**ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ**

**ΚΟΙΝΩΝΙΚΩΝ ΚΑΙ ΠΟΛΙΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ**

**ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ**

**Πτυχιακή Εργασία**

**Αυτοματισμός Κατοικίας**



Του φοιτητή Επιβλέπων Καθηγητής

Παχούμη Ιωάννη Χατζηγεωργίου Αλέξανδρος

Αρ. Μητρώου: 28/10

**Θεσσαλονίκη, Ιούνιος 2016**

# Πρόλογος

Η παρούσα πτυχιακή εργασία αναλύει την διαδικασία προγραμματισμού ενός συστήματος αυτοματισμού κατοικίας με την χρήση κινητής συσκευής για τον έλεγχο του. Το συγκεκριμένο θέμα επιλέχθηκε λόγω της επαναφοράς της τεχνολογίας στο προσκήνιο του τομέα της πληροφορικής αλλά και της πεποίθησης του συγγραφέα ότι η πληροφορική πρέπει να διευκολύνει την ζωή σε πιο άμεσο και φυσικό επίπεδο έτσι ώστε να εναγκαλιστεί από τον μέσο άνθρωπο σαν απαραίτητο κομμάτι της καθημερινότητας του.

Στα πλαίσια της πτυχιακής αυτής αναπτύχθηκαν τρία προγράμματα που υλοποιούν το σύστημα αυτοματισμού κατοικίας. Το πρώτο κομμάτι αφορά το υλικό ανοικτoύ κώδικα Arduino και προσομοιώνει τις πιο σημαντικές συσκευές ενός έξυπνου σπιτιού. Το δεύτερο κομμάτι αφορά έναν διακομιστή με τον οποίο επικοινωνεί το Arduino και στον οποίο ενημερώνεται η κατάσταση κάθε συσκευής. Το τρίτο και τελευταίο κομμάτι είναι μια εφαρμογή για κινητά από την οποία ο χρήστης μπορεί να ενημερωθεί για την κατάσταση του σπιτιού καθώς και να χρησιμοποιήσει τις συσκευές από απόσταση.

# Περίληψη

Στην εποχή μας οι υπολογιστές και η πληροφορική είναι απαραίτητα εργαλεία για την εξέλιξη της ανθρωπότητας. Χρησιμοποιούνται σχεδόν παντού και κάνουν την ζωή μας ευκολότερη. Ειδικά την τελευταία δεκαετία η χρήση των smartphones και των tablets αποδεικνύει πόσο θελκτική και αναγκαία έχει γίνει πια η χρήση ενός προσωπικού βοηθού στην μορφή υπολογιστή. Φυσικά το ίδιο μπορεί να γίνει και με το σπίτι μας. Ο αυτοματισμός κατοικίας είναι ένα σύστημα που διευκολύνει την διαχείρηση των συσκευών μέσα στο σπίτι με την χρήση αισθητήρων. Η πτυχιακή αυτή εργασία αναλύει το πώς μπορούμε να υλοποιήσουμε ένα σύστημα αυτοματισμού χρησιμοποιώντας υλικό ανοιχτού κώδικα Arduino.

# Abstract

In our times, information technolofy is an important tool guiding the evolution of mankind. It has a use almost everywhere and makes peoples’ lives easier. Especially in the last decade, the use of smartphones and tablets proves how alluring and necessary the use of a personal assistant in a form of a computer has become. The same can be done for our house. Home automation is a system that facilitates the management of devices inside the house by using sensors. This thesis analyses the development of such a system using the Arduino open-source hardware.

# Περιεχόμενα

[Πρόλογος 2](#_Toc453449253)

[Περίληψη 3](#_Toc453449254)

[Abstract 4](#_Toc453449255)

[Περιεχόμενα 5](#_Toc453449256)

[1. Εισαγωγή 7](#_Toc453449257)

[1.1 Έξυπνο Σπίτι 7](#_Toc453449258)

[1.2 Στόχοι της εργασίας 7](#_Toc453449259)

[1.3 Διάρθρωση της εργασίας 8](#_Toc453449260)

[2. Αυτοματισμός Κατοικίας 9](#_Toc453449261)

[2.1 Τεχνολογίες Αυτοματισμού Κατοικίας 9](#_Toc453449262)

[2.1.1 Πρωτόκολλο Χ10 9](#_Toc453449263)

[2.1.2 Πρωτόκολλο UPB 9](#_Toc453449264)

[2.1.3 Insteon 10](#_Toc453449265)

[2.1.4 Z-Wave 10](#_Toc453449266)

[2.1.5 ZigBee 11](#_Toc453449267)

[2.2 Εμπορικές Λύσεις 11](#_Toc453449268)

[2.2.1 Homeseer 12](#_Toc453449269)

[2.2.2 Control4 13](#_Toc453449270)

[2.2.3 Crestron 14](#_Toc453449271)

[3. Υπηρεσίες Ιστού 15](#_Toc453449272)

[3.1 Βασική Γνώση 15](#_Toc453449273)

[3.2 Υπηρεσίες Ιστού REST 16](#_Toc453449274)

[3.3 Προγραμματισμός Διακομιστή 17](#_Toc453449275)

[3.3.1 Υλοποίηση Κλάσεων Υλικού 18](#_Toc453449276)

[3.3.2 Υλοποίηση Κλάσης Βάσης Δεδομένων 19](#_Toc453449277)

[3.3.3 Υλοποίηση της Υπηρεσίας 20](#_Toc453449278)

[4. Arduino 28](#_Toc453449279)

[4.1 Πλακέτα Arduino Uno 28](#_Toc453449280)

[4.2 Σχεδιασμός Κυκλώματος Αυτοματισμού Κατοικίας 30](#_Toc453449281)

[4.3 Προγραμματισμός Arduino 32](#_Toc453449282)

[4.3.1 Υλοποίηση Συναρτήσεων Αιτημάτων HTTP 33](#_Toc453449283)

[4.3.2 Υλοποίηση Κυρίως Προγράμματος Arduino 34](#_Toc453449284)

[5. Android 40](#_Toc453449285)

[5.1 Σχεδιασμός Εφαρμογής 40](#_Toc453449286)

[5.2 Υλοποίηση Βοηθητικών Μεθόδων 43](#_Toc453449287)

[5.2.1 Μέθοδοι HTTP 43](#_Toc453449288)

[5.2.2 Καθολικές Μέθοδοι 45](#_Toc453449289)

[5.3 Υλοποίηση των Activities 48](#_Toc453449290)

[Συμπεράσματα 56](#_Toc453449291)

[Αναφορές 57](#_Toc453449292)

[Παράρτημα 59](#_Toc453449293)

# 1. Εισαγωγή

Σε αυτό το κεφάλαιο θα αναφερθούμε εισαγωγικά στην τεχνολογία αυτοματισμού κατοικίας. Θα εξηγήσουμε τον όρο “Internet of Things” και θα κάνουμε μια μικρή ιστορική αναδρομή. Επίσης θα αναφέρουμε τους στόχους της εργασίας και θα περιγράψουμε επιγραμματικά την δομή των κεφαλαίων.

## 1.1 Έξυπνο Σπίτι

Την σημερινή εποχή, που οι ηλεκτρονικές συσκευές όπως smartphones και tablets, είναι αναπόσπαστο κομμάτι της καθημερινότητας, το επόμενο βήμα είναι η πλήρη ενσωμάτωση της τεχνολογίας με την καθημερινότητα μέσω του “Internet of Things”.

Ο όρος “Internet of Things” συμπεριλαμβάνει όλα τα φυσικά αντικείμενα, όπως συσκευές, αυτοκίνητα κ.α. στα οποία έχουν ενσωματωθεί αισθητήρες με σκοπό να συλλέγουν και να ανταλλάσουν δεδομένα. Απώτερος σκοπός τέτοιων συστημάτων είναι η μεγαλύτερη αποτελεσματικότητα και ακρίβεια καθώς και το οικονομικό όφελος.

Ιστορικά, πωλούνταν ως ολοκληρωμένα συστήματα όπου ο καταναλωτής στηριζόταν σε έναν προμηθευτή για το σύνολο του συστήματος, το οποίο συμπεριλάμβανε το υλικό, το πρωτόκολλο επικοινωνίας, τον κεντρικό κόμβο και το περιβάλλον εργασίας του χρήστη. Ωστόσο, υπάρχουν πλέον συστήματα ανοικτού λογισμικού που μπορούν να χρησιμοποιηθούν με ιδιόκτητο υλικό.

Αν και η τεχνολογία δείχνει να έχει μέλλον, υπάρχει πιθανότητα οι καταναλωτές να επενδύουν τα χρήματα τους σε μια τεχνολογία που ίσως καταλήξει «abandonware”, δηλαδή να εγκαταλειφθεί. Μια έρευνα της Microsoft το 2011 διαπίστωσε ότι η χρήση αυτοματισμού κατοικίας συνεπάγεται υψηλό κόστος ιδιοκτησίας, ακαμψία των διασυνδεδεμένων συσκευών καθώς και δυσκολία στην διαχείριση.

## 1.2 Στόχοι της εργασίας

Η παρούσα εργασία έχει ως στόχο να:

* Παρουσιάσει τις τεχνολογίες έξυπνου σπιτιού και να αναφέρει τις εμπορικές λύσεις πάνω στο θέμα του αυτοματισμού κατοικίας.
* Αναλύσει τις υπηρεσίες ιστού τύπου REST και να υλοποιήσει έναν τέτοιου είδους διακομιστή.
* Αναλύσει την δημιουργία ενός κυκλώματος αυτοματισμού κατοικίας.
* Παρουσιάσει το υλικό ανοικτού κώδικα Arduino και την διαδικασία προγραμματισμού πάνω σε αυτό.
* Παρουσιάσει το λειτουργικό σύστημα Android και να υλοποιήσει μια εφαρμογή ελέγχου έξυπνου σπιτιού.

## 1.3 Διάρθρωση της εργασίας

Η εργασία χωρίζεται σε 5 κεφάλαια μαζί με την εισαγωγή.

Στο 2ο κεφάλαιο αναλύουμε τις τεχνολογίες αυτοματισμού κατοικίας που υπάρχουν και αναφερόμαστε στις τρεις μεγαλύτερες εταιρίες του κλάδου και τα αντίστοιχα συστήματα.

Στο 3ο κεφάλαιο αναλύουμε τις υπηρεσίες ιστού, δίνοντας στον αναγνώστη μια βασική γνώση πάνω στο θέμα. Ακολουθεί η ανάλυση της αρχιτεκτονικής τύπου REST και τέλος η υλοποίηση ενός διακομιστή τύπου REST με κατάλληλο σχολιασμό.

Στο 4ο κεφάλαιο αναλύουμε το υλικό ανοικτού κώδικα Arduino, παρουσιάζοντας τα χαρακτηριστικά του. Παρουσιάζουμε τον τρόπο δημιουργίας ενός κυκλώματος έξυπνης κατοικίας και σχολιάζουμε τον κώδικα Arduino που γράφτηκε για αυτό το κύκλωμα.

Στο 5ο κεφάλαιο αναλύουμε το λειτουργικό σύστημα Android. Σχεδιάζουμε μια εφαρμογή ελέγχου του συστήματος αυτοματισμού κατοικίας και επεξηγούμε την υλοποίηση της.

# 2. Αυτοματισμός Κατοικίας

Από το πρωτόκολλο επικοινωνίας Χ10 που εμφανίστηκε το 1975, οι τεχνολογίες αυτοματισμού κατοικίας εξελίχθηκαν ραγδαία και γίνανε πολύπλευρες και αξιόπιστες. Στο κεφάλαιο αυτό θα αναφέρουμε τις βασικές τεχνολογίες έξυπνου σπιτιού και θα παρουσιάσουμε τις πιο γνωστές εμπορικές λύσεις όσων αφορά τον αυτοματισμό κατοικίας.

## 2.1 Τεχνολογίες Αυτοματισμού Κατοικίας

### 2.1.1 Πρωτόκολλο Χ10

Το X10 είναι ένα πρωτόκολλο επικοινωνίας μεταξύ των ηλεκτρονικών συσκευών που χρησιμοποιούνται για τον αυτοματισμό κατοικίας και το οποίο χρησιμοποιεί κυρίως τις γραμμές ηλεκτρικής ενέργειας για τη σηματοδότηση και τον έλεγχο του συστήματος. Τα σήματα αποτελούνται από σύντομους παλμούς ραδιοσυχνοτήτων που εκπροσωπούν ψηφιακές πληροφορίες. Περιλαμβάνει επίσης πρωτόκολλο ασύρματης λειτουργίας.

Το X10 αναπτύχθηκε το 1975 από την Pico Electronics με σκοπό τον απομακρυσμένο έλεγχο οικιακών συσκευών. Ήταν η πρώτη γενικού σκοπού δικτυακή τεχνολογία αυτοματισμού και παραμένει η πιο ευρέως διαθέσιμη.

Παρά το γεγονός ότι υπάρχει πληθώρα εναλλακτικών λύσεων, το X10 παραμένει δημοφιλής με εκατομμύρια μονάδες σε χρήση σε όλο τον κόσμο χάρη στην χαμηλή τιμή των συμβατών στοιχείων αυτοματισμού. (Smarthome.com, 2016)

### 2.1.2 Πρωτόκολλο UPB

Το Universal Powerline Bus (UPB) είναι ένα πρωτόκολλο για την επικοινωνία μεταξύ των συσκευών που χρησιμοποιούνται για την αυτοματοποίηση κατοικίας. Χρησιμοποιεί τη γραμμή ηλεκτρικής ενέργειας για τη σηματοδότηση και τον έλεγχο του συστήματος.

Aναπτύχθηκε από την εταιρία Powerline Control Systems (PCS) και κυκλοφόρησε το 1999. Έχει βάση το πρότυπο X10 αλλά με βελτιωμένο ρυθμό μετάδοσης και μεγαλύτερη αξιοπιστία.

Το UPB είναι ένα πολύ τεχνικό και περίπλοκο σύστημα. Λόγω της ισχύος του συστήματος, οι χρήστες μπορεί να νιώσουν πως το σύστημα είναι δύσκολο στην εγκατάσταση αρχικά, αλλά μετά από μια μικρή εμπειρία, η λογική του σχεδιασμού του συστήματος γίνεται εμφανής.

Η επικοινωνία μπορεί να είναι "peer to peer", χωρίς να είναι απαραίτητος ο κεντρικός ελεγκτής. Το πρωτόκολλο UPB επιτρέπει 250 συσκευές ανά κατοικία και 250 σπίτια ανά μετασχηματιστή. Οι διακόπτες μπορούν να συνυπάρχουν με άλλα συστήματα βασισμένα στην γραμμή ηλεκτρικής ενέργειας μέσα στο ίδιο σπίτι. Ενώ το UPB είναι πιο αποτελεσματικό από το X10, έχει πολύ λιγότερα προϊόντα διαθέσιμα στην αγορά. (Wikipedia, 2016)

### 2.1.3 Insteon

Το Insteon είναι μια τεχνολογία οικιακού αυτοματισμού που επιτρέπει σε διακόπτες, φώτα, θερμοστάτες, ανιχνευτές διαρροών, τηλεχειριστήρια, αισθητήρες κίνησης, και άλλες ηλεκτρικές συσκευές να λειτουργούν συνεργατικά μέσω των γραμμών ηλεκτρικής ενέργειας, μέσω ραδιοσυχνότητας (RF), ή μέσω και των δύο.

Χρησιμοποιεί μια τοπολογία δικτύωσης διπλού πλέγματος, σύμφωνα με την οποία όλες οι συσκευές είναι "peers" και κάθε συσκευή μεταδίδει, λαμβάνει, και επαναλαμβάνει μηνύματα. Αυτά τα μηνύματα μεταδίδονται συνήθως και μέσω των γραμμών ηλεκτρικής ενέργειας και μέσω ραδιοσυχνότητας με αποτέλεσμα να υπάρχει μεγαλύτερη αξιοπιστία. Όπως και άλλα συστήματα οικιακού αυτοματισμού, το Insteon σχετίζεται με το "Internet of Things”.

Τα προϊόντα Insteon εκδόθηκαν για πρώτη φορά το 2005 από την εταιρία Smartlabs. Μια θυγατρική εταιρία της Smartlabs, που ονομάζεται επίσης INSTEON, δημιουργήθηκε για την εμπορεία της τεχνολογίας. (Insteon, 2016)

### 2.1.4 Z-Wave

To Z-Wave είναι ένα ασύρματο πρωτόκολλο επικοινωνίας για την αυτοματοποίηση μιας κατοικίας. Έχει ως στόχο να παρέχει μια απλή και αξιόπιστη μέθοδο για τον ασύρματο έλεγχο:

* του φωτισμού
* της θέρμανσης
* του εξαερισμού
* του κλιματισμού
* της ασφαλείας της κατοικίας
* του home cinema
* των αυτόματων παραθύρων
* της πισίνας
* του spa
* του γκαράζ
* της πρόσβασης στο σπίτι

Υπάρχουν εκατοντάδες διαλειτουργικά προϊόντα Z-Wave τα οποία διατίθενται στο εμπόριο και πάνω από 35 εκατομμύρια έχουν πουληθεί από το 2005 και μετά. Το Z-Wave αναπτύχθηκε από μια startup εταιρία στη Δανία η οποία ονομάζεται Zen-Sys και αποκτήθηκε από την Sigma Designs το 2008.

Ένα σύστημα αυτοματισμού Z-Wave μπορεί να ελεγχθεί απομακρυσμένα μέσω του Διαδικτύου, χρησιμοποιώντας μια Z-Wave gateway ή μια συσκευή κεντρικού ελέγχου η οποία χρησιμεύει και ως Z-Wave hub αλλά και ως πύλη προς τον έξω κόσμο. (Z-Wave Home control, 2016)

### 2.1.5 ZigBee

Το ZigBee είναι ένα πρωτόκολλο επικοινωνίας υψηλού επιπέδου βασισμένο στο IEEE 802.15.4 και χρησιμοποιείται για τη δημιουργία προσωπικών δικτύων (PAN) με χαμηλής ισχύος ψηφιακά ραδιόφωνα.

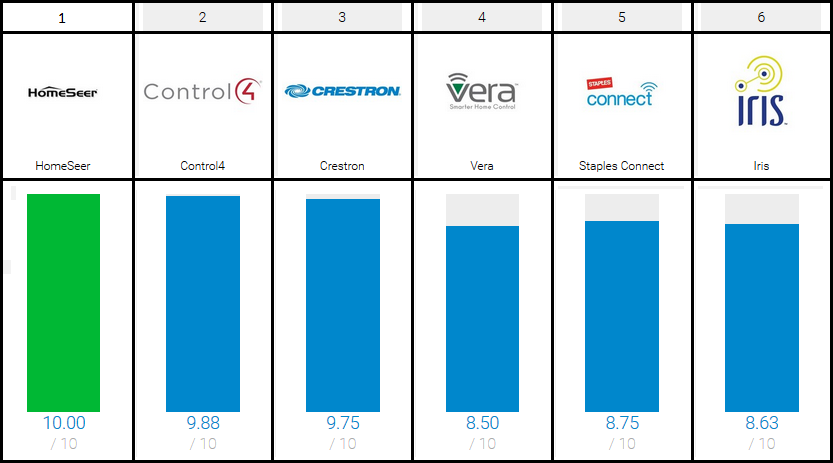
Η τεχνολογία που ορίζεται από το πρωτόκολλο ZigBee στοχεύει στο να είναι απλούστερη και λιγότερο δαπανηρή από ό, τι άλλα ασύρματα προσωπικά δίκτυα (WPANs), όπως Bluetooth ή Wi-Fi. Οι εφαρμογές περιλαμβάνουν ασύρματους διακόπτες φωτός, μετρητές ρεύματος με οθόνες ελέγχου, συστήματα διαχείρισης κυκλοφορίας και άλλο καταναλωτικό και βιομηχανικό εξοπλισμό που απαιτεί μικρής εμβέλειας χαμηλού ρυθμού ασύρματη μεταφορά δεδομένων.

Η χαμηλή κατανάλωση των συσκευών που χρησιμοποιούν ZigBee περιορίζει τις αποστάσεις μετάδοσης στα 10-100 μέτρα οπτικής επαφής, ανάλογα με την ισχύ εξόδου και περιβαλλοντικά χαρακτηριστικά. Οι συσκευές ZigBee μπορούν να μεταδώσουν δεδομένα σε μεγάλες αποστάσεις περνώντας τα δεδομένα μέσω ενός δικτύου πλέγματος αποτελούμενο από ενδιάμεσες συσκευές, με σκοπό να φτάσουν σε πιο μακρινές συσκευές. Το ZigBee χρησιμοποιείται συνήθως σε εφαρμογές χαμηλού ρυθμού δεδομένων που απαιτούν μεγάλη διάρκεια ζωής της μπαταρίας και ασφαλή δικτύωση.

Το ZigBee επινοήθηκε το 1998, τυποποιήθηκε το 2003 και αναθεωρήθηκε το 2006. Το όνομα αναφέρεται στο χορό των μελισσών μετά την επιστροφή τους στην κυψέλη. (Smarthome.com, 2016)

## 2.2 Εμπορικές Λύσεις

Ο παρακάτω πίνακας παρουσιάζει τις έξι καλύτερες εμπορικές λύσεις αυτοματισμού κατοικίας για το 2016 με βάση την λειτουργικότητα τους. Παρακάτω θα αναφερθούμε στις 3 καλύτερες. (TopTenREVIEWS, 2016)



Ε.1. Οι έξι καλύτερες λύσεις αυτοματισμού κατοικίας για το 2016 βάση λειτουργικότητας

### 2.2.1 Homeseer

Η HomeSeer Technologies LLC είναι μια εταιρεία τεχνολογίας που βασίζεται στο Μπέντφορντ, Νιού Χάμσαιρ, στις Ηνωμένες Πολιτείες. Η HomeSeer κατασκευάζει λογισμικό και ελεγκτές οικιακού αυτοματισμού. Σχεδιάστηκε έτσι ώστε να ενσωματώνει συσκευές φωτισμού, θέρμανσης, ασφαλείας και άλλα υποσυστήματα. Επιπλέον, η εταιρεία διατηρεί ένα ηλεκτρονικό κατάστημα συμβατών προϊόντων στο σύστημα οικιακού αυτοματισμού της και ένα φόρουμ για χρήστες και εμπόρους συσκευών.

Το λογισμικό είναι στον πυρήνα του κάθε συστήματος που παράγεται από την HomeSeer. Είναι αδειοδοτημένο ως αυτόνομο προϊόν (HS3) για τους χρήστες που επιθυμούν να το τρέξουν στον δικό τους προσωπικό υπολογιστή. Είναι επίσης προ-εγκατεστημένο σε HomeTroller ελεγκτές, για τους χρήστες που προτιμούν μια πιο πλήρως διαμορφωμένη λύση. Σε μια τυπική εγκατάσταση, τα συστήματα HS3 συνδέονται με το οικιακό δίκτυο των χρηστών ώστε να είναι δυνατός ο προγραμματισμός του με οποιοδήποτε φυλλομετρητή. Μόλις ο προγραμματισμός ολοκληρωθεί, το σύστημα HS3 είναι πια έτοιμο να αρχίσει διαδικασίες αυτοματισμού κατοικίας χωρίς επιπλέον πράξεις από την μεριά του χρήστη. Εάν είναι επιθυμητό, η εφαρμογή για κινητά HSTouch μπορεί να χρησιμοποιηθεί για την παρακολούθηση και τον έλεγχο των συσκευών στο σπίτι. Η υπηρεσία ιστού MyHS της HomeSeer έχει σχεδιαστεί για να παρέχει απομακρυσμένη πρόσβαση στις συσκευές και δεν απαιτείται για την επεξεργασία του συστήματος μέσα από το σπίτι. (HomeSeer Home Automation Systems, 2016)

Το σύστημα HomeSeer έχει σχεδιαστεί για να ενσωματώνει τις ακόλουθες τεχνολογίες και προϊόντα:

Συμβατές Τεχνολογίες

* Z-Wave
* INSTEON
* UPB
* X10
* PLCBUS

Συμβατά προϊόντα

* Διακόπτες φωτισμού
* θερμοστάτες
* κλειδαριές στις πόρτες
* ελεγκτές πόρτας γκαράζ
* βαλβίδες νερού
* συστήματα ασφαλείας
* δέκτες τηλεοράσεων, ήχου / βίντεο και συναφή προϊόντα
* IP κάμερες
* Αισθητήρες Περιβάλλοντος και Ασφαλείας

### 2.2.2 Control4

Τα συστήματα οικιακού αυτοματισμού Control4 έχουν παρομοιαστεί με ένα λειτουργικό σύστημα για το σπίτι. Η εταιρεία αναπτύσσει μια επώνυμη σειρά προϊόντων και υπηρεσιών για τον έλεγχο των συστημάτων φωτισμού στο σπίτι, ήχου και κλιματισμού που χρησιμοποιούν πρωτόκολλα επικοινωνίας, όπως ZigBee, Wi-Fi, Bluetooth, και Z-Wave. Το Control4 υποστηρίζει επίσης, περισσότερες από 8.000 συσκευές από τρίτους κατασκευαστές, συμπεριλαμβανομένου του θερμοστάτη της Nest για έξυπνα σπίτια, το σύστημα μουσικής Sonos και προϊόντα από τη Sony και τη Sub-Zero. Για παράδειγμα, με το Control4, ένας ιδιοκτήτης θα μπορούσε να "ελέγχει τον εξαερισμό στο γκαράζ του, να παίζει μουσική σε διάφορα δωμάτια του σπιτιού του, να ελέγχει κάθε φως, την τηλεόραση, το θερμοστάτη, ακόμη και τη συσκευή αναπαραγωγής Blu-Ray στον ξενώνα "από μια συσκευή iOS.

Το 2012, η Control4 κυκλοφόρησε το Simple Device Discovery Protocol (SDDP), το οποίο καθιστά τα προϊόντα με ενσωματωμένο κώδικα, να εντοπίζονται αυτόματα σε ένα δίκτυο Control4. Η εταιρεία αδειοδότησε το πρωτόκολλο, το οποίο είναι παρόμοιο με το Universal Plug and Play και για προϊόντα άλλων προμηθευτών.

Από τον Ιανουάριο του 2015, πάνω από εκατό προγραμματιστές υποστήριξαν το SDDP σε περισσότερες από 500 συσκευές, συμπεριλαμβανομένων των Panasonic, Dish Network, Bose, TiVo, Epson, Harman Kardon, Denon, Pioneer, Sony, Sharp, και η Yamaha.

Από τον Φεβρουάριο του 2015, το Control4 άρχισε να εμπορεύεται τα προϊόντα της μέσω ενός δικτύου από εξουσιοδοτημένους αντιπροσώπους σε 88 χώρες.

Επιπλέον, η Control4 συνεργάζεται με κατασκευαστές σπιτιών, συμπεριλαμβανομένης της Toll Brothers, για να παρέχει προαιρετικά πακέτα οικιακού αυτοματισμού σε αγοραστές σπιτιών. (Control4.com, 2016)

Προϊόντα:

* Ήχος / βίντεο - μια σειρά προϊόντων διανομής ήχου και βίντεο για τον έλεγχο της ψυχαγωγίας σε όλο το σπίτι, χρησιμοποιώντας μια κεντρική πηγή.
* Φωτισμός - ασύρματα, προγραμματιζόμενα συστήματα ελέγχου φωτισμού, που ελέγχονται από διακόπτες, ροοστάτες, πληκτρολόγια και απομακρυσμένες συσκευές.
* Θέρμανση και κλιματισμός - έξυπνοι θερμοστάτες, συμβατοί με τα περισσότερα συστήματα θέρμανσης και κλιματισμού σε όλο τον κόσμο, ελεγχόμενοι από απόσταση.
* Οθόνες αφής - Οθόνες αφής, χειρός και σταθερές, που λειτουργούν ως αποκλειστικές διεπαφές για συστήματα Control4
* Τηλεχειριστήρια - τηλεχειριστήρια που επιτρέπουν τον έλεγχο ενός ολόκληρου συστήματος έξυπνου σπίτιου.
* Ελεγκτές - κύρια συσκευή σύνδεσης και ελέγχου ολόκληρου του συστήματος.
* Αισθητήρες - αισθητήρες κίνησης και φωτός για να ενεργοποιούνται αυτόματα δράσεις στο πλαίσιο του συστήματος όπως το να ανάβει το φως όταν μπαίνουμε στο δωμάτιο
* Κάμερες - κάμερες ασφαλείας για ενδοεπικοινωνία.
* Δικτύωση - μετά την εξαγορά της εταιρίας Pakedge προσφέρονται και λειτουργίες δικτύωσης

### 2.2.3 Crestron

Η Crestron είναι μια κορυφαία εταιρεία του κλάδου, όσων αφορά την αυτοματοποίηση σε σχολεία, επιχειρήσεις και σπίτια. Αυτό το σύστημα οικιακού αυτοματισμού υποστηρίζει ένα από τα μεγαλύτερα εμπορικά δίκτυα στην αγορά. Η Creston προσφέρει συμβατές συσκευές με δυναμικά χαρακτηριστικά και συμβατότητα όσων αφορά το λογισμικό. Έχει ένα παγκόσμιο δίκτυο με επαγγελματίες - εγκαταστάτες.

Η Crestron προσφέρει ένα κομψό σύστημα με δυναμική λειτουργικότητα. Μόλις εγκατασταθεί το σύστημα, είναι εύκολο στη χρήση. Αν θέλετε απομακρυσμένη πρόσβαση στο έξυπνο σύστημα του σπιτιού σας, μπορείτε να χρησιμοποιήσετε μία από τις εφαρμογές για κινητά Crestron για να ανάψετε τα φώτα, να ανοίξετε τα παντζούρια και να ανάψετε το τζάκι. Στο κατάλληλο πλαίσιο και με τον κατάλληλο εξοπλισμό, η Crestron μπορεί να βοηθήσει μετατρέποντας το σπίτι σας σε μια αποδοτική μηχανή που έχει τη δυνατότητα όχι μόνο να κάνει το σπίτι σας πιο έξυπνο, αλλά και να εξοικονομήσει χρήματα για την ηλεκτρική ενέργεια και άλλες υπηρεσίες κοινής ωφέλειας.

Μπορείτε επίσης να ρυθμίσετε το σύστημα να συνεργαστεί με μια ποικιλία από προκαθορισμένα προγράμματα. Τα προγράμματα αυτά, που ονομάζονται triggers, μπορούν να εκτελέσουν βασικές εργασίες, όπως η ενεργοποίηση της καφετιέρας. Ωστόσο, μπορούν επίσης να εκτελούν περίπλοκες εργασίες όπως να ανάβουν τα φώτα σε διάφορα μέρη του σπιτιού σας σε τυχαία χρονικά διαστήματα για να αποτρέψει κλέφτες, ενώ είστε σε διακοπές. Ένα ωραίο χαρακτηριστικό αυτών των προγραμμάτων είναι ότι μπορείτε να τα ενεργοποιήσετε αυτόματα ή χειροκίνητα. Η Crestron υποστηρίζει επίσης φωνητικό έλεγχο για όταν δεν μπορείτε να βρείτε το τηλεχειριστήριο ή όταν είστε μακριά από μια κονσόλα ελέγχου. Λίγα συστήματα αυτοματισμού κατοικίας προσφέρουν φωνητικό έλεγχο. (Crestron.com, 2016)

# 3. Υπηρεσίες Ιστού

Σε αυτό το κεφάλαιο θα αναφερθούμε στις διαδικτυακές υπηρεσίες μιας και είναι η βάση ενός προγράμματος έξυπνης κατοικίας. Θα παρουσιάσουμε τις υπηρεσίες ιστού τύπου REST, καθώς και τα πλεονεκτήματα τους. Θα επεξηγήσουμε επίσης την διαδικασία που ακολουθήσαμε για την υλοποίηση του προγράμματος. Τέλος θα αναλύσουμε τον κώδικα του προγράμματος.

Όσων αφορά τον διακομιστή του συστήματος, χρησιμοποιήθηκε υπολογιστής που φιλοξενείται στο okeanos.grnet.gr. Το περιβάλλον ανάπτυξης λογισμικού που χρησιμοποιήθηκε για την ανάπτυξη της υπηρεσίας ιστού είναι το Eclipse IDE for Java EE Developers.

## 3.1 Βασική Γνώση

Οι υπηρεσίες ιστού είναι εφαρμογές πελάτη και διακομιστή που επικοινωνούν μέσω του πρωτόκολλου μεταφοράς υπερκειμένου (http) του World Wide Web (WWW) . Όπως περιγράφεται από την World Wide Web Consortium (W3C), οι υπηρεσίες ιστού παρέχουν ένα πρότυπο μέσο διαλειτουργικότητας μεταξύ των εφαρμογών λογισμικού, οι οποίες τρέχουν σε μια ποικιλία από πλατφόρμες και πλαίσια. Οι υπηρεσίες ιστού χαρακτηρίζονται από την μεγάλη επεκτασιμότητα τους.

Μια Υπηρεσία Ίστού προσφέρεται από μια ηλεκτρονική συσκευή σε μια άλλη ηλεκτρονική συσκευή, επικοινωνώντας μεταξύ τους μέσω του Παγκοσμίου Ιστου. Σε μια τέτοια υπηρεσία, μια τεχνολογία Ιστού (όπως το HTTP, το οποίο αρχικά είχε σχεδιαστεί για την επικοινωνία ανθρώπου-υπολογιστή) χρησιμοποιείται για την επικοινωνία μεταξύ υπολογιστών, ειδικότερα για τη μεταφορά μορφών αρχείων αναγνώσιμα από υπολογιστές όπως XML και JSON.

Στην πράξη, η υπηρεσία Ιστού παρέχει μια αντικειμενοστρεφή, διαδικτυακή διεπαφή σε ένα διακομιστή βάσης δεδομένων. Η υπηρεσία αυτή μπορεί να χρησιμοποιηθεί για παράδειγμα από ένα άλλο διακομιστή Ιστού ή από μια κινητή εφαρμογή που παρέχει μια διεπαφή για τον τελικό χρήστη. Μια άλλη χρήση τέτοιας υπηρεσίας μπορεί να είναι ένας συνδιασμός των δυο, όπου ένας διακομιστής Ιστού καταναλώνει διάφορες υπηρεσίες Ιστού από διαφορετικούς υπολογιστές, και συγκεντρώνει το περιεχόμενο τους σε ένα περιβάλλον εργασίας χρήστη. (Wikipedia, 2016)

Οι υπηρεσίες Ιστού είναι υπηρεσίες οι οποίες δεν εξαρτώνται από φυλλομετρητές και λειτουργικά συστήματα, το οποίο σημαίνει ότι μπορεί να τρέξουν σε οποιοδήποτε φυλλομετρητή χωρίς να χρειαστεί να γίνουν οποιεσδήποτε αλλαγές. Βασίζονται σε τεχνολογίες όπως HTTP, XML, SOAP, WSDL, SPARQL, κ.α. Τα σημαντικότερα πλεονεκτήματα που πηγάζουν απο την χρήση μιας υπηρεσίας Ιστού απο τον προγραμματιστή είναι τα εξής:

* Ευχρηστία: Οι υπηρεσίες Ιστού επιτρέπουν να εκτεθεί μέσω του Διαδικτύου η λογική πολλών διαφορετικών συστημάτων. Αυτό δίνει στις εφαρμογές την ελευθερία να επιλέξουν τις υπηρεσίες Ιστού που χρειάζονται. Έτσι, αντί να "ξανα-ανακαλύψουν τον τροχό" για κάθε πελάτη, το μόνο που χρειάζεται να κάνουν οι εφαρμογές αυτές είναι να συμπεριλάβουν επιπλέον λογική μόνο για την πλευρά του πελάτη. Είναι δηλαδή ένα είδος “online βιβλιοθήκης”.
* Διαλειτουργικότητα: Αυτό είναι το πιο σημαντικό όφελος των Υπηρεσιών Ιστού. Οι υπηρεσίες Ιστού τυπικά εργάζονται εκτός των ιδιωτικών δικτύων, προσφέροντας στους προγραμματιστές μια κοινόχρηστη διαδρομή στις προγραμματιστικές τους λύσεις. Επίσης τους δίνουν την δυνατότητα να χρησιμοποιήσουν τις γλώσσες προγραμματισμού που προτιμούν. Επιπλέον, χάρη στη χρήση των μεθόδων επικοινωνίας που βασίζονται σε πρότυπα, οι υπηρεσίες Ιστού είναι ουσιαστικά ανεξάρτητες από την πλατφόρμα προγραμματισμού που χρησιμοποιείται.
* Χαλαρά συνδεδεμένες: Κάθε υπηρεσία υπάρχει ανεξάρτητα από τις άλλες υπηρεσίες που απαρτίζουν την εφαρμογή. Μεμονωμένα κομμάτια της εφαρμογής μπορούν να τροποποιηθούν χωρίς να αλλάξουν τα υπόλοιπα κομμάτια του προγράμματος.
* Ευκολία στην Ενσωμάτωση: Τα δεδομένα είναι απομονωμένα μεταξύ των εφαρμογών. Οι υπηρεσίες Ιστού ενώνουν τα δεδομένα μεταξύ τους, επιτρέποντας ευκολότερη επικοινωνία εντός και μεταξύ των οργανισμών.
* Ευκολία στην Ανάπτυξη: Οι υπηρεσίες Web αναπτύσσονται πάνω σε καθιερωμένες τεχνολογίες του Διαδικτύου. Αυτό καθιστά δυνατή την ανάπτυξη υπηρεσιών Ιστού, πάνω από το τείχος προστασίας των υπολογιστών, σε διακομιστές στην άλλη πλευρά του πλανήτη. Επίσης, χάρη στη χρήση αποδεδειγμένων προτύπων, η ασφάλεια είναι ήδη ενσωματωμένη. (Mandliya, 2016)

## 3.2 Υπηρεσίες Ιστού REST

Representational State Transfer (REST) είναι ένα αρχιτεκτονικό στυλ που καθορίζει περιορισμούς, οι οποίοι αν εφαρμοστούν σε μια υπηρεσία ιστού έχουν ως αποτέλεσμα την καλύτερη απόδοση και δυνατότητα επέκτασης και τροποποιήσης, με αποτέλεσμα να λειτουργούν καλύτερα στο Διαδίκτυο.

Στην αρχιτεκτονική REST, τα δεδομένα και η λειτουργικότητα θεωρούνται πόροι και είναι προσβάσιμα χρησιμοποιώντας Uniform Resource Identifiers (URIs), δηλαδή υπερσυνδέσμους στο Διαδίκτυο. To αρχιτεκτονικό στύλ τύπου REST περιορίζει την αρχιτεκτονική σε έναν πελάτη / διακομιστή και έχει σχεδιαστεί για να χρησιμοποιεί πρωτόκολλο χωρίς επικοινωνία, συνήθως HTTP. Οι πελάτες και οι διακομιστές ανταλάσσουν αναπαραστάσεις πόρων χρησιμοποιώντας τυποποιημένη διεπαφή και πρωτόκολλο. Το περιεχόμενό των πόρων είναι προσβάσιμο από μια ποικιλία μορφών, όπως HTML, XML, απλό κείμενο, PDF, JPEG, JSON, κ.α. (Docs.oracle.com, 2016)

Οι ιδιότητες που επηρεάζονται από τους περιορισμούς του αρχιτεκτονικού στυλ τύπου REST είναι:

* Επίδοση - Οι αλληλεπιδράσεις μεταξύ τών στοιχείων της υπηρεσίας είναι ο κυρίαρχος παράγοντας της επίδοσης και της αποτελεσματικότητας του δικτύου.
* Επεκτασιμότητα - Είναι δυνατή η υποστήριξη μεγάλου αριθμού στοιχείων και αλληλεπιδράσεων μεταξύ αυτών των στοιχείων.
* Απλότητα των διασυνδέσεων
* Δυνατότητα τροποποιήσεως των στοιχείων για να ανταποκρίνεται στις μεταβαλλόμενες ανάγκες ακόμα και όταν η εφαρμογή εκτελείται. Προβολή της επικοινωνίας μεταξύ των εξαρτημάτων από πράκτορες της υπηρεσίας
* Φορητότητα των στοιχείων, μετακινώντας κώδικα προγράμματος μαζί με δεδομένα
* Αξιοπιστία - Ανθεκτικό σε αποτυχίες σε επίπεδο συστήματος, λόγω της παρουσίας επιπέδου αποτυχίας εκτέλεσης μέσα στα στοιχεία, στις υποδοχές, ή στα δεδομένα (Mandliya, 2016)

Ένα από τα βασικά χαρακτηριστικά μιας υπηρεσίας ιστού τύπου REST είναι η κλήση μεθόδων HTTP με τρόπο που ακολουθεί το πρωτόκολλο, όπως ορίζεται από το RFC 2616. Οι υπηρεσίες ιστού τύπου Rest ζητούν από τους προγραμματιστές να χρησιμοποιούν μεθόδους HTTP ρητά και με τρόπο που να είναι συνεπής με τον ορισμό του πρωτοκόλλου. Αυτή η βασική αρχή καθιερώνει μια χαρτογράφηση ένα προς ένα μεταξύ των διαδικασιών δημιουργίας, ανάκτησης, ανανέωσης και διαγραφής με τις αντίστοιχες μεθόδους HTTP. Σύμφωνα με αυτή τη χαρτογράφηση:

* Για να δημιουργήσετε έναν πόρο στο διακομιστή, χρησιμοποιήστε POST.
* Για να ανακτήσετε έναν πόρο, χρησιμοποιήστε GET.
* Για να αλλάξετε την κατάσταση ενός πόρου ή για να τον ενημερώσετε, χρησιμοποιήστε PUT.
* Για να αφαιρέσετε ή να διαγράψετε έναν πόρο, χρησιμοποιήστε DELETE.

Η αρχιτεκτονική τύπου REST ορίστηκε από τον Roy Thomas Fielding στην διδακτορική διατριβή του "Architectural Styles and the Design of Network-based Software Architectures".Ο Fielding ανέπτυξε την αρχιτεκτονική REST παράλληλα με το HTTP 1.1 του 1996-1999, με βάση το υπάρχον σχέδιο HTTP 1.0 του 1996.

## 3.3 Προγραμματισμός Διακομιστή

Για να προγραμματίσουμε οποιοδήποτε πρόγραμμα θα χρειαστούμε ένα περιβάλλον ανάπτυξης λογισμικού. Στην περίπτωση αυτού του διακομιστή επιλέχθηκε το Eclipse της Eclipse Foundation για έχει δυνατότητες να κατεβάζει αυτόματα τα εργαλεία και τα plugins που χρειαζόμαστε. Επίσης μας δίνεται η δυνατότητα να τρέξουμε τον διακομιστή τοπικά, για debugging, μιας και το eclipse μας δίνει την επιλογή να κατεβάσουμε αυτόματα διακομιστές όπως ο Apache Tomcat.

Για να προγραμματίσουμε μια υπηρεσία ιστού τύπου Rest θα χρειαστούμε ένα JAX-RS API. Mιας και η γλώσσα προγραμματισμού που θα χρησιμοποιήσουμε είναι η Java, θα χρειαστούμε το Jersey RESTful Web Services Framework. Το eclipse μας δίνει την επιλογή να το κατεβάσουμε αυτόματα.

Έπειτα θα πρέπει να σκεφτούμε την δομή του προγράμματος μας. Μια υπηρεσία ιστού REST αποτελείται από πόρους. Αρα κάθε διαφορετική συσκευή του έξυπνου σπιτιού θα πρέπει να αντιστοιχεί σε έναν πόρο στον διακομιστή. Επίσης κάθε αισθητήρας του έξυπνου σπιτιού θα είναι επίσης ένας πόρος στον διακομιστή.

Θα υλοποιήσουμε δύο βασικές συναρτήσεις HTTP για κάθε πόρο, την GET για να ανακτούμε την τιμή κάθε πόρου και την PUT για να ενημερώνουμε την τιμή κάθε πόρου. Οι συσκευές που θα χρησιμοποιήσουμε είναι οι εξής:

* Συναγερμος
* Δύο Φώτα Δωματίου
* Θερμοσίφωνας
* Κλιματιστικό
* Πατζούρι Παραθύρου

Οι αισθητήρες που θα χρησιμοποιήσουμε είναι οι εξής:

* Αισθητήρας Θερμοκρασίας
* Αισθητήρας Επιπέδου Φωτός
* Αισθητήρας Ζεστού Νερου στον Θερμοσίφωνα
* Αισθητήρας Παραβίασης Κατοικίας

Επίσης για κάθε συσκεή θα χρειαστούμε λειτουργίες ενεργοποίησης και απενεργοποίησης ανάλογα με την ώρα έτσι ώστε ο χρήστης να έχει και αυτή τη δυνατότητα. Ακόμα μια λειτουργία που θα συμπεριλάβουμε, θα είναι αυτή της ενεργοποίησης και απενεργοποίησης της συσκευής ανάλογα με την τιμή κάποιου αισθητήρα, με την προυπόθεση ότι κάθε συσκευή θα αντιστοιχεί σε έναν μόνο αισθητήρα. Έτσι ο χρήστης θα μπορεί για παράδειγμα να ενεργοποιήσει το κλιματιστικό αν η θερμοκρασία του δωματίου ξεπεράσει τους 30° C.

### 3.3.1 Υλοποίηση Κλάσεων Υλικού

Αρχικά πρέπει να υλοποιήσουμε δυο κλάσεις, μια για τις συσκευές και μία για τους αισθητήρες. Κάθε συσκευή και αισθητήρας θα είναι instances αυτών των δύο κλάσεων.

Η κλάση συσκευής υλοποιείται ώς εξής:

**public** **class** Device {

**private** String name;

**private** Boolean state;

**private** String enableTime;

**private** String disableTime;

**private** **int** enableSensorValue;

**private** **int** disableSensorValue;

Οι ιδιότητες της συσκεής είναι το όνομά της, η κατάσταση της, η ώρα ενέργοποίησης και απενεργοποίησης της καθώς και η τιμή ενεργοποίησης και απενεργοποίησης της βάση αισθητήρα.

Μέσα στον κατασκευαστή της κλάσης αρχικοποιούμε τις ιδιότητες της με null τιμές τις οποίες έχουμε ορίσει σε αρχείο σταθερών για καλύτερη διαφάνεια.

**public** Device(String name){

**this**.name = name;

**this**.state = Constant.***DEVICE\_OFF***;

**this**.enableTime = Constant.***NO\_TIME\_VALUE***;

**this**.disableTime = Constant.***NO\_TIME\_VALUE***;

**this**.enableSensorValue = Constant.***NO\_SENSOR\_VALUE***;

**this**.disableSensorValue = Constant.***NO\_SENSOR\_VALUE***;

}

Έπειτα ακολουθούν μέθοδοι ανάκτησης και ενημέρωσης της κάθε ιδιότητας. Επειδή είναι απλές “getters” και “setters”, δεν θα τις παραθέσω.

Η κλάση αισθητήρα είναι πιο απλή και υλοποιείται ως εξής:

**public** **class** Sensor {

**private** String name;

**private** **int** value;

Οι ιδιότητες του αισθητήρα είναι το όνομα του και η τιμή του.

Μέσα στον κατασκευαστή αρχικοποιούμε την τιμή του με την αντίστοιχη τιμή null.

**public** Sensor(String name){

**this**.name = name;

**this**.value = Constant.***SENSOR\_OFF***;

}

Έπειτα ακολουθούν μέθοδοι ανάκτησης και ενημέρωσης της κάθε ιδιότητας. Επειδή είναι απλές “getters” και “setters”, δεν θα τις παραθέσω.

### 3.3.2 Υλοποίηση Κλάσης Βάσης Δεδομένων

Η κλάση αυτή δεν είναι Βάση Δεδομένων με την κλασσική έννοια. Δεν συνδέεται με βάση δεδομένων για να ανακτήσει τις συσκευές. Απλά δημιουργεί ρητά τις συσκευές και τους αισθητήρες, μιας και ξέρουμε τον αριθμό του και την ιδιότητα του καθενός.

Η κλάση είναι η ακόλουθη:

**public** **class** DatabaseClass {

**private** **static** Map<Integer, Device> *devices* = **new** HashMap<>();

**private** **static** Map<Integer, Sensor> *sensors* = **new** HashMap<>();

Αρχικά δημιουργουμε δυο κενούς HashMaps οι οποίοι θα περιέχουν ο πρώτος τις συσκευές και ο δεύτερος τους αισθητήρες. Αντιστοιχούμε σε κάθε HashMap και έναν ακέραιο με κάθε συσκευή/αισθητήρα, ο οποίος ακέραιος είναι το αναγνωριστικό τους.

**static**{

*devices*.put(Constant.***DEVICE\_LIGHT\_1\_ID***, **new** Device("Light1"));

*devices*.put(Constant.***DEVICE\_LIGHT\_2\_ID***, **new** Device("Light2"));

*devices*.put(Constant.***DEVICE\_BOILER\_ID***, **new** Device("Boiler"));

*devices*.put(Constant.***DEVICE\_AIR\_CONDITION\_ID***,

**new** Device("AirCondition"));

*devices*.put(Constant.***DEVICE\_BLINDS\_ID***, **new** Device("Blinds"));

*devices*.put(Constant.***DEVICE\_ALARM\_ID***, **new** Device("Alarm"));

*sensors*.put(Constant.***SENSOR\_TEMPERATURE\_ID***,

**new** Sensor("Temperature"));

*sensors*.put(Constant.***SENSOR\_HOT\_WATER\_ID***,

**new** Sensor("Hot Water"));

*sensors*.put(Constant.***SENSOR\_LIGHT\_LEVEL\_ID***,

**new** Sensor("Light Level"));

*sensors*.put(Constant.***SENSOR\_BREAK\_IN\_ID***, **new** Sensor("Break In"));

}

Σε αυτό το σημείο δημιουργούμε στατικά την κάθε συσκευή και αισθητήρα και τις εισάγουμε στους HashMaps.

Τέλος ακολουθούν οι “Getters” και “Setters” για κάθε HashMap τους οποίους δεν παραθέτω λόγω απλότητας.

### 3.3.3 Υλοποίηση της Υπηρεσίας

Για την υλοποίηση της υπηρεσίας ιστού, θα χρειαστούμε δυο κλάσεις, μια κλάση που θα χειρίζεται τους πόρους και τα αιτήματα προς αυτούς και μια κλάση που θα εμπεριέχει την λογική την υπηρεσίας.

Η κλάση SmartHomeService είναι η κλάση που εμπεριέχει την λογική και ο κώδικας είναι ο εξής:

**public** **class** SmartHomeService {

**private** Map<Integer,Device> devices = DatabaseClass.*getDevices*();

**private** Map<Integer,Sensor> sensors = DatabaseClass.*getSensors*();

Calendar cal = Calendar.*getInstance*();

SimpleDateFormat sdf = **new** SimpleDateFormat("HH:mm");

Σε αυτό το σημείο εισάγονται όλες οι συσκευές απο την Βάση Δεδομένων μας. Επίσης για να μπορούμε να ελέγχουμε την ώρα κάθε φορά που θα τρέχει το πρόγραμμα, δημιουργούμε ένα instance του αντικειμένου Calendar και ένα instance του αντικειμένου SimpleDateFormat, για να αλλάξουμε την ώρα στη μορφή “HH:mm” όπως για παράδειγμα “23:34”.

Μέσα στον κατασκευαστή, καλούμε την συνάρτηση που μας επιστρέφει την ώρα και της αλλάζουμε μορφή.

**public** SmartHomeService(){

String currentTime = sdf.format(cal.getTime());

Διατρέχουμε τον πίνακα με τις συσκευές και αντιγράφουμε την συσκευή i στην μεταβλητή tempDevice για να την επεξεργαστούμε.

**for**(**int** i=1;i<7;i++){

Device tempDevice = devices.get(i);

Εάν η ώρα αυτή τη στιγμή είναι ίση με την ώρα που έχει βάλει ο χρήστης για να ανοίξει η συσκευή, ενεργοποίησε αυτη την συσκευή. Αντίστοιχα και με την απενεργοποίηση:

**if**(tempDevice.getEnableTime().equals(currentTime)){

tempDevice.setState(Constant.***DEVICE\_ON***);

}

**if**(tempDevice.getDisableTime().equals(currentTime)){

tempDevice.setState(Constant.***DEVICE\_OFF***);

}

Εάν υπάρχει τιμή ενεργοποίησης με βάση αισθητήρα στην συσκευή που αναλύουμε, επέλεξε το αναγνωριστικό της συσκευής.

**if**(tempDevice.getEnableSensorValue() != Constant.***NO\_SENSOR\_VALUE***){

**switch** (i){

Αν η συσκευή είναι συσκευή φωτισμού και το επίπεδο φωτός του αισθητήρα είναι μικρότερο η ίσο της τιμής του χρήστη, άναψε το φώς. Λειτουργία που ανάβει τα φώτα όταν νυχτώσει ή όταν συννεφιάσει ανάλογα με το ποσοστό φωτος που έχει δώσει ο χρήστης

**case** Constant.***DEVICE\_LIGHT\_1\_ID***:

**case** Constant.***DEVICE\_LIGHT\_2\_ID***:

**if**((sensors.get(Constant.***SENSOR\_LIGHT\_LEVEL\_ID***).getValue() <=

tempDevice.getEnableSensorValue())){

tempDevice.setState(Constant.***DEVICE\_ON***);

}

**break**;

Στην περίπτωση που η συσκευή είναι θερμοσίφωνας, αν η τιμή του ζεστού νερού του αισθητήρα είναι μικρότερη ή ίση από την τιμή που έχει δώσει ο χρήστης, άναψε τον θερμοσίφωνα. Λειτουργία που επιτρέπει τον χρήστη να ανάβει αυτόματα τον θερμοσίφωνα μόλις κρυώνει το νερό

**case** Constant.***DEVICE\_BOILER\_ID***:

**if**((sensors.get(Constant.***SENSOR\_HOT\_WATER\_ID***).getValue() <=

tempDevice.getEnableSensorValue())){

tempDevice.setState(Constant.***DEVICE\_ON***);

}

**break**;

Στην περίπτωση που η συσκευή είναι κλιματιστικό, αν η θερμοκρασία του αισθητήρα είναι μεγαλύτερη ή ίση από την θερμοκρασία που έχει βάλει ο χρήστης, άναψε το κλιματιστικό. Λειτουργία που επιτρέπει στον χρήστη να ανάβει αυτόματα το κλιματιτικό αν έχει πολύ ζέστη

**case** Constant.***DEVICE\_AIR\_CONDITION\_ID***:

**if**((sensors.get(Constant.***SENSOR\_TEMPERATURE\_ID***).getValue() >=

tempDevice.getEnableSensorValue())){

tempDevice.setState(Constant.***DEVICE\_ON***);

}

**break**;

Στην περίπτωση που η συσκευή είναι πατζούρι παραθύρου, αν το επίπεδο του φωτός του αισθητήρα είναι μεγαλύτερο ή ισο από το επίπεδο φωτός που έχει εισάγει ο χρήστης, άνοιξε τα παντζούρια. Λειτουργία που δίνει την δυνατότητα στον χρήστη να ανοίγει αυτόματα τα παντζούρια αν έχει ξημερώσει.

**case** Constant.***DEVICE\_BLINDS\_ID***:

**if**((sensors.get(Constant.***SENSOR\_LIGHT\_LEVEL\_ID***).getValue() >=

tempDevice.getEnableSensorValue())){

tempDevice.setState(Constant.***DEVICE\_ON***);

}

**break**;

}

}

Εδώ τελειώνει ο έλεγχος για ενεργοποίηση της συσκευής βάση τον αισθητήρα. Για την απενεργοποίηση της συσκευής βάση αισθητήρα, ο κώδικας είναι αντίστοιχος και η υλοποίηση, χωρίς σχόλια γιατί είναι προφανής, είναι η εξής:

**if**(tempDevice.getDisableSensorValue() != Constant.***NO\_SENSOR\_VALUE***){

**switch** (i){

**case** Constant.***DEVICE\_LIGHT\_1\_ID***:

**case** Constant.***DEVICE\_LIGHT\_2\_ID***:

**if**((sensors.get(Constant.***SENSOR\_LIGHT\_LEVEL\_ID***)

.getValue() >= tempDevice.getDisableSensorValue())){

tempDevice.setState(Constant.***DEVICE\_OFF***);

}

**break**;

**case** Constant.***DEVICE\_BOILER\_ID***:

**if**((sensors.get(Constant.***SENSOR\_HOT\_WATER\_ID***)

.getValue() >= tempDevice.getDisableSensorValue())){

tempDevice.setState(Constant.***DEVICE\_OFF***);

}

**break**;

**case** Constant.***DEVICE\_AIR\_CONDITION\_ID***:

**if**((sensors.get(Constant.***SENSOR\_TEMPERATURE\_ID***)

.getValue() <= tempDevice.getDisableSensorValue())){

tempDevice.setState(Constant.***DEVICE\_OFF***);

}

**break**;

**case** Constant.***DEVICE\_BLINDS\_ID***:

**if**((sensors.get(Constant.***SENSOR\_LIGHT\_LEVEL\_ID***)

.getValue() <= tempDevice.getDisableSensorValue())){

tempDevice.setState(Constant.***DEVICE\_OFF***);

}

**break**;

}

}

Τέλος εισάγουμε ξανά την συσκευή στην HashMap με τις συσκευές, στο ίδιο σημείο με πριν. Εδώ τελειώνει ο κατασκευαστής.

devices.put(i, tempDevice);

}

Οι επόμενες μέθοδοι θα καλούνται από την κλάση που θα διαχειρίζεται τους πόρους της υπηρεσίας με σκοπό να ανακτεί ή να ανανεώνει τις συσκευές στην μνήμη.

Ανακτά την συσκευή με το αναγνωριστικό που δόθηκε

**public** Device getDevice(**int** id){

**return** devices.get(id);

}

Ανανεώνει την κατάσταση μιας συσκευής

**public** **void** updateDevice(**int** id, Boolean state){

Device tempDevice = devices.get(id);

tempDevice.setState(state);

devices.put(id, tempDevice);

}

Ανακτά τον αισθητήρα με το αναγνωριστικό που δόθηκε

**public** Sensor getSensor(**int** id){

**return** sensors.get(id);

}

Ανανεώνει την τιμή ενός αισθητήρα

**public** **void** updateSensor(**int** id, **int** value){

Sensor tempSensor = sensors.get(id);

tempSensor.setValue(value); sensors.put(id,tempSensor);

}

Ανανεώνει την ώρα ενεργοποίησης μιας συσκευής

**public** **void** setDeviceEnableTime(**int** id, String time){

Device tempDevice = devices.get(id);

tempDevice.setEnableTime(time);

devices.put(id, tempDevice);

}

Ανανεώνει την ώρα απενεργοποίησης μιας συσκευής

**public** **void** setDeviceDisableTime(**int** id, String time){

Device tempDevice = devices.get(id);

tempDevice.setDisableTime(time);

devices.put(id, tempDevice);

}

Ανανεώνει την τιμή αισθητήρα για την ενεργοποίηση μιας συσκευή

**public** **void** setDeviceEnableSensorValue(**int** id, **int** value){

Device tempDevice = devices.get(id);

tempDevice.setEnableSensorValue(value);

devices.put(id, tempDevice);

}

Ανανεώνει την τιμή αισθητήρα για την απενεργοποίηση μιας συσκευής.

**public** **void** setDeviceDisableSensorValue(**int** id, **int** value){

Device tempDevice = devices.get(id);

tempDevice.setDisableSensorValue(value);

devices.put(id, tempDevice);

}

Η επόμενη και τελευταία κλάση της υπηρεσίας Ιστού που θα παρουσιαστεί είναι η κλάση που χειρίζεται τους πόρους και τις αιτήσεις προς αυτούς. Είναι η κλάση που χρησιμοποιεί το jersey και την αρχιτεκτονική τύπου REST. Η κλάση είναι η εξής:

@Path("/resources")

**public** **class** SmartHomeResource {

SmartHomeService shService = **new** SmartHomeService();

Κάθε φορά που κάποιος πελάτης θα καλεί την υπηρεσία με URI που αρχίζει με “/resources”, η κλάση αυτή θα εκτελείται. Γι’ αυτό το λόγο δημιουργούμε ένα instance της κλάσης SmartHomeService, έτσι ώστε να τρέχει ο κατασκευαστής της και να ελέγχει για τα πράγματα που προαναφέραμε (ώρα, τιμή αισθητήρα).

Εάν ο πελάτης ζητήσει HTTP GET, με URI που υποδεικνύεται στο @Path, για παράδειγμα “/resources/device/1”, μετέτρεψε την κατάσταση του αισθητήρα σε αλφαριθμητικό από λογική τιμή και επέστρεψε την σε μορφή απλού κειμένου (TEXT\_PLAIN).

@GET

@Path("device/{deviceId}")

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String getDeviceState(@PathParam("deviceId") **int** id) {

System.***out***.println("GET device/" + id);

Device tempDevice = shService.getDevice(id);

**return** Boolean.*toString*(tempDevice.getState());

}

Εάν ο πελάτης ζητήσει HTTP PUT, με URI που υποδεικνύεται στο @Path, για παράδειγμα “/resources/device/1”, πέρνα στην μεταβλητή state το αλφαριθμητικό που θα δεχτείς (μετατρέποντας το σε λογική τιμή), ανανέωσε την συσκευή με αναγνωριστικό “deviceId” ώστε να λάβει την καινούρια τιμή της και επέστρεψε το αλφαριθμητικό “Success” σε μορφή απλού κειμένου (TEXT\_PLAIN).

@PUT

@Path("/device/{deviceId}")

@Consumes(MediaType.***TEXT\_PLAIN***)

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String setDeviceState(@PathParam("deviceId")**int** id,String state){

System.***out***.println("PUT device/" + id + " State: " + state);

shService.updateDevice(id, Boolean.*valueOf*(state));

**return** "Success";

}

Αντίστοιχα και με τους αισθητήρες. Το μόνο που αλλάζει είναι το URI και η μετατροπή του αλφαριθμητικού σε ακέραιο αντί για λογική τιμή.

@GET

@Path("/sensor/{sensorId}")

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String getSensorState(@PathParam("sensorId") **int** id) {

System.***out***.println("GET sensor/" + id);

Sensor tempSensor = shService.getSensor(id);

**return** Integer.*toString*(tempSensor.getValue());

}

@PUT

@Path("sensor/{sensorId}")

@Consumes(MediaType.***TEXT\_PLAIN***)

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String setSensorValue(@PathParam("sensorId")**int** id,String value){

System.***out***.println("PUT sensor/" + id + " Value: " + value);

shService.updateSensor(id, Integer.*parseInt*(value));

**return** "Success";

}

Αντίστοιχα για την ενεργοποίηση/απενεργοποίηση της συσκευής με βάση την ώρα. Το μόνο που αλλάζει είναι το URI και η συνάρτηση που καλείται.

@GET

@Path("device/{deviceId}/enable/time")

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String getDeviceEnableTimeValue(@PathParam("deviceId") **int** id) {

System.***out***.println("GET device/" + id + "/enable/time");

Device tempDevice = shService.getDevice(id);

**return** tempDevice.getEnableTime();

}

@PUT

@Path("/device/{deviceId}/enable/time")

@Consumes(MediaType.***TEXT\_PLAIN***)

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String setDeviceEnableTime(@PathParam("deviceId") **int** id, String

time) {

System.***out***.println("PUT device/" + id + " Enable: Time: "+ time);

shService.setDeviceEnableTime(id, time);

**return** "Success";

}

@GET

@Path("device/{deviceId}/disable/time")

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String getDeviceDisableTime(@PathParam("deviceId") **int** id) {

System.***out***.println("GET device/" + id + "/disable/time");

Device tempDevice = shService.getDevice(id);

**return** tempDevice.getDisableTime();

}

@PUT

@Path("/device/{deviceId}/disable/time")

@Consumes(MediaType.***TEXT\_PLAIN***)

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String setDeviceDisableTime(@PathParam("deviceId") **int** id,

String time) {

System.***out***.println("PUT device/" + id + " Disable: Time: "+time);

shService.setDeviceDisableTime(id, time);

**return** "Success";

}

Αντίστοιχα για την ενεργοποίηση/απενεργοποίηση της συσκευής με βάση κάποιον αισθητήρα. Το μόνο που αλλάζει είναι το URI, η συνάρτηση που καλείται και η μετατροπή της τιμής.

@GET

@Path("device/{deviceId}/enable/sensorValue")

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String getDeviceEnableSensorValue(@PathParam("deviceId") **int** id){

System.***out***.println("GET device/" + id + "/enable/sensorValue");

Device tempDevice = shService.getDevice(id);

**return** Integer.*toString*(tempDevice.getEnableSensorValue());

}

@PUT

@Path("/device/{deviceId}/enable/sensorValue")

@Consumes(MediaType.***TEXT\_PLAIN***)

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String setDeviceEnableSensor(@PathParam("deviceId") **int** id,

String value) {

System.***out***.println("PUT device/" + id + " Enable: Sensor Value: "

+ value);

shService.setDeviceEnableSensorValue(id,Integer.*parseInt*(value));

**return** "Success";

}

@GET

@Path("device/{deviceId}/disable/sensorValue")

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String getDeviceDisableSensorValue(@PathParam("deviceId")**int** id){

System.***out***.println("GET device/" + id + "/disable/sensorValue");

Device tempDevice = shService.getDevice(id);

**return** Integer.*toString*(tempDevice.getDisableSensorValue());

}

@PUT

@Path("/device/{deviceId}/disable/sensorValue")

@Consumes(MediaType.***TEXT\_PLAIN***)

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String setDeviceDisableSensor(@PathParam("deviceId") **int** id,

String value) {

System.***out***.println("PUT device/" + id +" Disable: Sensor Value: "

+ value);

shService.setDeviceDisableSensorValue(id,Integer.*parseInt*(value));

**return** "Success";

}

Η υπηρεσία ιστού μας έχει ολοκληρωθεί. Χειρίζεται κάθε κλήση ανάκτησης και ανανέωσης από πελάτες υλοποιημένοι σε οποιαδήποτε γλώσσα προγραμματισμού διότι είναι αρχιτεκτονικού τύπου REST, μιας και επικοινωνεί με τις συναρτήσεις GET και PUT της HTTP. Κατέχει δυνατότητες ανάκτησης και ανανέωσης συσκευών και αισθητήρων. Επίσης υπάρχει η δυνατότητα ανάκτησης και ανανέωσης την ώρας χρονοπρογραμματισμού των συσκευών. Τέλος είναι δυνατό από τους πελάτες να ανακτήσουν και να ανανεώσουν τιμές αισθητήρα για τους οποίους η συσκευή ενεργοποιείται και απενεργοποιείται.

Τέλος παραθέτων την κλάση σταθερών της υπηρεσίας, όπου αναγράφονται οι τιμές όλων των σταθερών που χρησιμοποιεί ο διακομιστής. Είναι αδύνατο να δημιουργηθεί instance αυτής της κλάσης οπουδήποτε μέσα στο πρόγραμμα.

**public** **final** **class** Constant {

**public** **static** **final** **int** ***NO\_SENSOR\_VALUE*** = -1;

**public** **static** **final** String ***NO\_TIME\_VALUE*** = "";

**public** **static** **final** **int** ***SENSOR\_OFF*** = 0;

**public** **static** **final** Boolean ***DEVICE\_OFF*** = **false**;

**public** **static** **final** Boolean ***DEVICE\_ON*** = **true**;

**public** **static** **final** **int** ***SENSOR\_OFFSET*** = 5;

**public** **static** **final** **int** ***DEVICE\_LIGHT\_1\_ID*** = 1;

**public** **static** **final** **int** ***DEVICE\_LIGHT\_2\_ID*** = 2;

**public** **static** **final** **int** ***DEVICE\_BOILER\_ID*** = 3;

**public** **static** **final** **int** ***DEVICE\_AIR\_CONDITION\_ID*** = 4;

**public** **static** **final** **int** ***DEVICE\_BLINDS\_ID*** = 5;

**public** **static** **final** **int** ***DEVICE\_ALARM\_ID*** = 6;

**public** **static** **final** **int** ***SENSOR\_TEMPERATURE\_ID*** = 1;

**public** **static** **final** **int** ***SENSOR\_HOT\_WATER\_ID*** = 2;

**public** **static** **final** **int** ***SENSOR\_LIGHT\_LEVEL\_ID*** = 3;

**public** **static** **final** **int** ***SENSOR\_BREAK\_IN\_ID*** = 4;

**private** Constant(){

**throw** **new** AssertionError();

}

}

# 4. Arduino

Το Arduino είναι μια πλατφόρμα ανοικτού κώδικα, βασισμένη σε εύκολο στη χρήση υλικό και λογισμικό. Η πλακέτα Arduino είναι ικανή να διαβάσει από μια είσοδο (φως από έναν αισθητήρα, το πάτημα ενος κουμπιού ή ένα μήνυμα στο Twitter) και να το μετατρέψει σε μια έξοδο (ενεργοποίώντας ενα κινητήρα, ανάβοντας ένα LED, δημοσιεύοντας κάτι στο διαδίκτυο). Μπορείτε να πείτε στην πλακέτα σας τι να κάνει, στέλνοντας ενα σύνολο από εντολές στον μικροελεγκτή που βρίσκεται πάνω της. Αυτό γίνεται με την χρήση της γλώσσας προγραμματισμού Arduino (βασισμένη στην Wiring), και το λογισμικό Arduino (IDE), βασισμένο στο Processing.

Τα τελευταία χρόνια το Arduino χρησιμοποιείται από χιλιάδες projects, ελέγχοντας από καθημερινά αντικείμενα μέχρι και πολύπλοκα επιστημονικά όργανα. Μια παγκόσμια κοινότητα δημιουργών - φοιτητές, καλλιτέχνες, προγραμματιστές και επαγγελματίες - έχει συγκεντρωθεί γύρω από αυτή την πλατφόρμα ανοικτού κώδικα, και η συμβολή τους είχε αποτέλεσμα την δημιουργία ενός τεράστιου όγκου προσβάσιμης γνώσης, η οποία βοηθάει αρχάριους όσο και τους εμπειρογνώμονες πάνω στο θέμα.

Το Arduino γεννήθηκε στο Ivrea Interaction Design Institute ως ένα εύκολο εργαλείο για τη γρήγορη δημιουργία πρωτοτύπων, απευθυνόμενο σε μαθητές χωρίς υπόβαθρο στα ηλεκτρονικά και στον προγραμματισμό. Από τη στιγμή που έφτασε στην ευρύτερη κοινότητα, η πλακέτα Arduino άρχισε να αλλάζει για να προσαρμοστεί στις νέες ανάγκες και προκλήσεις, και άρχισε να προσφέρει διάφορα προιόντα, από απλές πλακέτες 8-bit σε προϊόντα για εφαρμογές IoT, wearables, 3D printingη, και ενσωματωμένα συστήματα. Όλες οι πλακέτες Arduino πλακέτες είναι ανοικτού λογισμικού, ωθώντας τους χρήστες να κατασκευάσουν τα συστήματα τους ως ανεξάρτητα, και έπειτα να τα προσαρμόσουν στις ιδιαίτερες ανάγκες τους. Το λογισμικό είναι επίσης ανοιχτό, και αναπτύσσεται μέσα από τις συνεισφορές των χρηστών από όλο τον κόσμο. (Arduino.cc, 2016)

## 4.1 Πλακέτα Arduino Uno

Για την προσομοίωση των συσκευών της κατοικίας, όσων αφορά το υλικό κομμάτι χρησιμοποιήθηκε η πλακέτα Arduino Uno, ένα Ethernet Shield, καθώς και μερικά εξαρτήματα απο το επίσημο Arduino Starter Kit. (Wikipedia, 2016)

"Uno" σημαίνει "ένα" στα ιταλικά και επιλέχτηκε για να σηματοδοτήσει την έκδοση του Arduino Software (IDE) 1.0. Η πλακέτα Uno και η έκδοση 1.0 του Arduino Software ήταν εκδόσεις αναφοράς του Arduino, το οποίο πλέον έχει εξελιχθεί σε νεότερες εκδόσεις. Η πλακέτα Uno είναι η πρώτη από μια σειρά από πλακέτες Arduino με USB, και το μοντέλο αναφοράς για την πλατφόρμα Arduino.

Το Arduino Ethernet Shield επιτρέπει σε μια πλακέτα Arduino να συνδεθεί στο διαδίκτυο. Βασίζεται πάνω στο Wiznet W5100 ethernet chip. Το Wiznet W5100 παρέχει ένα IP και είναι ικανό να συνδεθεί με προτόκολλα TCP και UDP. Υποστηρίζει έως και τέσσερις ταυτόχρονες υποδοχές σύνδεσης. Η ethernet shield συνδέεται με μια πλακέτα Arduino χρησιμοποιώντας pins που προεξέχουν από την ασπίδα. Αυτό επιτρέπει την προσθήκη και άλλων shields. Υπάρχει μια ενσωματωμένη υποδοχή κάρτας micro-SD, το οποίο μπορεί να χρησιμοποιηθεί για να αποθηκεύσει αρχεία για λειτουργίες που χρησιμοποιούν το διαδίκτυο. (Arduino.cc, 2016)

Ε.2. Arduino Uno Rev3 Ε.3. Arduino Ethernet Shield

|  |  |
| --- | --- |
| Μικροελεγκτής | ATmega328 |
| Operating Voltage | 5V |
| Τάση λειτουργείας (προτεινόμενη) | 7-12V |
| Τάση λειτουργείας (όρια) | 6-20V |
| Ψηφιακοί ακροδέκτες (εισόδου – εξόδου) | 14 (οι 6 επιτρέπουν έξοδο PWM) |
| Αναλογικοί ακροδέκτες (εισόδου ) | 6 |
| Συνεχές ρεύμα ανά ακροδέκτη (εισόδου - εξόδου) | 40 mA |
| Συνεχές ρεύμα για ακροδέκτη τάσης 3.3V Pin | 50 mA |
| Μνήμη Flash | 32 KB (ATmega328) από το οποίο 0,5ΚΒ χρησιμοποιείται για φόρτωση λογισμικού |
| SRAM | 2 KB (ATmega328) |
| EEPROM | 1 KB (ATmega328) |
| Ταχύτητα ρολογιού | 16 MHz |

Π.1. Χαρακτηριστικά Arduino Uno

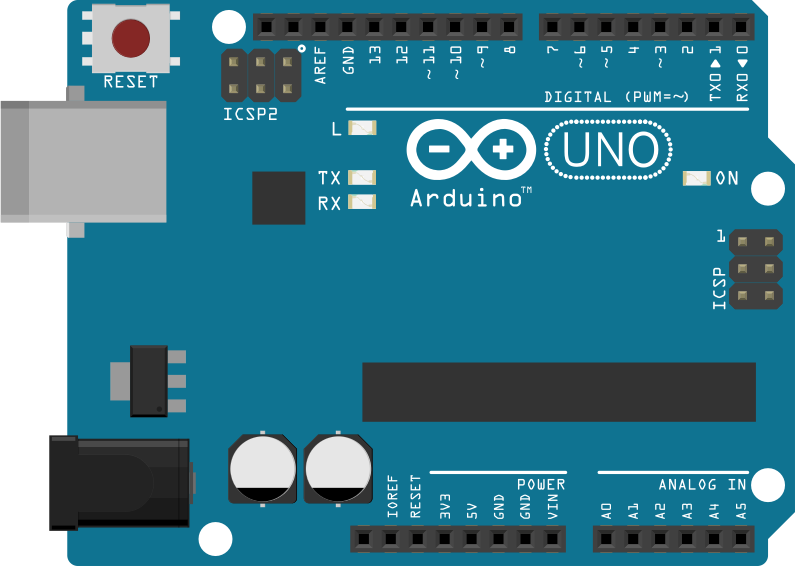
Συνολικά υπάρχουν 32 ακροδέκτες, στην πάνω πλευρά του Arduino χωρίζονται σε 3 κατηγορίες:

Ψηφιακοί (Digital): Οι πρώτοι 14 (0-13 pins) ακροδέκτες από δεξιά χρησιμοποιούνται από συσκευές που μπορούν να στείλουν και να παραλάβουν ψηφιακό σήμα.

Όσοι από αυτούς έχουν τον χαρακτήρα (~) σημειωμένο, είναι έξοδοι με δυνατότητα PWM. Pulse Width Modulation, ή PWM, είναι μια τεχνική που εξάγει αναλογικό αποτελέσματα με ψηφιακά μέσα. Δημιουργεί ένα τετραγωνικό κύμα, ένα σήμα που εναλάσσεται μεταξύ ανοικτου και κλειστού. Για να επιτύχουμε διαφορετικές αναλογικές τιμές, αλλάζουμε το πλάτος του παλμού αυτού.

Το GΝD, είναι γείωση και ανήκει στην κατηγορία της τροφοδοσίας, και το AREF χρησιμεύει για αναφορά στο ρεύμα που δέχεται το κύκλωμα από τις αναλογικές εισόδους.

Αναλογικοί (Analog In): Είναι οι ακροδέκτες A0, A1, A2, A3, A4, A5, είναι δίαυλοι εισόδου/εξόδου για αναλογικές συσκευές, και μπορούν να κάνουν μετατροπές από Αναλογικό σε Ψηφιακό σήμα μέχρι 10bit.

Τροφοδοσία (Power): Οι ακροδέκτες GND είναι η γείωση του κυκλώματος. Ο ακροδέκτης RESET κάνει επαναφορά της αρχικής κατάστασης του Arduino. Το Vin επιτρέπει την τροφοδοσία τάσης της πλακέτας από εξωτερική πηγή ενέργειας. Οι ακροδέκτες 5V και 3.3V είναι η τάση που θα τροφοδοτήσει η πλακέτα τα εξαρτήματα που συνδέονται. Τέλος το IOREF είναι αναφορά για την τάση που θα τροφοδοτηθεί σε κάποιο εξάρτημα μέσω ακροδέκτη.

Οι ψηφιακοί ακροδέκτες 0 και 1 χρησιμοποιούνται για να επικοινωνεί ο υπολογιστής με το Arduino, οπότε δεν μπορούμε να να χρησιμοποιήσουμε. Λόγω της χρήσης Ethernet Shield, οι ψηφιακοί ακροδέκτες 10 εως 13 χρησιμοποιούνται για την επικοινωνία του Arduino με την shield οπότε δεν μπορούμε να χρησιμοποιήσουμε ούτε αυτούς. Ο ακροδέκτης 4 χρησιμοποιείται για επικοινωνία με την κάρτα SD οπότε δεν τον χρησιμοποιούμε. Λόγω της έλλειψης αρκετών ψηφιακών ακροδεκτών, χρησιμοποιούμε τους αναλογικούς ακροδέκτες ως ψηφιακές εξόδους. (Arduino.cc, 2016)

Ε.4. Κάτοψη Arduino Uno

## 4.2 Σχεδιασμός Κυκλώματος Αυτοματισμού Κατοικίας

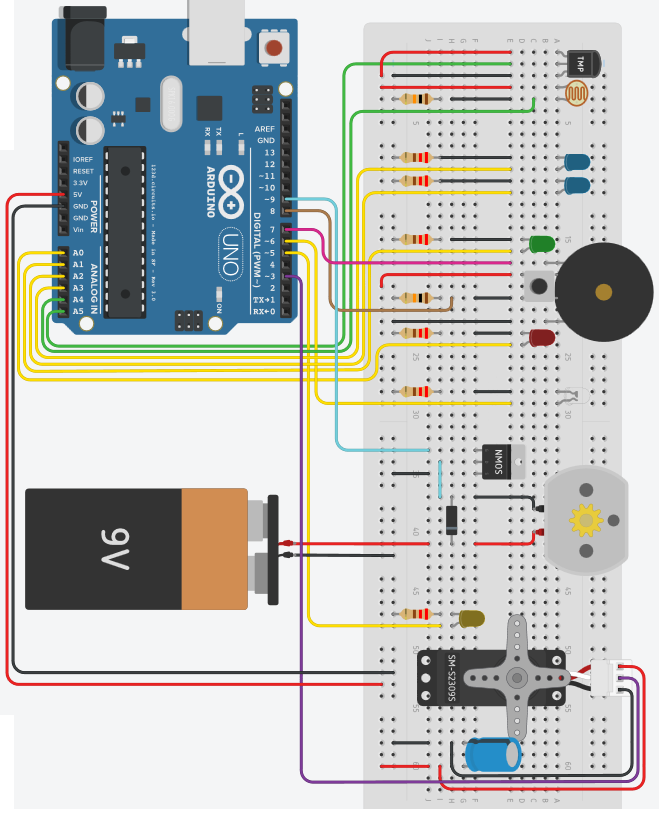
Όσων αφορά τον αυτοματισμό κατοικίας, χρειάστηκαν μερικά εξαρτήματα από το επίσημο Arduino Starter Kit:

* Ένα Solderless Breadboard για την κατασκευή των κυκλωμάτων
* Δυο μπλε, ένα κίτρινο, ένα κόκκινο, ένα πράσινο και ένα διάφανο LED
* Ένας αισθητήρας φωτός και ένας αισθητήρας θερμοκρασίας
* Ένα μικρό Servo Motor και ένα μικρό DC Motor
* Ένα κουμπί
* Ένας βομβητής
* Ένας πυκνωτής, μία δίοδος και αντιστάτες
* Ένα Τρανζίστορ τύπου MOSFET

Έπειτα θα προσομοιώσουμε κάθε συσκευή που υπάρχει στο έξυπνο σπίτι μας χρησιμοποιώντας τα παραπάνω εξαρτήματα

* Για τα Φώτα θα χρησιμοποιήσουμε δυο LED με αντιστάτες
* Για το Θερμοσίφωνο θα χρησιμοποιήσουμε το μικρο Servo Motor, ένα κίτρινο LED, μια αντίσταση και ένα πυκνωτή. Το Servo Motor θα προσωμοιώνει τον δείκτη ζεστού νερού στο θερμοσίφωνο (δυνατότητα περιστροφής 180°). Το κίτρινο LED θα προσωμοιώνει το φωτάκι ένδειξης.
* Για το κλιματιστικό θα χρησιμοποιήσουμε το DC Motor μαζί με το Τρανζίστορ τύπου MOSFET και την δίοδο
* Για τα Πατζούρια θα χρησιμοποιήσουμε ένα διάφανο LED. Όταν το LED είναι αναμμένο, τα πατζούρια είναι ανοιχτά. Θα χρησιμοποιήσουμε PWM για να προσωμοιώσουμε το αργό άνοιγμα των πατζουριών.
* Για τον συναγερμό θα χρησιμοποιήσουμε ενα πράσινο LED, ένα κόκκινο LED, τον βομβητή και το κουμπί. Όταν ο συναγερμός είναι απενεργοποιημένος θα ανάβει το πράσινο LED, όταν ενεργοποιείται το κόκκινο. Μόλις πατηθεί το κουμπί για αρκετή ώρα, σημαίνει ότι παραβιάζεται ο συναγερμός, και άρα ενεργοποιείται ο βομβητής.

Το κύκλωμα έχει ως εξής:



Ε.5. Το κύκλωμα προσομοίωσης του έξυπνου σπιτιού

Στην παραπάνω φωτογραφία επισημαίνουμε τα εξης:

* Κάθε κόκκινο καλώδιο συνδέεται με τάση 5V
* Κάθε μαύρο καλώδιο είναι γείωση
* Τα κίτρινα καλώδια συνδέονται με LED
* Το μοβ καλώδιο συνδέεται με το servo motor
* Το γαλάζιο καλώδιο συνδέεται με το DC motor
* Το ροζ καλώδιο συνδέεται με βομβητή
* Το καφέ καλώδιο συνδέεται με κουμπί
* Τα πράσινα καλώδιο συνδέονται με αισθητήρες

Κάποια πράγματα που πρέπει να αναφέρουμε όσων αφορά τα κυκλώματα είναι τα εξής:

* Για τα LED χρησιμοποιούμε αντιστάτες 220ohm
* Για τον αισθητήρα φωτός και το κουμπί χρησιμοποιούμε αντιστάτες 10K ohm
* Για το servo motor χρησιμοποιούμε πυκνωτή για ασφάλεια
* Για το DC motor, επειδή το Arduino μέσω των ακροδεκτώς χρησιμοποιεί 5V τάση η οποία δεν είναι επαρκής, χρησιμοποιούμε 9V μπαταρία. Για να το ενεργοποιήσουμε μέσω Arduino χρησιμοποιούμε το τρανζιστορ τύπου MOSFET με την παραπάνω συνδεσμολογία
* Για το DC motor, επειδή όταν σταματάει να περιστρέφεται, γυρνάει για λίγο ακόμα λόγω της ήδη κεκτημένης ταχύτητας, υπάρχει ροή ρεύματος προς τα πίσω. Για να το αποφύγουμε αυτο χρησιμοποιούμε δίοδο.

## 4.3 Προγραμματισμός Arduino

Για να υλοποιήσουμε την εφαρμογή μας, θα χρειαστούμε το κατάλληλο περιβάλλον ανάπτυξης λογισμικού. Το περιβάλλον που χρησιμοποιήθηκε για τον προγραμματισμό της πλακέτας είναι το Arduino Software 1.7.8.

Θα χρειαστεί να δημιουργήσουμε ένα νέο sketch. Ένα sketch αποτελείται από δυο βασικές συναρτήσεις, την setup και την loop. Η συνάρτηση setup τρέχει μια φορά στην αρχή του προγράμματος μας οπότε χρησιμοποιείται για αρχικοποιήσεις μεταβλητών ή για κώδικα που θέλουμε να τρέξει πριν το κυρίως πρόγραμμά μας. Η συνάρτηση loop επαναλαμβάνεται συνεχώς και εδώ γράφουμε το πρόγραμμα μας με την γνώση ότι θα επαναλαμβάνεται ο κώδικας επ άπειρον.

Η δομή του προγράμματος μας θα είναι η εξής:

* Συνδέσου με τον διακομιστή
* Ανάκτησε την κατάσταση όλων τον συσκευών
* Ανανέωσε την κατάσταση όλων των αισθητήρων
* Ανάλογα με την κατάσταση της κάθε συσκευής, εκτέλεσε τις αντίστοιχες λειτουργίες
* Επανέλαβε τα προηγούμενα βήματα

### 4.3.1 Υλοποίηση Συναρτήσεων Αιτημάτων HTTP

Πριν αρχίσουμε να γράφουμε το κυρίως πρόγραμμα, θα χρειαστούμε τις συναρτήσεις GET και PUT, οι οποίες θα συνδέονται με τον διακομιστή του κεφαλαίου 2 και θα ανακτούν και ανανεώνουν τα δεδομένα του διακομιστή αντίστοιχα. Μιας και έχουμε συσκευές, για τις οποίες θα χρειάζεται να ανακτούμε την κατάσταση τους απο τον διακομιστή, η συνάρτηση GET είναι απαραίτητη. Μιας και έχουμε αισθητήρες, των οποίων την κατάσταση θα χρειάζεται να ανανεώνουμε στον διακομιστή, η συνάρτηση PUT είναι επίσης απαραίτητη.

#### 4.3.1.1 Συνάρτηση getRequest

String getRequest(String theRequest){

**Serial**.print("Connecting : ");

while (!client.connect(restServer, 8080)) {

**Serial**.println("Connection failed");

**Serial**.print("Reconnecting : ");

}

client.println("GET /restServer/resources/" + theRequest);

**Serial**.print("GET /restServer/resources/" + theRequest + " : ");

delay(500);

String response = "";

while (client.available()) {

char c = client.read();

response.concat(c);

}

Serial.println(response);

client.flush();

if (!client.connected())

client.stop();

return response;

}

Όπως παρατηρούμε στον παραπάνω κώδικα, η συνάρτηση getRequest δέχεται ένα αλφαριθμητικό με το αίτημα προς τον διακομιστή (το οποίο είναι το URI μιας συσκευής ή αισθητήρα) και επιστρέφει την απάντηση του διακομιστή. Έπειτα συνδέεται με τον διακομιστή στην θύρα 8080, η θύρα που ακούει ο διακομιστής. Στέλνει ένα αλφαριθμητικό με την λέξη “GET” στην αρχή, δηλαδή δηλώνει τι είδους αίτημα είναι στον διακομιστή και έπειτα ακολουθεί το URI που έχει δεχτεί απο το πρόγραμμα. Διαβάζει τους χαρακτήρες της απαντήσεως και τους προσθέτει έναν έναν στο αλφαριθμητικό που θα επιστρέψει. Τέλος κλείνει τη σύνδεση.

#### 4.3.1.1 Συνάρτηση putRequest

void putRequest(String theRequest, String message){

String serverIP = "";

**Serial**.print("Connecting : ");

while (!client.connect(restServer, 8080)) {

**Serial**.println("Connection failed");

**Serial**.print("Reconnecting : ");

}

String tempRequest = "";

for (byte thisByte = 0; thisByte < 4; thisByte++) {

serverIP += String(restServer[thisByte]);

if(thisByte <3)

serverIP +=(".");

}

tempRequest += "PUT /restServer/resources/" + theRequest

+ " HTTP/1.1\r\n";

tempRequest += "Host: " + serverIP + "\r\n";

tempRequest += "Content-Type: text/plain; charset=UTF-8\r\n";

tempRequest += "Content-Length: " + String(message.length(),DEC)

+ "\r\n";

tempRequest += "\r\n";

tempRequest += message;

client.println(tempRequest);

**Serial**.println("PUT /restServer/resources/" + theRequest);

delay(500);

client.flush();

client.stop();

}

Όπως παρατηρούμε παραπάνω η συνάρτηση putRequest δεν επιστρέφει τίποτα, ενώ δέχεται ένα αλφαριθμητικό με το URI μιας συσκευής ή αισθητήρα καθώς και την τιμή με την οποία θα ανανεωθεί ο διακομιστής. Έπειτα συνδέεται με τον διακομιστή στην θύρα 8080, η θύρα που ακούει ο διακομιστής. Μετατρέπει την διεύθυνση του διακομιστή σε κατάλληλη μορφή για να σταλεί. Στέλνει ένα αλφαριθμητικό με την λέξη “PUT” στην αρχή, δηλαδή δηλώνει τι είδους αίτημα είναι στον διακομιστή και έπειτα ακολουθεί το URI που έχει δεχτεί απο το πρόγραμμα. Ακολουθεί η διεύθυνση του διακομιστή και η δήλωση ότι το μύνημα θα είναι σε μορφή απλού κειμένου. Μετά, αναγράφεται το μήκος του μηνύματος και ακολουθεί το ίδιο το μυνημα. Τέλος κλείνει τη σύνδεση.

### 4.3.2 Υλοποίηση Κυρίως Προγράμματος Arduino

Αρχικά περιλαμβάνουμε τις βιβλιοθήκες που θα χρησιμοποιήσουμε. Θέλουμε να έχουμε επικοινωνία με το Arduino μέσω της σειριακής θύρας, θέλουμε να χρησιμοποιήσουμε την θύρα Ethernet για να συνδεθούμε στο διαδίκτυο και τέλος θέλουμε να χειριζόμαστε ένα servo motor.

#include <SPI.h>

#include <Ethernet.h>

#include <Servo.h>

Έπειτα θα δηλώσουμε την διεύθυνση MAC του Ethernet shield. Η προτεινόμενη MAC βρίσκεται εκτυπωμένη σε ένα αυτοκόλλητο πάνω στην shield. Επίσης, δηλώνουμε την IP διεύθυνση του διακομιστή που θέλουμε να συνδεθούμε καθώς και την IP διεύθυνση που επιθυμούμε να πάρει η shield στο τοπικό δίκτυο:

byte mac[] = { 0x90,0xA2,0xDA, 0x0D, 0xD5,0x49 };

IPAddress restServer(83,212,118,164);

IPAddress ip(192,168,1,177);

Δηλώνουμε τις μεταβλητές για τις τιμές των συσκευών που ανακτούμε από τον διακομιστή:

boolean serverLight1;

boolean serverLight2;

boolean serverBoiler;

boolean serverAirCondition;

boolean serverBlinds;

boolean serverAlarm;

Πρέπει να δηλώσουμε το servo motor και τον πελάτη που θα συνδεθεί με τον διακομιστή:

EthernetClient client;

Servo boiler;

Ακολουθεί η δήλωση των ακροδεκτών και η αντιστοιχία τους με σταθερές

const int roomLight1 = A3;

const int roomLight2 = A2;

const int alarmGreenLight = A1;

const int alarmRedLight = A0;

const int boilerLight = 5;

const int blindsSimLight = 6;

const int alarmBeeper = 7;

const int alarmButton = 8;

const int airCondition = 9;

const int boilerServo = 3;

const int roomLightLevel = A4;

const int roomTempLevel = A5;

Ακολουθεί η δήλωση σταθερών αλφαριθμητικών η οποία αποτελείται κυρίως από URIs συσκευών και αισθητήρων

const String DEVICE\_LIGHT1 = "device/1";

const String DEVICE\_LIGHT2 = "device/2";

const String DEVICE\_BOILER = "device/3";

const String DEVICE\_AIRCONDITION = "device/4";

const String DEVICE\_BLINDS = "device/5";

const String DEVICE\_ALARM = "device/6";

const String SENSOR\_TEMPERATURE = "sensor/1";

const String SENSOR\_HOTWATER = "sensor/2";

const String SENSOR\_LIGHTLEVEL = "sensor/3";

const String SENSOR\_BREAKIN = "sensor/4";

const String DEVICE\_ON = "true";

const String DEVICE\_OFF = "false";

const String SENSOR\_ON = "1";

const String SENSOR\_OFF = "0";

Τέλος ακολουθεί η δήλωση μεταβλητών για τους αισθητήρες καθώς και διάφορες βοηθητικές μεταβλητές που χρησιμοποιούμε για διευκόλυνση μας μέσα στο πρόγραμμα

int boilerPercentage = 0;

int temperature;

int lightLevel;

int alarmTriggered = 0;

float tempTemp;

int boilerCounter = 0;

int blindsState = 0;

Μπορούμε να αρχίσουμε πια να αναλύσουμε τη συνάρτηση setup. Μέσα σε αυτή θέτουμε σε όλους τους ακροδέκτες τιμές INPUT ή OUTPUT, ανάλογα με το αν είναι ακροδέκτες εισόδου ή εξόδου αντίστοιχα. Αυτός είναι ο σκοπός της συνάρτησης pinMode. Έπειτα αρχικοποιούμε το servo motor μας, αρχίζουμε την σειριακή σύνδεση με τον υπολογιστή και λαμβάνουμε μια τοπική διεύθυνση IP.

void **setup**() {

pinMode(roomLight1,OUTPUT);

pinMode(roomLight2,OUTPUT);

pinMode(alarmGreenLight,OUTPUT);

pinMode(alarmRedLight,OUTPUT);

pinMode(boilerLight,OUTPUT);

pinMode(blindsSimLight,OUTPUT);

pinMode(alarmButton,INPUT);

pinMode(airCondition,OUTPUT);

boiler.attach(boilerServo);

boiler.write(0);

**Serial**.begin(9600);

**Serial**.println("Starting Ethernet Connection... ");

Ethernet.begin(mac,ip);

ip = Ethernet.localIP();

printIPAddress();

}

Ακολουθεί η συνάρτηση loop. Αρχικά συνδεόμαστε με τον διακομιστή και ανακτούμε τις τιμές των συσκευών, μετατρέποντας τες σε λογικές τιμές.

void **loop**() {

serverLight1 = toBoolean(getRequest(DEVICE\_LIGHT1));

serverLight2 = toBoolean(getRequest(DEVICE\_LIGHT2));

serverBoiler = toBoolean(getRequest(DEVICE\_BOILER));

serverAirCondition = toBoolean(getRequest(DEVICE\_AIRCONDITION));

serverBlinds = toBoolean(getRequest(DEVICE\_BLINDS));

serverAlarm = toBoolean(getRequest(DEVICE\_ALARM));

Παίρνουμε την τιμή της θερμοκρασίας από τον αντίστοιχο αισθητήρα και την μετατρέπουμε από τιμή τάσης ρεύματος σε θερμοκρασία. Στέλνουμε την καινούρια τιμή στον διακομιστή

temperature = analogRead(roomTempLevel);

tempTemp = (temperature/1024.0) \* 5.0;

tempTemp = (tempTemp - .5) \* 100;

temperature = (int)(tempTemp + .5);

putRequest(SENSOR\_TEMPERATURE,String(temperature));

Παίρνουμε την τιμή του επιπέδου φωτός από τον αντίστοιχο αισθητήρα και την μετατρέπουμε από τιμή τάσης ρεύματος σε ποσοστό τοις εκατό. Στέλνουμε την καινούρια τιμή στον διακομιστή.

lightLevel = analogRead(roomLightLevel);

lightLevel = map(lightLevel,0,1023,0,100);

putRequest(SENSOR\_LIGHTLEVEL,String(lightLevel));

Μετά αρχίζουν οι έλεγχοι με βάση τα δεδομένα που ανακτήθηκαν απο τον διακομιστή. Αρχίζουμε με τα δυο φώτα. Αν στον διακομιστή η τιμή για το κάθε φως είναι αληθής, τότε ανάβουμε τα φώτα. Αλλιώς τα σβήνουμε.

if(serverLight1)

digitalWrite(roomLight1,HIGH);

else

digitalWrite(roomLight1,LOW);

if(serverLight2)

digitalWrite(roomLight2,HIGH);

else

digitalWrite(roomLight2,LOW);

Για το θερμοσίφωνο θέλουμε όταν ενεργοποιείται, να ανάβει το κίτρινο LED και να αρχίσει να περιστρέφεται το servo motor, το οποίο προσομοιώνει τον δείκτη πάνω στο θερμοσίφωνο. Επίσης θέλουμε όταν ζεσταθεί 100% το νερό η όταν κλείσουμε το θερμοσίφωνο, να σβήσει το κίτρινο LED και να αρχίσει το servo motor να περιστρέφεται προς την αντίθετη μεριά. Επίσης υπάρχει ένας εικονικός αισθητήρας που μετράει το ποσοστό του ζεστού νερού στο θερμοσίφωνα. Θέλουμε σε κάθε επανάληψη να ανανεώνεται αυτός ο αισθητήρας στον διακομιστή. Αυτό το πετυχαίνουμε με τον παρακάτω κώδικα.

if(serverBoiler){

digitalWrite(boilerLight,HIGH);

if(boilerCounter < 176){

boilerCounter += 8;

boiler.write(boilerCounter);

boilerPercentage = map(boilerCounter,0,179,0,100);

putRequest(SENSOR\_HOTWATER,String(boilerPercentage));

}

else{

putRequest(DEVICE\_BOILER,DEVICE\_OFF);

}

}

else{

digitalWrite(boilerLight,LOW);

if (boilerCounter > 0){

boilerCounter-= 8;

boiler.write(boilerCounter);

boilerPercentage = map(boilerCounter,0,179,0,100);

putRequest(SENSOR\_HOTWATER,String(boilerPercentage));

}

}

Για το κλιματιστικό η διαδικασία είναι απλή όπως και στα φώτα

if(serverAirCondition)

digitalWrite(airCondition,HIGH);

else

digitalWrite(airCondition,LOW);

Για τα παντζούρια χρησιμοποιούμε μια επανάληψη και το analogWrite για να ανάψουμε σταδιακά το διάφανο LED. Η μεταβλητή blindsState ελέγχετε για να μην συμβαίνει η διαδικασία αυτή κάθε φορά, ακόμα και αν ήταν ήδη αναμμένο το LED. Υπάρχει καθυστέρηση 20 ms μετά από κάθε αλλαγή φωτεινότητας. Το LED έχει 256 επίπεδα φωτεινότητας.

if(serverBlinds && blindsState == 0){

for(int i=0;i<256;i+=4){

analogWrite(blindsSimLight,i);

delay(20);

}

blindsState = 1;

}

else if (!serverBlinds && blindsState == 1){

for(int i=252;i>=0;i-=4){

analogWrite(blindsSimLight,i);

delay(20);

}

blindsState = 0;

}

Για τον συναγερμό, αν έχει ενεργοποιηθεί στον διακομιστή, σβήνουμε το πράσινο LED και ανάβουμε το κόκκινο. Αλλιώς κάνουμε το ανάποδο:

if(serverAlarm){

digitalWrite(alarmRedLight,HIGH);

digitalWrite(alarmGreenLight,LOW);

}

else{

digitalWrite(alarmRedLight,LOW);

digitalWrite(alarmGreenLight,HIGH);

}

Τέλος θα πρέπει να προσθέσουμε λειτουργικότητα για εικονική παραβίαση του συναγερμου. Η προσομοίωση αυτή γίνεται με ένα κουμπί. Αν ο συναγερμός είναι ενεργοποιημένος και το κουμπί είναι πατημένο, ανανεώνουμε τον αισθητήρα στον διακομιστή. Αν ο αισθητήρας είναι ενεργοποιημένος στον διακομιστή, ενεργοποιείται ο βομβητής σαν ένα είδος σειρήνας. Αν όχι, απενεργοποιούμε τον βομβητή.

if(digitalRead(alarmButton) == HIGH && serverAlarm)

putRequest(SENSOR\_BREAKIN,SENSOR\_ON)

alarmTriggered = getRequest(SENSOR\_BREAKIN).toInt();

if (alarmTriggered == 1)

tone(alarmBeeper,330);

else

noTone(alarmBeeper);

# 5. Android

Το Android είναι ένα λειτουργικό σύστημα για κινητές συσκευές που έχει αναπτυχθεί από την Google και βασίζεται στον πυρήνα του Linux. Έχει σχεδιαστεί κυρίως για φορητές συσκευές με οθόνη αφής όπως smartphones και tablets. Είναι το λειτουργικό σύστημα με τις περισσότερες πωλήσεις σε tablets και smartphones από το 2013.

Η διεπαφή χρήστη του Android βασίζεται κυρίως στην άμεση επεξεργασία, παρέχοντας δυνατότητες αφής που αντιστοιχούν σε πραγματικές δράσεις, όπως το σύρσιμο, το άγγιγμα και το τσίμπημα, για να χρησιμοποιήσουν τα αντικείμενα που εμφανίζονται στην οθόνη. Επίσης περιλαμβάνει και ένα εικονικό πληκτρολόγιο για την εισαγωγή κειμένου. Η σχεδιαστική γλώσσα του λειτουργικού συστήματος είναι η Material Design της Google. Το κατάστημα εφαρμογών του Android είναι το Google Play, με πάνω από ένα εκατομμύριο δημοσιευμένες εφαρμογές Android ( "apps") και 50 δισεκατομμύρια λήψεις από τον Ιούλιο του 2013. Εκτός από τις συσκευές με οθόνη αφής, η Google έχει αναπτύξει περαιτέρω το Android για τηλεόραση, αυτοκίνητα, και ρολόγια χειρός, με παρόμοια διεπαφή. Εκδόσεις του Android χρησιμοποιούνται επίσης για φορητούς υπολογιστές, κονσόλες παιχνιδιών, ψηφιακές φωτογραφικές μηχανές και άλλα ηλεκτρονικά είδη.

Το Android είχε αρχικά σχεδιαστεί από την Android, Inc, την οποία αγόρασε η Google το 2005. Το λειτουργικό σύστημα παρουσιάστηκε το 2007, μαζί με την ίδρυση του Open Handset Alliance - μια κοινοπραξία εταιριών υλικού, λογισμικού και τηλεπικοινωνιών οι οποίες έχουν στόχο την προώθηση ανοιχτών προτύπων για φορητές συσκευές. Τον Σεπτέμβριο του 2015, το Android είχε 1,4 δισεκατομμύρια ενεργούς χρήστες μηνιαίως. (Wikipedia, 2016)

## 5.1 Σχεδιασμός Εφαρμογής

Σκοπός μας είναι να υλοποιήσουμε μια εφαρμογή Android θα χρειαστούμε ένα περιβάλλον ανάπτυξης λογισμικού. Αν και θα ήταν βολικό να χρησιμοποιήσουμε για ακόμα μια φορά το eclipse, το περιβάλλον υλοποίησης του διακομιστή μας, από τον Ιούνιο του 2015 δεν υπάρχει υποστήριξη των Android Developer Tools για eclipse. Αντί γι’ αυτό, υπάρχει το Android Studio της εταιρίας Google. (Developer.android.com, 2016)

Πρέπει επίσης να αποφασίσουμε σε ποια έκδοση του Android θα προγραμματίσουμε την εφαρμογή μας. Όσο πιο καινούρια η έκδοση, τόσο περισσότερες δυνατότητες δίνονται στον προγραμματιστή. Αυτό βέβαια σημαίνει επίσης ότι η εφαρμογή μας θα τρέχει σε λιγότερες κινητές συσκευές. Κάνουμε την επιλογή μας έτσι ώστε να η εφαρμογή να είναι προσβάσιμη από ένα καλό ποσοστό κινητών στον κόσμο.

Οι εκδόσεις Android είναι οι ακόλουθες:

|  |  |  |  |
| --- | --- | --- | --- |
| Έκδοση | Όνομα | API | Ποσοστό Χρήσης |
| 1.0 |  | 1 | 0.0% |
| 1.1 |  | 2 | 0.0% |
| 1.5 | [Cupcake](https://en.wikipedia.org/wiki/Android_Cupcake) | 3 | 0.0% |
| 1.6 | [Donut](https://en.wikipedia.org/wiki/Android_Donut) | 4 | 0.0% |
| 2.0–2.1 | [Eclair](https://en.wikipedia.org/wiki/Android_Eclair) | 5–7 | 0.0% |
| [2.2](https://developer.android.com/about/versions/android-2.2.html) | Froyo | 8 | 0.1% |
| [2.3.3 - 2.3.7](https://developer.android.com/about/versions/android-2.3.3.html) | Gingerbread | 10 | 2.0% |
| 3.0–3.2.6 | [Honeycomb](https://en.wikipedia.org/wiki/Android_Honeycomb) | 11–13 | 0.0% |
| [4.0 - 4.0.4](https://developer.android.com/about/versions/android-4.0.html) | Ice Cream Sandwich | 14-15 | 1.9% |
| [4.1.x](https://developer.android.com/about/versions/android-4.1.html) | Jelly Bean | 16 | 6.8% |
| [4.2.x](https://developer.android.com/about/versions/android-4.2.html) | 17 | 9.4% |
| [4.3](https://developer.android.com/about/versions/android-4.3.html) | 18 | 2.7% |
| [4.4](https://developer.android.com/about/versions/android-4.4.html) | KitKat | 19 | 31.6% |
| [5.0](https://developer.android.com/about/versions/android-5.0.html) | Lollipop | 21 | 15.4% |
| [5.1](https://developer.android.com/about/versions/android-5.1.html) | 22 | 20.0% |
| [6.0](https://developer.android.com/about/versions/marshmallow/index.html) | Marshmallow | 23 | 10.1% |

Π.2 Εκδόσεις Android

Θα επιλέξουμε να προγραμματίσουμε πάνω στο API 14 με το όνομα Ice Cream Sandwich διότι θα μπορεί να τρέξει στο 97.9% των συσκευών παγκοσμίως.

Για να σχεδιάσουμε τα παράθυρα ή αλλιώς activities της εφαρμογής μας, θα χρησιμοποιήσουμε τις δυνατότητες σχεδιασμού που μας δίνει το Android Studio. Οποιεσδήποτε σχεδιαστικές αλλαγές κάνουμε αποθηκεύονται με την μορφή xml αρχείου στο φάκελο layout του project μας.

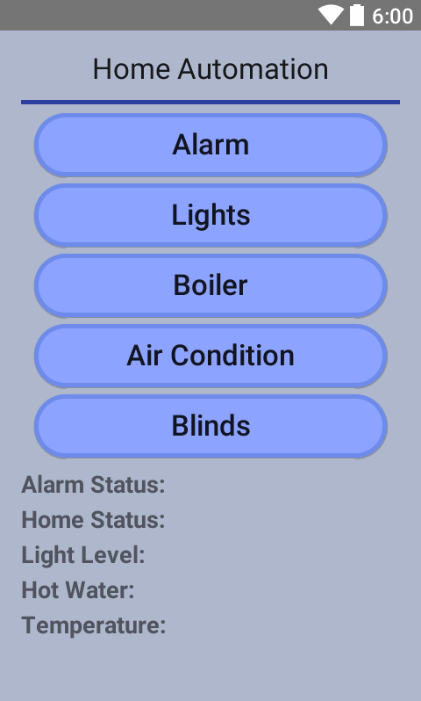
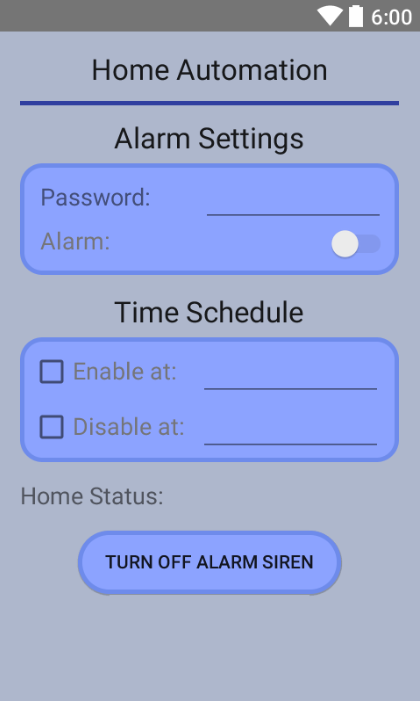
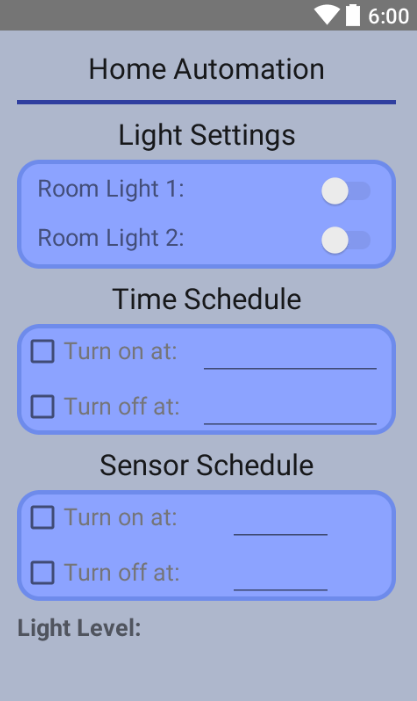
Αρχικά, όταν ο χρήστης θα τρέχει την εφαρμογή μας, θα ανοίγει το αρχικό activity στο οποίο να εμφανίζονται οι τιμές από όλους τους αισθητήρες. Το αρχικό αυτό παράθυρο θα πρέπει επίσης να περιέχει κουμπιά για τις διάφορες συσκευές του έξυπνου σπιτιού, δηλαδή για τον φωτισμό, το θερμοσίφωνο, το κλιματιστικό, τα παντζούρια και τον συναγερμό.

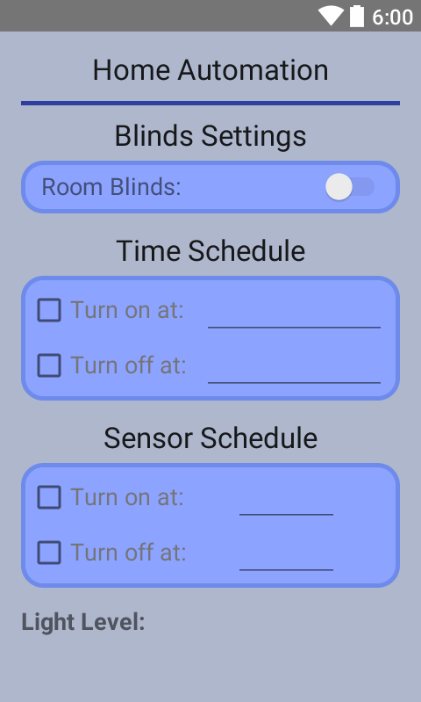
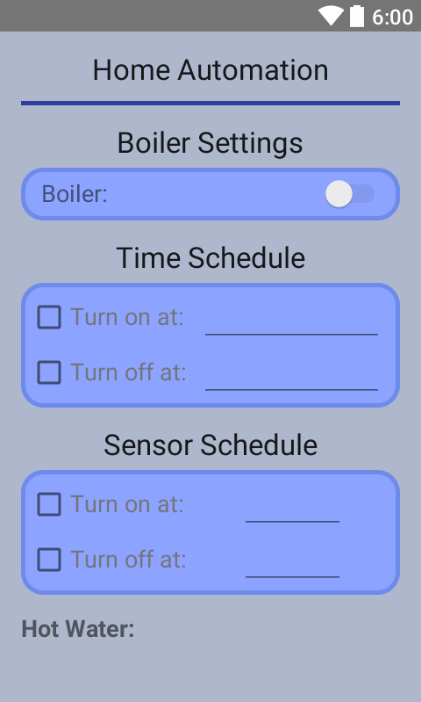
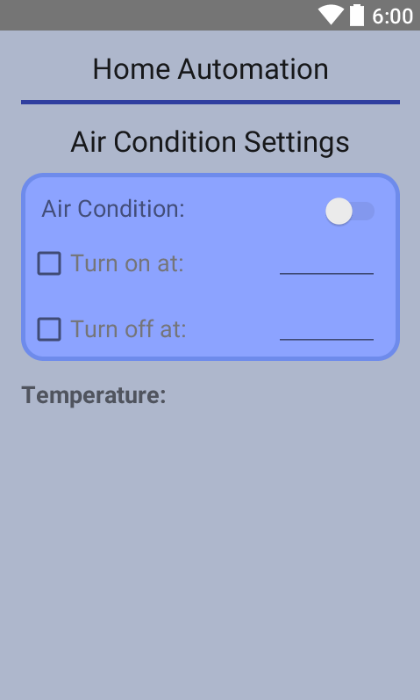
Όταν ο χρήστης πατάει πάνω στο κουμπί για τον φωτισμό, θα πηγαίνει στο παράθυρο φωτισμού, το οποίο θα περιέχει διακόπτες και για τα δυο φώτα. Επίσης θα περιέχει σημεία όπου ο χρήστης θα πληκτρολογεί την ώρα που θέλει να ενεργοποιηθούν/ απενεργοποιηθούν τα φώτα. Επιπλέον θα πρέπει να επιτρέπει στον χρήστη να ενεργοποιεί/απενεργοποιεί τον φωτισμό όταν το επίπεδο του φωτός είναι μικρότερο/μεγαλύτερο μιας τιμής αντίστοιχα. Τέλος, θα πρέπει να εμφανίζουμε το επίπεδο του φωτός του αισθητήρα.

Όσων αφορά το activity για τον θερμοσίφωνο, χρειαζόμαστε έναν διακόπτη για να ανάβουμε το θερμοσίφωνο, πεδία για αλλαγή της ώρας ενεργοποίησης και απενεργοποίησης του και πεδία για ενεργοποίηση/απενεργοποίηση βάση τις τιμής του ζεστού νερού. Τέλος θα πρέπει να εμφανίζουμε το ποσοστό ζεστού νερού μέσα στον θερμοσίφωνα.

Για το activity του κλιματιστικού χρειαζόμαστε έναν διακόπτη για την ενεργοποίηση και απενεργοποίηση του, πεδία για ενεργοποίηση και απενεργοποίηση ανάλογα με την θερμοκρασία του δωματίου και τέλος την εμφάνιση της ίδιας της θερμοκρασίας.

Για το activity των παντζουριών θα χρειαστούμε έναν διακόπτη για ενεργοποίηση/απενεργοποίηση, πεδία για ενεργοποίηση ή απενεργοποίηση ανάλογα με την ώρα που θα εισάγει ο χρήστης και ενεργοποίηση/απενεργοποίηση βάση της τιμής επιπέδου φωτός. Πρέπει επίσης να εμφανίζουμε το επίπεδο φωτός.

Για το activity του συναγερμού θα χρειαστούμε πεδίο που ο χρήστης θα εισάγει τον κωδικό ασφαλείας, έναν διακόπτη που θα ενεργοποιεί/απενεργοποιεί τον συναγερμό και πεδία για χρονοπρογραμματισμό του συναγερμού. Επίσης θα εμφανίζουμε αν έχει παραβιαστεί ο συναγερμός και κουμπί απενεργοποίησης της ηχητικής ειδοποίησης του συναγερμού.

 Ε.6.HomeScreen Design Ε.7. AlarmScreen Design Ε.8. LightScreen Design

Ε.9. BoilerScreen Design Ε.10. BlindsScreen Design Ε.11. AirConditionScreen Design

## 5.2 Υλοποίηση Βοηθητικών Μεθόδων

Αφού σχεδιάσαμε την διεπαφή του χρήστη, τώρα μπορούμε να αρχίσουμε να υλοποιούμε την λογική του προγράμματος. Το πρόγραμμά μας θα γραφτεί με την γλώσσα προγραμματισμού Java. Αρχικά θα δημιουργήσουμε μεθόδους GET και PUT που θα επικοινωνούν με τον διακομιστή μας. Έπειτα θα υλοποιήσουμε την λογική της διεπαφής. Επειδή όπως είδαμε στο προηγούμενο κεφάλαιο, τα στοιχεία των περισσότερων activities δεν διαφέρουν πολύ, θα μπορούσαμε να υλοποιήσουμε μεθόδους που θα καλεί το κάθε activity. Τέλος θα υλοποιήσουμε τα ίδια τα activities.

### 5.2.1 Μέθοδοι HTTP

Θα δημιουργήσουμε μια κλάση HttpRequests μέσα στην οποία θα υλοποιούμε τις δύο μεθόδους HTTP για την ανάκτηση και την ανανέωση τιμών του διακομιστή. Τα activities της εφαρμογής θα καλούν αυτές τις στατικές μεθόδους για την επεξεργασία των αιτημάτων του.

Η μέθοδο GET θα δέχεται ένα αλφαριθμητικό με το URI του στοιχείου που θέλουμε να ανακτήσουμε από τον διακομιστή και θα επιστρέφει την τιμή που θα στέλνει ο διακομιστής σαν απάντηση. Το URI που δέχεται από τα activities, το συνδυάζει με την IP διεύθυνση του διακομιστή και δημιουργεί ένα καινούριο instance της κλάσης url. Έπειτα ανοίγει μία HTTP σύνδεση με βάση αυτό το url. Θέτει την μέθοδο του αιτήματος προς τον διακομιστή να είναι η μέθοδο GET και θέτει τον χρόνο αναμονής στα δυο δευτερόλεπτα.

**public static** String getRequest(String urlText){  
 String response = **""**;  
 **try** {  
 URL url = **new** URL(**"http://"** + Constant.***SERVER\_IP*** +

**":8080/restServer/resources/"** + urlText);  
 HttpURLConnection urlConnection =

(HttpURLConnection) url.openConnection();  
 urlConnection.setRequestMethod(**"GET"**);  
 urlConnection.setConnectTimeout(2000);

Διαβάζει την επιστρεφόμενη τιμή μέσω ενός InputStream, την συλλέγει γράμμα γράμμα από τον διακομιστή, την επιστρέφει στο activity και αποσυνδέεται.

InputStream in = urlConnection.getInputStream();

InputStreamReader isw = **new** InputStreamReader(in);

**int** data = isw.read();

**while** (data != -1) {

**char** current = (**char**) data;

response+= Character.*toString*(current);

data = isw.read();

}  
urlConnection.disconnect();  
**return** response;

Όλο αυτό το κομμάτι περιβάλλεται από ένα try – catch με σκοπό η μέθοδο να προβλέψει τα σφάλματα και να τα χειριστεί. Χειρίζεται σφάλματα παραμορφωμένου url, σφάλματα εισόδου- εξόδου και σφάλματα μη εύρεσης του διακομιστή. Επιστρέφει σε αυτές τις περιπτώσεις το κατάλληλο μήνυμα.

}**catch**(MalformedURLException e){  
 System.***out***.println(**"MALFORMED URL ERROR"**);  
 **return "SERVER ERROR"**;  
 }**catch** (IOException e){  
 System.***out***.println(**"INPUT OUTPUT ERROR"**);  
 **return "SERVER ERROR"**;  
 }**catch** (Exception e){  
 System.***out***.println(**"SERVER NOT FOUND"**);  
 **return "SERVER ERROR"**;  
 }  
}

Η μέθοδο PUT θα δέχεται ένα αλφαριθμητικό με το URI του στοιχείου που θέλουμε να ανανεώσουμε στον διακομιστή και την τιμή ανανέωσης. Το URI που δέχεται από τα activities, το συνδυάζει με την IP διεύθυνση του διακομιστή και δημιουργεί ένα καινούριο instance της κλάσης url. Έπειτα ανοίγει μία HTTP σύνδεση με βάση αυτό το url. Σηματοδοτεί ότι στέλνει δεδομένα στον διακομιστή, θέτει την μέθοδο του αιτήματος προς τον διακομιστή να είναι η μέθοδο PUT και θέτει τον χρόνο αναμονής στα δυο δευτερόλεπτα. Σηματοδοτεί ότι τα δεδομένα που στέλνει είναι στη μορφή απλού κειμένου και τέλος κάνει την σύνδεση.

**public static void** putRequest(String urlText, String message){  
 **try** {  
 URL url = **new** URL(**"http://"** + Constant.***SERVER\_IP*** +

**":8080/restServer/resources/"** + urlText);  
 HttpURLConnection urlConnection =

(HttpURLConnection) url.openConnection();  
 urlConnection.setDoOutput(**true**);  
 urlConnection.setRequestMethod(**"PUT"**);  
 urlConnection.setConnectTimeout(2000);  
 urlConnection.setRequestProperty(**"Content-Type"**,

**"text/plain; charset=UTF-8"**);  
 urlConnection.connect();

Στέλνει το μήνυμα που δέχτηκε από το activity χρησιμοποιώντας ένα αντικείμενο OutputStream και εκτυπώνει τον κωδικό που επιστρέφει ο διακομιστής. Έπειτα αποσυνδέεται.

OutputStreamWriter out = **new** OutputStreamWriter

(urlConnection.getOutputStream());  
out.write(message);  
  
out.close();  
System.***out***.println(**"Response: "** +

urlConnection.getResponseMessage());  
urlConnection.disconnect();

Όπως και στην μέθοδο GET, έχουμε και εδώ χειρισμό σφαλμάτων.

}**catch** (ProtocolException e) {  
 System.***out***.println(**"PROTOCOL ERROR"**);  
 }**catch**(MalformedURLException e){  
 System.***out***.println(**"MALFORMED URL ERROR"**);  
 }**catch** (IOException e){  
 System.***out***.println(**"INPUT OUTPUT ERROR"**);  
 }  
 }  
}

### 5.2.2 Καθολικές Μέθοδοι

Σχεδόν όλα τα activities μοιράζονται χαρακτηριστικά όσων αφορά τον σχεδιασμό τους. Σχεδόν όλα έχουν διακόπτες για την κάθε λειτουργία, πεδία για χρονοπρογραμματισμό και για ενεργοποίηση/απενεργοποίηση με βάση κάποιον αισθητήρα. Για να κάνουμε την ζωή μας πιο εύκολη, μπορούμε να δημιουργήσουμε καθολικές μεθόδους οι οποίοι θα δέχονται τα αντικείμενα κάθε activity και θα δημιουργούν την λογική του. Γι’ αυτό τον λόγο δημιουργήσαμε μια κλάση GlobalMethods.

Θέλουμε επίσης, κάθε φορά που θα τρέχει ένα activity να εμφανίζει σωστά την κατάσταση των διακοπτών του για παράδειγμα. Αν δηλαδή το φως είναι αναμμένο, ο διακόπτης του φωτός όταν ανοίγουμε το activity για τον φωτισμό να είναι ενεργοποιημένος. Για να το πετύχουμε αυτό θέλουμε μεθόδους αρχικοποίησης των στοιχείων του activity. Για τους διακόπτες είναι η εξής:

**public class** GlobalMethods {  
**public static void** initializeSwitch(String deviceURI,Switch aSwitch){

**if**(HttpRequests.*getRequest*(deviceURI).equals(Constant.***DEVICE\_ON***))  
 aSwitch.setChecked(**true**);  
 **else** aSwitch.setChecked(**false**);  
 }

Για την αρχικοποίηση των πεδίων χρονοπρογραμματισμού και ενεργοποίησης/απενεργοποίησης αισθητήρα θα υλοποιήσουμε την συνάρτηση initializeCheckBoxAndField. Η συνάρτηση αυτή δέχεται το URI της συσκευής, την επέκταση URI για χρονοπρογραμματισμό ή αλλαγή βάση αισθητήρα, το checkbox και το πεδίο πληκτρολόγησης. Αρχικά ανακτά από τον διακομιστή την αντίστοιχη τιμή. Έπειτα ελέγχει αν το πεδίο είναι πεδίο για χρονοπρογραμματισμό ή βασίζεται σε αλλαγή αισθητήρα. Ανάλογα το αποτέλεσμα, χρησιμοποιεί διαφορετική τιμή null.

**public static void** initializeCheckBoxAndField(String deviceURI, String functionURI, CheckBox checkbox, EditText edittext){  
 String request = HttpRequests.*getRequest*(deviceURI + functionURI);  
 String nullValue;  
 **if**(functionURI.equals(Constant.***DEVICE\_TIME\_ENABLE\_URI***) ||

functionURI.equals(Constant.***DEVICE\_TIME\_DISABLE\_URI***))  
 nullValue = Constant.***NO\_TIME\_VALUE***; **else** nullValue = Constant.***NO\_SENSOR\_VALUE***;

Αν η τιμή που έστειλε ο διακομιστής σαν απάντηση είναι τιμή Null, απενεργοποίησε το checkbox, χρωμάτισε το γκρι και άφησε το κείμενο κενό. Αλλιώς, ενεργοποίησε το, χρωμάτισε το μπλε και εμφάνισε την τιμή του διακομιστή.

**if**(request.equals(nullValue)){  
 checkbox.setChecked(**false**);  
 checkbox.setEnabled(**false**);

checkbox.setClickable(**false**);  
 checkbox.setTextColor(Color.*rgb*(117,118,120)); edittext.setText(Constant.***NO\_DISPLAY\_VALUE***);

}**else**{  
 checkbox.setChecked(**true**);  
 checkbox.setEnabled(**true**);

checkbox.setClickable(**true**);  
 edittext.setText(request);

checkbox.setTextColor(Color.*rgb*(67,78,121)); }  
}

Όσων αφορά την λειτουργικότητα των checkboxes και edittexts, θα χρειαστεί να υλοποιήσουμε ανάλογες μεθόδους. Μέσα σε αυτές τις μεθόδους θα υλοποιούμε ακροατές, που θα περιμένουν για κάποιο συγκεκριμένο γεγονός να συμβεί πάνω σε ένα στοιχείο. Με το που θα συμβεί αυτό το γεγονός, θα ανανεώνουν τιμές του διακομιστή.

Για τους διακόπτες, η μέθοδος θα δέχεται έναν διακόπτη και το URI της συσκευής, και θα θέτουμε έναν ακροατή πάνω στον διακόπτη. Αν ο χρήστης ανοίξει τον διακόπτη, ανανέωσε την τιμή του διακομιστή σε αληθής. Αν ο χρήστης κλείσει τον διακόπτη, ανανέωσε την τιμή του διακομιστή σε ψευδή.

**public static void** switchFunctionality(Switch aSwitch,

**final** String deviceURI){  
 aSwitch.setOnCheckedChangeListener(**new** CompoundButton.

OnCheckedChangeListener() {  
 @Override  
 **public void** onCheckedChanged(CompoundButton buttonView,

**boolean** isChecked) {  
 **if**(isChecked){  
 HttpRequests.*putRequest*(deviceURI,Constant.***DEVICE\_ON***);  
 }**else**{  
 HttpRequests.*putRequest*(deviceURI,Constant.***DEVICE\_OFF***);  
 }  
 }  
 });  
}

Για τα checkboxes, η μέθοδος θα δέχεται ένα checkbox, το edittext που του αντιστοιχεί, το URI της συσκευής και το URI της λειτουργίας. Θέτουμε έναν ακροατή πάνω στο checkbox. Αν ο χρήστης πατήσει να ενεργοποιήσει το checkbox, στέλνουμε την τιμή του edittext στον διακομιστή. Αν ο χρήστης πατήσει να απενεργοποιήσει το checkbox, ανανέωσε την τιμή του διακομιστή σε null τιμή χρονοπρογραμματισμού ή null τιμή αισθητήρα ανάλογα με την λειτουργικότητα του checkbox. Απενεργοποίησε επίσης το checkbox και άλλαξε το χρώμα του σε γκρι.

**public static void** checkBoxFunctionality(**final** CheckBox checkbox, **final** EditText edittext, **final** String deviceURI, **final** String functionURI){

checkbox.setOnCheckedChangeListener(**new** CompoundButton.

OnCheckedChangeListener() {  
 @Override  
 **public void** onCheckedChanged(CompoundButton buttonView,

**boolean** isChecked) {

**if**(isChecked){  
 HttpRequests.*putRequest*(deviceURI + functionURI,

edittext.getText().toString());  
 }**else**{  
 **if**(functionURI.equals(Constant.***DEVICE\_TIME\_ENABLE\_URI***) ||

functionURI.equals(Constant.***DEVICE\_TIME\_DISABLE\_URI***)){

HttpRequests.*putRequest*(deviceURI + functionURI,

Constant.***NO\_TIME\_VALUE***);  
 }**else if**(functionURI.equals(Constant.***DEVICE\_SENSOR\_ENABLE\_URI***)

|| functionURI.equals(Constant.***DEVICE\_SENSOR\_DISABLE\_URI***)){  
 HttpRequests.*putRequest*(deviceURI + functionURI,

Constant.***NO\_SENSOR\_VALUE***);  
 }  
 checkbox.setEnabled(**false**);  
 checkbox.setClickable(**false**);  
 checkbox.setTextColor(Color.*rgb*(117,118,120));  
 }  
 }  
 });

}

Αυτή η μέθοδος θέτει την λειτουργικότητα του checkbox. Όταν ο χρήστης πατάει το checkbox, στέλνουμε ότι υπάρχει στο edittext. Θέλουμε όμως να στέλνουμε σωστά δεδομένα στον διακομιστή, δηλαδή την ώρα για παράδειγμα θέλουμε να την στέλνουμε στην σωστή μορφή. Επίσης εδώ όταν ο χρήστης πατήσει να απενεργοποιήσει το checkbox, δεν υπάρχει λειτουργία για την ενεργοποίηση του. Αυτές οι δυο λειτουργίες υλοποιούνται στην μέθοδο editTextFunctionality.

Σε αυτή την συνάρτηση δεχόμαστε ένα edittext, ένα checkbox και μια κανονική έκφραση ανάλογα με την λειτουργικότητα του checkbox. Αν το checkbox είναι για χρονοπρογραμματισμό, θα επιτρέπει μόνο εκφράσεις σε μορφή ώρας στο πεδίο. Αν το checkbox είναι με αλλαγή βάση σε αισθητήρα, θα δέχεται μόνο ποσοστό στο πεδίο. Αν ο χρήστης εισήγαγε σωστά την μορφή που επιτρέπεται, το checkbox ενεργοποιείται και γίνεται ξανά μπλε, αλλιώς γίνεται γκρι και απενεργοποιείται.

**public static void** editTextFunctionality(**final** EditText edittext,

**final** CheckBox checkbox, **final** String aPattern){  
 edittext.setOnEditorActionListener(**new**

TextView.OnEditorActionListener(){  
 **public boolean** onEditorAction(TextView exampleView,

**int** actionId, KeyEvent event) {  
 **if** (actionId == EditorInfo.***IME\_ACTION\_DONE***) {  
 Boolean matches;  
 **try** {  
 Pattern pattern = Pattern.*compile*(aPattern);  
 Matcher matcher = pattern.matcher(edittext.

getText().toString());  
 matches = matcher.matches();  
 } **catch** (RuntimeException e) {  
 matches = **false**;  
 }  
 **if**(matches){  
 checkbox.setEnabled(**true**);  
 checkbox.setClickable(**true**);  
 checkbox.setTextColor(Color.*rgb*(67,78,121));  
 }**else**{  
 checkbox.setEnabled(**false**);  
 checkbox.setClickable(**false**);  
 checkbox.setTextColor(Color.*rgb*(117,118,120));  
 }  
 }  
 **return false**;  
 }  
 });  
}

## 5.3 Υλοποίηση των Activities

Μιας και έχουμε υλοποιήσει πια τις βοηθητικές μεθόδους μπορούμε να αρχίσουμε να υλοποιούμε το κυρίως πρόγραμμα, δηλαδή τις κλάσεις Activity. Κάθε activity είναι δεμένη με το layout που σχεδιάσαμε στο 5.1.

Έτσι, αρχίζοντας με το Home Screen, βλέπουμε πως έχουμε 5 κουμπιά. Μπορούμε να υλοποιήσουμε μια μέθοδο που θα δίνει σε ένα κουμπί την σωστή λειτουργικότητα και να την επαναλάβουμε για κάθε κουμπί. Κάθε φορά που πατάμε ένα κουμπί, πηγαίνουμε σε μια καινούρια activity ανάλογα με ποιο κουμπί πατήσαμε.

Δεχόμαστε ένα κουμπί και μία κλάση η οποία θα είναι το καινούριο activity που θα μπει στο προσκήνιο της εφαρμογής. Δημιουργούμε έναν ακροατή και κάθε φορά που ο χρήστης θα πατάει πάνω στο κουμπί μας, θα ανοίγει το αντίστοιχο activity.

**private void** runActivityButton(Button aButton, **final** Class aClass){  
 **assert** aButton != **null**;  
 aButton.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 Intent anIntent = **new** Intent(HomeScreen.**this**, aClass);  
 HomeScreen.**this**.startActivity(anIntent);  
 }  
 });  
}

Είχαμε αναφέρει ότι στην αρχική οθόνη θα εμφανίζουμε τιμές από όλους τους αισθητήρες. Δηλώνουμε τα πεδία για τις τιμές αυτές. Δηλώνουμε επίσης ένα αντικείμενο ScheduledExecutorService το οποίο θα εξηγήσουμε αργότερα.

**public class** HomeScreen **extends** AppCompatActivity {  
  
 **private** ScheduledExecutorService **scheduledExecutorService**;  
 **private** TextView **alarmStatusText**;  
 **private** TextView **homeStatusText**;  
 **private** TextView **lightLevelText**;  
 **private** TextView **hotWaterText**;  
 **private** TextView **temperatureText**;

Η μέθοδος onCreate τρέχει όταν δημιουργείται το activity μας. Στις πρώτες σειρές θέτουμε σαν layout το αρχείο xml που σχεδιάσαμε στο 5.1

**protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_home\_screen***);

Στο επόμενο σημείο θέτουμε την πολιτική μας έτσι ώστε να μην υπάρχουν περιορισμοί για τα νήματα που δημιουργούμε

**if** (android.os.Build.VERSION.***SDK\_INT*** > 9) {  
 StrictMode.ThreadPolicy policy = **new** StrictMode.ThreadPolicy.

Builder().permitAll().build();  
 StrictMode.*setThreadPolicy*(policy);  
}

Δηλώνουμε τα κουμπιά και συνδέουμε τα κουμπιά αλλά και τα πεδία που δηλώσαμε στην αρχή με τα αντίστοιχα κουμπιά και πεδία που σχεδιάσαμε στο layout του activity μας.

Button alarmButton = (Button) findViewById(R.id.***hsAlarmButton***);  
Button lightsButton = (Button) findViewById(R.id.***hsLightsButton***);  
Button boilerButton = (Button) findViewById(R.id.***hsBoilerButton***);  
Button airConditionButton = (Button) findViewById(R.id.***hsAirConditionButton***);  
Button blindsButton = (Button) findViewById(R.id.***hsBlindsButton***);  
  
**alarmStatusText** = (TextView) findViewById(R.id.***hsAlarmStatusText***);  
**homeStatusText** = (TextView) findViewById(R.id.***hsHomeStatusText***);  
**lightLevelText** = (TextView) findViewById(R.id.***hsLightLevelText***);  
**hotWaterText** = (TextView) findViewById(R.id.***hsHotWaterText***);  
**temperatureText** = (TextView) findViewById(R.id.***hsTemperatureText***);

Τέλος δημιουργούμε τους ακροατές για τα κουμπιά μας.

runActivityButton(alarmButton,AlarmScreen.**class**);  
 runActivityButton(lightsButton,LightsScreen.**class**);  
 runActivityButton(boilerButton,BoilerScreen.**class**);  
 runActivityButton(airConditionButton,AirConditionScreen.**class**);  
 runActivityButton(blindsButton,BlindsScreen.**class**);  
}

Τώρα, όσων αφορά τις τιμές των αισθητήρων που εμφανίζουμε, θέλουμε να ανανεώνονται τακτικά. Θέλουμε επίσης να σταματούν να ανανεώνονται όταν ανοίγει άλλο activity και να ξαναρχίζουν να ανανεώνονται όταν επιστρέφουμε στο Home Screen. Για να το υλοποιήσουμε αυτό θα δημιουργήσουμε ένα καινούριο νήμα το οποίο σε κάθε ανανέωση θα αλλάζει τις τιμές των αισθητήρων με τις καινούριες τιμές που θα ανακτά από τον διακομιστή.

Θα αρχίζουμε το νήμα μέσα στην συνάρτηση onResume, η οποία τρέχει μια φορά μετά την OnCreate στην αρχή και μετά κάθε φορά που επιστρέφουμε στο activity μας. Θα σταματούμε το νήμα μας στην onPause, η οποία τρέχει όταν φύγουμε από το Home Screen και περάσουμε σε άλλο activity.

Για να εκτελείται το νήμα μας κάθε 10 δευτερόλεπτα και να ανανεώνει τις τιμές των αισθητήρων θα χρησιμοποιήσουμε το ScheduledExecutorService.

**protected void** onResume() {  
 **super**.onResume();  
**scheduledExecutorService** = Executors.*newSingleThreadScheduledExecutor*();  
 **scheduledExecutorService**.scheduleAtFixedRate(**new** Runnable(){  
  
 @Override  
 **public void** run() {  
 HomeScreen.**this**.runOnUiThread(**new** Runnable(){ @Override  
 **public void** run() {

Για την θερμοκρασία, το επίπεδο φωτός και το επίπεδο ζεστού νερού στον θερμοσίφωνα, η διαδικασία είναι απλή

String tempSensor = HttpRequests.*getRequest*(Constant.***SENSOR\_TEMPERATURE\_URI***);  
**temperatureText**.setText(tempSensor + **"° C"**);  
  
String hotWSensor = HttpRequests.*getRequest*(Constant.***SENSOR\_HOT\_WATER\_URI***);  
**hotWaterText**.setText(hotWSensor + **" %"**);  
  
String lightSensor = HttpRequests.*getRequest*(Constant.***SENSOR\_LIGHT\_LEVEL\_URI***);  
**lightLevelText**.setText(lightSensor + **" %"**);

Εμφανίζουμε την κατάσταση του συναγερμού. Αν ο συναγερμός είναι ενεργοποιημένος, εμφανίζουμε το κατάλληλο μήνυμα με πράσινο χρώμα. Αν είναι απενεργοποιημένος εμφανίσουμε το αντίστοιχο μήνυμα με κόκκινο χρώμα. Αν υπήρξε κάποιο σφάλμα, εμφανίζουμε το αντίστοιχο μήνυμα με γκρι χρώμα.

String alarmStatus = HttpRequests.*getRequest*(Constant.***DEVICE\_ALARM\_URI***);  
**switch** (alarmStatus) {  
 **case** Constant.***DEVICE\_ON***:  
 **alarmStatusText**.setText(R.string.***alarmEnabledMessage***);  
 **alarmStatusText**.setTextColor(Color.*rgb*(102, 153, 50)); **break**;  
 **case** Constant.***DEVICE\_OFF***:  
 **alarmStatusText**.setText(R.string.***alarmDisabledMessage***); **alarmStatusText**.setTextColor(Color.*rgb*(204, 0, 0));  
 **break**;  
 **default**:  
 **alarmStatusText**.setText(R.string.***alarmNoServiceMessage***);  
 **alarmStatusText**.setTextColor(Color.*rgb*(117, 118, 120)); **break**;  
}

Τέλος πρέπει να εμφανίσουμε την κατάσταση της κατοικίας, αν υπάρχει ένδειξη παραβίασης ή όχι. Για να το επιτύχουμε αυτό ελέγχουμε την κατάσταση του συναγερμού. Αν ο συναγερμός είναι απενεργοποιημένος, στην κατάσταση της κατοικίας αναγράφουμε το κατάλληλο μήνυμα σε πορτοκαλί χρώμα για να καταλάβει ο χρήστης ότι δεν είναι ενεργή η συσκευή. Αν ο συναγερμός είναι ενεργοποιημένος και υπάρχει παραβίαση, ενημερώνουμε με κατάλληλο μήνυμα σε κόκκινο χρώμα, αλλιώς αν δεν υπάρχει παραβίαση χρησιμοποιούμε πράσινο χρώμα.

Τέλος κλείνουμε το νήμα μας και το προγραμματίζουμε να τρέχει κάθε 10 δευτερόλεπτα.

String homeStatus=HttpRequests.*getRequest*(Constant.***SENSOR\_BREAK\_IN\_URI***);  
**switch** (alarmStatus) {  
 **case** Constant.***DEVICE\_OFF***:  
 **homeStatusText**.setText(R.string.***disabledAlarmMessage***);  
 **homeStatusText**.setTextColor(Color.*rgb*(255, 136, 0));  
 **break**;  
 **case** Constant.***DEVICE\_ON***:  
 **if** (homeStatus.equals(Constant.***SENSOR\_ON***)) {  
 **homeStatusText**.setText(R.string.***alarmMotionDetectionMessage***);  
 **homeStatusText**.setTextColor(Color.*rgb*(204, 0, 0));  
 } **else** {  
 **homeStatusText**.setText(R.string.***alarmMotionDetectionMessage***);  
 **homeStatusText**.setTextColor(Color.*rgb*(102, 153, 50));  
 }  
 **break**;  
 **default**:  
 **homeStatusText**.setText(R.string.***alarmNoServiceMessage***);  
 **homeStatusText**.setTextColor(Color.*rgb*(117, 118, 120));  
 **break**;  
}

}  
 });  
 }  
 },10,10, TimeUnit.***SECONDS***);  
}

Παγώνουμε επίσης το νήμα μας στην onPause. Το νήμα θα ξαναδημιουργηθεί όταν επιστρέψουμε στο συγκεκριμένο activity.

**protected void** onPause() {  
 **super**.onPause();  
  
 **scheduledExecutorService**.shutdownNow();  
  
}

Για τα υπόλοιπα activities η διαδικασία υλοποίησης τους είναι παρόμοια. Για να μην επαναλαμβανόμαστε θα παρουσιάσουμε την δομή μιας activity και σημεία διάφορων activity που διαφέρουν. Για παράδειγμα το activity για τον θερμοσίφωνα θα περιέχει κλήσεις στην κλάση GlobalMethods για την υλοποίηση των λειτουργιών της.

Δηλώνουμε και αντιστοιχούμε τα στοιχεία του activity με το αντίστοιχο layout.

Switch boilerSwitch = (Switch) findViewById(R.id.***bosBoilerSwitch***);  
CheckBox turnOnTimeBox = (CheckBox) findViewById(R.id.***bosTurnOnTimeBox***);  
EditText turnOnTimeField = (EditText) findViewById(R.id.***bosTurnOnTimeField***);  
CheckBox turnOffTimeBox = (CheckBox) findViewById(R.id.***bosTurnOffTimeBox***);  
EditText turnOffTimeField = (EditText) findViewById(R.id.***bosTurnOffTimeField***);  
CheckBox turnOnSensorBox = (CheckBox) findViewById(R.id.***bosTurnOnSensorBox***);  
EditText turnOnSensorField = (EditText) findViewById(R.id.***bosTurnOnSensorField***);  
CheckBox turnOffSensorBox = (CheckBox) findViewById(R.id.***bosTurnOffSensorBox***);  
EditText turnOffSensorField = (EditText) findViewById(R.id.***bosTurnOffSensorField***);  
**hotWaterText** = (TextView) findViewById(R.id.***bosHotWaterText***);

Αρχικοποιούμε το activity μας.

GlobalMethods.*initializeSwitch*(Constant.***DEVICE\_BOILER\_URI***, boilerSwitch);

GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_BOILER\_URI***,

Constant.***DEVICE\_TIME\_ENABLE\_URI***, turnOnTimeBox, turnOnTimeField);

GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_BOILER\_URI***,

Constant.***DEVICE\_TIME\_DISABLE\_URI***, turnOffTimeBox, turnOffTimeField);

GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_BOILER\_URI***,

Constant.***DEVICE\_SENSOR\_ENABLE\_URI***, turnOnSensorBox, turnOnSensorField);

GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_BOILER\_URI***,Constant.***DEVICE\_SENSOR\_DISABLE\_URI***, turnOffSensorBox, turnOffSensorField);

Προσθέτουμε την σωστή λειτουργικότητα στα στοιχεία μας

GlobalMethods.*switchFunctionality*(boilerSwitch,

Constant.***DEVICE\_BOILER\_URI***);  
GlobalMethods.*checkBoxFunctionality*(turnOnTimeBox, turnOnTimeField,

Constant.***DEVICE\_BOILER\_URI***,

Constant.***DEVICE\_TIME\_ENABLE\_URI***);  
GlobalMethods.*editTextFunctionality*(turnOnTimeField, turnOnTimeBox,

Constant.***TIME\_PATTERN***);  
GlobalMethods.*checkBoxFunctionality*(turnOffTimeBox, turnOffTimeField,

Constant.***DEVICE\_BOILER\_URI***,

Constant.***DEVICE\_TIME\_DISABLE\_URI***);  
GlobalMethods.*editTextFunctionality*(turnOffTimeField, turnOffTimeBox,

Constant.***TIME\_PATTERN***);  
GlobalMethods.*checkBoxFunctionality*(turnOnSensorBox, turnOnSensorField,

Constant.***DEVICE\_BOILER\_URI***,

Constant.***DEVICE\_SENSOR\_ENABLE\_URI***);  
GlobalMethods.*editTextFunctionality*(turnOnSensorField, turnOnSensorBox,

Constant.***PERCENTAGE\_PATTERN***);  
GlobalMethods.*checkBoxFunctionality*(turnOffSensorBox, turnOffSensorField,Constant.***DEVICE\_BOILER\_URI***,Constant.***DEVICE\_SENSOR\_DISABLE\_URI***);  
GlobalMethods.*editTextFunctionality*(turnOffSensorField, turnOffSensorBox,Constant.***PERCENTAGE\_PATTERN***);

Με τον ίδιο τρόπο υλοποιούμε όλα τα άλλα activities, με την μόνη διαφορά ότι έχουμε διαφορετικές μεταβλητές και στοιχεία. Επίσης τους αισθητήρες που εμφανίζουμε σε κάθε activity του υλοποιούμε και αυτούς με νήματα. Τα κουμπιά επίσης, τα υλοποιούμε όπως τα κουμπιά στην activity Home Screen. Το μόνο διαφορετικό κομμάτι που αξίζει να εξηγήσουμε είναι το πεδίο κωδικού στην Alarm Activity και η λειτουργικότητα των checkbox στο Lights Activity.

Το πεδίο του κωδικού στο Alarm Activity ενεργοποιεί τον διακόπτη για τον συναγερμό όταν εισάγει ο χρήστης τον σωστό κωδικό. Επίσης, αν ο χρήστης εισάγει τον σωστό κωδικό το κείμενο γίνεται πράσινο. Αντίστοιχα αν ο χρήστης πληκτρολογήσει λάθος κωδικό, το κείμενο γίνεται κόκκινο και ο διακόπτης του συναγερμού παραμένει ανενεργός.

**passwordField**.setOnEditorActionListener(**new** TextView.

OnEditorActionListener(){  
 **public boolean** onEditorAction(TextView exampleView, **int** actionId,

KeyEvent event) {  
 **if** (actionId == EditorInfo.***IME\_ACTION\_DONE***) {  
 **if**((**passwordField**.getText().toString()).

equals(Constant.***USER\_PASSWORD***)){  
 **alarmSwitch**.setClickable(**true**);  
 **alarmLabel**.setTextColor(Color.*rgb*(67,78,121));  
 **passwordLabel**.setTextColor(Color.*rgb*(102,153,50));  
 }  
 **else**{  
 **alarmSwitch**.setClickable(**false**);  
 **alarmLabel**.setTextColor(Color.*rgb*(117,118,120));  
 **passwordLabel**.setTextColor(Color.*rgb*(204,0,0));  
 }  
 }  
 **return false**;  
 }  
});

Η μόνη διαφορά στην λειτουργικότητα των checkboxes στο Lights Activity είναι ότι όταν ο χρήστης ενεργοποιεί το checkbox για τον χρονοπρογραμματισμό ή την αλλαγή βάση αισθητήρα, πρέπει να ανανεώνουμε τις τιμές στον διακομιστή και για τα δυο φώτα. Αυτό το υλοποιούμε ως εξής:

**private void** checkBoxDualFunctionality(**final** CheckBox checkbox,

**final** EditText edittext, **final** String functionURI){  
 checkbox.setOnCheckedChangeListener(**new** CompoundButton.

OnCheckedChangeListener() {  
 @Override  
 **public void** onCheckedChanged(CompoundButton buttonView,

**boolean** isChecked) {  
 **if**(isChecked){  
 HttpRequests.*putRequest*(Constant.***DEVICE\_LIGHT\_1\_URI*** +

functionURI,edittext.getText().toString());  
 HttpRequests.*putRequest*(Constant.***DEVICE\_LIGHT\_2\_URI*** +

functionURI,edittext.getText().toString());  
 }**else**{  
 **if**(functionURI.equals(Constant.***DEVICE\_TIME\_ENABLE\_URI***)

|| functionURI.equals

(Constant.***DEVICE\_TIME\_DISABLE\_URI***)){  
 HttpRequests.*putRequest*(Constant.***DEVICE\_LIGHT\_1\_URI***

+ functionURI,Constant.***NO\_TIME\_VALUE***);  
 HttpRequests.*putRequest*(Constant.***DEVICE\_LIGHT\_2\_URI***

+ functionURI,Constant.***NO\_TIME\_VALUE***);  
 }**else if**(functionURI.equals(Constant.

***DEVICE\_SENSOR\_ENABLE\_URI***) || functionURI.equals

(Constant.***DEVICE\_SENSOR\_DISABLE\_URI***)){  
 HttpRequests.*putRequest*(Constant.***DEVICE\_LIGHT\_1\_URI***

+ functionURI,Constant.***NO\_SENSOR\_VALUE***);  
 HttpRequests.*putRequest*(Constant.***DEVICE\_LIGHT\_2\_URI***

+ functionURI,Constant.***NO\_SENSOR\_VALUE***);  
 }  
 checkbox.setEnabled(**false**);  
 checkbox.setClickable(**false**);  
 checkbox.setTextColor(Color.*rgb*(117,118,120));  
 }  
 }  
 });  
}

# Συμπεράσματα

Στην παρούσα πτυχιακή εργασία δημιουργήσαμε ένα σύστημα αυτοματισμού κατοικίας χρησιμοποιώντας το ανοιχτό υλικό Arduino για την προσομοίωση των συσκευών της κατοικίας. Υλοποιήσαμε επίσης το λογισμικό για το Arduino, την υπηρεσία ιστού στην οποία συνδέεται και την εφαρμογή Android για διαχείριση της κατοικίας.

Ανακαλύψαμε ότι είναι δυνατή η δημιουργία ενός τέτοιου συστήματος και λόγω του χαμηλού κόστους για την αγορά των απαραίτητων εξαρτημάτων, αποτελεί μια καλή και οικονομική λύση. Αποδείξαμε επίσης ότι δεν χρειάζεται να βασίζεται ο καταναλωτής σε εταιρίες για τον αυτοματισμό της οικίας του.

Φυσικά, το σύστημα που περιγράφηκε είναι μια προσομοίωση ενός συστήματος έξυπνου σπιτιού και μπορεί να δεχθεί βελτιώσεις και επεκτάσεις. Κάποιες από αυτές μπορεί να είναι οι εξής:

* Υλοποίηση των άλλων δυο βασικών μεθόδων HTTP για δυνατότητες προσθήκης επιπλέων συσκευών και αισθητήρων καθώς και δυνατότητες διαγραφής των υπαρχουσών.
* Υλοποίηση και σύνδεση του συστήματος με βάση δεδομένων για μεγαλύτερη ελαστικότητα και διαφάνεια
* Υλοποίηση καλύτερης ασφάλειας με σωστές διαδικασίες ασφαλής αποστολής και αποδοχής αιτημάτων διακομιστή και επιβεβαίωση των χρηστών.
* Επέκταση του συστήματος με υλοποίηση κώδικα για πραγματικές συσκευές και όχι εικονικές χρησιμοποιώντας τα εξαρτήματα του Arduino
* Επέκταση της υλοποίησης του διακομιστή έτσι ώστε να υποστηρίζει περισσότερα από ένα έξυπνα σπίτια.
* Καλύτερη διαχείριση των πόρων από την εφαρμογή Android, έτσι ώστε να χρησιμοποιεί βέλτιστα την ενέργεια της κινητής συσκευής και την σύνδεση με το Internet.
* Βελτιστοποίηση του προγραμματισμού για να γίνει πιο αποδοτικός όσων αφορά την υπηρεσία ιστού αλλά και το Arduino.
* Επέκταση της υλοποίησης της εφαρμογής Android ώστε να προσφέρει φωνητικό έλεγχο για τις συσκευές της έξυπνης κατοικίας.
* Επέκταση του συστήματος έτσι ώστε εκτός από κινητή συσκευή, να μπορεί να ελέγχεται και από σταθερούς υπολογιστές ή smart TVs
* Επέκταση του συστήματος έτσι ώστε να απαντά στις εντολές του χρήστη έτσι ώστε να γίνει πιο άμεσο και πιο “φουτουριστικό” για τον χρήστη

# Αναφορές

Wikipedia. (2016). X10 (industry standard). [online] Available at: https://en.wikipedia.org/wiki/X10\_%28industry\_standard%29 [Accessed 11 Jun. 2016].

Smarthome.com. (2016). What is X10? | X10 Home Automation Products | X10 Systems. [online] Available at: http://www.smarthome.com/sc-what-is-x10-home-automation [Accessed 11 Jun. 2016].

Wikipedia. (2016). Universal powerline bus. [online] Available at: https://en.wikipedia.org/wiki/Universal\_powerline\_bus [Accessed 11 Jun. 2016].

Smarthome.com. (2016). What is UPB? | UPB Home Automation | Home Automation Systems. [online] Available at: http://www.smarthome.com/sc-what-is-upb-home-automation [Accessed 11 Jun. 2016].

Wikipedia. (2016). Insteon. [online] Available at: https://en.wikipedia.org/wiki/Insteon [Accessed 11 Jun. 2016].

Insteon. (2016). Home. [online] Available at: http://www.insteon.com/ [Accessed 11 Jun. 2016].

Wikipedia. (2016). *Z-Wave*. [online] Available at: https://en.wikipedia.org/wiki/Z-Wave [Accessed 11 Jun. 2016].

Z-Wave Home control. (2016). *Z-Wave Home control | Z-Wave Home control*. [online] Available at: http://www.z-wave.com/ [Accessed 11 Jun. 2016].

Zigbee.org. (2016). *What is ZigBee? | The ZigBee Alliance*. [online] Available at: http://www.zigbee.org/what-is-zigbee/ [Accessed 11 Jun. 2016].

Smarthome.com. (2016). *What is Zigbee? | Zigbee Home Automation | Home Automation Systems*. [online] Available at: http://www.smarthome.com/sc-what-is-zigbee-home-automation [Accessed 11 Jun. 2016].

Wikipedia. (2016). *Home automation*. [online] Available at: https://en.wikipedia.org/wiki/Home\_automation [Accessed 11 Jun. 2016].

TopTenREVIEWS. (2016). *Top 10 Home Automation Systems of 2016*. [online] Available at: http://home-automation-systems-review.toptenreviews.com/ [Accessed 11 Jun. 2016].

Wikipedia. (2016). *HomeSeer*. [online] Available at: https://en.wikipedia.org/wiki/HomeSeer#2014 [Accessed 11 Jun. 2016].

HomeSeer Home Automation Systems. (2016). *HomeSeer Home Automation Systems*. [online] Available at: http://www.homeseer.com/ [Accessed 11 Jun. 2016].

Wikipedia. (2016). *Control4*. [online] Available at: https://en.wikipedia.org/wiki/Control4 [Accessed 11 Jun. 2016].

Control4.com. (2016). *Home Automation and Smart Home Systems | Control4*. [online] Available at: http://www.control4.com/ [Accessed 11 Jun. 2016].

Crestron.com. (2016). *Control Systems for Home Automation, Campus &amp; Building Control by Crestron Electronics*. [online] Available at: https://www.crestron.com/ [Accessed 11 Jun. 2016].

Wikipedia. (2016). *Web service*. [online] Available at: https://en.wikipedia.org/wiki/Web\_service [Accessed 11 Jun. 2016].

Mandliya, A. (2016). *Web Service tutorial - Java tutorial for beginners*. [online] Java2blog.com. Available at: http://www.java2blog.com/2013/03/web-service-tutorial.html [Accessed 11 Jun. 2016].

Docs.oracle.com. (2016). *What Are RESTful Web Services? - The Java EE 6 Tutorial*. [online] Available at: http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html [Accessed 11 Jun. 2016].

Mandliya, A. (2016). *RESTful web service tutorial - Java tutorial for beginners*. [online] Java2blog.com. Available at: http://www.java2blog.com/2013/04/restful-web-service-tutorial.html [Accessed 11 Jun. 2016].

Arduino.cc. (2016). *Arduino - Home*. [online] Available at: https://www.arduino.cc/ [Accessed 11 Jun. 2016].

Arduino.cc. (2016). *Arduino - ArduinoEthernetShield*. [online] Available at: https://www.arduino.cc/en/Main/ArduinoEthernetShield [Accessed 11 Jun. 2016].

Wikipedia. (2016). *Arduino*. [online] Available at: https://en.wikipedia.org/wiki/Arduino [Accessed 11 Jun. 2016].

Arduino.cc. (2016). *Arduino - ArduinoBoardUno*. [online] Available at: https://www.arduino.cc/en/Main/ArduinoBoardUno [Accessed 11 Jun. 2016].

Wikipedia. (2016). *Android (operating system)*. [online] Available at: https://en.wikipedia.org/wiki/Android\_%28operating\_system%29 [Accessed 11 Jun. 2016].

Developer.android.com. (2016). *Download Android Studio and SDK Tools | Android Studio*. [online] Available at: https://developer.android.com/studio/index.html [Accessed 11 Jun. 2016].

# Παράρτημα

**Arduino**

#include <SPI.h>

#include <Ethernet.h>

#include <Servo.h>

//MAC and IP addresses initialization

byte mac[] = { 0x90,0xA2,0xDA, 0x0D, 0xD5,0x49 };

IPAddress restServer(83,212,118,164 );

IPAddress ip(192,168,1,177);

//Variables taken from the Server

boolean serverLight1;

boolean serverLight2;

boolean serverBoiler;

boolean serverAirCondition;

boolean serverBlinds;

boolean serverAlarm;

//The client that communicates with the REST server

EthernetClient client;

//Pin and Component matching

Servo boiler;

const int roomLight1 = A4;

const int roomLight2 = A2;

const int alarmGreenLight = A1;

const int alarmRedLight = A0;

const int boilerLight = 5;

const int blindsSimLight = 6;

const int alarmBeeper = 7;

const int alarmButton = 8;

const int airCondition = 9;

const int boilerServo = 3;

const int roomLightLevel = A5;

const int roomTempLevel = A3;

//Setting constants for devices, sensors and values

const String DEVICE\_LIGHT1 = "device/1";

const String DEVICE\_LIGHT2 = "device/2";

const String DEVICE\_BOILER = "device/3";

const String DEVICE\_AIRCONDITION = "device/4";

const String DEVICE\_BLINDS = "device/5";

const String DEVICE\_ALARM = "device/6";

const String SENSOR\_TEMPERATURE = "sensor/1";

const String SENSOR\_HOTWATER = "sensor/2";

const String SENSOR\_LIGHTLEVEL = "sensor/3";

const String SENSOR\_BREAKIN = "sensor/4";

const String DEVICE\_ON = "true";

const String DEVICE\_OFF = "false";

const String SENSOR\_ON = "1";

const String SENSOR\_OFF = "0";

//Sensor values

int boilerPercentage = 0;

int temperature;

int lightLevel;

int alarmTriggered = 0;

//Other variables

float tempTemp;

int boilerCounter = 0;

int blindsState = 0;

unsigned long time;

void setup() {

//Setting I/O for PINS

pinMode(roomLight1,OUTPUT);

pinMode(roomLight2,OUTPUT);

pinMode(alarmGreenLight,OUTPUT);

pinMode(alarmRedLight,OUTPUT);

pinMode(boilerLight,OUTPUT);

pinMode(blindsSimLight,OUTPUT);

pinMode(alarmButton,INPUT);

pinMode(airCondition,OUTPUT);

//Attaching and initializing boiler

boiler.attach(boilerServo);

boiler.write(0);

//Starting Serial port connection with computer

Serial.begin(9600);

//Obtaining and printing IP address

Serial.println("Starting Ethernet Connection... ");

Ethernet.begin(mac,ip);

ip = Ethernet.localIP();

printIPAddress();

}

void loop() {

//Initiating Cycle Time Calculation

time = millis();

/\*Getting device status from REST server and converting them

to boolean values \*/

serverLight1 = toBoolean(getRequest(DEVICE\_LIGHT1));

serverLight2 = toBoolean(getRequest(DEVICE\_LIGHT2));

serverBoiler = toBoolean(getRequest(DEVICE\_BOILER));

serverAirCondition = toBoolean(getRequest(DEVICE\_AIRCONDITION));

serverBlinds = toBoolean(getRequest(DEVICE\_BLINDS));

serverAlarm = toBoolean(getRequest(DEVICE\_ALARM));

/\*Reading the sensor temperature, modifying it

(Voltage to Celcius) and uploading it to server \*/

temperature = analogRead(roomTempLevel);

tempTemp = (temperature/1024.0) \* 5.0;

tempTemp = (tempTemp - .5) \* 100;

temperature = (int)(tempTemp + .5);

putRequest(SENSOR\_TEMPERATURE,String(temperature));

/\*Reading the sensor light level, modifying it

(Voltage to Percentage) and uploading it to server \*/

lightLevel = analogRead(roomLightLevel);

lightLevel = map(lightLevel,0,1023,0,100);

putRequest(SENSOR\_LIGHTLEVEL,String(lightLevel));

//Turning on/off the lights

if(serverLight1)

digitalWrite(roomLight1,HIGH);

else

digitalWrite(roomLight1,LOW);

if(serverLight2)

digitalWrite(roomLight2,HIGH);

else

digitalWrite(roomLight2,LOW);

/\* Turning on/off boiler

When the boiler is turned on, the servo starts to rotate slowly

just like the indicator needle on a real boiler. When the boiler

reaches ~100% then it shuts down by itself. If the boiler has hot

water and shuts down the water begins to cool and the needle to

rotate backwards till there is no hot water left. \*/

if(serverBoiler){

digitalWrite(boilerLight,HIGH);

if(boilerCounter < 176){

boilerCounter += 8;

boiler.write(boilerCounter);

boilerPercentage = map(boilerCounter,0,179,0,100);

putRequest(SENSOR\_HOTWATER,String(boilerPercentage));

}

else{

putRequest(DEVICE\_BOILER,DEVICE\_OFF);

}

}

else{

digitalWrite(boilerLight,LOW);

if (boilerCounter > 0){

boilerCounter-= 8;

boiler.write(boilerCounter);

boilerPercentage = map(boilerCounter,0,179,0,100);

putRequest(SENSOR\_HOTWATER,String(boilerPercentage));

}

}

/\* Turning on/off the air condition

The air condition system consists of a fan. It can only provide

cold air so it's not technically an air condition but a smart fan \*/

if(serverAirCondition)

digitalWrite(airCondition,HIGH);

else

digitalWrite(airCondition,LOW);

/\* Raising and lowering the window blinds

The window blinds are simulated by an LED. When the blinds are opening

the LED turns on gradually till it reaches 100% brightness and the

blinds are completely open. When the blinds are closing the LED

shines gradually less till it is turned off. \*/

if(serverBlinds && blindsState == 0){

for(int i=0;i<256;i+=4){

analogWrite(blindsSimLight,i);

delay(20);

}

blindsState = 1;

}

else if (!serverBlinds && blindsState == 1){

for(int i=252;i>=0;i-=4){

analogWrite(blindsSimLight,i);

delay(20);

}

blindsState = 0;

}

/\* Turning on/off the alarm.

When the alarm is on the red LED is on. When the

alarm is off the green LED is on. \*/

if(serverAlarm){

digitalWrite(alarmRedLight,HIGH);

digitalWrite(alarmGreenLight,LOW);

}

else{

digitalWrite(alarmRedLight,LOW);

digitalWrite(alarmGreenLight,HIGH);

}

/\* Simulated break in button

When this button is pressed for a couple of seconds, if

the alarm is on, the beeper goes off \*/

if(digitalRead(alarmButton) == HIGH && serverAlarm){

putRequest(SENSOR\_BREAKIN,SENSOR\_ON);

}

alarmTriggered = getRequest(SENSOR\_BREAKIN).toInt();

if (alarmTriggered == 1)

tone(alarmBeeper,330);

else

noTone(alarmBeeper);

//Displaying Cycle Time

time = millis() - time;

Serial.println();

Serial.println("Cycle Time : " + String(time/1000.0));

}

//Function that prints the IP Address

void printIPAddress(){

Serial.print("My IP address: ");

for (byte thisByte = 0; thisByte < 4; thisByte++) {

Serial.print(ip[thisByte], DEC);

Serial.print(".");

}

Serial.println();

}

/\*Function that sends HTTP GET requests to the REST server.

INPUT: The URI for the request.

OUTPUT: The server response string. \*/

String getRequest(String theRequest){

Serial.print("Connecting : ");

while (!client.connect(restServer, 8080)) {

Serial.println("Connection failed");

Serial.print("Reconnecting : ");

}

client.println("GET /restServer/resources/" + theRequest);

Serial.print("GET /restServer/resources/" + theRequest + " : ");

delay(500);

String response = "";

while (client.available()) {

char c = client.read();

response.concat(c);

}

Serial.println(response);

client.flush();

if (!client.connected())

client.stop();

return response;

}

/\*Function that sends HTTP PUT requests to the REST server.

INPUT: The URI for the request.

The message to be sent. \*/

void putRequest(String theRequest, String message){

String serverIP = "";

Serial.print("Connecting : ");

while (!client.connect(restServer, 8080)) {

Serial.println("Connection failed");

Serial.print("Reconnecting : ");

}

String tempRequest = "";

for (byte thisByte = 0; thisByte < 4; thisByte++) {

serverIP += String(restServer[thisByte]);

if(thisByte <3)

serverIP +=(".");

}

tempRequest += "PUT /restServer/resources/" + theRequest + " HTTP/1.1\r\n";

tempRequest += "Host: " + serverIP + "\r\n";

tempRequest += "Content-Type: text/plain; charset=UTF-8\r\n";

tempRequest += "Content-Length: "+ String(message.length(),DEC)+ "\r\n";

tempRequest += "\r\n";

tempRequest += message;

client.println(tempRequest);

Serial.println("PUT /restServer/resources/" + theRequest);

delay(500);

client.flush();

client.stop();

}

//Function that returns the boolean value of a string

boolean toBoolean(String aString){

if(aString.equals("true")){

return true;

}else{

return false;

}

}

WEB SERVICE

**package** org.pachoumis.thesis.restServer.utility;

/\*

\* This Constant class contains the information about the server constants

\*/

**public** **final** **class** Constant {

**public** **static** **final** **int** ***NO\_SENSOR\_VALUE*** = -1;

**public** **static** **final** String ***NO\_TIME\_VALUE*** = "";

**public** **static** **final** **int** ***SENSOR\_OFF*** = 0;

**public** **static** **final** Boolean ***DEVICE\_OFF*** = **false**;

**public** **static** **final** Boolean ***DEVICE\_ON*** = **true**;

**public** **static** **final** **int** ***SENSOR\_OFFSET*** = 5;

**public** **static** **final** **int** ***DEVICE\_LIGHT\_1\_ID*** = 1;

**public** **static** **final** **int** ***DEVICE\_LIGHT\_2\_ID*** = 2;

**public** **static** **final** **int** ***DEVICE\_BOILER\_ID*** = 3;

**public** **static** **final** **int** ***DEVICE\_AIR\_CONDITION\_ID*** = 4;

**public** **static** **final** **int** ***DEVICE\_BLINDS\_ID*** = 5;

**public** **static** **final** **int** ***DEVICE\_ALARM\_ID*** = 6;

**public** **static** **final** **int** ***SENSOR\_TEMPERATURE\_ID*** = 1;

**public** **static** **final** **int** ***SENSOR\_HOT\_WATER\_ID*** = 2;

**public** **static** **final** **int** ***SENSOR\_LIGHT\_LEVEL\_ID*** = 3;

**public** **static** **final** **int** ***SENSOR\_BREAK\_IN\_ID*** = 4;

**private** Constant(){

**throw** **new** AssertionError();

}

}

**package** org.pachoumis.thesis.restServer.hardware;

**import** org.pachoumis.thesis.restServer.utility.Constant;

/\*

\* This Device class contains the information about a single component of

\* a smart home. Comes with necessary getters and setters. Provides

\* function for enabling/disabling the device at a certain time of the

\* day as well as enabling/disabling the device based on sensor values.

\*/

**public** **class** Device {

**private** String name;

**private** Boolean state;

**private** String enableTime; //Turn on based on time

**private** String disableTime; //Turn off based on time

**private** **int** enableSensorValue; //Turn on based on a sensor

**private** **int** disableSensorValue; //Turn off based on a sensor

**public** Device(String name){

**this**.name = name;

**this**.state = Constant.***DEVICE\_OFF***;

**this**.enableTime = Constant.***NO\_TIME\_VALUE***;

**this**.disableTime = Constant.***NO\_TIME\_VALUE***;

**this**.enableSensorValue = Constant.***NO\_SENSOR\_VALUE***;

**this**.disableSensorValue = Constant.***NO\_SENSOR\_VALUE***;

}

**public** String getName() {

**return** name;

}

**public** Boolean getState() {

**return** state;

}

**public** **void** setState(Boolean state) {

**this**.state = state;

}

**public** String getEnableTime() {

**return** enableTime;

}

**public** **void** setEnableTime(String enableTime) {

**this**.enableTime = enableTime;

}

**public** String getDisableTime() {

**return** disableTime;

}

**public** **void** setDisableTime(String disableTime) {

**this**.disableTime = disableTime;

}

**public** **int** getEnableSensorValue() {

**return** enableSensorValue;

}

**public** **void** setEnableSensorValue(**int** enableSensorValue) {

**this**.enableSensorValue = enableSensorValue;

}

**public** **int** getDisableSensorValue() {

**return** disableSensorValue;

}

**public** **void** setDisableSensorValue(**int** disableSensorValue) {

**this**.disableSensorValue = disableSensorValue;

}

}

**package** org.pachoumis.thesis.restServer.hardware;

**import** org.pachoumis.thesis.restServer.utility.Constant;

/\*

\* This Sensor class contains the information about a single sensor or mock

\* sensor of a smart home. Comes with necessary getters and setters.

\* Provides function for changing the value of the sensor.

\*/

**public** **class** Sensor {

**private** String name;

**private** **int** value;

**public** Sensor(String name){

**this**.name = name;

**this**.value = Constant.***SENSOR\_OFF***;

}

**public** String getName() {

**return** name;

}

**public** **int** getValue() {

**return** value;

}

**public** **void** setValue(**int** value) {

**this**.value = value;

}

}

**package** org.pachoumis.thesis.restServer.database;

**import** java.util.HashMap;

**import** java.util.Map;

**import** org.pachoumis.thesis.restServer.hardware.Device;

**import** org.pachoumis.thesis.restServer.hardware.Sensor;

**import** org.pachoumis.thesis.restServer.utility.Constant;

/\*

\* This Database class contains the information about all devices and

\* sensors of a smart home. The devices and sensors are initialized here.

\* Comes with necessary getters and setters. Creates 6 devices and 4

\* sensors and maps them with IDs.

\*/

**public** **class** DatabaseClass {

**private** **static** Map<Integer, Device> *devices* = **new** HashMap<>();

**private** **static** Map<Integer, Sensor> *sensors* = **new** HashMap<>();

**static**{

*devices*.put(Constant.***DEVICE\_LIGHT\_1\_ID***, **new** Device("Light1"));

*devices*.put(Constant.***DEVICE\_LIGHT\_2\_ID***, **new** Device("Light2"));

*devices*.put(Constant.***DEVICE\_BOILER\_ID***, **new** Device("Boiler"));

*devices*.put(Constant.***DEVICE\_AIR\_CONDITION\_ID***, **new** Device("AirCondition"));

*devices*.put(Constant.***DEVICE\_BLINDS\_ID***, **new** Device("Blinds"));

*devices*.put(Constant.***DEVICE\_ALARM\_ID***, **new** Device("Alarm"));

*sensors*.put(Constant.***SENSOR\_TEMPERATURE\_ID***, **new** Sensor("Temperature"));

*sensors*.put(Constant.***SENSOR\_HOT\_WATER\_ID***, **new** Sensor("Hot Water"));

*sensors*.put(Constant.***SENSOR\_LIGHT\_LEVEL\_ID***, **new** Sensor("Light Level"));

*sensors*.put(Constant.***SENSOR\_BREAK\_IN\_ID***, **new** Sensor("Break In"));

}

**public** **static** Map<Integer, Device> getDevices(){

**return** *devices*;

}

**public** **static** Map<Integer, Sensor> getSensors(){

**return** *sensors*;

}

}

**package** org.pachoumis.thesis.restServer.service;

**import** java.text.SimpleDateFormat;

**import** java.util.Calendar;

**import** java.util.Map;

**import** org.pachoumis.thesis.restServer.database.DatabaseClass;

**import** org.pachoumis.thesis.restServer.hardware.Device;

**import** org.pachoumis.thesis.restServer.hardware.Sensor;

**import** org.pachoumis.thesis.restServer.utility.Constant;

/\*

\* This Smart Home Service class implements the Web Service. Inside the

\* constructor is an internal time calculator that gets the current time

\* of the system and compares it with the enable/disable properties of all

\* devices. Also checks and enables/disables the devices based on Sensor

\* data.

\*/

**public** **class** SmartHomeService {

**private** Map<Integer,Device> devices = DatabaseClass.*getDevices*();

**private** Map<Integer,Sensor> sensors = DatabaseClass.*getSensors*();

Calendar cal = Calendar.*getInstance*();

SimpleDateFormat sdf = **new** SimpleDateFormat("HH:mm");

**public** SmartHomeService(){

// long startTime = System.currentTimeMillis();

String currentTime = sdf.format(cal.getTime());

// System.out.println("Time: " + currentTime);

**for**(**int** i=1;i<7;i++){

Device tempDevice = devices.get(i);

**if**(tempDevice.getEnableTime().equals(currentTime)){

//Compares system time with device enable time

tempDevice.setState(Constant.***DEVICE\_ON***);

}

**if**(tempDevice.getDisableTime().equals(currentTime)){

//Compares system time with device disable time

tempDevice.setState(Constant.***DEVICE\_OFF***);

}

**if**(tempDevice.getEnableSensorValue() != Constant.***NO\_SENSOR\_VALUE***){ //Checks for Sensor enable data

**switch** (i){

**case** Constant.***DEVICE\_LIGHT\_1\_ID***:

**case** Constant.***DEVICE\_LIGHT\_2\_ID***:

/\*In case the device is a light, check based on

light level enable sensor data \*/

**if**((sensors.get(Constant.***SENSOR\_LIGHT\_LEVEL\_ID***).getValue()

<= tempDevice.getEnableSensorValue())){

tempDevice.setState(Constant.***DEVICE\_ON***);

}

**break**;

**case** Constant.***DEVICE\_BOILER\_ID***:

/\*In case the device is a boiler, check based on

hot water enable sensor data \*/

**if**((sensors.get(Constant.***SENSOR\_HOT\_WATER\_ID***).getValue()

<= tempDevice.getEnableSensorValue())){

tempDevice.setState(Constant.***DEVICE\_ON***);

}

**break**;

**case** Constant.***DEVICE\_AIR\_CONDITION\_ID***:

/\*In case the device is an air condition, check based on

temperature enable sensor data \*/

**if**((sensors.get(Constant.***SENSOR\_TEMPERATURE\_ID***).getValue()

>= tempDevice.getEnableSensorValue())){

tempDevice.setState(Constant.***DEVICE\_ON***);

}

**break**;

**case** Constant.***DEVICE\_BLINDS\_ID***:

/\*In case the device is window blinds, check based on light

level enable sensor data \*/

**if**((sensors.get(Constant.***SENSOR\_LIGHT\_LEVEL\_ID***).getValue()

>= tempDevice.getEnableSensorValue())){

tempDevice.setState(Constant.***DEVICE\_ON***);

}

**break**;

}

}

**if**(tempDevice.getDisableSensorValue() != Constant.***NO\_SENSOR\_VALUE***){

//Checks for Sensor disable data

**switch** (i){

**case** Constant.***DEVICE\_LIGHT\_1\_ID***:

**case** Constant.***DEVICE\_LIGHT\_2\_ID***:

/\*In case the device is a light, check based on light level

disable sensor data \*/

**if**((sensors.get(Constant.***SENSOR\_LIGHT\_LEVEL\_ID***).getValue()

>= tempDevice.getDisableSensorValue())){

tempDevice.setState(Constant.***DEVICE\_OFF***);

}

**break**;

**case** Constant.***DEVICE\_BOILER\_ID***:

/\*In case the device is a boiler, check based on hot water

disable sensor data \*/

**if**((sensors.get(Constant.***SENSOR\_HOT\_WATER\_ID***).getValue()

>= tempDevice.getDisableSensorValue())){

tempDevice.setState(Constant.***DEVICE\_OFF***);

}

**break**;

**case** Constant.***DEVICE\_AIR\_CONDITION\_ID***:

/\*In case the device is an air condition, check based

on temperature disable sensor data \*/

**if**((sensors.get(Constant.***SENSOR\_TEMPERATURE\_ID***).getValue()

<= tempDevice.getDisableSensorValue())){

tempDevice.setState(Constant.***DEVICE\_OFF***);

}

**break**;

**case** Constant.***DEVICE\_BLINDS\_ID***:

/\*In case the device is window blinds, check based on

light level disable sensor data\*/

**if**((sensors.get(Constant.***SENSOR\_LIGHT\_LEVEL\_ID***).getValue()

<= tempDevice.getDisableSensorValue())){

tempDevice.setState(Constant.***DEVICE\_OFF***);

}

**break**;

}

}

devices.put(i, tempDevice);

/\*Place the new device with the correct parameters in the

place of the initial device \*/

}

// long endTime = System.currentTimeMillis();

// long totalTime = endTime - startTime;

// double totalTime2 = totalTime/1000.0;

// System.out.println("Total Cycle: " + totalTime);

}

//Accepts an ID and returns the device with that ID

**public** Device getDevice(**int** id){

**return** devices.get(id);

}

//Accepts an ID and a Boolean and updates the state of the device

**public** **void** updateDevice(**int** id, Boolean state){

Device tempDevice = devices.get(id);

tempDevice.setState(state);

devices.put(id, tempDevice);

}

//Accepts an ID and returns the sensor with that ID

**public** Sensor getSensor(**int** id){

**return** sensors.get(id);

}

//Accepts an ID and an integer and updates the value of the sensor

**public** **void** updateSensor(**int** id, **int** value){

Sensor tempSensor = sensors.get(id);

tempSensor.setValue(value);

sensors.put(id, tempSensor);

}

/\*Accepts an ID and a String formatted to display time.

Updates the time to enable the device with the given ID \*/

**public** **void** setDeviceEnableTime(**int** id, String time){

Device tempDevice = devices.get(id);

tempDevice.setEnableTime(time);

devices.put(id, tempDevice);

}

/\*Accepts an ID and a String formatted to display time.

Updates the time to disable the device with the given ID \*/

**public** **void** setDeviceDisableTime(**int** id, String time){

Device tempDevice = devices.get(id);

tempDevice.setDisableTime(time);

devices.put(id, tempDevice);

}

/\*Accepts an ID and an integer value and updates the device to turn

on when a sensor reaches that value \*/

**public** **void** setDeviceEnableSensorValue(**int** id, **int** value){

Device tempDevice = devices.get(id);

tempDevice.setEnableSensorValue(value);

devices.put(id, tempDevice);

}

/\*Accepts an ID and an integer value and updates the device to turn

off when a sensor reaches that value \*/

**public** **void** setDeviceDisableSensorValue(**int** id, **int** value){

Device tempDevice = devices.get(id);

tempDevice.setDisableSensorValue(value);

devices.put(id, tempDevice);

}

}

**package** org.pachoumis.thesis.restServer.service;

**import** javax.ws.rs.\*;

**import** javax.ws.rs.core.MediaType;

**import** org.pachoumis.thesis.restServer.hardware.Device;

**import** org.pachoumis.thesis.restServer.hardware.Sensor;

/\*

\* This Smart Home Resource class implements the REST resources. It is

\* called every time a client makes a request on the REST server.

\*/

//Root URI

@Path("/resources")

**public** **class** SmartHomeResource {

SmartHomeService shService = **new** SmartHomeService();

//The Smart Home Service instance

/\*

\* Method that is called when the client sends an HTTP GET request

\* to the URI /resources. Returns a text/plain String to the caller

\* containing all the current service resources:

\* Device names

\* Device states

\* Enable/disable device time

\* Enable/disable device based on sensor

\* Sensor names

\* Sensor states

\* (For debugging purposes)

\*/

@GET

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String getInfo(){

String returnMessage = "";

**for**(**int** i=1;i<7;i++){

returnMessage += shService.getDevice(i).getName();

returnMessage += ": ";

returnMessage += shService.getDevice(i).getState();

returnMessage += System.*lineSeparator*();

returnMessage += " TimeEnable: ";

returnMessage += shService.getDevice(i).getEnableTime();

returnMessage += System.*lineSeparator*();

returnMessage += " TimeDisable: ";

returnMessage += shService.getDevice(i).getDisableTime();

returnMessage += System.*lineSeparator*();

returnMessage += " SensorEnable: ";

returnMessage += shService.getDevice(i).getEnableSensorValue();

returnMessage += System.*lineSeparator*();

returnMessage += " SensorDisable: ";

returnMessage += shService.getDevice(i).getDisableSensorValue();

returnMessage += System.*lineSeparator*();

}

returnMessage += System.*lineSeparator*();

**for**(**int** i=1;i<5;i++){

returnMessage += shService.getSensor(i).getName();

returnMessage += ": ";

returnMessage += shService.getSensor(i).getValue();

returnMessage += System.*lineSeparator*();

}

**return** returnMessage;

}

/\*

\* Method that is called when the client sends an HTTP GET request to

\* the URI /resources/device/{deviceId}

\* Returns a text/plain String to the caller containing the state of

\* the device "deviceId"

\*/

@GET

@Path("device/{deviceId}")

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String getDeviceState(@PathParam("deviceId") **int** id) {

System.***out***.println("GET device/" + id);

Device tempDevice = shService.getDevice(id);

**return** Boolean.*toString*(tempDevice.getState());

}

/\*

\* Method that is called when the client sends an HTTP PUT request to

\* the URI /resources/device/{deviceId}

\* It places the string "state" as the state of the device "deviceId"

\* Returns "Success"

\*/

@PUT

@Path("/device/{deviceId}")

@Consumes(MediaType.***TEXT\_PLAIN***)

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String setDeviceState(@PathParam("deviceId") **int** id, String state) {

System.***out***.println("PUT device/" + id + " State: " + state);

shService.updateDevice(id, Boolean.*valueOf*(state));

**return** "Success";

}

/\*

\* Method that is called when the client sends an HTTP GET request to

\* the URI /resources/sensor/{sensorId}

\* Returns a text/plain String to the caller containing the value of

\* the sensor "sensorId"

\*/

@GET

@Path("/sensor/{sensorId}")

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String getSensorState(@PathParam("sensorId") **int** id) {

System.***out***.println("GET sensor/" + id);

Sensor tempSensor = shService.getSensor(id);

**return** Integer.*toString*(tempSensor.getValue());

}

/\*

\* Method that is called when the client sends an HTTP PUT request to

\* the URI /resources/sensor/{sensorId}

\* It places the string "value" as the value of the sensor "sensorId"

\* Returns "Success"

\*/

@PUT

@Path("sensor/{sensorId}")

@Consumes(MediaType.***TEXT\_PLAIN***)

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String setSensorValue(@PathParam("sensorId") **int** id, String value) {

System.***out***.println("PUT sensor/" + id + " Value: " + value);

shService.updateSensor(id, Integer.*parseInt*(value));

**return** "Success";

}

/\*

\* Method that is called when the client sends an HTTP GET request to

\* the URI /resources/device/{deviceId}/enable/time

\* Returns a text/plain String to the caller containing the time when

\* the device "deviceId" is to be enabled

\*/

@GET

@Path("device/{deviceId}/enable/time")

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String getDeviceEnableTimeValue(@PathParam("deviceId")**int** id){

System.***out***.println("GET device/" + id + "/enable/time");

Device tempDevice = shService.getDevice(id);

**return** tempDevice.getEnableTime();

}

/\*

\* Method that is called when the client sends an HTTP PUT request to

\* the URI /resources/device/{deviceId}/enable/time

\* It places the string "time" as the time when the device "deviceId"

\* is to be enabled

\* Returns "Success"

\*/

@PUT

@Path("/device/{deviceId}/enable/time")

@Consumes(MediaType.***TEXT\_PLAIN***)

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String setDeviceEnableTime(@PathParam("deviceId") **int** id, String time) {

System.***out***.println("PUT device/"+id+" Enable: Time: " + time);

shService.setDeviceEnableTime(id, time);

**return** "Success";

}

/\*

\* Method that is called when the client sends an HTTP GET request to

\* the URI /resources/device/{deviceId}/disable/time

\* Returns a text/plain String to the caller containing the time when

\* the device "deviceId" is to be disabled

\*/

@GET

@Path("device/{deviceId}/disable/time")

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String getDeviceDisableTime(@PathParam("deviceId") **int** id) {

System.***out***.println("GET device/" + id + "/disable/time");

Device tempDevice = shService.getDevice(id);

**return** tempDevice.getDisableTime();

}

/\*

\* Method that is called when the client sends an HTTP PUT request to

\* the URI /resources/device/{deviceId}/disable/time

\* It places the string "time" as the time when the device "deviceId"

\* is to be disabled

\* Returns "Success"

\*/

@PUT

@Path("/device/{deviceId}/disable/time")

@Consumes(MediaType.***TEXT\_PLAIN***)

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String setDeviceDisableTime(@PathParam("deviceId") **int** id, String time) {

System.***out***.println("PUT device/"+id+" Disable: Time: "+ time);

shService.setDeviceDisableTime(id, time);

**return** "Success";

}

/\*

\* Method that is called when the client sends an HTTP GET request to

\* the URI /resources/device/{deviceId}/enable/sensorValue

\* Returns a text/plain String to the caller containing the value

\* that the sensor must be for the device "deviceId" to be enabled

\*/

@GET

@Path("device/{deviceId}/enable/sensorValue")

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String getDeviceEnableSensorValue(@PathParam("deviceId") **int** id) {

System.***out***.println("GET device/" + id+ "/enable/sensorValue");

Device tempDevice = shService.getDevice(id);

**return** Integer.*toString*(tempDevice.getEnableSensorValue());

}

/\*

\* Method that is called when the client sends an HTTP PUT request to

\* the URI /resources/device/{deviceId}/enable/sensorValue

\* It places the string "value" as the value that the sensor must be

\* for the device "deviceId" to be enabled

\* Returns "Success"

\*/

@PUT

@Path("/device/{deviceId}/enable/sensorValue")

@Consumes(MediaType.***TEXT\_PLAIN***)

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String setDeviceEnableSensor(@PathParam("deviceId") **int** id, String value) {

System.***out***.println("PUT device/" + id + " Enable: Sensor Value: " + value);

shService.setDeviceEnableSensorValue(id, Integer.*parseInt*(value));

**return** "Success";

}

/\*

\* Method that is called when the client sends an HTTP GET request to

\* the URI /resources/device/{deviceId}/disable/sensorValue

\* Returns a text/plain String to the caller containing the value

\* that the sensor must be for the device "deviceId" to be disabled

\*/

@GET

@Path("device/{deviceId}/disable/sensorValue")

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String getDeviceDisableSensorValue(@PathParam("deviceId") **int** id) {

System.***out***.println("GET device/" +id+ "/disable/sensorValue");

Device tempDevice = shService.getDevice(id);

**return** Integer.*toString*(tempDevice.getDisableSensorValue());

}

/\*

\* Method that is called when the client sends an HTTP PUT request to

\* the URI /resources/device/{deviceId}/disable/sensorValue

\* It places the string "value" as the value that the sensor must be

\* for the device "deviceId" to be disabled

\* Returns "Success"

\*/

@PUT

@Path("/device/{deviceId}/disable/sensorValue")

@Consumes(MediaType.***TEXT\_PLAIN***)

@Produces(MediaType.***TEXT\_PLAIN***)

**public** String setDeviceDisableSensor(@PathParam("deviceId") **int** id, String value) {

System.***out***.println("PUT device/" + id + " Disable: Sensor Value: " + value);

shService.setDeviceDisableSensorValue(id, Integer.*parseInt*(value));

**return** "Success";

}

}

Android App

**Resources – strings**

<**resources**>  
 <**string name="app\_name"**>Home Automation</**string**>  
 <**string name="disabledAlarmMessage"**>Security Offline</**string**>  
 <**string name="alarmMotionDetectionMessage"**>Motion Detected</**string**>  
 <**string name="alarmNoMotionDetectionMessage"**>No Motion Detected</**string**>  
 <**string name="alarmNoServiceMessage"**>"Loading… "</**string**>  
 <**string name="alarmEnabledMessage"**>Enabled</**string**>  
 <**string name="alarmDisabledMessage"**>Disabled</**string**>  
 <**string name="appTitle"**>Home Automation</**string**>  
 <**string name="airConditionTitle"**>Air Condition Settings</**string**>  
 <**string name="alarmTitle"**>Alarm Settings</**string**>  
 <**string name="blindsTitle"**>Blinds Settings</**string**>  
 <**string name="boilerTitle"**>Boiler Settings</**string**>  
 <**string name="alarmButtonText"**>Alarm</**string**>  
 <**string name="lightTitle"**>Light Settings</**string**>  
 <**string name="lightsButtonText"**>Lights</**string**>  
 <**string name="airConditionSwitchText"**>Air Condition:</**string**>  
 <**string name="passwordText"**>Password:</**string**>  
 <**string name="roomBlindsSwitchText"**>Room Blinds:</**string**>  
 <**string name="boilerSwitchText"**>Boiler:</**string**>  
 <**string name="roomLight1SwitchText"**>Room Light 1:</**string**>  
 <**string name="enableText"**>Turn on at:</**string**>  
 <**string name="roomLight2SwitchText"**>Room Light 2:</**string**>  
 <**string name="boilerButtonText"**>Boiler</**string**>  
 <**string name="timeScheduleText"**>Time Schedule</**string**>  
 <**string name="alarmSwitchText"**>Alarm:</**string**>  
 <**string name="airConditionButtonText"**>Air Condition</**string**>  
 <**string name="blindsButtonText"**>Blinds</**string**>  
 <**string name="disableText"**>Turn off at:</**string**>  
 <**string name="alarmStatusText"**>Alarm Status:</**string**>  
 <**string name="enable2Text"**>Enable at:</**string**>  
 <**string name="temperatureText"**>Temperature:</**string**>  
 <**string name="disable2Text"**>Disable at:</**string**>  
 <**string name="sensorScheduleText"**>Sensor Schedule</**string**>  
 <**string name="lightLevelText"**>Light Level:</**string**>  
 <**string name="homeStatusText"**>Home Status:</**string**>  
 <**string name="hotWaterText"**>Hot Water:</**string**>  
 <**string name="disableAlarmSirenText"**>Turn off Alarm Siren</**string**>  
</**resources**>

**Utility Pachage**

**package** com.smarthome.pachoumis.homeautomation.utilityPackage;  
  
*/\*  
The Constant class is used to store the global constants of the app. Contains information  
about the device and sensor valid states, the possible URIs of the REST server and their  
extensions, the null device and sensor states, the user password, the server IP and the  
regular expression patterns that are used throughout the app  
 \*/***public final class** Constant {  
  
 **public static final** String ***DEVICE\_ON*** = **"true"**;  
 **public static final** String ***DEVICE\_OFF*** = **"false"**;  
 **public static final** String ***SENSOR\_OFF*** = **"0"**;  
 **public static final** String ***SENSOR\_ON*** = **"1"**;  
  
 **public static final** String ***DEVICE\_LIGHT\_1\_URI*** = **"device/1"**;  
 **public static final** String ***DEVICE\_LIGHT\_2\_URI*** = **"device/2"**;  
 **public static final** String ***DEVICE\_BOILER\_URI*** = **"device/3"**;  
 **public static final** String ***DEVICE\_AIR\_CONDITION\_URI*** = **"device/4"**;  
 **public static final** String ***DEVICE\_BLINDS\_URI*** = **"device/5"**;  
 **public static final** String ***DEVICE\_ALARM\_URI*** = **"device/6"**;  
 **public static final** String ***SENSOR\_TEMPERATURE\_URI*** = **"sensor/1"**;  
 **public static final** String ***SENSOR\_HOT\_WATER\_URI*** = **"sensor/2"**;  
 **public static final** String ***SENSOR\_LIGHT\_LEVEL\_URI*** = **"sensor/3"**;  
 **public static final** String ***SENSOR\_BREAK\_IN\_URI*** = **"sensor/4"**;  
  
 **public static final** String ***DEVICE\_TIME\_ENABLE\_URI*** = **"/enable/time"**;  
 **public static final** String ***DEVICE\_TIME\_DISABLE\_URI*** = **"/disable/time"**;  
  
 **public static final** String ***DEVICE\_SENSOR\_ENABLE\_URI*** = **"/enable/sensorValue"**;  
 **public static final** String ***DEVICE\_SENSOR\_DISABLE\_URI*** = **"/disable/sensorValue"**;  
  
 **public static final** String ***NO\_TIME\_VALUE*** = **""**;  
 **public static final** String ***NO\_SENSOR\_VALUE*** = **"-1"**;  
 **public static final** String ***NO\_DISPLAY\_VALUE*** = **""**;  
  
 **public static final** String ***TIME\_PATTERN*** = **"(0[0-9]|1[1-9]|2[0-3]):[0-5][0-9]"**;  
 **public static final** String ***PERCENTAGE\_PATTERN*** = **"[0-9]|[1-9][0-9]|100"**;  
  
 **public static final** String ***USER\_PASSWORD*** = **"1234567"**;  
 **public static final** String ***SERVER\_IP*** = **"83.212.118.164"**;  
  
}

**package** com.smarthome.pachoumis.homeautomation.utilityPackage;  
  
**import** java.io.IOException;  
**import** java.io.InputStream;  
**import** java.io.InputStreamReader;  
**import** java.io.OutputStreamWriter;  
**import** java.net.HttpURLConnection;  
**import** java.net.MalformedURLException;  
**import** java.net.ProtocolException;  
**import** java.net.URL;  
  
*/\*  
The HTTP Requests class implements the two out of the four basic HTTP requests: the GET request and  
the PUT request. Those are called throughout the app to exchange information with the REST server.  
 \*/***public class** HttpRequests {  
  
 */\*  
 The getRequest method implements the GET HTTP Request. It accepts an url and tries to connect with  
 that url via the HTTP protocol. Has a timeout of 2 seconds per request. Uses a inputStreamReader  
 to read the response and returns it to the caller. It handles malformed url and i/o exceptions.  
 \*/* **public static** String getRequest(String urlText){  
 String response = **""**;  
 **try** {  
 URL url = **new** URL(**"http://"** + Constant.***SERVER\_IP*** +

**":8080/restServer/resources/"** + urlText);  
 HttpURLConnection urlConnection= (HttpURLConnection) url.openConnection();  
 urlConnection.setRequestMethod(**"GET"**);  
 urlConnection.setConnectTimeout(2000);  
  
 InputStream in = urlConnection.getInputStream();  
 InputStreamReader isw = **new** InputStreamReader(in);  
  
 **int** data = isw.read();  
 **while** (data != -1) {  
 **char** current = (**char**) data;  
 response+= Character.*toString*(current);  
 data = isw.read();  
 }  
  
 urlConnection.disconnect();  
 **return** response;  
  
 }**catch**(MalformedURLException e){  
 System.***out***.println(**"MALFORMED URL ERROR"**);  
 **return "SERVER ERROR"**;  
  
 }**catch** (IOException e){  
 System.***out***.println(**"INPUT OUTPUT ERROR"**);  
 **return "SERVER ERROR"**;  
  
 }**catch** (Exception e){  
 System.***out***.println(**"SERVER NOT FOUND"**);  
 **return "SERVER ERROR"**;  
  
 }  
  
 }  
  
 */\*  
 the putRequest method implements the PUT HTTP Request. It accepts an url and a message and tries  
 to connect with that url and deliver the message via the HTTP protocol. Has a timeout of 2 seconds  
 per request. Uses a outputStreamWriter to send the message to the REST server. Handles protocol,  
 malformed url and i/o exceptions.  
 \*/* **public static void** putRequest(String urlText, String message){  
 **try** {  
 URL url = **new** URL(**"http://"** + Constant.***SERVER\_IP*** +

**":8080/restServer/resources/"** + urlText);  
 HttpURLConnection urlConnection= (HttpURLConnection) url.openConnection();  
 urlConnection.setDoOutput(**true**);  
 urlConnection.setRequestMethod(**"PUT"**);  
 urlConnection.setConnectTimeout(2000);  
 urlConnection.setRequestProperty(**"Content-Type"**,

**"text/plain; charset=UTF-8"**);  
 urlConnection.connect();  
  
 OutputStreamWriter out = **new** OutputStreamWriter(urlConnection

.getOutputStream());  
 out.write(message);  
  
 out.close();  
 System.***out***.println(**"Response: "** + urlConnection.getResponseMessage());  
 urlConnection.disconnect();  
  
 }**catch** (ProtocolException e) {  
 System.***out***.println(**"PROTOCOL ERROR"**);  
  
 }**catch**(MalformedURLException e){  
 System.***out***.println(**"MALFORMED URL ERROR"**);  
  
 }**catch** (IOException e){  
 System.***out***.println(**"INPUT OUTPUT ERROR"**);  
  
 }  
  
 }  
  
}

**package** com.smarthome.pachoumis.homeautomation.utilityPackage;  
  
**import** android.graphics.Color;  
**import** android.view.KeyEvent;  
**import** android.view.inputmethod.EditorInfo;  
**import** android.widget.CheckBox;  
**import** android.widget.CompoundButton;  
**import** android.widget.EditText;  
**import** android.widget.Switch;  
**import** android.widget.TextView;  
  
**import** java.util.regex.Matcher;  
**import** java.util.regex.Pattern;  
  
*/\*  
The GlobalMethods class implements the methods that are used through the app. Instead of implementing the methods  
under each class, i implement them here because the only difference are the local variable names. Has methods about  
initializing View objects and their functionality.  
 \*/***public class** GlobalMethods {  
  
 */\*  
 The initializeSwitch method initializes a switch in the onCreate method of each activity. Checks with the server  
 and sets the state of the switch depending on the server values. Accepts the device URI and the switch object.  
 \*/* **public static void** initializeSwitch(String deviceURI, Switch aSwitch){  
 **if**(HttpRequests.*getRequest*(deviceURI).equals(Constant.***DEVICE\_ON***))  
 aSwitch.setChecked(**true**);  
 **else** aSwitch.setChecked(**false**);  
 }  
  
 */\*  
 The initializeCheckBoxAndField method sets up a checkbox and its corresponding editText field inside the onCreate  
 method of each activity. Accepts the device URI, the function URI extension of the request, a checkbox and an  
 editText field. It initializes the state of the checkbox and the text of the editText field depending on the  
 server values for these fields.  
 \*/* **public static void** initializeCheckBoxAndField(String deviceURI, String functionURI, CheckBox checkbox, EditText edittext){  
 String request = HttpRequests.*getRequest*(deviceURI + functionURI);  
 String nullValue;  
 **if**(functionURI.equals(Constant.***DEVICE\_TIME\_ENABLE\_URI***) || functionURI.equals(Constant.***DEVICE\_TIME\_DISABLE\_URI***))  
 nullValue = Constant.***NO\_TIME\_VALUE***;

*//If the extension is about time, set the null value to blank text* **else** nullValue = Constant.***NO\_SENSOR\_VALUE***; *//If the extension is about a sensor, set the null value to -1* **if**(request.equals(nullValue)){  
 checkbox.setChecked(**false**);  
 checkbox.setEnabled(**false**); *//Disable checkbox* checkbox.setClickable(**false**);  
 checkbox.setTextColor(Color.*rgb*(117,118,120)); *//Color Gray* edittext.setText(Constant.***NO\_DISPLAY\_VALUE***); *//Leave the editText blank* }**else**{  
 checkbox.setChecked(**true**);  
 checkbox.setEnabled(**true**); *//Enable checkbox* checkbox.setClickable(**true**);  
 edittext.setText(request); *//Display the value in the editText* checkbox.setTextColor(Color.*rgb*(67,78,121)); *//Color Blue* }  
 }  
  
 */\*  
 The switchFunctionality method implements the rules of the switch, how the switch must behave in the app. It  
 accepts the switch and its corresponding device URI and listens for an onCheckedChanged action that occurs  
 when the switch changes state. Then it sends the REST server information depending on the state of the switch  
 \*/* **public static void** switchFunctionality(Switch aSwitch, **final** String deviceURI){  
 aSwitch.setOnCheckedChangeListener(**new** CompoundButton.

OnCheckedChangeListener() {  
 @Override  
 **public void** onCheckedChanged(CompoundButton buttonView, **boolean** isChecked) {  
 **if**(isChecked){  
 HttpRequests.*putRequest*(deviceURI,Constant.***DEVICE\_ON***);  
 }**else**{  
 HttpRequests.*putRequest*(deviceURI,Constant.***DEVICE\_OFF***);  
 }  
 }  
 });  
 }  
  
 */\*  
 The checkBoxFunctionality method implements the rules of the checkbox and its corresponding editText field. It  
 accepts a checkbox, an editText field, the device the checkbox is bound to and the function URI extension. It  
 usually precedes the editTextFunctionality method. Implements a listener that listens for an onCheckedChanged  
 action that occurs when the checkbox changes state. Then it sends the REST server information depending on the  
 state of the checkbox.  
 \*/* **public static void** checkBoxFunctionality(**final** CheckBox checkbox, **final** EditText edittext, **final** String deviceURI, **final** String functionURI){  
 checkbox.setOnCheckedChangeListener(**new** CompoundButton.OnCheckedChangeListener() {  
 @Override  
 **public void** onCheckedChanged(CompoundButton buttonView, **boolean** isChecked) {  
 **if**(isChecked){  
 HttpRequests.*putRequest*(deviceURI + functionURI,edittext.getText().toString());  
 }**else**{  
 **if**(functionURI.equals(Constant.***DEVICE\_TIME\_ENABLE\_URI***) || functionURI.equals(Constant.***DEVICE\_TIME\_DISABLE\_URI***)){  
 HttpRequests.*putRequest*(deviceURI + functionURI,Constant.***NO\_TIME\_VALUE***);  
 }**else if**(functionURI.equals(Constant.***DEVICE\_SENSOR\_ENABLE\_URI***) || functionURI.equals(Constant.***DEVICE\_SENSOR\_DISABLE\_URI***)){  
 HttpRequests.*putRequest*(deviceURI + functionURI,Constant.***NO\_SENSOR\_VALUE***);  
 }  
 checkbox.setEnabled(**false**);  
 checkbox.setClickable(**false**);  
 checkbox.setTextColor(Color.*rgb*(117,118,120));  
 }  
 }  
 });  
 }  
  
 */\*  
 The editTextFunctionality method implements the rules for the editText field corresponding to a checkbox. It  
 accepts the editText field, the checkbox and a regular expression. When the user presses the "Done" button  
 on the device soft keyboard, the method checks if the text in the editText field matches the regular expression  
 given. If so it enables the checkbox changes its color to Blue. If not it disables the checkbox and sets its  
 color to grey.  
 \*/* **public static void** editTextFunctionality(**final** EditText edittext, **final** CheckBox checkbox, **final** String aPattern){  
 edittext.setOnEditorActionListener(**new** TextView.OnEditorActionListener(){  
 **public boolean** onEditorAction(TextView exampleView, **int** actionId, KeyEvent event) {  
 **if** (actionId == EditorInfo.***IME\_ACTION\_DONE***) {  
 Boolean matches;  
 **try** {  
 Pattern pattern = Pattern.*compile*(aPattern);  
 Matcher matcher = pattern.matcher(edittext.getText().toString());  
 matches = matcher.matches();  
 } **catch** (RuntimeException e) {  
 matches = **false**;  
 }  
 **if**(matches){  
 checkbox.setEnabled(**true**);  
 checkbox.setClickable(**true**);  
 checkbox.setTextColor(Color.*rgb*(67,78,121));  
 }**else**{  
 checkbox.setEnabled(**false**);  
 checkbox.setClickable(**false**);  
 checkbox.setTextColor(Color.*rgb*(117,118,120));  
 }  
 }  
 **return false**;  
 }  
 });  
 }  
}

**Activity Package**

**package** com.smarthome.pachoumis.homeautomation.activityPackage;  
  
**import** android.os.StrictMode;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;  
**import** android.widget.CheckBox;  
**import** android.widget.EditText;  
**import** android.widget.Switch;  
**import** android.widget.TextView;  
  
**import** com.smarthome.pachoumis.homeautomation.R;  
**import** com.smarthome.pachoumis.homeautomation.utilityPackage.Constant;  
**import** com.smarthome.pachoumis.homeautomation.utilityPackage.GlobalMethods;  
**import** com.smarthome.pachoumis.homeautomation.utilityPackage.HttpRequests;  
  
**import** java.util.concurrent.Executors;  
**import** java.util.concurrent.ScheduledExecutorService;  
**import** java.util.concurrent.TimeUnit;  
  
*/\*  
The AirConditionScreen class implements the activity that is responsible for the management  
of the smart home air condition device. Has the ability to enable and disable the air  
condition and set it to be enabled or disabled when the temperature reaches a certain  
value. Displays information about the temperature of the room.  
 \*/***public class** AirConditionScreen **extends** AppCompatActivity {  
  
 **private** TextView **temperatureText**;  
  
 */\*  
 Method called when the activity is created  
 \*/* @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_air\_condition\_screen***);  
  
 *//Permit all policy for the threads* **if** (android.os.Build.VERSION.***SDK\_INT*** > 9) {  
 StrictMode.ThreadPolicy policy = **new** StrictMode.ThreadPolicy.Builder().permitAll().build();  
 StrictMode.*setThreadPolicy*(policy);  
 }  
  
 *//The view objects and their bindings to the layout xml counterpart of the activity* Switch airConditionSwitch = (Switch) findViewById(R.id.***acsAirConditionSwitch***);  
 CheckBox turnOnTemperatureBox = (CheckBox) findViewById(R.id.***acsTurnOnTemperatureBox***);  
 CheckBox turnOffTemperatureBox = (CheckBox) findViewById(R.id.***acsTurnOffTemperatureBox***);  
 EditText turnOnTemperatureField = (EditText) findViewById(R.id.***acsTurnOnTemperatureField***);  
 EditText turnOffTemperatureField = (EditText) findViewById(R.id.***acsTurnOffTemperatureField***);  
 **temperatureText** = (TextView) findViewById(R.id.***acsTemperatureText***);  
  
 *//initializing the switch for the air condition and the checkboxes and fields for sensor value scheduling* GlobalMethods.*initializeSwitch*(Constant.***DEVICE\_AIR\_CONDITION\_URI***, airConditionSwitch);  
 GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_AIR\_CONDITION\_URI***,Constant.***DEVICE\_SENSOR\_ENABLE\_URI***, turnOnTemperatureBox, turnOnTemperatureField);  
 GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_AIR\_CONDITION\_URI***,Constant.***DEVICE\_SENSOR\_DISABLE\_URI***, turnOffTemperatureBox, turnOffTemperatureField);  
  
 *//methods that add functionality to the switch and the time schedule checkboxes and fields* GlobalMethods.*switchFunctionality*(airConditionSwitch,Constant.***DEVICE\_AIR\_CONDITION\_URI***);  
 GlobalMethods.*checkBoxFunctionality*(turnOnTemperatureBox, turnOnTemperatureField,Constant.***DEVICE\_AIR\_CONDITION\_URI***, Constant.***DEVICE\_SENSOR\_ENABLE\_URI***);  
 GlobalMethods.*editTextFunctionality*(turnOnTemperatureField, turnOnTemperatureBox,Constant.***PERCENTAGE\_PATTERN***);  
  
 *//methods that add functionality to the sensor schedule checkboxes and fields* GlobalMethods.*checkBoxFunctionality*(turnOffTemperatureBox, turnOffTemperatureField,Constant.***DEVICE\_AIR\_CONDITION\_URI***, Constant.***DEVICE\_SENSOR\_DISABLE\_URI***);  
 GlobalMethods.*editTextFunctionality*(turnOffTemperatureField, turnOffTemperatureBox,Constant.***PERCENTAGE\_PATTERN***);  
  
 }  
  
 */\*  
 Method called when the activity is resumed. Also runs after onCreate.  
 \*/* @Override  
 **protected void** onResume() {  
 **super**.onResume();  
  
 *//Scheduling a new Thread to run every 10 seconds* ScheduledExecutorService scheduledExecutorService = Executors.*newSingleThreadScheduledExecutor*();  
 scheduledExecutorService.scheduleAtFixedRate(**new** Runnable(){  
  
 @Override  
 **public void** run() {  
 AirConditionScreen.**this**.runOnUiThread(**new** Runnable(){ *//This runs on User Interface level* @Override  
 **public void** run() {  
 *//Get temperature information from the server and display it* String tempSensor = HttpRequests.*getRequest*(Constant.***SENSOR\_TEMPERATURE\_URI***);  
 **temperatureText**.setText(tempSensor + **"° C"**);  
  
 }  
 });  
 }  
 },10,10, TimeUnit.***SECONDS***);  
 }  
}

**package** com.smarthome.pachoumis.homeautomation.activityPackage;  
  
**import** android.graphics.Color;  
**import** android.os.StrictMode;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;  
**import** android.view.KeyEvent;  
**import** android.view.View;  
**import** android.view.inputmethod.EditorInfo;  
**import** android.widget.Button;  
**import** android.widget.CheckBox;  
**import** android.widget.EditText;  
**import** android.widget.Switch;  
**import** android.widget.TextView;  
  
**import** com.smarthome.pachoumis.homeautomation.R;  
**import** com.smarthome.pachoumis.homeautomation.utilityPackage.Constant;  
**import** com.smarthome.pachoumis.homeautomation.utilityPackage.GlobalMethods;  
**import** com.smarthome.pachoumis.homeautomation.utilityPackage.HttpRequests;  
  
**import** java.util.concurrent.Executors;  
**import** java.util.concurrent.ScheduledExecutorService;  
**import** java.util.concurrent.TimeUnit;  
  
*/\*  
The AlarmScreen class implements the activity that is responsible for the management  
of the smart home alarm device. Has the ability to enable and disable the home alarm  
after the password is given and set the alarm to be enabled or disabled at a certain  
time of the day. Displays information about the status of the alarm and if someone  
has broken in. Contains a button for disabling the alarm siren.  
 \*/***public class** AlarmScreen **extends** AppCompatActivity {  
  
 **private** EditText **passwordField**;  
 **private** Switch **alarmSwitch**;  
 **private** TextView **alarmLabel**;  
 **private** TextView passwordLabel;  
 **private** TextView **homeStatusText**;  
  
*/\*  
Method called when the activity is created  
 \*/* @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_alarm\_screen***);  
  
 *//Permit all policy for the threads* **if** (android.os.Build.VERSION.***SDK\_INT*** > 9) {  
 StrictMode.ThreadPolicy policy = **new** StrictMode.ThreadPolicy.Builder().permitAll().build();  
 StrictMode.*setThreadPolicy*(policy);  
 }  
  
 *//The view objects and their bindings to the layout xml counterpart of the activity* **passwordField** = (EditText) findViewById(R.id.***asPasswordField***);  
 **alarmSwitch** = (Switch) findViewById(R.id.***asAlarmSwitch***);  
 CheckBox enableTimeBox = (CheckBox) findViewById((R.id.***asEnableTimeBox***));  
 EditText enableTimeField = (EditText) findViewById(R.id.***asEnableTimeField***);  
 CheckBox disableTimeBox = (CheckBox) findViewById(R.id.***asDisableTimeBox***);  
 EditText disableTimeField = (EditText) findViewById(R.id.***asDisableTimeField***);  
 Button alarmOffButton = (Button) findViewById(R.id.***asAlarmOffButton***);  
 **alarmLabel** = (TextView) findViewById(R.id.***asAlarmLabel***);  
 passwordLabel = (TextView) findViewById(R.id.***asPasswordLabel***);  
 **homeStatusText** = (TextView) findViewById(R.id.***asHomeStatusText***);  
  
 *//initializing the switch for the alarm and the checkboxes and fields for time scheduling* GlobalMethods.*initializeSwitch*(Constant.***DEVICE\_ALARM\_URI***,**alarmSwitch**);  
 GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_ALARM\_URI***,Constant.***DEVICE\_TIME\_ENABLE\_URI***, enableTimeBox, enableTimeField);  
 GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_ALARM\_URI***,Constant.***DEVICE\_TIME\_DISABLE\_URI***, disableTimeBox, disableTimeField);  
  
 */\*  
 A listener for the password field. If the password matches the constant password then make  
 the alarm Label green to notify the user that it is enabled. If the password doesn't match  
 the constant password then make the alarm label red to notify the user of the incorrect password  
 \*/* **passwordField**.setOnEditorActionListener(**new** TextView.OnEditorActionListener(){  
 **public boolean** onEditorAction(TextView exampleView, **int** actionId, KeyEvent event) {  
 **if** (actionId == EditorInfo.***IME\_ACTION\_DONE***) {  
 **if**((**passwordField**.getText().toString()).equals(Constant.***USER\_PASSWORD***)){  
 **alarmSwitch**.setClickable(**true**);  
 **alarmLabel**.setTextColor(Color.*rgb*(67,78,121));  
 passwordLabel.setTextColor(Color.*rgb*(102,153,50));  
 }  
 **else**{  
 **alarmSwitch**.setClickable(**false**);  
 **alarmLabel**.setTextColor(Color.*rgb*(117,118,120));  
 passwordLabel.setTextColor(Color.*rgb*(204,0,0));  
 }  
 }  
 **return false**;  
 }  
 });  
  
 *//methods that add functionality to the switch and the time schedule checkboxes and fields* GlobalMethods.*switchFunctionality*(**alarmSwitch**,Constant.***DEVICE\_ALARM\_URI***);  
 GlobalMethods.*checkBoxFunctionality*(enableTimeBox, enableTimeField,Constant.***DEVICE\_ALARM\_URI***,Constant.***DEVICE\_TIME\_ENABLE\_URI***);  
 GlobalMethods.*editTextFunctionality*(enableTimeField, enableTimeBox,Constant.***TIME\_PATTERN***);  
 GlobalMethods.*checkBoxFunctionality*(disableTimeBox, disableTimeField,Constant.***DEVICE\_ALARM\_URI***,Constant.***DEVICE\_TIME\_DISABLE\_URI***);  
 GlobalMethods.*editTextFunctionality*(disableTimeField, disableTimeBox,Constant.***TIME\_PATTERN***);  
  
 *//If the user clicks this button the alarm siren is turned off* **assert** alarmOffButton != **null**;  
 alarmOffButton.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 HttpRequests.*putRequest*(Constant.***SENSOR\_BREAK\_IN\_URI***,Constant.***SENSOR\_OFF***);  
 }  
 });  
  
 }  
  
 */\*  
 Method called when the activity is resumed. Also runs after onCreate.  
 \*/* @Override  
 **protected void** onResume() {  
 **super**.onResume();  
  
 *//Scheduling a new Thread to run every 10 seconds* ScheduledExecutorService scheduledExecutorService = Executors.*newSingleThreadScheduledExecutor*();  
 scheduledExecutorService.scheduleAtFixedRate(**new** Runnable(){  
  
 @Override  
 **public void** run() {  
 AlarmScreen.**this**.runOnUiThread(**new** Runnable(){ *//This runs on User Interface level  
  
 /\*  
 This Runnable displays information about the break in sensor. If the alarm is enabled and  
 the break in sensor displays no "motion" the text becomes green and displays the appropriate  
 text. If the alarm is enabled and the break in sensor displays "motion", the text becomes  
 red and displays again the appropriate text. If the alarm is disabled then there is no  
 motion data to be displayed and the appropriate text is being displayed  
 \*/* @Override  
 **public void** run() {  
 String homeStatus = HttpRequests.*getRequest*(Constant.***SENSOR\_BREAK\_IN\_URI***);  
 **if**(!**alarmSwitch**.isChecked()) {  
 **homeStatusText**.setText(R.string.***disabledAlarmMessage***);  
 **homeStatusText**.setTextColor(Color.*rgb*(255,136,0));  
 }**else if**(**alarmSwitch**.isChecked()){  
 **if** (homeStatus.equals(Constant.***SENSOR\_ON***)) {  
 **homeStatusText**.setText(R.string.***alarmMotionDetectionMessage***);  
 **homeStatusText**.setTextColor(Color.*rgb*(204, 0, 0));  
 } **else** {  
 **homeStatusText**.setText(R.string.***alarmNoMotionDetectionMessage***);  
 **homeStatusText**.setTextColor(Color.*rgb*(102, 153, 50));  
 }  
 }  
 **else**{  
 **homeStatusText**.setText(R.string.***alarmNoServiceMessage***);  
 **homeStatusText**.setTextColor(Color.*rgb*(117,118,120));  
 }  
  
 }  
 });  
 }  
 },10,10, TimeUnit.***SECONDS***);  
 }  
}

**package** com.smarthome.pachoumis.homeautomation.activityPackage;  
  
**import** android.os.StrictMode;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;  
**import** android.widget.CheckBox;  
**import** android.widget.EditText;  
**import** android.widget.Switch;  
**import** android.widget.TextView;  
  
**import** com.smarthome.pachoumis.homeautomation.R;  
**import** com.smarthome.pachoumis.homeautomation.utilityPackage.Constant;  
**import** com.smarthome.pachoumis.homeautomation.utilityPackage.GlobalMethods;  
**import** com.smarthome.pachoumis.homeautomation.utilityPackage.HttpRequests;  
  
**import** java.util.concurrent.Executors;  
**import** java.util.concurrent.ScheduledExecutorService;  
**import** java.util.concurrent.TimeUnit;  
  
*/\*  
The BlindsScreen class implements the activity that is responsible for the management  
of the smart home window blinds device. Has the ability to enable and disable the  
blinds and/or set them to be enabled or disabled at a certain time of the day or  
when the light level reaches a certain value. Displays information about the light level  
 \*/***public class** BlindsScreen **extends** AppCompatActivity {  
  
 **private** TextView **lightLevelText**;  
  
 */\*  
 Method called when the activity is created  
 \*/* @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_blinds\_screen***);  
  
 *//Permit all policy for the threads* **if** (android.os.Build.VERSION.***SDK\_INT*** > 9) {  
 StrictMode.ThreadPolicy policy = **new** StrictMode.ThreadPolicy.Builder().permitAll().build();  
 StrictMode.*setThreadPolicy*(policy);  
 }  
  
 *//The view objects and their bindings to the layout xml counterpart of the activity* Switch blindsSwitch = (Switch) findViewById(R.id.***blsBlindsSwitch***);  
 CheckBox turnOnTimeBox = (CheckBox) findViewById(R.id.***blsTurnOnTimeBox***);  
 EditText turnOnTimeField = (EditText) findViewById(R.id.***blsTurnOnTimeField***);  
 CheckBox turnOffTimeBox = (CheckBox) findViewById(R.id.***blsTurnOffTimeBox***);  
 EditText turnOffTimeField = (EditText) findViewById(R.id.***blsTurnOffTimeField***);  
 CheckBox turnOnSensorBox = (CheckBox) findViewById(R.id.***blsTurnOnSensorBox***);  
 EditText turnOnSensorField = (EditText) findViewById(R.id.***blsTurnOnSensorField***);  
 CheckBox turnOffSensorBox = (CheckBox) findViewById(R.id.***blsTurnOffSensorBox***);  
 EditText turnOffSensorField = (EditText) findViewById(R.id.***blsTurnOffSensorField***);  
 **lightLevelText** = (TextView) findViewById(R.id.***blsLightLevelText***);  
  
 *//initializing the switch for the window blinds and the checkboxes and fields for time and sensor value scheduling* GlobalMethods.*initializeSwitch*(Constant.***DEVICE\_BLINDS\_URI***, blindsSwitch);  
 GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_BLINDS\_URI***,Constant.***DEVICE\_TIME\_ENABLE\_URI***, turnOnTimeBox, turnOnTimeField);  
 GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_BLINDS\_URI***,Constant.***DEVICE\_TIME\_DISABLE\_URI***, turnOffTimeBox, turnOffTimeField);  
 GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_BLINDS\_URI***,Constant.***DEVICE\_SENSOR\_ENABLE\_URI***, turnOnSensorBox, turnOnSensorField);  
 GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_BLINDS\_URI***,Constant.***DEVICE\_SENSOR\_DISABLE\_URI***, turnOffSensorBox, turnOffSensorField);  
  
 *//methods that add functionality to the switch, the time schedule checkboxes and fields and the sensor value checkboxes and fields* GlobalMethods.*switchFunctionality*(blindsSwitch,Constant.***DEVICE\_BLINDS\_URI***);  
 GlobalMethods.*checkBoxFunctionality*(turnOnTimeBox, turnOnTimeField,Constant.***DEVICE\_BLINDS\_URI***,Constant.***DEVICE\_TIME\_ENABLE\_URI***);  
 GlobalMethods.*editTextFunctionality*(turnOnTimeField, turnOnTimeBox,Constant.***TIME\_PATTERN***);  
 GlobalMethods.*checkBoxFunctionality*(turnOffTimeBox, turnOffTimeField,Constant.***DEVICE\_BLINDS\_URI***,Constant.***DEVICE\_TIME\_DISABLE\_URI***);  
 GlobalMethods.*editTextFunctionality*(turnOffTimeField, turnOffTimeBox,Constant.***TIME\_PATTERN***);  
 GlobalMethods.*checkBoxFunctionality*(turnOnSensorBox, turnOnSensorField,Constant.***DEVICE\_BLINDS\_URI***,Constant.***DEVICE\_SENSOR\_ENABLE\_URI***);  
 GlobalMethods.*editTextFunctionality*(turnOnSensorField, turnOnSensorBox,Constant.***PERCENTAGE\_PATTERN***);  
 GlobalMethods.*checkBoxFunctionality*(turnOffSensorBox, turnOffSensorField,Constant.***DEVICE\_BLINDS\_URI***,Constant.***DEVICE\_SENSOR\_DISABLE\_URI***);  
 GlobalMethods.*editTextFunctionality*(turnOffSensorField, turnOffSensorBox,Constant.***PERCENTAGE\_PATTERN***);  
 }  
  
 */\*  
 Method called when the activity is resumed. Also runs after onCreate.  
 \*/* @Override  
 **protected void** onResume() {  
 **super**.onResume();  
  
 *//Scheduling a new Thread to run every 10 seconds* ScheduledExecutorService scheduledExecutorService = Executors.*newSingleThreadScheduledExecutor*();  
 scheduledExecutorService.scheduleAtFixedRate(**new** Runnable(){  
  
 @Override  
 **public void** run() {  
 BlindsScreen.**this**.runOnUiThread(**new** Runnable(){ *//This runs on User Interface level* @Override  
 **public void** run() {  
  
 *//Get light level information from the server and display it* String lightSensor = HttpRequests.*getRequest*(Constant.***SENSOR\_LIGHT\_LEVEL\_URI***);  
 **lightLevelText**.setText(lightSensor + **" %"**);  
  
 }  
 });  
 }  
 },10,10, TimeUnit.***SECONDS***);  
 }  
}

**package** com.smarthome.pachoumis.homeautomation.activityPackage;  
  
**import** android.os.StrictMode;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;  
**import** android.widget.CheckBox;  
**import** android.widget.EditText;  
**import** android.widget.Switch;  
**import** android.widget.TextView;  
  
**import** com.smarthome.pachoumis.homeautomation.R;  
**import** com.smarthome.pachoumis.homeautomation.utilityPackage.Constant;  
**import** com.smarthome.pachoumis.homeautomation.utilityPackage.GlobalMethods;  
**import** com.smarthome.pachoumis.homeautomation.utilityPackage.HttpRequests;  
  
**import** java.util.concurrent.Executors;  
**import** java.util.concurrent.ScheduledExecutorService;  
**import** java.util.concurrent.TimeUnit;  
  
*/\*  
The BoilerScreen class implements the activity that is responsible for the management  
of the smart home boiler device. Has the ability to enable and disable the boiler and/or  
set it to be enabled or disabled when the hot water percentage reaches a certain value. \  
Displays information about the hot water inside the boiler.  
 \*/***public class** BoilerScreen **extends** AppCompatActivity {  
  
 **private** TextView **hotWaterText**;  
  
 */\*  
 Method called when the activity is created  
 \*/* @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_boiler\_screen***);  
  
 *//Permit all policy for the threads* **if** (android.os.Build.VERSION.***SDK\_INT*** > 9) {  
 StrictMode.ThreadPolicy policy = **new** StrictMode.ThreadPolicy.Builder().permitAll().build();  
 StrictMode.*setThreadPolicy*(policy);  
 }  
  
 *//The view objects and their bindings to the layout xml counterpart of the activity* Switch boilerSwitch = (Switch) findViewById(R.id.***bosBoilerSwitch***);  
 CheckBox turnOnTimeBox = (CheckBox) findViewById(R.id.***bosTurnOnTimeBox***);  
 EditText turnOnTimeField = (EditText) findViewById(R.id.***bosTurnOnTimeField***);  
 CheckBox turnOffTimeBox = (CheckBox) findViewById(R.id.***bosTurnOffTimeBox***);  
 EditText turnOffTimeField = (EditText) findViewById(R.id.***bosTurnOffTimeField***);  
 CheckBox turnOnSensorBox = (CheckBox) findViewById(R.id.***bosTurnOnSensorBox***);  
 EditText turnOnSensorField = (EditText) findViewById(R.id.***bosTurnOnSensorField***);  
 CheckBox turnOffSensorBox = (CheckBox) findViewById(R.id.***bosTurnOffSensorBox***);  
 EditText turnOffSensorField = (EditText) findViewById(R.id.***bosTurnOffSensorField***);  
 **hotWaterText** = (TextView) findViewById(R.id.***bosHotWaterText***);  
  
 *//initializing the switch for the boiler and the checkboxes and fields for time and sensor value scheduling* GlobalMethods.*initializeSwitch*(Constant.***DEVICE\_BOILER\_URI***, boilerSwitch);  
 GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_BOILER\_URI***,Constant.***DEVICE\_TIME\_ENABLE\_URI***, turnOnTimeBox, turnOnTimeField);  
 GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_BOILER\_URI***,Constant.***DEVICE\_TIME\_DISABLE\_URI***, turnOffTimeBox, turnOffTimeField);  
 GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_BOILER\_URI***,Constant.***DEVICE\_SENSOR\_ENABLE\_URI***, turnOnSensorBox, turnOnSensorField);  
 GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_BOILER\_URI***,Constant.***DEVICE\_SENSOR\_DISABLE\_URI***, turnOffSensorBox, turnOffSensorField);  
  
 *//methods that add functionality to the switch, the time schedule checkboxes and fields and the sensor value checkboxes and fields* GlobalMethods.*switchFunctionality*(boilerSwitch,Constant.***DEVICE\_BOILER\_URI***);  
 GlobalMethods.*checkBoxFunctionality*(turnOnTimeBox, turnOnTimeField,Constant.***DEVICE\_BOILER\_URI***,Constant.***DEVICE\_TIME\_ENABLE\_URI***);  
 GlobalMethods.*editTextFunctionality*(turnOnTimeField, turnOnTimeBox,Constant.***TIME\_PATTERN***);  
 GlobalMethods.*checkBoxFunctionality*(turnOffTimeBox, turnOffTimeField,Constant.***DEVICE\_BOILER\_URI***,Constant.***DEVICE\_TIME\_DISABLE\_URI***);  
 GlobalMethods.*editTextFunctionality*(turnOffTimeField, turnOffTimeBox,Constant.***TIME\_PATTERN***);  
 GlobalMethods.*checkBoxFunctionality*(turnOnSensorBox, turnOnSensorField,Constant.***DEVICE\_BOILER\_URI***,Constant.***DEVICE\_SENSOR\_ENABLE\_URI***);  
 GlobalMethods.*editTextFunctionality*(turnOnSensorField, turnOnSensorBox,Constant.***PERCENTAGE\_PATTERN***);  
 GlobalMethods.*checkBoxFunctionality*(turnOffSensorBox, turnOffSensorField,Constant.***DEVICE\_BOILER\_URI***,Constant.***DEVICE\_SENSOR\_DISABLE\_URI***);  
 GlobalMethods.*editTextFunctionality*(turnOffSensorField, turnOffSensorBox,Constant.***PERCENTAGE\_PATTERN***);  
 }  
  
 */\*  
 Method called when the activity is resumed. Also runs after onCreate.  
 \*/* @Override  
 **protected void** onResume() {  
 **super**.onResume();  
  
 *//Scheduling a new Thread to run every 10 seconds* ScheduledExecutorService scheduledExecutorService = Executors.*newSingleThreadScheduledExecutor*();  
 scheduledExecutorService.scheduleAtFixedRate(**new** Runnable(){  
  
 @Override  
 **public void** run() {  
 BoilerScreen.**this**.runOnUiThread(**new** Runnable(){ *//This runs on User Interface level* @Override  
 **public void** run() {  
  
 *//Get hot water information from the server and display it* String hotWaterSensor = HttpRequests.*getRequest*(Constant.***SENSOR\_HOT\_WATER\_URI***);  
 **hotWaterText**.setText(hotWaterSensor + **" %"**);  
  
 }  
 });  
 }  
 },10,10, TimeUnit.***SECONDS***);  
 }  
}  
  
**package** com.smarthome.pachoumis.homeautomation.activityPackage;  
  
**import** android.content.Intent;  
**import** android.graphics.Color;  
**import** android.os.StrictMode;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.TextView;  
  
**import** com.smarthome.pachoumis.homeautomation.R;  
**import** com.smarthome.pachoumis.homeautomation.utilityPackage.Constant;  
**import** com.smarthome.pachoumis.homeautomation.utilityPackage.HttpRequests;  
  
**import** java.util.concurrent.Executors;  
**import** java.util.concurrent.ScheduledExecutorService;  
**import** java.util.concurrent.TimeUnit;  
  
*/\*  
The HomeScreen class implements the activity that gives the user access to all the devices  
by calling instances of the other activities. Is the activity that launches when the user  
first runs the app. Displays information about the alarm status, the break in sensor status,  
the light level, the temperature and the hot water level.  
 \*/***public class** HomeScreen **extends** AppCompatActivity {  
  
  
 **private** ScheduledExecutorService **scheduledExecutorService**;  
 **private** TextView **alarmStatusText**;  
 **private** TextView **homeStatusText**;  
 **private** TextView **lightLevelText**;  
 **private** TextView **hotWaterText**;  
 **private** TextView **temperatureText**;  
  
 */\*  
 Method called when the activity is created  
 \*/* @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_home\_screen***);  
  
 *//Permit all policy for the threads* **if** (android.os.Build.VERSION.***SDK\_INT*** > 9) {  
 StrictMode.ThreadPolicy policy = **new** StrictMode.ThreadPolicy.Builder().permitAll().build();  
 StrictMode.*setThreadPolicy*(policy);  
 }  
  
 *//The view objects and their bindings to the layout xml counterpart of the activity* Button alarmButton = (Button) findViewById(R.id.***hsAlarmButton***);  
 Button lightsButton = (Button) findViewById(R.id.***hsLightsButton***);  
 Button boilerButton = (Button) findViewById(R.id.***hsBoilerButton***);  
 Button airConditionButton = (Button) findViewById(R.id.***hsAirConditionButton***);  
 Button blindsButton = (Button) findViewById(R.id.***hsBlindsButton***);  
  
 **alarmStatusText** = (TextView) findViewById(R.id.***hsAlarmStatusText***);  
 **homeStatusText** = (TextView) findViewById(R.id.***hsHomeStatusText***);  
 **lightLevelText** = (TextView) findViewById(R.id.***hsLightLevelText***);  
 **hotWaterText** = (TextView) findViewById(R.id.***hsHotWaterText***);  
 **temperatureText** = (TextView) findViewById(R.id.***hsTemperatureText***);  
  
 *//The buttons for navigating to the other activities* runActivityButton(alarmButton,AlarmScreen.**class**);  
 runActivityButton(lightsButton,LightsScreen.**class**);  
 runActivityButton(boilerButton,BoilerScreen.**class**);  
 runActivityButton(airConditionButton,AirConditionScreen.**class**);  
 runActivityButton(blindsButton,BlindsScreen.**class**);  
 }  
  
 */\*  
 Method called when the activity is resumed. Also runs after onCreate.  
 \*/* @Override  
 **protected void** onResume() {  
 **super**.onResume();  
  
 *//Scheduling a new Thread to run every 10 seconds* **scheduledExecutorService** = Executors.*newSingleThreadScheduledExecutor*();  
 **scheduledExecutorService**.scheduleAtFixedRate(**new** Runnable(){  
  
 @Override  
 **public void** run() {  
 HomeScreen.**this**.runOnUiThread(**new** Runnable(){ *//This runs on User Interface level* @Override  
 **public void** run() {  
  
 *//Get temperature information from the server and display it* String tempSensor = HttpRequests.*getRequest*(Constant.***SENSOR\_TEMPERATURE\_URI***);  
 **temperatureText**.setText(tempSensor + **"° C"**);  
  
 *//Get hot water sensor information from the server and display it* String hotWSensor = HttpRequests.*getRequest*(Constant.***SENSOR\_HOT\_WATER\_URI***);  
 **hotWaterText**.setText(hotWSensor + **" %"**);  
  
 *//Get light level information from the server and display it* String lightSensor = HttpRequests.*getRequest*(Constant.***SENSOR\_LIGHT\_LEVEL\_URI***);  
 **lightLevelText**.setText(lightSensor + **" %"**);  
  
 *//Get alarm status and display color and text accordingly* String alarmStatus = HttpRequests.*getRequest*(Constant.***DEVICE\_ALARM\_URI***);  
 **switch** (alarmStatus) {  
 **case** Constant.***DEVICE\_ON***:  
 **alarmStatusText**.setText(R.string.***alarmEnabledMessage***);  
 **alarmStatusText**.setTextColor(Color.*rgb*(102, 153, 50)); *//Green Color* **break**;  
 **case** Constant.***DEVICE\_OFF***:  
 **alarmStatusText**.setText(R.string.***alarmDisabledMessage***); *//Red Color* **alarmStatusText**.setTextColor(Color.*rgb*(204, 0, 0));  
 **break**;  
 **default**:  
 **alarmStatusText**.setText(R.string.***alarmNoServiceMessage***);  
 **alarmStatusText**.setTextColor(Color.*rgb*(117, 118, 120)); *//Grey Color* **break**;  
 }  
  
 */\*  
 Displays information about the break in sensor. If the alarm is enabled and the break in  
 sensor displays no "motion" the text becomes green and displays the appropriate text. If  
 the alarm is enabled and the break in sensor displays "motion", the text becomes red and  
 displays again the appropriate text. If the alarm is disabled then there is no motion data  
 to be displayed and the appropriate text is being displayed  
 \*/* String homeStatus = HttpRequests.*getRequest*(Constant.***SENSOR\_BREAK\_IN\_URI***);  
 **switch** (alarmStatus) {  
 **case** Constant.***DEVICE\_OFF***:  
 **homeStatusText**.setText(R.string.***disabledAlarmMessage***);  
 **homeStatusText**.setTextColor(Color.*rgb*(255, 136, 0));  
 **break**;  
 **case** Constant.***DEVICE\_ON***:  
 **if** (homeStatus.equals(Constant.***SENSOR\_ON***)) {  
 **homeStatusText**.setText(R.string.***alarmMotionDetectionMessage***);  
 **homeStatusText**.setTextColor(Color.*rgb*(204, 0, 0));  
 } **else** {  
 **homeStatusText**.setText(R.string.***alarmMotionDetectionMessage***);  
 **homeStatusText**.setTextColor(Color.*rgb*(102, 153, 50));  
 }  
 **break**;  
 **default**:  
 **homeStatusText**.setText(R.string.***alarmNoServiceMessage***);  
 **homeStatusText**.setTextColor(Color.*rgb*(117, 118, 120));  
 **break**;  
 }  
 }  
 });  
 }  
 },10,10, TimeUnit.***SECONDS***);  
 }  
  
 */\*  
 Method called when the activity is paused. Kills the scheduled thread  
 \*/* @Override  
 **protected void** onPause() {  
 **super**.onPause();  
  
 **scheduledExecutorService**.shutdownNow();  
  
 }  
  
 *//If the user presses the button, navigate to the corresponding activity* **private void** runActivityButton(Button aButton, **final** Class aClass){  
 **assert** aButton != **null**;  
 aButton.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 Intent anIntent = **new** Intent(HomeScreen.**this**, aClass);  
 HomeScreen.**this**.startActivity(anIntent);  
 }  
 });  
 }  
}

**package** com.smarthome.pachoumis.homeautomation.activityPackage;  
  
**import** android.graphics.Color;  
**import** android.os.StrictMode;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;  
**import** android.widget.CheckBox;  
**import** android.widget.CompoundButton;  
**import** android.widget.EditText;  
**import** android.widget.Switch;  
**import** android.widget.TextView;  
  
**import** com.smarthome.pachoumis.homeautomation.R;  
**import** com.smarthome.pachoumis.homeautomation.utilityPackage.Constant;  
**import** com.smarthome.pachoumis.homeautomation.utilityPackage.GlobalMethods;  
**import** com.smarthome.pachoumis.homeautomation.utilityPackage.HttpRequests;  
  
**import** java.util.concurrent.Executors;  
**import** java.util.concurrent.ScheduledExecutorService;  
**import** java.util.concurrent.TimeUnit;  
  
*/\*  
The LightsScreen class implements the activity that is responsible for the management  
of the smart home room lights. Has the ability to enable and disable the lights, set it  
to be enabled or disabled at a certain time of the day and/or when the light level  
percentage reaches a certain value. Displays information about the light level.  
 \*/***public class** LightsScreen **extends** AppCompatActivity {  
  
 **private** TextView **lightLevelText**;  
  
 */\*  
 Method called when the activity is created  
 \*/* @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_lights\_screen***);  
  
 *//Permit all policy for the threads* **if** (android.os.Build.VERSION.***SDK\_INT*** > 9) {  
 StrictMode.ThreadPolicy policy = **new** StrictMode.ThreadPolicy.Builder().permitAll().build();  
 StrictMode.*setThreadPolicy*(policy);  
 }  
  
 *//The view objects and their bindings to the layout xml counterpart of the activity* Switch light1Switch = (Switch) findViewById(R.id.***lsLight1Switch***);  
 Switch light2Switch = (Switch) findViewById(R.id.***lsLight2Switch***);  
 CheckBox turnOnTimeBox = (CheckBox) findViewById(R.id.***lsTurnOnTimeBox***);  
 EditText turnOnTimeField = (EditText) findViewById(R.id.***lsTurnOnTimeField***);  
 CheckBox turnOffTimeBox = (CheckBox) findViewById(R.id.***lsTurnOffTimeBox***);  
 EditText turnOffTimeField = (EditText) findViewById(R.id.***lsTurnOffTimeField***);  
 CheckBox turnOnSensorBox = (CheckBox) findViewById(R.id.***lsTurnOnSensorBox***);  
 EditText turnOnSensorField = (EditText) findViewById(R.id.***lsTurnOnSensorField***);  
 CheckBox turnOffSensorBox = (CheckBox) findViewById(R.id.***lsTurnOffSensorBox***);  
 EditText turnOffSensorField = (EditText) findViewById(R.id.***lsTurnOffSensorField***);  
 **lightLevelText** = (TextView) findViewById(R.id.***lsLightLevelText***);  
  
 *//initializing the switches for the lights and the checkboxes and fields for time and sensor value scheduling* GlobalMethods.*initializeSwitch*(Constant.***DEVICE\_LIGHT\_1\_URI***, light1Switch);  
 GlobalMethods.*initializeSwitch*(Constant.***DEVICE\_LIGHT\_2\_URI***, light2Switch);  
 GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_LIGHT\_1\_URI***,Constant.***DEVICE\_TIME\_ENABLE\_URI***, turnOnTimeBox, turnOnTimeField);  
 GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_LIGHT\_2\_URI***,Constant.***DEVICE\_TIME\_ENABLE\_URI***, turnOnTimeBox, turnOnTimeField);  
 GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_LIGHT\_1\_URI***,Constant.***DEVICE\_TIME\_DISABLE\_URI***, turnOffTimeBox, turnOffTimeField);  
 GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_LIGHT\_2\_URI***,Constant.***DEVICE\_TIME\_DISABLE\_URI***, turnOffTimeBox, turnOffTimeField);  
 GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_LIGHT\_1\_URI***,Constant.***DEVICE\_SENSOR\_ENABLE\_URI***, turnOnSensorBox, turnOnSensorField);  
 GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_LIGHT\_2\_URI***,Constant.***DEVICE\_SENSOR\_ENABLE\_URI***, turnOnSensorBox, turnOnSensorField);  
 GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_LIGHT\_1\_URI***,Constant.***DEVICE\_SENSOR\_DISABLE\_URI***, turnOffSensorBox, turnOffSensorField);  
 GlobalMethods.*initializeCheckBoxAndField*(Constant.***DEVICE\_LIGHT\_2\_URI***,Constant.***DEVICE\_SENSOR\_DISABLE\_URI***, turnOffSensorBox, turnOffSensorField);  
  
 *//methods that add functionality to the switches, the time schedule checkboxes and fields and the sensor value checkboxes and fields* GlobalMethods.*switchFunctionality*(light1Switch,Constant.***DEVICE\_LIGHT\_1\_URI***);  
 GlobalMethods.*switchFunctionality*(light2Switch,Constant.***DEVICE\_LIGHT\_2\_URI***);  
 checkBoxDualFunctionality(turnOnTimeBox, turnOnTimeField,Constant.***DEVICE\_TIME\_ENABLE\_URI***);  
 GlobalMethods.*editTextFunctionality*(turnOnTimeField, turnOnTimeBox,Constant.***TIME\_PATTERN***);  
 checkBoxDualFunctionality(turnOffTimeBox, turnOffTimeField,Constant.***DEVICE\_TIME\_DISABLE\_URI***);  
 GlobalMethods.*editTextFunctionality*(turnOffTimeField, turnOffTimeBox,Constant.***TIME\_PATTERN***);  
 checkBoxDualFunctionality(turnOnSensorBox, turnOnSensorField,Constant.***DEVICE\_SENSOR\_ENABLE\_URI***);  
 GlobalMethods.*editTextFunctionality*(turnOnSensorField, turnOnSensorBox,Constant.***PERCENTAGE\_PATTERN***);  
 checkBoxDualFunctionality(turnOffSensorBox, turnOffSensorField,Constant.***DEVICE\_SENSOR\_DISABLE\_URI***);  
 GlobalMethods.*editTextFunctionality*(turnOffSensorField, turnOffSensorBox,Constant.***PERCENTAGE\_PATTERN***);  
 }  
  
 */\*  
 Method called when the activity is resumed. Also runs after onCreate.  
 \*/* @Override  
 **protected void** onResume() {  
 **super**.onResume();  
  
 *//Scheduling a new Thread to run every 10 seconds* ScheduledExecutorService scheduledExecutorService = Executors.*newSingleThreadScheduledExecutor*();  
 scheduledExecutorService.scheduleAtFixedRate(**new** Runnable(){  
  
 @Override  
 **public void** run() {  
 LightsScreen.**this**.runOnUiThread(**new** Runnable(){ *//This runs on User Interface level* @Override  
 **public void** run() {  
  
 *//Get light level information from the server and display it* String lightSensor = HttpRequests.*getRequest*(Constant.***SENSOR\_LIGHT\_LEVEL\_URI***);  
 **lightLevelText**.setText(lightSensor + **" %"**);  
  
 }  
 });  
 }  
 },10,10, TimeUnit.***SECONDS***);  
 }  
  
 *//same functionality as the GlobalMethods.checkBoxFunctionality but modified to accept two lights instead of a device* **private void** checkBoxDualFunctionality(**final** CheckBox checkbox, **final** EditText edittext, **final** String functionURI){  
 checkbox.setOnCheckedChangeListener(**new** CompoundButton.OnCheckedChangeListener() {  
 @Override  
 **public void** onCheckedChanged(CompoundButton buttonView, **boolean** isChecked) {  
 **if**(isChecked){  
 HttpRequests.*putRequest*(Constant.***DEVICE\_LIGHT\_1\_URI*** + functionURI,edittext.getText().toString());  
 HttpRequests.*putRequest*(Constant.***DEVICE\_LIGHT\_2\_URI*** + functionURI,edittext.getText().toString());  
 }**else**{  
 **if**(functionURI.equals(Constant.***DEVICE\_TIME\_ENABLE\_URI***) || functionURI.equals(Constant.***DEVICE\_TIME\_DISABLE\_URI***)){  
 HttpRequests.*putRequest*(Constant.***DEVICE\_LIGHT\_1\_URI*** + functionURI,Constant.***NO\_TIME\_VALUE***);  
 HttpRequests.*putRequest*(Constant.***DEVICE\_LIGHT\_2\_URI*** + functionURI,Constant.***NO\_TIME\_VALUE***);  
 }**else if**(functionURI.equals(Constant.***DEVICE\_SENSOR\_ENABLE\_URI***) || functionURI.equals(Constant.***DEVICE\_SENSOR\_DISABLE\_URI***)){  
 HttpRequests.*putRequest*(Constant.***DEVICE\_LIGHT\_1\_URI*** + functionURI,Constant.***NO\_SENSOR\_VALUE***);  
 HttpRequests.*putRequest*(Constant.***DEVICE\_LIGHT\_2\_URI*** + functionURI,Constant.***NO\_SENSOR\_VALUE***);  
 }  
 checkbox.setEnabled(**false**);  
 checkbox.setClickable(**false**);  
 checkbox.setTextColor(Color.*rgb*(117,118,120));  
 }  
 }  
 });  
 }  
  
}