ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΦΥΣΙΚΗΣ

Μ.Δ.Ε στον Ηλεκτρονικό Αυτοματισμό

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Απομακρυσμένος έλεγχος σπιτιού με χρήση εφαρμογής Android

Καρτσωνάκης Αντώνιος

Α.Μ.:2016513

Επιβλέπων καθηγητής: Τσίλης Εμμανουήλ

Αθήνα 2019

Περιεχόμενα

1. Εισαγωγή

Ο όρος έξυπνο σπίτι υπάρχει και είναι ευρύτερα γνωστός εδώ και δεκαετίες. Παρ'όλη ομως την αλματώδη πρόοδο της τεχνολογίας τις δεκαετίες αυτές η χρήση συστημάτων αυτοματισμού σε σπίτια είναι πολύ περιορισμένη, ακόμα και σε πολύ αναπτυγμένες τεχνολογικά χώρες που αυτές οι υπηρεσίες είναι ποιο εύκολα προσβάσιμες και προωθούνται σημαντικά από τεχνολογικούς κολοσούς. Το πρόβλημα εντοπίζεται στο γεγονός οτι υπάρχει μεγάλος κατακερματισμός της συγκεκριμένης αγοράς και δεν ακολουθείται κάποιο εννιαίο πρωτόκολλο, αντιθέτως η κάθε εταιρεία χτίζει αυτά τα συστήματα αυτοματισμού γύρω από τις δικές τις υπηρεσίες κάτι που οδηγεί στην περιορισμένη λειτουργικότητα των συγκεκριμένων συστημάτων και με την πάροδο των ετών στην μειωμένη υποστήριξη των παλαιότερων συσκευών από τις εταιρείες. Επιπλέον, επειδή τα συστήματα αυτά χρησιμοποιούν για τον απομακρυσμένο χειρισμό διαφόρων συσκευών τους server των εκάστοτε εταιριών, υπάρχουν πολλά παραδείγματα με εταιρείες οι οποίες είτε έκλεισαν είτε εξαγοράστηκαν με αποτέλεσμα οι server αυτοί να σταματήσουν να λειτουργούν. Αυτό είχε σαν αποτέλεσμα στην καλύτερη περίπτωση την μείωση των λειτουργιών των συστημάτων και στην χειρότερη την πλήρη απαξίωσή τους, καθώς πολλές φορές απαιτείται σχεδόν μόνιμη σύνδεση στο internet.

Σκοπός της παρούσας πτυχιακής, είναι η δημιουργία ενός συστήματος αυτοματισμού το οποίο θα επιτρέπει τον έλεγχο των ηλεκτρικών και ηλεκτρονικών συσκευών ενός σπιτιού από οπουδήποτε, καθώς και την παρακολούθηση με διάφορους αισθητήρες του τι συμβαίνει στο σπίτι. Θα το κάνει δε αυτό με το μικρότερο δυνατό κόστος και θα είναι ανεξάρτητο από server των οποίων ο χρόνος ζωής καθώς και η δυνατότητα πρόσβασης στις υπηρεσίες τους δεν ελέγχεται ουσιαστικά από τον χρήστη, όπως είδαμε παραπάνω. Συγκεκριμένα, για την υλοποίηση του συγκεκριμένου συστήματος δημιουργείται ένας server με τη χρήση ενός Raspberry Pi ο οποίος επικοινωνεί με μια εφαρμογή Android μέσω της οποίας ελέγχονται οι ηλεκτρικές και ηλεκτρονικές συσκευές του σπιτιού με την χρήση πομπών IR και RF που είναι συνδεδεμένοι με τον server. Επιπλέον, με τη χρήση αισθητήρων θερμοκρασίας, υγρασίας και κίνησης δίνεται η δυνατότητα ελέγχου της κατάστασης του σπιτιού από οπουδήποτε, ώστε για παράδειγμα ο χρήστης να προβεί

στις κατάλληλες ενέργειες όταν η θερμοκρασία του σπιτιού είναι σχετικά υψηλή ή ανιχνευθεί κίνηση μέσα στο σπίτι ενώ εκείνος απουσιάζει.

Στις επόμενες παραγράφους, αρχικά περιγράφεται συνοπτικά τόσο ο server όσο και η εφαρμογή Android και στη συνέχεια περιγράφεται αναλυτικά πώς έγινε η υλοποίηση των λειτουργιών του συστήματος αυτοματισμού τόσο στο επίπεδο της εφαρμογής, όσο και στο επίπεδο του server.

2. Server

Η βάση του συστήματός μας είναι ένας server που τρέχει σε ένα Raspberry Pi[1],[2]. Το Raspberry είναι ένας υπολογιστής μεγέθους πιστωτικής κάρτας, ο οποίος αναπτύχθηκε από το Raspberry Pi Foundation στην Μεγάλη Βρετανία και ξεκίνησε να πωλείται το 2012. Αν και αρχικά προωθήθηκε ως μέσο για τη διδασκαλία της πληροφορικής σε αναπτυσσόμενες χώρες, γνώρισε μεγάλη εμπορική επιτυχία και σε άλλους τομείς όπως η ρομποτική. Αυτό οδήγησε στην κυκλοφορία πολλών εκδόσεων [2] η καθεμία από τις οποίες περιλαμβάνει σε σχέση με τις προηγούμενες εκδόσεις βελτιώσεις όσον αφορά την υπολογιστική ισχύ, με την χρήση ταχύτερου επεξεργαστή και περισσότερης μνήμης RAM, καθώς και την συνδεσιμότητα του Raspberry, με την ενσωμάτωση Wi-Fi και Bluetooth χωρίς την ανάγκη προσθήκης εξωτερικού προσαρμογέα. Μια ακόμη πολύ σημαντική διαφορά μεταξύ των εκδόσεων του Raspberry που αφορά την παρούσα διπλωματική είναι ο αριθμός των GPIO (General Purpose Input/Output) που είναι οι επαφές μέσω των οποίων το Raspberry θα επικοινωνεί τόσο με τους αισθητήρες όσο και με τους πομπούς και δέκτες που θα χρησιμοποιηθούν. Στην διπλωματική χρησιμοποιήθηκε το Raspberry Pi 3 Model B η πρότελευταία έκδοση του Raspberry με ελάχιστες διαφορές από την τελευταία. Για την υλοποίηση του συστήματός μας το Raspberry συνδέεται μέσω των GPIO με έναν αισθητήρα θερμοκρασίας και υγρασίας, με πομπούς και δέκτες IR και RF σημάτων καθώς και με έναν αισθητήρα κίνησης.

Το λειτουργικό σύστημα το οποίο τρέχει στο Raspberry είναι το Raspbian[3], μια Linux διανομή βασισμένη στο Debian και ειδικά βελτιστοποιημένο για τον χαμηλής απόδοσης επεξεργαστή ARM του Raspberry. Συγκεκριμένα, το Raspbian

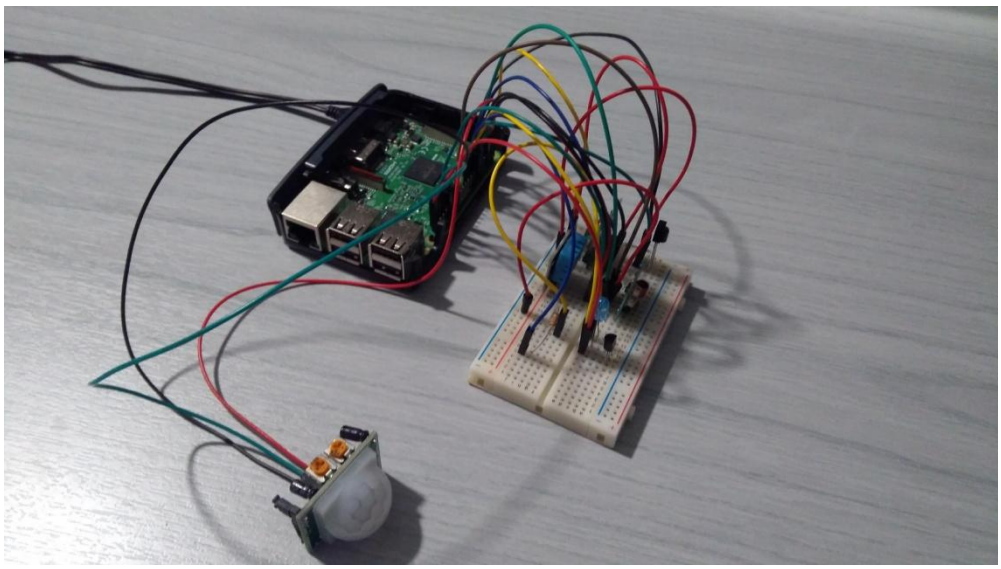
που χρησιμοποιείται βασίζεται στην έκδοση 9(Stretch) του Debian με έκδοση Linux Kernel 4.14, είναι δηλαδή η τελευταία έκδοση του συγκεκριμένου λειτουργικού[4]. Ο server έχει γραφτεί σε Flask[5], ένα web framework της Python που έχει ως σκοπό τη δημιουργία web apps. Το Flask ονομάζεται micro framework, καθώς σε αντίθεση με άλλα frameworks όπως το Django δεν έχει προεγκατεστημένα εργαλεία και βιβλιοθήκες για την υλοποίηση συνηθισμένων λειτουργιών μιας διαδικτυακής εφαρμογής, όπως είναι για παράδειγμα η διαχείρηση βάσεων δεδομένων. Ωστόσο, το Flask υποστηρίζει προεκτάσεις οι οποίες προσθέτουν αυτές τις λειτουργίες όταν ο προγραμματιστής τις χρειαστεί. Το Flask επιλέχθηκε ακριβώς λόγω αυτής της απλότητας, του γεγονότος δηλαδή ότι μπορείς να ξεκινήσεις την εφαρμογή χωρίς τις διάφορες βιβλιοθήκες και τα εργαλεία που στην περίπτωση της παρούσας εργασίας δεν θα χρησίμευαν καθώς τα περισσότερα αφορούν την διευκόλυνση της δημιουργίας μιας διαδικτυακής εφαρμογής. Ακόμα και όταν χρειάστηκε να υλοποιηθεί REST API για την επικοινωνία της Android εφαρμογής με τον server, δεν ήταν απαραίτητη η χρήση της αντίστοιχης επέκτασης Flask-RESTful [6]καθώς το Flask μπορεί να διαχειρίζεται RESTful requests. Το μόνο ουσιαστικά που πρόσθετε η επέκταση Flask-RESTful ήταν η καλύτερη οργάνωση του κώδικα όταν απαιτείται ένα πλήθος διαφορετικών request, στην περίπτωση δηλαδή που γράφεται μια πολύ μεγάλη διαδικτυακή εφαρμογή. Ο παρακάτω κώδικας δείχνει μια απλή εφαρμογή σε Flask που απλά τυπώνει "Hello world!":

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
        return "Hello World!"

if __name__ == "__main__":
        app.run()
```

Αρχικά, όπως φαίνεται εισάγεται το Flask στο Python script και στη συνέχεια ξεκινάει η εφαρμογή, η οποία στην κεντρική της σελίδα και στο port 5000 τυπώνει "Hello world!" όπως φαίνεται στο μοναδικό στην προκειμένη περίπτωση route. Τα

routes κατά τη δημιουργία web app καθορίζουν τα templates που θα εμφανίσει το πρόγραμμα περιήγησης στην αντίστοιχη σελίδα καθώς και τον κώδικα που θα εκτελεστεί όταν ο χρήστης αλληλεπιδράσει με την σελίδα. Στην περίπτωσή μας, τα routes θα χρησιμοποιηθούν για τον διαχωρισμό του κώδικα που θα εκτελείται αν το http request που δέχεται ο server αφορά τον αισθητήρα θερμοκρασίας/υγρασίας, τον έλεγχο των IR συσκευών, τον έλεγχο των RF συσκευών ή τον αισθητήρα κίνησης.

Στην παρακάτω εικόνα, φαίνεται ο server του συστήματος που προκύπτει όταν συνδεθούν οι αισθητήρες και οι πομποδέκτες μέσω των GPIO pins με το Raspberry.

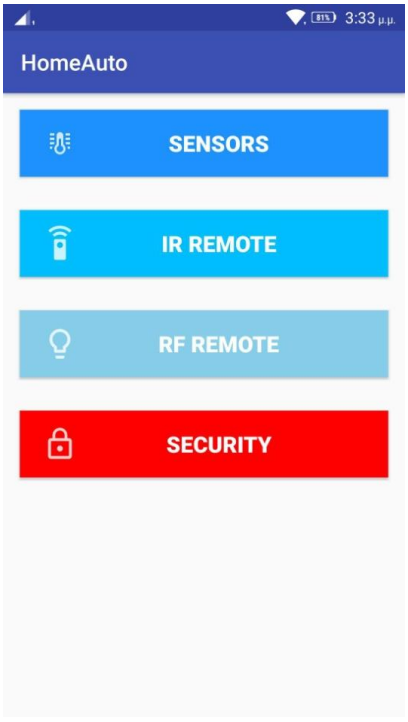


3. Εφαρμογή Android

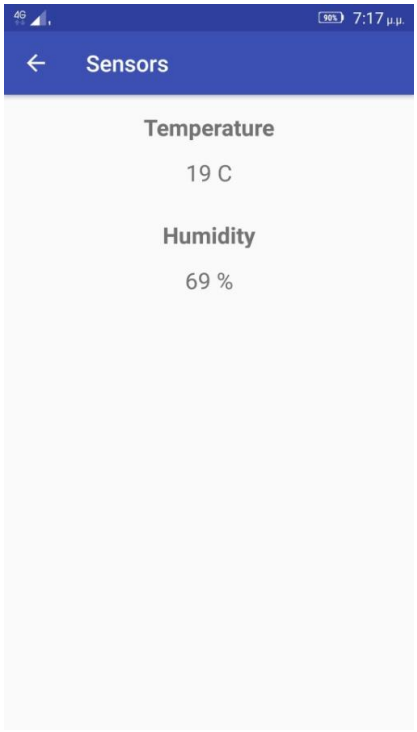
Η παρακολούθηση των δεδομένων από τους αισθητήρες που υπάρχουν στο σύστημα καθώς και ο χειρισμός των διάφορων συσκευών από τον χρήστη γίνεται με τη χρήση μιας εφαρμογής Android. Το Android είναι ένα λειτουργικό σύστημα για κινητές συσκευές που αναπτύχθηκε από την Google και βασίζεται σε έναν τροποποιημένο πυρήνα Linux. Κυκλοφόρησε για πρώτη φορά το 2008[7] και πλέον βρίσκεται στην ένατη έκδοσή του, την Android 9 Pie[8],[9]. Το Android επιλέχθηκε για την παρούσα εργασία λόγω της προσβασιμότητάς του τόσο ως προς τους χρήστες όσο και ως πρός τους προγραμματιστές των εφαρμογών. Όσον αφορά τους χρήστες, οι συσκευές Android καλύπτουν ένα μεγάλο εύρος τιμών, κάτι που έχει ως

αποτέλεσμα το Android να είναι το ποιό δημοφιλές λειτουργικό σύστημα για κινητές συσκευές παγκοσμίος και μάλιστα με ποσοστό που ξεπερνάει το 80%[10]. Τώρα, όσον αφορά τους προγραμματιστές το Android Studio[11] δηλαδή το IDE που χρησιμοποιείται για τη δημιουργία των Android εφαρμογών, μπορεί να εγκατασταθεί ουσιαστικά σε οποιοδήποτε λειτουργικό σύστημα καθώς έχει εκδόσεις για Windows, MacOS και Linux. Αντίθετα, το Xcode[12] της Apple το οποίο χρησιμοποιείται για τη δημιουργία εφαρμογών σε iOS, λειτουργεί μόνο σε MacOS και ώς εκ τούτου χρειάζεται ένας σχετικά υψηλού κόστους υπολογιστής της Apple για την ανάπτυξη εφαρμογών σε αυτό το λειτουργικό. Η εφαρμογή που δημιουργήθηκε για την παρούσα εργασία, τρέχει σε συσκευές Android με έκδοση 5.0 Lollipop και νεότερες, δηλαδή σύμφωνα και με τα τελευταία στατιστικά της Google[13] στο 90% των κινητών συσκευών με λειτουργικό σύστημα Android. Χρησιμοποιήθηκε η έκδοση 3.1.2 του Android Studio.

Πιο συγκεκριμένα, η εφαρμογή στην αρχική της οθόνη, όπως φαίνεται και στην παρακάτω φωτογραφία της, δίνει την δυνατότητα στον χρήστη να επιλέξει μέσω τεσσάρων κουμπιών για το αν θέλει να ενημερωθεί για την κατάσταση του σπιτιού χρησιμοποιώντας τους αισθητήρες θερμοκρασίας/υγρασίας αλλά και κίνησης ή αν θέλει να χειριστεί κάποια ηλεκτρική ή ηλεκτρονική συσκευή μέσω των πομπών IR και RF.



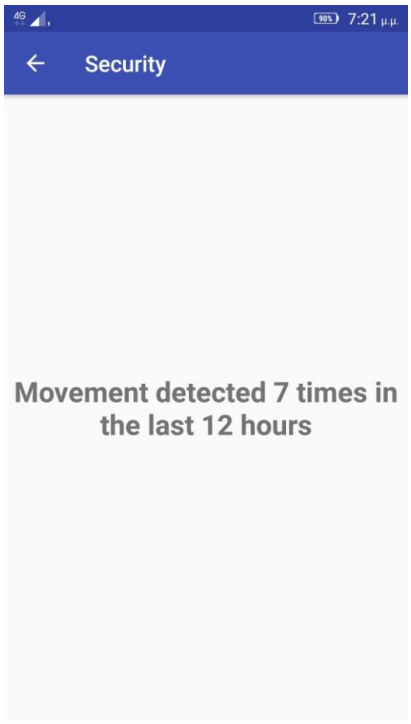
Όταν λοιπόν ο χρήστης θέλει να ενημερωθεί για τη θερμοκρασία και την υγρασία που επικρατούν στο σπίτι, πατάει το πρώτο κουμπί που υπάρχει στην κεντρική οθόνη και μεταφέρεται σε άλλη σελίδα (Activity) της εφαρμογής στην οποία μετά από επικοινωνία με τον server, παρουσιάζονται στον χρήστη οι παρατηρούμενες από τον αισθητήρα τιμές. Στην οθόνη αυτή της εφαρμογής που φαίνεται παρακάτω τα δεδομένα από τον αισθητήρα μπορούν να ανανεωθούν με Swipe Down[14], δηλαδή με την κίνηση του δείκτη από το πάνω προς το κάτω μέρος της οθόνης, κάτι που ακολουθει τις οδηγίες της Google όσον αφορά το material design, που είναι ουσιαστικά η σχεδιαστική γλώσσα του Android από την έκδοση 5 και μετά. Επιπλέον, λόγω της χρήσης του Swipe Down σε όλες τις δημοφιλείς εφαρμογές είναι κάτι το οποίο θεωρείται δεδομένο ως λειτουργία από τους χρήστες, καθως δίνει τη δυνατότητα άμεσης ανανέωσης των δεδομένων χωρίς να χρειάζεται η επιστροφή στην αρχική οθόνη.



Όταν ο χρήστης επιλέξει το δεύτερο κουμπί στην κεντρική οθόνη μεταφέρεται στην παρακάτω οθόνη, όπου του δίνεται η δυνατότητα να χειριστεί συσκευές που ελέγχονται μέσω IR, όπως για παράδειγμα τηλεοράσεις, air-condition, home cinema κτλ. Στην περίπτωση της παρούσας εφαρμογής δίνεται η δυνατότητα χειρισμού μιας τηλεόρασης.

Όταν ο χρήστης επιλέξει το τρίτο κουμπί στην κεντρική οθόνη μεταφέρεται στην παρακάτω οθόνη, όπου του δίνεται η δυνατότητα να χειριστεί συσκευές που ελέγχονται μέσω RF, τέτοιες συσκευές είναι οι ασύρματες πρίζες και τα ρελέ. Δηλαδή, δίνεται η δυνατότητα ενεργοποίησης και απενεργοποίησης σχεδόν οποιασδήποτε ηλεκτρικής και ηλεκτρονικής συσκευής, από κουζίνες και θερμοσίφωνες μέχρι τον έλεγχο του φωτισμού του σπιτιού. Στην παρούσα εφαρμογή δίνεται η δυνατότητα ελέγχου μιας ασύρματης πρίζας.

Τέλος, όταν επιλεγεί το τέταρτο και τελευταίο κουμπί ο χρήστης μεταφέρεται στην παρακάτω οθόνη που ενημερώνει τον χρήστη για το πόσες φορές έχει ανιχνεύσει κίνηση ο αισθητήρας τις τελευταίες 12 ώρες. Επιπλέον, κάθε 5 φορές που ανιχνεύεται κίνηση από τον αισθητήρα στέλνετε ειδοποίηση στον χρήστη χωρίς να χρειάζεται να είναι ανοικτή η εφαρμογή, απλά με το να είναι συνδεδεμένος στο internet. Όταν ο χρήστης επιλέξει αυτή την ειδοποίηση μεταφέρεται στην παρακάτω οθόνη ώστε να ενημερωθεί αναλυτικότερα. Και σε αυτή την οθόνη όπως και στην οθόνη με τις ενδείξεις θερμοκρασίας και υγρασίας, τα δεδομένα μπορούν να ανανεωθούν με Swipe Down.



4. Αισθητήρας θερμοκρασίας/υγρασίας

Για την παρακολούθηση της θερμοκρασίας και της υγρασίας που επικρατούν μέσα στο σπίτι χρησιμοποιείται ένας αισθητήρας DHT 11. Υπάρχουν πολλές διαφορετικές εκδοχές του συγκεκριμένου αισθητήρα. Στην παρούσα εργασία, χρησιμοποιήθηκε ο αισθητήρας αυτού του τύπου με τρία pins και breakout board.

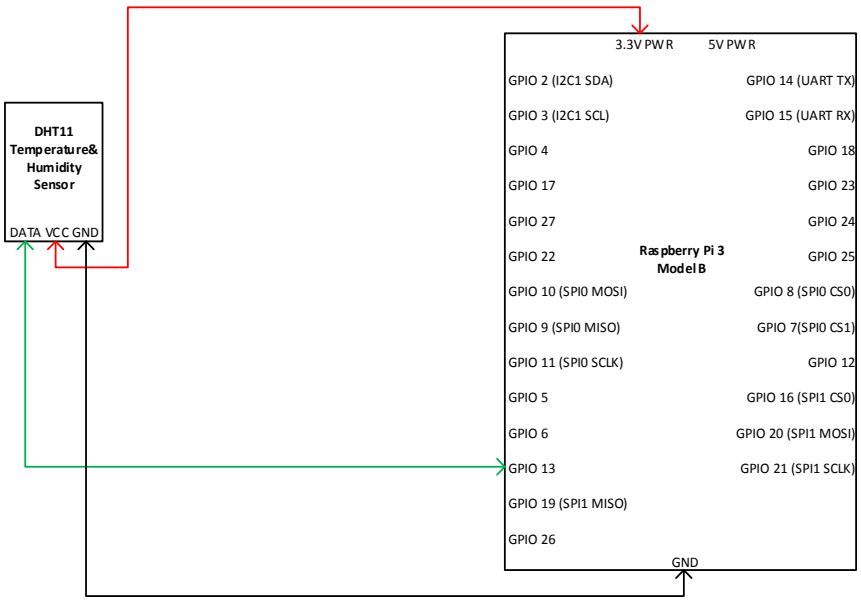
Ο DHT 11 μετράει τη σχετική υγρασία[15]. Η σχετική υγρασία είναι η ποσότητα των υδρατμών στον αέρα σε σχέση με το σημείο κορεσμού των υδρατμών

στον αέρα. Στο σημείο κορεσμού, ο υδρατμός αρχίζει να συμπυκνώνεται και να συσσωρεύεται σε επιφάνειες, σχηματίζοντας δρόσο. Το σημείο κορεσμού αλλάζει με τη θερμοκρασία του αέρα. Ο ψυχρός αέρας μπορεί να κρατήσει λιγότερους υδρατμούς πρίν κορεστεί και ο θερμός αέρας μπορεί να συγκρατήσει περισσότερους υδρατμούς προτού κορεστεί. Η σχετική υγρασία εκφράζεται ως ποσοστό. Όταν δηλαδή έχουμε σχετική υγρασία 100% συμβαίνει συμπήκνωση, ενώ όταν έχουμε 0% σχετική υγρασία, ο αέρας είναι εντελώς στεγνός.

Ο DHT 11 ανιχνεύει υδρατμούς μετρώντας την ηλεκτρική αντίσταση μεταξύ δυο ηλεκτροδίων. Το στοιχείο ανίχνευσης υγρασίας είναι ένα υπόστρωμα συγκράτησης υγρασίας με ηλεκτρόδια εφαρμοσμένα στην επιφάνεια. Όταν οι υδρατμοί απορροφούνται από το υπόστρωμα, ιόντα απελευθερώνονται από την επιφάνειά του, κάτι που αυξάνει την αγωγιμότητα μεταξύ των ηλεκτροδίων. Η μεταβολή της αντίστασης μεταξύ των δυο ηλεκτροδίων είναι ανάλογη της σχετικής υγρασίας. Η υψηλότερη σχετική υγρασία μειώνει την αντίσταση μεταξύ των ηλεκτροδίων, ενώ η χαμηλότερη σχετική υγρασία αυξάνει την αντίσταση μεταξύ των ηλεκτροδίων.

Ο DHT 11 μετρά τη θερμοκρασία με τη χρήση αισθητήρα θερμοκρασίας NTC (θερμίστορ). Τα θερμίστορ είναι ένας τύπος αντίστασης, η τιμή της οποίας επηρεάζεται από τη θερμοκρασία πολύ περισσότερο από όσο στις συνηθισμένες αντιστάσεις. Το θερμίστορ που χρησιμοποιείται στον DHT 11 είναι NTC, δηλαδή Negative Temperature Coefficient, που σημαίνει ότι η αντίσταση μειώνεται με την αύξηση της θερμοκρασίας.

Με τη χρήση jumper cables και ενός breadboard ενώνεται ο αισθητήρας με το Raspberry με τη χρήση των GPIO. Συγκεκριμένα, το αριστερό pin που χρησιμοποιείται για τη μεταφορά δεδομένων συνδέεται με το GPIO 13,το μεσαίο που αφορά την τροφοδοσία με το GPIO που παρέχει 3.3v, ενώ το δεξί είναι η γείωση που επίσης συνδέεται με το κατάλληλο GPIO. Στο παρακάτω σχηματικό φαίνεται αναλυτικά η συνδεσμολογία.



Για να ληφθούν τώρα δεδομένα από τον αισθητήρα χρησιμοποιήθηκε μια βιβλιοθήκη από το github[16] γραμμένη σε python και το script dht11.py μεταφέρεται στον ίδιο φάκελο με το αρχείο του Flask server, main.py και γίνεται import. Στη συνέχεια χρησιμοποιείται στο route του server που αφορά τον DHT 11 αισθητήρα, όπως φαίνεται παρακάτω. Αναλυτικότερα, αρχικά καθορίζεται ο τρόπος αρίθμησης που θα χρησιμοποιηθεί για τον προσδιορισμό του GPIO pin στο οποίο θα στέλνωνται τα δεδομένα από τον αισθητήρα. Εδω χρησιμοποιείται ο τρόπος αρίθμησης GPIO.BCM όπου προσδιορίζουμαι το pin γράφοντας τον αριθμό του GPIO, ακολουθώντας δηλαδή την αρίθμηση που δεν περιλαμβάνει τα pins που παρέχουν τροφοδοσία ή γείωση, ενώ υπάρχει και ο τρόπος GPIO.BOARD στον οποίο γράφουμε τον αριθμό του pin από 1 έως 40 που περιλαμβάνει και τα 40 pin που υπάρχουν στο board του Raspberry .Στη συνέχεια, με τη χρήση της βιβλιοθήκης και συγκεκριμένα της κλάσης DHT11 καθορίζεται το pin από το οποίο θα ληφθούν τα δεδομένα, τα οποία στη συνέχεια διαβάζονται και αποθηκεύονται στη μεταβλητή result με κλήση της συνάρτησης read της συγκεκριμένης κλάσης και διαχωρίζονται σε αυτά που αφορούν τη θερμοκρασία και σε αυτά που αφορούν την υγρασία. Τα δεδομένα που λαμβάνονται για τη θερμοκρασία και την υγρασία αποθηκεύονται στις μεταβλητές therm και hum αντίστοιχα. Τέλος, με τη χρήση της jsonify δημιουργείται το json με τα δεδομένα αυτά, τα οποία θα δοθούν από τον server όταν δεχθεί το GET Request.

```
@app.route("/sensors", methods = ['GET'])
def get_mes():
        # initialize GPIO
        GPIO.setwarnings(False)
        GPIO.setmode(GPIO.BCM)
        # set the GPIO used for data
        instance = dht11.DHT11(pin=13)
        # get sensor data
        result = instance.read()
        # use variables to get temperature and humidity data seperately
        therm = result.temperature
        hum = result.humidity
        # return json with sensor data
        return jsonify({"temperature": therm,"humidity":hum})
```

Από την πλευρά τώρα της εφαρμογής Android, χρησιμοποιείται η βιβλιοθήκη Volley[17] για την υλοποίηση των http requests, στην περίπτωσή μας του GET request. Η Volley είναι μια βιβλιοθήκη που επιτρέπει την υλοποίηση του δικτυακού τμήματος των εφαρμογών Android ευκολότερα και γρηγορότερα, με τη χρήση λιγότερου κώδικα. Επιπλέον, στη Volley όλες οι κλήσεις δικτύου (όπως τα GET και POST Requests) δουλεύουν ασύγχρονα οπότε δεν υπάρχει η ανάγκη χρήσης της AsyncTask όπως στο παρελθόν. Αναλυτικότερα, όπως φαινεται στον παρακάτω κώδικα Java του συγκεκριμένου Activity της εφαρμογής, αρχικά δημιουργείται ένα object του RequestQueue, στη συνέχεια γίνεται το GET Request, λαμβάνονται οι τιμές των values του json που θέλουμε μέσω της συνάρτησης onResponse που γίνεται Override και στη συνέχεια τυπώνονται στα επιθυμητά TextViews του Activity. Τα TextViews αυτά έχουν προσδιοριστεί στο αντίστοιχο xml και τους έχουν δοθεί τα κατάλληλα id ώστε να είναι προσβάσιμα στον κώδικα Java της εφαρμογής. Τέλος, για να γίνει το συγκεκριμένο Request το object του, objectRequest, προστίθεται στο RequestQueue. Ο παρακάτω κώδικας καλείται τόσο στην μέθοδο onCreate, όταν δηλαδη ο χρήστης ανοίγει το συγκεκριμένο Activity, καθώς και στην onRefresh ώστε όταν γίνεται Swipe Down τα δεδομένα να ανανεώνονται.

```
RequestQueue  requestqueue = Volley.newRequestQueue(this);
```

```
JsonObjectRequest objectRequest = new JsonObjectRequest(Request.Method.GET, url,
null,
        new Response.Listener<JSONObject>() {
          @Override
          public void onResponse(JSONObject response) {
            try{
              int Temp = response.getInt("temperature");
              int Humidity = response.getInt("humidity");

              TempText.setText(String.valueOf(Temp)+" C");
              HumText.setText(String.valueOf(Humidity)+" %");
            }
            catch (JSONException e){
              e.printStackTrace();
            }
          }
        },
        new Response.ErrorListener() {
          @Override
          public void onErrorResponse(VolleyError error) {
            TempText.setText("-");
            HumText.setText("-");
          }
        });

requestqueue.add(objectRequest);
```

5. Έλεγχος IR συσκευών

Η υπέρυθρη ακτινοβολία είναι ένας τύπος φωτός παρόμοιος με το φως που βλέπουμε γύρω μας[18]. Η μόνη διαφορά μεταξύ του υπέρυθρου φωτός και του ορατού φωτός είναι η συχνότητα και το μήκος κύματος. Η υπέρυθρη ακτινοβολία

δηλαδή είναι έξω από το φάσμα του ορατού φωτός, γι'αυτό δεν είναι ορατή με το ανθρώπινο μάτι.

Επειδή το IR είναι ένας τύπος φωτός, η IR επικοινωνία απαιτεί άμεση οπτική επαφή από τον δέκτη στον πομπό. Δεν μπορεί να γίνει δηλαδή μετάδοση μέσω τοίχων ή άλλων υλικών όπως γίνεται με το WiFi ή το Bluetooth. Ένα τυπικό σύστημα επικοινωνίας υπερύθρων απαιτεί έναν πομπό IR και έναν δέκτη IR. Ο πομπός μοιάζει με ένα τυπικό LED, με τη διαφορά ότι παράγει φως στο υπέρυθρο φάσμα αντί να το παράγει στο ορατό φάσμα. Ο δέκτης IR είναι μια φωτοδίοδος και ένας προενισχυτής που μετατρέπει το υπέρυθρο φως σε ηλεκτρικό σήμα.

Όμως υπέρυθρο φως εκπέμπεται από τον ήλιο, τους λαμπτήρες και οτιδήποτε άλλο παράγει θερμότητα. Αυτό σημαίνει ότι υπάρχει πολύς θόρυβος από υπέρυθρο φως γύρω μας. Για να αποφευχθεί η παρεμβολή αυτού του θορύβου στο IR σήμα, χρησιμοποιείται μια τεχνική διαμόρφωσης σήματος.

Στη διαμόρφωση τώρα του σήματος IR, ένας κωδικοποιητής στο τηλεχειριστήριο IR μετατρέπει ένα δυαδικό σήμα σε ένα διαμορφωμένο ηλεκτρικό σήμα. Αυτό το ηλεκτρικό σήμα αποστέλλεται στο LED εκπομπής. Το LED εκπομπής μετατρέπει το διαμορφωμένο ηλεκτρικό σήμα σε διαμορφωμένο σήμα φωτός IR. Ο δέκτης IR αποδιαμορφώνει τότε το σήμα φωτός IR και το μετατρέπει σε δυαδική μορφή πρίν μεταβιβάσει την πληροφορία σε ένα μικροελεγκτή.

Το διαμορφωμένο IR σήμα είναι μια σειρά από παλμούς φωτός IR που ενεργοποιούνται και απενεργοποιούνται σε μια υψηλή συχνότητα γνωστή ως φέρουσα συχνότητα. Η φέρουσα συχνότητα που χρησιμοποιείται από τους περισσότερους πομπούς είναι 38kHz, επειδή είναι σπάνια στη φύση και έτσι μπορεί να διακριθεί από τον θόρυβο του περιβάλλοντός. Με αυτό τον τρόπο ο δέκτης IR γνωρίζει ότι το σήμα των 38kHz στάλθηκε από τον πομπό και δεν έχει ληφθεί από το περιβάλλον. Η δίοδος δέκτη λοιπόν ανιχνεύει όλες τις συχνότητες φωτός IR, αλλά έχει ένα ζωνοπερατό φίλτρο που επιτρέπει τη διέλευση υπερύθρων στα 38kHz.

Κάθε φορά που πατιέται ένα κουμπί στο τηλεχειριστήριο, δημιουργείται ένας μοναδικός δεκαεξαδικός κωδικός. Αυτή είναι η πληροφορία που διαμορφώνεται και αποστέλλεται μέσω IR στον δέκτη. Για να αποκρυπτογραφήσει ποιο πλήκτρο πιέζεται, ο μικροελεγκτής λήψης πρέπει να γνωρίζει ποιος κωδικός αντιστοιχεί σε κάθε πλήκτρο του τηλεχειριστηρίου. Επιπλέον, διαφορετικά τηλεχειριστήρια στέλνουν διαφορετικούς κωδικούς για τα ίδια κουμπιά, επομένως πρέπει να

καθοριστεί ο κωδικός που παράγεται για κάθε κουμπί στο συγκεκριμένο τηλεχειριστήριο.

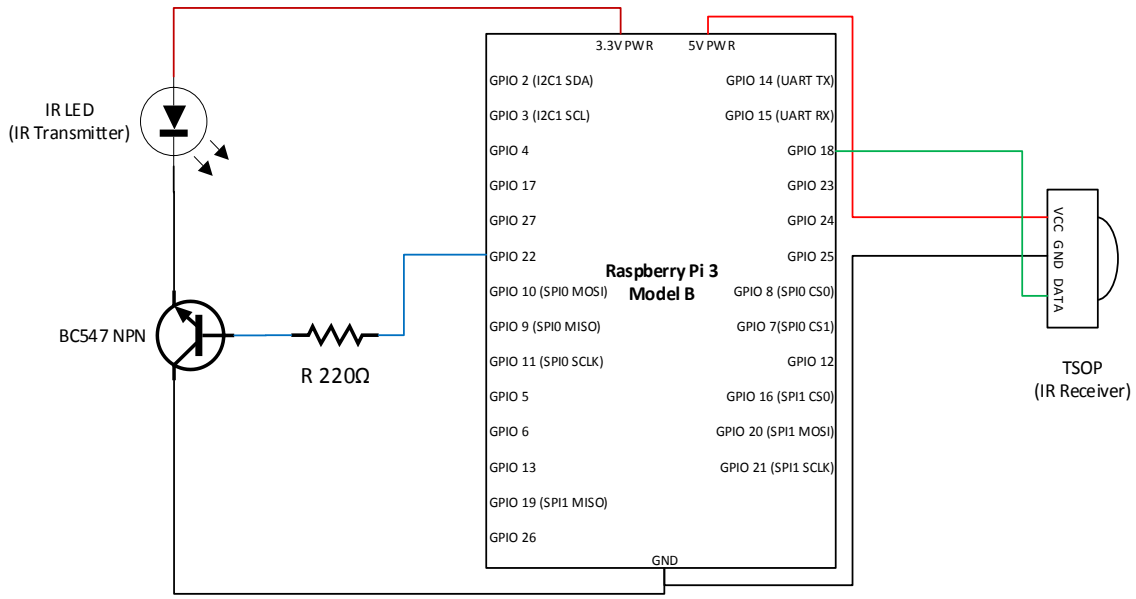
Για τον έλεγχο των IR συσκευών χρησιμοποιήθηκε η βιβλιοθήκη LIRC[19],[20] η οποία επιτρέπει την αποκωδικοποίηση και την αποστολή IR σημάτων, από υπολογιστές με λειτουργικό σύστημα Linux. Στην ουσία δηλαδή το Raspberry με τη βοήθεια της συγκεκριμένης βιβλιοθήκης, παίρνει αρχικά τον ρόλο του μικροελεγκτή λήψης και γίνεται η αντιστοίχηση των πλήκτρων με τους κωδικούς τους και στη συνέχεια έχοντας αυτή την πληροφορία χρησιμοποιείται από τον χρήστη για την αποστολή των επιθυμητών εντολών στη συσκευή IR. Για να γίνει αυτό ακολουθείται μια συγκεκριμένη διαδικασία.

Αρχικά λοιπόν, εγκαθίσταται η συγκεκριμένη βιβλιοθήκη στο Raspbian με την εντολή κονσόλας sudo apt-get install lirc και ορίζονται με επεξεργασία των αρχείων της βιβλιοθήκης τα GPIO pins μέσω των οποίων θα γίνεται λήψη και αποστολή των σημάτων. Στη συνέχεια, αποκωδικοποιούνται τα σήματα από το τηλεχειριστήριο με τη χρήση της εντολής irrecord της συγκεκριμένης βιβλιοθήκης και σε επίπεδο hardware με τη χρήση ενός TSOP 38238 IR Receiver. Ο IR Receiver αυτός συνδέεται με τα κατάλληλα GPIO pins ώστε να τροφοδοτείται με 5v, να είναι γειωμένος και να μεταφέρει τα δεδομένα που λαμβάνει από το IR τηλεχειριστήριο στο Raspberry μέσω του GPIO 18. Στο τέλος της διαδικασίας αποκωδικοποίησης δημιουργείται ένα configuration file το οποίο στη συνέχεια επιτρέπει την αποστολή IR σημάτων. Για την αποστολή των IR σημάτων χρησιμοποιείται ένα IR led, το οποίο στην άνοδο συνδέεται με 3v τροφοδοσία από το Raspberry ενώ η κάθοδος του συνδέεται με τον εκπομπό ενός BC 547 npn transistor του οποίου η βάση συνδέεται, με την παρεμβολή και μιας αντίστασης 220 Ohm, με το GPIO 22 για τα δεδομένα, ενώ ο συλλέκτης συνδέεται στη γείωση. Παρακάτω, δίνεται σχηματικά η συνδεσμολογία τόσο του IR Receiver όσο και του IR transmitter (IR LED) που περιγράφηκε παραπάνω.

Τα IR σήματα αποστέλονται με τη χρήση της εντολής irsend της βιβλιοθήκης LIRC και του ονόματος που έχουμε δώσει στο configuration file του χειριστηρίου. Στην περίπτωση μας για παράδειγμα, που δημιουργείται ένα configuration file με όνομα samsung η εντολή κονσόλας για την ενεργοποίηση ή απενεργοποίηση της συσκευής είναι irsend SEND_ONCE samsung KEY_POWER.

Για λόγους πληρότητας, αξίζει να αναφερθεί πως όταν προσπάθησα να καταγράψω το σήμα του χειριστηρίου ενός air condition με την εντολή irrecord της

βιβλιοθήκης LIRC, η διαδικασία δεν προχωρούσε όπως πρίν καθώς δεν λαμβάνονταν κωδικοί από το πρόγραμμα. Όπως προέκυψε μετά από δοκιμές και περαιτέρω έρευνα, το πρόβλημα ήταν ότι τα περισσότερα σύγχρονα air condition ελέγχονται μέσω ενός τηλεχειριστηρίου με οθόνη, η οποία δείχνει όλες τις τρέχουσες ρυθμίσεις του AC. Αυτά τα χειριστήρια είναι διαφορετικά από ένα χειριστήριο τηλεόρασης το οποίο στέλνει για κάθε κουμπί ένα διαφορετικό σήμα με τις πληροφορίες μόνο αυτού του κουμπιού (για παράδειγμα αύξηση της έντασης του ήχου). Τα χειριστήρια των AC από την άλλη πλευρά στέλνουν όλες τις πληροφορίες, όπως θερμοκρασία και ταχύτητα ανεμιστήρα, κάθε φορά που πατάμε ένα κουμπί. Αυτό συμβαίνει για να συγχρονιστεί το τηλεχειριστήριο (όπου βλέπουμε τις αλλαγές στις ρυθμίσεις) με τη μονάδα AC (όπου πρέπει να συμβούν οι αλλαγές). Η καταγραφή λοιπόν των κωδικών σε αυτή την περίπτωση μπορεί να γίνει με χρήση της mode2 λειτουργίας της LIRC βιβλιοθήκης που καταγράφει τους κωδικούς, χωρίς να γίνει η διαδικασία που χρειάστηκε για το χειριστήριο της τηλεόρασης[21]. Δημιουργείται λοιπόν με χρήση της εντολής mode2 –m -d /dev/lirc0>ac.conf ένα raw configuration file το οποίο επεξεργαζόμαστε στη συνέχεια, ώστε να πάρει την μορφή των configuration file που προέκυπταν με τη χρήση της εντολής irrecord και να μπορεί να χρησιμοποιηθεί στη συνέχεια για τον χειρισμό της αντίστοιχης συσκευής.



Στην εφαρμογή Android, χρησιμοποιείται όπως και για τον αισθητήρα θερμοκρασίας/υγρασίας η βιβλιοθήκη Volley για τα απαραίτητα http requests, όμως αυτή τη φορά έχουμε να κάνουμε με POST Requests καθώς δεν θέλουμε να πάρουμε

δεδομένα από τον Server αλλά να του στείλουμε δεδομένα όταν πατάμε κάποιο κουμπί και ανάλογα με το τι του στέλνουμε να εκτελέσει την κατάλληλη εντολή. Όπως βλέπουμε λοιπόν στον παρακάτω κώδικα, η κύρια διαφορά είναι η δημιουργία ενός HashMap με το όνομα params το οποίο αντιστοιχεί keys με values και στη συνέχεια στέλνεται ως JSON object. Αυτό φαίνεται κατά τη δημιουργία του request με το JSONObject(params) που γράφεται ώς παράμετρος. Επιπλέον, κατά τη δημιουργία του request object καθορίζεται ότι πρόκειται για post request με το Request.Method.POST. Τέλος, το request προστίθεται στο request queue που έχει δημιουργηθεί προηγουμένως.

```
HashMap<String, String> params = new HashMap<>();
params.put("button", "volUp");

 RequestQueue postqueue = Volley.newRequestQueue(this);

JsonObjectRequest postRequest = new JsonObjectRequest(Request.Method.POST, url, new
JSONObject(params) ,
            new Response.Listener<JSONObject>() {
        @Override
        public void onResponse(JSONObject response) {

        }
    },
            new Response.ErrorListener() {
               @Override
               public void onErrorResponse(VolleyError error) {

               }
            });

postqueue.add(postRequest);
```

Όταν φτάσει λοιπόν το Request στον server και μάλιστα στο συγκεκριμένο route που δίνεται παρακάτω και για το οποίο έχει οριστεί ότι θα δέχεται POST Requests με το methods=['POST'] που δέχεται ο decorator @app.route ώς παράμετρο,

λαμβάνεται το JSON και θέτουμε μια μεταβλητή ίση με την τιμή που έχει το key ‘’button’’ του JSON. Ανάλογα λοιπόν με την τιμή αυτή, η οποία ελέγχεται με μια δομή ελέγχου if εκτελείται και η κατάλληλη εντολή κονσόλας με την χρήση της εντολής irsend της LIRC όπως αναφέρθηκε και προηγουμένως, ώστε να δοθεί στην συσκευή η επιθυμητη εντολή.

```
@app.route("/ir", methods = ['POST'])
def ir_remote():
        #receive json and save in a variable
        data = request.get_json()
        #get the value of the key button and save it to button variable
        button = data['button']
        #choose what console command to execute using if statement
        if  button == "power":
                        subprocess.call("irsend SEND_ONCE samsung KEY_POWER",
                        shell =    True)
        elif  button == "volUp":
                subprocess.call("irsend SEND_ONCE samsung KEY_VOLUMEUP", shell =
                True)
        elif  button == "volDown":
                subprocess.call("irsend SEND_ONCE samsung KEY_VOLUMEDOWN",
                shell = True)
        elif  button == "chUp":
                subprocess.call("irsend SEND_ONCE samsung KEY_CHANNELUP", shell
                = True)
        elif  button == "chDown":
                subprocess.call("irsend SEND_ONCE samsung KEY_CHANNELDOWN",
                shell = True)
        else:
                print("Wrong!!!")
        return "ok"
```

6. Έλεγχος RF συσκευών

Από τη δεκαετία του 1990, η ασύρματη επικοινωνία βρήκε ένα διαφορετικό πεδίο πέρα από τη χρήση της στις τηλεπικοινωνίες, τις στρατιωτικές και διαστημικές εφαρμογές. Αυτό ήταν η χρήση επικοινωνίας ραδιοσυχνοτήτων χαμηλής ζώνης για μεταφορά δεδομένων σε μικρές αποστάσεις[22]. Ο τομέας των τηλεπικοινωνιών το χρησιμοποίησε, εισάγοντας τεχνολογίες όπως το Bluetooth για τη μεταφορά δεδομένων μεταξύ συσκευών που απέχουν λίγα μέτρα και το Wi-Fi για την ασύρματη πρόσβαση στο Internet.

Η απόδοση οποιουδήποτε ασύρματου συστήματος κρίνεται βάσει δυο παραμέτρων, της απόστασης που μπορεί να μεταφέρει δεδομένα και του ρυθμού με τον οποίο μπορεί να μεταδώσει και να λάβει δεδομένα. Τα ασύρματα συστήματα χρησιμοποιούν ένα ευρύ φάσμα συχνοτήτων από 30KHz έως 300GHz. Όσο υψηλότερη είναι η συχνότητα τόσο μεγαλύτερη είναι η απόσταση λειτουργίας. Για επικοινωνία μικρών αποστάσεων, χρησιμοποιούνται χαμηλές συχνότητες των μερικών KHz ή MHz.

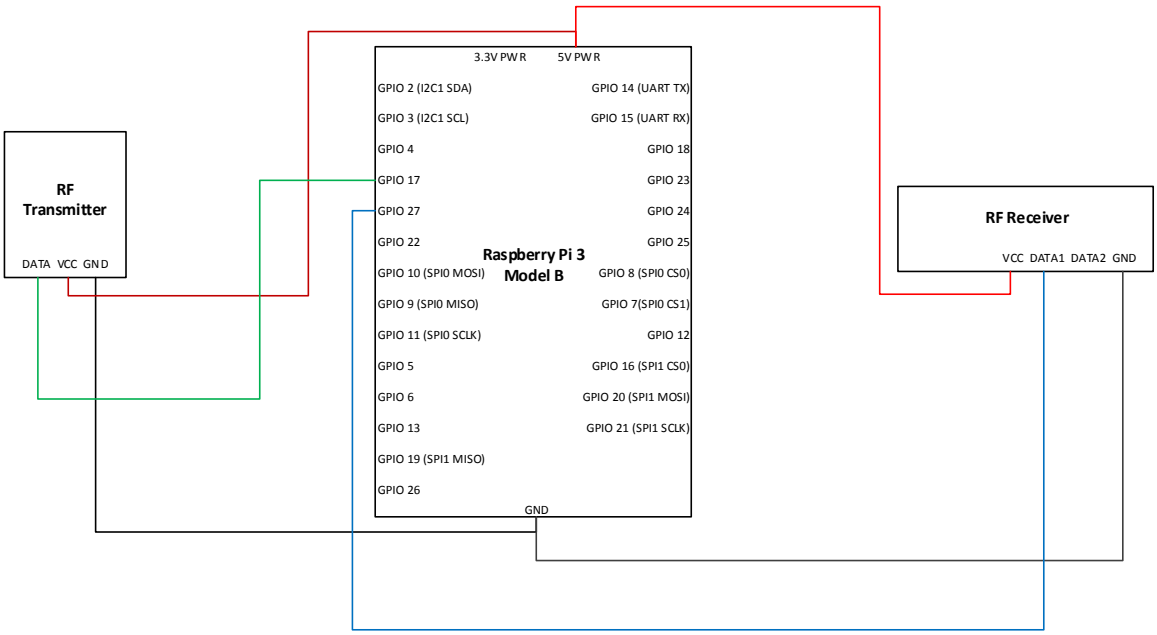
Όπως κάθε σύστημα επικοινωνίας, τα συστήματα RF έχουν έναν πομπό που μεταδίδει ηλεκτρομαγνητικά κύματα με δεδομένα κωδικοποιημένα σε αυτά και έναν δέκτη που λαμβάνει αυτά τα κύματα και ανακτά τα δεδομένα από αυτά. Στην παρούσα εργασία, χρησιμοποιήθηκαν πομπός και δέκτης RF που λειτουργούν στα 433MHz. Αυτές οι μονάδες ραδιοεπικοινωνίας, έχουν σημαντικά πλεονεκτήματα σε σχέση με άλλες τεχνολογίες απομακρυσμένου ελέγχου,όπως οι υπέρυθρες, καθώς δεν χρειάζεται οπτική επαφή μεταξύ πομπού και δέκτη για να λειτουργήσουν.

Διάφορες φέρουσες συχνότητες χρησιμοποιούνται συνήθως σε εμπορικά διαθέσιμες μονάδες RF, συμπεριλαμβανομένων εκείνων στις βιομηχανικές, επιστημονικές και ιατρικές ραδιοσυχνότητες, όπως 433,92MHz, 915MHz και 2400MHz. Η επιλογή των 433MHz μεταξύ αυτών των συχνοτήτων οφείλεται σε εθνικούς και διεθνείς κανονισμούς που διέπουν τη χρήση ραδιοσυχνοτήτων για επικοινωνία. Οι συσκευές μικρής εμβέλειας μπορούν επίσης να χρησιμοποιούν συχνότητες όπως 315MHz και 868MHz, οι οποίες είναι διαθέσιμες χωρίς άδεια.

Στον πομπό, τα ηλεκτρομαγνητικά κύματα πρέπει να διαμορφωθούν για την κωδικοποίηση δεδομένων σε αυτά. Υπάρχουν πολλές τεχνικές διαμόρφωσης και η επιλογή της κατάλληλης εξαρτάται από την εφαρμογή και τις απαιτήσεις. Οι μονάδες

RF που χρησιμοποιήθηκαν στην παρούσα εργασία, χρησιμοποιούν την τεχνική διαμόρφωσης OOK(On-off keying). Η OOK είναι η απλούστερη μορφή της ASK(Amplitude-shift keying) διαμόρφωσης και χρησιμοποιείται πολύ συχνά σε εφαρμογές απομακρυσμένου ελέγχου λόγω της απλότητας και του χαμηλού κόστους υλοποίησης. Στον δέκτη, το σήμα αποδιαμορφώνεται με την ίδια τεχνική (OOK) και τα δεδομένα εξάγονται από το φέρον κύμα.

Για την υλοποίηση του ελέγχου των RF συσκευών χρησιμοποιείται το rpi-rf [23]package που βασίζεται στην rc_switch library, η οποία αρχικά δημιουργήθηκε για τον απομακρυσμένο έλεγχο RC (Radio Controlled) συσκευών μέσω Arduino. Αρχικά λοιπόν, γίνεται εγκατάσταση του rpi-rf στο Raspberry και χρησιμοποιούνται τα scripts receive.py και send.py τα οποία χρησιμοποιώντας συναρτήσεις του rpi-rf package, μας βοηθούν να καταγράψουμε και στη συνέχεια να στείλουμε, μέσω του Raspberry πλέον, τους κωδικούς του χειριστηρίου της RF συσκευής[24]. Σε επίπεδο hardware, όπως αναφέρθηκε προηγουμένως χρησιμοποιούνται ένας RF Receiver και ένας RF Transmitter σημάτων RF 433MHz. Ο RF Receiver συνδέεται στο GPIO 27 με τη χρήση ενός εκ των δυο ακροδεκτών που αφορούν τη μεταφορά δεδομένων, τροφοδοτείται από την παροχή 5v του Raspberry και συνδέεται και στη γείωση. Η συνδεσμολογία τόσο για τον πομπό όσο και για τον δέκτη RF φαίνεται αναλυτικά και στο παρακάτω σχηματικό.



Τρέχουμε τώρα το script receive.py που δίνεται παρακάτω και το οποίο, με τη χρήση συναρτήσεων της κλάσης RFDevice του module rpi_rf καθώς και του argparse module της Python, μας βοηθάει στη λήψη των RF σημάτων και στην εξαγωγή από αυτά των απαραίτητων κωδικών για την λειτουργία του συστήματός μας ως χειριστήριο RF συσκευών.

```
import argparse
import signal
import sys
import time
import logging

#import RFDevice class from rpi_rf module
from rpi_rf import RFDevice

rfdevice = None

# pylint: disable=unused-argument
def exithandler(signal, frame):
    rfdevice.cleanup()
    sys.exit(0)

#create log structure
logging.basicConfig(level=logging.INFO, datefmt='%Y-%m-%d %H:%M:%S',
                    format='%(asctime)-15s - [%(levelname)s] %(module)s: %(message)s', )

#use argpase python module to initialize parser and set gpio from which it takes the data
parser = argparse.ArgumentParser(description='Receives a decimal code via a 433/315MHz GPIO device')
parser.add_argument('-g', dest='gpio', type=int, default=27,
                        help="GPIO pin (Default: 27)")
args = parser.parse_args()
#using fuctions from RFDevice class and the parser to enable signal detection from the set #GPIO
signal.signal(signal.SIGINT, exithandler)
rfdevice = RFDevice(args.gpio)
```

```
rfdevice.enable_rx()
timestamp = None
logging.info("Listening for codes on GPIO " + str(args.gpio))
#Use infinite loop to receive and present rf signal characteristics
while True:
    if rfdevice.rx_code_timestamp != timestamp:
        timestamp = rfdevice.rx_code_timestamp
        logging.info(str(rfdevice.rx_code) +
                     " [pulselength " + str(rfdevice.rx_pulselength) +
                     ", protocol " + str(rfdevice.rx_proto) + "]")
    time.sleep(0.01)
rfdevice.cleanup()
```

Πατάμε λοιπόν, στην προκειμένη περίπτωση τα κουμπιά on και off του τηλεχειριστηρίου της ασύρματης πρίζας των οποίων τη λειτουργία θέλουμε να αναπαράγουμε. Εμφανίζεται τότε όπως φαίνεται παρακάτω ο κωδικός, μαζί με το μήκος κύματος και το πρωτόκολλο του σήματος που στέλνει το κάθε κουμπί. Όπως βλέπουμε, κάποιες τιμές δεν ταιριάζουν με τις άλλες κυρίως λόγω θορύβου, οπότε καταγράφουμε τις τιμές που εμφανίζονται τις περισσότερες φορές.

```
2018-10-26 12:39:55 - [INFO] recieve: 242 [pulselength 410, protocol 1]
2018-10-26 12:39:55 - [INFO] recieve: 21811 [pulselength 180, protocol 1]
2018-10-26 12:39:55 - [INFO] recieve: 21811 [pulselength 186, protocol 1]
2018-10-26 12:39:55 - [INFO] recieve: 21811 [pulselength 182, protocol 1]
2018-10-26 12:39:55 - [INFO] recieve: 21811 [pulselength 182, protocol 1]
2018-10-26 12:39:58 - [INFO] recieve: 21820 [pulselength 180, protocol 1]
2018-10-26 12:39:58 - [INFO] recieve: 21820 [pulselength 181, protocol 1]
2018-10-26 12:39:58 - [INFO] recieve: 21820 [pulselength 185, protocol 1]
2018-10-26 12:39:58 - [INFO] recieve: 21820 [pulselength 182, protocol 1]
2018-10-26 12:41:20 - [INFO] recieve: 21811 [pulselength 180, protocol 1]
2018-10-26 12:41:20 - [INFO] recieve: 21811 [pulselength 181, protocol 1]
2018-10-26 12:41:20 - [INFO] recieve: 21811 [pulselength 182, protocol 1]
2018-10-26 12:41:20 - [INFO] recieve: 21811 [pulselength 182, protocol 1]
2018-10-26 12:41:32 - [INFO] recieve: 256 [pulselength 1102, protocol 2]
2018-10-26 12:41:32 - [INFO] recieve: 21820 [pulselength 180, protocol 1]
2018-10-26 12:41:32 - [INFO] recieve: 21820 [pulselength 181, protocol 1]
2018-10-26 12:41:32 - [INFO] recieve: 21820 [pulselength 182, protocol 1]
2018-10-26 12:41:32 - [INFO] recieve: 21820 [pulselength 182, protocol 1]
2018-10-26 12:41:48 - [INFO] recieve: 21811 [pulselength 180, protocol 1]
2018-10-26 12:41:48 - [INFO] recieve: 21811 [pulselength 181, protocol 1]
2018-10-26 12:41:49 - [INFO] recieve: 21811 [pulselength 182, protocol 1]
2018-10-26 12:41:49 - [INFO] recieve: 21811 [pulselength 182, protocol 1]
2018-10-26 12:41:53 - [INFO] recieve: 252 [pulselength 409, protocol 1]
```

Στη συνέχεια συνδέουμε τον RF Transmitter στην παροχή 5v του Raspberry, στη γείωση και για τα δεδομένα στο GPIO 17. Χρησιμοποιώντας λοιπόν τους κωδικούς

και τα υπόλοιπα χαρακτηριστικά των σημάτων RF που καταγράψαμε με την παραπάνω διαδικασία από τον δέκτη, τα στέλνουμε χρησιμοποιώντας το send.py όπως φαίνεται παρακάτω. Το send.py script είναι παρόμοιο σε δομή, δηλαδή γίνεται προφανώς χρήση τόσο του argparse module της Python όσο και συναρτήσεων της κλάσης RFDevice του module rpi_rf , όμως αυτή τη φορά χρησιμοποιούνται συναρτησεις που βοηθούν στην αποστολή των διάφορων παραμέτρων του καταγεγραμμένου σήματος.

```
pi@raspberrypi:~/project $ python3 send.py -p 182 -t 1 21820
2018-11-16 23:06:10 - [INFO] send: 21820 [protocol: 1, pulselength: 182]
pi@raspberrypi:~/project $ python3 send.py -p 182 -t 1 21811
2018-11-16 23:06:19 - [INFO] send: 21811 [protocol: 1, pulselength: 182]
pi@raspberrypi:~/project $ python3 send.py -p 182 -t 1 21820
2018-11-16 23:06:24 - [INFO] send: 21820 [protocol: 1, pulselength: 182]
pi@raspberrypi:~/project $ python3 send.py -p 182 -t 1 21811
2018-11-16 23:06:26 - [INFO] send: 21811 [protocol: 1, pulselength: 182]
```

Στην εφαρμογή Android, χρησιμοποιούμε όπως και στον έλεγχο των IR συσκευών τη βιβλιοθήκη Volley για την αποστολή των POST Requests ώστε να ενεργοποιηθεί ή να απενεργοποιηθεί η ελεγχόμενη μεσω RF συσκευή, στην περίπτωσή μας μια πρίζα, ανάλογα με το κουμπί που πατάει ο χρήστης της εφαρμογής.

```
HashMap<String, String> params = new HashMap<>();
params.put("state", "1");
RequestQueue postqueue1 = Volley.newRequestQueue(this);

 JsonObjectRequest postRequest1 = new JsonObjectRequest(Request.Method.POST, url, new
JSONObject(params) ,
            new Response.Listener<JSONObject>() {
        @Override
        public void onResponse(JSONObject response) {
         }
},
            new Response.ErrorListener() {
               @Override
               public void onErrorResponse(VolleyError error) {
                }
            });
```

```
postqueue1.add(postRequest1);
```

Και στον server, όπως και για τις IR συσκευές λαμβάνεται το αρχείο JSON και θέτουμε μια μεταβλητή ίση με την τιμή του key "state" του JSON. Η μόνη διαφορά είναι ότι εδώ η τιμή του key μετατρέπεται σε ακέραιο πρίν την θέσουμε ίση με τη μεταβλητή. Στη συνέχεια, ανάλογα με την τιμή της μεταβλητής εκτελέιται η ανάλογη εντολή κονσόλας ώστε με τη χρήση του send.py να σταλεί το επιθυμητό RF σήμα.

```
@app.route("/rf", methods = ['POST'])
def  rf_remote():
        #receive json and save in a variable
        data = request.get_json()
        #get the value of key state turn it to integer anve to choice variable
        choice = int(data['state'])
        #choose what console command to execute using if statement
        if  choice == 1:
                subprocess.call("python3 send.py -p 182 -t 1 21811", shell = True)
                print ("Socket on")
        elif  choice == 0:
                subprocess.call("python3 send.py -p 182 -t 1 21820", shell = True)
                print ("Socket off")
        else:
                print("Wrong!!!")
        return "ok"
```

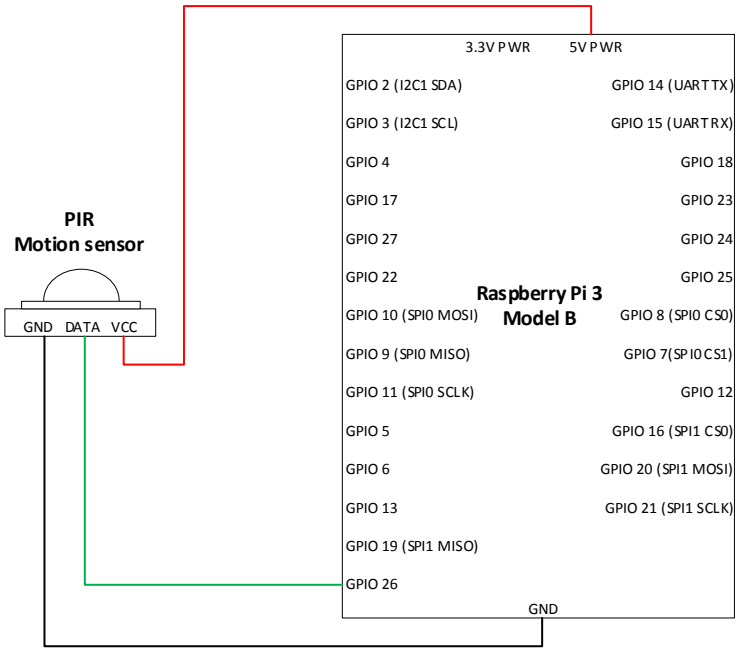
7. Αισθητήρας κίνησης

Για την παρακολούθηση από τον χρήστη της ύπαρξης κίνησης μέσα στο σπίτι για λόγους ασφαλείας, όταν εκείνος βρίσκεται μακριά χρησιμοποιείται σε επίπεδο hardware ένας PIR (Passive Infrared) Sensor[25]. Ο αισθητήρας αυτός αποτελείται από πυροηλεκτρικό υλικό, το οποίο μπορεί να ανιχνεύσει υπέρυθρη ακτινοβολία.

Συγκεκριμένα, ο PIR αισθητήρας αποτελείται από δυο τμήματα πυροηλεκτρικού υλικού. Όταν λοιπόν ο αισθητήρας είναι αδρανής και τα δυο τμήματα ανιχνεύουν την ίδια ποσότητα IR. Όταν όμως ένα ζεστό σώμα, όπως για παράδειγμα ένας άνθρωπος, διέλθει μπροστά από τον αισθητήρα, αρχικά αλληλεπιδρά με το πρώτο τμήμα του PIR αισθητήρα και αυτό δημιουργεί θετική διαφορά δυναμικού ανάμεσα στα δυο τμήματα του αισθητήρα. Στη συνέχεια, όταν το ζεστό σώμα εγκαταλείπει την περιοχή ανίχνευσης συμβαίνει το αντίστροφο, δηλαδή δημιουργείται αρνητική διαφορά δυναμικού ανάμεσα στα δυο τμήματα του αισθητήρα. Αυτός ο παλμός που καταγράφεται αποτελεί την ένδειξη ανίχνευσης κίνησης από τον αισθητήρα.

Ο IR αισθητήρας είναι μέσα σε ερμητικά σφραγισμένο μεταλλικό δοχείο για τη βελτίωση της θωράκισης από τον θόρυβο, την θερμοκρασία και την υγρασία. Υπάρχει ένα παράθυρο, κατασκευασμένο από υλικό που επιτρέπει τη διέλευση της υπέρυθρης ακτινοβολίας (συνήθως επικαλυμμένο πυρίτιο), που προστατεύει το αισθητήριο. Πίσω από αυτό το παράθυρο βρίσκονται οι δυο αισθητήρες.

Σε επίπεδο software, ο χρήστης δέχεται στην Android συσκευή του ειδοποίηση όταν ανιχνευτεί κίνηση ενώ έχει τη δυνατότητα επιλέγοντας το τέταρτο και τελευταίο κουμπί στην κεντρική οθόνη της εφαρμογής με τίτλο Security να μεταφερθεί στη κατάλληλη οθόνη ώστε να ενημερωθεί για το πόσες φορές ο αισθητήρας κατέγραψε κίνηση την τελευταία, για παράδειγμα, 1 ώρα. Αρχικά λοιπόν, όπως φαίνεται και στο παρακάτω σχεδιάγραμμα, συνδέεται ο PIR αισθητήρας κίνησης στην 5v παροχή του Raspberry, στη γείωση και για την επικοινωνία στο GPIO 26.



Πρέπει λοιπόν, να υπάρχει η δυνανότητα αποστολής ενημέρωσης στον χρήστη όταν ανιχνευτεί κίνηση, αλλά και να μπορεί ο χρήστης να μάθει το πόσες φορές ανιχνεύτηκε κίνηση από τον αισθητήρα σε κάποιο χρονικό διάστημα. Αυτό προϋποθέτει, εκτός από το κομμάτι του Server που εξυπηρετεί τα http requests να υπάρχει και ένα πρόγραμμα που να τρέχει παράλληλα και συνεχόμενα από τη στιγμή που γίνεται εκκίνηση του server. Αυτό το πρόγραμμα, διαχειρίζεται τόσο την αποστολή ειδοποιήσεων όταν ικανοποιούνται κάποια κριτήρια και καταγράφει για το επιθυμητό χρονικό διάστημα το πόσες φορές ανιχνεύθηκε κίνηση από τον αισθητήρα.

Στον server λοιπόν, για να τρέξει το πρόγραμμα αυτό που αναφέρθηκε παραπάνω δημιουργήθηκε ένα νέο thread για την function που θα τρέχει παράλληλα με το routing[26]. Η συγκεκριμένη function, βρίσκεται αυτή την φορά κάτω από τον decorator @app.before_first_request[27] ώστε να τρέχει ο κώδικας που περιέχεται σε αυτόν με την εκκίνηση του server χωρίς την ανάγκη να γίνει πρώτα κάποιο request από τον χρήστη.

```
@app.before_first_request
def motion():
    def motion_counter():
        #counter2 variable marked as global so it can be used in the routing
        global counter2
        pir_sensor = 26
        #set the period we keep the count of detected movements
        now = datetime.datetime.now()
        tdelta = datetime.timedelta(minutes=5)
        wdif = now + tdelta

        GPIO.setmode(GPIO.BCM)

        GPIO.setup(pir_sensor, GPIO.IN)
        #initialize the variables
        current_state = 0
        counter = 0
        counter2 = 0
        #infinite loop
```

```
while True:
        #update now variable everytime
        now = datetime.datetime.now()
        #check sensor state
        current_state = GPIO.input(pir_sensor)
        #if time is past the set period update the variables
        if now >= wdif:
                wdif = now + tdelta
                counter2 = 0
        # if movement is detected update the counter variables
        if current_state == 1:
                print("GPIO pin %s is %s" % (pir_sensor, current_state))
                counter += 1
                counter2 += 1
                time.sleep(1)
                #if movement detected for five times in the set period send
                #notification
                if counter == 5:
                        subprocess.call("python3 notification.py", shell =
                        True)
                        counter = 0

Thread(target=motion_counter).start()
```

Σε αυτόν τον κώδικα, αρχικά με την χρήση του datetime module [28]καθορίζεται το χρονικό διάστημα για το οποίο ο χρήστης θα ενημερώνεται πόσες φορές ανιχνεύθηκε κίνηση. Για λόγους που αφορούν τον έλεγχο της σωστής λειτουργίας του προγράμματος το χρονικό διάστημα ορίστηκε στα πέντε λεπτά, αλλά πολύ εύκολα μπορεί να αλλάξει, για παράδειγμα, σε μια ώρα περνώντας ως παράμετρο hours=1 αντί για minutes=5 στην timedelta. Στην συνέχεια χρησιμοποιούνται δυο counters, ο ένας ώστε να σταλεί ειδοποίηση στον χρήστη όποτε ανιχνευθεί πέντε φορές κίνηση από τον αισθητήρα και ο δεύτερος για να απαρριθμεί τις φορές που ο αισθητήρας ανίχνευσε κίνηση μέσα στο χρονικό διάστημα που έχει καθοριστεί. Η ειδοποίηση δεν στέλνεται με το που ανιχνευθεί μια φορά κίνηση από τον αισθητήρα αλλά όταν αυτό γίνει πέντε φορές, καθώς αυτοί οι αισθητήρες είναι πολύ ευαίσθητοι και πολλές φορές

μπορεί για παράδειγμα να θεωρήσουν κίνηση την αλλαγή φωτισμού σε κάποιο αντικείμενο που βρίσκεται στην εμβέλειά τους. Οπότε, χρησιμοποιείται αυτός ο αριθμός ώστε όταν σταλεί η ειδοποίηση στον χρήστη υπάρχει όντως πιθανότητα να υπάρχει ανεπιθύμητη κίνηση μέσα στο σπίτι και δεν είναι απλά ένα λάθος λόγω της ευαισθησίας του αισθητήρα.

Από την πλευρά της εφαρμογής Android τώρα, έπρεπε να βρεθεί ένας τρόπος ώστε ο χρήστης να ενημερώνεται για το γεγονός ότι ανιχνεύθηκε κίνηση στο σπίτι με μόνη προϋπόθεση να είναι συνδεδεμένος στο internet και χωρίς να χρειάζεται να έχει ανοιχτή την εφαρμογή. Η λύση ήταν τα push notifications τα οποία στην περίπτωση του Android υλοποιούνται μέσω του Firebase Cloud Messaging (FCM), που είναι μια υπηρεσία στην οποία στέλνεται από τον server η ειδοποίηση, την οποία λαμβάνει ο χρήστης μόλις συνδεθεί στο internet. Η προσθήκη του FCM στην εφαρμογή, γίνεται με προσθήκη των κατάλληλων dependencies στο build.gradle της εφαρμογής και τη δημιουργία μιας νέας κλάσης, που είναι ορισμένη ως service και όχι ως activity στο manifest της εφαρμογής και κάνει extend την FirebaseMessagingService[29]. Σε αυτή την κλάση γίνεται override η onNewToken, η οποία καλείται όταν αλλάζει το token της εφαρμογής, δηλαδή το αναγνωριστικό της συσκευής από τον server, συνήθως για λόγους που αφορούν την ασφάλεια. Επιπλέον, γινεται override η onMessageReceived που δέχεται την ειδοποίηση από τον server και εισάγει τα δεδομένα της ως παραμέτρους στην μέθοδο sendNotification. Στην sendNotification, δημιουργείται ένα pending intent ώστε να καθοριστεί τι θα γίνει όταν ο χρήστης πατήσει πάνω στην ειδοποίηση, δημιουργείται ο notification manager ώστε να εμφανιστεί η ενημέρωση στον χρήστη και τέλος υπάρχει και ο Notification Builder ο οποίος καθορίζει τις διάφορες παραμέτρους, όπως τίτλο και εικονίδιο, της ενημέρωσης που τελικά θα εμφανιστεί.

```
NotificationCompat.Builder notificationBuilder = new NotificationCompat.Builder(this,
CHANNEL_ID)
        .setSmallIcon(R.mipmap.ic_launcher)
        .setContentTitle("Notification")
        .setContentText(messageBody)
        .setTicker("Alert!")
        .setAutoCancel(true)
        .setDefaults(Notification.DEFAULT_SOUND)
```

```
        .setContentIntent(pendingIntent)
        .setPriority(NotificationCompat.PRIORITY_DEFAULT);
```

Από την κονσόλα του project που έχει δημιουργηθεί στο Firebase για την εφαρμογή βρίσκουμε το server key το οποίο χρησιμοποιείται μαζί με το token στο python script notification.py που χρησιμοποιείται από τον server μας ώστε να αποσταλεί η ειδοποίηση στην εφαρμογή. Η αποστολή των ειδοποιήσεων μέσω του notification.py script όπως φαίνεται και στον παρακάτω κώδικα υλοποιείται με τη χρήση του pyfcm[30], ενός python client για το FCM.

```
from pyfcm import FCMNotification

push_service = FCMNotification(api_key="AAAAnFeEzk0:APA91bFS9Rhuzqb_fuk4o-
T3iW2TsYxCJ5So45Wvnd3vSFjS00Z_eI21bcNAqS.....")

# The api-key can be gotten from:  https://console.firebase.google.com/project/<project-
name>/settings/cloudmessaging

registration_id = "e5Wk69Lw_M4:APA91bE1vEBg-yOChNmlmTChG6rf4OA-
3Ht9MRRDBEcIJfeljWwbJMvyTsmYmCmSF0Esg....."
message_title = "Home security warning!"
message_body = "Sensors detected movement inside the house"
result = push_service.notify_single_device(registration_id=registration_id,
message_title=message_title, message_body=message_body)

print (result)
```

Τέλος, ο χρήστης ενημερώνεται για τις φορές που έχει ενεργοποιηθεί ο αισθητήρας το καθορισμένο χρονικό διάστημα όταν επισκέπτεται το security activity, με τη χρήση http requests προς τον server και συγκεκριμένα GET requests, όπως και στην περίπτωση των αισθητήρων θερμοκρασίας και υγρασίας. Το αντίστοιχο λοιπόν route χρησιμοποιεί το κατάλληλο counter από τη συνάρτηση που τρέχει παράλληλα με το routing και επιστρέφει με τη μορφή json την πληροφορία όταν ζητηθεί από τον χρήστη. Επιπλέον, ο χρήστης μπορεί να ανανεώσει τα δεδομένα με Swipe Down χωρίς να χρειάζεται να φύγει από το activity.

```
@app.route("/security", methods = ['GET'])
def sensor_counter():
        return jsonify({"alarms":counter2})
```

8. Απομακρυσμένη πρόσβαση

Όλες λοιπόν οι παραπάνω λειτουργίες που προσφέρει ο server, είναι προσβάσιμες με τη χρήση της εφαρμογής Android όταν τόσο το κινητό τηλέφωνο στο οποίο ειναι εγκατεστημένη η εφαρμογή όσο και το Raspberry είναι συνδεδεμένα στο ίδιο τοπικό δίκτυο δηλαδή στο ίδιο router. Όμως, για να μιλάμε για απομακρυσμένο έλεγχο θα πρέπει οι λειτουργίες αυτές να είναι προσβάσιμες μέσω της εφαρμογής Android από οπουδήποτε κι αν συνδεθεί ο χρήστης στο internet. Κάτι τέτοιο προϋποθέτει ο server να είναι προσβάσιμος από το διαδίκτυο, να φαίνεται δηλαδή πίσω από το router του τοπικού δικτύου στο οποίο είναι συνδεδεμένος.

Αυτό γίνεται με το port forwarding, το οποίο ενεργοποιείται στις ρυθμίσεις του router και ορίζεται ένα μοναδικό port, το οποίο μαζί με τη δημόσια IP με την οποία το router είναι συνδεδεμένο στο διαδίκτυο, δίνει την δυνατότητα πρόσβασης στον server που είναι συνδεδεμένος με μια τοπική IP με το router[31],[32]. Συγκεκριμένα, από τις ρυθμίσεις port forwarding του router έγινε αντιστοίχηση του port 80, που είναι η προκαθορισμένη θύρα μέσω της οποίας ένας web server ακούει requests και στέλνει responses πρός το δίκτυο, με το port 5000 που είναι η θύρα μέσω της οποίας είναι προσβάσιμος ο Flask server στο τοπικό δίκτυο.

Ακόμα και μετά από αυτό, υπάρχει περίπτωση να μην υπάρχει απομακρυσμένη πρόσβαση στον server καθώς οι ISPs συνήθως παρέχουν πρόσβαση μέσω DHCP , το οποίο έχει ως αποτέλεσμα η δημόσια IP του δικτύου μας να αλλάζει ανα κάποιο χρονικό διάστημα. Σε πολλές περιπτώσεις μάλιστα, η δημόσια IP του τοπικού δικτύου αλλάζει κάθε φορά που γίνεται επανεκκίνηση του router. Μια λύση

σε αυτό θα μπορούσε να ήταν να ζητηθεί από τον ISP να μας αποδοθεί στατική IP, όμως κάτι τέτοιο είναι ακριβό και παρέχεται συνήθως στο πλαίσιο επαγγελματικών προγραμμάτων των παρόχων κάτι που ανεβάζει περαιτέρω το κόστος και κάνει το σύστημα της παρούσας εργασίας λιγότερο προσβάσιμο στους οικιακούς χρήστες στους οποίους κιόλας απευθύνεται. Έτσι οδηγούμαστε στο ότι λύση σε αυτό το πρόβλημα είναι η χρήση μιας υπηρεσίας Dynamic DNS(Domain Name System), η οποία ανεξάρτητα από τις αλλαγές της δημόσιας IP καθιστά το δίκτυο προσβάσιμο μέσω ενός σταθερού domain name[33],[34]. Αναλυτικότερα, γίνεται εγγραφή σε μια από τις εταιρίες, όπως η No-IP, που παρέχουν τέτοιες υπηρεσίες και επιλέγεται ένα hostname. Στη συνέχεια, εγκαθίσταται στο Raspberry ο Dynamic Update Client (DUC) του No-IP[35], που είναι μια εφαρμογή η οποία ελέγχει κατά διαστήματα την διεύθυνση IP και αν δει ότι αυτή έχει αλλάξει ενημερώνει τον πάροχο DDNS ώστε να γίνει αντιστοίχηση της νέας IP διεύθυνσης με το επιλεγμένο από εμάς domain name. Η εγκατάσταση του update client, γίνεται αρχικά με τη δημιουργία ενός νέου directory και το κατέβασμα στη συνέχεια του κατάλληλου software με τις παρακάτω εντολές.

```
mkdir /home/pi/noip
cd /home/pi/noip
wget http://www.no-ip.com/client/linux/noip-duc-linux.tar.gz
tar vzxf noip-duc-linux.tar.gz
```

Μεταβαίνουμε στη συνέχεια στο directory που δημιουργήθηκε και εγκαθιστούμε το λογισμικό με την χρήση της εντολής sudo make install. Τέλος, γίνεται login στον no-ip λογαριασμό και γίνεται εκκίνηση της υπηρεσίας με τη χρήση της εντολής sudo /usr/local/bin/noip2. Μπορεί να γίνει και επιβεβαίωση του οτι τρέχει κανονικά η υπηρεσία με την εντολή sudo noip2 -S.

Αξίζει να σημειωθεί ότι η δουλειά που γίνεται από τον update client που εγκαταστάθηκε στο Raspberry, δηλαδή η ενημέρωση του παρόχου Dynamic DNS για την αλλαγή της δημόσιας IP του δικτύου μας, μπορεί να γίνει και μέσω του router του δικτύου. Συγκεκριμένα, πολλά router υποστηρίζουν τις υπηρεσίες DDNS και δίνουν την δυνατότητα, αφού ο χρήστης συμπληρώσει μια φόρμα με τις παραμέτρους της DDNS υπηρεσίας, όπως ο κωδικός, το όνομα χρήστη και το hostname που έχει επιλεγεί, να ενημερώνουν τον πάροχο της υπηρεσίας για την αλλαγή της διεύθυνσης

IP. Αυτή η δυνατότητα του router δεν χρησιμοποιήθηκε στην παρούσα εργασία, καθώς το κάθε router ανάλογα με τον κατασκευαστή του και την έκδοση του λογισμικού του υποστηρίζει διαφορετικούς παρόχους DDNS. Για τον λόγο αυτό, προτιμήθηκε η εγκατάσταση του update client στο Raspberry Pi, ώστε το σύστημα να είναι ανεξάρτητο από το router του δικτύου.

9. Συμπεράσματα

Η παρούσα διπλωματική είχε ως στόχο την υλοποίηση ενός συστήματος απομακρυσμένου ελέγχου σπιτιού με τη χρήση μιας εφαρμογής Android. Αυτό το σύστημα έχει ως βάση του έναν server που τρέχει σε Raspberry Pi, ο οποίος χρησιμοποιείται για τον χειρισμό των ηλεκτρικών και ηλεκτρονικών συσκευών, καθώς και για την ενημέρωση του χρήστη της εφαρμογής σχετικά με τις συνθήκες που επικρατούν στο σπίτι με τη χρήση των κατάλληλων αισθητήρων.

Αρχικά, έγινε περιγραφή του server ο οποίος τρέχει σε ένα Raspberry Pi και υλοποιείται με τη χρήση του Flask, ενός Micro web framework της Python το οποίο επιλέχθηκε λόγω του ότι σε αντίθεση με άλλα web framework, όπως για παράδειγμα το django, δεν ήταν από την αρχή φορτωμένο με όλες τις βιβλιοθήκες και τα εργαλεία που χρησιμεύουν στην δημιουργία μιας web app, αλλά δεν θα είχαν καμία χρησιμότητα στην δημιουργία του server για το παρόν σύστημα. Ο server χρησιμοποιεί τα GPIO του Raspberry για την επικοινωνία με τους πομπούς, τους δέκτες και τους αισθητήρες που χρησιμοποιούνται για τον έλεγχο των συσκευών και την ενημέρωση του χρήστη μέσω της εφαρμογής.

Στην συνέχεια, έγινε περιγραφή της εφαρμογής Android, η οποία στην αρχική της οθόνη δίνει την δυνατότητα στον χρήστη να επιλέξει κάποια από τις λειτουργίες του συστήματος. Οι λειτουργίες αυτές είναι η ενημέρωση για την θερμοκρασία και την υγρασία στον χώρο του σπίτιού, ο χειρισμός IR συσκευών, ο χειρισμός RF συσκευών και η ενημέρωση του αν ανιχνεύθηκε κίνηση στον χώρο του σπιτιού από

τον αντίστοιχο αισθητήρα. Επιπλέον, κάθε φορά που ανιχνευθεί 5 φορές κίνηση από τον αισθητήρα αποστέλεται στον χρήστη ειδοποίηση με μόνη προυπόθεση να είναι συνδεδεμένος στο internet, χωρίς δηλαδή να χρειάζεται να έχει ανοικτή την εφαρμογή.

Ακολούθως, περιγράφηκε αναλυτικά η υλοποίηση των τεσσάρων λειτουργιών του συστήματος τόσο σε επίπεδο server όσο και σε επίπεδο εφαρμογής. Συγκεκριμένα, αναλύθηκε η υλοποίηση του REST API στον server ώστε να διαχειρίζεται κατάλληλα τα http requests που δέχεται κατά περίπτωση από την εφαρμογή, η οποία χρησιμοποιεί την βιβλιοθήκη Volley του Android για την αποστολή των επιθυμητών requests. Επιπλέον, στον server δημιουργείται ένα ακόμα thread που τρέχει παράλληλα με το routing που χρησιμοποιείται για το REST API, καθώς χρειάζεται να ελέγχεται συνεχώς η κατάσταση του αισθητήρα κίνησης ώστε να σταλεί ειδοποίηση στον χρήστη αν χρειαστεί αλλά και να υπάρχει η δυνατότητα ενημέρωσής του για το πόσες φορές τις τελευταίες ώρες έχει ανιχνευθεί κίνηση.

Τέλος, για να είναι προσβάσιμες από παντού οι λειτουργίες του server μέσω της εφαρμογής Android, γίνεται Port forwarding στο router ώστε τα request που έρχονται για τον server στην IP μας να προωθούνται σε αυτόν. Επειδή όμως η IP του router μας, ουσιαστικά δηλαδή η διεύθυνσή μας στο internet, μεταβάλεται συνεχώς λόγω της χρήσης DCHP από τους ISPs για την απόδοση διευθύνσεων, οδηγούμαστε και στην χρήση των υπηρεσιών Dynamic DNS. Με τις υπηρεσίες αυτές είναι δυνατή η πρόσβαση στον server από παντού, ανεξάρτητα από τις αλλαγές στην IP διεύθυνση του router, με τη χρήση ενός συγκεκριμένου domain name που κατοχυρώνουμε.

10. Παράρτημα

A. Κώδικας server σε Python (Flask framework)

```
from flask import Flask, render_template, jsonify, request
import subprocess
import re
import RPi.GPIO as GPIO
```

```
import datetime
import time
import dht11
#import motion
from threading import Thread

app = Flask(__name__)


@app.before_first_request
def  motion():
        def  motion_counter():
                #counter2 variable marked as global so it can be used in the routing
                global counter2
                pir_sensor = 26
                #set the period we keep the count of detected movements
                now = datetime.datetime.now()
                tdelta = datetime.timedelta(minutes=5)
                wdif = now + tdelta

                GPIO.setmode(GPIO.BCM)

                GPIO.setup(pir_sensor, GPIO.IN)
                #initialize the variables
                current_state = 0
                counter = 0
                counter2 = 0
                #infinite loop
                while True:
                        #update now variable everytime
                        now = datetime.datetime.now()
                        #check sensor state
                        current_state = GPIO.input(pir_sensor)
                        #if time is past the set period update the variables
                        if now >= wdif:
                                wdif = now + tdelta
                                counter2 = 0
                        # if movement is detected update the counter variables
                        if current_state == 1:
                                print("GPIO pin %s is %s" % (pir_sensor, current_state))
```

```
                                        counter += 1
                                        counter2 += 1
                                        time.sleep(1)
                                        #if movement detected for five times in the set period send
                                        notification
                                        if counter == 5:
                                                subprocess.call("python3 notification.py", shell = True)
                                                counter = 0

        Thread(target=motion_counter).start()



@app.route("/sensors", methods = ['GET'])
def get_mes():
        # initialize GPIO
        GPIO.setwarnings(False)
        GPIO.setmode(GPIO.BCM)
        # set the GPIO used for data
        instance = dht11.DHT11(pin=13)
        # get sensor data
        result = instance.read()
        # use variables to get temperature and humidity data seperately
        therm = result.temperature
        hum = result.humidity
        # return json with sensor data
        return jsonify({"temperature": therm,"humidity":hum})

@app.route("/ir", methods = ['POST'])
def   ir_remote():
        #receive json and save in a variable
        data = request.get_json()
        #get the value of the key button and save it to button variable
        button = data['button']
        #choose what console command to execute using if statement
        if  button == "power":
                subprocess.call("irsend SEND_ONCE samsung KEY_POWER", shell = True)
        elif  button == "volUp":
                subprocess.call("irsend SEND_ONCE samsung KEY_VOLUMEUP", shell = True)
        elif  button == "volDown":
```

```
                subprocess.call("irsend SEND_ONCE samsung KEY_VOLUMEDOWN", shell =
                True)
        elif button == "chUp":
                subprocess.call("irsend SEND_ONCE samsung KEY_CHANNELUP", shell = True)
        elif  button == "chDown":
                subprocess.call("irsend SEND_ONCE samsung KEY_CHANNELDOWN", shell =
                True)
        else:
                print("Wrong!!!")
        return  "ok"

@app.route("/rf", methods = ['POST'])
def rf_remote():
        #receive json and save in a variable
        data = request.get_json()
        #get the value of key state turn it to integer anve to choice variable
        choice = int(data['state'])
        #choose what console command to execute using if statement
        if choice == 1:
                subprocess.call("python3 send.py -p 182 -t 1 21811", shell = True)
                print ("Socket on")
        elif choice == 0:
                subprocess.call("python3 send.py -p 182 -t 1 21820", shell = True)
                print ("Socket off")
        else:
                print("Wrong!!!")
        return "ok"

@app.route("/security", methods = ['GET'])
def sensor_counter():
        return jsonify({"alarms":counter2})



if __name__=="__main__":
   app.run(host='0.0.0.0', debug=True)
```

Β.Κώδικας εφαρμογής Android

i.Κώδικας Java κεντρικής οθόνης της εφαρμογής

```
import android.content.Intent;
import android.support.annotation.NonNull;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.JsonObjectRequest;
import com.android.volley.toolbox.Volley;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.iid.FirebaseInstanceId;
import com.google.firebase.iid.InstanceIdResult;

import org.json.JSONException;
import org.json.JSONObject;

import java.util.HashMap;
import java.util.Map;

public class MainActivity extends AppCompatActivity {

    private static final String TAG = "MainActivity";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button Button1 = findViewById(R.id.first_button);
        Button Button2 = findViewById(R.id.second_button);
```

```
        Button Button3 = findViewById(R.id.third_button);
        Button Button4 = findViewById(R.id.fourth_button);


        Button1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, Sensors.class);
                startActivity(intent);
            }
        });

        Button2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, IRremote.class);
                startActivity(intent);
            }
        });

        Button3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, RFremote.class);
                startActivity(intent);
            }
        });

        Button4.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, Security.class);
                startActivity(intent);
            }
        });

    }
}
```

ii.Κώδικας Java της οθόνης της εφαρμογής που αφορά τον αισθητήρα θερμοκρασίας/υγρασίας

```
import android.support.v4.widget.SwipeRefreshLayout;
import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.JsonObjectRequest;
import com.android.volley.toolbox.Volley;

import org.json.JSONException;
import org.json.JSONObject;

public class Sensors extends AppCompatActivity implements SwipeRefreshLayout.OnRefreshListener
{

    private TextView TempText, HumText;
    String url ="http://192.168.43.85:5000/sensors";
    private SwipeRefreshLayout sr;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_sensors);
        TempText = findViewById(R.id.textView1);
        HumText = findViewById(R.id.textView2);
        sr = findViewById(R.id.sr);
        sr.setOnRefreshListener(this);

        ActionBar actionBar = getSupportActionBar();
        actionBar.setTitle("Sensors");

        get_readings();
```

```
}

@Override
public void onRefresh() {
   get_readings();
   sr.setRefreshing(false);
}

public void get_readings(){
   RequestQueue requestqueue = Volley.newRequestQueue(this);

   JsonObjectRequest objectRequest = new JsonObjectRequest(Request.Method.GET, url, null,
         new Response.Listener<JSONObject>() {
            @Override
            public void onResponse(JSONObject response) {
               try{
                  Log.d("response", response.toString()); //json response
                  int Temp = response.getInt("temperature");
                  int Humidity = response.getInt("humidity");

                  TempText.setText(String.valueOf(Temp)+" C");
                  HumText.setText(String.valueOf(Humidity)+" %");
               }
               catch (JSONException e){
                  e.printStackTrace();
               }
            }
         },
         new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
               TempText.setText("-");
               HumText.setText("-");
            }
         });

   objectRequest.setShouldCache(false);
   requestqueue.add(objectRequest);
}
```

iii.Κώδικας Java της οθόνης της εφαρμογής που αφορά τον χειρισμό συσκευών μέσω IR

```
package com.example.homeauto;

import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.JsonObjectRequest;
import com.android.volley.toolbox.Volley;

import org.json.JSONObject;

import java.util.HashMap;

public class IRremote extends AppCompatActivity {

    String url = "http://192.168.43.85:5000/ir";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_irremote);
        Button powerButton = findViewById(R.id.pwrBtn);
        Button volUpButton = findViewById(R.id.volUpBtn);
        Button volDownButton = findViewById(R.id.volDownBtn);
        Button chUpButton = findViewById(R.id.chUpBtn);
        Button chDownButton = findViewById(R.id.chDownBtn);

        ActionBar actionBar = getSupportActionBar();
        actionBar.setTitle("IR Remote");

        powerButton.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
        public void onClick(View v) {
            powerControl();
        }
    });

    volUpButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            volumeUp();
        }
    });

    volDownButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            volumeDown();
        }
    });

    chUpButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            channelUp();
        }
    });

     chDownButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            channelDown();
        }
    });
}

private void powerControl(){

    HashMap<String, String> params = new HashMap<>();
    params.put("button", "power");

    RequestQueue postqueue = Volley.newRequestQueue(this);
```

```
    JsonObjectRequest postRequest = new JsonObjectRequest(Request.Method.POST, url, new
JSONObject(params) , new Response.Listener<JSONObject>() {
        @Override
        public void onResponse(JSONObject response) {
          }
    },
            new Response.ErrorListener() {
               @Override
               public void onErrorResponse(VolleyError error) {
                  }
            });

    postqueue.add(postRequest);
  }

  private void volumeUp(){

    HashMap<String, String> params = new HashMap<>();
    params.put("button", "volUp");

    RequestQueue postqueue = Volley.newRequestQueue(this);

    JsonObjectRequest postRequest = new JsonObjectRequest(Request.Method.POST, url, new
JSONObject(params) ,
            new Response.Listener<JSONObject>() {
        @Override
        public void onResponse(JSONObject response) {
           }
    },
            new Response.ErrorListener() {
               @Override
               public void onErrorResponse(VolleyError error) {
                }
            });

    postqueue.add(postRequest);
  }

  private void volumeDown(){
```

```
        HashMap<String, String> params = new HashMap<>();
        params.put("button", "volDown");

        RequestQueue postqueue = Volley.newRequestQueue(this);

        JsonObjectRequest postRequest = new JsonObjectRequest(Request.Method.POST, url, new
JSONObject(params) , new Response.Listener<JSONObject>() {
            @Override
            public void onResponse(JSONObject response) {
             }
        },
                new Response.ErrorListener() {
                    @Override
                    public void onErrorResponse(VolleyError error) {
                     }
                });

        postqueue.add(postRequest);
    }

    private void channelUp(){

         HashMap<String, String> params = new HashMap<>();
        params.put("button", "chUp");

        RequestQueue postqueue = Volley.newRequestQueue(this);

        JsonObjectRequest postRequest = new JsonObjectRequest(Request.Method.POST, url, new
JSONObject(params) , new Response.Listener<JSONObject>() {
            @Override
            public void onResponse(JSONObject response) {
              }
        },
                new Response.ErrorListener() {
                    @Override
                    public void onErrorResponse(VolleyError error) {
                     }
                });
```

```
        postqueue.add(postRequest);
    }

    private void channelDown(){

        HashMap<String, String> params = new HashMap<>();
        params.put("button", "chDown");

        RequestQueue postqueue = Volley.newRequestQueue(this);

        JsonObjectRequest postRequest = new JsonObjectRequest(Request.Method.POST, url, new
JSONObject(params) , new Response.Listener<JSONObject>() {
            @Override
            public void onResponse(JSONObject response) {
             }
        },
                new Response.ErrorListener() {
                    @Override
                   public void onErrorResponse(VolleyError error) {
                  }
                });

        postqueue.add(postRequest);
    }
}
```

iv.Κώδικας Java της οθόνης της εφαρμογής που αφορά τον χειρισμό συσκευών μέσω RF

```
import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.JsonObjectRequest;
```

```
import com.android.volley.toolbox.Volley;
import org.json.JSONObject;
import java.util.HashMap;

public class RFremote extends AppCompatActivity {

    String url = "http://192.168.43.85:5000/rf";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_rfremote);
        Button onButton = findViewById(R.id.onBtn);
        Button offButton = findViewById(R.id.offBtn);

        ActionBar actionBar = getSupportActionBar();
        actionBar.setTitle("RF Remote");

        onButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                outletOn();
            }
        });

        offButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                outletOff();
            }
        });

    }

    private void outletOn(){

        HashMap<String, String> params = new HashMap<>();
        params.put("state", "1");

        RequestQueue postqueue1 = Volley.newRequestQueue(this);
```

```
    JsonObjectRequest postRequest1 = new JsonObjectRequest(Request.Method.POST, url, new
JSONObject(params) ,
            new Response.Listener<JSONObject>() {
        @Override
        public void onResponse(JSONObject response) {
         }
    },
            new Response.ErrorListener() {
               @Override
               public void onErrorResponse(VolleyError error) {
                }
            });

    postqueue1.add(postRequest1);
  }

  private void outletOff(){

    HashMap<String, String> params = new HashMap<>();
    params.put("state", "0");

    RequestQueue postqueue2 = Volley.newRequestQueue(this);

    JsonObjectRequest postRequest2 = new JsonObjectRequest(Request.Method.POST, url, new
JSONObject(params) , new Response.Listener<JSONObject>() {
        @Override
        public void onResponse(JSONObject response) {
         }
    },
            new Response.ErrorListener() {
               @Override
               public void onErrorResponse(VolleyError error) {
                }
            });

    postqueue2.add(postRequest2);
  }
}
```

v.Κώδικας Java της οθόνης της εφαρμογής που αφορά τον αισθητήρα κίνησης

```
import android.support.v4.widget.SwipeRefreshLayout;
import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;

import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.JsonObjectRequest;
import com.android.volley.toolbox.Volley;

import org.json.JSONException;
import org.json.JSONObject;

public class Security extends AppCompatActivity implements SwipeRefreshLayout.OnRefreshListener
{

    private TextView SecText;
    String url ="http://192.168.43.85:5000/security";
    private SwipeRefreshLayout sr2;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_security);
        SecText = findViewById(R.id.security_text);
        sr2 = (SwipeRefreshLayout)findViewById(R.id.sr2);
        sr2.setOnRefreshListener(this);

        ActionBar actionBar = getSupportActionBar();
        actionBar.setTitle("Security");

        get_movement();
    }

    public void get_movement(){
```

```
        RequestQueue requestqueue2 = Volley.newRequestQueue(this);

        JsonObjectRequest objectRequest = new JsonObjectRequest(Request.Method.GET, url, null,
                new Response.Listener<JSONObject>() {
                    @Override
                    public void onResponse(JSONObject response) {
                        try{
                            int AlarmsCount = response.getInt("alarms");

                            SecText.setText("Movement detected " + String.valueOf(AlarmsCount)+ " times in
the last 12 hours");
                        }
                        catch (JSONException e){
                            e.printStackTrace();
                        }
                    }
                },
                new Response.ErrorListener() {
                    @Override
                    public void onErrorResponse(VolleyError error) {
                        SecText.setText("Server connection Error");
                    }
                });

        objectRequest.setShouldCache(false);
        requestqueue2.add(objectRequest);
    }

    @Override
    public void onRefresh() {
        get_movement();
        sr2.setRefreshing(false);
    }
}
```

vi.Κώδικας Java της εφαρμογής που αφορά τη σύνδεση με την υπηρεσία Firebase Messaging και την προβολή ειδοποιήσεων στον χρήστη

```
import android.app.Notification;
import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.graphics.Color;
import android.media.RingtoneManager;
import android.net.Uri;
import android.os.Build;
import android.support.v4.app.NotificationCompat;
import android.util.Log;

import com.google.firebase.messaging.FirebaseMessagingService;
import com.google.firebase.messaging.RemoteMessage;

public class MyFirebaseMessaging extends FirebaseMessagingService {

   @Override
   public void onNewToken(String token) {
      super.onNewToken(token);
      Log.d("TAG", "Refreshed token: " + token);
      // sendRegistrationToServer(token);
   }

   @Override
   public void onMessageReceived(RemoteMessage remoteMessage) {
      Log.e("TAG", "From: " + remoteMessage.getFrom());
      Log.e("TAG", "Notification Message Body: " + remoteMessage.getData().get("CardName")+"  :
"+remoteMessage.getData().get("CardCode"));
      if (remoteMessage.getNotification()!= null)
      { sendNotification(remoteMessage.getNotification().getBody()); }
      else{Log.e("TAG", "Notification failed");}

   }
```

```
    private void sendNotification(String messageBody) {
        Intent intent = new Intent(this, Security.class);
        String CHANNEL_ID = "my_channel_id";
        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        PendingIntent pendingIntent = PendingIntent.getActivity(this, 0 /* Request code */, intent,
                PendingIntent.FLAG_ONE_SHOT);

        NotificationManager notificationManager =
                (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            NotificationChannel notificationChannel = new NotificationChannel(CHANNEL_ID, "My
Notifications", NotificationManager.IMPORTANCE_HIGH);
            if (notificationManager != null) {
                // Configure the notification channel.
                notificationChannel.setDescription("Channel description");
                notificationChannel.enableLights(true);
                notificationChannel.setLightColor(Color.RED);
                notificationChannel.setVibrationPattern(new long[]{0, 1000, 500, 1000});
                notificationChannel.enableVibration(true);
                notificationManager.createNotificationChannel(notificationChannel);
            }
        }

        NotificationCompat.Builder notificationBuilder = new NotificationCompat.Builder(this,
CHANNEL_ID)
                .setSmallIcon(R.mipmap.ic_launcher)
                .setContentTitle("Notification")
                .setContentText(messageBody)
                .setTicker("Alert!")
                .setAutoCancel(true)
                .setDefaults(Notification.DEFAULT_SOUND)
                .setContentIntent(pendingIntent)
                .setPriority(NotificationCompat.PRIORITY_DEFAULT);

        notificationManager.notify(0 /* ID of notification */, notificationBuilder.build());
    }
}
```

11. Βιβλιογραφία

[1]www.raspberrypi.org Raspberry Pi Foundation

[2]en.wikipedia.org/wiki/Raspberry_Pi Raspberry Pi-Wikipedia

[3]https://www.raspbian.org Raspbian home page

[4]https://en.wikipedia.org/wiki/Raspbian Raspbian-Wikipedia

[5]http://flask.pocoo.org Flask a Python Microframework

[6]flask-restful.readthedocs.io Flask-RESTful 0.3.6 documentation

[7]https://android-developers.googleblog.com/2008/09/announcing-android-10-sdk-release-1.html Android Developers Blog:Announcing the Android 1.0 SDK, Release 1

[8]https://en.wikipedia.org/wiki/Android_version_history Android version history - Wikipedia

[9]www.android.com/history/ Android - History

[10]https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/ Mobile OS market share 2018|Statista

[11]https://developer.android.com/studio/intro/ Meet Android Studio | Android Developers

[12]https://developer.apple.com/documentation/xcode_release_notes/xcode_10_release_notes Xcode 10 Release Notes | Apple Developer Documentation

[13]https://developer.android.com/about/dashboards/ Distribution dashboard | Android Developers

[14]https://developer.android.com/training/swipe/add-swipe-interface Adding Swipe-to-Refresh To Your App | Android Developers

[15]http://www.circuitbasics.com/how-to-set-up-the-dht11-humidity-sensor-on-an-arduino/ How to set up the DHT11 humidity sensor on an Arduino

[16]https://github.com/szazo/DHT11_Python Github - szazo/DHT11_Python:Pure Python library for reading DHT11 sensor on Raspberry Pi

[17]https://developer.android.com/training/volley/ Volley overview | Android Developers

[18] http://www.circuitbasics.com/arduino-ir-remote-receiver-tutorial/ How to set up an IR remote and receiver on an arduino

[19]http://www.lirc.org Linux Infrared Remote Control

[20]https://www.piddlerintheroot.com/ir-blaster-lirc/ IR Blaster (LIRC)-piddlerintheroot

[21]https://www.cnx-software.com/2017/03/12/how-to-control-your-air-conditioner-with-raspberry-pi-board-and-anavi-infrared-phat/ How to Control Your Air Conditioner with Raspberry Pi Board and ANAVI Infrared pHAT

[22]https://www.engineersgarage.com/contribution/Basic-Model-of-RF-Transmitter-Receiver Basic Model of RF Transmitter and Receiver|EngineersGarage

[23]https://pypi.org/project/rpi-rf/ rpi-rf -PyPI

[24]https://www.piddlerintheroot.com/rf-433-mhz/ RF 433 MHZ-piddlerintheroot

[25]https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/overview Overview|PIR Motion Sensor|Adafruit Learning System

[26]https://networklore.com/start-task-with-flask/ Starting a task at startup in Flask

[27]http://flask.pocoo.org/docs/1.0/api/ API - Flask 1.0.2 documentation

[28]https://docs.python.org/3.2/library/datetime.html 7.1. datetime — Basic date and time types — Python v3.2.6 documentation

[29]https://firebase.google.com/docs/cloud-messaging/android/client Set up a Firebase Cloud Messaging client app on Android | Firebase

[30]https://pypi.org/project/pyfcm/ pyfcm - PyPI

[31]en.wikipedia.org/wiki/Port_forwarding Port forwarding - Wikipedia

[32]https://www.pcmag.com/encyclopedia/term/49509/port-forwarding port forwarding Definition from PC Magazine Encyclopedia

[33]https://www.howtogeek.com/66438/how-to-easily-access-your-home-network-from-anywhere-with-ddns/ How To Easily Access Your Home Network From Anywhere With Dynamic DNS

[34]https://www.lifewire.com/definition-of-dynamic-dns-816294 What's DDNS and How Does It Work

[35]https://www.noip.com/support/knowledgebase/install-ip-duc-onto-raspberry-pi/ How to Install the No-IP DUC on Raspberry Pi | Support | No-IP Knowledge Base