);
12. editTxtTime.setVisibility(View.VISIBLE);
13. dateFormat = new SimpleDateFormat("HH:mm");
14. editTxtTime.setText(dateFormat.format(calendar.getTime()));
15.
16. spinnerEpanalipsi.setSelection(Integer.parseInt(contentJob.get(2)));
```

Απόσπασμα κώδικα 4.3.5.3.2

Το αποτέλεσμα της εκτέλεσης του παραπάνω κώδικα φαίνεται στην εικόνα 42.

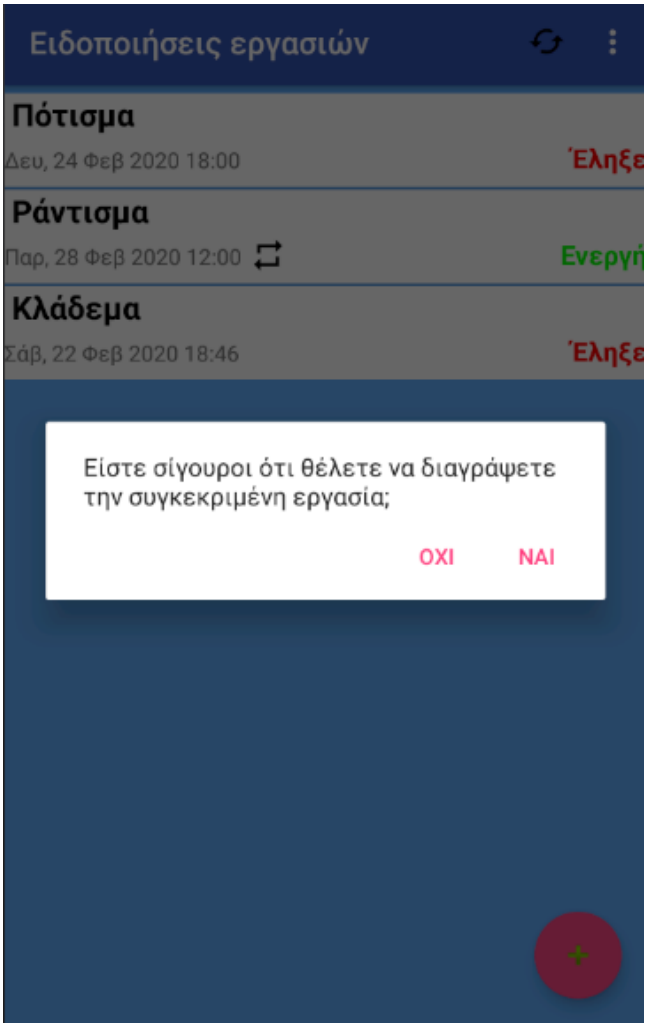


Εικόνα 42 – Επεξεργασία υπάρχουσας εργασίας

Τώρα, έχει τη δυνατότητα ο χρήστης να αλλάξει τα δεδομένα απ' όποιο πεδίο επιθυμεί και να κάνει εκ νέου αποθήκευση της εργασίας πατώντας το κουμπί «ΑΠΟΘΗΚΕΥΣΗ ΕΡΓΑΣΙΑΣ». Όταν πατήσει αυτό το κουμπί εκτός από την αποθήκευση της εργασίας εμφανίζεται μήνυμα το οποίο ενημερώνει τον χρήστη ότι η εργασία άλλαξε με επιτυχία.

4.3.5.4 – Διαγραφή εργασιών

Για να μπορέσει ο χρήστης να διαγράψει μια εργασία θα πρέπει να πατήσει πάνω της παρατεταμένα μέχρι να βγει το μήνυμα που φαίνεται στην εικόνα 43.



Εικόνα 43 – Μήνυμα διαγραφής εργασίας

Στην περίπτωση που ο χρήστης πατήσει «ΝΑΙ» στο μήνυμα που φαίνεται στη παραπάνω εικόνα, τότε θα εκτελεστεί το απόσπασμα κώδικα 4.3.5.4.1 [42][53].

```
01. LinearLayout linearLayoutParent = (LinearLayout) view;
02. LinearLayout linearLayoutChild = (LinearLayout) linearLayoutParent.getChildAt(0);
03. TextView textViewID = (TextView) linearLayoutChild.getChildAt(1);
04.
05. int id = Integer.parseInt(textViewID.getText().toString());
06. int requestCode = databaseQuery.selectRequestCode(id);
07.
08. ViewInsertJob objViewInsertJob = new ViewInsertJob();
09. objViewInsertJob.deleteJob(requestCode, context);
10. database.deleteJob(id);
11. loaderJobs();
12. Toast.makeText(context, "Η διαγραφή της εργασίας ολοκληρώθηκε με επιτυχία", Toast.LENGTH_LONG).show();
```

Απόσπασμα κώδικα 4.3.5.4.1

Αφ’ ότου πάρουμε το id της επιλεγμένης εργασίας, θα το χρησιμοποιήσουμε για να κάνουμε request στη βάση δεδομένων με σκοπό να μας επιστρέψει τον «requestCode» (id εργασίας) έτσι, ώστε να ακυρώσουμε την προγραμματισμένη εργασία (γραμμή 9). Ο κώδικας της ακύρωσης της εργασίας φαίνεται στο παρακάτω απόσπασμα κώδικα [55].

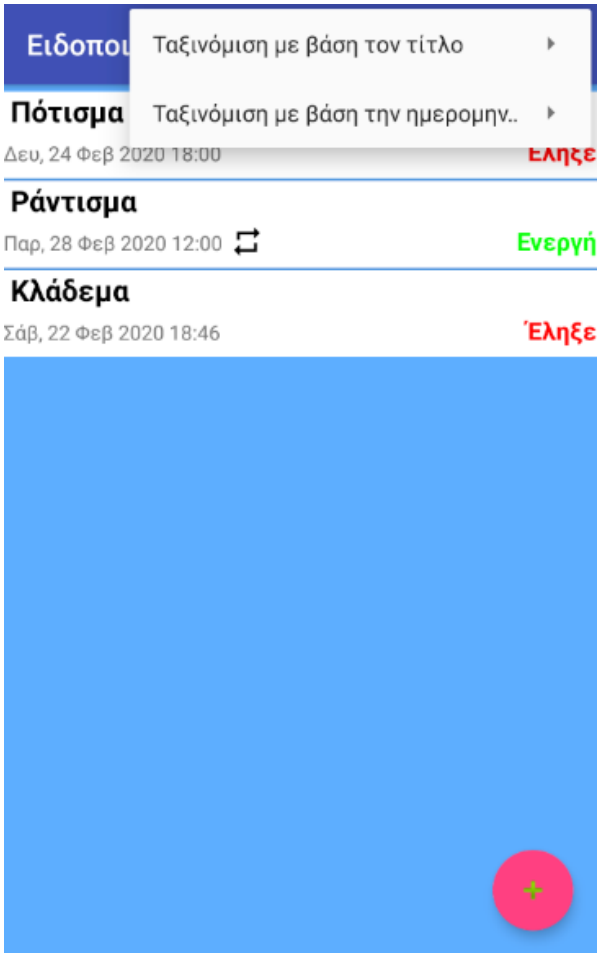
```
01. AlarmManager alarmManager = (AlarmManager) context.getSystemService(ALARM_SERVICE);
02.
03. Intent intent = new Intent(context, JobScheduleMyReceiver.class);
04. PendingIntent pendingIntent = PendingIntent.getBroadcast(context, requestCode, intent, 0);
05.
06. alarmManager.cancel(pendingIntent);
```

Απόσπασμα κώδικα 4.3.5.4.2

Έπειτα, διαγράφουμε την επιλεγμένη εργασία από την βάση δεδομένων (απόσπασμα κώδικα 4.3.5.4.1 γραμμή 10) και στη συνέχεια ανανεώνουμε το UI της εφαρμογής (γραμμή 11). Τέλος, εμφανίζουμε το μήνυμα που φαίνεται στη γραμμή 12.

4.3.5.5 – Ταξινόμηση εργασιών

Μια ακόμα δυνατότητα που παρέχει το activity της ειδοποίησης εργασιών είναι η ταξινόμηση κατά αύξουσα ή κατά φθίνουσα σειρά. Η ταξινόμηση μπορεί να γίνει είτε με βάση τον τίτλο της εργασίας είτε με βάση την ημερομηνία εκτέλεσης (εικόνα 44).



Εικόνα 44 – Μενού ταξινόμησης

4.3.5.5.1 – Ταξινόμηση με βάση τον τίτλο

Για να μπορέσει να υλοποιηθεί η ταξινόμηση με βάση τον τίτλο των εργασιών θα πρέπει να δημιουργήσουμε μέσα στη κλάση η οποία είναι υπεύθυνη για την δημιουργία εργασιών τη μέθοδο «compareTo». Ο κώδικας ο οποίος εκτελείται για να ταξινομήσει τις εργασίες με βάση τον τίτλο φαίνεται στο απόσπασμα κώδικα 4.3.5.5.1.1 [46].

```
01.  if (flagCompareTitleDate)
02.      if (getTitle() == null || obj.getTitle() == null)
03.          return 0;
04.      else {
05.          //auksousa
06.          if (flagAuksousaFthinousaTaxinomisi)
07.              return getTitle().compareTo(obj.getTitle());
08.          //fthinousa
09.          else
10.              return (obj.getTitle()).compareTo(getTitle());
11.      }
```

Απόσπασμα κώδικα 4.3.5.5.1.1

Στον παραπάνω κώδικα έχουμε δύο static μεταβλητές (flagCompareTitleDate και flagAuksousaFthinousaTaxinomisi). Τις κάναμε static έτσι, ώστε από το κύριο activity της εφαρμογής να μπορούμε να εισάγουμε τιμές (true, false) στη κάθε μία ανάλογα με το αν ο χρήστης επιλέξει ταξινόμηση με βάση τον τίτλο (γραμμή 1) και αν επιθυμεί κατά αύξουσα ή φθίνουσα σειρά (γραμμή 6 και 9).

4.3.5.5.2 – Ταξινόμηση με βάση την ημερομηνία

Για να γίνει ταξινόμηση με βάση την ημερομηνία ενεργοποίησης των εργασιών όπως και στην περίπτωση της ταξινόμησης με βάση τον τίτλο (προηγούμενη υπό ενότητα) θα πρέπει να χρησιμοποιηθεί η μέθοδος «compareTo». Αλλά επειδή μέσα σε μία κλάση δε μπορεί να υπάρχουν περισσότερες από μία ίδιες μεθόδους (ίδιο όνομα, ίδιες παράμετροι) χρησιμοποιούμε static μεταβλητές. Ο κώδικας ο οποίος χρησιμοποιείται για την υλοποίηση της ταξινόμησης των εργασιών με βάση την ημερομηνία ενεργοποίησης φαίνεται στο απόσπασμα κώδικα 4.3.5.5.2.1 [46].

```
01.  else {
02.      //auksousa
03.      if (flagAuksousaFthinousaTaxinomisi)
04.          return getDate().compareTo(obj.getDate());
05.      //fthinousa
06.      else
07.          return (obj.getDate()).compareTo(getDate());
08.  }
```

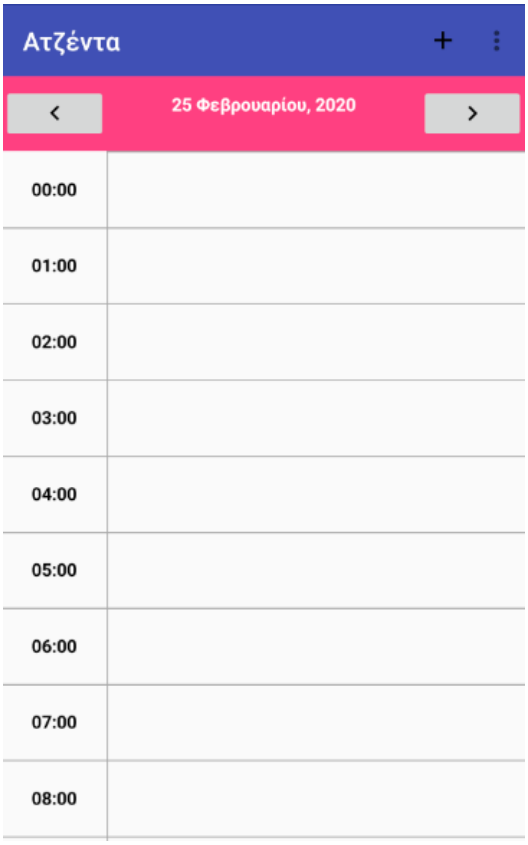
Απόσπασμα κώδικα 4.3.5.5.2.1

4.3.6 – Ατζέντα

Μια ακόμα υπηρεσία που παρέχει η εφαρμογή είναι η ατζέντα. Στην ατζέντα ο χρήστης μπορεί να προσθέσει γεγονότα στα οποίο επιθυμεί να παρευρεθεί έτσι, ώστε να μη ξεχάσει τη μέρα και την ώρα που θα πραγματοποιηθούν. Η προβολή των γεγονότων μπορεί να γίνει με τρεις τρόπους. Είτε ημερήσια είτε εβδομαδιαία είτε μηνιαία.

4.3.6.1 – Ημερήσια προβολή γεγονότων

Στην ημερήσια προβολή των γεγονότων βλέπουμε τα γεγονότα που έχουμε δημιουργήσει ανά ημέρα (εικόνα 45).

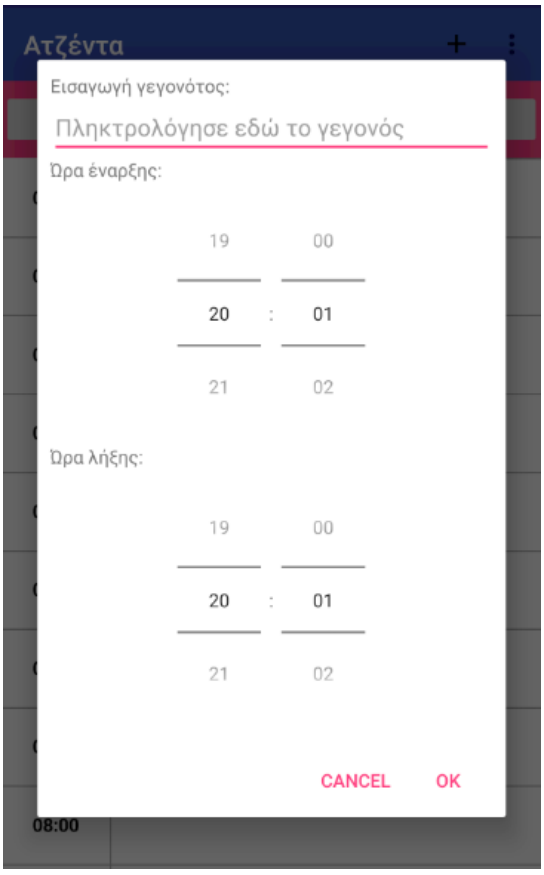


Εικόνα 45 – UI ημερήσια προβολή γεγονότων

Επίσης, έχουμε την δυνατότητα να μεταβούμε σε προηγούμενες και επόμενες μέρες πατώντας τα βελάκια αριστερά και δεξιά. Ακόμα, πατώντας πάνω στην ημερομηνία επιστρέφουμε στη σημερινή μέρα.

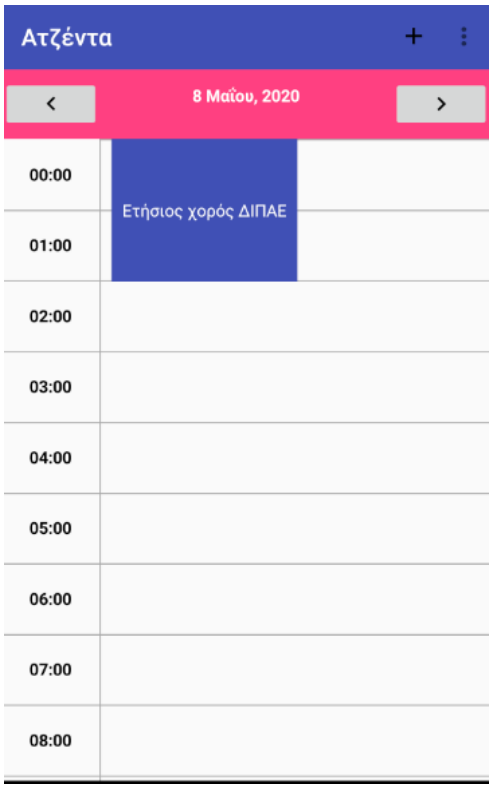
4.3.6.1.1 – Δημιουργία γεγονότων

Για να δημιουργήσουμε ένα γεγονός πατάμε στο εικονίδιο + το οποίο βρίσκεται πάνω δεξιά στην εφαρμογή. Όταν πατήσουμε πάνω σε αυτό το εικονίδιο ανοίγει το παράθυρο που φαίνεται στην εικόνα 46.



Εικόνα 46 – Προσθήκη γεγονότος, ημερήσια προβολή

Σε αυτό το παράθυρο που εμφανίστηκε καταχωρούμε πληροφορίες για το γεγονός, καθώς και ώρα έναρξης και λήξης. Σε περίπτωση που δεν εισάγει πληροφορίες για το γεγονός ο χρήστης εμφανίζεται αντίστοιχο μήνυμα. Επίσης, αν η ώρα λήξης είναι προγενέστερη της ώρας έναρξης πάλι εμφανίζεται αντίστοιχο μήνυμα. Αν όλα πάνε καλά με την εισαγωγή του γεγονότος θα έχουμε το αποτέλεσμα που φαίνεται στην εικόνα 47.



Εικόνα 47 – UI μετά την εισαγωγή γεγονότος

Ο κώδικας ο οποίος είναι υπεύθυνος για τη δημιουργία της εργασίας φαίνεται παρακάτω [33][42][49][56][58][62].

```
01. cal.set(Calendar.HOUR_OF_DAY, timePickerStart.getCurrentHour());
02. cal.set(Calendar.MINUTE, timePickerStart.getCurrentMinute());
03. cal.set(Calendar.SECOND, 0);
04.
05. SimpleDateFormat dateFormat = new SimpleDateFormat("d-MM-yyyy HH:mm");
06. String dateStartFormatted = dateFormat.format(cal.getTime());
07.
08. cal.set(Calendar.HOUR_OF_DAY, timePickerEnd.getCurrentHour());
09. cal.set(Calendar.MINUTE, timePickerEnd.getCurrentMinute());
10. cal.set(Calendar.SECOND, 0);
11. String dateEndFormatted = dateFormat.format(cal.getTime());
12.
13. int eventBlockHeight = getEventTimeFrame(convertStringToDate(dateStartFormatted), convertStringToDate(dateEndFormatted));
14. alertDialog.dismiss();
15. displayEventSection(convertStringToDate(dateStartFormatted), eventBlockHeight, editTxtEisagogiGegonotos.getText().toString());
16. database.addEvent(editTxtEisagogiGegonotos.getText().toString(), dateStartFormatted, dateEndFormatted);
17. Toast.makeText(context, "Το γεγονός αποθηκεύτικε με επιτυχία", Toast.LENGTH_LONG).show();
```

Απόσπασμα κώδικα 4.3.6.1.1.1

Στις πρώτες γραμμές του παραπάνω κώδικα (1-11) έχουμε τις διαδικασίες που γίνονται, για να πάρουμε την ώρα έναρξης και λήξης του γεγονότος από τα time picker. Αφ' ότου γίνει αυτό, χρησιμοποιούμε τη μέθοδο «getEventTimeFrame», για να πάρουμε το διάστημα έναρξης και λήξης του γεγονότος. Τώρα, πάμε να πούμε το σκεπτικό της διαδικασίας αυτής. Αρχικά τοποθετούμε την ημερομηνία (από την ημερομηνία θα πάρουμε την ώρα (απόσπασμα κώδικα 4.3.6.1.1.2 γραμμή 4 και 7), καθώς και τα λεπτά (μόνο για την ώρα λήξης έτσι, ώστε να εκτείνεται η γραμμή στο κατάλληλο σημείο, αν για παράδειγμα επιλέξει ο χρήστης ότι η ώρα λήξης είναι 18:30. Σε περίπτωση που επιλέξει ώρα αρχής 18:30 το πώς θα τοποθετηθεί η

αρχή του γεγονότος στο σωστό σημείο θα το δούμε παρακάτω) αρχής και τέλους του γεγονότος (γραμμές 3 και 6). Λόγω της κατασκευής του layout αν η ώρα έναρξης είναι μονός αριθμός (1,3,5,7, κτλ) τότε το διάστημα της μιας ώρας είναι 157 dp (γραμμή 10), αν είναι ζυγός αριθμός (0,2,4,6, κτλ) τότε το διάστημα είναι 158 dp γι' αυτό αυξάνουμε τη μεταβλητή «defaultHeight» κατά ένα (γραμμή 15) και το υποδιπλάσιο της διαφοράς μεταξύ της ώρας λήξης και αρχής (γραμμή 12) το μετατρέπουμε σε αρνητικό αριθμό (γραμμή 16) γιατί αλλιώς θα θεωρούνταν ότι όλα τα διαστήματα απέχουν 158 dp, αλλά όπως αναφέραμε παραπάνω το μέγεθος του διαστήματος αλλάζει ανάλογα με το αν η ώρα αρχής είναι μονός ή ζυγός αριθμός. Σε περίπτωση που ο χρήστης στην ώρα λήξης εισάγει και λεπτά τότε εκτελούνται οι γραμμές 19-34. Τα διάστημα της ώρας από το 1 λεπτό έως το 59 είναι 154 dp για την μονή ώρα έναρξης και 155 dp για την ζυγή γι' αυτό γίνεται έλεγχος στη γραμμή 32 για το αν η ώρα αρχής είναι ζυγός αριθμός και αν είναι αυξάνουμε τη μεταβλητή «katastasiMeVasiTaLepta» κατά ένα (γραμμή 33). Κάνοντας κάποιες δοκιμές, παρατηρήσαμε ότι αν πολλαπλασιάσουμε το πεντηκοστό ένατο (59) λεπτό με το δύο (59 * 2 = 118), δεν ξεπερνάμε το διάστημα των 154 dp. Από το αποτέλεσμα που βγάλαμε στον πολλαπλασιασμό (το 118) υπολογίσαμε πόσο απέχει για να φτάσει το 118 στο 154, η διαφορά είναι στα 36 dp. Και για να είναι όσο το δυνατόν πιο σωστή η αντιστοίχιση, δημιουργήσαμε έξι ομάδες υπολογισμού ανά δέκα λεπτά. Όσο ανεβαίνει η ομάδα των λεπτών ανεβαίνει και η σχέση υπολογισμού ανά έξι. Η μέθοδος αυτή φαίνεται παρακάτω [40][57][62].

```
01. private int getEventTimeFrame(Date start, Date end){
02.     Calendar mCal = Calendar.getInstance();
03.     mCal.setTimeInMillis(start.getTime());
04.     int hourStart = mCal.get(Calendar.HOUR_OF_DAY);
05.
06.     mCal.setTimeInMillis(end.getTime());
07.     int hourEnd = mCal.get(Calendar.HOUR_OF_DAY);
08.     int minutesEnd = mCal.get(Calendar.MINUTE);
09.
10.     int defaultHeight = 157;
11.     int differenceHours = hourEnd - hourStart;
12.     int differenceMonaZiga = differenceHours / 2;
13.
14.     if((hourStart % 2) == 0) {
15.         defaultHeight++;
16.         differenceMonaZiga = -differenceMonaZiga;
17.    }
18.     int katataksiMeVasiTaLepta = 0;
19.     if (minutesEnd != 0) {
20.         if (minutesEnd <= 10)
21.             katataksiMeVasiTaLepta = minutesEnd * 2 + 6;
22.         else if (minutesEnd <= 20)
23.             katataksiMeVasiTaLepta = minutesEnd * 2 + 12;
24.         else if (minutesEnd <= 30)
25.             katataksiMeVasiTaLepta = minutesEnd * 2 + 18;
26.         else if (minutesEnd <= 40)
27.             katataksiMeVasiTaLepta = minutesEnd * 2 + 24;
28.         else if (minutesEnd <= 50)
29.             katataksiMeVasiTaLepta = minutesEnd * 2 + 30;
30.         else
31.             katataksiMeVasiTaLepta = minutesEnd * 2 + 36;
32.     if((hourStart % 2) == 0)
33.         katataksiMeVasiTaLepta++;
34.     }
35.     return differenceHours * defaultHeight + 3 + differenceMonaZiga + katataksiMeVasiTaLepta;
36. }
```

Απόσπασμα κώδικα 4.3.6.1.1.2

Έπειτα, εκτελείται η μέθοδος «displayEventSection» η οποία χρησιμοποιεί την ώρα έναρξης του γεγονότος [αφού κάνει κάποια επεξεργασία πάνω σ' αυτή (απόσπασμα κώδικα 4.3.6.1.1.3 γραμμή 8)], για να τοποθετήσει το γεγονός στη σωστή χρονική στιγμή. Η μέθοδος αυτή φαίνεται στο απόσπασμα κώδικα παρακάτω [33][58].

```
01. private void displayEventSection(Date eventDate, int height, String messenge){
02.     SimpleDateFormat timeFormatter = new SimpleDateFormat("HH:mm", Locale.getDefault());
03.     String displayValue = timeFormatter.format(eventDate);
04.
05.     String hourMinutes[] = displayValue.split(":");
06.     int hours = Integer.parseInt(hourMinutes[0]);
07.     int minutes = Integer.parseInt(hourMinutes[1]);
08.
09.     createEventView(pushEvent(hours, minutes), height, messenge);
10. }
```

Απόσπασμα κώδικα 4.3.6.1.1.3

Πριν εκτελεστεί η μέθοδος «createEventView» εκτελείται η «pushEvent» η οποία θα τοποθετήσει την αρχή του γεγονότος στο σωστό σημείο. Η μέθοδος «pushEvent» φαίνεται παρακάτω.

```
01.  private int pushEvent(int hour, int minutes) {
02.      int push = hour * 158;
03.      int minus = hour / 2;
04.      int minute = minutes * 2;
05.
06.      int katataksiMeVasiTaLepta = 0;
07.      if (minutes != 0) {
08.          if (minutes <= 10)
09.              katataksiMeVasiTaLepta = minute + 6;
10.          else if (minutes <= 20)
11.              katataksiMeVasiTaLepta = minute + 12;
12.          else if (minutes <= 30)
13.              katataksiMeVasiTaLepta = minute + 18;
14.          else if (minutes <= 40)
15.              katataksiMeVasiTaLepta = minute + 24;
16.          else if (minutes <= 50)
17.              katataksiMeVasiTaLepta = minute + 30;
18.          else
19.              katataksiMeVasiTaLepta = minute + 36;
20.          if ((hour % 2) == 0)
21.              katataksiMeVasiTaLepta++;
22.          minusEventFrame = katataksiMeVasiTaLepta;
23.      }
24.      return push - minus + katataksiMeVasiTaLepta;
25.  }
```

Απόσπασμα κώδικα 4.3.6.1.1.4

Η λογική τοποθέτησης των λεπτών όσο αφορά το διάστημα μεταξύ της αρχής και της λήξης του γεγονότος (απόσπασμα κώδικα 4.3.6.1.1.2) είναι ίδια και στην τοποθέτηση του γεγονότος στο σημείο αρχής (γραμμές 7-21 απόσπασμα κώδικα 4.3.6.1.1.4). Μια διαφορά υπάρχει στη γραμμή κώδικα 22, η οποία χρησιμοποιείται για την περίπτωση που ο χρήστης εισάγει και λεπτά (εκτός από ώρα) στην ώρα έναρξης και λήξης έτσι, ώστε να γίνει αφαίρεση από το συνολικό διάστημα. Γιατί αν εισάγει και λεπτά το διάστημα (αρχής και λήξης) θα μεταβληθεί πιο κάτω. Εμείς αφαιρούμε το διάστημα το οποίο προστέθηκε λόγω της εισαγωγής λεπτών (είτε στην ώρα έναρξης είτε στην ώρα λήξης) έτσι, ώστε αν βάλαμε 18:10 έως 19:00 να λήξει στο σημείο που ξεκινάει η ώρα «19:00» και όχι 19:10. Μια ακόμα διαφορά είναι ότι εδώ χρησιμοποιούμε τα 158 dp σαν προεπιλεγμένο διάστημα μιας ώρας ενώ στο απόσπασμα κώδικα 4.3.6.1.1.2 χρησιμοποιήσαμε τα 157 dp. Αλλά όπως και στο απόσπασμα κώδικα 4.3.6.1.1.2 χρησιμοποιήσαμε κάποιες μαθηματικές παραστάσεις, για να γίνεται σωστός υπολογισμός των διαστημάτων των μονών και ζυγών ωρών, έτσι και στο παραπάνω απόσπασμα κώδικα κάνουμε το ίδιο (γραμμές 2-4).

Στη συνέχεια, εκτελούμε τη μέθοδο «createEventView» η οποία είναι υπεύθυνη για τη δημιουργία του γεγονότος στο UI της εφαρμογής. Στη μέθοδο αυτήν περνάμε την επεξεργασμένη ώρα έναρξης του γεγονότος, το επεξεργασμένο διάστημα έναρξης λήξης του γεγονότος, καθώς και την περιγραφή του γεγονότος σαν παραμέτρους. Η μέθοδος αυτή φαίνεται παρακάτω [59][60][61].

```
01. private void createEventView(int topMargin, int height, String message) {
02.     height -= minusEventFrame;
03.     final TextView mEventView = new TextView(Agenda.this);
04.
05.     RelativeLayout.LayoutParams lParam = new RelativeLayout.LayoutParams(
06.         LinearLayout.LayoutParams.WRAP_CONTENT, LinearLayout.LayoutParams.WRAP_CONTENT);
07.     lParam.addRule(RelativeLayout.ALIGN_PARENT_TOP);
08.     lParam.topMargin = topMargin;
09.     lParam.leftMargin = 24;
10.
11.     mEventView.setLayoutParams(lParam);
12.     mEventView.setPadding(24, 0, 24, 0);
13.     mEventView.setHeight(height);
14.     mEventView.setGravity(0x11);
15.     mEventView.setTextColor(Color.parseColor("#ffffff"));
16.     mEventView.setText(message);
17.     mEventView.setBackgroundColor(Color.parseColor("#3F51B5"));
18.
19.     relativeLayoutCount++;
20.     mLayout.addView(mEventView, eventIndex - 1);
21.     minusEventFrame = 0;
22. }
```

Απόσπασμα κώδικα 4.3.6.1.1.5

Πριν περιγράψουμε τη διαδικασία για τη δημιουργία του γεγονότος στο UI να πούμε ότι στη γραμμή 2 γίνεται η μείωση του διαστήματος αρχής και λήξης σε περίπτωση που ο χρήστης εισάγει και λεπτά.

Για να ορίσουμε το γεγονός στο UI της εφαρμογής αρχικά θα δημιουργήσουμε ένα text view. Έπειτα, δημιουργούμε ένα relative layout στο οποίο περνάμε κάποιες παραμέτρους αρχικοποίησης. Μέσα σε αυτές τις παραμέτρους που περνάμε, περνάμε και την επεξεργασμένη ώρα έναρξης (γραμμή 8). Στη συνέχεια, περνάμε κάποιες παραμέτρους στο text view. Μέσα σε αυτές τις παραμέτρους είναι και το επεξεργασμένο διάστημα μεταξύ έναρξης και λήξης (γραμμή 13), καθώς και η περιγραφή του γεγονότος (γραμμή 16). Έπειτα, αυξάνεται κατά ένα ο μετρητής ο οποίος χρησιμοποιείται, για να μετρήσει το πλήθος των γεγονότων που έχουν εισαχθεί (γραμμή 19) και γίνεται εισαγωγή του γεγονότος στο UI (γραμμή 20). Τέλος, στη γραμμή 21 μηδενίζουμε τη μεταβλητή «minusEventFrame» έτσι, ώστε αν ο χρήστης δεν εισάγει λεπτά να μην γίνει μείωση του διαστήματος η οποία έμεινε από προηγούμενη εισαγωγή γεγονότος.

4.3.6.1.2 – Φόρτωση γεγονότων

Για να φορτώσουμε τα αποθηκευμένα γεγονότα από τη βάση δεδομένων θα χρησιμοποιήσουμε τη μέθοδο «displayDailyEvents». Η μέθοδος αυτή φαίνεται παρακάτω [62].

```
01. private void displayDailyEvents() {
02.     Date calendarDate = cal.getTime();
03.     List<EventObjects> dailyEvent = mQuery.getAllFutureEvents(calendarDate);
04.
05.     relativeLayoutCount = 0;
06.     selectedDate = cal.getTime();
07.
08.     for (EventObjects eObject: dailyEvent) {
09.         Date eventDate = eObject.getDate();
10.         Date endDate = eObject.getEnd();
11.         String eventMessage = eObject.getMessage();
12.         int eventBlockHeight = getEventTimeFrame(eventDate, endDate);
13.         displayEventSection(eventDate, eventBlockHeight, eventMessage);
14.     }
15. }
```

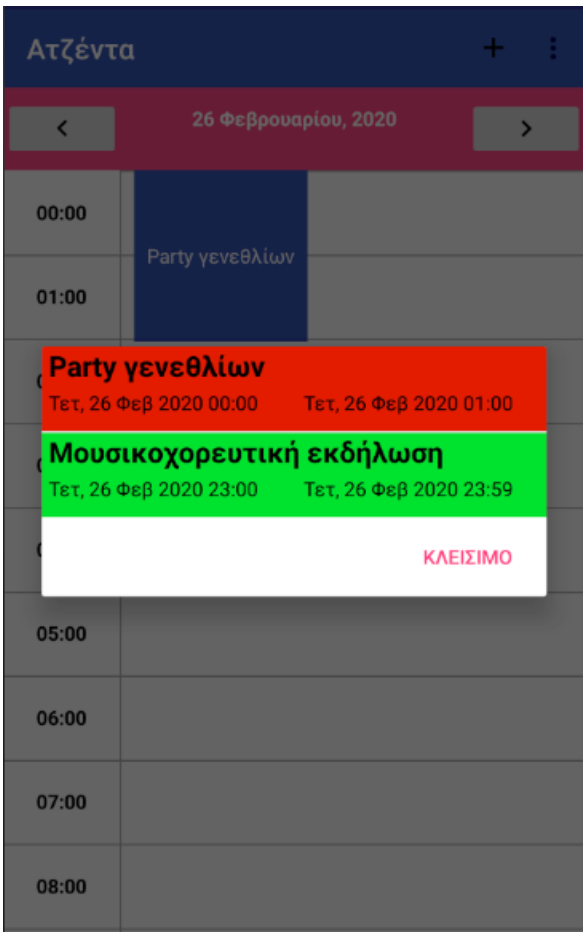
Απόσπασμα κώδικα 4.3.6.1.2.1

Να σημειωθεί σε αυτό το σημείο ότι η μέθοδος αυτή τρέχει κάθε φορά που γίνεται αλλαγή ημέρας. Δημιουργούμε μια λίστα η οποία θα υποδεχτεί τα γεγονότα από τη βάση δεδομένων. Έπειτα, μηδενίζουμε τη μεταβλητή που υπολογίζει τα γεγονότα που εισάγονται. Αυτό γίνεται επειδή θα παρουσιάσουμε τα γεγονότα άλλης μέρας. Επίσης, τοποθετούμε την ημερομηνία των γεγονότων που θα παρουσιαστούν στη μεταβλητή «selectedDate». Αυτή τη μεταβλητή θα τη χρειαστούμε σε περίπτωση που ο χρήστης θελήσει να επεξεργαστεί ή να διαγράψει ένα γεγονός. Αυτό το κάνουμε έτσι, ώστε να την έχουμε σαν σημείο

αναφοράς, για να ξανά προβάλουμε τα γεγονότα της μέρας εκ νέου (περισσότερες λεπτομέρειες θα αναφέρουμε στη παρακάτω υπό ενότητα). Στη συνέχεια ακολουθεί μια επαναληπτική διαδικασία με την οποία γίνεται τοποθέτηση των γεγονότων κατάλληλα στο UI. Δεν θα αναφέρουμε λεπτομέρειες γιατί η διαδικασία αυτή παρουσιάστηκε στη προηγούμενη υποενότητα.

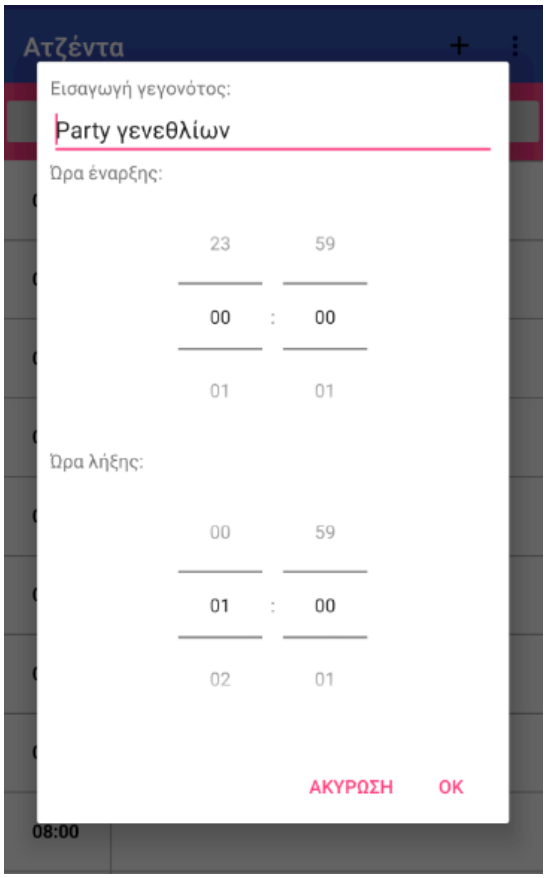
4.3.6.1.3 – Συγκεντρωτική εμφάνιση, επεξεργασία και διαγραφή γεγονότων

Για να γίνει συγκεντρωτική εμφάνιση όλων των γεγονότων μιας μέρας πατάμε πάνω στο ροζ πλαίσιο που περιβάλλει την ημερομηνία. Αν στη συγκεκριμένη ημέρα δεν έχει αποθηκευτεί κάποιο γεγονός θα εμφανιστεί αντίστοιχο μήνυμα. Στη περίπτωση που έχει αποθηκευτεί γεγονός ή γεγονότα θα εμφανιστεί η λίστα που φαίνεται στην εικόνα 48. Η εμφάνιση της λίστας υλοποιείται με την χρήση adapter του οποίου τις διαδικασίες δεν θα τις εξηγήσουμε γιατί το κάναμε σε προηγούμενη υποενότητα.



Εικόνα 48 – Συγκεντρωτική εμφάνιση γεγονότων

Στην εικόνα 48 παρατηρούμε ότι ένα γεγονός έχει κόκκινο χρώμα και το άλλο έχει πράσινο. Το γεγονός που έχει σηματοδοτηθεί με κόκκινο χρώμα σημαίνει ότι έχει παρέλθει, ενώ το γεγονός με πράσινο σημαίνει ότι ακόμα είναι ενεργό. Πατώντας πάνω σε κάποιο γεγονός μπορούμε να το επεξεργαστούμε (εικόνα 49).



Εικόνα 49 – Επεξεργασία υπάρχοντος γεγονότος

Αφ' ότου κάνει τις αλλαγές που επιθυμεί ο χρήστης και πατήσει «OK», τότε λαμβάνουν χώρα οι εξής ενέργειες: αποθήκευση των αλλαγών στη βάση δεδομένων, διαγραφή των υπαρχόντων γεγονότων από το UI της εφαρμογής (για να τοποθετηθούν τα ενημερωμένα), τοποθέτηση στη μεταβλητή «cal» την ημερομηνία που αποθηκεύτηκε στη «selectedDate» έτσι, ώστε στη συνέχεια που θα εκτελεστεί η μέθοδος «displayDailyEvents» να εμφανίσει τα ενημερωμένα πλέον γεγονότα (απόσπασμα κώδικα 4.3.6.1.3.1) [54].

```
01.  for (int i = 1; i <= relativeLayoutCount; i++)
02.      mLayout.removeViewAt(eventIndex - 1);
03.
04.  cal.setTime(selectedDate);
05.  displayDailyEvents();
```

Απόσπασμα κώδικα 4.3.6.1.3.1

Επίσης ο χρήστης, πατώντας παρατεταμένα πάνω σε ένα γεγονός, έχει τη δυνατότητα να το διαγράψει. Σε περίπτωση που στη λίστα υπάρχουν περισσότερα από ένα γεγονότα, τότε το παράθυρο της λίστας δεν εξαφανίζεται, αν υπάρχει μόνο ένα, τότε θα εξαφανιστεί.

4.3.6.1.4 – Μετάβαση σε προηγούμενη / επόμενη μέρα και στο σήμερα

Για να μεταβούμε είτε σε προηγούμενη είτε σε επόμενη μέρα χρησιμοποιούμε τα κουμπιά με το αριστερό και το δεξί βελάκι. Ο κώδικας της μετάβασης σε επόμενη μέρα φαίνεται στο παρακάτω απόσπασμα [49][62].

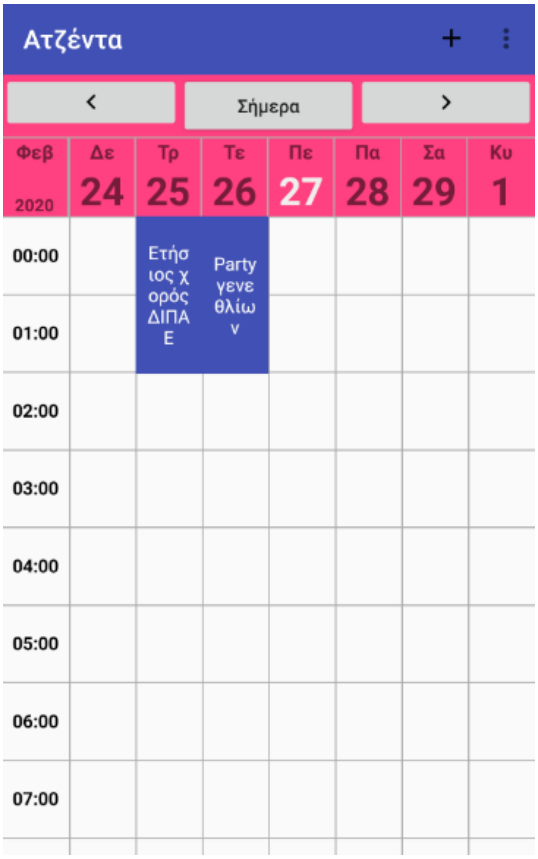
```
01. private void nextCalendarDate() {
02.     for (int i = 1; i <= relativeLayoutCount; i++)
03.         mLayout.removeViewAt(eventIndex - 1);
04.
05.     cal.add(Calendar.DAY_OF_MONTH, 1);
06.     displayCurrentDate.setText(displayDateInString(cal.getTime()));
07.     displayDailyEvents();
08. }
```

Απόσπασμα κώδικα 4.3.6.1.4.1

Στην αρχή διαγράφουμε τα γεγονότα της μέρας που παρουσιάζονται αυτήν τη στιγμή, έπειτα κάνουμε την ημερομηνία συν μία μέρα, μετά ενημερώνουμε το πλαίσιο ημερομηνίας στο πάνω μέρος του activity και στη συνέχεια εκτελούμε τη μέθοδο «displayDailyEvents», για να τοποθετήσει τα δεδομένα της επόμενης μέρας στο UI της εφαρμογής. Τον κώδικα μετάβασης σε προηγούμενη μέρα δεν χρειάζεται να τον παρουσιάσουμε και να τον εξηγήσουμε, γιατί είναι ακριβώς ίδιος με τον κώδικα μετάβασης σε επόμενη μέρα, η μόνη διαφορά είναι στη γραμμή 5, αντί για «1» έχει «-1». Επίσης, ούτε τον κώδικα μετάβασης στο «σήμερα» δεν χρειάζεται να αναφερθεί γιατί και αυτός είναι παρόμοιος με τον κώδικα παραπάνω.

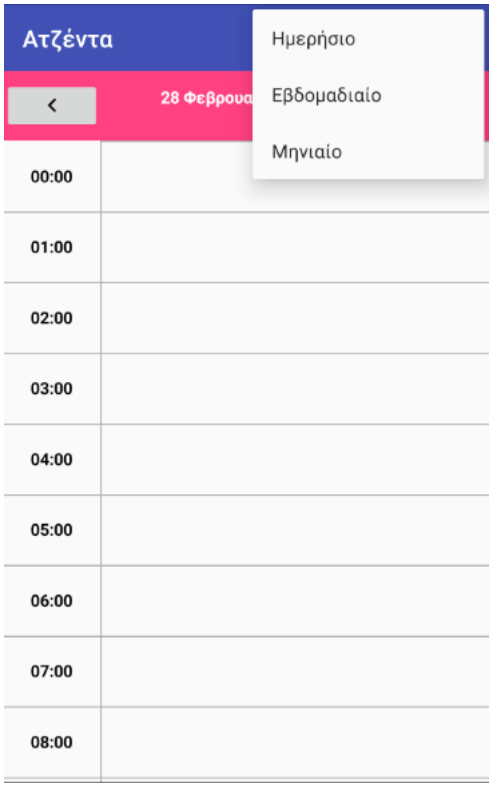
4.3.6.2 – Εβδομαδιαία προβολή γεγονότων

Σε αυτό το activity προβάλλονται τα γεγονότα ανά εβδομάδα (εικόνα 50).



Εικόνα 50 – UI εβδομαδιαίας προβολής γεγονότων

Χρησιμοποιώντας το δεξί και αριστερό βελάκι μπορεί ο χρήστης να μεταφέρεται σε προηγούμενη και επόμενη αντίστοιχα εβδομάδα. Επίσης, πατώντας το κουμπί «Σήμερα» γίνεται μεταφορά στη τρέχουσα εβδομάδα. Για να ανοίξει αυτό το activity θα πατήσουμε τις τρεις τελίτσες οι οποίες βρίσκονται πάνω δεξιά δίπλα από το εικονίδιο της προσθήκης γεγονότος στο activity της προηγούμενης υπό ενότητας (εικόνα 51).



Εικόνα 51 – Μετάβαση στα άλλα activities

4.3.6.2.1 – Αρχικοποίηση εβδομάδας και φόρτωση γεγονότων

Όταν ξεκινάει το activity, γίνονται κάποιες διαδικασίες έτσι, ώστε να φορτωθεί η τρέχουσα εβδομάδα. Πριν ξεκινήσει η φόρτωση, γίνεται έλεγχος για το ποια μέρα είναι σήμερα (Δευτέρα, Τρίτη, κτλ) έτσι, ώστε η σημερινή μέρα να ξεχωρίζει από τις άλλες φέροντας διαφορετικό χρώμα (άσπρο) (απόσπασμα κώδικα 4.3.6.2.1.1) [49][62].

```
01.  private void setWeek() {
02.          String day = calendarForNextPreviousWeek.getTime().toString().substring(0, 3);
03.          Calendar cal = (Calendar) calendarForNextPreviousWeek.clone();
04.
05.          if (day.equals("Mon")) {
06.              if (currentDateString.equals(dateFormat.format(cal.getTime()))) {
07.                  mondayDateTextView.setTextColor(Color.parseColor("#f2eeed"));
08.                  setDefaultTextColorInTextView = mondayDateTextView;
09.              }
10.              setDays(cal);
11.          } else if (day.equals("Tue")) {
12.              if (currentDateString.equals(dateFormat.format(cal.getTime()))) {
13.                  tuesdayDateTextView.setTextColor(Color.parseColor("#f2eeed"));
14.                  setDefaultTextColorInTextView = tuesdayDateTextView;
15.              }
16.              cal.add(Calendar.DAY_OF_MONTH, -1);
17.              setDays(cal);
18.          }
```

Απόσπασμα κώδικα 4.3.6.2.1.1

(Το παραπάνω απόσπασμα κώδικα δεν είναι ολόκληρο για λόγους εξοικονόμησης χώρου. Εξάλλου και στις άλλες μέρες επαναλαμβάνεται το ίδιο όπως και στις μέρες που φαίνονται με ελάχιστες αλλαγές οι οποίες υπάρχουν στις δύο μέρες που φαίνονται παραπάνω).

Αφ' ότου ορίσουμε με άσπρο χρώμα τη σημερινή μέρα, τοποθετούμε στη μεταβλητή «setDefaultTextColorInTextView» ποια μέρα είναι με άσπρο έτσι, ώστε αν μεταβεί ο χρήστης σε προηγούμενη ή επόμενη εβδομάδα η ίδια μέρα να μην είναι επισημασμένη με άσπρο. Έπειτα, αν η σημερινή μέρα δεν είναι η «Δευτέρα» κάνουμε να γίνει έτσι, ώστε να είναι εφικτό να δημιουργηθεί η εβδομάδα στο UI (επειδή στην εφαρμογή αυτή ορίζεται ως πρώτη μέρα της εβδομάδας η Δευτέρα) και στη συνέχεια καλούμε τη μέθοδο «setDays» η οποία θα τοποθετήσει τις μέρες τις εβδομάδας στο UI. Το περιεχόμενο αυτής της μεθόδου φαίνεται στο παρακάτω απόσπασμα κώδικα [40][49][62].

```
01. for (int i = 0; i < textViews.length; i++) {
02.     textViews[i].setText("" + cal.get(Calendar.DAY_OF_MONTH));
03.
04.     mLayout = relativeLayouts[i];
05.     eventIndex = mLayout.getChildCount();
06.     displayDailyEvents(cal);
07.
08.     daysOfWeek[i] = cal.getTime();
09.     cal.add(Calendar.DAY_OF_MONTH, 1);
10. }
```

Απόσπασμα κώδικα 4.3.6.2.1.2

Στη γραμμή 2 γίνεται η τοποθέτηση των ημερών τις εβδομάδας στο UI. Οι γραμμές 4-6 χρησιμοποιούνται για την τοποθέτηση του/των γεγονότος/γεγονότων στη μέρα όπου ανήκει το καθένα. Ο κώδικας που κάνει τη δημιουργία και την τοποθέτηση των γεγονότων είναι ο παρακάτω [59][60][61][63].

```
01. private void createEventView(int topMargin, int height, String message) {
02.     height -= minusEventFrame;
03.     final TextView mEventView = new TextView(Weekly.this);
04.
05.     RelativeLayout.LayoutParams lParam = new RelativeLayout.LayoutParams(
06.         LinearLayout.LayoutParams.WRAP_CONTENT, LinearLayout.LayoutParams.WRAP_CONTENT);
07.     lParam.addRule(RelativeLayout.ALIGN_PARENT_TOP);
08.     lParam.topMargin = topMargin;
09.
10.     mEventView.setLayoutParams(lParam);
11.     mEventView.setPadding(24, 0, 24, 0);
12.     mEventView.setHeight(height);
13.     mEventView.setGravity(0x11);
14.     mEventView.setTextColor(Color.parseColor("#ffffff"));
15.     mEventView.setText(message);
16.     mEventView.setBackgroundColor(Color.parseColor("#3F51B5"));
17.     mEventView.setElevation(1.0 f);
18.
19.     String mLayouts[] = (mLayout.toString()).split("/");
20.     relativeLayoutList.add(mLayouts[1].substring(0, mLayouts[1].indexOf('}')));
21.     mLayout.addView(mEventView, eventIndex - 1);
22.     minusEventFrame = 0;
23. }
```

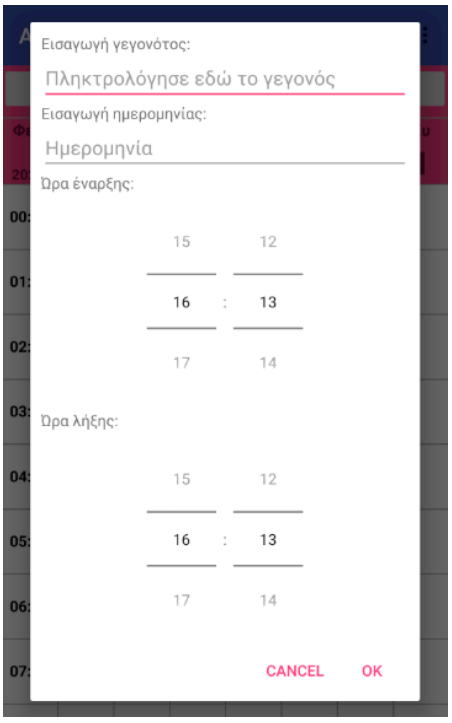
Απόσπασμα κώδικα 4.3.6.2.1.3

Το παραπάνω απόσπασμα κώδικα για τη δημιουργία και την τοποθέτηση γεγονότων στο UI είναι σχεδόν το ίδιο με τον αντίστοιχο κώδικα δημιουργίας και τοποθέτησης γεγονότων στην ημερήσια προβολή. Μια διαφορά υπάρχει στο relative layout. Σ' αυτόν τον κώδικα αριστερά από το γεγονός δεν υπάρχει κενό. Επίσης, ορίσαμε το γεγονός να βρίσκεται πάνω από τις γραμμές που διαχωρίζουν τις ημέρες μεταξύ τους, καθώς και τις ώρες. Αυτές είναι οι διαφορές όσον αφορά στο κομμάτι της εμφάνισης. Υπάρχουν όμως και διαφορές που αφορούν στο κομμάτι του κώδικα. Σ' αυτό το απόσπασμα κώδικα έχουμε μια λίστα (γραμμή 20) στην οποία κάθε φορά που γίνεται τοποθέτηση ενός γεγονότος, εισάγουμε σε ποια μέρα έχει τοποθετηθεί έτσι ώστε όταν ο χρήστης μεταβεί σε άλλη εβδομάδα, να ξέρουμε πόσα γεγονότα και σε ποιες μέρες τοποθετήθηκαν, για να τα διαγράψουμε και να εισάγουμε τα γεγονότα της εβδομάδας που έγινε η μετάβαση.

Στη γραμμή 8 του αποσπάσματος κώδικα 4.3.6.2.1.2 γίνεται αποθήκευση των ημερομηνιών της εβδομάδας έτσι, ώστε να μπορεί να επιτευχθεί η συγκεντρωτική προβολή των γεγονότων. Την συγκεντρωτική προβολή των γεγονότων θα την εξηγήσουμε σε επόμενη υποενότητα.

4.3.6.2.2 – Δημιουργία γεγονότων

Η δημιουργία γεγονότων στην εβδομαδιαία προβολή δεν διαφέρει και πολύ με τη δημιουργία γεγονότων στην ημερήσια προβολή. Η μόνη διαφορά είναι ότι σε αυτό το activity υπάρχει ένα επιπλέον πεδίο, το πεδίο της ημερομηνίας (εικόνα 52).



Εικόνα 52 – Προσθήκη γεγονότος, εβδομαδιαία προβολή

Λόγω της ύπαρξης αυτού του επιπλέον πεδίου θα πρέπει κατά την εισαγωγή γεγονότος, να ελέγξουμε αν η ημερομηνία στην οποία επιθυμεί ο χρήστης να το δημιουργήσει βρίσκεται μέσα στην εβδομάδα που βλέπει αυτή τη στιγμή. Αν βρίσκεται θα το εμφανίσει στη κατάλληλη μέρα, αν όχι, απλά θα γίνει αποθήκευση αυτού του γεγονότος στη βάση δεδομένων και στη συνέχεια όταν μεταβεί στην εβδομάδα που το δημιούργησε θα εμφανιστεί (απόσπασμα κώδικα 4.3.6.2.2.1) [40][42][56][64].

```
01. if (!(currentDate.before(setDayMondayForCheckIfSameWeek) || currentDate.after(setDaySundayForCheckIfSameWeek))) {
02.     int currentNumberDay = calend.get(Calendar.DAY_OF_WEEK) - 2;
03.     if (currentNumberDay == -1)
04.         currentNumberDay = 6;
05.     mLayout = relativeLayouts[currentNumberDay];
06.     int eventBlockHeight = getEventTimeFrame(convertStringToDate(dateStartFormatted), convertStringToDate(dateEndFormatted));
07.     displayEventSection(convertStringToDate(dateStartFormatted), eventBlockHeight, editTxtEisagogiGegonotos.getText().toString());
08. }
09. database.addEvent(editTxtEisagogiGegonotos.getText().toString(), dateStartFormatted, dateEndFormatted);
10. alertDialog.dismiss();
11. Toast.makeText(context, "Το γεγονός αποθηκεύτικε με επιτυχία", Toast.LENGTH_LONG).show();
```

Απόσπασμα κώδικα 4.3.6.2.2.1

4.3.6.2.3 – Συγκεντρωτική εμφάνιση, επεξεργασία και διαγραφή γεγονότων

Για να γίνει συγκεντρωτική εμφάνιση όλων των γεγονότων μιας μέρας πατάμε πάνω στο ροζ πλαίσιο που περιβάλλει την ημέρα. Για να γίνει αυτό, θα πρέπει σε όλες τις μέρες τις εβδομάδας να ορίσουμε έναν click listener. Ο ορισμός του listener σε κάθε μέρα της εβδομάδας γίνεται με την χρήση των αποσπασμάτων κώδικα παρακάτω.

```
01. private void setOnClickRelativesHeaders() {
02.     for (int i = 0; i < daysOfWeek.length; i++)
03.         setOnClickRelativeHeader(i);
04. }
```

Απόσπασμα κώδικα 4.3.6.2.3.1

```
01.  private void setOnClickRelativeHeader(final int metr) {
02.      RelativeLayout[] relativeLayoutHeader = {
03.          mondayHeaderRelativeLayout,
04.          tuesdayHeaderRelativeLayout,
05.          wednesdayHeaderRelativeLayout,
06.          thursdayHeaderRelativeLayout,
07.          fridayHeaderRelativeLayout,
08.          saturdayHeaderRelativeLayout,
09.          sundayHeaderRelativeLayout
10.      };
11.      relativeLayoutHeader[metr].setOnClickListener(new View.OnClickListener() {
12.          @Override
13.          public void onClick(View view) {
14.              selectedDate = daysOfWeek[metr];
15.              displayEventFromSelectedDate(daysOfWeek[metr]);
16.          }
17.      });
18.  }
```

Απόσπασμα κώδικα 4.3.6.2.3.2

Συνεχίζοντας, σε περίπτωση που στη συγκεκριμένη μέρα δεν έχει εκχωρηθεί κάποιο γεγονός θα εμφανίσει αντίστοιχο μήνυμα στο χρήστη. Σε περίπτωση όμως, που έχει εκχωρηθεί κάποιο γεγονός θα ανοίξει μια λίστα που να περιέχει όλα τα γεγονότα της συγκεκριμένης μέρας. Πατώντας πάνω σε κάποιο γεγονός, δίνεται η δυνατότητα στο χρήστη να επεξεργαστεί το περιεχόμενο του (περιγραφή, ημερομηνία, ώρα έναρξης και ώρα λήξης). Όταν ολοκληρώσει την επεξεργασία ο χρήστης και πατήσει «OK», γίνεται αποθήκευση στη βάση δεδομένων και στη συνέχεια ενημερώνεται το UI της εφαρμογής (απόσπασμα κώδικα 4.3.6.2.3.3) [42][54][56].

```
01.  database.updateEvent(event.getId(), "" + editTxtEisagogiGegonotos.getText(), dateStartFormatted, dateEndFormatted);
02.  Toast.makeText(context, "Το γεγονός ενημερώθηκε με επιτυχία", Toast.LENGTH_LONG).show();
03.  alertDialog.dismiss();
04.  removeRelativesLayouts();
05.  calendarForNextPreviousWeek.setTime(selectedDate);
06.  setWeek();
```

Απόσπασμα κώδικα 4.3.6.2.3.3

Για να γίνει ενημέρωση του UI με το/τα επεξεργασμένο/α γεγονότα θα πρέπει πρώτα να γίνει διαγραφή των υπαρχόντων. Η διαγραφή των υπαρχόντων γίνεται με την κλίση της μεθόδου «removeRelativesLayouts» (απόσπασμα κώδικα 4.3.6.2.3.3 γραμμή 4). Η μέθοδος αυτή φαίνεται παρακάτω.

```
01.  for (String relativeLayout: relativeLayoutList)
02.      if (relativeLayout.equals("mondayRelativeLayout"))
03.          mondayRelativeLayout.removeViewAt(eventIndex - 1);
04.      else if (relativeLayout.equals("tuesdayRelativeLayout"))
05.          tuesdayRelativeLayout.removeViewAt(eventIndex - 1);
06.      else if (relativeLayout.equals("wednesdayRelativeLayout"))
07.          wednesdayRelativeLayout.removeViewAt(eventIndex - 1);
08.      else if (relativeLayout.equals("thursdayRelativeLayout"))
09.          thursdayRelativeLayout.removeViewAt(eventIndex - 1);
10.      else if (relativeLayout.equals("fridayRelativeLayout"))
11.          fridayRelativeLayout.removeViewAt(eventIndex - 1);
12.      else if (relativeLayout.equals("saturdayRelativeLayout"))
13.          saturdayRelativeLayout.removeViewAt(eventIndex - 1);
14.      else if (relativeLayout.equals("sundayRelativeLayout"))
15.          sundayRelativeLayout.removeViewAt(eventIndex - 1);
```

Απόσπασμα κώδικα 4.3.6.2.3.4

Στη συνέχεια εκτελείται η μέθοδος «setWeek» η οποία κάνει τις απαραίτητες ενέργειες (τις οποίες τις εξηγήσαμε παραπάνω), για να γίνει εκ νέου φόρτωση των ενημερωμένων γεγονότων.

4.3.6.2.4 – Μετάβαση σε προηγούμενη / επόμενη εβδομάδα και στη τρέχουσα

Για να μεταβούμε είτε σε προηγούμενη είτε σε επόμενη εβδομάδα χρησιμοποιούμε τα κουμπιά με το αριστερό και το δεξί βελάκι. Ο κώδικας της μετάβασης σε προηγούμενη εβδομάδα φαίνεται στο παρακάτω απόσπασμα [49].

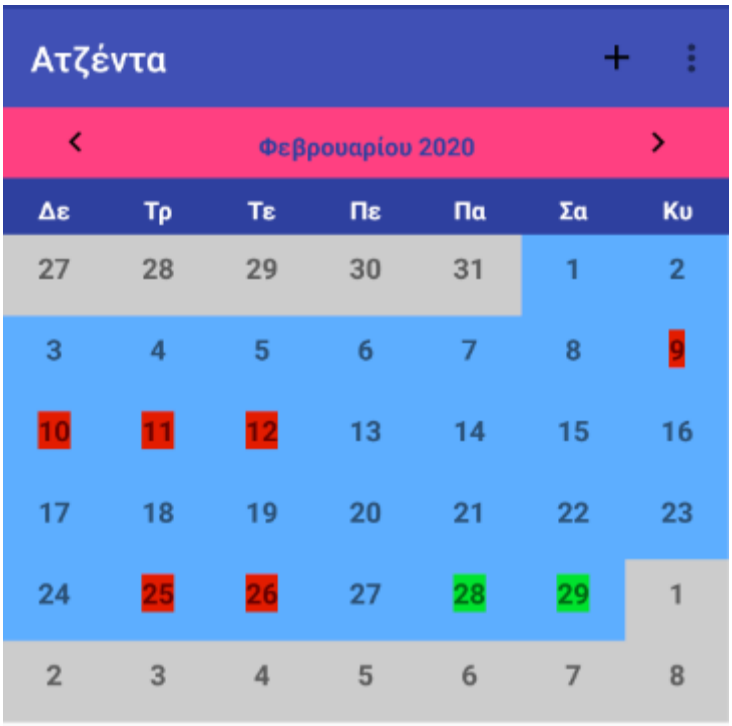
```
01.  private void previousWeek() {
02.      removeRelativesLayouts();
03.
04.      calendarForNextPreviousWeek.add(Calendar.WEEK_OF_MONTH, -1);
05.      setDefaultTextColorInTextView.setTextColor(defaultTextColor);
06.      setWeek();
07.  }
```

Απόσπασμα κώδικα 4.3.6.2.4.1

Στην αρχή διαγράφουμε τα γεγονότα της εβδομάδας που παρουσιάζεται αυτή τη στιγμή, έπειτα κάνουμε την ημερομηνία συν μια εβδομάδα, μετά στο text view της σημερινής μέρας ορίζουμε το default χρώμα έτσι, ώστε η Παρασκευή για παράδειγμα που στη τρέχουσα εβδομάδα έχει άσπρο χρώμα να μην έχει και στην εβδομάδα που θα γίνει η μετάβαση (προηγούμενη στο παράδειγμα μας). Στη συνέχεια εκτελείται η μέθοδος «setWeek», για να φορτώσει τα γεγονότα της εβδομάδας στην οποία έγινε η μετάβαση. Όσο για τη μετάβαση σε επόμενη εβδομάδα ή στη τρέχουσα εβδομάδα, δεν θα εξηγήσουμε τους κώδικες, γιατί έχουν την ίδια λογική με τον παραπάνω.

4.3.6.3 – Μηνιαία προβολή γεγονότων

Σε αυτό το activity παρέχεται η δυνατότητα στον χρήστη να δει τα γεγονότα σε μηνιαία προβολή (εικόνα 53).



Εικόνα 53 – UI μηνιαία προβολής γεγονότων

Στη παραπάνω εικόνα βλέπουμε ήδη κάποια εκχωρημένα γεγονότα. Επίσης, παρατηρούμε ότι κάποιες μέρες είναι επισημασμένες με κόκκινο και κάποιες με πράσινο. Αυτές που είναι με κόκκινο, δηλώνουν ότι το/τα γεγονός/τα που είναι δημιουργημένο/α εκεί έχει/έχουν λήξει. Αντίθετα, αυτές που είναι επισημασμένες με πράσινο, δηλώνουν ότι έστω και ένα γεγονός που έχει δημιουργηθεί εκεί είναι ενεργό.

4.3.6.3.1 – Δημιουργία μηνιαίου ημερολογίου

Το ημερολόγιο που παρέχει το android studio δεν μας δίνει πολλές δυνατότητες. Το μόνο που μπορείς να κάνεις είναι να περιηγηθείς στους μήνες και να επιλέξεις ημερομηνία. Ενώ στο ημερολόγιο που είδαμε στη παραπάνω εικόνα το οποίο το δημιουργήσαμε εμείς, μπορούμε να αλλάξουμε τα χρώματα των ημερών, να ορίσουμε τι θα γίνει αν ο χρήστης πατήσει πάνω σε κάποια μέρα κτλ. Στο παρακάτω κώδικα (γραμμές 15-19) γίνεται τοποθέτηση του ημερολογίου στο layout της εφαρμογής [65].

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.dimos_ssd.myapplication3.Agenda.Agenda"
    android:orientation="vertical">

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="@color/colorPrimary" />

    <com.example.dimos_ssd.myapplication3.Agenda.CalendarCustomView
        android:id="@+id/custom_calendar"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    </com.example.dimos_ssd.myapplication3.Agenda.CalendarCustomView>
</LinearLayout>
```

Απόσπασμα κώδικα 4.3.6.3.1.1

Εφόσον γίνει η τοποθέτηση του ημερολογίου στο layout εκτελείται ο δομητής της κλάσης «CalendarCustomView» που φαίνεται παρακάτω.

```java
public CalendarCustomView(Context context, AttributeSet attrs) {
    super(context, attrs);
    this.context = context;
    initializeUILayout();
    setUpCalendarAdapter();
    setPreviousButtonClickEvent();
    setNextButtonClickEvent();
    setGridCellClickEvents();
    moveToday();
}
```

Απόσπασμα κώδικα 4.3.6.3.1.2

Στο δομητή τοποθετούμε τις μεθόδους οι οποίες αρχικοποιούν τις ενέργειες που γίνονται στο UI (αρχικοποίηση πεδίων layout, δημιουργία ημερών ημερολογίου, μετακίνηση σε επόμενο προηγούμενο μήνα, ορισμός ενέργειας όταν γίνεται κλικ σε μια μέρα και επαναφορά στο τρέχοντα μήνα). Αφού είπαμε κάποια εισαγωγικά για το custom ημερολόγιο, πάμε τώρα να περιγράψουμε με ποιο τρόπο «γεμίζουμε» τις μέρες του μήνα. Το γέμισμα αυτό είναι υπεύθυνη να το κάνει η μέθοδος «setUpCalendraAdapter». Η μέθοδος αυτή φαίνεται στο παρακάτω απόσπασμα κώδικα [40][49][62].

```
01. public void setUpCalendarAdapter(){
02.         List<Date> dayValueInCells = new ArrayList<Date>();
03.         mQuery = new DatabaseQuery(context);
04.         List<EventObjects> mEvents = mQuery.getAllFutureEvents();
05.         Calendar mCal = (Calendar)cal.clone();
06.         mCal.set(Calendar.DAY_OF_MONTH, 1);
07.
08.         int firstDayOfTheMonth = mCal.get(Calendar.DAY_OF_WEEK) + 5;
09.         if (firstDayOfTheMonth > 6)
10.             firstDayOfTheMonth = mCal.get(Calendar.DAY_OF_WEEK) -2;
11.         mCal.add(Calendar.DAY_OF_MONTH, -firstDayOfTheMonth);
12.
13.         while(dayValueInCells.size() < MAX_CALENDAR_COLUMN){
14.             dayValueInCells.add(mCal.getTime());
15.             mCal.add(Calendar.DAY_OF_MONTH, 1);
16.         }
17.         String sDate = formatter.format(cal.getTime());
18.         currentDate.setText(sDate);
19.         mAdapter = new GridAdapter(context, dayValueInCells, cal, mEvents);
20.         calendarGridView.setAdapter(mAdapter);
21. }
```

Απόσπασμα κώδικα 4.3.6.3.1.3

Αρχικά γίνεται request στη βάση δεδομένων, για να μας επιστρέψει όλα τα γεγονότα (τα οποία θα τα χρειαστούμε παρακάτω). Στη συνέχεια, οι γραμμές 8-11 είναι υπεύθυνες, για να παρουσιάζουν τις μέρες του μήνα στη πρώτη γραμμή πάντα και ότι περισσεύει να εμφανίζουν από τον προηγούμενο και τον επόμενο. Παρακάτω στο while γίνεται τοποθέτηση των ημερών που θα εμφανιστούν στο ημερολόγιο σε μια λίστα. Τέλος, με τη βοήθεια της κλάσης του adapter θα τοποθετήσουμε τις μέρες αυτές στο ημερολόγιο, θα ορίσουμε διαφορετικό χρώμα για τις μέρες του προηγούμενου και του επόμενου μήνα και διαφορετικό για τις μέρες του μήνα που παρουσιάζεται. Ακόμη, θα γίνει έλεγχος σε όλα τα γεγονότα που είναι δημιουργημένα για το αν έχουν λήξει ή όχι έτσι, ώστε οι μέρες οι οποίες είναι δημιουργημένα να επισημανθούν με τα κατάλληλα χρώματα (πράσινο→έστω ένα ενεργό, κόκκινο→έχει/έχουν λήξει). Παρακάτω θα δούμε τους κώδικες στη κλάση του adapter που υλοποιούν αυτά που είπαμε παραπάνω. Στο απόσπασμα κώδικα που ακολουθεί θα δούμε με ποιο τρόπο θα γίνει η τοποθέτηση των ημερών στο ημερολόγιο, με ποιο τρόπο παίρνεται η απόφαση για το χρώμα των ημερών, καθώς και τη δημιουργία view (απόσπασμα κώδικα 4.3.6.3.1.4) [40][54][66].

```
01. @Override
02. public View getView(int position, View convertView, ViewGroup parent) {
03.         Date mDate = monthlyDates.get(position);
04.         Calendar dateCal = Calendar.getInstance();
05.         dateCal.setTime(mDate);
06.
07.         int dayValue = dateCal.get(Calendar.DAY_OF_MONTH);
08.         int displayMonth = dateCal.get(Calendar.MONTH) + 1;
09.         int displayYear = dateCal.get(Calendar.YEAR);
10.         int currentMonth = currentDate.get(Calendar.MONTH) + 1;
11.         int currentYear = currentDate.get(Calendar.YEAR);
12.
13.         View view = convertView;
14.         if (view == null)
15.             view = mInflater.inflate(R.layout.agenda_single_cell_layout, parent, false);
16.
17.         if (displayMonth == currentMonth && displayYear == currentYear)
18.             view.setBackgroundColor(Color.parseColor("#5eaeff"));
19.         else
20.             view.setBackgroundColor(Color.parseColor("#cccccc"));
21.         TextView cellNumber = (TextView) view.findViewById(R.id.calendar_date_id);
22.         cellNumber.setText(String.valueOf(dayValue));
```

Απόσπασμα κώδικα 4.3.6.3.1.4

Στις γραμμές 13-15 γίνεται η δημιουργία των view (μέρες τις εβδομάδας), τα οποία επειδή τα αναλύσαμε σε προηγούμενη ενότητα δεν θα τα αναλύσουμε και σε αυτήν. Στη συνέχεια, γίνεται έλεγχος για το αν η μέρα που πάμε να προσθέσουμε στο ημερολόγιο ανήκει στο μήνα και στο χρόνο προβολής. Αν ναι, θα επισημανθεί με σιέλ, αν όχι, θα επισημανθεί με ανοιχτό γκρι (γραμμές 17-20). Έπειτα, γίνεται η τοποθέτηση της μέρας στο ημερολόγιο (γραμμές 21-22). Στον επόμενο κώδικα, γίνεται έλεγχος όλων των δημιουργημένων γεγονότων έτσι, ώστε οι μέρες στις οποίες είναι τοποθετημένα να λάβουν το αντίστοιχο χρώμα [40][54].

```
01. Calendar eventCalendar = Calendar.getInstance();
02. for (int i = 0; i < allEvents.size(); i++) {
03.     eventCalendar.setTime(allEvents.get(i).getDate());
04.     if (dayValue == eventCalendar.get(Calendar.DAY_OF_MONTH) &&
05.         displayMonth == eventCalendar.get(Calendar.MONTH) + 1 &&
06.         displayYear == eventCalendar.get(Calendar.YEAR)) {
07.         if (chechExpireEvents(allEvents, allEvents.get(i).getEnd())) {
08.             cellNumber.setBackgroundColor(Color.parseColor("#0ee32e"));
09.             break;
10.         } else
11.             cellNumber.setBackgroundColor(Color.parseColor("#e31c0e"));
12.     }
13. }
```

Απόσπασμα κώδικα 4.3.6.3.1.5

Αρχικά τοποθετούμε την ημερομηνία του πρώτου γεγονότος που μας ήρθε από τη βάση στη μεταβλητή «eventCalendar» (γραμμή 3), έπειτα ελέγχουμε αν είναι ίση με κάποια ημερομηνία απ' αυτές που τοποθετήθηκαν στη λίστα (απόσπασμα κώδικα 4.3.6.3.1.3 γραμμή 14). Σε περίπτωση που είναι, θα γίνει νέος έλεγχος για το αν τα γεγονότα σε αυτή την ημερομηνία είναι όλα ενεργά ή όλα έχουν λήξει ή έχει και ενεργά και κάποια που έχουν λήξει. Τον έλεγχο αυτό θα τον κάνει ο παρακάτω κώδικας [33][40][58][62][64].

```
01. private boolean chechExpireEvents(List<EventObjects> events, Date dateEnd) {
02.     List<Date> dates = new ArrayList<> ();
03.     SimpleDateFormat simpleDateFormat = new SimpleDateFormat("d-MM-yyyy");
04.
05.     int count = 0;
06.     for (int i = 0; i < events.size(); i++)
07.         if (simpleDateFormat.format(events.get(i).getEnd()).equals(simpleDateFormat.format(dateEnd))) {
08.             dates.add(dateEnd);
09.             count++;
10.         }
11.     Calendar calendar = Calendar.getInstance();
12.     if (count > 1) {
13.         boolean checkEvent = false;
14.         for (int i = 0; i < dates.size(); i++)
15.             if (dates.get(i).after(calendar.getTime())) {
16.                 checkEvent = true;
17.                 break;
18.             }
19.         return checkEvent;
20.     }else
21.         return dateEnd.after(calendar.getTime());
22. }
```

Απόσπασμα κώδικα 4.3.6.3.1.6

Εν ολίγοις, ο παραπάνω κώδικας ελέγχει το/τα γεγονός/τα το/τα οποίο/α έχει/έχουν δημιουργηθεί σε μία συγκεκριμένη μέρα για το αν είναι ενεργά ή αν έχουν λήξει όλα ή κάποια απ' αυτά. Στη περίπτωση που έστω και ένα γεγονός είναι ενεργό επιστρέφει «true» και η μέρα σηματοδοτείται με πράσινο χρώμα. Στην αντίθετη περίπτωση, η μέρα σηματοδοτείται με κόκκινο χρώμα.

4.3.6.3.2 – Δημιουργία, συγκεντρωτική προβολή, επεξεργασία, διαγραφή γεγονότος

Η δημιουργία του γεγονότος όπως αναφέραμε και σε προηγούμενες υποενότητες γίνεται πατώντας το εικονίδιο της προσθήκης πάνω δεξιά. Συνεχίζοντας, πατώντας πάνω σε μια μέρα στο ημερολόγιο γίνεται προβολή όλων των γεγονότων που έχουν δημιουργηθεί στη συγκεκριμένη μέρα. Η προβολή αυτή γίνεται σε λίστα με τον ίδιο τρόπο που περιγράψαμε και στα άλλα δύο activities της ατζέντας. Επίσης, πατώντας πάνω σε ένα γεγονός ο χρήστης μπορεί να το επεξεργαστεί. Σε περίπτωση που θέλει να διαγράψει κάποιο, αρκεί να πατήσει παρατεταμένα πάνω του και στο μήνυμα που θα εμφανιστεί να πατήσει «ΝΑΙ». Για να είναι όμως εφικτό να εμφανιστεί η λίστα με τα γεγονότα, θα πρέπει να εκτελεστεί ο παρακάτω κώδικας.

```
01. public void setGridCellClickEvents() {
02.     calendarGridView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
03.         @Override
04.         public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
05.             selectedDate = (Date) parent.getItemAtPosition(position);
06.             displayEventFromSelectedDate(selectedDate);
07.         }
08.     });
09. }
```

Απόσπασμα κώδικα 4.3.6.3.2.1

Ο κώδικας αυτός ορίζει σε κάθε μέρα του ημερολογίου την ενέργεια click listener έτσι, ώστε όταν γίνει κλικ σε μια μέρα, να ξέρουμε ποια είναι και στη συνέχεια να εμφανίσουμε τα γεγονότα στη μέρα αυτή. Την εμφάνιση των γεγονότων σε μια συγκεκριμένη μέρα την αναλαμβάνει η μέθοδος στη γραμμή 6. Δεν θα την εξηγήσουμε, γιατί την εξηγήσαμε σε προηγούμενη υποενότητα.

4.3.6.3.3 – Μετάβαση σε προηγούμενο / επόμενο μήνα και στον τρέχοντα

Η μετάβαση σε προηγούμενο και επόμενο μήνα γίνεται με τα βελάκια αριστερά και δεξιά. Επίσης, η μετάβαση στο τρέχοντα μήνα γίνεται πατώντας πάνω στο μήνα. Επειδή στα δύο προηγούμενα activities περιγράψαμε τη μετάβαση σε επόμενο και προηγούμενο, σε αυτό το activity θα περιγράψουμε τη μετάβαση στο τρέχοντα (μήνα). Ο κώδικας που το κάνει αυτό είναι αυτός που ακολουθεί [40].

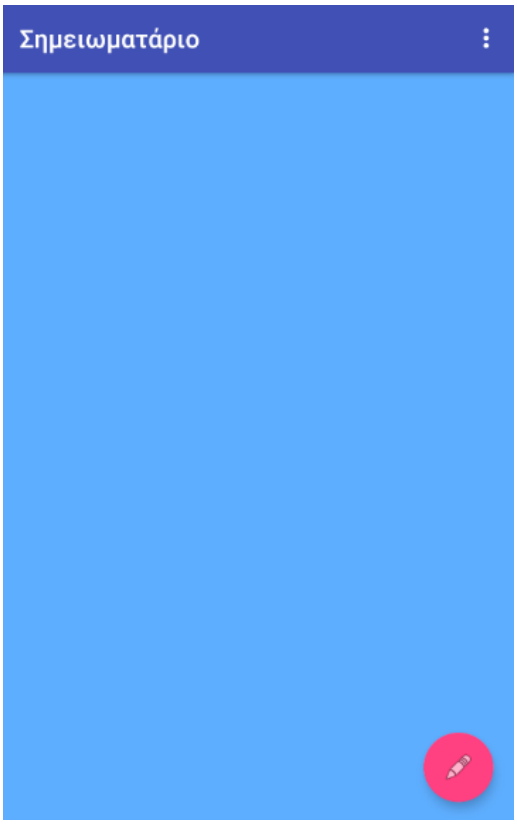
```
01. private void moveToday() {
02.     currentDate.setOnClickListener(new OnClickListener() {
03.         @Override
04.         public void onClick(View view) {
05.             cal = Calendar.getInstance();
06.             setUpCalendarAdapter();
07.         }
08.     });
09. }
```

Απόσπασμα κώδικα 4.3.6.3.2.2

Ο παραπάνω κώδικας ορίζει την τωρινή ημερομηνία στη μεταβλητή «cal» και στη συνέχεια εκτελεί τη μέθοδο που φαίνεται στη γραμμή 6, για να γίνει ξανά η φόρτωση του ημερολογίου με τον τρέχοντα μήνα.

4.3.7 – Σημειωματάριο

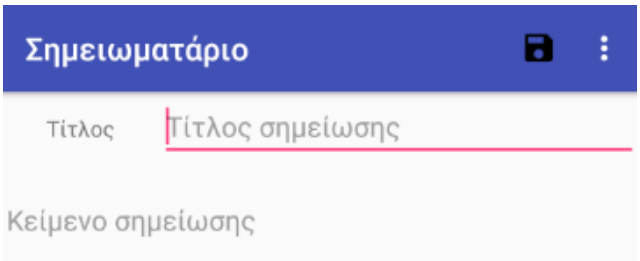
Σε αυτή την υπηρεσία που παρέχει η εφαρμογή ο χρήστης μπορεί να κρατήσει σημείωση για οτιδήποτε επιθυμεί. Όταν ανοίξει το activity της υπηρεσίας αυτής έχουμε το αποτέλεσμα που φαίνεται στην εικόνα παρακάτω.



Εικόνα 54 – UI σημειωματάριου

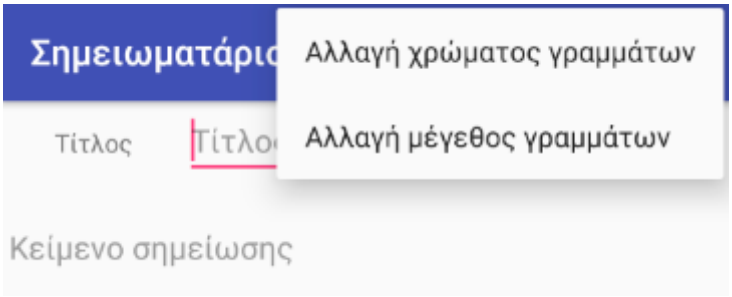
4.3.7.1 – Δημιουργία σημείωσης

Για να δημιουργήσει μια σημείωση ο χρήστης αρκεί να πατήσει στο εικονίδιο με το μολύβι κάτω δεξιά. Όταν πατήσει πάνω στο εικονίδιο αυτό θα ανοίξει το activity που φαίνεται στην εικόνα 55.



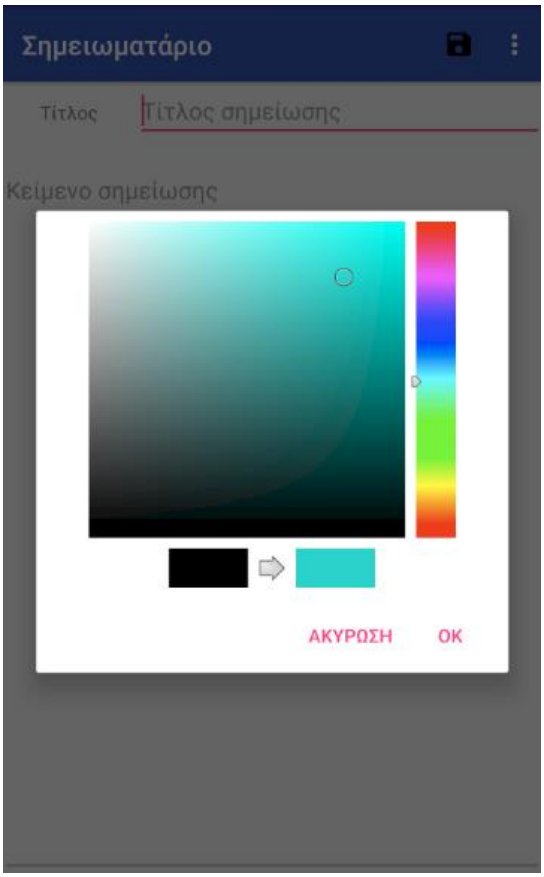
Εικόνα 55 – UI δημιουργίας σημείωσης

Σε αυτό το activity ο χρήστης γράφει το τίτλο της σημείωσης καθώς και το περιεχόμενο της. Επίσης, του παρέχεται η δυνατότητα να αλλάξει το χρώμα καθώς και το μέγεθος των γραμμάτων στο περιεχόμενο της σημείωσης. Αυτό μπορεί να το κάνει πατώντας τις τρεις τελίτσες πάνω δεξιά (εικόνα 56).



Εικόνα 56 – Μενού αλλαγής χρώματος και μεγέθους

Όταν πατήσει στην επιλογή της αλλαγής χρώματος, θα εμφανιστεί η παλέτα χρωμάτων που βλέπουμε στην εικόνα 57.



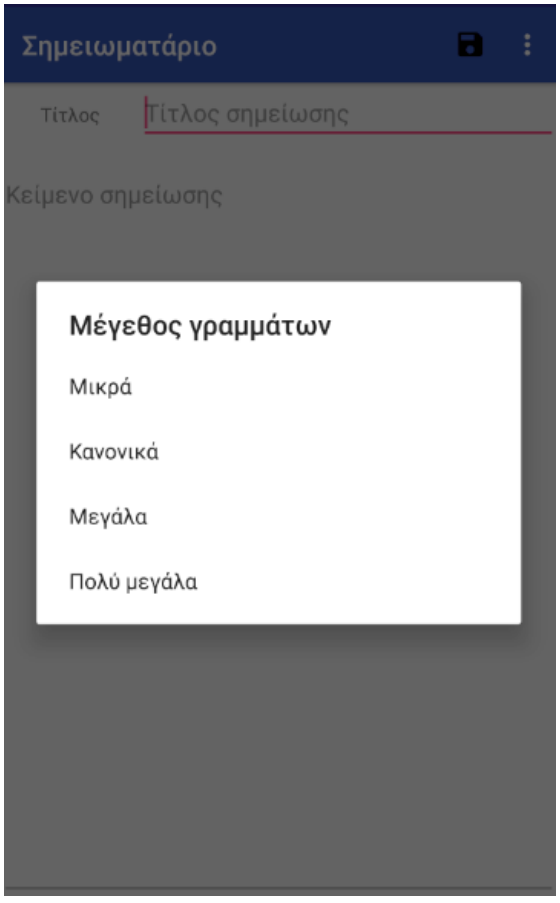
Εικόνα 57 – Παλέτα χρωμάτων

Η εμφάνιση της παλέτας χρωμάτων γίνεται με τη χρήση του παρακάτω κώδικα [26].

```
01. AmbilWarnaDialog colorPocker = new AmbilWarnaDialog(context, -16777216, new AmbilWarnaDialog.OnAmbilWarnaListener() {
02.     @Override
03.     public void onCancel(AmbilWarnaDialog dialog) {
04.     }
05.     @Override
06.     public void onOk(AmbilWarnaDialog dialog, int color) {
07.         editTxtContent.setTextColor(color);
08.         textColor = "" + color;
09.     }
10. });
11. colorPocker.show();
```

Απόσπασμα κώδικα 4.3.6.7.1.1

Στη γραμμή 1 εκτός από τη δημιουργία της παλέτας γίνεται και ορισμός του default χρώματος (για τη συγκεκριμένη παλέτα χρώματος το «μαύρο» αντιστοιχίζεται στον αριθμό «-16777216»). Αφ' ότου επιλέξει χρώμα ο χρήστης και πατήσει «OK», γίνεται ορισμός του χρώματος αυτού στο text view που περιέχει το κείμενο της σημείωσης, καθώς και τοποθέτηση του αριθμού χρώματος σε μεταβλητή (γραμμή 8) έτσι, ώστε να αποθηκευτεί στη βάση δεδομένων μαζί με την σημείωση. Ακόμη, όταν πατήσει στην επιλογή για την αλλαγή του μεγέθους των γραμμάτων θα εμφανιστούν οι επιλογές μεγέθους (εικόνα 58).



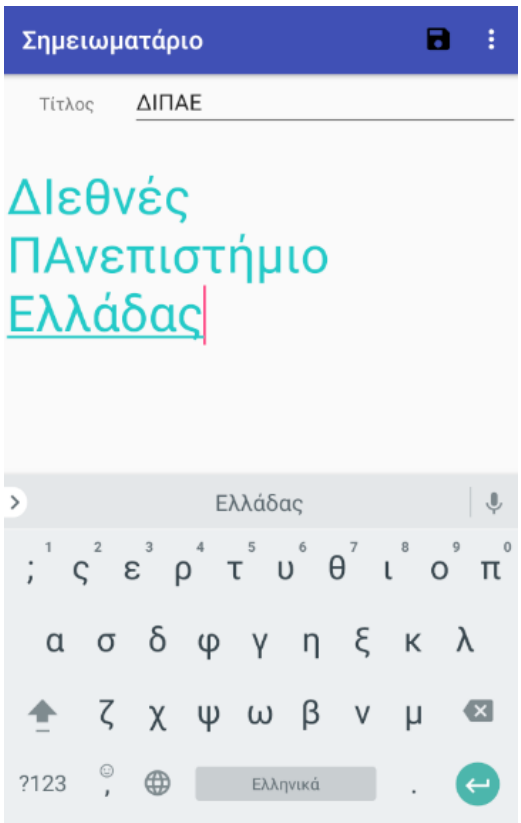
Εικόνα 58 – Επιλογή μεγέθους

Η εμφάνιση της επιλογής μεγέθους γραμμάτων γίνεται με τη χρήση του ακόλουθου κώδικα [67].

```
01. AlertDialog.Builder builder = new AlertDialog.Builder(this);
02. builder.setTitle("Μέγεθος γραμμάτων")
03.     .setItems(new String[] {"Μικρά","Κανονικά","Μεγάλα","Πολύ μεγάλα"},
04.     new DialogInterface.OnClickListener() {
05.         public void onClick(DialogInterface dialog, int which) {
06.             if (which == 0) {
07.                 editTxtContent.setTextSize(15);
08.                 textSize = "15";
09.             } else if (which == 1) {
10.                 editTxtContent.setTextSize(20);
11.                 textSize = "20";
12.             } else if (which == 2) {
13.                 editTxtContent.setTextSize(30);
14.                 textSize = "30";
15.             } else {
16.                 editTxtContent.setTextSize(40);
17.                 textSize = "40";
18.             }
19.         }
20.     });
21. builder.create();
22. builder.show();
```

Απόσπασμα κώδικα 4.3.6.7.1.2

Ανάλογα με την επιλογή του χρήστη, αλλάζει το μέγεθος γραμμάτων στο text view που περιέχει το κείμενο της σημείωσης, καθώς και γίνεται τοποθέτηση του επιλεγμένου μεγέθους σε μεταβλητή έτσι, ώστε να αποθηκευτεί στη βάση δεδομένων μαζί με τη σημείωση. Στη παρακάτω εικόνα φαίνεται το αποτέλεσμα μετά την αλλαγή χρώματος και μεγέθους γραμμάτων (εικόνα 59).



Εικόνα 59 – Επεξεργασμένη σημείωση

Όταν ολοκληρώσει ο χρήστης τη σημείωση, μαζί με τις αλλαγές χρώματος και μεγέθους, σειρά έχει η αποθήκευση της σημείωσης. Για να γίνει αποθήκευση της σημείωσης αρκεί ο χρήστης να πατήσει τη δισκέτα (πάνω δεξιά). Αφ' ότου πατήσει πάνω στη δισκέτα γίνεται αποθήκευση στη βάση δεδομένων του τίτλου, του περιεχομένου της σημείωσης, του χρώματος (αν έχει οριστεί, αλλιώς θα αποθηκευτεί το default) και του μεγέθους των γραμμάτων (αν έχει οριστεί, αλλιώς θα αποθηκευτεί το default). Μετά την ολοκλήρωση της αποθήκευσης, βλέπουμε στο UI της εφαρμογής το αποτέλεσμα που φαίνεται στην εικόνα 60.

Εικόνα 60 – UI εφαρμογής μετά την αποθήκευση σημειώσεων

Στη πρώτη σημείωση βλέπουμε ότι το χρώμα που ορίσαμε στο σώμα της σημείωσης όταν τη δημιουργήσαμε είναι ίδιο με το χρώμα που παρουσιάζεται στο UI. Ακόμη, στη δεύτερη σημείωση παρατηρούμε ότι δίπλα από τον τίτλο και από το σώμα της σημείωσης έχει αποσιωπητικά (…). Αυτό γίνεται επειδή ο τίτλος που έγραψε ο χρήστης όπως και το σώμα της σημείωσης είναι πιο μεγάλο από το επιτρεπτό όριο που ορίσαμε. Το όριο που ορίστηκε φαίνεται στο παρακάτω απόσπασμα κώδικα.

```
01. if (objNote.getTitle().length() >= 15)
02.     length15Title = (objNote.getTitle()).substring(0, 11) + "...";
03. else
04.     length15Title = objNote.getTitle();
05.
06. if (objNote.getContentNote().length() >= 30)
07.     length30Content = (objNote.getContentNote().substring(0, 26) + "...");
08. else
09.     length30Content = objNote.getContentNote();
```

Απόσπασμα κώδικα 4.3.6.7.1.3

4.3.7.2 – Φόρτωση, επεξεργασία, διαγραφή σημειώσεων

Η φόρτωση σημειώσεων ακολουθεί την ίδια λογική με την φόρτωση εργασιών (υποενότητα 4.3.5). Για να γίνει η φόρτωση των σημειώσεων χρησιμοποιείται adapter. Περισσότερες λεπτομέρειες δεν θα πούμε γιατί η εξήγηση έγινε σε προηγούμενες ενότητες. Επίσης, ο χρήστης έχει τη δυνατότητα να επεξεργαστεί τη/τις σημείωση/εις που δημιούργησε κάνοντας κλικ πάνω της/τους. Ούτε γι' αυτό δεν θα πούμε λεπτομέρειες γιατί είπαμε σε προηγούμενες ενότητες. Τέλος, σε περίπτωση που θέλει να διαγράψει μια σημείωση αρκεί να πατήσει παρατεταμένα πάνω της και στο μήνυμα που θα εμφανιστεί να πατήσει «ΝΑΙ».

4.3.7.3 – Ταξινόμηση σημειώσεων

Μια ακόμα δυνατότητα που προσφέρει το activity του σημειωματάριου στο χρήστη είναι η ταξινόμηση των σημειώσεων σε αύξουσα και φθίνουσα σειρά με βάση την ημερομηνία. Για να κάνει ταξινόμηση ο χρήστης θα πρέπει να πατήσει τις τρεις τελίτσες πάνω δεξιά στο UI του σημειωματάριου. Κάνοντας αυτό, θα ανοίξει το μενού με την επιλογή της ταξινόμησης (αύξουσα, φθίνουσα). Επειδή και την ταξινόμηση την εξηγήσαμε σε προηγούμενη ενότητα δεν θα πούμε περισσότερα εδώ.

4.3.8 – Βάση δεδομένων

Κάποιες υπηρεσίες που παρέχει η εφαρμογή χρησιμοποιούν βάση δεδομένων για την αποθήκευση των στοιχείων τους. Οι υπηρεσίες που χρησιμοποιούν τη βάση δεδομένων για αποθήκευση είναι: Ειδοποιήσεις εργασιών, Ατζέντα και Σημειωματάριο. Παρακάτω θα εξηγήσουμε τη δημιουργία βάσης δεδομένων για το activity «Ειδοποιήσεις εργασιών». Σημείωση: η επιλογή δεν έχει κάποια σημασία γιατί και τα άλλα δύο activities με τον ίδιο τρόπο δημιουργούν τη βάση δεδομένων.

Στο παρακάτω απόσπασμα κώδικα βλέπουμε τη δημιουργία της βάσης δεδομένων για τις εργασίες [68].

```
01. public class Database extends SQLiteOpenHelper {
02.     private static final int DATABASE_VERSION = 1;
03.     public static final String DATABASE_NAME = "jobs";
04.
05.     public Database(Context context) {
06.         super(context, DATABASE_NAME, null, DATABASE_VERSION);
07.     }
08.     @Override
09.     public void onCreate(SQLiteDatabase sqLiteDatabase) {
10.         String CREATE_TABLE = "CREATE TABLE \"jobs\" ( " +
11.             "`_id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, " +
12.             "`title` TEXT NOT NULL, " +
13.             "`date` TEXT, " +
14.             "`repeat` INTEGER, " +
15.             "`requestCode` INTEGER)";
16.         sqLiteDatabase.execSQL(CREATE_TABLE);
17.     }
18.     @Override
19.     public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i1) {
20.         sqLiteDatabase.execSQL("DROP TABLE IF EXISTS jobs");
21.         onCreate(sqLiteDatabase);
22.     }
```

Απόσπασμα κώδικα 4.3.8.1

Συνεχίζοντας, στο κομμάτι κώδικα που ακολουθεί βλέπουμε τη δημιουργία μιας κλάσης, η οποία κληρονομείται από την κλάση στην οποία δημιουργούνται τα sql ερωτήματα. Η κλάση που φαίνεται στο απόσπασμα κώδικα 4.3.8.2 [68] είναι ο ενδιάμεσος, θα λέγαμε, ανάμεσα στη κλάση που δημιουργεί τη βάση δεδομένων και την κλάση που δημιουργεί τα sql ερωτήματα. Δηλαδή, δια μέσου αυτής της κλάσης η κλάση που δημιουργεί τα sql ερωτήματα μπορεί να έχει πρόσβαση στη βάση δεδομένων.

```
01. public class DatabaseObject {
02.     private static Database dbHelper;
03.     private SQLiteDatabase db;
04.
05.     public DatabaseObject(Context context) {
06.         dbHelper = new Database(context);
07.         this.dbHelper.getWritableDatabase();
08.         this.db = dbHelper.getReadableDatabase();
09.     }
10.     public SQLiteDatabase getDbConnection() {
11.         return this.db;
12.     }
13.     public void closeDbConnection() {
14.         if (this.db != null)
15.             this.db.close();
16.     }
17. }
```

Απόσπασμα κώδικα 4.3.8.2

Δεν χρειάζεται να αναφέρουμε τις μεθόδους της κλάσης στην οποία δημιουργούνται τα sql ερωτήματα, γιατί αναφέραμε και εξηγήσαμε κάποιες μεθόδους της σε προηγούμενη ενότητα.

Επίλογος

Στο κεφάλαιο παραπάνω περιγράψαμε όλο το πρακτικό κομμάτι αυτής της πτυχιακής. Οι περιγραφές που έλαβαν χώρα ήταν: το server side κομμάτι και η android εφαρμογή. Στο πρώτο περιγράψαμε τα scripts τα οποία είναι τοποθετημένα στον server καθώς και τις ενέργειες που έλαβαν χώρα για να εκτελούνται. Στο δεύτερο περιγράψαμε τις υπηρεσίες που παρέχει η εφαρμογή στο χρήστη όσον αφορά στο να μάθει αν ποτίστηκε το χωράφι του, να δει τον καιρό που κάνει αυτή τη στιγμή σε μία πόλη που έχει επιλέξει ή να δει την πρόγνωση του καιρού σε αυτή τη πόλη, να μαθαίνει τα τελευταία αγροτικά νέα, να δημιουργεί εργασίες για τις οποίες θέλει να τον ενημερώσει η εφαρμογή όταν έρθει ο καιρός τους, να δημιουργεί γεγονότα στο ημερολόγιο της εφαρμογής στα οποία θέλει να παρευρεθεί και να δημιουργεί σημειώσεις στο κινητό του, αντί να πρέπει να κουβαλάει πάντα χαρτί και στυλό μαζί του. Με το κεφάλαιο 4 ολοκληρώθηκε η παρούσα πτυχιακή εργασία, στο επόμενο κεφάλαιο θα συνοψίσουμε όλα όσα ειπώθηκαν στα προηγούμενα κεφάλαια, τα συμπεράσματα που βγάλαμε και τις παρατηρήσεις που κάναμε, καθώς και μελλοντικές επεκτάσεις του πρακτικού κομματιού της πτυχιακής εργασίας.

ΚΕΦΑΛΑΙΟ 5 – Συμπεράσματα και μελλοντικές επεκτάσεις

5.1 – Ανασκόπηση

Ανακεφαλαιώνοντας να πούμε ότι σε αυτήν την πτυχιακή εργασία μιλήσαμε για το Διαδίκτυο των Πραγμάτων (Internet of Things (IoT)), πότε δημιουργήθηκε, πώς λειτουργεί και τι ασφάλεια παρέχει μέχρι στιγμής. Έπειτα αναφέραμε τι είναι η «Έξυπνη Γεωργία», καθώς και το πως οι τεχνολογίες του διαδικτύου των πραγμάτων (IoT) θα συμβάλουν στην ανάπτυξη της. Μιλήσαμε για τα πληροφοριακά συστήματα διοίκησης της γεωργίας, για τη γεωργία ακριβείας, καθώς και για τους γεωργικούς αυτοματισμούς και τη ρομποτική. Επίσης, αναφέραμε τα πλεονεκτήματα και τα μειονεκτήματα των τεχνολογιών αυτών, καθώς και κάποια παραδείγματα αγροτικών μηχανημάτων τα οποία είναι εφοδιασμένα με τεχνολογίες IoT. Στη συνέχεια, αρχίσαμε να φεύγουμε από τα τελείως θεωρητικά και να ακουμπάμε λίγο πιο τεχνικά θέματα όπως είναι η εξήγηση των τεχνολογιών τις οποίες χρησιμοποιήσαμε, για να δημιουργήσουμε το πρακτικό κομμάτι της πτυχιακής εργασίας. Και τέλος, εμβαθύναμε τελείως σε τεχνικά θέματα όπως είναι η εξήγηση του κώδικα ο οποίος χρησιμοποιήθηκε, για να υλοποιηθούν όλες οι υπηρεσίες της εφαρμογής που δημιουργήσαμε.

5.2 – Συμπεράσματα και προτάσεις

Μετά την ολοκλήρωση του πρακτικού κομματιού της πτυχιακής εργασίας, έχουμε να κάνουμε κάποια σχόλια όσο αφορά τη λειτουργία της. Το πιο κύριο σχόλιο είναι ότι ο έλεγχος της υγρασίας του χώματος γίνεται δέκα λεπτά πριν εκτελεστεί το script του ποτίσματος (στην ουσία όταν εκτελεστεί αυτό, πραγματοποιείται ή όχι πότισμα). Μέσα σε αυτά τα δέκα λεπτά ο χρήστης μπορεί να έχει ενημέρωση για την υγρασία του χώματος. Αυτό όπως καταλαβαίνουμε δεν είναι πολύ πρακτικό, γιατί θα πρέπει να έχει το νου του (ο χρήστης) να βάλει υπενθύμιση για το πότε θα έρθει το δεκάλεπτο του ελέγχου της υγρασίας. Ο έλεγχος αυτός γίνεται ανά τρεις ώρες. Αυτό οδηγεί στο εξής πρόβλημα. Σε περίπτωση που θέλει να δει την υγρασία του χώματος στις δυόμιση ώρες για παράδειγμα έτσι, ώστε να πάρει μια απόφαση δεν θα μπορεί να το κάνει. Για τον λόγο αυτό, η πρόταση που έχουμε να κάνουμε σε κάποιον που θέλει να υλοποιήσει μια παρόμοια εφαρμογή είναι να δημιουργήσει έναν ακροατή (daemon) ο οποίος θα τρέχει επ' αόριστον περιμένοντας τον χρήστη να κάνει αίτημα για το ποσοστό της υγρασίας του χώματος έτσι, ώστε να διενεργήσει έλεγχο και να στείλει πίσω την απάντηση οποιαδήποτε στιγμή το επιθυμήσει ο χρήστης και όχι μια μόνο συγκεκριμένη χρονική στιγμή.

Ένα δεύτερο σχόλιο το οποίο αυτήν τη φορά δεν έχει να κάνει με το server side (όπως το παραπάνω) αλλά με την εφαρμογή android είναι η χρήση της γλώσσας προγραμματισμού. Όπως αναφέρθηκε σε προηγούμενα κεφάλαια, η γλώσσα προγραμματισμού που χρησιμοποιήσαμε σε αυτήν την πτυχιακή είναι η java. Ωστόσο, θα συμβουλεύαμε όχι μόνο κάποιον ο οποίος θέλει να υλοποιήσει μια παρόμοια android εφαρμογή με αυτήν που υλοποιήσαμε εμείς για τις ανάγκες της παρούσας πτυχιακής, αλλά γενικά οποιαδήποτε υλοποίηση android εφαρμογής να μην χρησιμοποιήσει αποκλειστικά και μόνο την java, αλλά και την kotlin. Η kotlin είναι μια νέα γλώσσα προγραμματισμού την οποία υποστηρίζει το android studio για δημιουργία εφαρμογών. Μπορεί να είναι καινούρια σε σχέση με την java, αλλά φημίζεται για την δυνατότητα που παρέχει στους προγραμματιστές να δημιουργούν σύντομο και ευανάγνωστο κώδικα. Σύμφωνα με κάποιες δοκιμές που έχουν εφαρμοστεί, με την kotlin γίνεται εξοικονόμηση μέχρι και τις μισές γραμμές κώδικα σε σύγκριση με το αν γράψεις το ίδιο πρόγραμμα σε java. Επίσης, κάποιες λειτουργίες μπορούν να προγραμματιστούν πιο εύκολα σε kotlin παρά σε java. Τελειώνοντας, η kotlin είναι ακόμα σε πρώιμο στάδιο και δεν μπορούμε να πούμε με σιγουριά ότι μπορεί να αντικαταστήσει πλήρως την java, αν και έχουν δημιουργηθεί αρκετές βιβλιοθήκες οι οποίες βοηθούν τους προγραμματιστές να γράψουν ένα πρόγραμμα εύκολα και σε μικρό χρονικό διάστημα. Στο στάδιο όμως που βρίσκεται τώρα η kotlin, καλό είναι όταν γράφουμε ένα πρόγραμμα, να την χρησιμοποιούμε μόνο στο σημείο όπου θα μας βοηθήσει να γράψουμε πιο εύκολα και γρήγορα κώδικα και όχι καθολικά για όλο το project.

5.3 – Μελλοντικές επεκτάσεις

Στις πρώτες ενότητες του κεφαλαίου 4 έγινε αναφορά στο ότι η λήψη της υγρασίας από το χώμα γίνεται με την χρήση ενός νοητού αισθητήρα. Μια μελλοντική επέκταση της εφαρμογής θα μπορούσε να είναι η λήψη της υγρασίας από υπαρκτό αισθητήρα. Επίσης, ακόμα μία επέκταση που θα μπορούσε να εφαρμοστεί είναι να γίνεται ενημέρωση για την υγρασία του χώματος κατ' απαίτηση και όχι μόνο όταν τρέχει το script του ποτίσματος (ανά τρεις ώρες).

Μια άλλη μελλοντική επέκταση που θα μπορούσε να εφαρμοστεί είναι στα activities «Ο Καιρός τώρα» και «Πρόγνωση του καιρού» όταν πάει να προσθέσει πόλη ο χρήστης και αρχίσει να πληκτρολογεί γράμματα να του προτείνει πόλεις οι οποίες ταιριάζουν στα γράμματα που έχει πληκτρολογήσει μέχρι στιγμής.

Μια άλλη μελλοντική επέκταση που θα μπορούσε να υλοποιηθεί είναι στο activity «Ειδοποιήσεις εργασιών». Η επέκταση αυτή, θα δίνει περισσότερες επιλογές στο χρήστη όσον αφορά στο κάθε πότε να επαναλαμβάνεται η εργασία που θα δημιουργήσει.

Μια άλλη μελλοντική επέκταση που θα μπορούσε να υλοποιηθεί είναι στο activity «Ατζέντα». Η επέκταση αυτή θα δίνει τη δυνατότητα στον χρήστη να δημιουργεί ομάδες γεγονότων που η κάθε ομάδα θα έχει διαφορετικό χρώμα το οποίο θα το επιλέγει αυτός. Κι έτσι, όταν ορίσει ένα γεγονός στην ομάδα «εκδήλωση» για παράδειγμα, το γεγονός θα φαίνεται στην ατζέντα με το χρώμα που έχει ορίσει να έχει αυτή η ομάδα γεγονότων.

Μια ακόμα επέκταση που θα μπορούσε να εισαχθεί είναι η δυνατότητα αποθήκευσης στο cloud των γεγονότων, καθώς και των σημειώσεων έτσι, ώστε σε περίπτωση που γίνει κάτι και χαθούν ή χαθεί η συσκευή του χρήστη ή χαλάσει να μπορεί να τα επαναφέρει.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] L. Atzori, A. Iera and G. Morabito, “The Internet of Things: A survey”, Computer Networks, vol. 54, no. 15, Oct. 2010, pp. 2787-2805.

[2] Q. Hassan, “Introduction to the Internet of Things”, Internet of Things A to Z: Technologies and Applications, May 2018, pp. 3, 27-28.

[3] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash, “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications”, IEEE Communications Surveys & Tutorials, vol. 17, no. 4, Jun. 2015, pp. 2347-2376.

[4] M.M. Ogonji, G. Okeyo, J.M. Wafula, “A survey on privacy and security of Internet of Things”, Computer Science Review, vol. 38, Nov. 2020.

[5] O. Elijah, T.A. Rahman, I. Orikumhi, C.Y. Leow, M.N. Hindia, “An Overview of Internet of Things (IoT) and Data Analytics in Agriculture: Benefits and Challenges”, IEEE Internet of Things Journal, vol. 5, no. 5, Oct. 2018, pp. 3758-3773.

[6] B. Ragavi, L. Pavithra, P. Sandhiyadevi, G.K. Mohanapriya, S. Harikirubha, “Smart Agriculture with AI Sensor by Using Agrobot”, 2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, Mar. 2020.

[7] M. Ayaz, M. Ammad-Uddin, Z. Sharif, A. Mansour, El-H.M. Aggoune, “Internet-of-Things (IoT)-Based Smart Agriculture: Toward Making the Fields Talk”, IEEE Access, vol. 7, Aug. 2019, pp. 129551-129583.

[8] Ομάδα γεωπόνων της Farmacon - Farmacon Team, «Έξυπνη γεωργία σημαίνει αυτοματοποιημένη και συνδεδεμένη γεωργία!», FarmaBlog, 2018. [Online]. Available: https://blog.farmacon.gr/katigories/texniki-arthrografia/georgia-akriveias/item/2008-eksypni-georgia-simainei-aftomatopoiimeni-kai-syndedemeni-georgia.

[9] M.S. Mekala, P. Viswanathan, “A Survey: Smart agriculture IoT with cloud computing”, 2017 International conference on Microelectronic Devices, Circuits and Systems (ICMDCS), Vellore, India, Aug. 2017.

[10] S. Namani, B. Gonen, “Smart Agriculture Based on IoT and Cloud Computing”, 2020 3rd International Conference on Information and Computer Technologies (ICICT), San Jose, California, United States of America, Mar. 2020.

[11] M.F. Krafft, “The Debian System: Concepts and Techniques”, United States of America, 2005.

[12] M. Lutz, “Programming Python, Second Edition”, United States of America, 2001.

[13] Ανάβασις – Εκπαιδευτικός Όμιλος, «Η γλώσσα προγραμματισμού Python – Απλή και δυναμική», Ανάβασις – Εκπαιδευτικός Όμιλος. [Online]. Available: https://www.anavasis.gr/blog/i-glwssa-programmatismou-python.

[14] R. Herdianto, E.B. Setiawan, “ANDROID MOBILE APPLICATION DEVELOPMENT FOR FIELD VISIT DOCUMENTATION”.

[15] W. Jackson, “Watch Face Weather Design: Using Weather APIs”, SmartWatch Design Fundamentals, Mar. 2019, pp. 345-383.

[16] Z. Chen, “Java Card Technology for Smart Cards: Architecture and Programmer's Guide”, United States of America, Jun. 2000.

[17] D. Flanagan, “Java in a Nutshell”, United States of America, Mar. 2005.

[18] M. Yener, O. Dundar, “Expert Android Studio”, United States of America, Sep. 2016.

[19] M. Wickham, “Practical Android: 14 Complete Projects on Advanced Techniques and Approaches”, Dallas, Texas, United States of America, 2018.

[20] JournalDev, “Android Picasso Tutorial”. [Online]. Available: https://www.journaldev.com/13759/android-picasso-tutorial.

[21] B. Hammersley, “Content Syndication with RSS”, United States of America, Mar. 2003.

[22] P. Mainkar, “Expert Android Programming: Master skills to build enterprise grade Android applications”, Birmingham, United Kingdom, Sep. 2017.

[23] Developers Android, “Schedule repeating alarms”, 2020. [Online]. Available: https://developer.android.com/training/scheduling/alarms.

[24] S. Komatineni, D. MacLean, S. Hashimi, “Pro Android 3”, New York, United States of America, Apr. 2011.

[25] J. Morris, “Android User Interface Development: Beginner's Guide”, Birmingham, United Kingdom, Feb. 2011.

[26] Coding in Flow, “Color Picker Dialog (AmbilWarna) - Android Studio Tutorial”, Nov. 2017. [Online]. Available: https://codinginflow.com/tutorials/android/ambilwarna-color-picker-dialog.

[27] Stack Overflow, “How to get JSON from webpage into Python script”, Oct. 2012. [Online]. Available: https://stackoverflow.com/questions/12965203/how-to-get-json-from-webpage-into-python-script.

[28] Stack Overflow, “Why can't Python parse this JSON data?”, May 2010. [Online]. Available: https://stackoverflow.com/questions/2835559/why-cant-python-parse-this-json-data.

[29] Stack Abuse, “Reading and Writing JSON to a File in Python”, 2020. [Online]. Available: https://stackabuse.com/reading-and-writing-json-to-a-file-in-python/.

[30] How to Geek, “How to Schedule Tasks on Linux: An Introduction to Crontab Files”, Jul 2017. [Online]. Available: https://www.howtogeek.com/101288/how-to-schedule-tasks-on-linux-an-introduction-to-crontab-files/.

[31] Crontab Guru, “At minute 0 past every 3rd hour”, 2020. [Online]. Available: https://crontab.guru/every-3-hours.

[32] Stack Overflow, “How to check internet access on Android? InetAddress never times out”, Oct. 2009. [Online]. Available: https://stackoverflow.com/questions/1560788/how-to-check-internet-access-on-android-inetaddress-never-times-out?page=1&tab=votes.

[33] W.M. Lee, “Beginning Android 4 Application Development”, United States of America, Jan. 2012.

[34] Android Beginners, “Android Snackbar Example”, Jan. 2018. [Online]. Available: https://medium.com/android-beginners/android-snackbar-example-tutorial-a40aae0fc620.

[35] Android Code Ninja, “Simple Android JSON Parser Example Code with URL and Logcat Output”, Sep. 2015. [Online]. Available: https://www.androidcode.ninja/android-json-parsing-tutorial/.

[36] Stack Overflow, “Android “Only the original thread that created a view hierarchy can touch its views.””, Mar. 2011. [Online]. Available: https://stackoverflow.com/questions/5161951/android-only-the-original-thread-that-created-a-view-hierarchy-can-touch-its-vi.

[37] Android Code Ninja, “How to Send Data From Android to PHP Server? Android Post Request Help”, Nov. 2016. [Online]. Available: https://www.androidcode.ninja/android-http-client/.

[38] Strife's devLog, “Android: How to Send Request (POST) to the Server (Full Application)”, Dec. 2011. [Online]. Available: http://strife.pl/2011/12/android-how-to-send-request-post-to-the-server-full-application/.

[39] G.B. Meike, “Android Concurrency”, United States of America, Jun. 2016.

[40] H. Vasconcelos, “Asynchronous Android Programming”, Birmingham, United Kingdom, Jul. 2016.

[41] W. Jackson, “Pro Android Wearables: Building Apps for Smartwatches”, California, United States of America, Jun. 2015.

[42] Stack Overflow, “Can't create handler inside thread that has not called Looper.prepare()”, Oct. 2010. [Online]. Available: https://stackoverflow.com/questions/3875184/cant-create-handler-inside-thread-that-has-not-called-looper-prepare.

[43] Java T Point, “Android RSS Feed Reader”, 2020. [Online]. Available: https://www.javatpoint.com/android-rss-feed-reader.

[44] Loop Wiki, “Android RSS Feed Reader Application Development”, Oct. 2017. [Online]. Available: https://www.loopwiki.com/application/android-rss-feed-reader-application/.

[45] Stack Overflow, “How to run a certain activity in Android Studio?”, Jul. 2016. [Online]. Available: https://stackoverflow.com/questions/38308161/how-to-run-a-certain-activity-in-android-studio.

[46] Z.R. Mednieks, L. Dornin, G.B. Meike, M. Nakamura, “Programming Android”, United States of America, Sep. 2012.

[47] Codota, “How to use putExtras method in android.content.Intent”, 2020. [Online]. Available: https://www.codota.com/code/java/methods/android.content.Intent/putExtras.

[48] Coding in Flow, “Notifications Tutorial Part 1 - NOTIFICATION CHANNELS - Android Studio Tutorial”, May 2018. [Online]. Available: https://codinginflow.com/tutorials/android/notifications-notification-channels/part-1-notification-channels.

[49] Stack Overflow, “How to set the calendar in android for particular hour”, Apr. 2011. [Online]. Available: https://stackoverflow.com/questions/5770219/how-to-set-the-calendar-in-android-for-particular-hour.

[50] Code Bind, “Android SQLite Database Tutorial (Select, Insert, Update, Delete)”, Aug. 2016. [Online]. Available: http://www.codebind.com/android-tutorials-and-examples/android-sqlite-tutorial-example/.

[51] Tutorials Point, “How to use update command in Android sqlite?”, Mar. 2019. [Online]. Available: https://www.tutorialspoint.com/how-to-use-update-command-in-android-sqlite.

[52] Developers Android, “Cursor”, Feb. 2021. [Online]. Available: https://developer.android.com/reference/android/database/Cursor#getColumnIndexOrThrow(java.lang.String).

[53] Vimsky, “Java LinearLayout.getChildAt方法代碼示例”, 2020. [Online]. Available: https://vimsky.com/zh-tw/examples/detail/java-method-android.widget.LinearLayout.getChildAt.html.

[54] Codata, “How to use setTime method in java.util.Calendar”, 2020. [Online]. Available: https://www.codota.com/code/java/methods/java.util.Calendar/setTime.

[55] Stack Overflow, “How to cancel alarm from AlarmManager”, Mar. 2015. [Online]. Available: https://stackoverflow.com/questions/28922521/how-to-cancel-alarm-from-alarmmanager/28922621.

[56] Stack Overflow, “How to dismiss AlertDialog.Builder?”, Jul. 2012. [Online]. Available: https://stackoverflow.com/questions/11285235/how-to-dismiss-alertdialog-builder.

[57] Tutorials Point, “Java.util.Calendar.setTimeInMillis() Method”, 2020. [Online]. Available: https://www.tutorialspoint.com/java/util/calendar_settimeinmillis.htm.

[58] Android Wave, “Format DateTime in Android”, Dec. 2018. [Online]. Available: https://androidwave.com/format-datetime-in-android/.

[59] Stack Overflow, “How to set RelativeLayout layout params in code not in xml?”, Mar. 2011. [Online]. Available: https://stackoverflow.com/questions/5191099/how-to-set-relativelayout-layout-params-in-code-not-in-xml.

[60] Codata, “How to use RelativeLayout$LayoutParams in android.widget”, 2020. [Online]. Available: https://www.codota.com/code/java/classes/android.widget.RelativeLayout$LayoutParams.

[61] Codata, “How to use setLayoutParams method in android.widget.TextView”, 2020. [Online]. Available: https://www.codota.com/code/java/methods/android.widget.TextView/setLayoutParams.

[62] Stack Overflow, “Using Android Calendar to get the current time”, Nov. 2012. [Online]. Available: https://stackoverflow.com/questions/13290200/using-android-calendar-to-get-the-current-time.

[63] Codata, “How to use setElevation method in android.widget.TextView”, 2020. [Online]. Available: https://www.codota.com/code/java/methods/android.widget.TextView/setElevation.

[64] Developers Android, “Calendar”, Feb. 2021. [Online]. Available: https://developer.android.com/reference/java/util/Calendar.

[65] Medium, “Android Custom Calendar with events”, Jun. 2018. [Online]. Available: https://medium.com/@Patel_Prashant_/android-custom-calendar-with-events-fa48dfca8257.

[66] Mkyong, “Android prompt user input dialog example”, Aug. 2012. [Online]. Available: https://mkyong.com/android/android-prompt-user-input-dialog-example/.

[67] Developers Android, “Adding a list”, Dec. 2019. [Online]. Available: https://developer.android.com/guide/topics/ui/dialogs#AddingAList.

[68] JournalDev, “Android SQLite Database Example Tutorial”, 2020. [Online]. Available: https://www.journaldev.com/9438/android-sqlite-database-example-tutorial.

ΠΑΡΑΡΤΗΜΑ

Στο παρόν παράρτημα θα παραθέσουμε τα κομμάτια κώδικα τα οποία είναι υπεύθυνα για τη δημιουργία της βάσης δεδομένων. Για να έχει τη δυνατότητα μια κλάση να δημιουργεί, να ανοίγει (αν υπάρχει) ή να αναβαθμίζει τη βάση δεδομένων, θα πρέπει να επεκτείνει τη κλάση «SQLiteOpenHelper». Εφόσον γίνει επέκταση της κλάσης αυτής, στη συνέχεια πρέπει να υπερφορτωθούν (να γίνουν «@Override») οι βασικές της μέθοδοι. Οι βασικές μέθοδοι της κλάσης «SQLiteOpenHelper» είναι η «onCreate» και η «onUpgrade» [68].

```
01. @Override
02. public void onCreate(SQLiteDatabase sqLiteDatabase) {
03.     String CREATE_TABLE = "CREATE TABLE \"events\" ( " +
04.         "`_id` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, " +
05.         "`message` TEXT NOT NULL, " +
06.         "`reminder` TEXT NOT NULL, " +
07.         "`end` TEXT )";
08.     sqLiteDatabase.execSQL(CREATE_TABLE);
09. }
```

Απόσπασμα κώδικα 1

Όταν καλείται η μέθοδος «onCreate» για πρώτη φορά, δημιουργείται η βάση δεδομένων. Στο σώμα της μεθόδου τοποθετούμε τον κώδικα ο οποίος θα χρησιμοποιηθεί, για να δημιουργηθεί ο πίνακας ή οι πίνακες (σε περίπτωση που χρειάζεται η εφαρμογή περισσότερους από έναν) οι οποίοι θα αποτελούν τη βάση δεδομένων.

```
01. @Override
02. public void onUpgrade(SQLiteDatabase sqLiteDatabase, int oldVersion, int newVersion) {
03.     sqLiteDatabase.execSQL("DROP TABLE IF EXISTS events");
04.     onCreate(sqLiteDatabase);
05. }
```

Απόσπασμα κώδικα 2

Η μέθοδος «onUpgrade» [68] καλείται όταν πρέπει να αναβαθμιστεί η βάση δεδομένων. Η εφαρμογή (σε περίπτωση που χρειαστεί) θα πρέπει να χρησιμοποιήσει αυτή τη μέθοδο για να σβήσει (drop) πίνακα, να προσθέσει πίνακα ή να κάνει οτιδήποτε άλλο χρειάζεται, για να αναβαθμίσει το σχήμα της βάσης δεδομένων στη νέα του έκδοση. Επίσης, αν θέλουμε να προσθέσουμε νέες στήλες σε έναν υπάρχοντα πίνακα μπορούμε να χρησιμοποιήσουμε το «ALTER TABLE». Σε περίπτωση που μετονομάσουμε ή αφαιρέσουμε στήλες, μπορούμε πάλι να χρησιμοποιήσουμε το «ALTER TABLE», για να αλλάξουμε το όνομα από τον παλιό πίνακα, να δημιουργήσουμε καινούριο και έπειτα να τοποθετήσουμε το περιεχόμενο του παλιού στον καινούριο. Τέλος να πούμε, ότι η μέθοδος αυτή εκτελείται σε ένα transaction. Αυτό σημαίνει ότι σε περίπτωση που προκληθεί εξαίρεση, όλες οι αλλαγές που έγιναν θα αναιρεθούν αυτόματα.