ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Επέκταση της πλατφόρμας Tuya για τον έλεγχο της κατανάλωσης σε έξυπνες οικίες

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Χαράλαμπος Ν. Δάλπης

Επιβλέπων: Παναγιώτης Τσανάκας

Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2023

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Επέκταση της πλατφόρμας Tuya για τον έλεγχο της κατανάλωσης σε έξυπνες οικίες

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Χαράλαμπος Ν. Δάλπης

Επιβλέπων: Παναγιώτης Τσανάκας
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 1η Νοεμβρίου 2023

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
……..	……..	……..
Παναγιώτης Τσανάκας	Γεώργιος Κορρές	Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2023

……

Χαράλαμπος Ν. Δάλπης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η πλατφόρμα Tuya είναι μια ευέλικτη και ανοιχτή πλατφόρμα για το Διαδίκτυο των Πραγμάτων η οποία επιτρέπει τη σύνδεση και τον έλεγχο διάφορων έξυπνων συσκευών σε μια κοινή ενοποιημένη υποδομή. Στο μοντέλο της δυναμικής τιμολόγησης, το οποίο αναμένεται να ξεκινήσει να εφαρμόζεται στο επόμενο χρονικό διάστημα και στην Ελλάδα, η τιμή της ηλεκτρικής ενέργειας μεταβάλλεται ανά ώρα και για κάθε εικοσιτετράωρο ανακοινώνεται από μία μέρα έως κάποιες ώρες νωρίτερα. Σκοπός της παρούσας έρευνας είναι ο σχεδιασμός και η ανάπτυξη μιας υπηρεσίας/εφαρμογής για κινητές συσκευές Android η οποία αξιοποιώντας τα εργαλεία και τις δυνατότητες που παρέχει η πλατφόρμα Tuya υλοποιεί έναν έξυπνο αλγόριθμο απόφασης ο οποίος αναλαμβάνει τη χρονική μετάθεση της λειτουργίας του ηλεκτρικού θερμοσίφωνα της οικίας σε ώρες της ημέρας όπου οι ενεργειακές απαιτήσεις του δικτύου και κατά συνέπεια και οι χρεώσεις του ηλεκτρικού ρεύματος είναι χαμηλότερες. Για τη βελτιστοποίηση των προβλέψεων του αλγορίθμου χρησιμοποιήθηκαν στατιστικές μέθοδοι καθώς και μέθοδοι πρόβλεψης χρονοσειρών. Παράλληλα, δημιουργήθηκε περιβάλλον επίβλεψης της λειτουργίας της υπηρεσίας για τον χρήστη από τον υπολογιστή (web server) και πραγματοποιήθηκαν δοκιμές σε πραγματικές συνθήκες με σκοπό την αξιολόγηση της αποτελεσματικότητας της υπηρεσίας. Τα συμπεράσματα είναι ενθαρρυντικά καθώς φαίνεται η υπηρεσία να μειώνει σημαντικά το κόστος ενέργειας συμβάλλοντας παράλληλα στην αποφόρτιση του δικτύου κατά τις ώρες αιχμής. Για την ανάπτυξη της υπηρεσίας/εφαρμογής αξιοποιήθηκαν οι γλώσσες προγραμματισμού Python και Java με τις βιβλιοθήκες τους Tuya IoT SDK και το προγραμματιστικό περιβάλλον Android Studio.

Λέξεις Κλειδιά

Έξυπνα σπίτια, Έξυπνη Συσκευή, Ηλεκτρικός θερμοσίφωνας, Πλατφόρμα Tuya, Python, Java, Δυναμική τιμολόγηση

Abstract

The Tuya platform is a flexible and open platform for the Internet of Things that enables the connection and control of various smart devices on a common unified infrastructure. In the dynamic pricing model, which is expected to start being implemented in Greece in the following months, the price of electricity changes hourly and for every twenty-four hours it is announced from one day to several hours in advance. The purpose of this research is the design and development of a service/application for Android mobile devices which by utilizing the tools and capabilities that the Tuya platform provides, implements an intelligent decision algorithm that undertakes the time shift of the operation of the electric water heater of the smart house in the hours of the day when the energy requirements of the network and consequently the electricity charges are lower. Statistical methods, as well as time series forecasting methods, were used to optimize the algorithm's predictions. At the same time, an environment was created for the user to supervise the operation of the service from a computer (web server) and tests were carried out in actual conditions to evaluate the effectiveness of the service. The conclusions are encouraging as the service appears to significantly reduce energy costs and at the same time contributes to offloading the network during peak hours. For the development of the service/application, the Python and Java programming languages with their Tuya IoT SDK libraries and the Android Studio programming environment were used.

Keywords

Smart Home, Smart Device, Electric Water Heater, Tuya Platform, Python, Java, Dynamic Pricing

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου κ. Παναγιώτη Τσανάκα για την πολύτιμη καθοδήγηση, την άμεση ανταπόκρισή του και την υπομονή του καθ' όλη τη διάρκεια της εκπόνησης της διπλωματικής μου εργασίας.

Επίσης, ευχαριστώ ειλικρινά τον υποψήφιο διδάκτορα κ. Κωνσταντίνο Αθανασιάδη για την ουσιαστική του βοήθεια και τη συνεχή ενθάρρυνση που μου παρείχε σε όλα τα στάδια της διπλωματικής μου εργασίας.

Ακόμα, θα ήθελα να ευχαριστήσω την οικογένειά μου για την κατανόηση και την υπομονή που έδειξαν όλο αυτό το διάστημα και ιδιαίτερα τη μητέρα μου η οποία στήριξε με όλες της τις δυνάμεις την προσπάθειά μου.

Τέλος, ιδιαίτερες ευχαριστίες οφείλω στους φίλους μου για την ψυχολογική υποστήριξη που μου παρείχαν και που ήταν πάντα κοντά μου όταν τους χρειαζόμουν.

Ευρετήριο Εικόνων

Περιεχόμενα

Α. ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

ΚΕΦΑΛΑΙΟ 1 - Προβληματική της έρευνας

1.1 Έξυπνο σπίτι

Στην ενότητα αυτή πραγματοποιείται μια εισαγωγή στην έννοια του Έξυπνου Σπιτιού (Smart Home). Συγκεκριμένα, δίνονται οι σημαντικότεροι ορισμοί όπως αυτοί διατυπώνονται στη σχετική βιβλιογραφία και στη συνέχεια η ενότητα εστιάζει στις εφαρμογές/υπηρεσίες των Έξυπνων Σπιτιών και στα πλεονεκτήματα και μειονεκτήματά τους.

1.1.1 Ορισμοί

Σύμφωνα με τον Lutolf [1] το Έξυπνο Σπίτι ορίζεται ως ένα σπίτι που ενσωματώνει διαφορετικές υπηρεσίες με τη χρήση ενός κοινού συστήματος επικοινωνίας και περιλαμβάνει υψηλό βαθμό έξυπνης λειτουργικότητας και ευελιξίας, εξασφαλίζοντας την οικονομική, ασφαλή και άνετη λειτουργία του. Ο Aldrich [2] ορίζει το Έξυπνο Σπίτι ως μια κατοικία εξοπλισμένη με τεχνολογία η οποία μπορεί να προβλέπει και να ανταποκρίνεται στις ανάγκες των ενοίκων της, προωθώντας την άνεση, την ευκολία, την ασφάλεια και την ψυχαγωγία τους μέσω διαχείρισης μέσα από το σπίτι ή και απομακρυσμένα. Οι De Silva et al. [3] θεωρούν ότι Έξυπνο Σπίτι είναι το σπίτι που διαθέτει ευφυΐα περιβάλλοντος και αυτοματισμούς ώστε να μπορεί να ανταποκρίνεται στη συμπεριφορά των κατοίκων του και να τους παρέχει διάφορες διευκολύνσεις. Κατά τους Balta-Ozkan et al. [4] Έξυπνο Σπίτι είναι μία κατοικία εξοπλισμένη με δίκτυο επικοινωνιών η οποία διασυνδέει αισθητήρες και συσκευές ώστε αυτές να παρακολουθούνται και να ελέγχονται από απόσταση ανάλογα με τις ανάγκες των ενοίκων της. Οι Hargreaves και Wilson [5] αναφέρουν ότι ένα Έξυπνο Σπίτι συλλέγει και αναλύει δεδομένα για το τοπικό περιβάλλον, μεταδίδει πληροφορίες σε χρήστες (και παρόχους υπηρεσιών) και δίνει τη δυνατότητα διαχείρισης διαφορετικών λειτουργιών (π.χ. θέρμανση, φωτισμός, ψυχαγωγία). Κατά τους Shin et al. [6] το Έξυπνο Σπίτι είναι ένα έξυπνο περιβάλλον το οποίο είναι σε θέση να μαθαίνει από τις συνήθειες των κατοίκων του και να χρησιμοποιεί αυτή τη γνώση προκειμένου να προσαρμόζεται και να ανταποκρίνεται στους στόχους της άνεσης και της αποτελεσματικότητας. Τέλος, οι Gram-Hanssen και Darby [7] ορίζουν το Έξυπνο Σπίτι ως ένα σπίτι στο οποίο ένα δίκτυο επικοινωνιών συνδέει αισθητήρες, συσκευές και χειριστήρια ώστε να επιτρέπει την απομακρυσμένη παρακολούθηση από τους ενοίκους του, ενώ οι Marikyan et al. [8] ως μια κατοικία εξοπλισμένη με έξυπνες τεχνολογίες που στοχεύουν στην παροχή εξατομικευμένων υπηρεσιών για τους χρήστες.

Συμπερασματικά, μπορούμε να πούμε ότι Έξυπνο Σπίτι είναι μια κατοικία που χρησιμοποιεί έξυπνες συσκευές και τεχνολογία για την αυτοματοποίηση των εργασιών και τη βελτίωση της ποιότητας ζωής των κατοίκων του. Οι συσκευές συνδέονται σε έναν κεντρικό κόμβο και οι ιδιοκτήτες του σπιτιού μπορούν να τις ελέγχουν εξ αποστάσεως χρησιμοποιώντας μία κινητή συσκευή (smartphone ή tablet) ή έναν φωνητικό βοηθό. Μπορεί κάποιος να δημιουργήσει ένα ολοκληρωτικά έξυπνο σπίτι, ή ένα σπίτι που χρησιμοποιεί κάποιες έξυπνες λύσεις και αυτοματισμούς. Όσες

περισσότερες smart λύσεις ενσωματώνει κανείς σε έναν χώρο, τόσες περισσότερες συσκευές μπορούν να συνδέονται και να επικοινωνούν μεταξύ τους διαμορφώνοντας ιδανικές συνθήκες ζωής στο σπίτι .

1.1.2. Εφαρμογές Έξυπνου Σπιτιού

Οι εφαρμογές – υπηρεσίες που μπορεί να παρέχει ένα Έξυπνο Σπίτι φέρνουν πραγματική επανάσταση στον τρόπο που ζούμε και περιορίζονται μόνο από τη φαντασία μας. Δεν είναι τυχαίο άλλωστε ότι υπάρχουν σήμερα 300 εκατομμύρια Έξυπνα Σπίτια παγκοσμίως [9]. Ο κάθε ιδιοκτήτης μπορεί να πραγματοποιήσει στην κατοικία του όποιες αυτοματοποιήσεις επιλέξει (πολλαπλά σενάρια) βάσει χρονικών προγραμμάτων, σε συνδυασμό με τις καιρικές συνθήκες, τις προσωπικές του επιθυμίες κλπ και αναφορικά με όλες τις ανάγκες λειτουργίας του σπιτιού (θέρμανση, ψύξη, φωτισμό, αερισμό, ασφάλεια, άνεση και προστασία) (βλ. Εικόνα 1). Ακολουθούν μερικά παραδείγματα:

Εικόνα 1. Έξυπνο σπίτι

Αυτοματισμοί Ασφάλειας Σπιτιού: Οι δραστηριότητες μέσα και γύρω από το σπίτι μπορούν να παρακολουθούνται από απόσταση και να δημιουργούνται συνθήκες πανικού εφόσον χρειαστεί (άναμμα όλων των φωτιστικών, ανέβασμα ρολών, ενεργοποίηση σειρήνας, προσομοίωση ανθρώπινης παρουσίας, προσομοίωση ύπαρξης σκύλων μέσα στο σπίτι κλπ).

Αυτοματισμοί Φωτισμού: Όλα τα φωτιστικά του σπιτιού μπορούν να ανάβουν, να σβήνουν και να αυξομειώνουν την ένταση (dim) αυτόματα βάσει προγράμματος ή δραστηριότητας, από όπου και αν βρισκόμαστε.

Αυτοματισμοί Θερμοσίφωνα, Συστήματος Ποτίσματος & Ηλεκτρικών Συσκευών: Όλες οι συνδεδεμένες ηλεκτρικές συσκευές μπορούν να ενεργοποιούνται και να απενεργοποιούνται με το πάτημα ενός διακόπτη, ακόμη και απομακρυσμένα.

Αυτοματισμοί Θέρμανσης και Ψύξης: Η θέρμανση και η ψύξη του σπιτιού μπορούν να ρυθμίζονται από απόσταση, όχι απλά σαν λειτουργία on/off, αλλά αναλαμβάνοντας όλες τις λειτουργίες ρύθμισης, που μπορεί να εκτελέσει ο τηλεχειρισμός του κλιματιστικού. Μπορεί να προγραμματιστεί εκ των προτέρων η αλλαγή θερμοκρασίας στο σπίτι ανάλογα με την εποχή, την ώρα της ημέρας, ή ακόμα τις προσωπικές προτιμήσεις των ενοίκων, με ένα άγγιγμα ή μια απλή φωνητική εντολή.

Αυτοματισμοί Ρολών και Τεντών (Σκίασης): Τα ρολά και οι τέντες μπορούν να ανεβαίνουν και να κατεβαίνουν μεμονωμένα ή σε ομάδες με το πάτημα ενός διακόπτη. Η σκίαση τις κατάλληλες ώρες οδηγεί σε μειωμένες θερμοκρασίες στο εσωτερικό του σπιτιού και άρα μείωση των αναγκών για ψύξη.

Αυτοματισμοί Ήχου και Εικόνας: Οι συσκευές αναπαραγωγής ήχου κι εικόνας του σπιτιού ρυθμίζονται εύκολα ώστε η αγαπημένη μουσική ή εκπομπή/ταινία των ενοίκων να ξεκινάει τη στιγμή που το επιθυμούν. Επίσης, ο έλεγχος ανά πάσα στιγμή αν το ραδιόφωνο ή/και η τηλεόραση είναι ή όχι σε λειτουργία αποτελεί μία απλή διαδικασία, εφικτή ακόμα και αφού κάποιος έχει φύγει από το σπίτι [10][11].

1.1.3 Πλεονεκτήματα - Μειονεκτήματα

Από όσα προαναφέρθηκαν είναι προφανή τα πλεονεκτήματα της τεχνολογίας του Έξυπνου Σπιτιού.

Το βασικότερο όφελος φαίνεται να είναι η εξοικονόμηση ενέργειας και χρημάτων. Οι έξυπνες συσκευές ρυθμίζονται να λειτουργούν συγκεκριμένες ώρες της ημέρας και να κλείνουν αυτόματα μόνες τους όταν δεν χρησιμοποιούνται. Επιπλέον, είναι δυνατή η παρακολούθηση της κατανάλωσης ενέργειας για κάθε ξεχωριστή συσκευή μειώνοντας ακόμη περισσότερο τον μηνιαίο λογαριασμό.

Η πρόσθετη ασφάλεια και η προστασία που μπορεί να παρέχει ένα Έξυπνο Σπίτι είναι επίσης ένα σημαντικό πλεονέκτημα. Με τα φώτα, τους ανιχνευτές κίνησης, τις κάμερες παρακολούθησης, τις έξυπνες κλειδαριές και τους συναγερμούς γίνεται εφικτή η διαχείριση των εσωτερικών και των εξωτερικών χώρων του σπιτιού απομακρυσμένα, αποτρέποντας επίδοξους διαρρήκτες και επιτρέποντας την είσοδο μόνο σε έμπιστα άτομα. Επιπλέον, κατάλληλα συστήματα σε ένα Έξυπνο Σπίτι επιτρέπουν την παρακολούθηση του ηλεκτρικού κυκλώματος εντός της κατοικίας και διακόπτουν το ρεύμα σε περίπτωση βραχυκυκλώματος. Τέλος, ειδικοί αισθητήρες μπορούν να εντοπίσουν τη διαρροή νερού και αερίου καθώς και ενδείξεις καπνού ειδοποιώντας τους ιδιοκτήτες.

Η ευκολία που προσφέρουν οι αυτοματισμοί ενός Έξυπνου Σπιτιού είναι ένα ακόμα μεγάλο όφελος για τους ενοίκους του συμβάλλοντας σε ένα υψηλό επίπεδο ζωής. Το

γεγονός ότι οι ένοικοι μπορούν να θέτουν σε λειτουργία τις συνδεδεμένες συσκευές με το πάτημα ενός κουμπιού ή ακόμα και μόνο με τη φωνή τους καθιστά πιο άνετη την καθημερινότητά τους. Για παράδειγμα, ένα πλήκτρο «Φεύγω» μπορεί να κλείσει τα ρολά, να οπλίσει το σύστημα ασφάλειας και να θέσει σε λειτουργία το πλυντήριο όταν θα ξεκινήσει το νυχτερινό τιμολόγιο.

Τέλος, στα θετικά της τεχνολογίας του Έξυπνου Σπιτιού συγκαταλέγεται και η επεκτασιμότητά του. Η αναβάθμιση μιας κατοικίας μπορεί να πραγματοποιηθεί ανά πάσα στιγμή και ανάλογα με τις οικονομικές δυνατότητες του ιδιοκτήτη της, με την σταδιακή προσθήκη τεχνολογίας σε όλο και περισσότερες συσκευές.

Αναφορικά με τα μειονεκτήματα του Έξυπνου Σπιτιού αναφέρονται στη βιβλιογραφία η εξοικείωση των ενοίκων του με την τεχνολογία η οποία ειδικά στην αρχή μπορεί να τους δημιουργήσει δυσκολίες στον χειρισμό των έξυπνων συσκευών και να μην τους επιτρέπει να εκμεταλλευτούν στο μέγιστο τις δυνατότητες που προσφέρουν προκειμένου να απολαύσουν όλα τα οφέλη. Ένα ακόμα μειονέκτημα είναι το υψηλό κόστος που συνεπάγεται η εγκατάσταση έξυπνων συσκευών λόγω της χρήσης προηγμένης τεχνολογίας σε αυτές. Επιπλέον, σημαντικά είναι και τα ποσά που θα πρέπει να δαπανούνται για την επισκευή και τη συντήρηση αυτών των συσκευών [12] [13] [14].

1.2 Πλατφόρμες για Έξυπνα Σπίτια

Στην ενότητα αυτή αρχικά εξηγείται η έννοια της πλατφόρμας για Έξυπνα Σπίτια, επισημαίνονται τα κύρια χαρακτηριστικά της, αναφέρονται οι πιο διαδεδομένες πλατφόρμες και τέλος παρουσιάζεται η πλατφόρμα Tuya και τα πλεονεκτήματά της.

1.2.1 Ο όρος «Πλατφόρμα»

Προκειμένου να γίνει κατανοητή η έννοια της πλατφόρμας ενός Έξυπνου Σπιτιού είναι σημαντικό να αποσαφηνιστεί η διαφορά μεταξύ ενός αυτοματοποιημένου σπιτιού και ενός Έξυπνου Σπιτιού, η οποία προσδιορίζεται στον τρόπο με τον οποίο ενσωματώνεται η εγκατεστημένη τεχνολογία. Σε ένα αυτοματοποιημένο σπίτι, μπορεί να υπάρχουν πολλές έξυπνες συσκευές οι οποίες όμως ελέγχονται από τηλεχειριστήρια ή ανεξάρτητες εφαρμογές στην κινητή συσκευή (smartphone ή tablet). Η διαχείριση όλων αυτών των ξεχωριστών συσκευών μπορεί να είναι περίπλοκη και ενοχλητική και συχνά, ένας ιδιοκτήτης σπιτιού διαπιστώνει ότι ενώ έχει έξυπνες συσκευές, δεν έχει Έξυπνο Σπίτι. Για να μετατραπεί ένα σπίτι με αυτοματισμούς σε ένα Έξυπνο Σπίτι απαιτείται μια ενιαία πλατφόρμα αυτοματισμού για όλο το σπίτι, έτσι ώστε ο χρήστης να χρειάζεται να μάθει να χρησιμοποιεί μόνο ένα σύστημα και όχι μια ξεχωριστή εφαρμογή για κάθε έξυπνη συσκευή [15].

Οι πλατφόρμες για Έξυπνα Σπίτια (Smart Home Platforms), επίσης γνωστές ως οικοσυστήματα (Ecosystems) για Έξυπνα Σπίτια είναι πλαίσια υλικού ή/και λογισμικού τα οποία παρέχουν ένα ενοποιημένο περιβάλλον διεπαφής, καθώς και υποδομές για τον έλεγχο και τη διαχείριση πολλαπλών συσκευών εντός ενός Έξυπνου

Σπιτιού. Οι πλατφόρμες αυτές επιτρέπουν την ενσωμάτωση και την αυτοματοποίηση λειτουργιών μεταξύ διαφορετικών συσκευών, πρωτοκόλλων και υπηρεσιών, παρέχοντας εξαιρετική εμπειρία χρήστη στο Έξυπνο Σπίτι.

Μερικές από τις πιο δημοφιλείς πλατφόρμες για Έξυπνα Σπίτια είναι οι εξής:

- Amazon Alexa
- Google Home
- Apple Home
- Samsung SmartThings
- Home Assistant
- Tuya Smart [16]

1.2.2 Χαρακτηριστικά Πλατφόρμας

Κατά τη δημιουργία ενός Έξυπνου Σπιτιού η επιλογή της πλατφόρμας που θα χρησιμοποιηθεί αποτελεί μία πολύ σημαντική απόφαση που θα πρέπει να ληφθεί πρωταρχικά. Ακολουθούν τα σημαντικότερα χαρακτηριστικά που είναι χρήσιμο να ληφθούν υπόψιν:

- Διαλειτουργικότητα (Interoperability): Η πλατφόρμα εξασφαλίζει την συμβατότητα και διαλειτουργικότητα μεταξύ των διαφορετικών έξυπνων συσκευών και συστημάτων, ακόμα κι αν προέρχονται από διαφορετικούς κατασκευαστές.

- Ευκολία στην Χρήση (Ease of Use): Η πλατφόρμα παρέχει στον χρήστη ένα ενοποιημένο περιβάλλον διεπαφής, όπως για παράδειγμα μία εφαρμογή, η οποία επιτρέπει τον έλεγχο και τη διαχείριση όλων των συνδεδεμένων συσκευών από ένα σημείο ελέγχου.

- Ευκολία στην Αυτοματοποίηση (Seamless Automation): Επιλέγοντας ένα οικοσύστημα, ο χρήστης μπορεί να δημιουργήσει σύνθετες ρουτίνες και σενάρια αυτοματισμού μέσα στο Έξυπνο Σπίτι. Για παράδειγμα, μπορεί να ρυθμίσει ρουτίνες που προσαρμόζουν αυτόματα τις ρυθμίσεις φωτισμού, θερμοκρασίας και ασφάλειας, βάσει συγκεκριμένων ενεργειών ή χρονοδιαγραμμάτων.

- Υποστήριξη και Ενημερώσεις (Support and Updates): Η επιλογή ενός δημοφιλούς οικοσυστήματος Έξυπνου Σπιτιού εξασφαλίζει συνεχή υποστήριξη, τακτικές ενημερώσεις υλικολογισμικού και πρόσβαση σε νέες δυνατότητες και βελτιώσεις. Αυτό βοηθά στην διατήρηση της ασφάλειας και της αξιοπιστίας του έξυπνου οικοσυστήματος.

- Ενσωμάτωση Φωνητικών Βοηθών (Integration with Voice Assistant): Πολλά έξυπνα οικοσυστήματα παρέχουν ενσωμάτωση δημοφιλών φωνητικών βοηθών όπως το Amazon Alexa ή το Google Assistant, επιτρέποντας τον έλεγχο των έξυπνων συσκευών χρησιμοποιώντας φωνητικές εντολές.

- Κοινότητα και Υποστήριξη Χρηστών (Community and User Support): Η επιλογή ενός ευρέως διαδεδομένου οικοσυστήματος Έξυπνου Σπιτιού καθιστά τον χρήστη μέρος μιας ευρύτερης κοινότητας χρηστών. Αυτή η κοινότητα μπορεί να παρέχει πολύτιμους πόρους, συμβουλές και υποστήριξη για την αντιμετώπιση προβλημάτων και την ανακάλυψη νέων τρόπων βελτίωσης της εμπειρίας του Έξυπνου Σπιτιού [17].

1.2.3 Η Πλατφόρμα Tuya

Η πλατφόρμα Tuya είναι μια ευέλικτη και ανοιχτή πλατφόρμα για το Διαδίκτυο των Πραγμάτων (Internet of Things - IoT) που επιτρέπει τη σύνδεση και τον έλεγχο διάφορων έξυπνων συσκευών σε μια κοινή ενοποιημένη υποδομή (βλ. Εικόνα 2). Η Tuya προσφέρει λογισμικό ανάπτυξης (SDKs) και συσκευές που είναι συμβατές με την πλατφόρμα της, επιτρέποντας στους κατασκευαστές συσκευών να ενσωματώνουν εύκολα την τεχνολογία IoT στα προϊόντα τους [18].

Οι συσκευές που βασίζονται στην τεχνολογία Tuya καλύπτουν ένα ευρύ φάσμα κατηγοριών, συμπεριλαμβανομένων των φώτων, των ηλεκτρικών συσκευών, των αισθητήρων, των καλωδίων και άλλων έξυπνων συσκευών για το σπίτι. Οι χρήστες μπορούν να συνδέσουν αυτές τις συσκευές με την πλατφόρμα Tuya για να τις ελέγχουν από ένα κεντρικό σημείο, να προγραμματίσουν σενάρια αυτοματισμού και να εκτελούν διάφορες λειτουργίες μέσω εφαρμογών.

1.2.4 Πλεονεκτήματα της Πλατφόρμας Tuya

Η πλατφόρμα Tuya διαθέτει πληθώρα πλεονεκτημάτων, τόσο για τον χρήστη που θα την επιλέξει για το οικοσύστημα του Έξυπνου Σπιτιού, όσο και για τον προγραμματιστή ο οποίος θα αναπτύξει τεχνολογίες IoT πάνω σε αυτήν.

Κάποια από τα βασικά πλεονεκτήματα της πλατφόρμας Tuya τα οποία την καθιστούν μία δημοφιλή επιλογή για τους ενοίκους ενός Έξυπνου Σπιτιού είναι:

- Μεγάλο Εύρος Συμβατών Συσκευών (Wide Device Compatibility): Η πλατφόρμα Tuya υποστηρίζει ένα ευρύ φάσμα έξυπνων συσκευών από διάφορους κατασκευαστές. Αυτό παρέχει στους χρήστες πληθώρα επιλογών και συμβατότητα κατά την επιλογή έξυπνων συσκευών για το σπίτι τους.

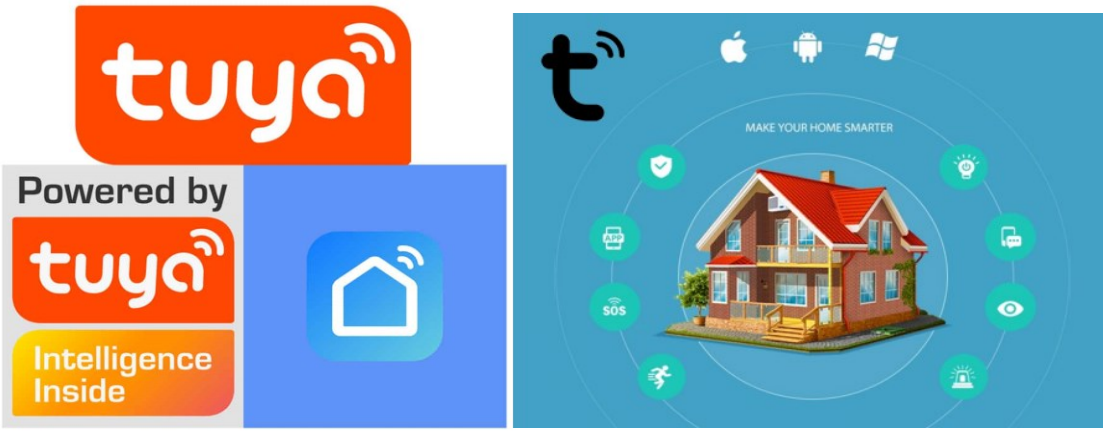
- Ενοποιημένος Έλεγχος και Διαχείριση (Unified Control and Management): Η πλατφόρμα Tuya παρέχει ένα ενοποιημένο περιβάλλον διεπαφής υπό μορφή εφαρμογής για κινητές συσκευές το οποίο επιτρέπει στους χρήστες να ελέγχουν και να διαχειρίζονται τις συνδεδεμένες συσκευές τους από ένα μόνο σημείο.

Αυτό απλοποιεί την εμπειρία του χρήστη παρέχοντας ένα κεντρικό σημείο ελέγχου για όλες τις συμβατές συσκευές Tuya.

- Ενσωμάτωση Φωνητικών Βοηθών (Integration with Voice Assistants): Η πλατφόρμα Tuya υποστηρίζει δημοφιλείς φωνητικούς βοηθούς, όπως ο Amazon Alexa και ο Google Assistant. Αυτό επιτρέπει στους χρήστες να ελέγχουν τις συσκευές Tuya με φωνητικές εντολές, προσθέτοντας ένα επιπλέον επίπεδο ευκολίας και αυτόματης λειτουργίας χωρίς να απαιτείται χρήση των χεριών.

- Σύνδεση με το Cloud και Απομακρυσμένη Πρόσβαση (Cloud Connectivity and Remote Access): Η αρχιτεκτονική βασισμένη στο Cloud της Tuya επιτρέπει την απομακρυσμένη πρόσβαση στις έξυπνες συσκευές. Οι χρήστες μπορούν να ελέγχουν και να παρακολουθούν τις συσκευές τους από οπουδήποτε χρησιμοποιώντας σχετική εφαρμογή σε κινητές συσκευές ή τη διαδικτυακή διεπαφή της Tuya.

- Εξατομίκευση και Αυτοματισμός (Customization and Automation): Η πλατφόρμα Tuya δίνει τη δυνατότητα στους χρήστες να εξατομικεύουν και να αυτοματοποιούν το Έξυπνο Σπίτι τους σύμφωνα με τις προτιμήσεις τους. Η πλατφόρμα επιτρέπει τη δημιουργία προσαρμοσμένων σεναρίων και αυτοματισμών, καθορίζοντας δράσεις βάσει των αναγκών του χρήστη. Αυτό επιτρέπει στους χρήστες να δημιουργήσουν εξατομικευμένες εμπειρίες και να βελτιστοποιήσουν τις καθημερινές τους ρουτίνες στο Έξυπνο Σπίτι τους.



Εικόνα 2. Tuya

Όσον αφορά τους προγραμματιστές οι οποίοι θα επιλέξουν την πλατφόρμα Tuya για να αναπτύξουν προϊόντα και υπηρεσίες που χρησιμοποιούν τεχνολογία IoT αναφέρονται τα παρακάτω πλεονεκτήματα:

- Απλοποιημένη Ενσωμάτωση Συσκευών (Simplified Device Integration): Η πλατφόρμα Tuya παρέχει στους προγραμματιστές ένα σύνολο εργαλείων ανάπτυξης (SDKs) και διεπαφές προγραμματισμού (APIs) που διευκολύνουν την ενσωμάτωση έξυπνων συσκευών στην πλατφόρμα Tuya.

- Κλιμακωτή Υποδομή Cloud (Scalable Cloud Infrastructure): Η πλατφόρμα Tuya παρέχει μια ανθεκτική και κλιμακωτή υποδομή Cloud που αναλαμβάνει τη διαχείριση της συνδεσιμότητας των συσκευών, την αποθήκευση δεδομένων και την επικοινωνία μεταξύ των συσκευών και του Cloud. Οι προγραμματιστές μπορούν να αξιοποιήσουν τις υπηρεσίες Cloud της Tuya, μειώνοντας την ανάγκη για κατασκευή και διαχείριση της δικής τους υποδομής backend.

- Έτοιμα Πρότυπα Εφαρμογών (Ready-Made App Templates): Η πλατφόρμα Tuya παρέχει έτοιμα πρότυπα εφαρμογών και στοιχεία διεπαφής χρήστη, που επιταχύνουν την ανάπτυξη εφαρμογών για κινητές συσκευές που ελέγχουν τις συσκευές που είναι συμβατές με την Tuya. Οι προγραμματιστές μπορούν να χρησιμοποιήσουν αυτά τα πρότυπα ως αφετηρία και να προσαρμόσουν την εμφάνιση και τη λειτουργικότητα των εφαρμογών τους.

- Επέκταση στην Παγκόσμια Αγορά (Global Market Reach): Η Tuya έχει μια παγκόσμια παρουσία και έχει κατοχυρώσει μια σημαντική θέση στην αγορά του Έξυπνου Σπιτιού. Μέσω της ενσωμάτωσης συσκευών και της ανάπτυξης λύσεων πάνω στην πλατφόρμα Tuya, οι προγραμματιστές αποκτούν πρόσβαση σε ένα μεγάλο αριθμό χρηστών παγκοσμίως.

- Υποστήριξη και Ενημερώσεις (Ongoing Support and Updates): Η Tuya παρέχει τακτικές ενημερώσεις και υποστήριξη στους προγραμματιστές, διασφαλίζοντας τη συμβατότητα με νέες συσκευές, πρωτόκολλα και τεχνολογίες. Οι προγραμματιστές μπορούν να επωφεληθούν από τις συνεχείς βελτιώσεις της Tuya, τις ενημερώσεις ασφαλείας και την κυκλοφορία νέων χαρακτηριστικών, διατηρώντας τις λύσεις τους για Έξυπνα Σπίτια ενημερωμένες και συμβατές με τις εξελίξεις του χώρου [18].

1.3 Δυναμική τιμολόγηση ηλεκτρικής ενέργειας

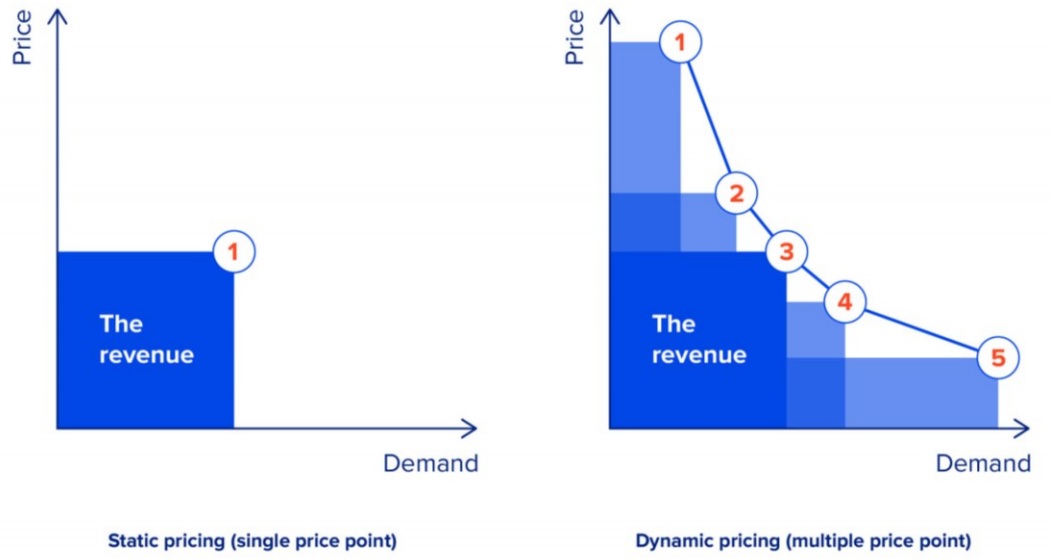
Στην ενότητα αυτή εξηγείται η έννοια της δυναμικής τιμολόγησης της ηλεκτρικής ενέργειας. Αρχικά αναφέρονται οι κυριότεροι ορισμοί και τα σημαντικότερα μοντέλα της και στη συνέχεια παρουσιάζονται οι έξυπνοι μετρητές και οι βασικές λειτουργίες τους. Στο τέλος του κεφαλαίου γίνεται αναφορά στην αναμενόμενη εφαρμογή της δυναμικής τιμολόγησης της ηλεκτρικής ενέργειας στην Ελλάδα.

1.3.1 Ορισμοί

Η εταιρεία American Airlines, δίνοντας έναν από τους πρώτους και πιο διαδεδομένους ορισμούς, προσδιόρισε τη Δυναμική Τιμολόγηση (Dynamic Pricing) ως ένα εργαλείο μεγιστοποίησης εσόδων «πωλώντας ένα κατάλληλο προϊόν, σε έναν κατάλληλο πελάτη και σε μια κατάλληλη τιμή». Αργότερα αυτός ο ορισμός συμπληρώθηκε με τις λέξεις «σε κατάλληλο χρόνο» [19]. Στη βιβλιογραφία βρίσκουμε επίσης ορισμούς που εστιάζουν στην προσφορά ή/και στη ζήτηση σύμφωνα με τους οποίους η δυναμική τιμολόγηση είναι ένα εργαλείο αξιοποίησης πόρων το οποίο ρυθμίζει την τιμή τους έτσι ώστε να επιτυγχάνεται μεγιστοποίηση των εσόδων ή των κερδών [20].

Σύμφωνα με το [21] και σχετικά με την παροχή ηλεκτρικής ενέργειας η Δυναμική Τιμολόγηση είναι μία καινοτομία στις αγορές λιανικής που συνεχώς εξελίσσεται και αναφέρεται σε τιμές λιανικής ηλεκτρικής ενέργειας που μετακυλίουν ένα μέρος της διακύμανσης των τιμών χονδρικής στους τελικούς καταναλωτές. Καθίσταται δυνατή με την ανάπτυξη αποτελεσματικών αγορών χονδρικής και τη χρήση δεδομένων που συλλέγονται από έξυπνους μετρητές. Μέσω της Δυναμικής Τιμολόγησης οι πελάτες αποκτούν κίνητρα για να καταναλώνουν ηλεκτρική ενέργεια σε χρονικές στιγμές που η τιμή λιανικής είναι πολύ χαμηλή.

Με πιο απλά λόγια, αντί να υπάρχει σταθερή τιμή ηλεκτρικής ενέργειας κατά τη διάρκεια του εικοσιτετράωρου, η τιμή της ποικίλλει ανάλογα με την ώρα της ημέρας ή τις πραγματικές συνθήκες ζήτησης και προσφοράς στην αγορά ηλεκτρικής ενέργειας (βλ. Εικόνα 3). Οι προμηθευτές δίνουν στους καταναλωτές/πελάτες τους τιμές που δεν παραμένουν σταθερές, αλλά μεταβάλλονται ανά ώρα και μάλιστα για κάθε εικοσιτετράωρο οι τιμές αυτές ανακοινώνονται από μία μέρα έως κάποιες ώρες νωρίτερα. Έτσι οι καταναλωτές ενθαρρύνονται να μετατοπίσουν τη χρήση ηλεκτρικής ενέργειας σε ώρες εκτός αιχμής ζήτησης, λειτουργώντας ενεργοβόρες συσκευές (πχ. θερμοσίφωνα, φούρνο κλπ) όταν το ηλεκτρικό ρεύμα κοστίζει λιγότερο, ενώ παράλληλα οι επιχειρήσεις κοινής ωφέλειας και οι φορείς εκμετάλλευσης του δικτύου μπορούν να εξισορροπούν το φορτίο στο ηλεκτρικό δίκτυο πιο αποτελεσματικά και να μειώνουν την ανάγκη για ακριβή παραγωγή ενέργειας αιχμής.



Εικόνα 3. Static pricing vs Dynamic pricing

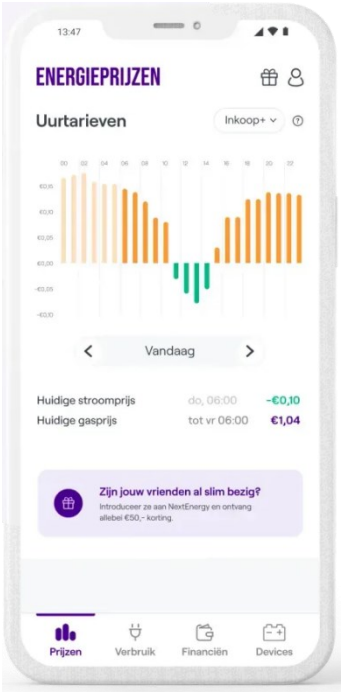
1.3.2 Μοντέλα Δυναμικής Τιμολόγησης

Υπάρχουν διαφορετικές προσεγγίσεις για τη Δυναμική Τιμολόγηση της ηλεκτρικής ενέργειας, οι σημαντικότερες από τις οποίες παρουσιάζονται παρακάτω:

- Τιμολόγηση χρόνου χρήσης / Time-of-use (TOU): Οι τιμές της ηλεκτρικής ενέργειας είναι προκαθορισμένες και ποικίλλουν κατά τις διάφορες ώρες της

ημέρας. Πιο συγκεκριμένα, είναι υψηλά στις ώρες αιχμής και χαμηλά στις υπόλοιπες ώρες της ημέρας. Η τιμολόγηση TOU μπορεί να αποτελείται από δύο ή τρία επίπεδα τιμών ανά ημέρα, ενώ υπάρχει η πιθανότητα οι τιμές να διαφέρουν και ανά εποχή.

- Τιμολόγηση κρίσιμης αιχμής / Critical peak pricing (CPP): Στις ώρες αιχμής η τιμή της ηλεκτρικής ενέργειας είναι σημαντικά υψηλότερη από την τυπική τιμή (κατά δύο, τρεις ή και περισσότερο φορές). Οι υψηλότερες αυτές τιμές αποσκοπούν στο να δώσουν κίνητρα στους καταναλωτές να μειώσουν τη χρήση ηλεκτρικής ενέργειας κατά τις ώρες αιχμής. Παράλληλα, οι επιχειρήσεις κοινής ωφέλειας αποφεύγουν την καταφυγή σε ακριβές και λιγότερο αποδοτικές μεθόδους παραγωγής ηλεκτρικής ενέργειας για την κάλυψη της ζήτησης αιχμής, όπως η χρήση μονάδων αιχμής οι οποίες είναι συχνά λιγότερο φιλικές προς το περιβάλλον.
- Τιμολόγηση σε πραγματικό χρόνο / Real Time Pricing (RTP): Με το RTP, οι τιμές ηλεκτρικής ενέργειας μπορούν να αλλάζουν συχνά (ανά ώρα ή ακόμα πιο συχνά) με βάση τις διακυμάνσεις σε πραγματικό χρόνο στις συνθήκες προσφοράς και ζήτησης ηλεκτρικής ενέργειας. Οι καταναλωτές μπορούν να έχουν πρόσβαση σε αυτές τις τιμές σε πραγματικό χρόνο και να προσαρμόζουν ανάλογα τη χρήση ηλεκτρικής ενέργειας (βλ. Εικόνα 4) [22].



Εικόνα 4. Παρακολούθηση της τιμής της ενέργειας από εφαρμογή στο κινητό (Ολλανδία)

1.3.3 Έξυπνοι μετρητές

Οι έξυπνοι μετρητές ηλεκτρικής ενέργειας είναι ψηφιακές συσκευές νέας τεχνολογίας οι οποίες παρακολουθούν και ελέγχουν την ηλεκτρική ενέργεια που καταναλώνει ένα σπίτι ή μία επιχείρηση και πρόκειται να αντικαταστήσουν σταδιακά τους υπάρχοντες αναλογικού τύπου μετρητές αναβαθμίζοντας το δίκτυο διανομής ηλεκτρικής ενέργειας

(βλ. Εικόνα 5). Οι έξυπνοι μετρητές παρέχουν στους καταναλωτές και στις εταιρίες παροχής ηλεκτρικού ρεύματος συνεχή στοιχεία της κατανάλωσης ρεύματος.

Εικόνα 5. Αναλογικός και ψηφιακός μετρητής

Μερικές από τις βασικές λειτουργίες των έξυπνων μετρητών είναι οι παρακάτω:

- Μέτρηση της ηλεκτρικής ενέργειας εξ' αποστάσεως σε πραγματικό χρόνο από τον διαχειριστή του δικτύου
- Δυνατότητα απομακρυσμένης διακοπής και επανασύνδεσης του καταναλωτή
- Ασφαλής μετάδοση των μετρήσεων
- Πρόληψη και ανίχνευση της κλοπής ρεύματος
- Εξοικονόμηση ενέργειας (ο καταναλωτής μπορεί να έχει σε πραγματικό χρόνο τα δεδομένα της κατανάλωσής του προκειμένου να προσαρμόζει την ενεργειακή του συμπεριφορά)
- Δυνατότητα παρακολούθησης και ελέγχου μέσω διαδικτύου
- Παρατήρηση κατανάλωσης και απόδοσης ενέργειας ανά ώρα, ημέρα ή οποιοδήποτε σενάριο
- Υπολογισμός και εμφάνιση κατανάλωσης με σύνολα ημέρας/εβδομάδας/μήνα
- Σύγκριση μιας δεδομένης περιόδου με μία παρόμοια στο κοντινό παρελθόν
- Ορισμός ειδοποίησης όταν η κατανάλωση περάσει τα καθορισμένα όρια [23].

1.3.4 Η δυναμική τιμολόγηση ηλεκτρικής ενέργειας στην Ελλάδα

Η οδηγία 2019/944 του Ευρωπαϊκού Κοινοβουλίου σχετικά με τους κοινούς κανόνες για την εσωτερική αγορά ηλεκτρικής ενέργειας [24] αναφέρει μεταξύ άλλων ότι «Τα κράτη μέλη θα πρέπει να μεριμνούν ώστε όλοι οι δικαιούχοι ρυθμιζόμενων τιμών να μπορούν να επωφελούνται πλήρως από τις προσφορές της ανταγωνιστικής αγοράς όταν το επιλέγουν. Για τον σκοπό αυτό, θα πρέπει να εφοδιάζονται με έξυπνα συστήματα μέτρησης και να έχουν πρόσβαση σε συμβάσεις δυναμικής τιμολόγησης ηλεκτρικής ενέργειας. Επιπλέον, θα πρέπει να ενημερώνονται άμεσα και τακτικά για τις προσφορές και τις δυνατότητες εξοικονόμησης που είναι διαθέσιμες στην ανταγωνιστική αγορά, ιδίως για τις συμβάσεις δυναμικής τιμολόγησης ηλεκτρικής ενέργειας, και θα πρέπει

να τους παρέχεται βοήθεια ώστε να αξιοποιούν και να επωφελούνται από προσφορές που βασίζονται στην αγορά».

Το ελληνικό υπουργείο Περιβάλλοντος και Ενέργειας ΥΠΕΝ προκειμένου να ενσωματώσει την παραπάνω κοινοτική οδηγία συνέταξε νομοσχέδιο το οποίο ψήφισε η Βουλή (ν.4986/2022, ΦΕΚ 204 Α΄/28.10.2022) με τις διατάξεις του οποίου καθιερώθηκε η σύμβαση δυναμικής τιμολόγησης ηλεκτρικής ενέργειας [25]. Με βάση το πλαίσιο αυτό οι προμηθευτές μπορούν να παρέχουν σύμβαση δυναμικής τιμολόγησης ηλεκτρικής ενέργειας. Οι τελικοί πελάτες που διαθέτουν έξυπνο μετρητή μπορούν να ζητούν τη σύναψη σύμβασης δυναμικής τιμολόγησης ηλεκτρικής ενέργειας από κάθε προμηθευτή που έχει περισσότερους από 200.000 τελικούς πελάτες. Επίσης, με τις διατάξεις του νομοσχεδίου εξασφαλίζεται ότι οι τελικοί πελάτες έχουν ολοκληρωμένη ενημέρωση από τους προμηθευτές σχετικά με τις ευκαιρίες, το κόστος και τους κινδύνους που συνεπάγεται αυτή η σύμβαση δυναμικής τιμολόγησης ηλεκτρικής ενέργειας και θα διασφαλίζει ότι οι προμηθευτές υποχρεούνται να ενημερώνουν καταλλήλως τους τελικούς πελάτες σχετικά με την ανάγκη εγκατάστασης κατάλληλου μετρητή ηλεκτρικής ενέργειας. Επιπλέον, θα διασφαλίζονται τα προσωπικά δεδομένα των καταναλωτών, ενώ θα παρέχονται κατάλληλες συμβουλές και πληροφορίες στους τελικούς πελάτες πριν από ή κατά τον χρόνο εγκατάστασης έξυπνων μετρητών, κυρίως σχετικά με το σύνολο των δυνατοτήτων τους όσον αφορά τη χρήση των ενδείξεων του μετρητή, την παρακολούθηση της κατανάλωσης ενέργειας, καθώς και τη συλλογή και την επεξεργασία δεδομένων προσωπικού χαρακτήρα, σύμφωνα με την ισχύουσα ενωσιακή νομοθεσία για την προστασία των δεδομένων.

Άλλωστε, τα παραπάνω προβλέπονται και στη Βίβλο Ψηφιακού Μετασχηματισμού 2020-2025 και συγκεκριμένα στο μεσοπρόθεσμο έργο «Δημιουργία συστήματος δυναμικής ενεργειακής τιμολόγησης» όπου αναφέρεται «Σκοπός του έργου είναι, εγκαθιστώντας τις κατάλληλες υποδομές υλικού (π.χ. έξυπνους μετρητές) και λογισμικού και διαμορφώνοντας το κατάλληλο θεσμικό πλαίσιο καθώς και τις απαραίτητες διαδικασίες, να είναι πλέον εφικτή η δυναμική τιμολόγηση από τους παραγωγούς και παρόχους αναλόγως των συνθηκών της αγοράς, τόσο από την πλευρά της παραγωγής και παροχής ηλεκτρικής ενέργειας, όσο και από την πλευρά της ζήτησης εκ μέρους των καταναλωτών. Το έργο αρχικά θα στοχεύσει στους καταναλωτές μέσης τάσης» [26].

Τέλος, η Ρυθμιστική Αρχή Ενέργειας (ΡΑΕ), ως υπεύθυνη να εποπτεύει την εγχώρια αγορά ενέργειας σε όλους τους τομείς της, έχει κάνει ήδη προτάσεις για τα τιμολόγια ηλεκτρικού ρεύματος και την διαμόρφωσή τους μετά το τέλος των κρατικών επιδοτήσεων που εφαρμόστηκαν λόγω της πρόσφατης ενεργειακής κρίσης (2022-23). Πρόκειται για τέσσερις κατηγορίες τιμολογίων ηλεκτρικού ρεύματος. Από 1η Οκτωβρίου 2023 αναμένεται οι καταναλωτές να μπορούν να επιλέξουν ανάμεσα σε τέσσερις τρόπους τιμολόγησης: το σταθερό τιμολόγιο, το παλιό κυμαινόμενο τιμολόγιο με ρήτρα αναπροσαρμογής, το νέο κυμαινόμενο τιμολόγιο η τιμή του οποίου θα

γνωστοποιείται την 1η κάθε μήνα και τη δυναμική τιμολόγηση που αφορά τους λίγους καταναλωτές που διαθέτουν ψηφιακούς μετρητές.

1.4 Σκοπός και Στόχοι της Έρευνας

Σκοπός της παρούσας έρευνας είναι ο σχεδιασμός και η ανάπτυξη μιας υπηρεσίας η οποία αξιοποιώντας τα εργαλεία και τις δυνατότητες που παρέχει η πλατφόρμα Tuya θα συμβάλλει στη μείωση του ενεργειακού και οικονομικού κόστους μιας έξυπνης οικίας. Η υπηρεσία αυτή θα εφαρμόζει έναν έξυπνο αλγόριθμο ο οποίος θα αναλαμβάνει τη χρονική μετάθεση της λειτουργίας των ενεργειακά κοστοβόρων συσκευών της οικίας σε ώρες της ημέρας όπου οι ενεργειακές απαιτήσεις του δικτύου και κατά συνέπεια και οι χρεώσεις του ηλεκτρικού ρεύματος είναι χαμηλότερες.

Στόχοι της έρευνας είναι:

- Η ανάπτυξη εφαρμογής IoT για λειτουργικό σύστημα Android με περιβάλλον διεπαφής για τον χρήστη η οποία θα υλοποιεί τον έξυπνο αλγόριθμο απόφασης χρησιμοποιώντας το Tuya API.
- Η δημιουργία περιβάλλοντος επίβλεψης της λειτουργίας της υπηρεσίας για τον χρήστη από τον υπολογιστή (web server).
- Η πραγματοποίηση δοκιμών σε πραγματικές συνθήκες με σκοπό την αξιολόγηση της αποτελεσματικότητας της υπηρεσίας.

1.5 Παραδοχές

Για την οικονομία της παρούσας έρευνας και με σκοπό την απλοποίηση των εξεταζόμενων περιπτώσεων, η υλοποίηση θα στηριχτεί στις παρακάτω παραδοχές:

1. Από όλες τις ενεργειακά κοστοβόρες συσκευές που υπάρχουν σε ένα Έξυπνο Σπίτι (πλυντήριο ρούχων, πλυντήριο πιάτων, ηλεκτρικός θερμοσίφωνας, air condition κλπ) επιλέγεται ο ηλεκτρικός θερμοσίφωνας.
2. Θεωρούμε ότι ο ηλεκτρικός θερμοσίφωνας διατηρεί το νερό σε αποδεκτή θερμοκρασία για χρήση για χρονικό διάστημα 6 ωρών από την στιγμή που θα τεθεί σε λειτουργία. Στην πραγματικότητα, το χρονικό αυτό διάστημα εξαρτάται από διάφορους παράγοντες όπως ο τύπος του θερμοσίφωνα, το μέγεθος του καζανιού, η μόνωσή του κ.α.
3. Καθότι το μοντέλο δυναμικής τιμολόγησης δεν εφαρμόζεται ακόμα στην Ελλάδα, θεωρούμε ότι η τιμή της κιλοβατώρας (kWh) είναι διαφορετική σε κάθε παράθυρο μίας ώρας του εικοσιτετραώρου και εξαρτάται γραμμικά από την πανελλήνια κατανάλωση ρεύματος σε μεγαβατώρες (MWh) τη συγκεκριμένη ώρα. Έτσι, ξεκινώντας από το παράθυρο 00:00 - 01:00 μέχρι και το παράθυρο 23:00-00:00 η τιμή της κιλοβατώρας λαμβάνει συνολικά 24 τιμές κάθε ημέρα.

ΚΕΦΑΛΑΙΟ 2 - Μεθοδολογίες

2.1 Η περίπτωση του ηλεκτρικού θερμοσίφωνα

Σύμφωνα με την παραδοχή 1 (βλ. Ενότητα 1.5), το ζητούμενο γίνεται πιο συγκεκριμένο και η έρευνα επικεντρώνεται στη μείωση της κατανάλωσης του ηλεκτρικού θερμοσίφωνα σε ένα Έξυπνο Σπίτι. Για να επιτευχθεί η μείωση αυτή εκμεταλλευόμαστε το μοντέλο δυναμικής τιμολόγησης ηλεκτρικού ρεύματος και επιλέγουμε να λειτουργούμε τον ηλεκτρικό θερμοσίφωνα τις ώρες της ημέρας όπου η τιμή του ηλεκτρικού ρεύματος είναι χαμηλότερη. Όμως η χρήση του θερμοσίφωνα σε ένα σπίτι είναι μια διαδικασία η οποία δεν μπορεί να μετατίθεται χρονικά, αφού αυτό θα επηρέαζε την άνεση των κατοίκων του σπιτιού. Αυτό που μπορούμε όμως να κάνουμε, δεδομένου ότι το νερό παραμένει ζεστό μέσα στο καζάνι του θερμοσίφωνα για κάποιες ώρες, είναι να επιλέξουμε να ζεστάνουμε το νερό στον θερμοσίφωνα την κατάλληλη στιγμή, την ώρα δηλαδή που η τιμή του ηλεκτρικού ρεύματος είναι χαμηλότερη. Βάσει της παραδοχής 2 (βλ. Ενότητα 1.5), το πρόσφορο χρονικό παράθυρο είναι 6 ώρες πριν την χρήση του θερμοσίφωνα.

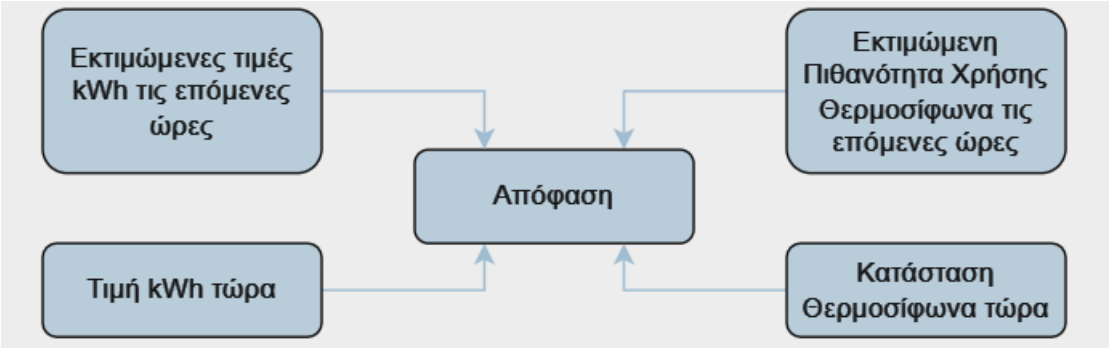
Στην παρούσα έρευνα αναπτύσσεται ένας αλγόριθμος απόφασης ο οποίος όταν εκτελείται διακρίνει αν η δεδομένη στιγμή είναι η βέλτιστη για την ενεργοποίηση του θερμοσίφωνα, ή όχι. Αφού λάβει την απόφαση πράττει αναλόγως, είτε θέτοντας τον θερμοσίφωνα σε λειτουργία (στην πρώτη περίπτωση), είτε περιμένοντας χωρίς κάποια ενέργεια (στην δεύτερη περίπτωση).

Για να μπορέσει όμως ο αλγόριθμος να αποφανθεί για το ποια είναι η βέλτιστη στιγμή για την ενεργοποίηση ή μη του θερμοσίφωνα είναι απαραίτητο να γνωρίζει την τιμή του ρεύματος στο παρόν και στις επόμενες 6 ώρες, καθώς και εάν πρόκειται να πραγματοποιηθεί χρήση του θερμοσίφωνα μέσα στις επόμενες 6 ώρες. Προφανώς, τα δεδομένα αυτά δεν είναι δυνατόν να είναι γνωστά, για τον λόγο αυτό η προσέγγιση θα στηριχτεί σε εκτιμήσεις και προβλέψεις. Οι προβλέψεις αυτές προκύπτουν από ανάλυση παλαιότερων δεδομένων, όπως οι προηγούμενες τιμές του ηλεκτρικού ρεύματος και οι συνήθειες των κατοίκων του σπιτιού όσον αφορά τη χρήση του ηλεκτρικού θερμοσίφωνα και αξιοποιούν μεθόδους που περιγράφονται στις επόμενες ενότητες.

Συμπερασματικά, η προσέγγιση θα βασιστεί σε τέσσερις παράγοντες που είναι απαραίτητοι για την απόφαση του αλγορίθμου (βλ. Εικόνα 6):

- Την τιμή του ηλεκτρικού ρεύματος τώρα, η οποία είναι δημόσια στην ιστοσελίδα της ΔΕΔΗΕ.
- Το ιστορικό προηγούμενων τιμών του ηλεκτρικού ρεύματος, από το οποίο θα προκύψει η πρόβλεψη για τις μελλοντικές τιμές.
- Το ιστορικό χρήσης του θερμοσίφωνα στο σπίτι, από το οποίο θα προκύψει η πιθανότητα χρήσης του θερμοσίφωνα για τις επόμενες 6 ώρες, βάσει της παραδοχής 2.

- Την κατάσταση του θερμοσίφωνα την δεδομένη στιγμή.



Εικόνα 6. Παράγοντες λήψης απόφασης

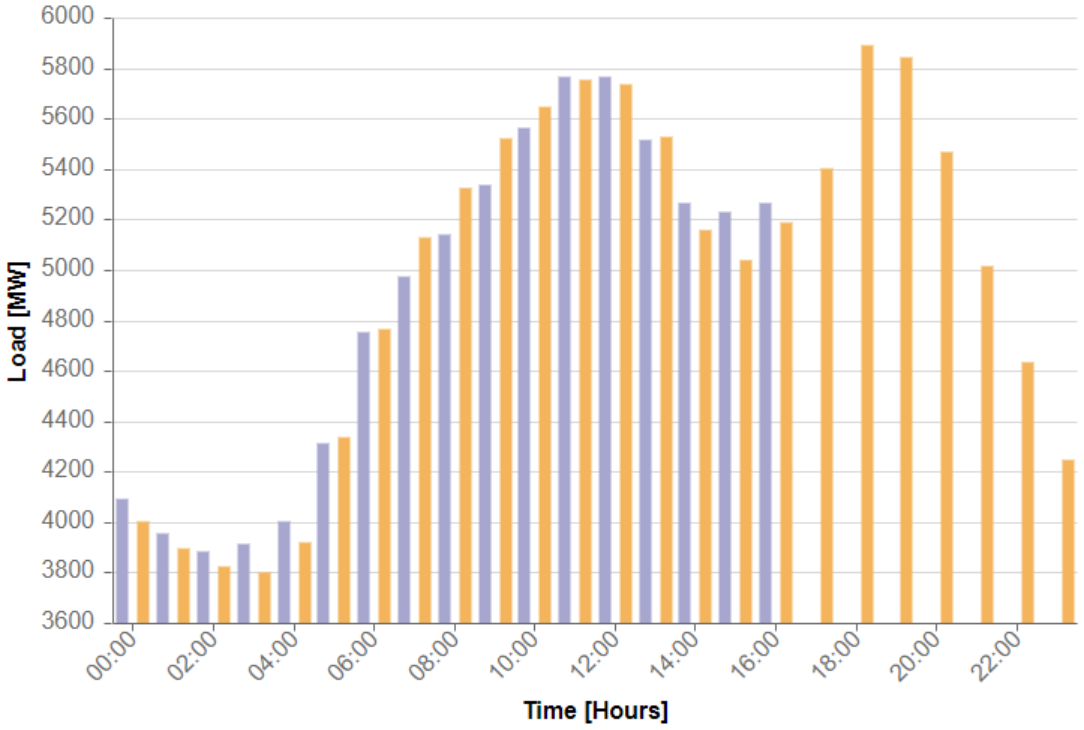
2.2 Εκτίμηση Τιμής Ενέργειας σε Μελλοντικό Χρόνο

Για να μπορέσει ο αλγόριθμος να αποφανθεί για το πότε είναι η πιο συμφέρουσα στιγμή βάσει κόστους kWh να τεθεί σε λειτουργία ο θερμοσίφωνας θα πρέπει, εκτός από την τιμή της kWh τη δεδομένη στιγμή, να γνωρίζει επίσης τις τιμές της kWh για τις επόμενες 6 ώρες. Επειδή όμως οι τιμές αυτές δεν είναι γνωστές, αφού υπολογίζονται και ορίζονται δυναμικά κάθε ώρα από τον πάροχο, θα χρειαστεί να γίνει μία πρόβλεψη για αυτές.

Για τον σκοπό αυτό θα χρησιμοποιηθούν οι προηγούμενες καταναλώσεις από το ιστορικό της πανελλήνιας κατανάλωσης ηλεκτρικής ενέργειας σε MWh ανά ώρα για κάθε ημέρα οι οποίες είναι διαθέσιμες στην ιστοσελίδα της ENTSO-E [27]. Παρατηρώντας τα διαγράμματα βλέπουμε ότι η κατανάλωση σε MWh ακολουθεί ένα μοτίβο που εξαρτάται από την ημέρα της εβδομάδας (διαφοροποίηση τα Σάββατα και τις Κυριακές από τις καθημερινές) και την ώρα της ημέρας, όπως αυτό ήταν αναμενόμενο. Θα θεωρήσουμε λοιπόν ότι οι επιμέρους καταναλώσεις σε MWh ακολουθούν το μοτίβο αυτό και θα βασίσουμε την πρόβλεψή μας πάνω σε αυτό (βλ. Εικόνα 7).

Επομένως, για να πραγματοποιήσουμε τις εκτιμήσεις μας για μία συγκεκριμένη ώρα και για μια συγκεκριμένη ημέρα, θα λάβουμε υπόψιν μας τα διαθέσιμα δεδομένα για τις επιμέρους καταναλώσεις σε MWh για κάποιο πρόσφατο χρονικό διάστημα και θα κρατήσουμε μόνο τις καταναλώσεις που αφορούν τη συγκεκριμένη ώρα και ημέρα. Από τις καταναλώσεις αυτές θα βρούμε τον μέσο όρο τους και η τιμή που θα προκύψει θα ανταποκρίνεται στην εκτιμώμενη τιμή των συνολικών MWh για την συγκεκριμένη ώρα και ημέρα. Θα θεωρήσουμε επίσης ότι η τιμή της kWh για μια συγκεκριμένη ώρα και ημέρα είναι αντιστρόφως ανάλογη με την συνολική πανελλήνια κατανάλωση σε MWh, βάσει της παραδοχής 3 (βλ. Ενότητα 1.5). Τελικά, θα πραγματοποιηθεί κανονικοποίηση της τιμής που υπολογίσαμε πολλαπλασιάζοντας με κάποιον συντελεστή και το τελικό αποτέλεσμα θα συνιστά την πρόβλεψή μας για την τιμή της kWh για τη συγκεκριμένη ώρα και ημέρα.

Τα πειραματικά δεδομένα υποδεικνύουν έναν συντελεστή περίπου ίσο με $2.1*10^{-5}$.



Εικόνα 7. Ενδεικτικό διάγραμμα κατανάλωσης ενέργειας ανά ώρα

2.3 Υπολογισμός Πιθανότητας Χρήσης Θερμοσίφωνα σε Μελλοντικό Χρόνο

Ο αλγόριθμος χρειάζεται επίσης να γνωρίζει ποιες στιγμές μέσα στις επόμενες 6 ώρες υπάρχει σημαντική πιθανότητα να γίνει χρήση του θερμοσίφωνα. Οι πιθανότητες αυτές μπορούν να υπολογιστούν από την ανάλυση των σχετικών συνηθειών των ενοίκων του σπιτιού για κάποιο χρονικό διάστημα. Για τον λόγο αυτό υπάρχει μια αρχική περίοδος «εκπαίδευσης» του συστήματος πριν μπορέσει ο αλγόριθμος να λαμβάνει ασφαλείς αποφάσεις και να είναι παραγωγικός. Ορίζουμε την περίοδο αυτή στον ένα μήνα ο οποίος κρίνεται επαρκής για να αναδειχτούν οι συνήθεις χρήσεις του ηλεκτρικού θερμοσίφωνα στο έξυπνο σπίτι. Για το χρονικό διάστημα αυτό κάθε φορά που πραγματοποιείται χρήση του θερμοσίφωνα, γίνεται καταγραφή του συμβάντος (ημερομηνία και ώρα) σε ένα αρχείο, έτσι ώστε να μπορούν τα δεδομένα αυτά να χρησιμοποιηθούν στο μέλλον για τους υπολογισμούς μας. Παράδειγμα ενός τέτοιου αρχείου φαίνεται στην Εικόνα 121 (βλ. ΠΑΡΑΡΤΗΜΑ Δ).

Για τον υπολογισμό των πιθανοτήτων θα χρησιμοποιηθούν δύο μέθοδοι, οι οποίες θα συμπεριληφθούν στην τελική υλοποίηση: μια στατιστική προσέγγιση και μία προσέγγιση που περιλαμβάνει την ενσωμάτωση συστήματος πρόβλεψης χρονοσειράς.

2.3.1 Στατιστική Προσέγγιση

Ένας απλός τρόπος υπολογισμού της πιθανότητας χρήσης του θερμοσίφωνα μία συγκεκριμένη ώρα και ημέρα είναι ακολουθώντας μια καθαρά στατιστική προσέγγιση. Αρχικά χωρίζουμε την ημέρα σε 24 «παράθυρα» μίας ώρας το καθένα, δημιουργώντας έναν πίνακα με 24 θέσεις και αρχικοποιούμε όλες τις θέσεις με 0. Γεμίζουμε τον πίνακα αυτό διατρέχοντας το αρχείο καταγραφών που περιγράφεται στην Ενότητα 2.3 και για κάθε καταγραφή που συναντάμε αυξάνουμε κατά 1 μονάδα την τιμή στη θέση του πίνακα που αντιστοιχεί στο παράθυρο που περιλαμβάνει την ώρα της συγκεκριμένης καταγραφής. Αν, για παράδειγμα, μια καταγραφή χρήσης του θερμοσίφωνα είναι στις 18:35:27 αυξάνουμε τη θέση 18 του πίνακα κατά 1 και ούτω καθεξής. Αφού ολοκληρώσουμε αυτή τη διαδικασία διαιρούμε όλες τις τιμές του πίνακα με το συνολικό πλήθος των καταγραφών και οι τιμές που προκύπτουν αποτελούν μια εκτίμηση της πιθανότητας χρήσης του θερμοσίφωνα για κάθε ένα από τα 24 «παράθυρα» μίας ώρας.

Επειδή όμως οι συνήθειες του σπιτιού όσον αφορά τη χρήση του θερμοσίφωνα αλλάζουν με την εποχή, για παράδειγμα λόγω της αλλαγής του καιρού, θα πρέπει να δώσουμε μεγαλύτερη βαρύτητα στις πιο πρόσφατες εγγραφές και μικρότερη στις παλαιότερες. Βελτιώνουμε λοιπόν αυτόν τον τρόπο υπολογισμού των πιθανοτήτων εισάγοντας ένα σύστημα στάθμισης το οποίο προσθέτει μεγαλύτερο βάρος στις πιο πρόσφατες καταγραφές και μικρότερο βάρος στις παλαιότερες. Η κατανομή των βαρών θα ακολουθεί φθίνουσα εκθετική πορεία της μορφής $f(t) = \mathrm{a} + b * e^{-c*t}$ όπου το t συμβολίζει το πλήθος των ημερών που έχουν περάσει από την ημερομηνία της καταγραφής μέχρι και σήμερα. Θεωρούμε πιο σημαντικές τις καταγραφές των τελευταίων 30 ημερών και απαιτούμε: συντελεστή 2 για τις σημερινές καταγραφές, συντελεστή 1 για αυτές που είναι 30 ημέρες πριν και συντελεστή 0.5 για τις παλαιότερες (όσο τείνουμε στο -∞). Με τις παραπάνω συνθήκες η συνάρτηση βαρών που προκύπτει είναι η $f(t) = 0.5 + 1.5 * e^{-\frac{ln3}{30}*t}$.

Επαναλαμβάνουμε τη διαδικασία που περιγράψαμε στην πρώτη παράγραφο, με την εξής διαφοροποίηση: για κάθε καταγραφή, αυξάνουμε την τιμή της αντίστοιχης θέσης του πίνακα κατά το βάρος f(t) που αντιστοιχεί στην καταγραφή αυτή, αντί για τον αριθμό 1. Αφού ολοκληρωθεί αυτή η διαδικασία, διαιρούμε όλες τις τιμές του πίνακα με το συνολικό άθροισμα των βαρών και οι τιμές που προκύπτουν αποτελούν την σταθμισμένη πιθανότητα χρήσης του θερμοσίφωνα για κάθε ένα από τα 24 «παράθυρα» μίας ώρας.

2.3.2 Πρόβλεψη Χρονοσειράς

Για μεγαλύτερη ακρίβεια στον υπολογισμό των πιθανοτήτων, πέρα από την αμιγώς στατιστική προσέγγιση, θα χρησιμοποιηθεί ένα σύστημα πρόβλεψης χρονοσειρών (time series forecasting). Η πρόβλεψη χρονοσειρών είναι μια τεχνική ανάλυσης δεδομένων που χρησιμοποιείται για να προβλέψει τις μελλοντικές τιμές ενός μεταβαλλόμενου μεγέθους με βάση τα παρελθοντικά δεδομένα. Οι μέθοδοι πρόβλεψης

χρονοσειρών είναι κατάλληλες για διεργασίες όπως αυτή, καθώς μπορούν να αναγνωρίσουν πολύπλοκα χρονικά μοτίβα και σχέσεις μεταξύ των δεδομένων. Στην προκειμένη περίπτωση, οι συνήθειες που έχουν οι άνθρωποι όσον αφορά την ώρα που χρησιμοποιούν τον θερμοσίφωνα καθημερινά για μπάνιο, για παράδειγμα, δεν είναι τυχαίες αλλά ακολουθούν ορισμένα μοτίβα. Θα χρησιμοποιήσουμε την πρόβλεψη χρονοσειράς για να υπολογίσουμε την πιθανότητα χρήσης του θερμοσίφωνα στα ζητούμενα «παράθυρα» μίας ώρας.

Τα μοντέλα πρόβλεψης χρονοσειρών μπορούν να παρέχουν πιο ακριβείς προβλέψεις από απλούς υπολογισμούς πιθανοτήτων, ειδικά όταν τα μοτίβα που παρουσιάζουν δεν είναι εύκολα προβλέψιμα χρησιμοποιώντας απλά στατιστικά στοιχεία. Ωστόσο, τα μοντέλα αυτά απαιτούν περισσότερα δεδομένα και ρύθμιση για τη δημιουργία και τη συντήρησή τους.

Το μοντέλο πρόβλεψης χρονοσειρών αν και είναι πιο ακριβές στους υπολογισμούς του από την απλή στατιστική προσέγγιση, απαιτεί μεγαλύτερο όγκο δεδομένων για να εκπαιδευτεί και να αρχίσει να παράγει αξιόπιστα αποτελέσματα. Για τον λόγο αυτό θα χρησιμοποιήσουμε το απλό στατιστικό μοντέλο για τον πρώτο έναν μήνα, όπου παράλληλα θα συγκεντρώνεται και θα αποθηκεύεται όγκος δεδομένων. Στη συνέχεια, αφού το μοντέλο πρόβλεψης χρονοσειρών έχει αρκετά δεδομένα στην διάθεσή του μπορεί να τεθεί σε λειτουργία και να αντικαταστήσει την απλή στατιστική προσέγγιση.

2.4 Θερμοκρασία στο Εσωτερικό του Θερμοσίφωνα

Τέλος, απαραίτητο για την απόφαση του αλγορίθμου είναι να μπορεί να μετράει τη θερμοκρασία του νερού στο εσωτερικό του θερμοσίφωνα. Με τον τρόπο αυτό μπορεί να εξάγει συμπεράσματα για την κατάσταση στην οποία βρίσκεται ο θερμοσίφωνας τη δεδομένη στιγμή και να προσαρμόζει την απόφασή του. Αν για παράδειγμα, προβλέπει σημαντική πιθανότητα για χρήση του θερμοσίφωνα μέσα στην επόμενη ώρα, αλλά το νερό στο εσωτερικό του καζανιού είναι ήδη ζεστό, δε χρειάζεται να κάνει κάποια ενέργεια. Για τον λόγο αυτό, θα χρειαστεί να εγκατασταθεί ένα όργανο μέτρησης θερμοκρασίας, το οποίο θα παρέχει τα απαιτούμενα δεδομένα.

2.4.1 Μετρητής Θερμοκρασίας και Υγρασίας (ANDWOL AS90W)

Οι μετρητές θερμοκρασίας και υγρασίας είναι ηλεκτρονικά μέσα που χρησιμοποιούνται για τη μέτρηση της θερμοκρασίας και της υγρασίας (σχετικής υγρασίας) του περιβάλλοντος. Είναι συχνά χρήσιμοι σε πολλές εφαρμογές, όπως σε εργαστήρια, βιομηχανίες, αγροτικές εργασίες, κτίρια, αποθήκες και περιβαλλοντικές μετρήσεις.

Συνήθως, οι μετρητές θερμοκρασίας και υγρασίας είναι συνδεδεμένοι με αισθητήρες οι οποίοι μετρούν τις εν λόγω τιμές και εμφανίζουν τα αποτελέσματα σε μία οθόνη. Σε ορισμένες περιπτώσεις οι μετρητές αυτοί μπορεί να είναι εξοπλισμένοι με δυνατότητες αποθήκευσης δεδομένων ή σύνδεσης με τον υπολογιστή ή το δίκτυο για την παρακολούθηση και την ανάλυση των μετρήσεων.

Υπάρχουν επίσης μετρητές θερμοκρασίας και υγρασίας που μπορούν να συνδεθούν μέσω ασύρματης δικτύωσης Wi-Fi με έξυπνα σπίτια και συγκεκριμένα με την πλατφόρμα έξυπνου σπιτιού Tuya.

Για τους σκοπούς της εργασίας αυτής θα χρησιμοποιηθεί ο μετρητής ANDWOL AS90W, ο οποίος είναι συμβατός με την πλατφόρμα Tuya (βλ. Εικόνες 8, 9). Ο μετρητής αυτός διαθέτει αισθητήρα κατάλληλο για προσαρμογή μέσα στο καζάνι του θερμοσίφωνα, έτσι ώστε να μπορεί να μετράει ανά πάσα στιγμή τη θερμοκρασία του νερού στο εσωτερικό του.

Εικόνα 8. Μετρητής θερμοκρασίας και υγρασίας ANDWOL AS90W

Εικόνα 9. Χαρακτηριστικά μετρητή θερμοκρασίας και υγρασίας ANDWOL AS90W

2.4.2 Εγκατάσταση Μετρητή

Ο μετρητής θερμοκρασίας και υγρασίας ANDWOL AS90W διαθέτει αισθητήριο όργανο για τη μέτρηση της θερμοκρασίας το οποίο προσαρμόζεται στην ειδική υποδοχή που διαθέτουν οι ηλεκτρικοί θερμοσίφωνες με εύκολο τρόπο (βλ. Εικόνα 10). Με αυτή την απλή εγκατάσταση αποκτάμε άμεση πρόσβαση στη μέτρηση της θερμοκρασίας του νερού στο εσωτερικό του καζανιού του θερμοσίφωνα.

Εικόνα 10. Τοποθέτηση αισθητήρα θερμοκρασίας και υγρασίας στον θερμοσίφωνα

2.4.3 Κατάσταση Θερμοσίφωνα

Ένας θερμοσίφωνας μπορεί να βρίσκεται σε μία από τις εξής 4 καταστάσεις:

- Κατάσταση ηρεμίας (idle): η θερμοκρασία παραμένει σχετικά σταθερή με ελάχιστες διακυμάνσεις
- Κατάσταση θέρμανσης (heating): η θερμοκρασία αυξάνεται γρήγορα
- Κατάσταση ψύξης (cooling): η θερμοκρασία μειώνεται αργά (λόγω της εξισορρόπησης με το περιβάλλον)
- Κατάσταση χρήσης (use): η θερμοκρασία μειώνεται γρηγορότερα (το ζεστό νερό που χρησιμοποιείται αντικαθίσταται με κρύο και η μέση θερμοκρασία στο εσωτερικό μειώνεται)

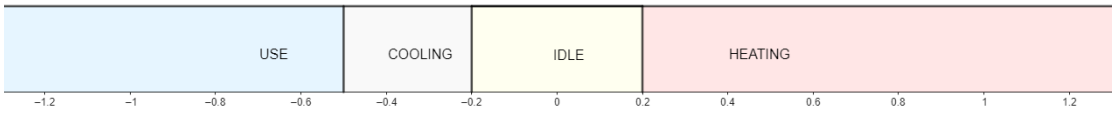
Μετρώντας τη θερμοκρασία του νερού στο εσωτερικό του θερμοσίφωνα ανά τακτά χρονικά διαστήματα μπορούμε να γνωρίζουμε σε ποια από τις παραπάνω καταστάσεις βρίσκεται ο θερμοσίφωνας.

Μετράμε τη θερμοκρασία σε βαθμούς Κελσίου (ºC) κάθε 5 λεπτά και ορίζουμε τις συνθήκες μεταξύ της προηγούμενης θερμοκρασίας που μετρήσαμε και της τωρινής οι οποίες καθορίζουν την κατάσταση στην οποία βρίσκεται ο θερμοσίφωνας.

Έστω T_{curr} η θερμοκρασία που μετρήθηκε την τωρινή χρονική στιγμή και T_{last} η θερμοκρασία που μετρήθηκε 5 λεπτά πριν.

- Αν $|T_{curr} - T_{last}| \leq 0.2$ τότε ο θερμοσίφωνας βρίσκεται στην κατάσταση ηρεμίας (idle)
- Αν $T_{curr} - T_{last} > 0.2$ τότε ο θερμοσίφωνας βρίσκεται στην κατάσταση θέρμανσης (heating)

- Αν $-0.5 < T_{curr} - T_{last} < -0.2$ τότε ο θερμοσίφωνας βρίσκεται στην κατάσταση ψύξης (cooling)
- Αν $T_{curr} - T_{last} \leq -0.5$ τότε ο θερμοσίφωνας βρίσκεται στην κατάσταση χρήσης (use)



Εικόνα 11. Καταστάσεις θερμοσίφωνα

Οι παραπάνω συνθήκες προέκυψαν μετά από παρατήρηση της θερμοκρασίας του νερού στο εσωτερικό του θερμοσίφωνα σε κάθε μία από τις παραπάνω καταστάσεις (Βλ. Εικόνα 11).

2.5 Αλγόριθμος Απόφασης

Στις προηγούμενες ενότητες του παρόντος κεφαλαίου περιγράφηκαν οι τρόποι με τους οποίους αποκτήθηκαν τα απαραίτητα δεδομένα, είτε άμεσο είτε έμμεσα, συνθέτοντας άλλα πρωτογενή δεδομένα. Έχοντας στη διάθεσή μας τις πληροφορίες αυτές, διατυπώνουμε τον ζητούμενο αλγόριθμο ο οποίος τελικά αποφαίνεται αν η στιγμή που καλείται είναι η βέλτιστη στιγμή για να τεθεί σε λειτουργία ο ηλεκτρικός θερμοσίφωνας ή όχι.

Ο αλγόριθμος απόφασης θα πρέπει να εκτελείται ανά τακτά χρονικά διαστήματα έτσι ώστε να αναζητά μέσα στη διάρκεια της ημέρας τις κατάλληλες στιγμές για να ενεργοποιήσει τον ηλεκτρικό θερμοσίφωνα. Αφού η τιμή του ηλεκτρικού ρεύματος, σύμφωνα με το δυναμικό μοντέλο τιμολόγησης αλλάζει από τον πάροχο κάθε μία ώρα, επιλέγουμε το διάστημα της μίας ώρας ως το διάστημα μεταξύ δύο διαδοχικών εκτελέσεων του αλγορίθμου απόφασης.

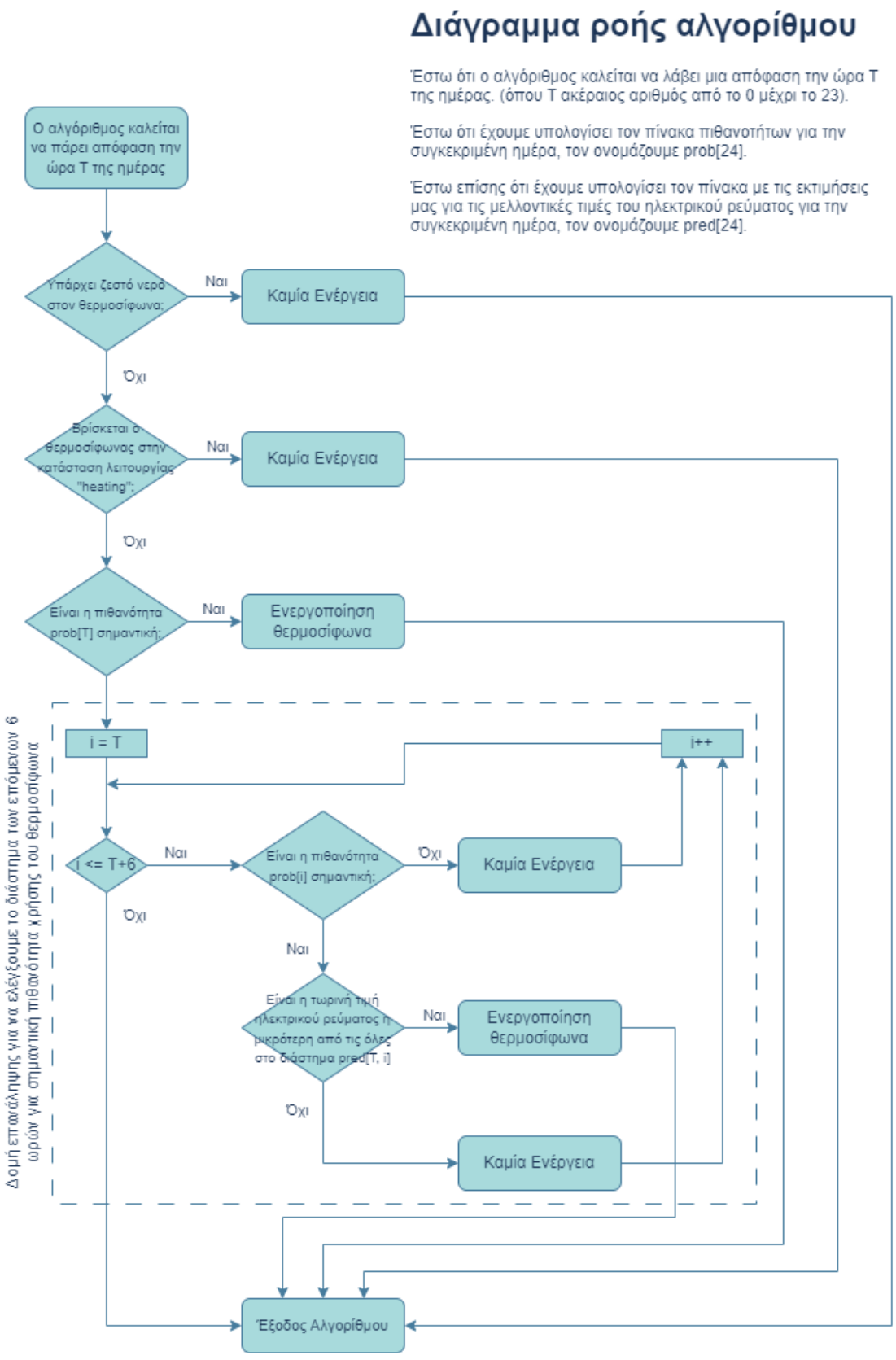
Αρχικά ο αλγόριθμος ελέγχει δύο συνθήκες. Αν υπάρχει ήδη ζεστό νερό στον θερμοσίφωνα τότε δε χρειάζεται να πραγματοποιηθεί κάποια ενέργεια αφού είναι έτοιμος σε περίπτωση ενδεχόμενης χρήσης. Επίσης, αν ο θερμοσίφωνας βρίσκεται στην κατάσταση λειτουργίας ή “heating” από εξωτερικό παράγοντα, τότε επίσης δε χρειάζεται να πραγματοποιηθεί κάποια ενέργεια για τον ίδιο λόγο.

Στη συνέχεια ελέγχεται αν η πιθανότητα χρήσης του θερμοσίφωνα για το παράθυρο μίας ώρας που βρίσκεται τη δεδομένη στιγμή είναι σημαντική (βλ. Ενότητα 2.3). Αν είναι σημαντική, τότε ενεργοποιείται αμέσως ο θερμοσίφωνας, αφού αναμένεται άμεση χρήση (μέσα στην επόμενη μία ώρα), δεν υπάρχει περιθώριο αναζήτησης της βέλτιστης στιγμής και ο αλγόριθμος τερματίζει. Αν η πιθανότητα δεν είναι σημαντική συνεχίζεται η εκτέλεση του αλγορίθμου.

Κατόπιν γίνεται αναζήτηση μέσα στο διάστημα των επόμενων 6 ωρών, βάσει την παραδοχής 2 (βλ. Ενότητα 1.5), αν υπάρχει παράθυρο μίας ώρας με σημαντική

πιθανότητα χρήσης (βλ. Ενότητα 2.3). Αν δεν υπάρχει τότε ο αλγόριθμος τερματίζει χωρίς να λάβει χώρα κάποια ενέργεια, καθώς οι επόμενες ώρες είναι «κενές», δηλαδή χωρίς καμία σημαντική πιθανότητα για χρήση. Αν υπάρχει, τότε ξεκινάει από αυτό το πρώτο «σημαντικό» παράθυρο που εντόπισε. Για το παράθυρο αυτό, συγκρίνει την τωρινή τιμή του ηλεκτρικού ρεύματος με τις τιμές που έχουν προβλεφθεί (βλ. Ενότητα 2.2) για τα ενδιάμεσα παράθυρα μίας ώρας (από τώρα μέχρι και το «σημαντικό» παράθυρο που εξετάζει). Αν η τωρινή τιμή είναι η μικρότερη από όλες αυτές τις ενδιάμεσες τιμές, τότε επιλέγει να ενεργοποιήσει τον ηλεκτρικό θερμοσίφωνα τώρα αφού μετά αναμένεται η τιμή του ηλεκτρικού ρεύματος να αυξηθεί και ο αλγόριθμος τερματίζει. Αν η τωρινή τιμή δεν είναι η μικρότερη, σημαίνει ότι υπάρχει στιγμή παρακάτω που αναμένεται η τιμή να είναι πιο συμφέρουσα, οπότε δεν πραγματοποιείται κάποια ενέργεια. Ο αλγόριθμος συνεχίζει εξετάζοντας όλα τα «σημαντικά» παράθυρα που θα εντοπίσει μέσα στις επόμενες 6 ώρες.

Το διάγραμμα ροής του αλγορίθμου φαίνεται παρακάτω (βλ. Εικόνα 12).



Εικόνα 12. Διάγραμμα ροής αλγορίθμου απόφασης

Β. ΠΡΑΚΤΙΚΟ ΜΕΡΟΣ

ΚΕΦΑΛΑΙΟ 3 – Εργαλεία Ανάπτυξης

Στο κεφάλαιο αυτό παρουσιάζονται τα εργαλεία προγραμματισμού, οι γλώσσες και τα περιβάλλοντα λογισμικού που χρησιμοποιήθηκαν για την ανάπτυξη του αλγορίθμου ελέγχου της κατανάλωσης του ηλεκτρικού θερμοσίφωνα σε ένα έξυπνο σπίτι, τη δημιουργία σχετικής εφαρμογής για κινητές συσκευές Android και του εργαλείου επίβλεψης της λειτουργίας της υπηρεσίας από τον υπολογιστή (web server).

3.1 Tuya IoT Development Platform

Η Tuya IoT Development Platform είναι μια πλατφόρμα που αναπτύχθηκε από την Tuya Technology, μια εταιρεία με έδρα την Κίνα που ειδικεύεται στον τομέα του Internet of Things (IoT). Η πλατφόρμα αυτή προσφέρει εργαλεία και υπηρεσίες για τη δημιουργία και τη διαχείριση συσκευών που συνδέονται στο διαδίκτυο. Παρέχει λογισμικό ανάπτυξης, όπως SDKs (Software Development Kits) και APIs (Application Programming Interfaces) που επιτρέπουν στους προγραμματιστές να δημιουργούν εφαρμογές και λογισμικό για τις συσκευές IoT. Επίσης, παρέχει υποδομές για την ασφαλή σύνδεση, τη διαχείριση και τον έλεγχο των συσκευών αυτών μέσω του διαδικτύου [18]. Τα χαρακτηριστικά της πλατφόρμας Tuya αναφέρονται αναλυτικά στην ενότητα 1.2.4.

3.2 Python

Η Python είναι μία δημοφιλής, υψηλού επιπέδου γλώσσα προγραμματισμού γενικής χρήσης. Δημιουργήθηκε από τον Guido van Rossum και κυκλοφόρησε για πρώτη φορά το 1991. Ξεχωρίζει για τη συντακτική της απλότητα και την ευκολία χρήσης και χρησιμοποιείται τόσο από αρχάριους όσο και από επαγγελματίες προγραμματιστές. Άλλα σημαντικά χαρακτηριστικά της Python είναι ότι εκτελείται σε πολλά λειτουργικά συστήματα και πλατφόρμες, καθιστώντας τον κώδικα φορητό και επίσης ότι υπάρχει μια μεγάλη και ενεργή κοινότητα προγραμματιστών Python που παρέχει υποστήριξη, βιβλιοθήκες και πληροφορίες. Επιπλέον, η Python είναι ελεύθερη και ανοιχτού κώδικα, προσφέροντας ελευθερία στους χρήστες να χρησιμοποιήσουν, να τροποποιήσουν και να διανείμουν τον κώδικα. Χρησιμοποιείται ευρέως σε διάφορους τομείς, όπως η ανάλυση δεδομένων, η τεχνητή νοημοσύνη, οι αυτοματισμοί και αλλού [28].

3.2.1 Tuya IoT Python SDK

Το Tuya IoT Python SDK είναι ένα προγραμματιστικό εργαλείο που παρέχεται από την Tuya Technology και επιτρέπει στους προγραμματιστές να αναπτύσσουν εφαρμογές και λογισμικό που συνδέονται με συσκευές IoT οι οποίες χρησιμοποιούν την πλατφόρμα Tuya. Αυτό το SDK είναι γραμμένο στη γλώσσα προγραμματισμού Python και παρέχει πρόσβαση σε πολλές λειτουργίες και υπηρεσίες που προσφέρονται από την Tuya IoT Development Platform. Το Tuya IoT Python SDK διευκολύνει την ανάπτυξη εφαρμογών IoT με χρήση της γλώσσας προγραμματισμού Python [29].

Η εγκατάσταση της παραπάνω βιβλιοθήκης γίνεται από την γραμμή εντολών με την χρήση του pip, εκτελώντας την εντολή:

```
pip3 install tuya-iot-py-sdk
```

3.2.2 Flask

Το Flask είναι ένα ελαφρύ, ανοικτού κώδικα πλαίσιο (framework) για την ανάπτυξη web εφαρμογών σε γλώσσα Python. Το Flask είναι σχεδιασμένο για να είναι απλό και ευέλικτο, επιτρέποντας στους προγραμματιστές να δημιουργούν γρήγορα εφαρμογές web και απλές API. Είναι κατάλληλο για μικρές έως μεσαίες εφαρμογές και είναι εύκολο στη χρήση. Μπορεί να επεκταθεί προσθέτοντας επιπλέον βιβλιοθήκες και εργαλεία ανάλογα με τις ανάγκες της εφαρμογής. Τέλος, το Flask διαθέτει έναν ενσωματωμένο διακομιστή ανάπτυξης, ο οποίος επιτρέπει την εύκολη ανάπτυξη και δοκιμή των εφαρμογών μας.

Το Flask είναι βασισμένο στο Werkzeug WSGI toolkit, το Jinja template engine, και το Click CLI toolkit και παρέχει στον προγραμματιστή διάφορες δυνατότητες όπως url routing, template engine, RESTful request dispatching [30].

Η εγκατάσταση του Flask Framework γίνεται από την γραμμή εντολών με την χρήση του pip, εκτελώντας την εντολή:

```
pip install flask
```

3.2.3 Flask-SocketIO

Το Flask-SocketIO είναι μία επέκταση για το Flask η οποία δίνει στις εφαρμογές Flask τη δυνατότητα αμφίδρομης επικοινωνίας μεταξύ του εξυπηρετητή (server) και του πελάτη (client) με πολύ χαμηλό χρόνο απόκρισης. Η εφαρμογή από πλευράς του εξυπηρετητή (server-side) μπορεί να χρησιμοποιήσει την βιβλιοθήκη SocketIO σε Python για να δημιουργήσει έναν δίαυλο επικοινωνίας με την πλευρά του πελάτη (client-side). Από πλευράς του πελάτη χρησιμοποιείται η βιβλιοθήκη SocketIO σε JavaScript για την επικοινωνία με τον εξυπηρετητή [31].

Η εγκατάσταση του Flask-SocketIO γίνεται από την γραμμή εντολών με τη χρήση του pip, εκτελώντας την εντολή:

```
pip install flask-socketio
```

3.3 HTML & CSS

Η HTML (HyperText Markup Language, Γλώσσα Σήμανσης Υπερκειμένου) είναι η κύρια γλώσσα σήμανσης για τη δημιουργία μιας ιστοσελίδας. Τα βασικά στοιχεία της HTML είναι οι ετικέτες (tags). Με τις ετικέτες δηλώνουμε τα διάφορα δομικά στοιχεία μιας ιστοσελίδας όπως κείμενο, εικόνες, πίνακες, παραγράφους, επικεφαλίδες, τίτλο

της σελίδας κλπ. Οι περιηγητές (browsers) διαβάζουν τα αρχεία HTML, μεταφράζουν τις ετικέτες και παρουσιάζουν κατάλληλα το περιεχόμενο της ιστοσελίδας. Η HTML επιτρέπει επίσης την ενσωμάτωση σεναρίων εντολών σε γλώσσες όπως η JavaScript, τα οποία επηρεάζουν τη συμπεριφορά των ιστοσελίδων HTML και από στατικές τις κάνουν διαδραστικές [32].

Η CSS (Cascading Style Sheets, Διαδοχικά Φύλλα Ύφους ή Επάλληλα Φύλλα Ύφους) είναι μια γλώσσα υπολογιστή η οποία ανήκει στην κατηγορία των γλωσσών φύλλων ύφους και χρησιμοποιείται για τον έλεγχο της εμφάνισης ενός εγγράφου που έχει γραφτεί με μια γλώσσα σήμανσης, όπως οι HTML και XHTML. Χρησιμοποιείται δηλαδή για να καθορίσει τον τρόπο με τον οποίο θα εμφανίζονται στον επισκέπτη τα διάφορα στοιχεία της HTML σε μια ιστοσελίδα και γενικότερα σε έναν ιστότοπο. Προστέθηκε στην HTML 4.0 για να επιλύσει το πρόβλημα μορφοποίησης των σελίδων, μειώνοντας σημαντικά τον όγκο της εργασίας των σχεδιαστών [32].

3.4 JavaScript

Η JavaScript (JS) είναι μια γλώσσα προγραμματισμού η οποία αποτελεί μία από τις κύριες τεχνολογίες που χρησιμοποιούνται στον Παγκόσμιο Ιστό μαζί με την HTML και τη CSS και δίνει τη δυνατότητα της επικοινωνίας μεταξύ των client scripts και του χρήστη. Η σύνταξή της είναι επηρεασμένη από τη C, αλλά γενικά οι δύο αυτές γλώσσες δε σχετίζονται και έχουν πολύ διαφορετική σημασιολογία. Είναι δυναμική, διαθέτει συναρτήσεις και συνδυάζει αντικειμενοστραφές, προστακτικό και συναρτησιακό στυλ προγραμματισμού [33].

3.5 Java

Η Java είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού υψηλού επιπέδου που δημιουργήθηκε από την εταιρεία Sun Microsystems (τώρα μέρος της Oracle Corporation) τη δεκαετία του '90. Πρόκειται για μια από τις πιο δημοφιλείς και ευρέως χρησιμοποιούμενες γλώσσες προγραμματισμού στον κόσμο. Η Java είναι γνωστή για την ικανότητά της να λειτουργεί σε διάφορες πλατφόρμες χωρίς αλλαγές στον πηγαίο κώδικα, αφού είναι σχεδιασμένη να έχει όσο το δυνατόν λιγότερες εξαρτήσεις (dependencies). Οι εφαρμογές Java μεταγλωττίζονται και σε ενδιάμεσο κώδικα (bytecode) και μπορούν να λειτουργήσουν σε οποιοδήποτε σύστημα διαθέτει εικονικό περιβάλλον Java (JVM), ανεξάρτητα από την αρχιτεκτονική. Αυτό το χαρακτηριστικό της Java την καθιστά κατάλληλη για επιστημονικές εφαρμογές που απαιτούν φορητότητα μεταξύ διαφορετικών συστημάτων.

Το συντακτικό της Java είναι παρόμοιο με αυτό της C και της C++, απλά με λιγότερες λειτουργίες χαμηλού επιπέδου και από τις δύο. Το περιβάλλον της Java παρέχει δυναμικές προοπτικές, όπως η ανάκλαση (reflection) και η τροποποίηση του κώδικα κατά την εκτέλεση (runtime code modification), δυνατότητες οι οποίες δεν είναι διαθέσιμες στις περισσότερες γλώσσες προγραμματισμού. Τέλος, η Java χρησιμοποιείται σε ένα μεγάλο εύρος εφαρμογών όπως εφαρμογές για κινητά και

σταθερούς υπολογιστές, διαδικτυακές εφαρμογές, διαδικτυακούς εξυπηρετητές (web servers), παιχνίδια, βάσεις δεδομένων και πολλά άλλα [34].

3.5.1 Android Studio και Ανάπτυξη Εφαρμογής Android με Java

Το Android Studio είναι το επίσημο ολοκληρωμένο περιβάλλον ανάπτυξης (Integrated Development Environment - IDE) για το λειτουργικό σύστημα Android της Google. Είναι χτισμένο πάνω στο λογισμικό IntelliJ IDEA της JetBrains και έχει σχεδιαστεί ειδικά για ανάπτυξη εφαρμογών Android. Το Android Studio παρέχει στα προγράμματα δημιουργίας εφαρμογών ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) βελτιστοποιημένο για εφαρμογές Android. Είναι διαθέσιμο για λήψη σε λειτουργικά συστήματα Windows, macOS και Linux. Το Android Studio παρέχει ένα πλήρες IDE, συμπεριλαμβανομένου και ενός προηγμένου επεξεργαστή κώδικα, προτύπων εφαρμογών και εργαλείων για ανάπτυξη, εντοπισμό σφαλμάτων και δοκιμές που καθιστούν ταχύτερη και ευκολότερη την ανάπτυξη εφαρμογών. Περιέχει επίσης εργαλεία για τη δημιουργία και την εκτέλεση της εφαρμογής, την ανάπτυξή της σε μια εικονική ή φυσική συσκευή και την υπογραφή και τη δημοσίευσή της στο Google Play Store. Συνολικά, το Android Studio είναι ένα ισχυρό εργαλείο για τη δημιουργία μιας εφαρμογής Android.

Για την ανάπτυξη εφαρμογών Android οι πιο συχνά χρησιμοποιούμενες γλώσσες προγραμματισμού είναι η Java και η Kotlin. Η Java ήταν η επίσημη γλώσσα για την ανάπτυξη εφαρμογών Android αλλά έχει αντικατασταθεί από την Kotlin από το 2019. Άλλες γλώσσες που μπορούν να χρησιμοποιηθούν για την ανάπτυξη εφαρμογών Android είναι η C#, η Python και η Corona. Ωστόσο, η Java και η Kotlin παραμένουν οι πιο δημοφιλείς και ευρέως χρησιμοποιούμενες γλώσσες για την ανάπτυξη εφαρμογών Android στο Android Studio [35].

3.5.2 Tuya Android Smart Life App SDK Sample

Η Tuya παρέχει στους προγραμματιστές το Tuya Smart Life App SDK διευκολύνοντας τους να ενσωματώσουν λειτουργίες του Tuya Open API στην εφαρμογή τους ή και να δημιουργήσουν εφαρμογές IoT από την αρχή. Αυτό το kit ανάπτυξης εφαρμογών περιλαμβάνει όλα τα απαραίτητα εργαλεία για τη διαχείριση των έξυπνων συσκευών Tuya σε ένα έξυπνο σπίτι και τη δημιουργία λύσεων IoT.

Στο αποθετήριο (repository) της Tuya στο github υπάρχουν 4 εκδοχές αυτού του SDK, υλοποιημένες σε 4 προγραμματιστικές γλώσσες: Objective-C και Swift για ανάπτυξη εφαρμογών σε λειτουργικά συστήματα iOS και επίσης Java και Kotlin για ανάπτυξη εφαρμογών σε λειτουργικά συστήματα Android. Τα SDK αυτά περιλαμβάνουν ένα δείγμα εφαρμογής με σκοπό την παρουσίαση των λειτουργιών και των δυνατοτήτων του Tuya Smart Life App SDK, από τα οποία στη συνέχεια μπορεί ο προγραμματιστής να αντιγράψει και να επεξεργαστεί αποσπάσματα κώδικα για την ανάπτυξη δικής του εφαρμογής [36].

Η εγκατάσταση των παραπάνω SDK γίνεται από την γραμμή εντολών με την χρήση του git, εκτελώντας την εντολή:

```
git clone --recurse-submodules git@github.com:tuya/tuya-smart-life-app-sdk.git
```

Στην παρούσα έρευνα για την υλοποίηση της εφαρμογής ελέγχου της κατανάλωσης του ηλεκτρικού θερμοσίφωνα σε ένα έξυπνο σπίτι χρησιμοποιήθηκε το Tuya Android Smart Life App SDK Sample και συγκεκριμένα ένα δείγμα υλοποιημένο σε Java με το όνομα tuya-home-android-sdk-sample-java. Αυτό το δείγμα παρουσιάζει τη χρήση του SDK αυτού για τη δημιουργία μιας εφαρμογής IoT από την αρχή. Χωρίζεται σε πολλές ομάδες λειτουργιών για να δώσει στους προγραμματιστές μια σαφή εικόνα της υλοποίησης για διαφορετικές περιπτώσεις, περιλαμβάνει τη διαδικασία εγγραφής χρήστη, τη διαχείριση οικίας για διαφορετικούς χρήστες, τη διαμόρφωση δικτύου συσκευών και διάφορα βασικά στοιχεία ελέγχου. Για τη διαμόρφωση του δικτύου συσκευών, υλοποιείται η λειτουργία EZ και η λειτουργία AP, που επιτρέπουν στους προγραμματιστές να ζευγαρώνουν συσκευές μέσω Wi-Fi, καθώς και να τις ελέγχουν μέσω LAN και MQTT. Για τον έλεγχο της συσκευής, το SDK παρέχει έναν απλό πίνακα ελέγχου για την αποστολή και τη λήψη κάθε είδους δεδομένων [37].

3.5.3 WekaForecaster

Η Weka είναι μια δημοφιλής βιβλιοθήκη για την ανάπτυξη αλγορίθμων μηχανικής μάθησης και ανάλυσης δεδομένων, γραμμένη σε Java. Παρέχει πολλούς αλγόριθμους μηχανικής μάθησης, εργαλεία για την προεπεξεργασία δεδομένων, εξαγωγή χαρακτηριστικών, εκπαίδευση μοντέλων και αξιολόγηση των αποτελεσμάτων. Είναι ένα πολύ χρήσιμο εργαλείο για την ανάπτυξη και τον πειραματισμό με αλγορίθμους μηχανικής μάθησης.

Το WekaForecaster είναι ένα εργαλείο που βασίζεται στη βιβλιοθήκη Weka και είναι σχεδιασμένο ειδικά για την πρόβλεψη χρονοσειρών. Αυτό το εργαλείο προσφέρει τη δυνατότητα ανάλυσης και πρόβλεψης των χρονοσειρών χρησιμοποιώντας διάφορους αλγόριθμους, όπως τα αριθμητικά μοντέλα, τα νευρωνικά δίκτυα και τα μοντέλα ARIMA (Αυτο-Αναφορικά Κινούμενα Μέσα).

3.6 XML

Η XML (Extensible Markup Language) είναι μια γλώσσα σήμανσης που περιέχει ένα σύνολο κανόνων για την ηλεκτρονική κωδικοποίηση κειμένων. Σχεδιάστηκε δίνοντας έμφαση στην απλότητα, τη γενικότητα και τη χρησιμότητα στο Διαδίκτυο. Είναι μία μορφοποίηση δεδομένων κειμένου, με ισχυρή υποστήριξη Unicode για όλες τις γλώσσες του κόσμου. Αν και η σχεδίασή της εστιάζει στα κείμενα, χρησιμοποιείται ευρέως για την αναπαράσταση αυθαίρετων δομών δεδομένων που προκύπτουν για παράδειγμα στις υπηρεσίες ιστού. Επίσης, χρησιμοποιείται για τη μεταφορά δεδομένων ανάμεσα σε εφαρμογές και για την αποθήκευση δεδομένων.

Βασικό χαρακτηριστικό της XML είναι ότι είναι επεκτάσιμη (extensible), οι χρήστες μπορούν δηλαδή να ορίσουν τις δικές τους ετικέτες (tags) και δομές δεδομένων προσαρμόζοντας τη γλώσσα στις ανάγκες τους. Διαθέτει επίσης ιεραρχική δομή (hierarchical structure), τα δεδομένα αποθηκεύονται σε μορφή δομημένου δέντρου με γονείς και παιδιά χρησιμοποιώντας ετικέτες. Επίσης, η γλώσσα XML είναι διαλειτουργική (interoperable), μπορεί δηλαδή να χρησιμοποιηθεί για την ανταλλαγή δεδομένων ανάμεσα σε εφαρμογές και πλατφόρμες που λειτουργούν σε διαφορετικά περιβάλλοντα.

Η XML έχει ευρύτατες εφαρμογές σε πολλούς τομείς, συμπεριλαμβανομένων των ιστοσελίδων (για ανακατασκευή δεδομένων), των βάσεων δεδομένων, των επιστημονικών εφαρμογών και γενικά χρησιμοποιείαται όπου απαιτείται η αποθήκευση και ο ενσωματωμένος χειρισμός δεδομένων σε μορφή κειμένου [38].

ΚΕΦΑΛΑΙΟ 4 – Υλοποίηση και Ερμηνεία του Κώδικα

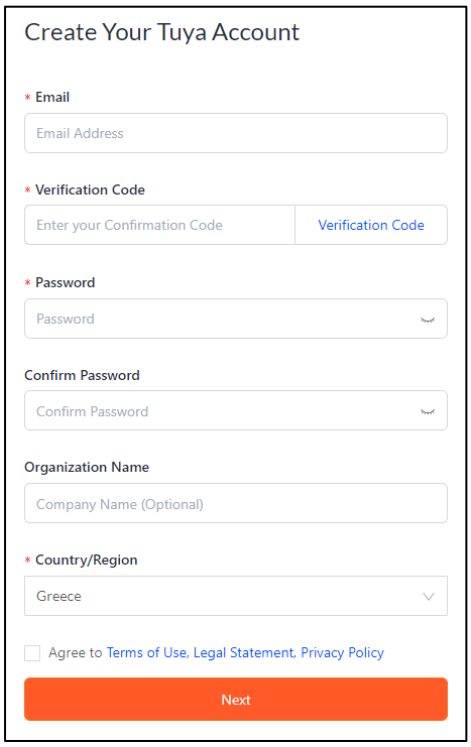
Στο κεφάλαιο αυτό περιγράφονται αναλυτικά τα βήματα υλοποίησης του αλγορίθμου ελέγχου της κατανάλωσης του ηλεκτρικού θερμοσίφωνα σε ένα έξυπνο σπίτι, της δημιουργίας της εφαρμογής IoT για κινητές συσκευές Android και του εργαλείου επίβλεψης της λειτουργίας της υπηρεσίας από τον υπολογιστή (web server).

4.1 Προετοιμασία Πλατφόρμας Tuya

Στην ενότητα αυτή παρουσιάζονται τα στάδια προετοιμασίας του Project στο Tuya Cloud.

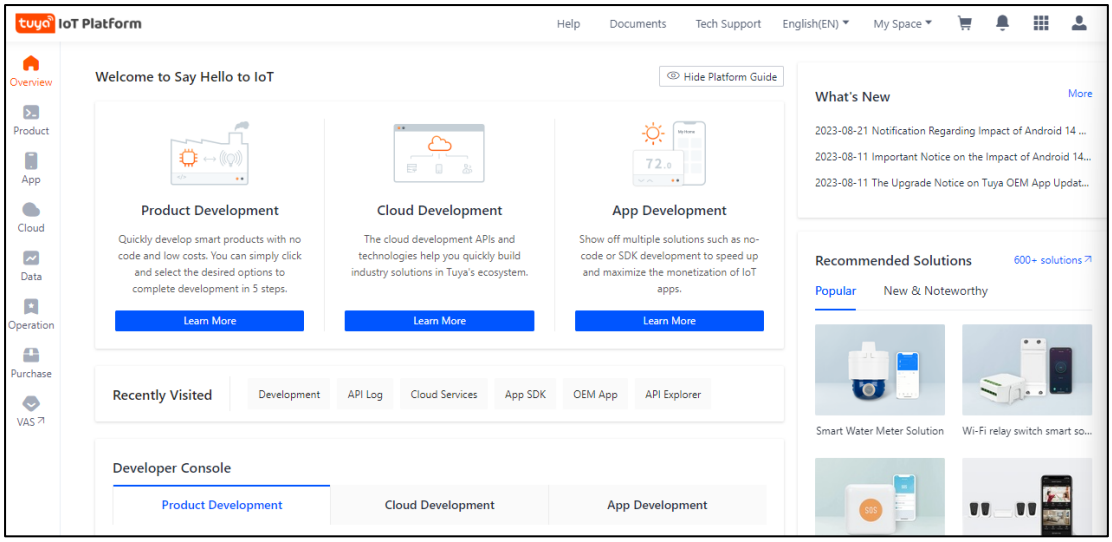
4.1.1 Δημιουργία Λογαριασμού Tuya IoT Development Platform

Αρχικά, απαιτείται η δημιουργία λογαριασμού Tuya για προγραμματιστές (Tuya IoT Developer Account) στην ιστοσελίδα της Tuya. Ακολουθούμε τον σύνδεσμο https://developer.tuya.com/en/, πατάμε το πλήκτρο της εγγραφής και συμπληρώνουμε τα απαιτούμενα στοιχεία (βλ. Εικόνα 13).



Εικόνα 13. Δημιουργία λογαριασμού Tuya

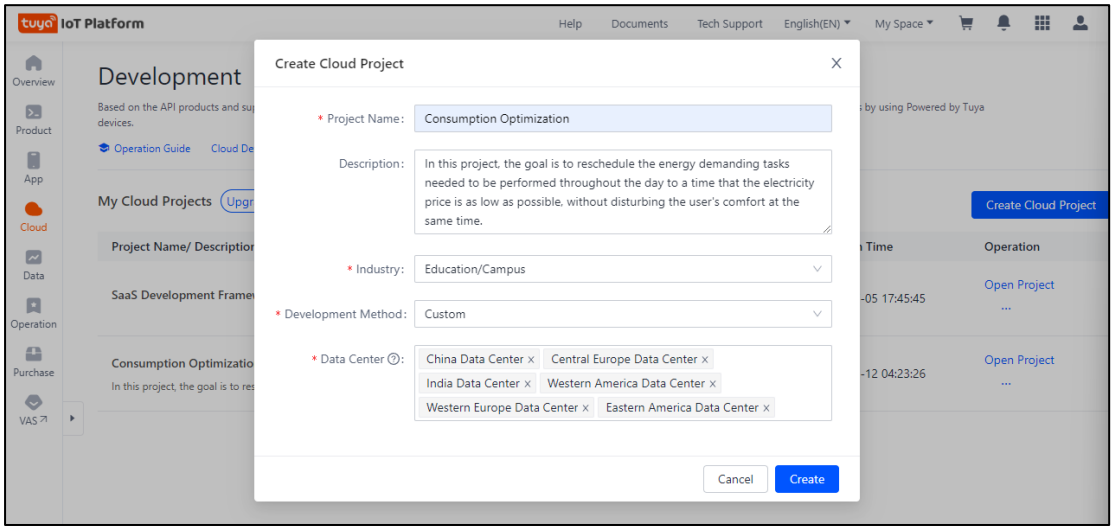
Με τη δημιουργία του λογαριασμού, αποκτάμε πρόσβαση στην πλατφόρμα ανάπτυξης IoT Tuya και μπορούμε να ξεκινήσουμε τη δημιουργία εφαρμογών IoT, χρησιμοποιώντας τα εργαλεία ανάπτυξης που παρέχει η πλατφόρμα, όπως αυτά φαίνονται στη στήλη στο αριστερό μέρος της οθόνης (βλ. Εικόνα 14).



Εικόνα 14. Περιβάλλον Tuya

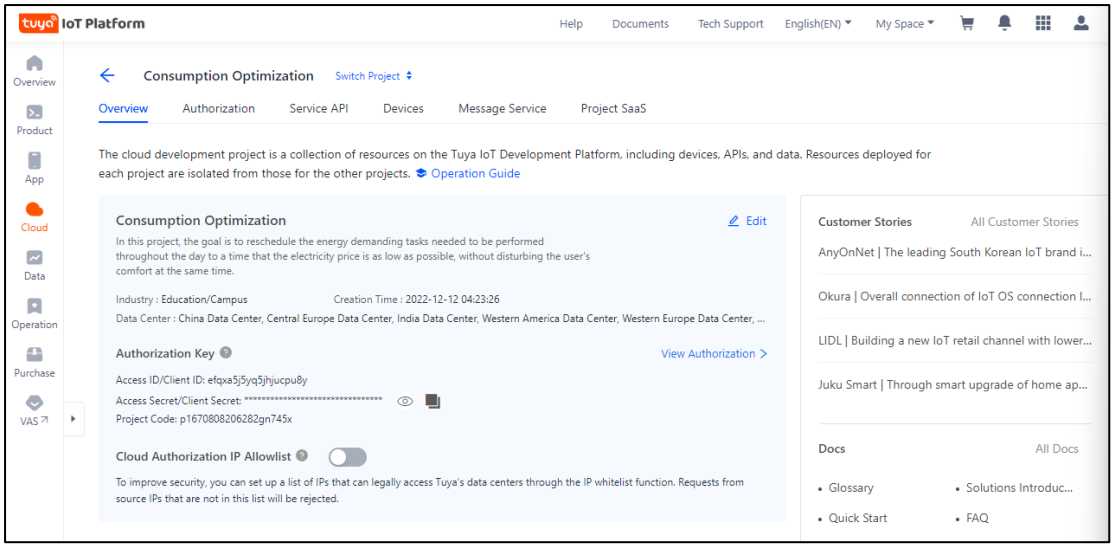
4.1.2 Δημιουργία Tuya Cloud Project

Στο αριστερό μέρος της οθόνης, πατάμε *Cloud* > *Development* και βρισκόμαστε στην οθόνη διαχείρισης των Cloud Project που έχουμε ήδη δημιουργήσει. Δημιουργούμε ένα νέο Project πατώντας το πλήκτρο *Create Cloud Project* πάνω δεξιά και συμπληρώνοντας τα απαιτούμενα στοιχεία (βλ. Εικόνα 15). Ονομάζουμε το Project «*Consumption Optimization*».



Εικόνα 15. Δημιουργία Tuya Cloud Project

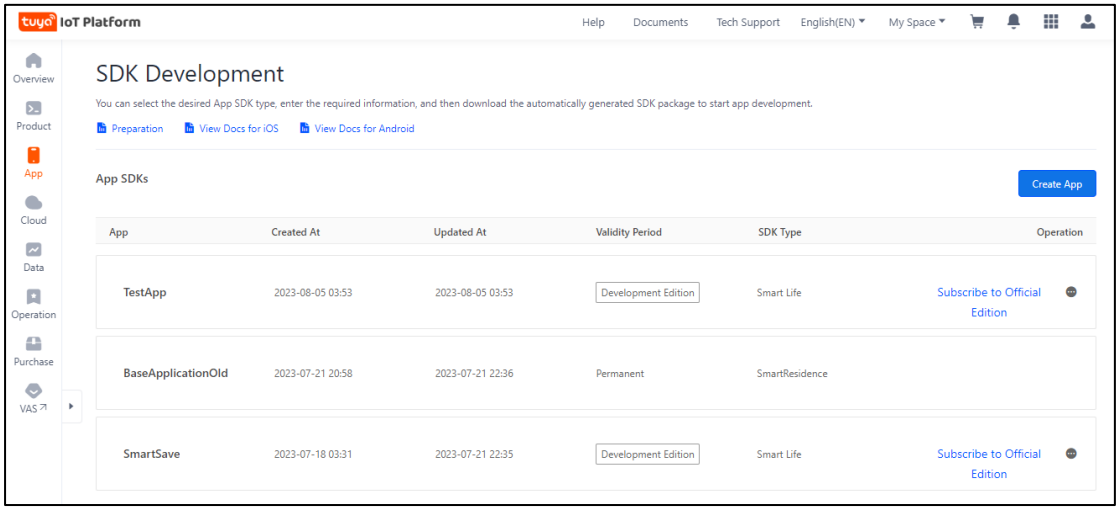
Μετά την δημιουργία του Project, πατάμε το πλήκτρο *Open Project* στο δεξί μέρος της οθόνης και βλέπουμε την καρτέλα Overview (βλ. Εικόνα 16). Αποθηκεύουμε το Access ID/Client ID και Access Secret/Client Secret που δημιουργήθηκαν αυτόματα από την Tuya, καθώς θα τα χρειαστούμε αργότερα.



Εικόνα 16. Καρτέλα Overview

4.1.3 Εκκίνηση Εφαρμογής στην πλατφόρμα Tuya

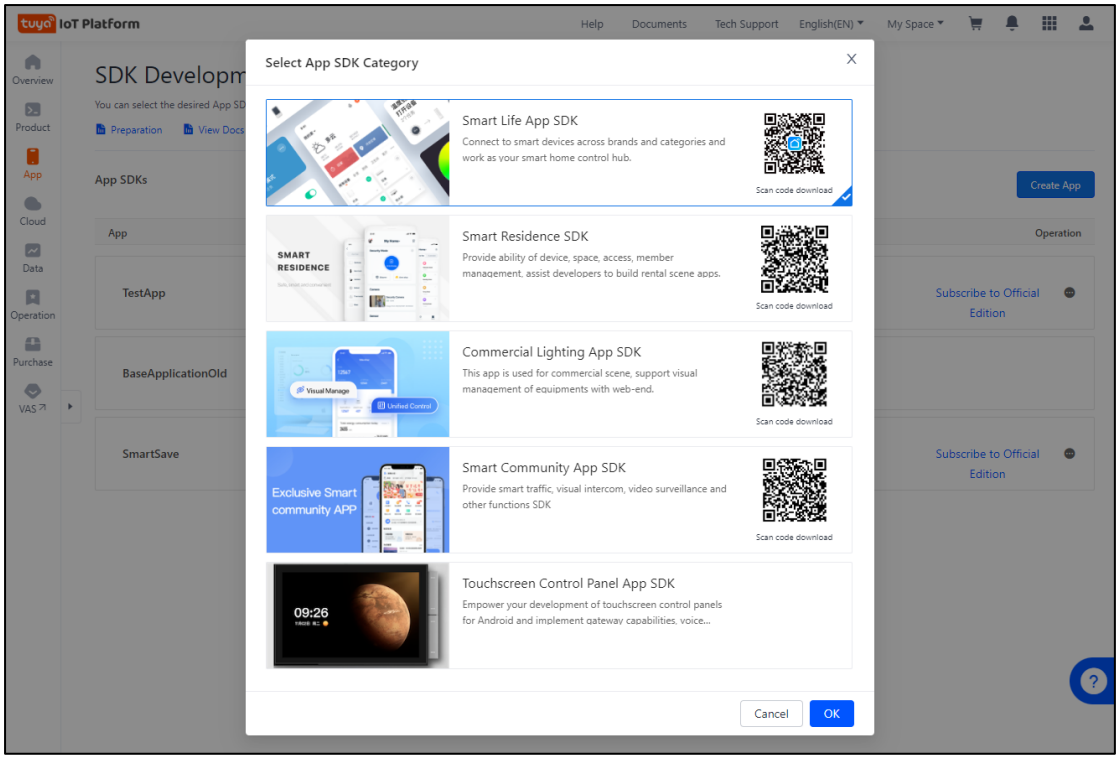
Στην πλατφόρμα ανάπτυξης Tuya IoT, στο αριστερό μέρος της οθόνης πατάμε *App* > *SDK Development* και βρισκόμαστε στην οθόνη διαχείρισης των εφαρμογών που έχουμε δημιουργήσει με τα πακέτα SDK που παρέχει η πλατφόρμα Tuya (βλ. Εικόνα 17).



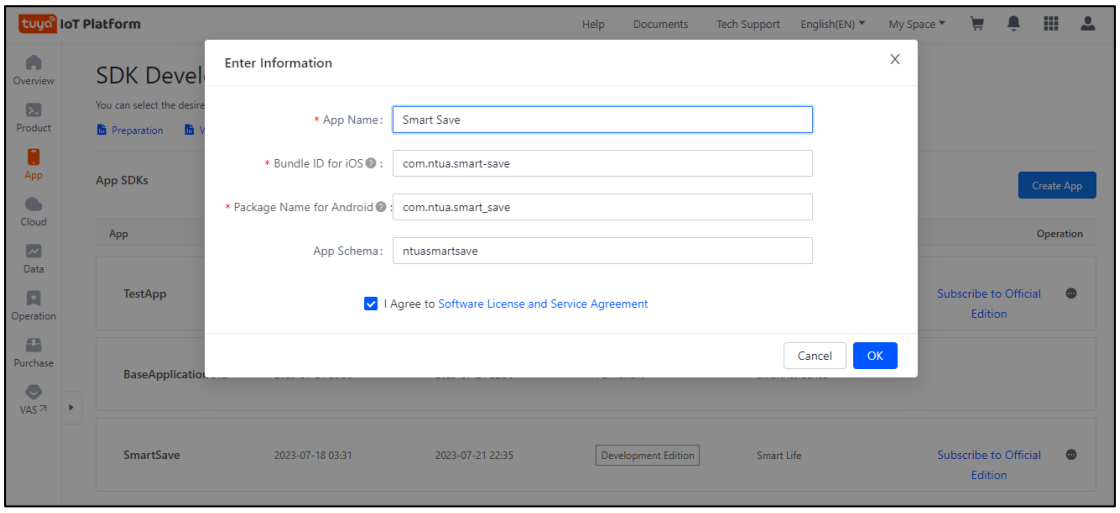
Εικόνα 17. SDK Development

Εδώ, επιλέγουμε τον επιθυμητό τύπο App SDK, εισάγουμε τις απαιτούμενες πληροφορίες και στη συνέχεια κάνουμε λήψη του πακέτου SDK, το οποίο δημιουργείται αυτόματα από την Tuya για να ξεκινήσουμε την ανάπτυξη της εφαρμογής.

Πατάμε το πλήκτρο *Create App* πάνω δεξιά, επιλέγουμε *Smart Life App SDK* και συμπληρώνουμε τις απαραίτητες πληροφορίες (βλ. Εικόνες 18, 19).

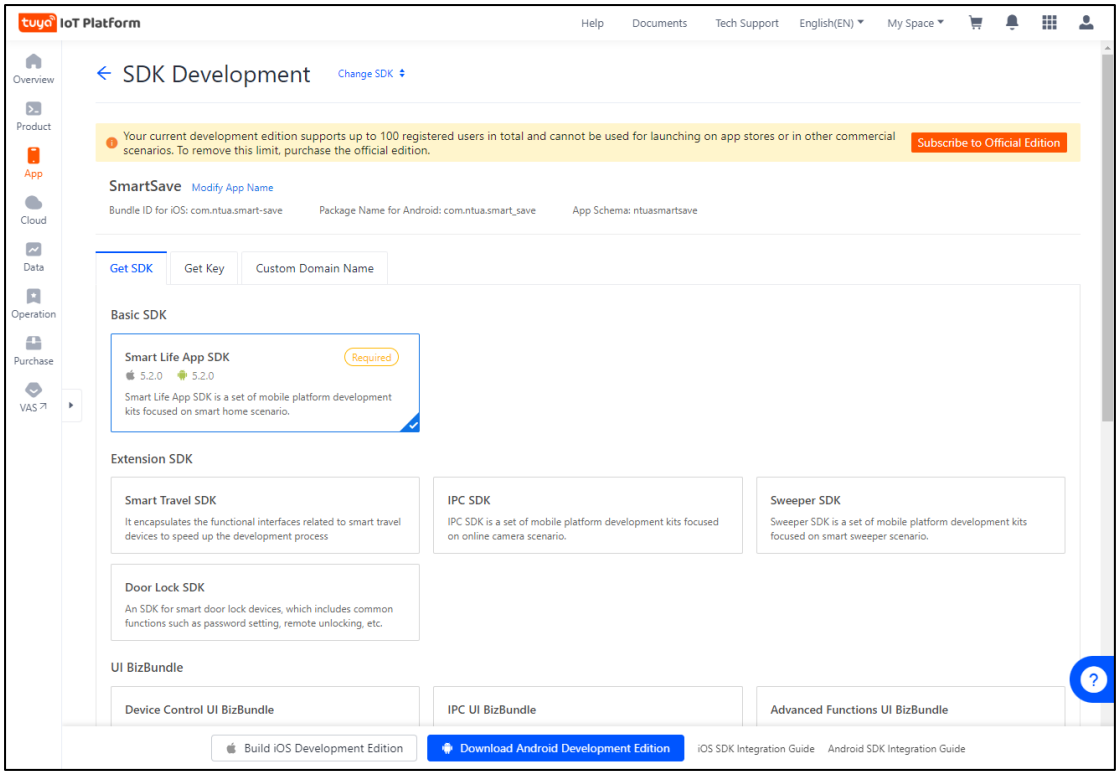


Εικόνα 18. Επιλογή Smart Life App SDK



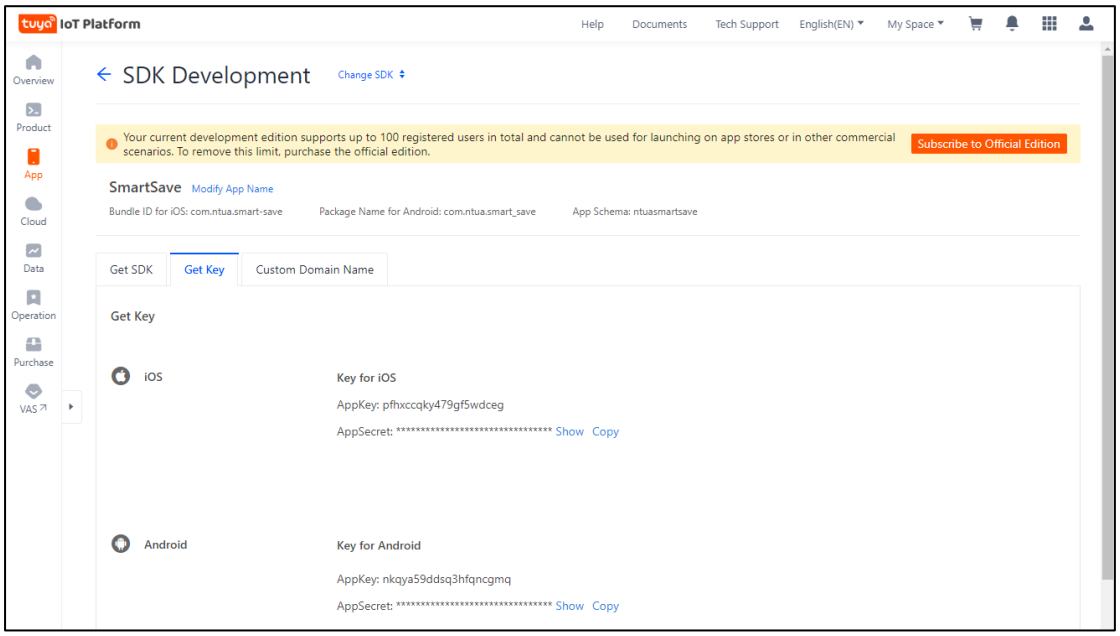
Εικόνα 19. Δημιουργία Smart Life App SDK

Στη συνέχεια, πατάμε πάνω στην εφαρμογή που μόλις δημιουργήθηκε και ενώ βρισκόμαστε στην καρτέλα *Get SDK*, στο κάτω μέρος της οθόνης πατάμε το πλήκτρο *Build Android Development Edition*. Η πλατφόρμα δημιουργεί το App SDK σύμφωνα με τις προδιαγραφές που επιλέξαμε και πατώντας το πλήκτρο *Download Android Development Edition* ξεκινάει η λήψη του SDK (βλ. Εικόνα 20).



Εικόνα 20. Λήψη Smart Life App SDK

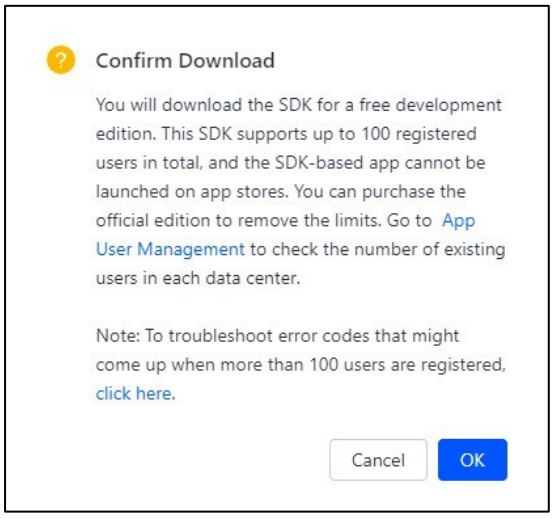
Επόμενο βήμα είναι στη διπλανή καρτέλα *Get Key* να αποθηκεύσουμε το *AppKey* και το *AppSecret* που αντιστοιχούν στο Android, καθώς θα τα χρειαστούμε αργότερα στην υλοποίηση της εφαρμογής (βλ. Εικόνα 21).



Εικόνα 21. Αποθήκευση AppKey και AppSecret

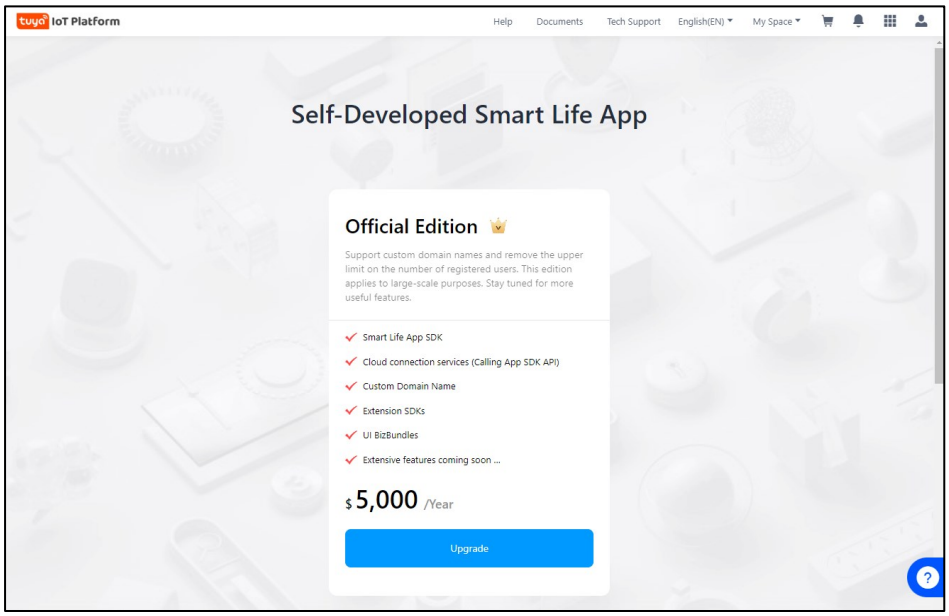
4.1.4 Έκδοση Ανάπτυξης και Δυνατότητα Αναβάθμισης

Η εφαρμογή που δημιουργήθηκε στα προηγούμενα βήματα αποτελεί μία έκδοση ανάπτυξης (*development edition*) και παρέχεται από την Tuya δωρεάν στους προγραμματιστές για την ανάπτυξη εφαρμογών θέτοντας κάποιους περιορισμούς (βλ. Εικόνα 22). Η έκδοση αυτή υποστηρίζει μέχρι 100 εγγεγραμμένους χρήστες, και επίσης δεν επιτρέπει τη δημοσίευση της εφαρμογής σε αγορές εφαρμογών (app stores). Το παρακάτω μήνυμα εμφανίζεται στον χρήστη πριν ξεκινήσει η λήψη του App SDK στη διαδικασία που περιγράφεται στην Ενότητα 4.1.3.



Εικόνα 22. Development Edition

Η επίσημη έκδοση παρέχει τις πλήρεις δυνατότητες του App SDK και προορίζεται για μεγαλύτερης κλίμακας σκοπούς. Είναι διαθέσιμη για αναβάθμιση με κόστος συνδρομής τα 5000$ ετησίως (βλ. Εικόνα 23).



Εικόνα 23. Official Edition

Τα πλεονεκτήματα που προσφέρει η επίσημη έκδοση σε σχέση με την βασική έκδοση ανάπτυξης φαίνονται στον παρακάτω πίνακα (βλ. Εικόνα 24). Για τους σκοπούς της παρούσας έρευνας είναι επαρκής η έκδοση ανάπτυξης.

SDK Benefits

Benefit	Development Edition	Official Edition 👑
Smart Life App SDK	✓	✓
Cloud connection services (Calling App SDK API)	✓ Up to 1000,000 calls	✓ Up to 100 million calls
Custom Domain Name	Not supported	✓
Extension SDKs	✓	✓
UI BizBundles	✓	✓
Total number of registered app users	100 or less	Unlimited

Εικόνα 24. Development Edition vs Official Edition

Περισσότερες πληροφορίες για τα προγράμματα χρέωσης της Tuya για προγραμματιστές είναι διαθέσιμες στον επίσημη ιστοσελίδα της Tuya, στην κατηγορία Developers – Pricing [39].

4.2 Εφαρμογή Android

Στην ενότητα αυτή περιγράφονται τα βήματα ανάπτυξης της εφαρμογής Android με τη χρήση του Android Studio.

4.2.1 Ενσωμάτωση του Tuya SDK

Πριν την υλοποίηση των βασικών λειτουργιών της εφαρμογής, είναι απαραίτητο να γίνει μία αρχική ρύθμιση κάποιων παραμέτρων, σύμφωνα με τις οδηγίες που παρέχονται από την Tuya.

4.2.1.1 Ρύθμιση Package Name

Στο αρχείο *build.gradle* ορίστηκε το *applicationID* να είναι ίδιο με το *Package Name for Android* που ορίστηκε στην ενότητα 4.1.3 (βλ. Εικόνα 25). Αν τα δύο ονόματα δεν είναι ίδια η εφαρμογή θα επιστρέφει μήνυμα σφάλματος κατά τη σύνδεση με το Tuya Cloud με κωδικό ILLEGAL_CLIENT_ID.

build.gradle (:app)

```
18    defaultConfig {
19        applicationId "com.ntua.smart_save"
20        minSdkVersion 23
21        targetSdkVersion 30
22        versionCode 1
23        versionName "1.0"
```

Εικόνα 25. Ρύθμιση Package Name

4.2.1.2 Ρύθμιση build.gradle

Προστέθηκαν στο αρχείο build.gradle τα dependencies που απαιτούνται από τον οδηγό της Tuya (βλ. Εικόνα 26).

```
android {
    defaultConfig {
        ndk {
            abiFilters "armeabi-v7a", "arm64-v8a"
        }
    }
    packagingOptions {
        pickFirst 'lib/*/libc++_shared.so' // An Android Archive (AAR) file contains an Android library
    }
}

configurations.all {
    exclude group: "com.thingclips.smart" ,module: 'thingsmart-modularCampAnno'
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.aar'])
    implementation 'com.alibaba:fastjson:1.1.67.android'
    implementation 'com.squareup.okhttp3:okhttp-urlconnection:3.14.9'

    // The latest stable App SDK for Android.
    implementation 'com.facebook.soloader:soloader:0.10.4+'
    implementation 'com.thingclips.smart:thingsmart:5.5.2'
}
```

Εικόνα 26. Ρύθμιση build.gradle - dependencies

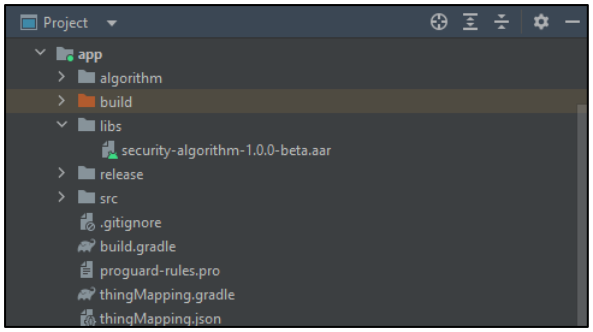
Προστέθηκε επίσης στο αρχείο build.gradle το Tuya IoT Maven repository URL στη λίστα με τα repositories (βλ. Εικόνα 27).

```
repositories {
    jcenter()
    maven { url 'https://maven-other.tuya.com/repository/maven-releases/' }
    maven { url "https://maven-other.tuya.com/repository/maven-commercial-releases/" }
    maven { url 'https://jitpack.io' }
    google()
    mavenCentral()
    maven { url 'https://maven.aliyun.com/repository/public' }
    maven { url 'https://central.maven.org/maven2/' }
    maven { url 'https://oss.sonatype.org/content/repositories/snapshots/' }
    maven { url 'https://developer.huawei.com/repo/' }
}
```

Εικόνα 27. Ρύθμιση build.gradle - repositories

4.2.1.3 Ενσωμάτωση Στοιχείου Ασφαλείας (Security Component)

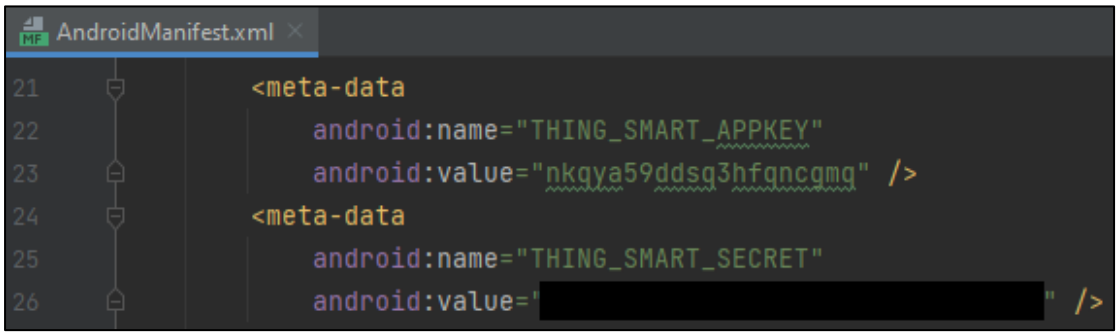
Στο App SDK που κάναμε λήψη στην ενότητα 4.1.3, εντοπίστηκε το αρχείο *security-algorithm-1.0.0-beta.aar* μέσα στο φάκελο *libs* του project και αντιγράφτηκε στον φάκελο *libs* του δικού μας project (βλ. Εικόνα 28).



Εικόνα 28. Security Component

4.2.1.4 Ρύθμιση AppKey και AppSecret

Στο αρχείο *AndroidManifest.xml* ορίστηκαν τα πεδία THING_SMART_APPKEY και THING_SMART_SECRET με τις τιμές των *AppKey* και *AppSecret* που αποθηκεύσαμε στην ενότητα 4.1.3 αντίστοιχα (βλ. Εικόνα 29).



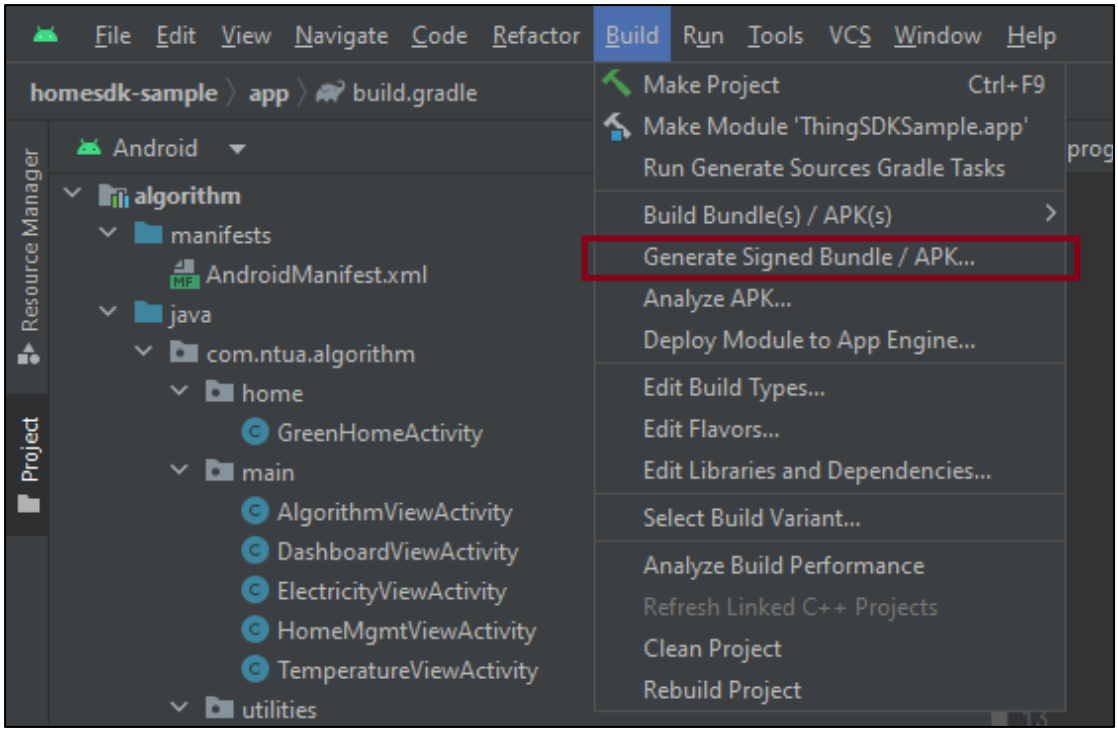
Εικόνα 29. Ρύθμιση Appkey και AppSecret

4.2.1.5 Προσθήκη Πιστοποιητικού SHA256

Για την ενίσχυση της ασφάλειας της εφαρμογής η Tuya απαιτεί υπογραφή της εφαρμογής με κλειδί SHA256.

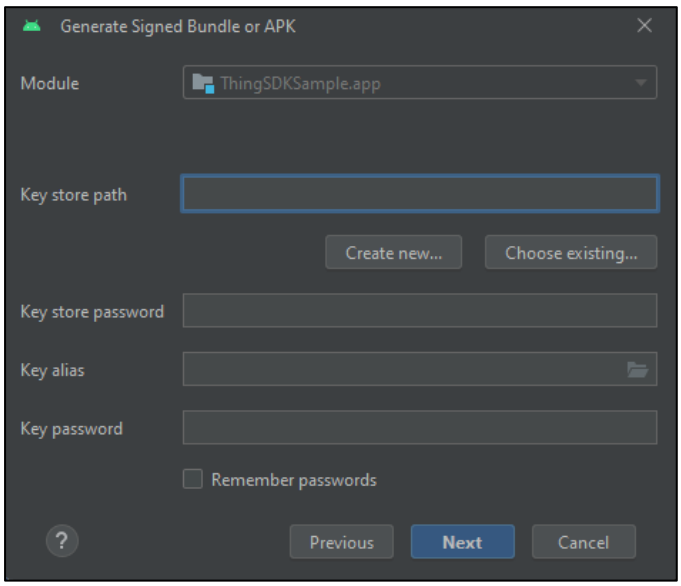
Το SHA256 είναι ένας τύπος ασφαλούς αλγόριθμου κατακερματισμού (secure hash algorithm). Τα δεδομένα που είναι κρυπτογραφημένα με τον αλγόριθμο SHA256 μετατρέπονται σε δυαδική τιμή σταθερού μεγέθους 256 bit. Η έξοδος ποικίλλει ανάλογα με την είσοδο και αυτό ενισχύει την ασφάλεια των δεδομένων. Για τον λόγο αυτό, το Smart Life App SDK και το Commercial Lighting App SDK χρησιμοποιούν κλειδιά SHA256.

Για τη ρύθμιση αυτή, μεταβαίνουμε στο Android Studio στο οποίο έχουμε ανοικτό το project μας και επιλέγουμε *Build* > *Generate Signed APK* (βλ. Εικόνα 30).



Εικόνα 30. Generate Signed APK

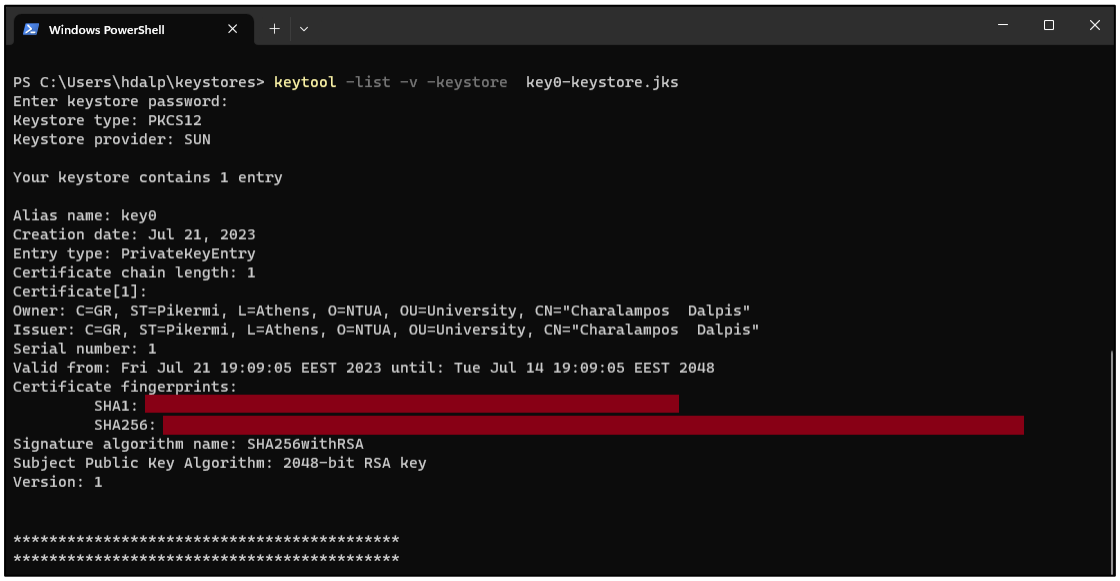
Επιλέξαμε APK και συμπληρώσαμε τα απαιτούμενα στοιχεία (βλ. Εικόνα 31) με αποτέλεσμα να δημιουργηθεί ένα αρχείο *keystore* που περιλαμβάνει την ψηφιακή υπογραφή με τα στοιχεία αυτά.



Εικόνα 31. Δημιουργία αρχείου ψηφιακής υπογραφής

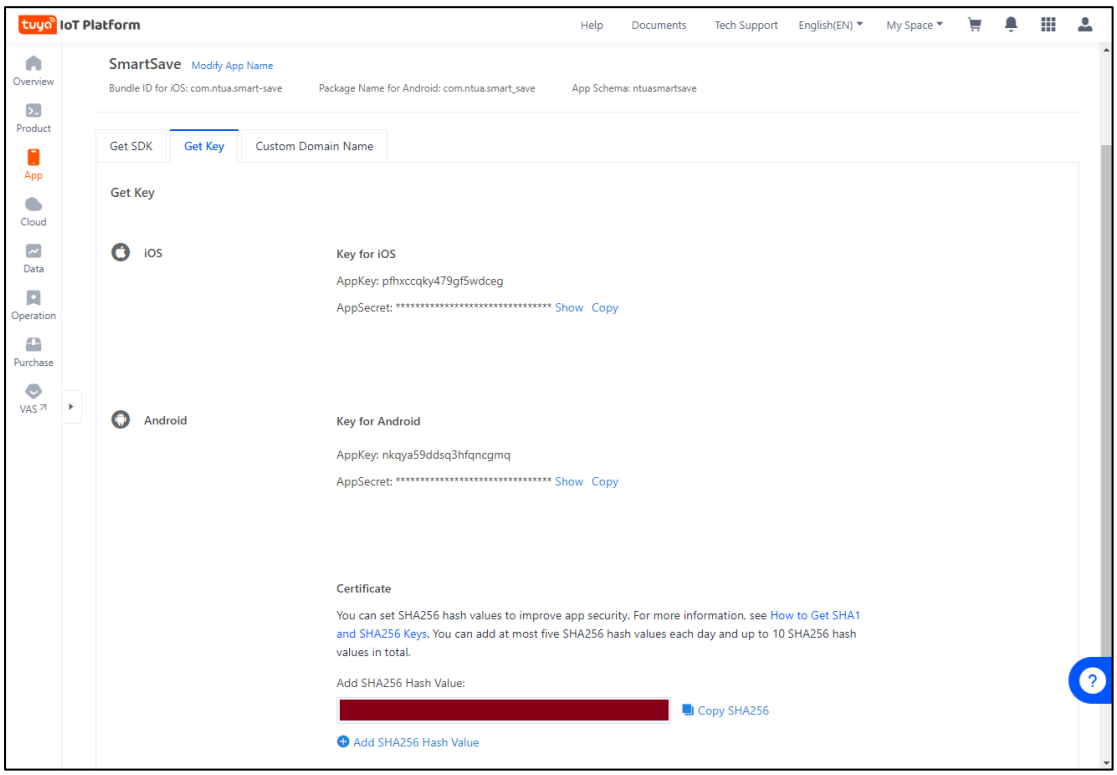
Στον φάκελο όπου αποθηκεύσαμε το αρχείο *keystore*, ανοίγουμε το *powershell* και εκτελούμε την εντολή *keytool -list -v -keystore key0-keystore.jks,* όπου *key0-keystore.jks* είναι το αρχείο *keystore* που δημιουργήθηκε παραπάνω. Αφού συμπληρώσουμε τον κωδικό που ορίσαμε προηγουμένως μπορούμε να δούμε τις

λεπτομέρειες του ιδιωτικού κλειδιού που περιλαμβάνει και συγκεκριμένα την τιμή SHA256 που είναι αυτό που μας ενδιαφέρει (βλ. Εικόνα 32).



Εικόνα 32. Πιστοποιητικό Sha256

Επιστρέφουμε στην πλατφόρμα ανάπτυξης Tuya IoT, στο αριστερό μέρος της οθόνης πατάμε *App* > *SDK Development* και επιλέγουμε το project μας. Στην καρτέλα *Get Key*, στο κάτω μέρος της οθόνης συμπληρώνουμε στο πεδίο του πιστοποιητικού το κλειδί SHA256 που αποκτήσαμε παραπάνω (βλ. Εικόνα 33).



Εικόνα 33. Εισαγωγή πιστοποιητικού Sha256 στο Tuya project

Τέλος, συμπληρώνουμε στο αρχείο build.gradle τις παρακάτω γραμμές κώδικα (βλ. Εικόνα 34).

build.gradle (:app)

```
android {
    signingConfigs {
        debug {
            storeFile file('C:/Users/hdalp/keystores/key0-keystore.jks')
            storePassword '[redacted]'
            keyAlias 'key0'
            keyPassword '[redacted]'
        }
    }
```

Εικόνα 34. Ενημέρωση αρχείου build.gradle

Μετά τις τροποποιήσεις που έγιναν στις ενότητες 4.2.1.1 μέχρι και 4.2.1.5 το αρχείο build.gradle έχει διαμορφωθεί όπως παρακάτω (βλ. Εικόνες 35, 36, 37).

build.gradle (:app)

```
plugins {
    id 'com.android.application'

}
apply from : "thingMapping.gradle"

android {
    signingConfigs {
        debug {
            storeFile file('C:/Users/hdalp/keystores/key0-keystore.jks')
            storePassword '[redacted]'
            keyAlias 'key0'
            keyPassword '[redacted]'
        }
    }
    compileSdkVersion 33

    defaultConfig {
        applicationId "com.ntua.smart_save"
        minSdkVersion 23
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"

        Properties properties = new Properties()
        properties.load(project.rootProject.file('local.properties').newDataInputStream())
        manifestPlaceholders = [
                TUYA_SMART_APPKEY: "${properties.getProperty("appKey")}",
                TUYA_SMART_SECRET: "${properties.getProperty("appSecret")}",
        ]
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"

        ndk {
            abiFilters "armeabi-v7a", "arm64-v8a"
        }
    }
```

Εικόνα 35. Τελική μορφή του αρχείου build.gradle (1/3)

```
    packagingOptions {
        pickFirst 'lib/*/libc++_shared.so'
        pickFirst 'lib/*/libyuv.so'
        pickFirst 'lib/*/libopenh264.so'
        pickFirst 'lib/*/liblog.so'
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    namespace 'com.tuya.appsdk.sample'


}

configurations.all {
    exclude group: "com.thingclips.smart" ,module: 'thingsmart-modularCampAnno'
}
```

Εικόνα 36. Τελική μορφή του αρχείου build.gradle (2/3)

```
dependencies {

    implementation 'androidx.core:core-ktx:1.3.2'
    implementation 'androidx.appcompat:appcompat:1.6.1'
    implementation 'com.google.android.material:material:1.3.0-alpha04'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    implementation 'androidx.navigation:navigation-fragment-ktx:2.2.2'
    implementation 'androidx.navigation:navigation-ui-ktx:2.2.2'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'

    implementation 'com.alibaba:fastjson:1.1.67.android'
    implementation 'com.squareup.okhttp3:okhttp-urlconnection:3.14.9'


    [redacted]


    implementation fileTree(dir: 'libs', include: ['*.aar'])
    implementation 'com.facebook.soloader:soloader:0.10.5'
    implementation 'com.thingclips.smart:thingsmart:5.1.0'
    implementation 'com.thingclips.smart:thingsmart-logsdk:5.0.0'
    implementation 'com.thingclips.smart:thing-log-sdk:5.0.0'
}
```

Εικόνα 37. Τελική μορφή του αρχείου build.gradle (3/3)

4.2.1.6 Ρύθμιση proguard-rules.pro

Στο αρχείο proguard-rules.pro προσθέτουμε τις παρακάτω γραμμές (βλ. Εικόνα 38).

```
proguard-rules.pro
#fastJson
-keep class com.alibaba.fastjson.**{*;}
-dontwarn com.alibaba.fastjson.**

#mqtt
-keep class com.thingclips.smart.mqttclient.mqttv3.** { *; }
-dontwarn com.thingclips.smart.mqttclient.mqttv3.**

#OkHttp3
-keep class okhttp3.** { *; }
-keep interface okhttp3.** { *; }
-dontwarn okhttp3.**

#Okio
-keep class okio.** { *; }
-dontwarn okio.**

#Tuya
-keep class com.thingclips.**{*;}
-dontwarn com.thingclips.**

# Matter SDK
-keep class chip.** { *; }
-dontwarn chip.**
```

Εικόνα 38. Ρύθμιση proguard-rules.pro

4.2.2 Αρχικές Ρυθμίσεις και Οργάνωση

Αρχικά για να μπορέσουν να λειτουργήσουν σωστά στη συσκευή οι συναρτήσεις που θα υλοποιήσουμε, πρέπει να έχουν οριστεί στο αρχείο AndroidManifest.xml οι απαραίτητες άδειες, όπως φαίνονται παρακάτω (βλ. Εικόνα 39).

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

    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.POST_NOTIFICATIONS" />
```

Εικόνα 39. Άδειες στο αρχείο AndroidManifest.xml

Συγκεκριμένα η άδεια *READ_EXTERNAL_STORAGE* χρειάζεται για να έχει προσβάση η εφαρμογή στην ανάγνωση εξωτερικών αρχείων στα οποία βρίσκονται αποθηκευμένα απαραίτητα δεδομένα για την λειτουργία της εφαρμογής. Για παράδειγμα τέτοια αρχεία είναι: το αρχείο *electricity_price_log.xml* που περιλαμβάνει

τις παλιές τιμές του ηλεκτρικού ρεύματος και το αρχείο *water_heater_log.xml* που περιλαμβάνει τις καταγραφές χρήσης του θερμοσίφωνα.

Επίσης, η άδεια *BLUETOOTH* και *BLUETOOTH_ADMIN* χρειάζεται για να μπορέσει η εφαρμογή να έχει πρόσβαση στον αισθητήρα Bluetooth και να συνδεθεί με τον μετρητή θερμοκρασίας και υγρασίας ANDWOL AS90W.

Ακόμα, χρειάζεται η άδεια *POST_NOTIFICATIONS* για να μπορεί η εφαρμογή να εμφανίζει ειδοποίηση στον χρήστη κάθε φορά που ο αλγόριθμος λαμβάνει μια απόφαση.

Στη συνέχεια του αρχείου *AndroidManifest.xml* ορίζουμε τα βασικά χαρακτηριστικά της εφαρμογής, επιλέγουμε το όνομα Smart Save και ορίζουμε ως βασική κλάση υλοποίησης της εφαρμογής μας την κλάση *BaseApplication.java* (βλ. Εικόνα 40).

```
12      <application
13          android:allowBackup="true"
14          android:icon="@mipmap/ic_launcher"
15          android:label="Smart Save"
16          android:supportsRtl="true"
17          android:name=".BaseApplication"
18          android:theme="@style/Theme.MaterialComponents.Light.NoActionBar"
19          tools:replace="android:allowBackup,android:supportsRtl">
```

Εικόνα 40. Ορισμός Ονόματος και Κλάσης της εφαρμογής

Ο κώδικας του αρχείου BaseApplication.java παρατίθεται παρακάτω (βλ. Εικόνα 41).

BaseApplication.java

```
37  public final class BaseApplication extends Application {
        3 usages
38      private static final String TAG = "MainActivity";
39
40      @Override
41      public void onCreate() {
42          super.onCreate();
43          ThingHomeSdk.init( application: this);
44          ThingHomeSdk.setDebugMode(true);
45          ThingOptimusSdk.init( context: this);
46
47          // TemperatureMonitor is a Singleton
48          TemperatureMonitor temperatureMonitor = TemperatureMonitor.getInstance();
49          temperatureMonitor.initXML( path: "temp_log.xml",  context: this);
50      }
```

Εικόνα 41. Αρχείο BaseApplication.java

Στο αρχείο αυτό πραγματοποιείται η εκκίνηση των αντικειμένων *ThingHomeSdk* και *ThingOptimusSdk* όπως προβλέπεται από το Tuya SDK έτσι ώστε να μπορούν να χρησιμοποιηθούν οι συναρτήσεις που παρέχουν στο κύριο σώμα της εφαρμογής.

Επίσης, γίνεται δημιουργία και αρχικοποίηση ενός αντικειμένου *TemperatureMonitor*, μία κλάση που δημιουργήθηκε με σκοπό την παρακολούθηση της θερμοκρασίας στο εσωτερικό του θερμοσίφωνα από τον αισθητήρα θερμοκρασίας και υγρασίας (βλ. ΠΑΡΑΡΤΗΜΑ Α).

Στη συνέχεια, μεταξύ όλων των δραστηριοτήτων (activities) που ορίζονται στο αρχείο αυτό, δίνεται ιδιαίτερη προσοχή στον ορισμό της δραστηριότητας UserFuncActivity.java, η οποία αποτελεί την οθόνη εκκίνησης της εφαρμογής και παρέχει δυνατότητες εγγραφής και σύνδεσης των χρηστών. Ο κώδικας που ορίζει την οθόνη αυτή ως οθόνη εκκίνησης φαίνεται παρακάτω (βλ. Εικόνα 42).

```
35      <activity
36          android:name=".user.main.UserFuncActivity"
37          android:screenOrientation="portrait"
38          android:exported="true">
39          <intent-filter>
40              <action android:name="android.intent.action.MAIN" />
41              <category android:name="android.intent.category.LAUNCHER" />
42          </intent-filter>
43      </activity>
```

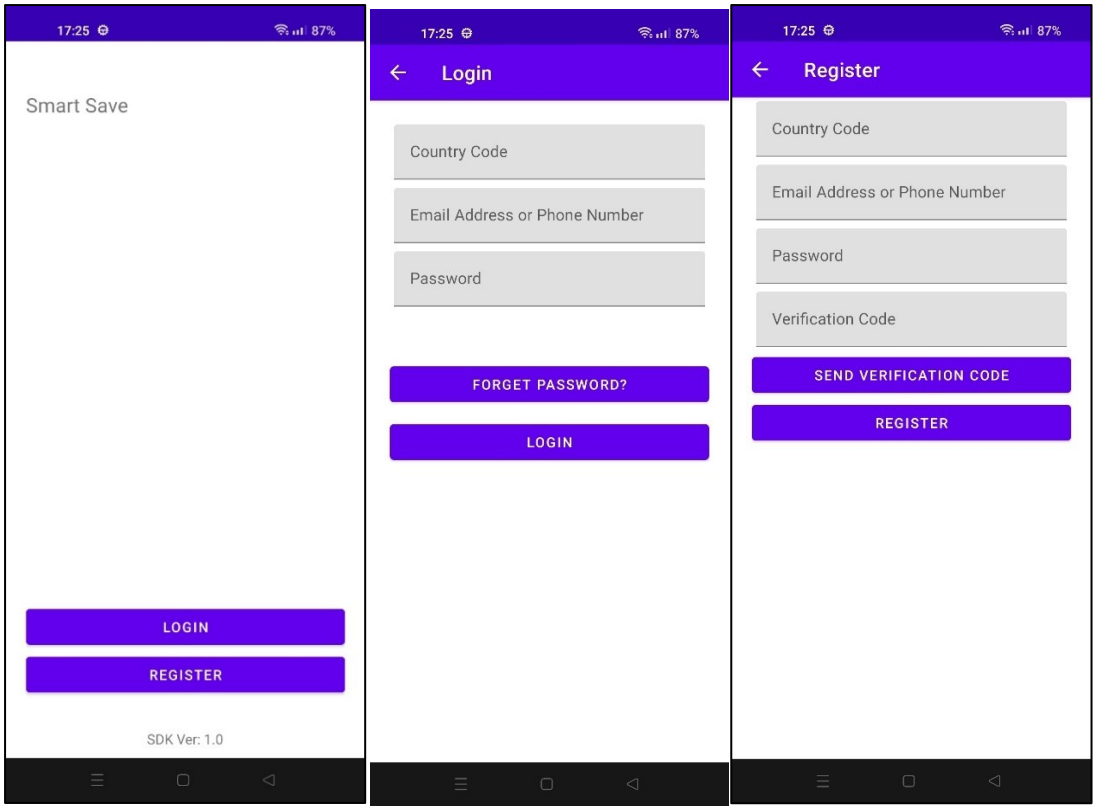
Εικόνα 42. Ορισμός οθόνης εκκίνησης της εφαρμογής

4.2.3 Εγγραφή και Σύνδεση Χρηστών

Κατά το πρώτο άνοιγμα της εφαρμογής, η πρώτη οθόνη που εμφανίζεται στον χρήστη, του δίνει την δυνατότητα να δημιουργήσει καινούριο λογαριασμό (register) ή να συνδεθεί αν έχει ήδη λογαριασμό (login).

Στην περίπτωση δημιουργίας νέου λογαριασμού, ο χρήστης καλείται να συμπληρώσει τα στοιχεία του: κωδικό χώρας (30 για Ελλάδα), e-mail ή κινητό τηλέφωνο και τον κωδικό του. Στην συνέχεια με το πλήκτρο *Send Verification Code* λαμβάνει κωδικό επιβεβαίωσης στο e-mail ή στο κινητό του τηλέφωνο, τον οποίο συμπληρώνει στο αντίστοιχο πεδίο και ολοκληρώνει την εγγραφή του στο σύστημα της Tuya για σύνδεση σε αυτήν την εφαρμογή (βλ. Εικόνα 43). Υπενθυμίζεται ότι η έκδοση του App SDK που χρησιμοποιήσαμε επιτρέπει μέχρι 100 εγγεγραμμένους χρήστες.

Στην περίπτωση σύνδεσης με ήδη υπάρχοντα λογαριασμό, ο χρήστης καλείται να συμπληρώσει τα στοιχεία του ξανά: κωδικό χώρα, e-mail ή κινητό τηλέφωνο και τον κωδικό του και πατώντας το πλήκτρο *Login* εισέρχεται στην αρχική σελίδα της εφαρμογής *Smart Save*.



Εικόνα 43. Εγγραφή και Σύνδεση χρηστών

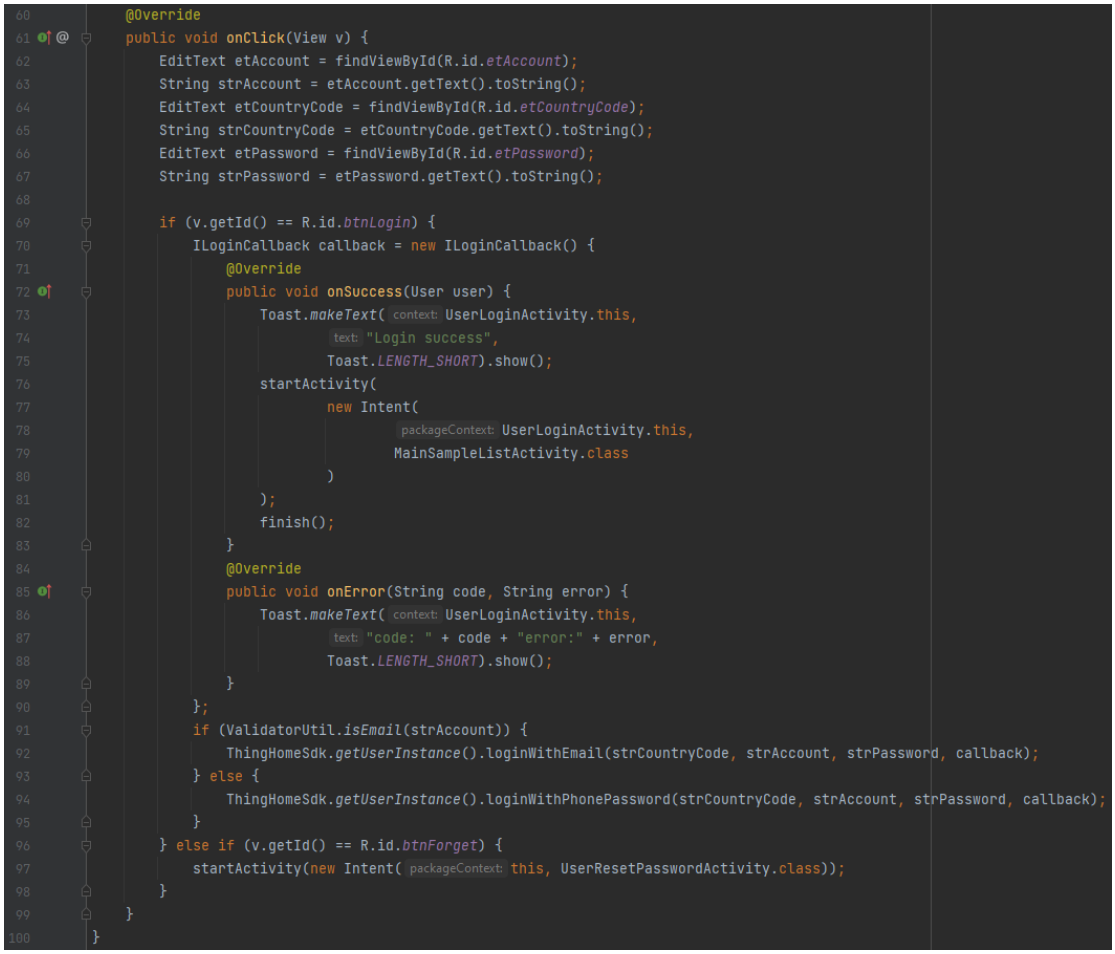
Για την υλοποίηση των λειτουργιών *Register* και *Login* τροποποιήθηκε η ήδη υπάρχουσα υλοποίηση στο *Tuya Android Smart Life App SDK Sample*.

Ο κώδικας που υλοποιεί τις παραπάνω λειτουργίες βρίσκεται στα αρχεία *UserRegisterActivity.java* και *UserLoginActivity.java*. Παρατίθεται ο κώδικας του *UserLoginActivity.java* (βλ. Εικόνες 44, 45).

UserLoginActivity.java

```
39  public class UserLoginActivity extends AppCompatActivity implements View.OnClickListener {
40      @Override
41      protected void onCreate(Bundle savedInstanceState) {
42          super.onCreate(savedInstanceState);
43          setContentView(R.layout.user_activity_login);
44
45          Toolbar toolbar = findViewById(R.id.topAppBar);
46          setSupportActionBar(toolbar);
47          toolbar.setNavigationOnClickListener(new View.OnClickListener() {
48              @Override
49              public void onClick(View v) { finish(); }
52          });
53
54          Button btnLogin = findViewById(R.id.btnLogin);
55          Button btnForget = findViewById(R.id.btnForget);
56          btnLogin.setOnClickListener(this);
57          btnForget.setOnClickListener(this);
58      }
```

Εικόνα 44. Σύνδεση χρήστη, UserLoginActivity.java (1/2)



```
    @Override
    public void onClick(View v) {
        EditText etAccount = findViewById(R.id.etAccount);
        String strAccount = etAccount.getText().toString();
        EditText etCountryCode = findViewById(R.id.etCountryCode);
        String strCountryCode = etCountryCode.getText().toString();
        EditText etPassword = findViewById(R.id.etPassword);
        String strPassword = etPassword.getText().toString();

        if (v.getId() == R.id.btnLogin) {
            ILoginCallback callback = new ILoginCallback() {
                @Override
                public void onSuccess(User user) {
                    Toast.makeText( context: UserLoginActivity.this,
                            text: "Login success",
                            Toast.LENGTH_SHORT).show();
                    startActivity(
                            new Intent(
                                    packageContext: UserLoginActivity.this,
                                    MainSampleListActivity.class
                            )
                    );
                    finish();
                }
                @Override
                public void onError(String code, String error) {
                    Toast.makeText( context: UserLoginActivity.this,
                            text: "code: " + code + "error:" + error,
                            Toast.LENGTH_SHORT).show();
                }
            };
            if (ValidatorUtil.isEmail(strAccount)) {
                ThingHomeSdk.getUserInstance().loginWithEmail(strCountryCode, strAccount, strPassword, callback);
            } else {
                ThingHomeSdk.getUserInstance().loginWithPhonePassword(strCountryCode, strAccount, strPassword, callback);
            }
        } else if (v.getId() == R.id.btnForget) {
            startActivity(new Intent( packageContext: this, UserResetPasswordActivity.class));
        }
    }
}
```

Εικόνα 45. Σύνδεση χρήστη, UserLoginActivity.java (2/2)

Όταν ο χρήστης πατήσει το πλήκτρο Login, αφού έχει πρώτα συμπληρώσει τα στοιχεία του, καλείται η συνάρτηση επιστροφής κλήσης *ILoginCallback* (συνάρτηση του Tuya SDK). Η συνάρτηση αυτή είναι μια συνάρτηση επιστροφής κλήσης (callback function) η οποία αναφέρεται στον μηχανισμό χειρισμού του αποτελέσματος της λειτουργίας ταυτοποίησης της Tuya. Αν η επιστροφή της συνάρτησης είναι επιτυχής τότε ο χρήστης συνδέεται, προχωράει στην κεντρική οθόνη της εφαρμογής και εμφανίζεται το μήνυμα *Login Success* στην οθόνη. Αν η επιστροφή της συνάρτησης είναι αποτυχημένη, τότε εμφανίζεται στην οθόνη το μήνυμα του σφάλματος που προέκυψε. Με όμοιο τρόπο λειτουργεί και η κλάση *UserRegisterActivity*.

4.2.4 Αρχική οθόνη εφαρμογής (Home Menu)

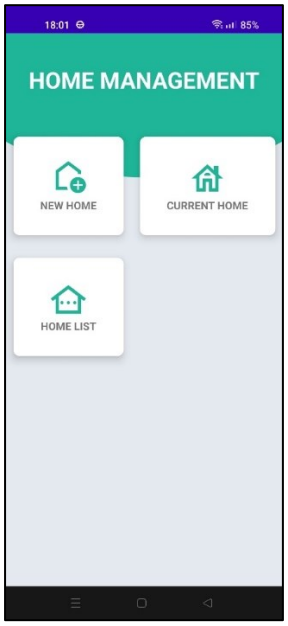
Μόλις ολοκληρώσει τη σύνδεση στην πρώτη οθόνη, ο χρήστης εισέρχεται στην αρχική οθόνη της εφαρμογής όπου μπορεί να δει συγκεντρωμένες όλες τις διαθέσιμες λειτουργίες (βλ. Εικόνα 46).

Εικόνα 46. Αρχική οθόνη εφαρμογής

4.2.4.1 Ορισμός Οικίας (Home Settings)

Το App SDK της Tuya ορίζει κάποιους κανόνες για τη σύνδεση και τη διαχείριση των έξυπνων συσκευών μέσω της εφαρμογής. Αρχικά, για να μπορέσει να συνδεθεί μία συσκευή θα πρέπει πρώτα να έχει δημιουργηθεί μία οικία και στη συνέχεια να οριστεί η οικία αυτή ως *τρέχουσα οικία*. Στις ρυθμίσεις αυτές θα μπορεί να έχει πρόσβαση ο χρήστης μέσω της οθόνης Home Settings (βλ. Εικόνα 47).

Πατώντας το πλήκτρο *Home Settings* ο χρήστης κατευθύνεται σε μια νέα οθόνη όπου μπορεί να δημιουργήσει νέα οικία, να αλλάξει την τρέχουσα οικία και να δει αναλυτικά τις λεπτομέρειες για όλες τις οικίες που έχει καταχωρίσει (βλ. Εικόνα 48).

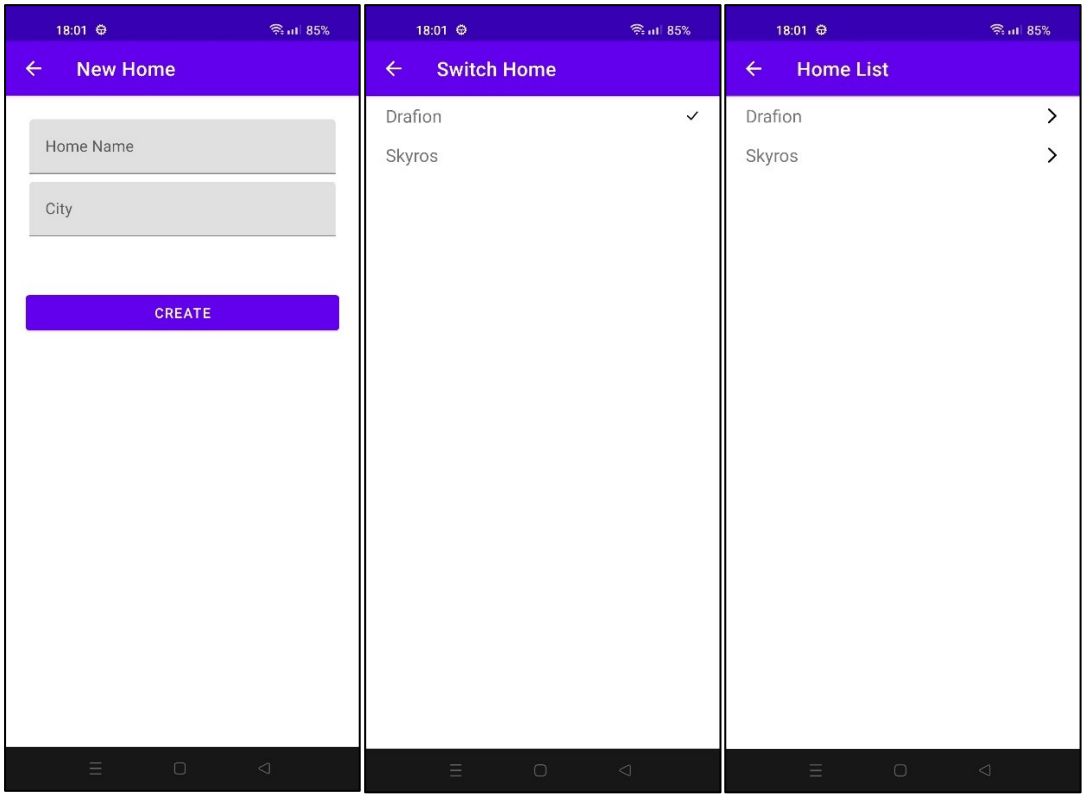


Εικόνα 47. Home Settings

Με την επιλογή *New Home* ο χρήστης κατευθύνεται σε νέα οθόνη, όπου συμπληρώνει τα στοιχεία της νέας οικίας.

Με την επιλογή *Current Home* ο χρήστης κατευθύνεται σε νέα οθόνη, όπου μπορεί να επιλέξει και να ορίσει ως τρέχουσα μια οικία από τη λίστα με όλες τις οικίες που έχει δημιουργήσει. Ο ορισμός της τρέχουσας οικίας είναι σημαντικός, καθώς όταν στη συνέχεια συνδέσουμε τη συσκευή μας, αυτή συσχετίζεται αυτόματα με την τρέχουσα οικία κατά τη στιγμή της σύνδεσης. Με αυτόν τον τρόπο μπορεί ο χρήστης να έχει διαφορετικές συσκευές σε διαφορετικές οικίες του και να είναι διαχωρισμένες μεταξύ τους.

Με την επιλογή *Home List* ο χρήστης κατευθύνεται σε νέα οθόνη, όπου μπορεί να δει τη λίστα με όλες τις οικίες που έχει δημιουργήσει και να δει λεπτομέρειες για οποιαδήποτε από αυτές πατώντας πάνω της.



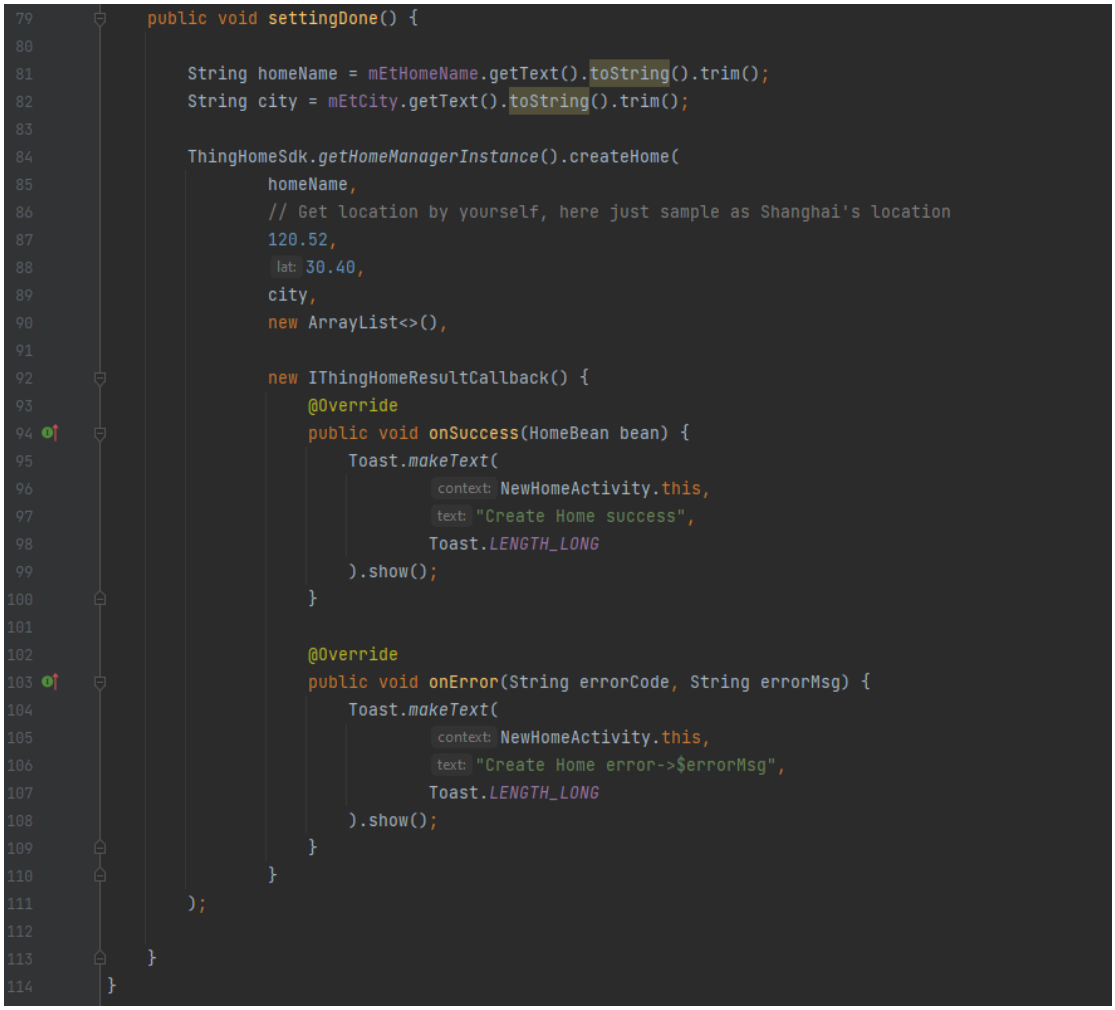
Εικόνα 48. Home Management

Ο κώδικας που υλοποιεί τις παραπάνω λειτουργίες βρίσκεται στα αρχεία *NewHomeActivity.java* και *HomeListActivity.java*. Παρατίθεται ο κώδικας του αρχείου *NewHomeActivity.java* για τη δημιουργία νέας οικίας (βλ. Εικόνες 49, 50).

NewHomeActivity.java

```java
public class NewHomeActivity extends AppCompatActivity implements View.OnClickListener {
    2 usages
    private Toolbar mToolbar;
    2 usages
    private Button mBtDone;
    2 usages
    private TextInputEditText mEtHomeName, mEtCity;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.home_activity_new_home);

        initView();
        initData();
    }
    1 usage
    private void initView() {
        mToolbar = findViewById(R.id.topAppBar);
        mBtDone = findViewById(R.id.btnDone);


        mEtHomeName = findViewById(R.id.etHomeName);
        mEtCity = findViewById(R.id.etCity);

    }
    1 usage
    private void initData() {
        mToolbar.setNavigationOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) { finish(); }
        });

        mBtDone.setOnClickListener(this);
    }
    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.btnDone) {
            settingDone();
        }
    }
```

Εικόνα 49. Δημιουργία νέας οικίας (1/2)



```
79      public void settingDone() {
80
81          String homeName = mEtHomeName.getText().toString().trim();
82          String city = mEtCity.getText().toString().trim();
83
84          ThingHomeSdk.getHomeManagerInstance().createHome(
85                  homeName,
86                  // Get location by yourself, here just sample as Shanghai's location
87                  120.52,
88                   lat: 30.40,
89                  city,
90                  new ArrayList<>(),
91
92                  new IThingHomeResultCallback() {
93                      @Override
94                      public void onSuccess(HomeBean bean) {
95                          Toast.makeText(
96                                  context: NewHomeActivity.this,
97                                  text: "Create Home success",
98                                  Toast.LENGTH_LONG
99                          ).show();
100                     }
101
102                     @Override
103                     public void onError(String errorCode, String errorMsg) {
104                         Toast.makeText(
105                                 context: NewHomeActivity.this,
106                                 text: "Create Home error->$errorMsg",
107                                 Toast.LENGTH_LONG
108                         ).show();
109                     }
110                 }
111         );
112
113     }
114 }
```

Εικόνα 50. Δημιουργία νέας οικίας (2/2)

Για την υλοποίηση των δύο αυτών λειτουργιών χρησιμοποιήθηκαν τροποποιημένα μέρη της ήδη υπάρχουσας υλοποίησης για δημιουργία και διαχείριση οικιών από το *Tuya Android Smart Life App SDK Sample.*

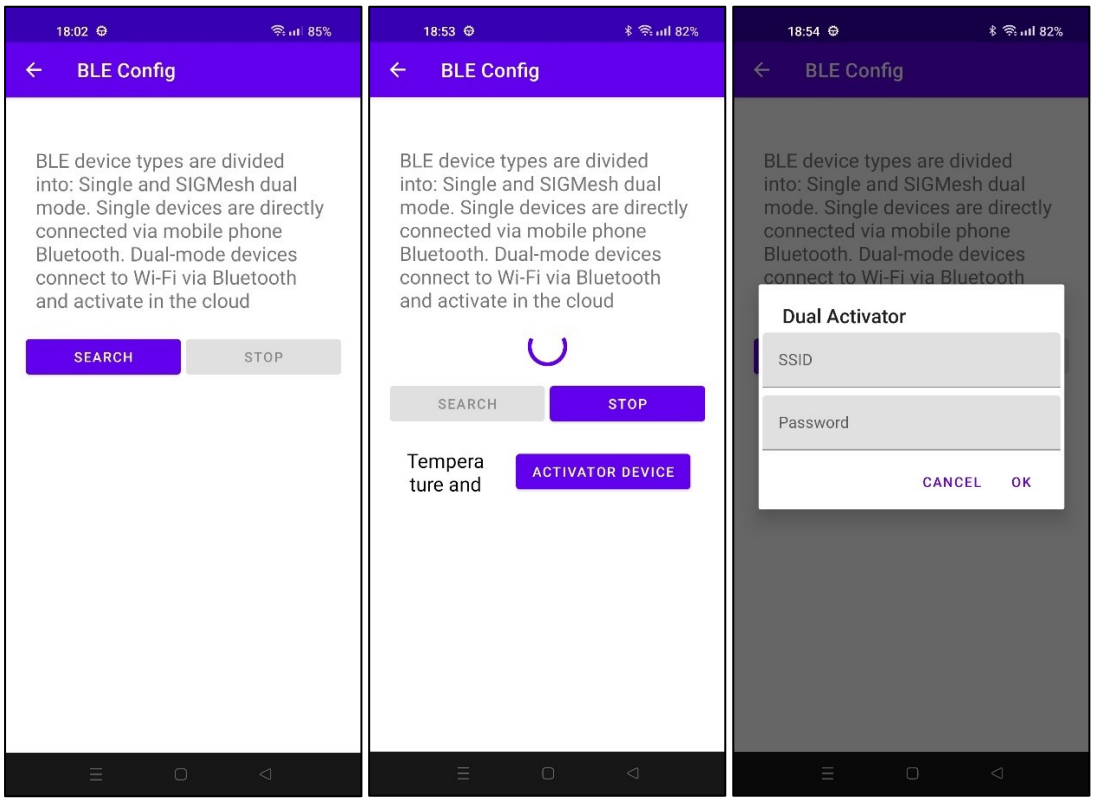
4.2.4.2 Σύνδεση Συσκευής (Connect Device)

Αφού ο χρήστης δημιουργήσει μία οικία και την ορίσει ως τρέχουσα μπορεί στη συνέχεια να συνδέσει μία έξυπνη συσκευή. Στην παρούσα έρευνα θα συνδέσουμε τον μετρητή θερμοκρασίας και υγρασίας ANDWOL AS90W με την εφαρμογή.

Πατώντας το πλήκτρο *Connect Device* ο χρήστης κατευθύνεται σε μια νέα οθόνη όπου μπορεί να πραγματοποιήσει τη σύνδεση του μετρητή με την εφαρμογή μέσω Bluetooth (βλ. Εικόνα 51). Τα βήματα για τη σύνδεση της συσκευής είναι τα παρακάτω:

1. Βάζουμε τον μετρητή ANDWOL AS90W σε λειτουργία σύζευξης, πατώντας παρατεταμένα το πλήκτρο στο πίσω μέρος της συσκευής (το πράσινο λαμπάκι στο μπροστινό μέρος της συσκευής θα αρχίσει να αναβοσβήνει)
2. Ανοίγουμε το Bluetooth στο κινητό μας

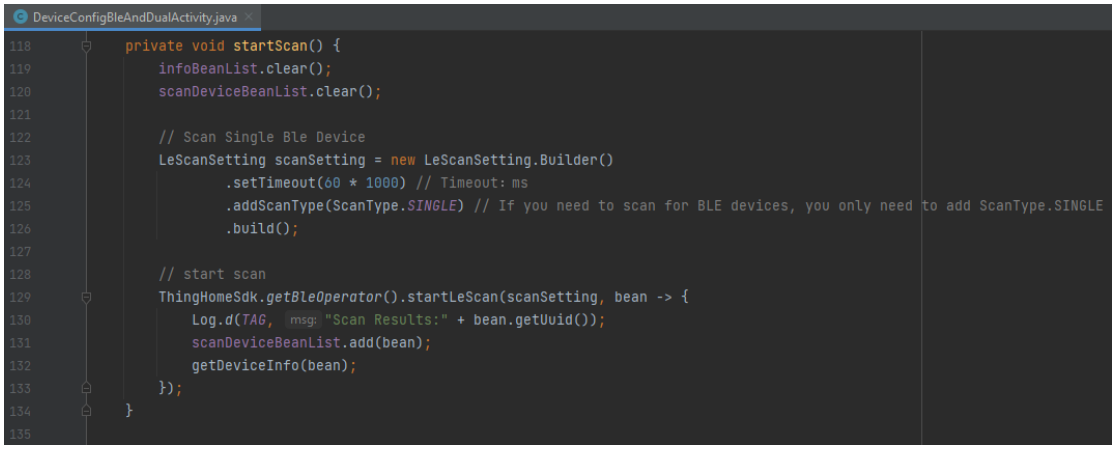
3. Πατάμε το πλήκτρο *Search Device* και περιμένουμε μέχρι η συσκευή μας να εμφανιστεί στην λίστα με τις διαθέσιμες συσκευές
4. Μόλις εμφανιστεί η συσκευή, πατάμε το πλήκτρο *Activator Device*
5. Συμπληρώνουμε τα στοιχεία *SSID* και *Password* του δικτύου Wi-Fi του σπιτιού στο οποίο θα συνδεθεί η συσκευή και πατάμε OK
6. Περιμένουμε μέχρι να ολοκληρωθεί η διαδικασία και να εμφανιστεί στο κάτω μέρος της οθόνης το μήνυμα *config success*



Εικόνα 51. Σύνδεση συσκευής μέσω Bluetooth

Ο κώδικας που υλοποιεί τις παραπάνω λειτουργίες βρίσκεται στο αρχείο *DeviceConfigBleAndDualActivity.java.* Στη συνέχεια παρατίθενται κάποια κρίσιμα κομμάτια από τον κώδικα αυτού του αρχείου.

Πατώντας το πλήκτρο *Search*, εκετελείται η συνάρτηση *startScan* (βλ. Εικόνα 52) η οποία ξεκινάει την αναζήτηση συσκευών Bluetooth που διαρκεί 1 λεπτό. Αυτό το επιτυγχάνει καλώντας με τη σειρά της τη συνάρτηση *startLeScan* πάνω στον Bluetooth Operator (δομές υλοποιημένες μέσα στο Tuya SDK) και εμφανίζει τις συσκευές που εντόπισε με τις σχετικές τους πληροφορίες, με την συνάρτηση *getDeviceInfo* (βλ. Εικόνα 53).



Εικόνα 52. Αναζήτηση συσκευών Bluetooth

Εικόνα 53. Εμφάνιση των συσκευών Bluetooth που εντοπίστηκαν

Αφού εντοπιστεί η σχετική συσκευή, με το πλήκτρο *Activator Device* καλείται η συνάρτηση *dualActivatorDialog*, η οποία εμφανίζει το μήνυμα διαλόγου που φαίνεται στην Εικόνα 51(3/3) και ζητάει από τον χρήστη να συμπληρώσει τις απαραίτητες πληροφορίες (βλ. Εικόνα 54).

```
private void dualActivatorDialog(int pos) {
    final View v = LayoutInflater.from( context: this).inflate(R.layout.ble_dual_activator_dialog,  root: null);
    TextInputEditText etSSID = v.findViewById(R.id.et_ssid);
    TextInputEditText etPwd = v.findViewById(R.id.et_pwd);
    AlertDialog.Builder builder = new AlertDialog.Builder( context: this);
    builder.setTitle("Dual Activator")
            .setView(v)
            .setNegativeButton( text: "Cancel", (dialog, which) -> dialog.dismiss())
            .setPositiveButton( text: "Ok", (dialog, which) -> {
                String ssid = Objects.requireNonNull(etSSID.getText()).toString();
                if (TextUtils.isEmpty(ssid)) {
                    Toast.makeText( context: DeviceConfigBleAndDualActivity.this,  text: "SSID is Null", Toast.LENGTH_SHORT).show();
                    Log.d(TAG,  msg: "SSID is Null");
                }
                // Wi-Fi password can be null
                String pwd = Objects.requireNonNull(etPwd.getText()).toString();

                startDualActivator(pos, ssid, pwd);
            });
    builder.show();

}
```

Εικόνα 54. Εισαγωγή SSID και password

Τέλος, μετά την επιτυχή συμπλήρωση των στοιχείων SSID και password από τον χρήστη, εκτελείται η συνάρτηση *startDualActivator* η οποία επιτυγχάνει τη σύνδεση με τη συσκευή και εγκαθιδρύει έναν δίαυλο επικοινωνίας και ανταλλαγής δεδομένων με την εφαρμογή. Η συνάρτηση αυτή είναι υλοποιημένη μέσα στο Tuya SDK και ο κώδικας της παρατίθεται παρακάτω (βλ. Εικόνες 55, 56).

Για την υλοποίηση αυτής της λειτουργίας χρησιμοποιήθηκαν τροποποιημένα μέρη της ήδη υπάρχουσας υλοποίησης για σύνδεση συσκευών BLE από το *Tuya Android Smart Life App SDK Sample.*

```
private void startDualActivator(int pos, String ssid, String pwd) {
    cpiLoading.setVisibility(View.VISIBLE);
    long homeId = HomeModel.getCurrentHome( context: this);
    ThingHomeSdk.getActivatorInstance().getActivatorToken(homeId,
            new IThingActivatorGetToken() {

                // get Token
                @Override
                public void onSuccess(String token) {
                    Log.d(TAG,  msg: "getToken success, token :" + token);
                    multiModeActivatorBean = new MultiModeActivatorBean();
                    multiModeActivatorBean.deviceType = scanDeviceBeanList.get(pos).getDeviceType();
                    multiModeActivatorBean.uuid = scanDeviceBeanList.get(pos).getUuid();
                    multiModeActivatorBean.address = scanDeviceBeanList.get(pos).getAddress();
                    multiModeActivatorBean.mac = scanDeviceBeanList.get(pos).getMac();
                    multiModeActivatorBean.ssid = ssid;
                    multiModeActivatorBean.pwd = pwd;
                    multiModeActivatorBean.token = token;
                    multiModeActivatorBean.homeId = homeId;
                    multiModeActivatorBean.timeout = 120 * 1000;
```

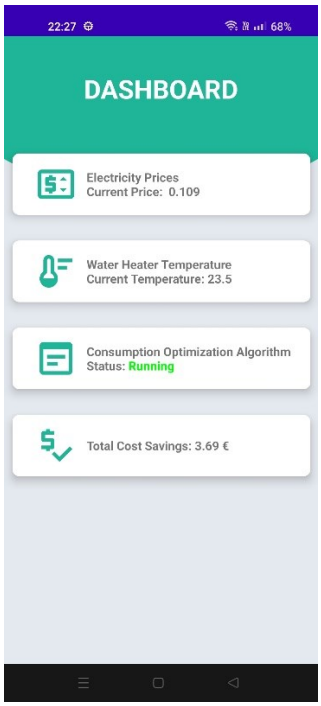
Εικόνα 55. Συνάρτηση σύνδεσης συσκευής με την εφαρμογή (1/2)

```
                // start activator
                mMultiModeActivator.startActivator(multiModeActivatorBean, new IMultiModeActivatorListener() {
                    @Override
                    public void onSuccess(DeviceBean deviceBean) {
                        if (deviceBean != null) {
                            Toast.makeText( context: DeviceConfigBleAndDualActivity.this,  text: "config success", Toast.LENGTH_SHORT).show();
                            Log.d(TAG,  msg: "Success:" + deviceBean.getName());
                            cpiLoading.setVisibility(View.GONE);

                        }
                        multiModeActivatorBean = null;
                    }

                    @Override
                    public void onFailure(int code, String msg, Object handle) {
                        Log.d(TAG,  msg: "error:" + msg);
                        Toast.makeText( context: DeviceConfigBleAndDualActivity.this,  text: "error:" + msg, Toast.LENGTH_SHORT).show();
                        cpiLoading.setVisibility(View.GONE);
                        multiModeActivatorBean = null;
                    }
                });
            }

            @Override
            public void onFailure(String code, String msg) {
                Log.e(TAG,  msg: "getToken failed:" + msg);
            }
        });
```

Εικόνα 56. Συνάρτηση σύνδεσης συσκευής με την εφαρμογή (2/2)

4.2.4.3 Πίνακας Ελέγχου (Dashboard)

Πατώντας το πλήκτρο *Dashboard* ο χρήστης κατευθύνεται σε μια νέα οθόνη όπου μπορεί να δει εποπτικά την εκτέλεση του αλγορίθμου απόφασης και τις επιμέρους λειτουργίες του (βλ. Εικόνα 57).



Εικόνα 57. Πίνακας ελέγχου (Dashboard)

Στην πρώτη επιλογή *Electricity Prices* ο χρήστης μπορεί να δει άμεσα την τρέχουσα τιμή του ηλεκτρικού ρεύματος και πατώντας το πλήκτρο να δει αναλυτικά περισσότερες λεπτομέρειες για την τιμή του ηλεκτρικού ρεύματος την τελευταία ημέρα.

Στη δεύτερη επιλογή *Water Heater Temperature* ο χρήστης μπορεί να δει την τρέχουσα τιμή της θερμοκρασίας στο εσωτερικό του θερμοσίφωνα και πατώντας το πλήκτρο να δει λεπτομέρειες για τις προηγούμενες τιμές της θερμοκρασίας.

Στην τρίτη επιλογή *Consumption Optimization Algorithm* ο χρήστης μπορεί να δει την κατάσταση λειτουργίας του αλγορίθμου η οποία, εκτός απροόπτου, θα βρίσκεται διαρκώς σε λειτουργία και θα αναγράφεται η ένδειξη *running*. Πατώντας το πλήκτρο μπορεί να δει περισσότερες λεπτομέρειες για τη λειτουργία του.

Στην τέταρτη επιλογή *Total Cost Savings* ο χρήστης μπορεί να δει την εξοικονόμηση σε ευρώ που έχει πετύχει μέχρι στιγμής με τη λειτουργία της εφαρμογής. Η τιμή αυτή ενημερώνεται κάθε φορά που λαμβάνεται μια απόφαση (αν χρειάζεται), ανάλογα με το αν η απόφαση ήταν επιτυχημένη ή όχι. Υπολογίζεται η διαφορά της τρέχουσας τιμής (που επέλεξε ο αλγόριθμος ως πιο ευνοϊκή) με την πραγματική τιμή που προέκυψε στην συνέχεια, προσθέτοντας ή αφαιρώντας αντίστοιχα στο μέχρι τώρα σύνολο. Τελικά, αποθηκεύεται στο αρχείο *system_info.xml* από όπου και διαβάζεται και εμφανίζεται στην οθόνη.

Ο κώδικας που υλοποιεί τις παραπάνω λειτουργίες βρίσκεται στο αρχείο *DashboardViewActivity.java.* Στη συνέχεια, παρατίθενται κάποια κρίσιμα κομμάτια από τον κώδικα αυτού του αρχείου.

Εικόνα 58. Συνάρτηση προγραμματισμού εκτέλεσης του αλγορίθμου

Κατά την πρώτη είσοδο του χρήστη στην οθόνη *Dashboard* εκτελείται η συνάρτηση *scheduleDecision* η οποία προγραμματίζει τον αλγόριθμο απόφασης να εκτελείται κάθε μία ώρα (βλ. Εικόνα 58). Για τον σκοπό αυτό χρησιμοποιούμε τη βιβλιοθήκη της Java *JobScheduler*. Για να πραγματοποιηθεί ο προγραμματισμός του αλγορίθμου απαιτείται επίσης να υπάρχει κάποιο είδος σύνδεσης στο δίκτυο, αφού για την εκτέλεσή του απαιτείται η ανάγνωση της τρέχουσας τιμής του ηλεκτρικού ρεύματος από το διαδίκτυο. Η παραπάνω συνάρτηση ορίζει κάθε μία ώρα την εκτέλεση του *JobService* με όνομα *Decision.class* (βλ. Εικόνα 59).

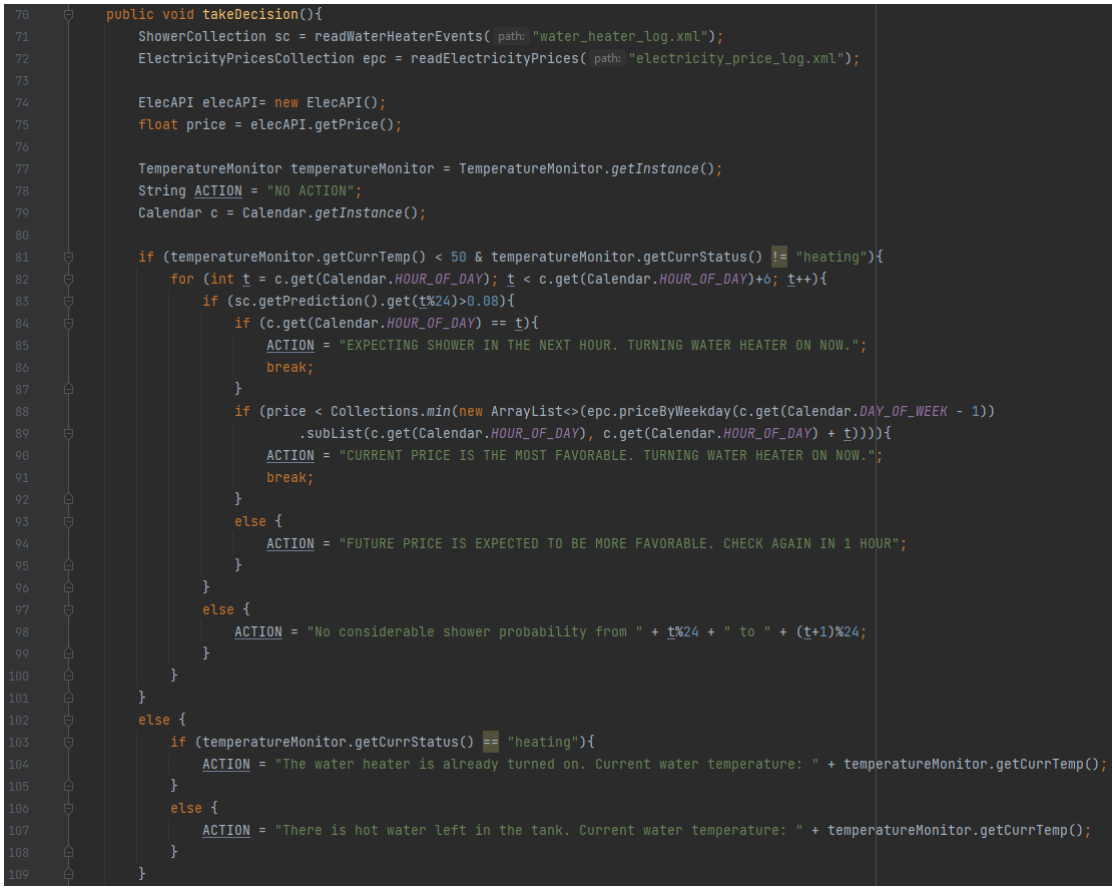
Η συνάρτηση *onStartJob()* καλείται κάθε φορά που εκτελείται η συγκεκριμένη προγραμματισμένη διεργασία, η οποία με τη σειρά της καλεί τη συνάρτηση

takeDecision Η συνάρτηση *onStopJob()* καλείται κάθε φορά που διακόπτεται η προγραμματισμένη διεργασία για οποιονδήποτε λόγο.

```java
Decision.java
7 usages
public class Decision extends JobService {
    2 usages
    private static final String TAG = "DECISION";

    no usages
    @Override
    public boolean onStartJob(final JobParameters params) {

        HandlerThread handlerThread = new HandlerThread( name: "NewThread");
        handlerThread.start();

        Handler handler = new Handler(handlerThread.getLooper());
        handler.post(() -> {
            Log.e(TAG, msg: "Running");
            takeDecision();
            jobFinished(params, wantsReschedule: true);
        });

        return true;
    }

    no usages
    @Override
    public boolean onStopJob(final JobParameters params) {
        Log.d(TAG, msg: "onStopJob() was called");
        return true;
    }
```

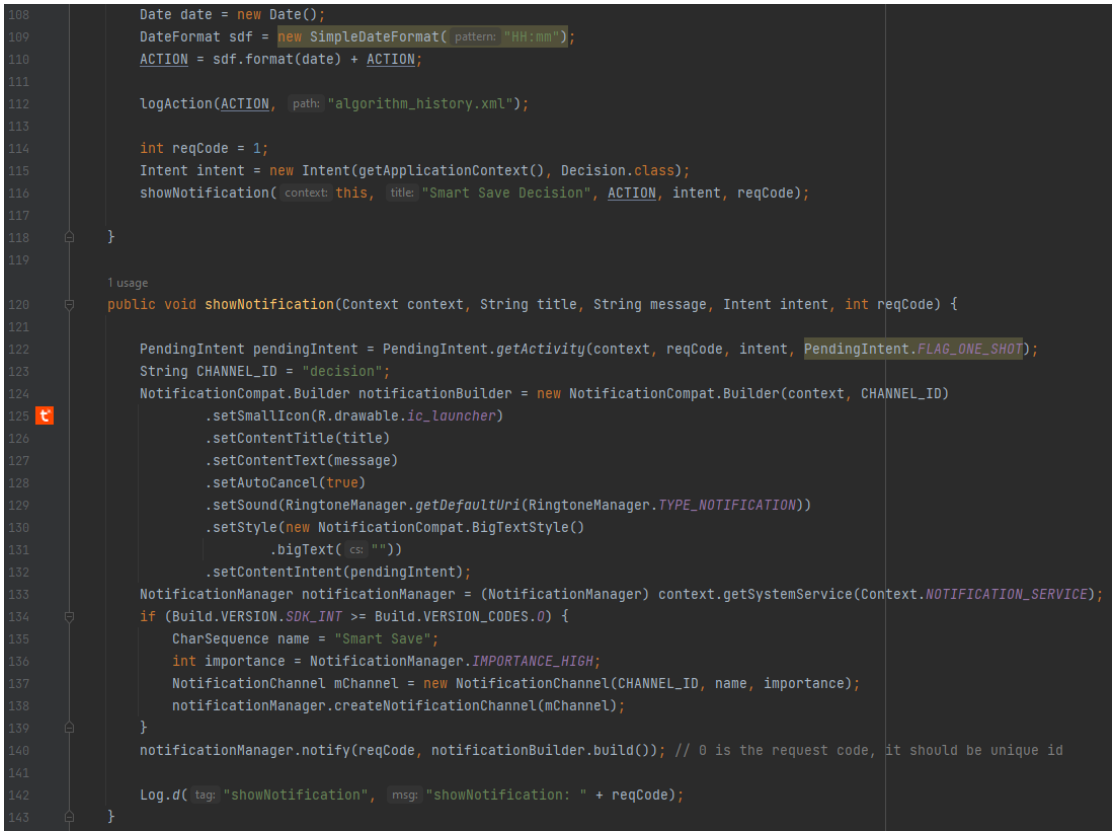
Εικόνα 59. Κλάση Decision (JobService)

Η συνάρτηση *takeDecision* (βλ. Εικόνα 60) η οποία και αποτελεί την υλοποίηση του αλγορίθμου που δημιουργήθηκε σε αυτή τη διπλωματική εργασία, ακολουθεί το διάγραμμα ροής που περιγράφηκε στην ενότητα 2.5 (βλ. Εικόνα 12) και παρατίθεται παρακάτω.



```
public void takeDecision(){
    ShowerCollection sc = readWaterHeaterEvents( path: "water_heater_log.xml");
    ElectricityPricesCollection epc = readElectricityPrices( path: "electricity_price_log.xml");

    ElecAPI elecAPI= new ElecAPI();
    float price = elecAPI.getPrice();

    TemperatureMonitor temperatureMonitor = TemperatureMonitor.getInstance();
    String ACTION = "NO ACTION";
    Calendar c = Calendar.getInstance();

    if (temperatureMonitor.getCurrTemp() < 50 & temperatureMonitor.getCurrStatus() != "heating"){
        for (int t = c.get(Calendar.HOUR_OF_DAY); t < c.get(Calendar.HOUR_OF_DAY)+6; t++){
            if (sc.getPrediction().get(t%24)>0.08){
                if (c.get(Calendar.HOUR_OF_DAY) == t){
                    ACTION = "EXPECTING SHOWER IN THE NEXT HOUR. TURNING WATER HEATER ON NOW.";
                    break;
                }
                if (price < Collections.min(new ArrayList<>(epc.priceByWeekday(c.get(Calendar.DAY_OF_WEEK - 1))
                        .subList(c.get(Calendar.HOUR_OF_DAY), c.get(Calendar.HOUR_OF_DAY) + t)))){
                    ACTION = "CURRENT PRICE IS THE MOST FAVORABLE. TURNING WATER HEATER ON NOW.";
                    break;
                }
                else {
                    ACTION = "FUTURE PRICE IS EXPECTED TO BE MORE FAVORABLE. CHECK AGAIN IN 1 HOUR";
                }
            }
            else {
                ACTION = "No considerable shower probability from " + t%24 + " to " + (t+1)%24;
            }
        }
    }
    else {
        if (temperatureMonitor.getCurrStatus() == "heating"){
            ACTION = "The water heater is already turned on. Current water temperature: " + temperatureMonitor.getCurrTemp();
        }
        else {
            ACTION = "There is hot water left in the tank. Current water temperature: " + temperatureMonitor.getCurrTemp();
        }
    }
```

Εικόνα 60. Υλοποίηση αλγορίθμου απόφασης (1/3)



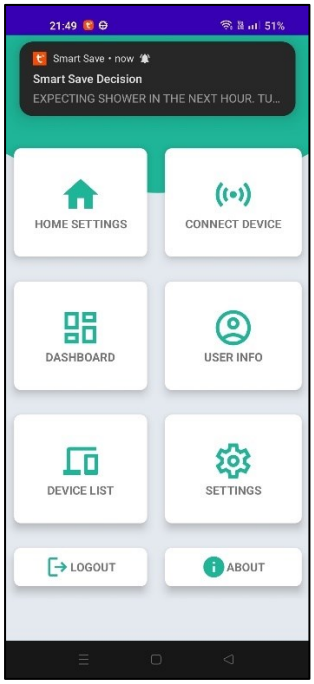
```
    Date date = new Date();
    DateFormat sdf = new SimpleDateFormat( pattern: "HH:mm");
    ACTION = sdf.format(date) + ACTION;

    logAction(ACTION, path: "algorithm_history.xml");

    int reqCode = 1;
    Intent intent = new Intent(getApplicationContext(), Decision.class);
    showNotification( context: this, title: "Smart Save Decision", ACTION, intent, reqCode);

}

1 usage
public void showNotification(Context context, String title, String message, Intent intent, int reqCode) {

    PendingIntent pendingIntent = PendingIntent.getActivity(context, reqCode, intent, PendingIntent.FLAG_ONE_SHOT);
    String CHANNEL_ID = "decision";
    NotificationCompat.Builder notificationBuilder = new NotificationCompat.Builder(context, CHANNEL_ID)
            .setSmallIcon(R.drawable.ic_launcher)
            .setContentTitle(title)
            .setContentText(message)
            .setAutoCancel(true)
            .setSound(RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION))
            .setStyle(new NotificationCompat.BigTextStyle()
                    .bigText( cs: ""))
            .setContentIntent(pendingIntent);
    NotificationManager notificationManager = (NotificationManager) context.getSystemService(Context.NOTIFICATION_SERVICE);
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        CharSequence name = "Smart Save";
        int importance = NotificationManager.IMPORTANCE_HIGH;
        NotificationChannel mChannel = new NotificationChannel(CHANNEL_ID, name, importance);
        notificationManager.createNotificationChannel(mChannel);
    }
    notificationManager.notify(reqCode, notificationBuilder.build()); // 0 is the request code, it should be unique id

    Log.d( tag: "showNotification", msg: "showNotification: " + reqCode);
}
```

Εικόνα 61. Υλοποίηση αλγορίθμου απόφασης (2/3)

```
145     private void logAction(String action, String path) {
146
147         try {
148             InputStream is = getAssets().open(path);
149
150             DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
151             DocumentBuilder builder = factory.newDocumentBuilder();
152
153             Document doc = builder.parse(is);
154
155             Element decisions = doc.getDocumentElement();
156             ((Element)decisions).setAttribute( s: "ACTION", action);
157
158         } catch (SAXException | ParserConfigurationException | IOException e1) {
159             e1.printStackTrace();
160         }
161
162     }
```

Εικόνα 62. Υλοποίηση αλγορίθμου απόφασης (3/3)

Χρησιμοποιούνται επίσης οι βοηθητικές συναρτήσεις *readWaterHeaterEvents* και *readElectricityPrices* που διαβάζουν τις καταγραφές χρήσης του θερμοσίφωνα και τις παρελθοντικές τιμές του ηλεκτρικού ρεύματος από τα αντίστοιχα αρχεία και δημιουργούν τα αντικείμενα τύπου *ShowerCollection* και *ElectricityPricesCollection* αντίστοιχα. Επιπλέον, δημιουργήθηκε η συνάρτηση *getPrediction*, η οποία επιστρέφει μία λίστα με τις προβλέψεις του αλγορίθμου για την πιθανότητα χρήσης του θερμοσίφωνα σε κάθε παράθυρο μίας ώρας και η συνάρτηση *priceByWeekday* που επιστρέφει την εκτίμηση της τιμής του ηλεκτρικού ρεύματος με βάση τον μέσο όρο των παρελθοντικών τιμών για την ημέρα αυτή. Οι υλοποιήσεις των παραπάνω βοηθητικών συναρτήσεων βρίσκονται στο *ΠΑΡΑΡΤΗΜΑ – Βοηθητικές συναρτήσεις και δομές για την εφαρμογή*.



Εικόνα 63. Ειδοποίηση στον χρήστη

Τέλος, η απόφαση καταγράφεται στο αρχείο *algorithm_history.xml* με τη συνάρτηση *logAction* και εμφανίζεται ειδοποίηση (βλ. Εικόνα 63) η οποία ενημερώνει τον χρήστη για την απόφαση που λήφθηκε. Ένα παράδειγμα του αρχείου *algorithm_history.xml* φαίνεται στην Εικόνα 126 (βλ. ΠΑΡΑΡΤΗΜΑ Ζ).

Όμοια εκτελείται η συνάρτηση *scheduleTemperatureUpdate* (βλ. Εικόνα 64) η οποία προγραμματίζει τη διαδικασία ενημέρωσης της θερμοκρασίας να εκτελείται κάθε πέντε λεπτά. Η συνάρτηση αυτή ορίζει κάθε πέντε λεπτά την εκτέλεση του JobService με όνομα TemperatureUpdate.class.

```
188     private void scheduleTemperatureUpdate(){
189         final JobScheduler jobScheduler = (JobScheduler) getSystemService(
190                 Context.JOB_SCHEDULER_SERVICE);
191
192         // The JobService that we want to run
193         final ComponentName name = new ComponentName( pkg: this, TemperatureUpdate.class);
194
195         // Schedule the job
196         final int result = jobScheduler.schedule(getJob2Info( id: 124,  minutes: 5, name));
197
198         // If successfully scheduled, log this thing
199         if (result == JobScheduler.RESULT_SUCCESS) {
200             Log.d( tag: "TEMP UPDATE",  msg: "Scheduled job successfully!");
201         }
202     }
203
        1 usage
204     private JobInfo getJob2Info(final int id, final long minutes, final ComponentName name) {
205         final long interval = TimeUnit.MINUTES.toMillis(minutes); // run every 5 minutes
206         final boolean isPersistent = true; // persist through boot
207         final int networkType = JobInfo.NETWORK_TYPE_ANY; // Requires some sort of connectivity
208
209         final JobInfo jobInfo;
210
211         if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) {
212             jobInfo = new JobInfo.Builder(id, name)
213                     .setMinimumLatency(interval)
214                     .setRequiredNetworkType(networkType)
215                     .setPersisted(isPersistent)
216                     .build();
217         } else {
218             jobInfo = new JobInfo.Builder(id, name)
219                     .setPeriodic(interval)
220                     .setRequiredNetworkType(networkType)
221                     .setPersisted(isPersistent)
222                     .build();
223         }
224
225         return jobInfo;
226     }
```

Εικόνα 64. Συνάρτηση scheduleTemperatureUpdate

Όμοια με προηγουμένως, υπάρχουν οι συναρτήσεις *onStartJob()* και *onStopJob()* και η εκτελέσιμη συνάρτηση *updateTemp* (βλ. Εικόνα 65*)* η οποία διαβάζει τη νέα τιμή θερμοκρασίας από τον μετρητή και ενημερώνει το αντικείμενο *TemperatureMonitor*. Καταγράφει την τιμή στο αρχείο *temp_log.xml* με τη συνάρτηση *logTemperature* και αν αντιληφθεί μεταβολή στην κατάσταση του θερμοσίφωνα, ότι δηλαδή τέθηκε σε

χρήση, το καταγράφει στο αρχείο *water_heater_log.xml* με τη συνάρτηση *logWaterHeater* (βλ. Εικόνα 66).

```
54    public void updateTemp(){
55        DeviceBean deviceBean = ThingHomeSdk.getDataInstance().getDeviceBean(deviceId);
56
57        TemperatureMonitor temperatureMonitor = null;
58        temperatureMonitor.updateTemperature(Float.valueOf(deviceBean.getDps().get("1").toString())/10.0f);
59
60        logTemperature(temperatureMonitor.getCurrTemp(), path: "temp_log.xml");
61        if (Objects.equals(temperatureMonitor.getCurrStatus(), b: "heating") &&
62                !Objects.equals(temperatureMonitor.getLastStatus(), b: "heating")){
63            logWaterHeater( path: "water_heater_log.xml");
64        }
65    }
66
```

Εικόνα 65. Συνάρτηση updateTemp

```
67    public void logTemperature(Float temp, String path){
68        try {
69            InputStream is = getAssets().open(path);
70
71            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
72            DocumentBuilder builder = factory.newDocumentBuilder();
73
74            Document doc = builder.parse(is);
75
76            Element decisions = doc.getDocumentElement();
77            ((Element)decisions).setAttribute( s: "TEMP", String.valueOf(temp));
78
79        } catch (SAXException | ParserConfigurationException | IOException e1) {
80            e1.printStackTrace();
81        }
82    }
83
      1 usage
84    public void logWaterHeater(String path){
85        DateFormat format = new SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm:ss", Locale.ENGLISH);
86        Calendar c = Calendar.getInstance();
87        try {
88            InputStream is = getAssets().open(path);
89
90            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
91            DocumentBuilder builder = factory.newDocumentBuilder();
92
93            Document doc = builder.parse(is);
94
95            Element decisions = doc.getDocumentElement();
96            ((Element)decisions).setAttribute( s: "EVENT", format.format(c));
97
98        } catch (SAXException | ParserConfigurationException | IOException e1) {
99            e1.printStackTrace();
100       }
101   }
```

Εικόνα 66. Συναρτήσεις logTemperature και logWaterHeater

4.2.4.4 Electricity Prices

Στην οθόνη *Electricity Prices* (βλ. Εικόνα 67) εμφανίζεται στο πάνω μέρος ένα διάγραμμα της εξέλιξης της τιμής του ηλεκτρικού ρεύματος το τελευταίο εικοσιτετράωρο. Στο κάτω μέρος της οθόνης μπορεί ο χρήστης να δει την τρέχουσα τιμή του ηλεκτρικού ρεύματος, κάτω από την ένδειξη *Current Price* καθώς και την εκτιμώμενη μελλοντική τιμή την επόμενη ώρα, κάτω από την ένδειξη *Est. Future Price*.

Εικόνα 67. Εξέλιξη τιμής ηλεκτρικού ρεύματος

Για την υλοποίηση της παραπάνω λειτουργίας, αρχικά βεβαιωνόμαστε ότι έχουν γίνει τα απαραίτητα imports της βιβλιοθήκης XYPlot, με την οποία θα πραγματοποιηθεί η απεικόνιση του διαγράμματος (βλ. Εικόνα 68).

```
17  import com.androidplot.xy.CatmullRomInterpolator;
18  import com.androidplot.xy.LineAndPointFormatter;
19  import com.androidplot.xy.SimpleXYSeries;
20  import com.androidplot.xy.XYGraphWidget;
21  import com.androidplot.xy.XYPlot;
22  import com.androidplot.xy.XYSeries;
```

Εικόνα 68. Imports για απεικόνιση διαγράμματος

Στη συνέχεια ελέγχεται αν έχει χορηγηθεί από τον χρήστη η άδεια πρόσβασης σε εξωτερικά αρχεία και στην περίπτωση που δεν έχει εμφανίζεται το μήνυμα αίτησης της εν λόγω άδειας. Η άδεια αυτή είναι απαραίτητη καθώς για να εμφανιστεί το διάγραμμα τιμής του ηλεκτρικού ρεύματος το τελευταίο εικοσιτετράωρο, πρέπει να γίνει ανάγνωση των τιμών από το αρχείο *electricity_price_log.xml*. Ακολούθως, γίνεται ανάγνωση των τιμών από το αρχείο αυτό. Ένα παράδειγμα τέτοιου αρχείου φαίνεται στην Εικόνα 123 (βλ. ΠΑΡΑΡΤΗΜΑ Ε) και με τις τιμές αυτές δημιουργείται το αντικείμενο *epc* τύπου *ElectricityPricesCollection*, μία κλάση που έχει δημιουργηθεί για αυτόν τον σκοπό (βλ. ΠΑΡΑΡΤΗΜΑ Γ). Μετά, εμφανίζονται στο κάτω μέρος της

οθόνης οι δύο τιμές: η τρέχουσα, η οποία διαβάζεται από το *ElecAPI* (βλ. ΠΑΡΑΡΤΗΜΑ ΣΤ) και η εκτιμώμενη, η οποία εξάγεται από τη συνάρτηση *priceByDayHour*. Τέλος, γίνεται το διάγραμμα με βάση τις τιμές του τελευταίου εικοσιτετράωρου όπως αυτές έχουν καταγραφεί στο αρχείο καταγραφών.

Ο κώδικας που υλοποιεί τα παραπάνω βρίσκεται στο αρχείο *ElectricityViewActivity.java* (βλ. Εικόνα 69).

ElectricityViewActivity.java

```
53      @Override
54      protected void onCreate(Bundle savedInstanceState) {
55          super.onCreate(savedInstanceState);
56          setContentView(R.layout.activity_electricity_view);
57
58          int grant = ContextCompat.checkSelfPermission( context: this, Manifest.permission.READ_EXTERNAL_STORAGE);
59          if (grant != PackageManager.PERMISSION_GRANTED) {
60              String[] permission_list = new String[1];
61              permission_list[0] = Manifest.permission.READ_EXTERNAL_STORAGE;
62              ActivityCompat.requestPermissions( activity: this, permission_list,  requestCode: 1);
63          }
64
65          currentPrice = findViewById(R.id.currentPrice);
66          futurePrice = findViewById((R.id.futurePrice));
67
68          //LOAD XML
69          ElectricityPricesCollection epc = readElectricityPrices( path: "electricity_price_log.xml");
70          Calendar calendar = Calendar.getInstance();
71          Float pred;
72          if (calendar.get(Calendar.HOUR_OF_DAY)<23){
73              pred = epc.priceByDayHour( weekday: calendar.get(Calendar.DAY_OF_WEEK)-1,  hour: calendar.get(Calendar.HOUR_OF_DAY)+1);
74          }else{
75              pred = epc.priceByDayHour(calendar.get(Calendar.DAY_OF_WEEK),  hour: 0);
76          }
77
78          //SHOW VALUES
79          ElecAPI elecAPI= new ElecAPI();
80          currentPrice.setText(String.valueOf(elecAPI.getPrice()));
81          futurePrice.setText(String.valueOf(pred));
82
83          //PLOT DIAGRAM
84          ElectricityPrices last_ep = epc.getList().get(epc.getList().size()-1);
85          Number[] series1Numbers = last_ep.getPrices().toArray(new Float[0]);
86          plotElecPrices(series1Numbers, last_ep.getDate().toString());
87      }
```

Εικόνα 69. Δημιουργία οθόνης Electricity Prices

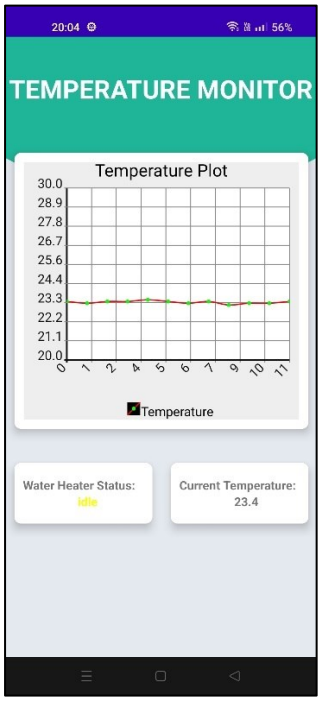
Για την απεικόνιση του διαγράμματος χρησιμοποιήσαμε την βιβλιοθήκη XYPlot και η υλοποίηση φαίνεται στην εικόνα 70. Σημειώνεται ότι ορίζουμε στον άξονα X όρια από το 0 έως το 23, που αντιπροσωπεύουν τις ώρες της ημέρας και στον άξονα Y από το 0 έως το 0.2, αφού η τιμή της κιλοβατώρας κυμαίνεται μεταξύ αυτών των τιμών.

```
protected void plotElecPrices(Number[] series, String title){
    plot = findViewById(R.id.plot);

    //Create a arrays of y-value to plot:
    final Number[] domainLabels = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23};


    // Turn the above arrays into XYSeries
    XYSeries series1 = new SimpleXYSeries(Arrays.asList(series), SimpleXYSeries.ArrayFormat.Y_VALS_ONLY, title);

    LineAndPointFormatter series1Format = new LineAndPointFormatter(Color.RED, Color.GREEN, fillColor: null, plf: null);

    series1Format.setInterpolationParams(new CatmullRomInterpolator.Params( pointPerSegment: 10,
            CatmullRomInterpolator.Type.Centripetal));

    plot.addSeries(series1, series1Format);

    plot.setDomainBoundaries( lowerBoundary: 0, upperBoundary: 23, BoundaryMode.FIXED);
    plot.setRangeBoundaries( lowerBoundary: 0.00, upperBoundary: 0.20, BoundaryMode.FIXED);
    plot.setDomainStep(StepMode.INCREMENT_BY_VAL, value: 2);
    plot.setRangeStep(StepMode.SUBDIVIDE, value: 5);

    //PanZoom.attach(plot);
}
```

Εικόνα 70. Απεικόνιση διαγράμματος τιμών ηλεκτρικού ρεύματος

4.2.4.5 Water Heater Temperature

Στην οθόνη *Water Heater Temperature* (βλ. Εικόνα 71) εμφανίζεται στο πάνω μέρος ένα διάγραμμα της εξέλιξης της θερμοκρασίας στο εσωτερικό του θερμοσίφωνα την τελευταία ώρα. Στο κάτω μέρος της οθόνης μπορεί ο χρήστης να δει την κατάσταση του θερμοσίφωνα (όπως ορίστηκε στην ενότητα 2.4.3), κάτω από την ένδειξη *Water Heater Status*, καθώς και την τρέχουσα θερμοκρασία στο εσωτερικό του θερμοσίφωνα, κάτω από την ένδειξη *Current Temperature*.



Εικόνα 71. Οθόνη κατάστασης θερμοσίφωνα

Ο κώδικας που υλοποιεί τις παραπάνω λειτουργίες βρίσκεται στο αρχείο *TemperatureViewActivity.java* (βλ. Εικόνα 72).

Βεβαιωνόμαστε και πάλι ότι έχουν γίνει τα απαραίτητα imports της βιβλιοθήκης XYPlot, με την οποία θα πραγματοποιηθεί η απεικόνιση του διαγράμματος.

```
45      @Override
46      protected void onCreate(Bundle savedInstanceState) {
47          super.onCreate(savedInstanceState);
48          setContentView(R.layout.activity_temperature_view);
49
50          String deviceId = getIntent().getStringExtra( name: "deviceId");
51          DeviceBean deviceBean = ThingHomeSdk.getDataInstance().getDeviceBean(deviceId);
52
53          //LOAD TEMPERATURE MONITOR
54          TemperatureMonitor temperatureMonitor = TemperatureMonitor.getInstance();
55          TextView status = findViewById(R.id.status);
56          status.setText(temperatureMonitor.getCurrStatus());
57          switch (temperatureMonitor.getCurrStatus()) {
58              case "idle":
59                  status.setTextColor(Color.YELLOW);
60                  break;
61              case "heating":
62                  status.setTextColor(Color.RED);
63                  break;
64              case "use":
65                  status.setTextColor(Color.BLUE);
66                  break;
67              case "cooling":
68                  status.setTextColor(Color.WHITE);
69                  break;
70          }
71          TextView currentTemp = findViewById(R.id.currentTemp);
72          Float tmp = Float.valueOf(deviceBean.getDps().get("1").toString())/10.0f;
73          currentTemp.setText(String.valueOf(tmp));
74
75          //PLOT DIAGRAM
76          Number[] series1Numbers = temperatureMonitor.getTemperatures().toArray(new Float[0]);
77          plotTemps(series1Numbers);
78      }
```

Εικόνα 72. Δημιουργία οθόνης Temperature Monitor

Αρχικά, ορίζεται η επαφή με τον μετρητή θερμοκρασίας και υγρασίας με βάση το αναγνωριστικό του *deviceID* και δημιουργείται το αντικείμενο *deviceBean*, τύπου *DeviceBean*, ένας τύπος αντικειμένου υλοποιημένο στο Tuya SDK που αντιστοιχεί σε μία συνδεδεμένη συσκευή. Στη συνέχεια, εμφανίζονται στο κάτω μέρος της οθόνης η κατάσταση του μετρητή με το κατάλληλο αναγνωριστικό χρώμα (κίτρινο για την κατάσταση "*idle*", κόκκινο για την "*heating*", μπλε για την "*use*" και λευκό για την "*cooling*") και η τρέχουσα τιμή της θερμοκρασίας στο εσωτερικό του θερμοσίφωνα με την εντολή στην γραμμή 79 (διαβάζοντας το datapoint των δεδομένων της συσκευής με αριθμό 1). Τέλος, γίνεται το διάγραμμα με βάση τις τιμές θερμοκρασίας της τελευταίας ώρας που έχουν αποθηκευτεί στο αντικείμενο *TemperatureMonitor* που δημιουργήθηκε κατά την εκκίνηση της εφαρμογής (βλ. ΠΑΡΑΡΤΗΜΑ Α).

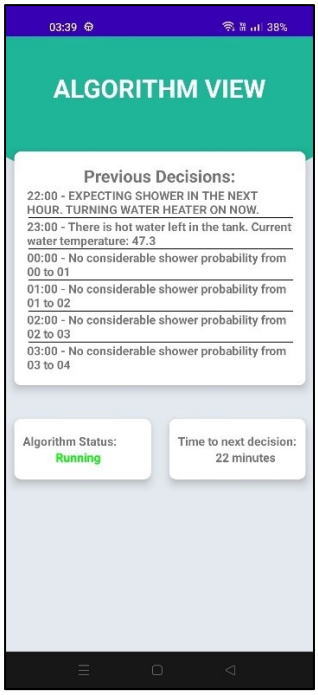
Για την απεικόνιση του διαγράμματος χρησιμοποιήσαμε ξανά την βιβλιοθήκη XYPlot και η υλοποίηση φαίνεται στην Εικόνα 73. Σημειώνεται ότι ορίζουμε στον άξονα Y όρια από το 20 έως το 60, που αντιπροσωπεύουν τις τιμές θερμοκρασίας που αναπτύσσονται συνήθως στο εσωτερικό ενός θερμοσίφωνα.

```
protected void plotTemps(Number[] series){
    plot = findViewById(R.id.plot);

    //Create a arrays of y-value to plot:
    final Number[] domainLabels = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};


    // Turn the above arrays into XYSeries
    XYSeries series1 = new SimpleXYSeries(Arrays.asList(series),
            SimpleXYSeries.ArrayFormat.Y_VALS_ONLY, title: "Temperature");

    LineAndPointFormatter series1Format = new LineAndPointFormatter(Color.RED, Color.GREEN, fillColor: null, plf: null);

    series1Format.setInterpolationParams(new CatmullRomInterpolator.Params( pointPerSegment: 10,
            CatmullRomInterpolator.Type.Centripetal));

    plot.addSeries(series1, series1Format);

    plot.setRangeBoundaries( lowerBoundary: 20, upperBoundary: 60, BoundaryMode.FIXED);

    plot.getGraph().getLineLabelStyle(XYGraphWidget.Edge.BOTTOM).setFormat(new Format() {
        @Override
        public StringBuffer format(Object obj, StringBuffer toAppendTo, FieldPosition pos) {
            int i = Math.round(((Number) obj).floatValue());
            return toAppendTo.append(domainLabels[i]);
        }

        @Override
        public Object parseObject(String source, ParsePosition pos) { return null; }
    });
}
```

Εικόνα 73. Απεικόνιση διαγράμματος θερμοκρασίας θερμοσίφωνα

4.2.4.6 Consumption Optimization Algorithm

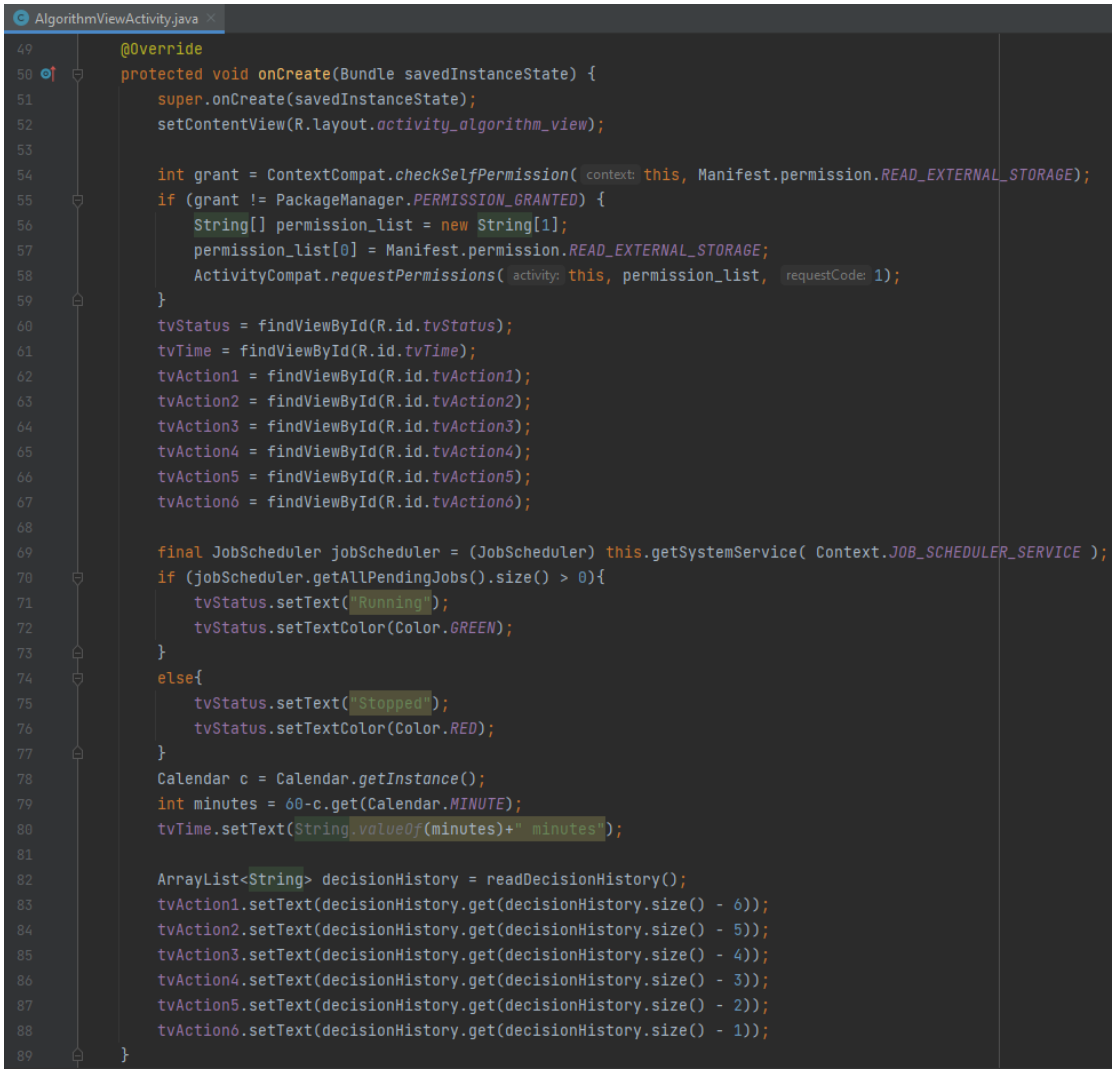
Στην οθόνη *Consumption Optimization Algorithm* (βλ. Εικόνα 74) εμφανίζεται στο πάνω μέρος μια λίστα με τις τελευταίες έξι ενέργειες του αλγορίθμου. Στο κάτω μέρος της οθόνης μπορεί ο χρήστης να δει την κατάσταση λειτουργίας του αλγορίθμου κάτω από την ένδειξη *Algorithm Status* (*Running* αν ο αλγόριθμος βρίσκεται σε εξέλιξη και *Stopped* αν για κάποιο λόγο η λειτουργία έχει σταματήσει), καθώς και τα λεπτά που απομένουν μέχρι την επόμενη εκτέλεση του αλγορίθμου, κάτω από την ένδειξη *Time to next Decision*. Υπενθυμίζεται ότι ο αλγόριθμος εκτελείται κάθε μία ώρα (ξεκινώντας από τις 00:00).



Εικόνα 74. Κατάσταση αλγορίθμου

Ο κώδικας που υλοποιεί τις παραπάνω λειτουργίες βρίσκεται στο αρχείο *AlgorithmViewActivity.java*. Παρακάτω παρατίθεται ο κώδικας αυτού του αρχείου (βλ. Εικόνα 75).

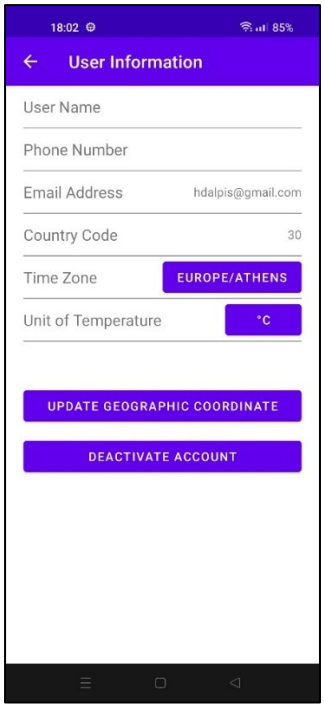
Αρχικά, ελέγχεται αν έχει χορηγηθεί από τον χρήστη η άδεια πρόσβασης σε εξωτερικά αρχεία και στην περίπτωση που δεν έχει εμφανίζεται το μήνυμα αίτησης της εν λόγω άδειας. Η άδεια αυτή είναι απαραίτητη καθώς για να εμφανιστεί η λίστα με τις τελευταίες εκτελέσεις του αλγορίθμου, πρέπει να γίνει ανάγνωση των καταγραφών από το αρχείο *algorithm_history.xml* με την βοήθεια της συνάρτησης *readDecisionHistory*. Στη συνέχεια, γίνεται έλεγχος στην γραμμή 69 αν ο προγραμματισμός του αλγορίθμου βρίσκεται σε εξέλιξη και εμφανίζεται το ανάλογο μήνυμα με το αντίστοιχο χρώμα, πράσινο για την ένδειξη *Running* αν ο προγραμματισμός του αλγορίθμου βρίσκεται σε εξέλιξη και κόκκινο για την ένδειξη *Stopped* αν ο προγραμματισμός του αλγορίθμου έχει σταματήσει. Υπολογίζονται επίσης τα λεπτά που απομένουν μέχρι την επόμενη εκτέλεση, καθώς και συμπληρώνεται η λίστα με το ιστορικό των τελευταίων έξι εκτελέσεων όπως διαβάζονται από το αρχείο *algorithm_history.xml*. Ο κώδικας που πραγματοποιεί την ανάγνωση του αρχείου καθώς και ένα δείγμα τέτοιου αρχείου παρατίθενται στην Εικόνα 126 (βλ. ΠΑΡΑΡΤΗΜΑ Ζ).



Εικόνα 75. Δημιουργία οθόνης Algorithm View

4.2.4.7 Πληροφορίες Χρήστη (User Info)

Πατώντας το πλήκτρο *User Info* στην αρχική οθόνη ο χρήστης κατευθύνεται σε μια νέα οθόνη όπου μπορεί να δει και να τροποποιήσει τα στοιχεία του λογαριασμού που έχει δημιουργήσει στην Tuya (βλ. Εικόνα 76).

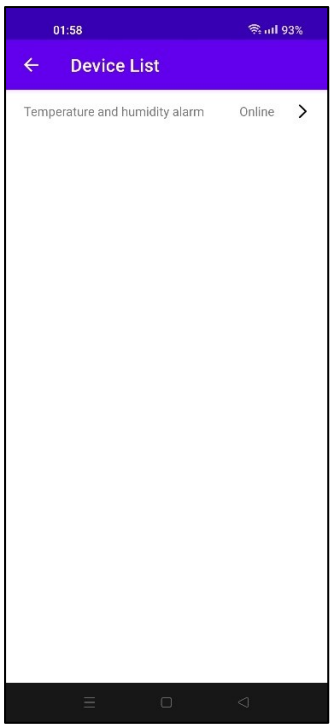


Εικόνα 76. Πληροφορίες χρήστη

Ο κώδικας που υλοποιεί την παραπάνω οθόνη βρίσκεται στο αρχείο *UserInfoActivity.java*. Για την υλοποίηση της τροποποιήθηκε η ήδη υπάρχουσα υλοποίηση για την προβολή του λογαριασμού του χρήστη από το *Tuya Android Smart Life App SDK Sample.*

4.2.4.8 Λίστα Συσκευών (Device List)

Πατώντας το πλήκτρο *Device List* στην αρχική οθόνη ο χρήστης κατευθύνεται σε μια νέα οθόνη όπου μπορεί να δει όλες τις συνδεδεμένες συσκευές, καθώς και την κατάσταση στην οποία βρίσκεται η κάθε συσκευή, εντός ή εκτός σύνδεσης (online ή offline) (βλ. Εικόνα 77).



Εικόνα 77. Λίστα συσκευών

Ο κώδικας που υλοποιεί την παραπάνω λειτουργία βρίσκεται στα αρχεία *DeviceMgtListActivity.java* και *DeviceMgtAdapter.java*. Παρατίθενται αποσπάσματα του κώδικα των αρχείων *DeviceMgtListActivity.java* και *DeviceMgtAdapter.java* για την προβολή συνδεδεμένων συσκευών (βλ. Εικόνες 78, 79).

```
DeviceMgtListActivity.java
72          RecyclerView rvList = findViewById(R.id.rvList);
73          rvList.setLayoutManager(new LinearLayoutManager( context: this, LinearLayoutManager.VERTICAL,  reverseLayout: false));
74
75          adapter = new DeviceMgtAdapter();
76          rvList.setAdapter(adapter);
77      }
```

Εικόνα 78. Τμήμα του αρχείου DeviceMgtListActivity.java

Στο αρχείο *DeviceMgtListActivity.java* δημιουργείται μία κυλιόμενη προβολή *RecyclerView* και κατόπιν δημιουργείται ένα στιγμιότυπο αντικειμένου τύπου *DeviceMgtAdapter*, μια κλάση η οποία εντοπίζει τις συνδεδεμένες συσκευές και τις προσθέτει στην προβολή αυτήν.

Στη συνέχεια το αντικείμενο αυτό συνδέεται με το *RecyclerView* και κατά τη σύνδεσή του προσθέτει στη λίστα το όνομα της συσκευής και δίπλα την κατάστασή της (status), στοιχεία τα οποία λαμβάνει με τις εντολές *bean.name* και *bean.getIsOnline*. Το αντικείμενο bean είναι ένα αντικείμενο τύπου *DeviceBean*, ένας τύπος αντικειμένου υλοποιημένο στο Tuya SDK που αντιστοιχεί σε μία συνδεδεμένη συσκευή.

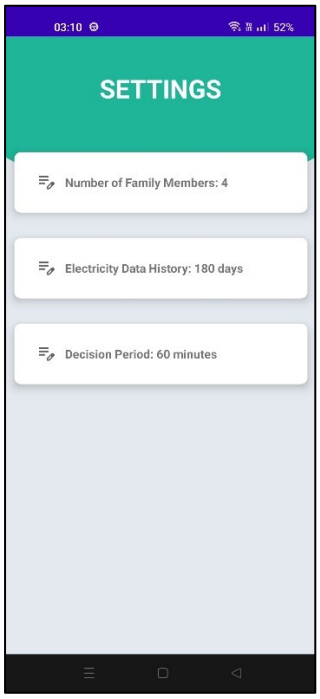
```java
@Override
public void onBindViewHolder(@NonNull DeviceMgtAdapter.ViewHolder holder, int position) {
    DeviceBean bean = (DeviceBean) data.get(position);
    holder.tvDeviceName.setText(bean.name);
    holder.tvStatus.setText(holder.itemView.getContext().getString(bean.getIsOnline() ? "Online" : "Offline"));

}
```

Εικόνα 79. Τμήμα του αρχείου DeviceMgtAdapter.java

Για την υλοποίηση αυτής της λειτουργίας χρησιμοποιήθηκαν τροποποιημένα μέρη της ήδη υπάρχουσας υλοποίησης για την προβολή συνδεδεμένων συσκευών από το *Tuya Android Smart Life App SDK Sample.*

4.2.4.9 Ρυθμίσεις (Settings)

Πατώντας το πλήκτρο *Settings* στην αρχική οθόνη ο χρήστης κατευθύνεται σε μια νέα οθόνη (βλ. Εικόνα 80) όπου έχει πρόσβαση σε κάποιες βασικές ρυθμίσεις για την λειτουργία της εφαρμογής. Οι ρυθμίσεις αυτές περιλαμβάνουν τον αριθμό τον ατόμων τα οποία κατοικούν στο σπίτι, το χρονικό διάστημα για το οποίο ο αλγόριθμος διαβάζει τα ιστορικά δεδομένα των τιμών του ρεύματος (προεπιλεγμένη τιμή είναι οι 6 μήνες) και το χρονικό διάστημα ανά το οποίο εκτελείται ο αλγόριθμος απόφασης (προεπιλεγμένη τιμή είναι η 1 ώρα).



Εικόνα 80. Οθόνη Ρυθμίσεις

4.2.4.10 Λοιπές Λειτουργίες

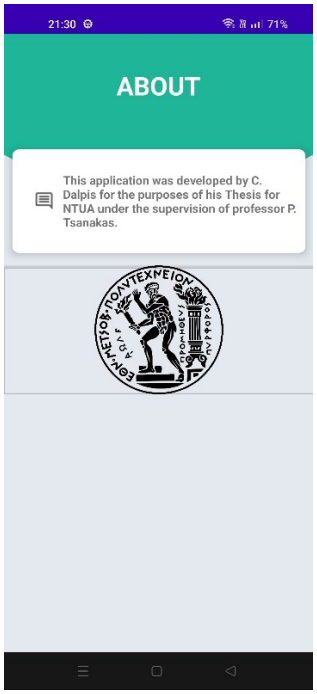
Στο κάτω μέρος της αρχικής οθόνης, υπάρχουν τα πλήκτρα *Logout* και *About*. Με το πλήκτρο *Logout* ο χρήστης αποσυνδέεται από τον λογαριασμό του και επιστρέφει στην οθόνη εγγραφής / σύνδεσης. Εκτελείται το παρακάτω τμήμα κώδικα (βλ. Εικόνα 81)

στο οποίο καθαρίζεται η μνήμη cache από τις μέχρι τώρα ενέργειες στην εφαρμογή και επιστρέφει τον χρήστη στην οθόνη *UserFuncActivity*.

```
99          findViewById(R.id.cvLogout).setOnClickListener(new View.OnClickListener() {
100             @Override
101             public void onClick(View v) {
102                 ThingHomeSdk.getUserInstance().logout(new ILogoutCallback() {
103                     @Override
104                     public void onSuccess() {
105                         // Clear cache
106                         HomeModel.INSTANCE.clear( context: MainSampleListActivity.this);
107
108                         // Navigate to User Func Navigation Page
109                         Intent intent = new Intent( packageContext: MainSampleListActivity.this, UserFuncActivity.class);
110                         intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TOP);
111                         startActivity(intent);
112                     }
113
114                     @Override
115                     public void onError(String errorCode, String errorMsg) {
116                     }
117                 });
118             }
119         });
```

Εικόνα 81. Αποσύνδεση χρήστη από την εφαρμογή

Με το πλήκτρο *About* εμφανίζεται ένα σύντομο μήνυμα στην οθόνη για τη δημιουργία της εφαρμογής (βλ. Εικόνα 82).



Εικόνα 82. Σχετικά με την εφαρμογή

4.3 Web Server Monitor Tool

Στην ενότητα αυτή περιγράφονται αναλυτικά τα βήματα υλοποίησης ενός εργαλείου επίβλεψης της έξυπνης συσκευής που εγκαταστάθηκε καθώς και της κατάστασης λειτουργίας του ηλεκτρικού θερμοσίφωνα στο έξυπνο σπίτι. Το εργαλείο αυτό διαθέτει

περιβάλλον διεπαφής προσβάσιμο τοπικά στον φυλλομετρητή με τη βοήθεια ενός τοπικού διακομιστή (local web server) και υλοποιήθηκε στη γλώσσα προγραμματισμού Python με την βοήθεια του Tuya IoT Python SDK.

4.3.1 Ενσωμάτωση Tuya IoT Python SDK

Αρχικά δημιουργήθηκε το αρχείο *env.py* στο οποίο ορίσαμε κάποιες μεταβλητές απαραίτητες για τη σύνδεση με τον διακομιστή της Tuya και την ανταλλαγή δεδομένων με τον μετρητή θερμοκρασίας και υγρασίας που συνδέσαμε προηγουμένως. Οι μεταβλητές αυτές περιλαμβάνουν το *ACCESS_ID* και *ACCESS_KEY* που αποθηκεύσαμε στην ενότητα 4.1.2, το όνομα χρήστη (*USERNAME*) και τον κωδικό πρόσβασής μας (*PASSWORD*) στην Tuya, το αναγνωριστικό της έξυπνης συσκευής μας (*DEVICE_ID*) και το σημείο σύνδεσης με τον κατάλληλο διακομιστή της Tuya (*ENDPOINT*) (βλ. Εικόνα 83).

env.py

```
1  ACCESS_ID = 'efqxa5j5yq5jhjucpu8y'
2  ACCESS_KEY = '[redacted]'
3  USERNAME = '[redacted]'
4  PASSWORD = '[redacted]'
5  DEVICE_ID = '[redacted]'
6  ENDPOINT = "https://openapi.tuyaeu.com"
```

Εικόνα 83. Ορισμός απαραίτητων μεταβλητών

Στην κεντρική εφαρμογή με όνομα app.py πραγματοποιήθηκε η απαραίτητη δήλωση import όπως φαίνεται στην Εικόνα 84.

```
9  from tuya_iot import TuyaOpenAPI
```

Εικόνα 84. Import TuyaOpenAPI

Πραγματοποιούμε την εκκίνηση του API με τις παρακάτω εντολές (βλ. Εικόνα 85) χρησιμοποιώντας τα στοιχεία που ορίσαμε παραπάνω στο αρχείο *env.py* και με βάση το αποτέλεσμα *success / fail* πραγματοποιούμε την εκκίνηση του αντικειμένου *TemperatureMonitor*, το οποίο διαβάζει και αποθηκεύει τις τιμές θερμοκρασίας και την κατάσταση του ηλεκτρικού θερμοσίφωνα. Αποθηκεύουμε επίσης το αποτέλεσμα *success / fail* για να το εμφανίσουμε κατάλληλα αργότερα.

```
62  # Initialization of tuya openapi
63  openapi = TuyaOpenAPI(ENDPOINT, ACCESS_ID, ACCESS_KEY)
64  action = openapi.connect(USERNAME, PASSWORD, "30", 'smartlife')
65
66  temp_monitor = initTM()
67  if action['success']:
68      temp_monitor = initTM(1)
69      sign = "green"
70  else:
71      temp_monitor = initTM(0)
72      sign = "red"
```

Εικόνα 85. Εκκίνηση API

Η συνάρτηση *initTM* (βλ. Εικόνα 86) αρχικοποιεί το αντικείμενο *TemperatureMonitor* (βλ. Ενότητα 4.3.5) με βάση το αποτέλεσμα της σύνδεσης, αν είναι επιτυχημένη διαβάζει την τρέχουσα τιμή της θερμοκρασίας από τη συσκευή, αν όχι αρχικοποιείται κενό. Η υλοποίηση της συνάρτησης *initTM* βρίσκεται στο αρχείο *controls.py* και παρατίθεται παρακάτω.

```
27  def initTM(temp):
28      if temp == 0:
29          tm = TempMonitor()
30      else:
31          tm = TempMonitor(readTemp())
32      return tm
```

Εικόνα 86. Συνάρτηση initTM

Η συνάρτηση *readTemp* διαβάζει την τρέχουσα τιμή θερμοκρασίας από τον μετρητή θερμοκρασίας και η υλοποίηση της παρατίθεται παρακάτω (βλ. Εικόνα 87).

```
13  def readTemp():
14      openapi = TuyaOpenAPI(ENDPOINT, ACCESS_ID, ACCESS_KEY)
15      action = openapi.connect(USERNAME, PASSWORD, "30", 'smartlife')
16      response = openapi.get("/v1.0/iot-03/devices/{}/status".format(DEVICE_ID))
17      if response['success']:
18          temperature = response['result'][0]['value']/10
19      else:
20          temperature = 0
21      return temperature
```

Εικόνα 87. Συνάρτηση readTemp

Ο τρόπος επικοινωνίας με τη συσκευή πραγματοποιείται μέσω του Cloud, με τη χρήση GET requests, τα οποία επιστρέφουν αρχεία json. Η ζητούμενη τιμή της θερμοκρασίας είναι αυτή που φαίνεται στην γραμμή 18. Ένα παράδειγμα ενός τέτοιου αρχείου json φαίνεται στην Εικόνα 88.

Response


```
{
  "result": [
    {
      "id": "bf79f2dbb7695f8c23ylr5",
      "status": [
        {
          "code": "va_temperature",
          "value": 234
        },
        {
          "code": "va_humidity",
          "value": 654
        },
        {
          "code": "battery_percentage",
          "value": 100
        },
        {
          "code": "charge_state",
          "value": false
        },
        {
          "code": "temp_unit_convert",
          "value": "c"
        },
        {
          "code": "maxtemp_set",
          "value": 500
        },
        {
          "code": "minitemp_set",
          "value": -10
        },
        {
          "code": "maxhum_set"
```


Εικόνα 88. Αρχείο απάντησης json στο GET request

4.3.2 Δημιουργία Web Server με το Flask

Στη συνέχεια δημιουργήθηκε *Web Server* με την χρήση των βιβλιοθηκών *Flask* και *Flask-SocketIO* για την ανταλλαγή πληροφοριών με την ιστοσελίδα σε πραγματικό χρόνο.

Στην κεντρική εφαρμογή με όνομα *app.py* πραγματοποιήθηκε η απαραίτητη δήλωση *import* όπως φαίνεται στην Εικόνα 89.

```python
from flask import Flask, render_template
from flask_socketio import SocketIO, emit
import threading
```

Εικόνα 89. Import Flask και Socket-IO

Δημιουργήθηκε μια διαδρομή (route) για την αρχική σελίδα του περιβάλλοντος διεπαφής (index.html) στην οποία συνδέεται ο χρήστης και ορίστηκαν τα αρχικά δεδομένα που θα σταλούν στη σελίδα αυτή για να προβληθούν (βλ. Εικόνα 90). Τα αρχικά δεδομένα περιλαμβάνουν τα τεχνικά χαρακτηριστικά της συσκευής, το αρχείο ιστορικού action_log που στην αρχή είναι κενό, τον δείκτη sign που έχει τιμές green / red ανάλογα με το αν ήταν επιτυχής ή όχι η σύνδεση με τον διακομιστή της Tuya αντίστοιχα, την τρέχουσα τιμή ηλεκτρικού ρεύματος (βλ. Ενότητα 4.3.5) και το ιστορικό θερμοκρασίας που στην αρχή είναι επίσης κενό.

```python
# FLASK WEB SERVER FUNCTIONALITIES
# Route for the homepage
@app.route('/')
def homepage():
    return render_template('index.html')

# WebSocket event to send the initial action log to the client when connected
@socketio.on('connect')
def handle_connect():
    global action_log
    get_specs(DEVICE_ID)
    emit('initial_actions', {'actions': action_log})
    emit('tuya_status', {'status': sign})
    emit('init_elec_price', {'price': readElecPrice()})
    emit('initial_temperature_data', {'temperatureData': temperature_data})
```

Εικόνα 90. Ορισμός routes για την αρχική σύνδεση στην σελίδα

Τα τεχνικά χαρακτηριστικά της συσκευής λαμβάνονται με τη συνάρτηση get_specs (βλ. Εικόνα 91), όπου με το αναγνωριστικό της συσκευής (DEVICE_ID) στέλνουμε το κατάλληλο GET request και λαμβάνουμε το αποτέλεσμα.

```python
def get_specs(sensor_id):
    openapi = TuyaOpenAPI(ENDPOINT, ACCESS_ID, ACCESS_KEY)
    action = openapi.connect(USERNAME, PASSWORD, "30", 'smartlife')
    response = openapi.get("/v1.1/iot-03/devices/{}".format(sensor_id))
    socketio.emit('device_specs', {'specs': [response['result']['category_name'], response['result']['id'],
                                             response['result']['ip'], response['result']['model'],
                                             response['result']['product_name'], response['result']['sn']]})
```

Εικόνα 91. Συνάρτηση get_specs

Τέλος, μπορούμε να εκκινήσουμε τον Web Server που δημιουργήσαμε με την εντολή που φαίνεται στην Εικόνα 92. Ο Web Server είναι διαθέσιμος στον χρήστη στην τοπική διεύθυνση *http://127.0.0.1:5000/*

```python
if __name__ == '__main__':
    socketio.run(app, host='0.0.0.0', port=5000, debug=True)
```

Εικόνα 92. Εντολή εκκίνησης του Web Server

4.3.3 Ανανέωση Τιμής Θερμοκρασίας

Η τιμή της θερμοκρασίας στο εσωτερικό του θερμοσίφωνα διαβάζεται κάθε 5 λεπτά και δημιουργούμε διάγραμμα στο οποίο ο χρήστης μπορεί να δει ζωντανά την εξέλιξη της θερμοκρασίας.

Κάνουμε την απαραίτητη δήλωση import όπως φαίνεται στην Εικόνα 93.

```python
import time
import sched
```

Εικόνα 93. Import

Δημιουργούμε τον προγραμματιστή (scheduler) ο οποίος εκτελείται κάθε 5 λεπτά και καλεί την συνάρτηση *check_state*.

```python
# Scheduler function to run the action every 5 minutes
def scheduler():
    while True:
        check_state()
        time.sleep(300)  # 5 minutes (300 seconds)
```

Εικόνα 94. Scheduler

Η συνάρτηση *check_state* (βλ. Εικόνα 95) διαβάζει την τρέχουσα τιμή θερμοκρασίας, ενημερώνει το αντικείμενο *TemperatureMonitor* και με βάση τις προηγούμενες τιμές θερμοκρασίας, γνωρίζει την κατάσταση που βρίσκεται ο θερμοσίφωνας τη δεδομένη στιγμή. Στη συνέχεια αποστέλλει τα δεδομένα αυτά με τις εντολές *socket.emit* όπως προηγουμένως για να εμφανιστούν στην ιστοσελίδα.

```python
# Function to run every 5 minutes
def check_state():
    temperature = readTemp()
    temp_monitor.updateTemp(temperature)

    timestamp = datetime.datetime.now()
    action = timestamp.strftime('%Y-%m-%d %H:%M') + " | Water Heater Temperature: " + str(temperature)
    action = action + ". Water Heater is in " + temp_monitor.getStatus() + " state."

    action_log.append(action)
    socketio.emit('new_action', {'action': action})

    temperature_data.append({'time': timestamp.strftime('%H:%M'), 'temperature': temperature})
    socketio.emit('new_temperature_data', {'time': timestamp.strftime('%H:%M'), 'temperature': temperature})

    socketio.emit('curr_elec_price', {'price': readElecPrice()})
```

Εικόνα 95. Συνάρτηση check_state

4.3.4 Εμφάνιση Αποτελεσμάτων στην Ιστοσελίδα

Το περιβάλλον διεπαφής της ιστοσελίδας αποτελείται από ένα πλέγμα τεσσάρων τμημάτων, σε κάθε ένα από τα οποία φαίνονται διάφορες πληροφορίες για τη λειτουργία της υπηρεσίας. Στο πάνω αριστερά μέρος φαίνεται η κατάσταση σύνδεσης με τον διακομιστή της Tuya καθώς επίσης και τα τεχνικά χαρακτηριστικά της συνδεδεμένης έξυπνης συσκευής. Στο πάνω δεξιά μέρος παρουσιάζεται ένα διάγραμμα με την εξέλιξη της θερμοκρασίας στο εσωτερικό του θερμοσίφωνα, το οποίο ενημερώνεται σε πραγματικό χρόνο. Στο κάτω αριστερά μέρος φαίνεται η τρέχουσα τιμή του ηλεκτρικού ρεύματος, όπως είναι διαθέσιμη από τον πάροχο. Στο κάτω δεξιά μέρος παρουσιάζεται ένας πίνακας με την κατάσταση του θερμοσίφωνα και την τιμή θερμοκρασίας στο εσωτερικό του που ενημερώνεται κάθε 5 λεπτά σε πραγματικό χρόνο.

Ο κώδικας HTML που δημιουργεί την σελίδα που περιγράψαμε παρατίθεται στις εικόνες 96 και 97.

index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Tuya Smart Save Application</title>
    <link rel= "stylesheet" type= "text/css" href= "{{ url_for('static',filename='styles/my-style.css') }}">
    <!-- Include Chart.js Library -->
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script src="https://cdn.socket.io/3.1.3/socket.io.min.js"
    integrity="sha384-cPwlPLvBTa3sKAgddT6krw0cJat7egBga3DJepJyrLl4Q9/5WLra3rrnMcyTyOnh" crossorigin="anonymous"></script>
</head>
<body>
    <div class="header">
        <h1>Tuya Smart Save Application</h1>
    </div>
    <div class="control-panel">
        <div class="upper-left">
            <!-- Device Specifications and Equipment Information -->
            <h2>Tuya Connection Status</h2>
            <div id="tuya_status_id">Null</div>
            <h2>Device Specifications</h2>
            <div style="margin-bottom: 15px; display: flex; justify-content: start;">
                <div style="font-weight: bold;">Category:</div> <div id="spec_category"></div>
            </div>
            <div style="margin-bottom: 15px; display: flex; justify-content: start;">
                <div style="font-weight: bold;">Device ID:</div> <div id="spec_id"></div>
            </div>
            <div style="margin-bottom: 15px; display: flex; justify-content: start;">
                <div style="font-weight: bold;">Device IP:</div> <div id="spec_ip"></div>
            </div>
            <div style="margin-bottom: 15px; display: flex; justify-content: start;">
                <div style="font-weight: bold;">Device Model:</div> <div id="spec_model"></div>
            </div>
            <div style="margin-bottom: 15px; display: flex; justify-content: start;">
                <div style="font-weight: bold;">Product Name:</div> <div id="spec_name"></div>
            </div>
            <div style="margin-bottom: 15px; display: flex; justify-content: start;">
                <div style="font-weight: bold;">Serial Number:</div> <div id="spec_sn"></div>
            </div>
        </div>
```

Εικόνα 96. Εμφάνιση αποτελεσμάτων στην Ιστοσελίδα (1/2)

```
        <div class="upper-right">
            <!-- Temperature-Time Diagram -->
            <h2>Temperature - Time Diagram</h2>
            <div class="chart-container">
                <canvas id="temperatureChart"></canvas>
            </div>
        </div>
        <div class="lower-left">
            <!-- Electricity Price Information -->
            <h2>Electricity Price Information</h2>
            <p style="display:inline">Current Electricity Price: </p>
            <p id="elec_price" style="display:inline"></p>
            <p style="display:inline">€ per kWh</p>
        </div>
        <div class="lower-right">
            <!-- Status Log Window -->
            <h2>Status Log Window</h2>
            <div id="status-log" class="log-table">
                <table id="action-table">
                    <thead>
                        <tr>
                            <th>Action</th>
                        </tr>
                    </thead>
                    <tbody>
                    </tbody>
                </table>
            </div>
        </div>
    </div>
```

Εικόνα 97. Εμφάνιση αποτελεσμάτων στην Ιστοσελίδα (2/2)

Με τη χρήση της γλώσσας *JavaScript* διαβάζουμε τις πληροφορίες που αναφέρονται στις Ενότητες 4.3.2 και 4.3.3 και τις εμφανίζουμε στην ιστοσελίδα σε κατάλληλες θέσεις με την χρήση των id που έχουμε ορίσει σε κάποια σημεία (βλ. Εικόνα 98).

```
        // Connect to the WebSocket server
        const socket = io.connect();

        // Handle the initial temperature data when connected to the server
        socket.on('initial_temperature_data', (data) => {
            const { temperatureData } = data;
            temperatureData.forEach((data) => {
                updateTemperatureChart(data.time, data.temperature);
            });
        });

        // Handle new temperature data received from the server
        socket.on('new_temperature_data', (data) => {
            const { time, temperature } = data;
            updateTemperatureChart(time, temperature);
        });
```

Εικόνα 98. Συναρτήσεις λήψης δεδομένων θερμοκρασίας

Αρχικά, συνδεόμαστε στο socket που ορίσαμε προηγουμένως για να έχουμε πρόσβαση στον δίαυλο επικοινωνίας και να μπορούμε να διαβάζουμε τα ζητούμενα δεδομένα. Με τις δύο παραπάνω συναρτήσεις ενημερώνουμε το διάγραμμα στο πάνω δεξιά τμήμα, με την πρώτη κατά την αρχική σύνδεση και με την δεύτερη κάθε 5 λεπτά με κάθε ενημέρωση νέας θερμοκρασίας. Η συνάρτηση που πραγματοποιεί αυτή τη λειτουργία

είναι η συνάρτηση *updateTemperatureChart*, η υλοποίηση της οποίας παρατίθεται παρακάτω (βλ. Εικόνες 99, 100).

```
74          let temperatureChart;
75          // Function to update the temperature-time chart
76          function updateTemperatureChart(time, temperature) {
77              if (!temperatureChart) {
78                  const ctx = document.getElementById("temperatureChart").getContext("2d");
79                  temperatureChart = new Chart(ctx, {
80                      type: "line",
81                      data: {
82                          labels: [],
83                          datasets: [{
84                              label: "Temperature (°C)",
85                              data: [],
86                              borderColor: "rgb(75, 192, 192)",
87                              backgroundColor: "rgba(75, 192, 192, 0.2)",
88                              borderWidth: 2,
89                              fill: true,
90                          }],
91                      },
92                      options: {
93                          responsive: true,
94                          maintainAspectRatio: false,
95                          scales: {
96                              x: {
97                                  display: true,
98                                  title: {
99                                      display: true,
100                                     text: "Time (HH:mm)",
101                                 },
102                             },
103                             y: {
104                                 display: true,
105                                 title: {
106                                     display: true,
107                                     text: "Temperature (°C)",
108                                 },
109                                 suggestedMin: 0, // Set the minimum value for the y-axis
110                                 suggestedMax: 40, // Set the maximum value for the y-axis
111                             },
112                         },
113                     },
114                 });
115             }
```

Εικόνα 99. Συνάρτηση updateTemperature (1/2)

```
            // Update the chart dataset with the new temperature data
            const data = temperatureChart.data;
            data.labels.push(time);
            data.datasets[0].data.push(temperature);

            // Limit the number of data points shown on the chart
            const maxDataPoints = 12;
            while (data.labels.length > maxDataPoints) {
                data.labels.shift();
                data.datasets[0].data.shift();
            }

            // Update the chart
            temperatureChart.update();
        }
```

Εικόνα 100. Συνάρτηση updateTemperature (2/2)

Η παραπάνω συνάρτηση δημιουργεί το διάγραμμα temperatureChart και κατόπιν προσθέτει τις τιμές θερμοκρασίας, κρατώντας τις 12 τελευταίες τιμές οι οποίες αντιστοιχούν στην τελευταία μία ώρα.

```
        // Handle the initial actions when connected to the server
        socket.on('initial_actions', (data) => {
            const { actions } = data;
            actions.forEach((action) => {
                updateActionLog(new Date().toLocaleTimeString(), action);
            });
        });

        // Handle new actions received from the server
        socket.on('new_action', (data) => {
            const { action } = data;
            updateActionLog(new Date().toLocaleTimeString(), action);
        });
```

Εικόνα 101. Συναρτήσεις λήψης πληροφοριών για την κατάσταση του θερμοσίφωνα

Με τις δύο παραπάνω συναρτήσεις ενημερώνουμε τον πίνακα στο κάτω δεξιά τμήμα, με την πρώτη κατά την αρχική σύνδεση και με την δεύτερη κάθε 5 λεπτά δηλαδή όταν γίνεται ανάγνωση νέας τιμής θερμοκρασίας (βλ. Εικόνα 101). Η συνάρτηση που πραγματοποιεί αυτή τη λειτουργία είναι η συνάρτηση *updateActionLog* (βλ. Εικόνα 102).

```
151         // Function to update the action log table
152         function updateActionLog(time, action) {
153             const tableBody = document.querySelector("#action-table tbody");
154             const newRow = document.createElement("tr");
155 
156             const actionCell = document.createElement("td");
157             actionCell.textContent = action;
158 
159             newRow.appendChild(actionCell);
160 
161             tableBody.appendChild(newRow);
162 
163             // Scroll to the bottom of the table to show the latest action
164             tableBody.parentElement.scrollTop = tableBody.parentElement.scrollHeight;
165         }
```

Εικόνα 102. Συνάρτηση updateActionLog

Τέλος, συμπληρώνονται η κατάσταση σύνδεσης με τον διακομιστή της Tuya, τα τεχνικά χαρακτηριστικά της συσκευής και η τρέχουσα τιμή του ηλεκτρικού ρεύματος, πληροφορίες που στάλθηκαν μέσω του διαύλου επικοινωνίας *socket.io* κατά την έναρξη της εφαρμογής. Οι συναρτήσεις που πραγματοποιούν αυτές τις λειτουργίες παρατίθενται στην Εικόνα 103.

```
185         socket.on('init_elec_price', (data) => {
186             const { price } = data;
187             document.getElementById("elec_price").textContent=price;
188         });
189 
190         socket.on('curr_elec_price', (data) => {
191             const { price } = data;
192             document.getElementById("elec_price").textContent=price;
193         });
194 
195         socket.on('tuya_status', (data) => {
196             const { status } = data;
197             if (status == "green") {
198                 document.getElementById("tuya_status_id").textContent="Connected";
199                 document.getElementById("tuya_status_id").style.color = '#008000';
200             }
201             else if (status == "red") {
202                 document.getElementById("tuya_status_id").textContent="Disconnected";
203                 document.getElementById("tuya_status_id").style.color = '#FF0000';
204             }
205         });
206 
207         socket.on('device_specs', (data) => {
208             const { specs } = data;
209             document.getElementById("spec_category").textContent=specs[0];
210             document.getElementById("spec_id").textContent=specs[1];
211             document.getElementById("spec_ip").textContent=specs[2];
212             document.getElementById("spec_model").textContent=specs[3];
213             document.getElementById("spec_name").textContent=specs[4];
214             document.getElementById("spec_sn").textContent=specs[5];
215         });
```

Εικόνα 103. Συναρτήσεις λήψης γενικών πληροφοριών

4.3.5 Βοηθητικές Συναρτήσεις

Η κλάση *TemperatureMonitor* (βλ. Εικόνα 104) είναι μία κλάση που δημιουργήθηκε με σκοπό την παρακολούθηση των τιμών θερμοκρασίας που διαβάζονται από τη συσκευή. Κρατάει αποθηκευμένες τις δώδεκα τελευταίες τιμές θερμοκρασίας σε μία δομή *deque* η οποία γεμίζει μέχρι δώδεκα τιμές και με την προσθήκη της δέκατης τρίτης αφαιρεί την παλαιότερη τιμή. Επίσης, κατά την ενημέρωση της θερμοκρασίας, ενημερώνεται και η κατάσταση (*status*) του θερμοσίφωνα με σύγκριση της τρέχουσας τιμής με την προηγούμενη με βάση την ενότητα 2.4.3.

TemperatureMonitor.py

```
import datetime
import collections

class TempMonitor:
    def __init__(self, temp = 20):
        self.temps = collections.deque([], 12)
        self.temps.appendleft(temp)
        self.status = "idle"

    def getCurrTemp(self):
        return self.temps[0]

    def getLastTemp(self):
        return self.temps[1]

    def getTemps(self):
        return self.temps

    def getStatus(self):
        return self.status

    def updateTemp(self, temp):
        self.temps.appendleft(temp)
        if abs(self.getCurrTemp() - self.getLastTemp())<=0.2:
            self.status = "idle"
        elif self.getCurrTemp() - self.getLastTemp()>0.2:
            self.status = "heating"
        elif (self.getCurrTemp() - self.getLastTemp()<-0.2) and
             (self.getCurrTemp() - self.getLastTemp()>-0.5):
            self.status = "cooling"
        elif self.getCurrTemp() - self.getLastTemp()<=-0.5:
            self.status = "use"
```

Εικόνα 104. Κλάση TemperatureMonitor

Η τρέχουσα τιμή του ηλεκτρικού ρεύματος κανονικά διαβάζεται απευθείας από τον πάροχο ηλεκτρικής ενέργειας με τη συνάρτηση *readElecPrice*. Επειδή κάτι τέτοιο δεν είναι διαθέσιμο στην Ελλάδα δημιουργήθηκε μία προσομοίωση η οποία δίνει μία ενδεικτική τιμή.

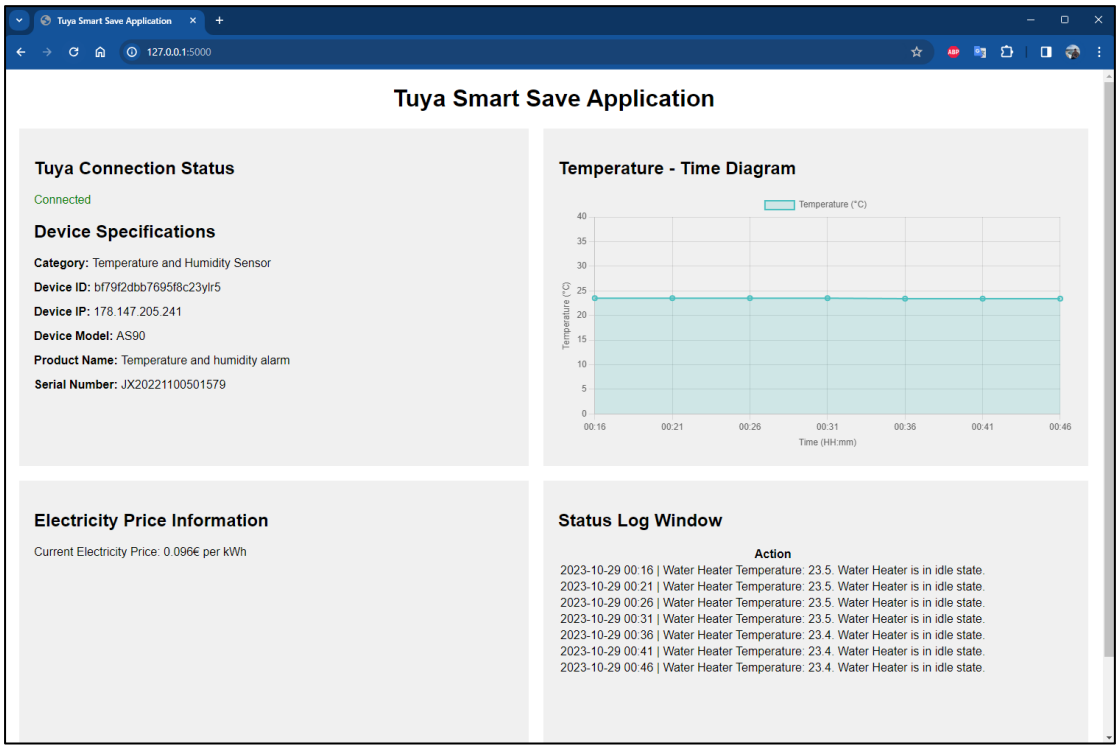
Η υλοποίηση της συνάρτησης *readElecPrice* παρατίθεται στην Εικόνα 105.

```
def readElecPrice():
    price_list = [0.103, 0.096, 0.093, 0.088, 0.084, 0.083,
                  0.085, 0.088, 0.091, 0.1, 0.11, 0.117,
                  0.118, 0.113, 0.102, 0.101, 0.101, 0.107,
                  0.117, 0.119, 0.119, 0.116, 0.109, 0.104]
    return price_list[datetime.datetime.now().hour]
```

Εικόνα 105. Συνάρτηση readElecPrice

4.3.6 Οθόνη Web Server

Με βάση τα βήματα που περιγράφηκαν στις προηγούμενες ενότητες δημιουργήθηκε ο τοπικός διακομιστής Tuya Smart Save Application. Με την εκτέλεση του αρχείου *app.py* δημιουργείται μία διεργασία παρασκηνίου και επιτρέπει στον χρήστη να επισκεφθεί το περιβάλλον επίβλεψης στην διεύθυνση http://127.0.0.1:5000/. Το τελικό αποτέλεσμα είναι αυτό που φαίνεται στην Εικόνα 106.



Εικόνα 106. Περιβάλλον Web Server

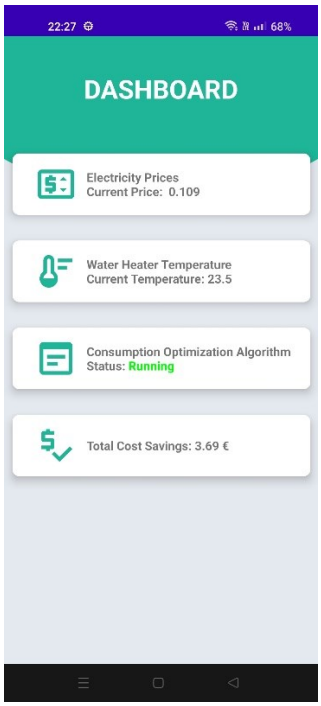
4.4 Δοκιμές και Αποτελέσματα

Για να διαπιστώσουμε την αποτελεσματικότητα της υπηρεσίας, θέσαμε την εφαρμογή σε λειτουργία για συνολικό χρονικό διάστημα έξι εβδομάδων και καταγράψαμε τις καταναλώσεις και το ποσό που εξοικονομήθηκε συνολικά κάθε εβδομάδα (βλ. Εικόνα 107). Η μέση κατανάλωση ρεύματος για τη θέρμανση του νερού ήταν 2.6 kWh και η επιθυμητή χρήση ήταν 7 φορές την εβδομάδα.

Εβδομάδα	Χρονικό Διάστημα (ημέρες)	Εξοικονόμηση (€) για το χρον. διάστημα	Σύνολο ημερών	Εξοικονόμηση (€) Συνολική
1η	7	0.58500	7	0.58500
2η	7	0.61854	14	1.20354
3η	7	0.56628	21	1.76982
4η	7	0.57720	28	2.34702
5η	7	0.66300	35	3.01002
6η	7	0.67860	42	3.68862

Εικόνα 107. Πίνακας εξοικονόμησης σε €

Στο τέλος των έξι εβδομάδων το συνολικό ποσό που εξοικονομήθηκε ήταν περίπου 3.69€ (βλ. Εικόνα 108).



Εικόνα 108. Dashboard με την ένδειξη εξοικονόμησης

Παρατηρούμε ότι υπάρχει μία σχετικά σταθερή εξοικονόμηση, η οποία είναι κατά μέσο όρο 0.61477€ την εβδομάδα. Παρατηρείται επίσης μία μικρή αύξηση στις τιμές τις τελευταίες δύο εβδομάδες, το οποίο μπορεί να δικαιολογηθεί από το γεγονός ότι μετά το πέρασμα των πρώτων 30 ημερών χρησιμοποιούμε το πιο προηγμένο και εξατομικευμένο σύστημα υπολογισμού της πιθανότητας χρήσης του θερμοσίφωνα. Συνολικά, η εξοικονόμηση που επιτεύχθηκε (αν λάβουμε ως μέσο όρο τιμής της κιλοβατώρας ίσο με 0.103€ και 2.6 kWh για κάθε χρήση) αντιστοιχεί σε περίπου 3,68862 / (0.103*2.6) = 13.77, δηλαδή ~14 χρήσεις του θερμοσίφωνα στο συνολικό διάστημα των έξι εβδομάδων (ποσοστιαία εξοικονόμηση 32.8%).

Συμπερασματικά, βλέπουμε ότι η λειτουργία της υπηρεσίας έχει θετικό πρόσημο και παρέχει στον χρήστη οφέλη όπως η εξοικονόμηση χρημάτων αλλά και η διευκόλυνσή του στη χρήση του ηλεκτρικού θερμοσίφωνα (αυτόματη ενεργοποίηση). Σημαντικά είναι επίσης και τα οφέλη που προκύπτουν στο δίκτυο ηλεκτρικής ενέργειας αφού η μετατόπιση της χρήσης ηλεκτρικής ενέργειας εκτός ωρών αιχμής βοηθάει στη μείωση του φόρτου στο ηλεκτρικό δίκτυο για αυτές τις ώρες. Αυτό είναι σημαντικό καθώς συμβάλλει στην αποφυγή διακοπών ρεύματος και στη σταθερότητα του δικτύου.

Τέλος, αναμένεται να κριθεί η αποτελεσματικότητα της υπηρεσίας σε μεγαλύτερα χρονικά διαστήματα, όπου παράγοντες όπως η αλλαγή της εποχής και του καιρού και κατά συνέπεια η αλλαγή της καθημερινότητας των χρηστών θα επηρεάσουν την λειτουργία της.

ΚΕΦΑΛΑΙΟ 5 – Μελλοντικές επεκτάσεις

Στην τελευταία ενότητα της παρούσας έρευνας παρουσιάζονται πιθανές μελλοντικές βελτιώσεις και επεκτάσεις που θα μπορούσαν να υλοποιηθούν σε μεταγενέστερο στάδιο προκειμένου να ενισχυθεί η αποτελεσματικότητα, η βιωσιμότητα και η προσαρμοστικότητα του αλγορίθμου εξοικονόμησης ενέργειας σε έξυπνα σπίτια.

5.1 Επέκταση και σε άλλες Συσκευές

Το σύστημα που αναπτύχθηκε σε αυτή τη διπλωματική εργασία προορίζεται για την βελτιστοποίηση του κόστους χρήσης του ηλεκτρικού θερμοσίφωνα σε ένα έξυπνο σπίτι, όπως ορίστηκε με βάση την παραδοχή 1. Θα μπορούσε η λειτουργία του αλγορίθμου να επεκταθεί και σε άλλες ενεργειοβόρες συσκευές, όπως για παράδειγμα το πλυντήριο ρούχων και το πλυντήριο πιάτων ή όποια άλλη συσκευή καταλαμβάνει μεγάλο μερίδιο στην ενεργειακή κατανάλωση ενός έξυπνου σπιτιού. Για να γνωρίζει ο αλγόριθμος πότε λειτουργούν οι συσκευές αυτές και κατά συνέπεια να μπορεί να μεταθέσει χρονικά τη λειτουργία τους σε ενεργειακά φθηνότερες ζώνες θα μπορούσε να χρησιμοποιηθεί μια τεχνική ανάλυσης της κατανάλωσης ενέργειας (consumption decomposition). Η διαδικασία αυτή περιλαμβάνει την εξέταση της συνολικής κατανάλωσης ενέργειας ενός συστήματος με σκοπό την αναγνώριση των διαφόρων στοιχείων και συσκευών που την απαρτίζουν.

5.2 Εξατομίκευση με Βάση τα Χαρακτηριστικά του Ηλεκτρικού Θερμοσίφωνα

Συγκεκριμένα για την περίπτωση του ηλεκτρικού θερμοσίφωνα, θα μπορούσε να διεξαχθεί μια πιο παραμετροποιημένη ανάλυση η οποία να λαμβάνει υπόψιν λεπτομερώς τα τεχνικά χαρακτηριστικά του θερμοσίφωνα. Χαρακτηριστικά όπως ο τύπος του θερμοσίφωνα, το μέγεθος του καζανιού και η μόνωση του επηρεάζουν τις απώλειες ενέργειας του θερμοσίφωνα σημαντικά και κατά συνέπεια και την θερμοκρασία στο εσωτερικό του. Με μια μελέτη των χαρακτηριστικών αυτών θα μπορούσαν να γίνουν πιο εξατομικευμένες προβλέψεις για την κατάσταση στο εσωτερικό του θερμοσίφωνα με βάση την εγκατάσταση στο εκάστοτε έξυπνο σπίτι.

5.3 Προσαρμογή στο μοντέλο δυναμικής τιμολόγησης της Ελλάδας

Όταν το μοντέλο δυναμικής τιμολόγησης γίνει διαθέσιμο στην Ελλάδα η εφαρμογή που δημιουργήθηκε θα μπορεί να προσαρμοστεί ώστε να υπάρχει μια διασύνδεση προγραμματισμού εφαρμογών (Application Programming Interface) η οποία θα επιτρέπει την ανάγνωση της τιμής της ηλεκτρικής ενέργειας σε πραγματικό χρόνο απευθείας από τον πάροχο.

5.4 Αναβάθμιση με Μηχανική Μάθηση

Μία πιθανή μελλοντική επέκταση του αλγορίθμου θα μπορούσε να περιλαμβάνει την ενσωμάτωση ενός μοντέλου μηχανικής μάθησης. Για την εκτίμηση των μελλοντικών τιμών του ηλεκτρικού ρεύματος (βλ. Ενότητα 2.2) και τον υπολογισμό της πιθανότητας

χρήσης του ηλεκτρικού θερμοσίφωνα (βλ. Ενότητα 2.3) χρησιμοποιήθηκαν στατιστικές προσεγγίσεις και το μοντέλο πρόβλεψης χρονοσειρών (για το δεύτερο). Και στις δύο περιπτώσεις, οι ακολουθίες που εξετάζονται και αναλύονται ακολουθούν ορισμένα περιοδικά μοτίβα τα οποία πολλές φορές μπορεί να μην ανιχνεύονται επαρκώς από τις απλές στατιστικές προσεγγίσεις. Κατάλληλα μοντέλα μηχανικής μάθησης θα μπορούσαν να ανιχνεύσουν αυτά τα μοτίβα και να κάνουν πιο στοχευμένες προβλέψεις δίνοντας ακριβέστερα αποτελέσματα. Για παράδειγμα, μοντέλα που θα μπορούσαν να εξεταστούν είναι: το μοντέλο ARIMA (Autoregressive Integrated Moving Average) κατάλληλο για δεδομένα που είναι σταθερά στον χρόνο, νευρωνικά δίκτυα RNNs (Recurrent Neural Networks) και LSTM (Long Short-Term Memory) που είναι μοντέλα βαθιάς μάθησης ικανά να εντοπίσουν πιο περίπλοκα μοτίβα σε ένα σύνολο δεδομένων.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] R. Lutolf, "Smart Home concept and the integration of energy meters into a home based system," *Seventh International Conference on Metering Apparatus and Tariffs for Electricity Supply 1992*, Glasgow, UK, 1992, pp. 277-278.

[2] F. K. Aldrich, "Smart Homes: Past, Present and Future," *Inside the Smart Home*, pp. 17–39, 2003, doi: https://doi.org/10.1007/1-85233-854-7_2.

[3] L. C. De Silva, C. Morikawa, and I. M. Petra, "State of the art of smart homes," *Engineering Applications of Artificial Intelligence*, vol. 25, no. 7, pp. 1313–1321, Oct. 2012, doi: https://doi.org/10.1016/j.engappai.2012.05.002.

[4] N. Balta-Ozkan, R. Davidson, M. Bicket, and L. Whitmarsh, "Social barriers to the adoption of smart homes," *Energy Policy*, vol. 63, pp. 363–374, Dec. 2013, doi: https://doi.org/10.1016/j.enpol.2013.08.043.

[5] T. Hargreaves and C. Wilson, "Introduction: Smart Homes and Their Users," *Human–Computer Interaction Series*, pp. 1–14, 2017, doi: https://doi.org/10.1007/978-3-319-68018-7_1.

[6] J. Shin, Y. Park, and D. Lee, "Who will be smart home users? An analysis of adoption and diffusion of smart homes," *Technological Forecasting and Social Change*, vol. 134, pp. 246–253, Sep. 2018, doi: https://doi.org/10.1016/j.techfore.2018.06.029.

[7] K. Gram-Hanssen and S. J. Darby, "'Home is where the smart is'? Evaluating smart home research and approaches against the concept of home," *Energy Research & Social Science*, vol. 37, pp. 94–101, Mar. 2018, doi: https://doi.org/10.1016/j.erss.2017.09.037.

[8] D. Marikyan, S. Papagiannidis, and E. Alamanos, "A systematic review of the smart home literature: A user perspective," *Technological Forecasting and Social Change*, vol. 138, no. 138, pp. 139–154, Jan. 2019, doi: https://doi.org/10.1016/j.techfore.2018.08.015.

[9] "Smart Home - number of households in the segment Smart Home in the World 2025," *Statista*. https://www.statista.com/forecasts/887613/number-of-smart-homes-in-the-smart-home-market-in-the-world.

[10] K. Moser, J. Harder, and S. G. M. Koo, "Internet of things in home automation and energy efficient smart home technologies," *IEEE Xplore*, Oct. 01, 2014. doi: https://doi.org/10.1109/SMC.2014.6974087.

[11] S. Shea, “What is smart home or building (home automation or domotics)? - Definition from WhatIs.com,” *IoT Agenda*, Jul. 2020. https://www.techtarget.com/iotagenda/definition/smart-home-or-building.

[12] B. K. Sovacool and D. D. Furszyfer Del Rio, “Smart home technologies in Europe: A critical review of concepts, benefits, risks and policies,” *Renewable and Sustainable Energy Reviews*, vol. 120, p. 109663, Mar. 2020, doi: https://doi.org/10.1016/j.rser.2019.109663.

[13] “Pros and Cons of Smart Home Technology Encompass,” *Encompassinsurance.com*, 2020. https://www.encompassinsurance.com/insurance-resources/articles/home/smart-home-technology.aspx.

[14] “30 Key Pros & Cons Of Smart Homes,” *E&C*, Jun. 12, 2020. https://environmental-conscience.com/smart-homes-pros-cons/.

[15] P. Chalikias, “Μερικά σενάρια οικιακού αυτοματισμού που πρέπει να έχει κάθε έξυπνο σπίτι,” https://www.onoffelectric.gr/articles/merika-senaria-oikiakou-automatismou-pou-prepei-na-echei-kathe-exypno-spiti.html.

[16] “Smart Home,” *Techopedia*, Jun. 29, 2023. https://www.techopedia.com/definition/smart-home.

[17] J. P. Tuohy, “How to pick a smart home platform,” *The Verge*, Jun. 12, 2023. https://www.theverge.com/23751295/smart-home-platform-google-amazon-apple-samsung?fbclid=IwAR0Senfpw5kavP6OsDOPHdzIpOQy3kQtWxIQwbt97_nNudiBxq2YEzo803E.

[18] “Tuya Smart - World’s leading IoT Platform | Bringing Best Smart Home Devices to Life for Smart Home Automation Industry,” https://www.tuya.com/.

[19] L. R. Weatherford and S. E. Bodily, “A Taxonomy and Research Overview of Perishable-Asset Revenue Management: Yield Management, Overbooking, and Pricing,” *Operations Research*, vol. 40, no. 5, pp. 831–844, Oct. 1992, doi: https://doi.org/10.1287/opre.40.5.831.

[20] I. Deksnytė and Z. Lydeka, “Dynamic pricing models and its forming factors,” International Journal of Business and Social Science, vol. 3, no. 23, pp. 213–220, 2012, https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=98bd98f38b7386ba22a52907c7ce6d586338fb30.

[21] “Dynamic Pricing What is dynamic pricing in electricity supply? 1.” Available: https://www.eurelectric.org/media/2453/dynamic-pricing-final.pdf.

[22] G. Dutta and K. Mitra, “A literature review on dynamic pricing of electricity,” *Journal of the Operational Research Society*, vol. 68, no. 10, pp. 1131–1145, Oct. 2017, doi: https://doi.org/10.1057/s41274-016-0149-4.

[23] “Έξυπνοι μετρητές ηλεκτρικής ενέργειας | EnergyLab.” http://www.energylab.gr/products/energy-monitoring.

[24] Οδηγία (ΕΕ) 2019/944 του Ευρωπαϊκού Κοινοβουλίου, https://eur-lex.europa.eu/legal-content/EL/TXT/PDF/?uri=CELEX:32019L0944

[25] ν.4986/2022, ΦΕΚ 204 Α΄/28.10.2022, https://www.nomotelia.gr/photos/File/204a-22.pdf

[26] Βίβλος Ψηφιακού Μετασχηματισμού 2020-2025 https://digitalstrategy.gov.gr/project/dimioyrgia_systimatos_dynamikis_energeiakis_timologisis

[27] ENTSOE European Network of Transmission System Operators for Electricity https://transparency.entsoe.eu/load-domain/r2/totalLoadR2/show

[28] “Python,”. https://www.python.org/

[29] “Tuya IoT Python SDK,” *GitHub*, https://github.com/tuya/tuya-iot-python-sdk (accessed Jun. 16, 2023).

[30] “Welcome to Flask — Flask Documentation (3.0.x),”. https://flask.palletsprojects.com/en/3.0.x/

[31] “Welcome to Flask-SocketIO’s documentation! — Flask-SocketIO documentation,”. https://flask-socketio.readthedocs.io/en/latest/

[32] F. Moraes, “Home,” *Html.com*, 2015. https://html.com/

[33] JavaScript, “Free JavaScript training, resources and examples for the community,” *Javascript.com*, 2016. https://www.javascript.com/

[34] “java.com: Java + You,” *Java.com*, Apr. 14, 2019. https://www.java.com/en/

[35] Android Studio, “Download Android Studio and SDK tools,” *Android Developers*, 2019. https://developer.android.com/studio

[36] “Tuya Smart Life App SDK,” *GitHub*. https://github.com/tuya/tuya-smart-life-app-sdk (accessed Jun. 16, 2023).

[37] "Tuya Android Smart Life App SDK Sample," *GitHub*. https://github.com/tuya/tuya-home-android-sdk-sample-java (accessed Jun. 16, 2023).

[38] "XML.com," *Xml.com*, 2019. https://www.xml.com/

[39] Προγράμματα χρέωσης για Tuya Developers https://developer.tuya.com/en/docs/app-development/app-sdk-price?id=Kbu0tcr2cbx3o

ΠΑΡΑΡΤΗΜΑ - Βοηθητικές συναρτήσεις και δομές για την εφαρμογή

A. Κλάση TemperatureMonitor τύπου Singleton

Η κλάση αυτή είναι μία κλάση τύπου Singleton (είδος κλάσης στην Java), η οποία έχει την ιδιαιτερότητα να έχει μόνο ένα αντίγραφο στιγμιότυπου αντικειμένου, δημιουργείται δηλαδή μόνο ένα αντικείμενο αυτής της κλάσης και παραμένει μοναδικό καθ' όλη τη λειτουργία της εφαρμογής. Το αντικείμενο *TemperatureMonitor* περιλαμβάνει μία λίστα με τις τελευταίες δώδεκα τιμές θερμοκρασίας που καταγράφηκαν από τον μετρητή. Όταν ενημερώνεται η θερμοκρασία και προστίθεται μία δέκατη τρίτη τιμή, αφαιρείται από την λίστα η παλαιότερη τιμή (βλ. Εικόνα 109).

TemperatureMonitor.java

```
18  public class TemperatureMonitor {
        4 usages
19      private static volatile TemperatureMonitor INSTANCE = null;
        7 usages
20      private ArrayList<Float> temperatures = new ArrayList<>( initialCapacity: 12);
        3 usages
21      private String curr_status = "idle", last_status = "idle";
        1 usage
22      private TemperatureMonitor() {}
        3 usages
23      public static TemperatureMonitor getInstance() {
24          if (INSTANCE == null) {
25              synchronized (TemperatureMonitor.class) {
26                  if (INSTANCE == null) {
27                      INSTANCE = new TemperatureMonitor();
28                  }
29              }
30          }
31          return INSTANCE;
32      }
```

Εικόνα 109. Κλάση TemperatureMonitor

```
52      public Float getCurrTemp(){ return this.temperatures.get(11); }
        5 usages
53      public Float getLastTemp(){ return this.temperatures.get(10); }
        1 usage
54      public ArrayList<Float> getTemperatures() { return this.temperatures; }
        1 usage
57      public void updateTemperature(Float temperature) {
58          for (int i = 1; i < 12; i++){
59              this.temperatures.set(i-1, this.temperatures.get(i));
60          }
61          this.temperatures.set(11, temperature);
62
63          this.last_status = this.curr_status;
64          if (abs(this.getCurrTemp()-this.getLastTemp()) <= 0.2){
65              this.curr_status = "idle";
66          }
67          else if (this.getCurrTemp()-this.getLastTemp() > 0.2) {
68              this.curr_status = "heating";
69          }
70          else if (this.getCurrTemp()-this.getLastTemp() < -0.2 && this.getCurrTemp()-this.getLastTemp()>-0.5) {
71              this.curr_status = "cooling";
72          }
73          else if (this.getCurrTemp()-this.getLastTemp() <= -0.5) {
74              this.curr_status = "use";
75          }
76      }
        4 usages
77      public String getCurrStatus(){
78          return curr_status;
79      }
        no usages
80      public String getLastStatus(){
81          return last_status;
82      }
```

Εικόνα 110. Συνάρτηση updateTemperature

Η συνάρτηση *updateTemperature* (βλ. Εικόνα 110) ενημερώνει την λίστα προσθέτοντας τη νέα τιμή της θερμοκρασίας και παράλληλα ενημερώνει την κατάσταση του θερμοσίφωνα συγκρίνοντας την τρέχουσα με την προηγούμενη τιμή σύμφωνα με τα κριτήρια που περιγράφονται στην ενότητα 2.4.3.

Β. Κλάσεις Shower και ShowerCollection

Η κλάση Shower (βλ. Εικόνα 111) περιλαμβάνει έναν αύξοντα αριθμό και μία ημερομηνία / ώρα που αντιστοιχεί στην ώρα όπου έγινε η χρήση του θερμοσίφωνα.

Shower.java

```java
package com.ntua.algorithm.utilities;

import java.util.Date;

7 usages
public class Shower {

    2 usages
    private final String event_id;
    2 usages
    private final Date event_datetime;
    1 usage
    public Shower(String id, Date datetime){
        event_id = id;
        event_datetime = datetime;
    }
    no usages
    public String getID() { return event_id; }
    1 usage
    public Date getDateTime() { return event_datetime; }
}
```

Εικόνα 111. Κλάση Shower

Η κλάση *ShowerCollection* (βλ. Εικόνες 113, 114, 115) αποτελείται από μία λίστα από αντικείμενα τύπου Shower. Για να χρησιμοποιήσουμε τη βιβλιοθήκη wekaForecaster προσθέτουμε την παρακάτω γραμμή στο αρχείο build.gradle (βλ. Εικόνα 112) και μετά μπορούμε να κάνουμε τα ζητούμενα imports όπως φαίνεται παρακάτω.

```
    implementation 'nz.ac.waikato.cms.weka:timeseriesForecasting:1.1.27'
```

Εικόνα 112. Ενημέρωση αρχείου build.gradle

ShowerCollection.java

```java
import weka.core.Attribute;
import weka.core.DenseInstance;
import weka.core.Instances;
import weka.classifiers.timeseries.WekaForecaster;
5 usages
public class ShowerCollection {
    8 usages
    private final ArrayList<Shower> shower_list;
    1 usage
    public ShowerCollection() { shower_list = new ArrayList<Shower>(); }
    1 usage
    public void addEvent(Shower shower) { shower_list.add(shower); }
    no usages
    public ArrayList<Shower> getEventList() { return shower_list; }
    4 usages
    public ArrayList<Date> getDateList(){
        ArrayList<Date> dates = new ArrayList<~>();
        for (int i = 0; i < shower_list.size(); i++){
            dates.add(shower_list.get(i).getDateTime());
        }
        return dates;
    }
    2 usages
    public Integer getTotalEvents() { return shower_list.size(); }
    1 usage
    public ArrayList<Integer> getEventOcc(){
        ArrayList<Integer> occ = new ArrayList<~>( initialCapacity: 24);
        for (int i = 0; i < 24; i++){
            occ.add(0);
        }
        Calendar c = Calendar.getInstance();
        for (int i = 0; i < shower_list.size(); i++){
            c.setTime(this.getDateList().get(i));
            occ.set(c.get(Calendar.HOUR_OF_DAY), occ.get(c.get(Calendar.HOUR_OF_DAY))+1);
        }
        return occ;
    }
```

Εικόνα 113. Κλάση ShowerCollection (1/3)

```java
public ArrayList<Float> getWeightedEventProbs(){
    ArrayList<Float> w_occ = new ArrayList<>(), w_probs = new ArrayList<>();
    Float sum = 0.0f;

    for (int i = 0; i < 24; i++){
        w_occ.add(0.0f);
        w_probs.add(0.0f);
    }

    Date today = new Date();
    TimeUnit timeUnit = TimeUnit.DAYS;
    Calendar c = Calendar.getInstance();

    for (int i = 0; i < shower_list.size(); i++){
        c.setTime(this.getDateList().get(i));
        Integer elapsed = Math.toIntExact(timeUnit.convert((today.getTime() - this.getDateList().get(i).getTime()), TimeUnit.MILLISECONDS));
        w_occ.set(c.get(Calendar.HOUR_OF_DAY), (float) (w_occ.get(c.get(Calendar.HOUR_OF_DAY))+0.5+1.5*Math.exp(-1*elapsed*Math.log(3)/21)));
    }
    for (int i = 0; i < 24; i++){
        sum+=w_occ.get(i);
    }
    for (int i = 0; i < 24; i++){
        w_probs.set(i, round( d: w_occ.get(i)/sum,  decimalPlace: 3));
    }
    return w_probs;
}
2 usages
public static float round(float d, int decimalPlace) {
    BigDecimal bd = new BigDecimal(Float.toString(d));
    bd = bd.setScale(decimalPlace, BigDecimal.ROUND_HALF_UP);
    return bd.floatValue();
}
```

Εικόνα 114. Κλάση ShowerCollection (2/3)

```java
public ArrayList<Float> wekaPredictor(){
    try {
        // Create an ARFF header
        ArrayList<Attribute> attributes = new ArrayList<>();
        Attribute timestampAttribute = new Attribute( attributeName: "timestamp",  dateFormat: "yyyy-MM-dd HH:mm:ss");
        Attribute eventAttribute = new Attribute( attributeName: "Event");
        attributes.add(timestampAttribute);
        attributes.add(eventAttribute);
        Instances dataset = new Instances( name: "forecast", attributes,  capacity: 0);
        List<Date> datetimeList = this.getDateList();

        // Convert datetime objects to ARFF data rows
        for (Date datetime : datetimeList) {
            double[] values = new double[2];
            values[0] = (double) datetime.getTime();

            DenseInstance instance = new DenseInstance( weight: 1.0, values);
            dataset.add(instance);
        }
        // Initialize the Weka Forecaster
        WekaForecaster forecaster = new WekaForecaster();
        forecaster.setFieldsToForecast("Event");
        // Configure the forecasting horizon (24 hours)
        forecaster.getTSLagMaker().setTimeStampField("timestamp");
        forecaster.getTSLagMaker().setMinLag(1);
        forecaster.getTSLagMaker().setMaxLag(24);
        // Build the forecasting model
        forecaster.buildForecaster(dataset, System.out);
        // Prime the forecaster with the data
        forecaster.primeForecaster(dataset);
        // Forecast the next 24 hours
        ArrayList<Float> forecastedValues = new ArrayList<>();
        for (int i = 0; i < 24; i++) {
            forecastedValues.add((float)forecaster.forecast( numSteps: 1).get(i).get(0).predicted());
        }
        return forecastedValues;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return new ArrayList<>();
}
```

Εικόνα 115. Κλάση ShowerCollection (3/3)

Οι συναρτήσεις *getWeightedEventProbs* και *wekaForecaster* υλοποιούν τις μεθόδους που περιγράφηκαν στις ενότητες 2.3.1 και 2.3.2 αντίστοιχα.

Με την κλήση της συνάρτησης *getPrediction* (βλ. Εικόνα 116) επιστρέφεται η τιμή της πρώτης μεθόδου όταν το διάστημα που έχει περάσει είναι μικρότερο από ένα μήνα και η τιμή της δεύτερης μεθόδου αν πρόκειται για διάστημα μεγαλύτερο από έναν μήνα, όπου θεωρητικά ο όγκος δεδομένων είναι επαρκής ώστε η πρόβλεψη χρονοσειρών να έχει ικανοποιητικά αποτελέσματα.

```
128     public ArrayList<Float> getPrediction(){
129         if (this.getTotalEvents()<=30){
130             return this.getWeightedEventProbs();
131         }
132         else{
133             return this.wekaPredictor();
134         }
135     }
136 }
```

Εικόνα 116. Συνάρτηση getPrediction

Γ. Κλάσεις ElectricityPrices και ElectricityPricesCollection

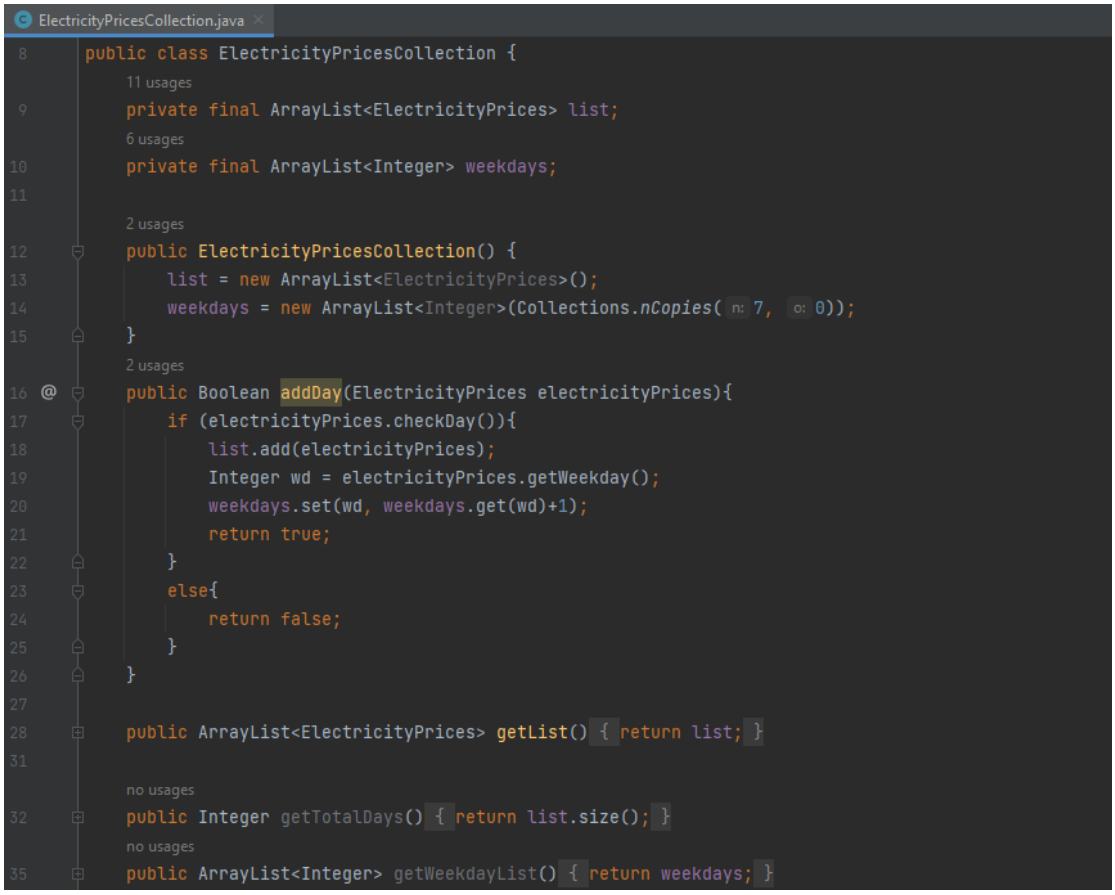
Η κλάση *ElectricityPrices* (βλ. Εικόνα 117) περιλαμβάνει μία ημερομηνία που αντιστοιχεί στην ημέρα όπου αναφέρεται το αντικείμενο και μία λίστα με τις είκοσι τέσσερις τιμές του ηλεκτρικού ρεύματος για την ημέρα αυτή.

ElectricityPrices.java

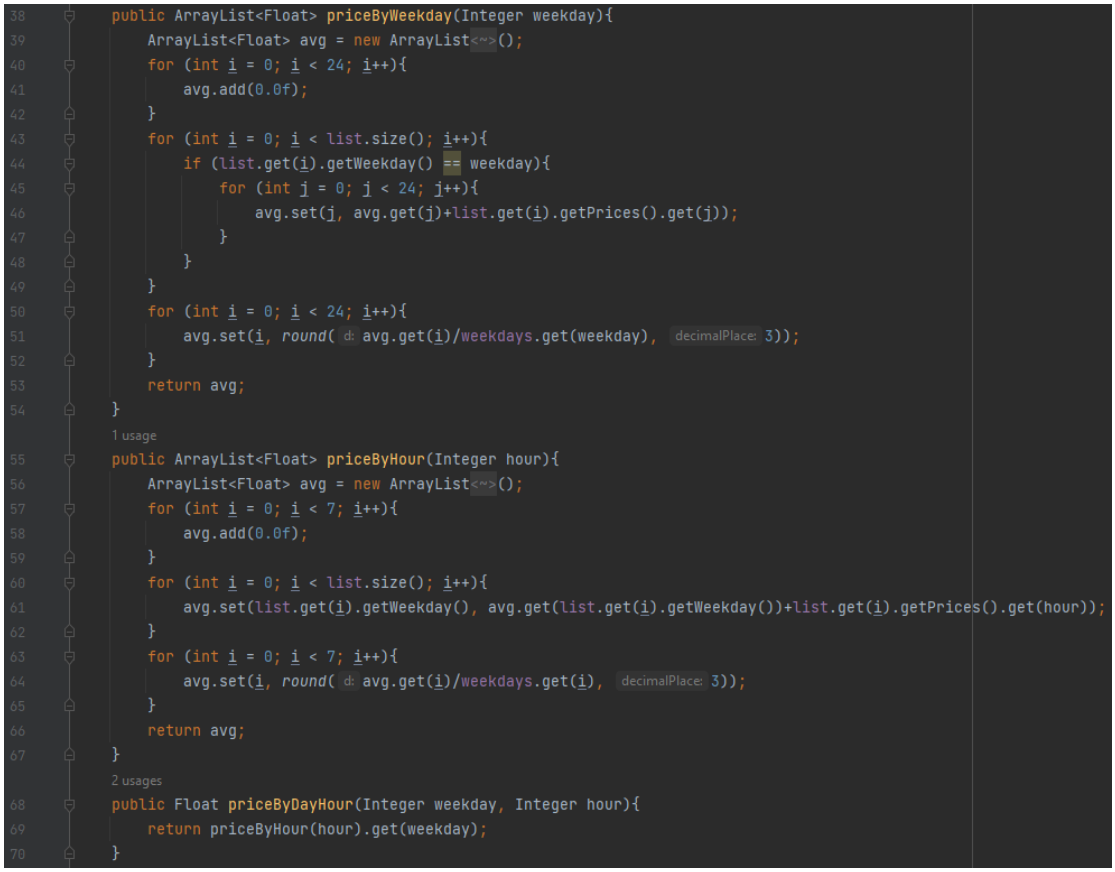
```java
package com.ntua.algorithm.utilities;

import ...

11 usages
public class ElectricityPrices {
    3 usages
    private Date date;
    5 usages
    private final ArrayList<Float> prices;

    2 usages
    public ElectricityPrices() { prices = new ArrayList<Float>(); }
    public Date getDate() { return date; }
    public void setDate(Date date) { this.date = date; }
    3 usages
    public ArrayList<Float> getPrices() { return prices; }
    1 usage
    public Integer getSize() { return this.prices.size(); }
    2 usages
    public Boolean addPrice(Float price){
        if (this.prices.size() < 24){
            this.prices.add(price);
            return true;
        }
        else{
            return false;
        }
    }
    3 usages
    public Boolean checkDay() { return this.getSize() == 24; }
    4 usages
    public Integer getWeekday(){
        Calendar c = Calendar.getInstance();
        c.setTime(this.date);
        return c.get(Calendar.DAY_OF_WEEK)-1;
    }
}
```

Εικόνα 117. Κλάση ElectricityPrices

Η κλάση *ElectricityPricesCollection* (βλ. Εικόνα 118) αποτελείται από μία λίστα από αντικείμενα τύπου *ElectricityPrices*.



Εικόνα 118. Κλάση ElectricityPricesCollection



Εικόνα 119. Συναρτήσεις priceByWeekDay, priceByHour και priceByDayHour

Οι συναρτήσεις *priceByWeekDay*, *priceByHour* και *priceByDayHour* (βλ. Εικόνα 119) επιστρέφουν τον μέσο όρο των τιμών του ηλεκτρικού ρεύματος που αντιστοιχούν σε μία συγκεκριμένη ημέρα της εβδομάδας, σε μία συγκεκριμένη ώρα της ημέρας και σε μία συγκεκριμένη ώρα μίας συγκεκριμένης ημέρας, αντίστοιχα. Υλοποιούνται εύκολα ξεχωρίζοντας από το σύνολο τις τιμές που τις αφορούν σε κάθε περίπτωση και υπολογίζοντας τον μέσο όρο τους.

Δ. Συνάρτηση readWaterHeaterEvents και αρχείο water_heater_log.xml

Η συνάρτηση αυτή διαβάζει το αρχείο *water_heater_log.xml* και επιστρέφει ένα αντικείμενο *ShowerCollection* (βλ. Εικόνα 120).

```
163  @   private ShowerCollection readWaterHeaterEvents(String path) {
164          DateFormat format = new SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm:ss", Locale.ENGLISH);
165          ShowerCollection sc = new ShowerCollection();
166          try {
167              InputStream is = getAssets().open(path);
168
169              DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
170              DocumentBuilder builder = factory.newDocumentBuilder();
171
172              Document doc = builder.parse(is);
173
174              NodeList nodeList = doc.getElementsByTagName( s: "EVENT");
175
176              for (int i = 0; i < nodeList.getLength(); i++){
177                  try {
178                      sc.addEvent(new Shower(nodeList.item(i).getChildNodes().item( i: 1).getTextContent(),
179                              format.parse(nodeList.item(i).getChildNodes().item( i: 3).getTextContent())));
180                  } catch (ParseException e){
181                      e.printStackTrace();
182                  }
183              }
184          } catch (SAXException | ParserConfigurationException | IOException e1) {
185              e1.printStackTrace();
186          }
187          return sc;
188      }
189
```

Εικόνα 120. Συνάρτηση readWaterHeaterEvents

Ένα δείγμα ενός τέτοιου αρχείου XML φαίνεται παρακάτω (βλ. Εικόνα 121).

water_heater_log.xml

```xml
<?xml version="1.0" ?>
<WATER_HEATER_EVENTS>
  <EVENT>
    <EVENT_ID>1</EVENT_ID>
    <EVENT_DATETIME>2023-01-01 07:58:00</EVENT_DATETIME>
  </EVENT>
  <EVENT>
    <EVENT_ID>2</EVENT_ID>
    <EVENT_DATETIME>2023-01-01 14:54:00</EVENT_DATETIME>
  </EVENT>
  <EVENT>
    <EVENT_ID>3</EVENT_ID>
    <EVENT_DATETIME>2023-01-01 19:02:00</EVENT_DATETIME>
  </EVENT>
  <EVENT>
    <EVENT_ID>4</EVENT_ID>
    <EVENT_DATETIME>2023-01-01 21:24:00</EVENT_DATETIME>
  </EVENT>
  <EVENT>
    <EVENT_ID>5</EVENT_ID>
    <EVENT_DATETIME>2023-01-01 22:54:00</EVENT_DATETIME>
  </EVENT>
  <EVENT>
    <EVENT_ID>6</EVENT_ID>
    <EVENT_DATETIME>2023-01-02 07:52:00</EVENT_DATETIME>
  </EVENT>
  <EVENT>
    <EVENT_ID>7</EVENT_ID>
    <EVENT_DATETIME>2023-01-02 15:11:00</EVENT_DATETIME>
  </EVENT>
  <EVENT>
    <EVENT_ID>8</EVENT_ID>
    <EVENT_DATETIME>2023-01-02 18:44:00</EVENT_DATETIME>
  </EVENT>
  <EVENT>
    <EVENT_ID>9</EVENT_ID>
    <EVENT_DATETIME>2023-01-02 23:08:00</EVENT_DATETIME>
  </EVENT>
  <EVENT>
    <EVENT_ID>10</EVENT_ID>
```

Εικόνα 121. Αρχείο καταγραφών χρήσης θερμοσίφωνα

Ε. Συνάρτηση readElectricityPrices και αρχείο electricity_price_log.xml

Η συνάρτηση αυτή διαβάζει το αρχείο *electricity_prices.xml* και επιστρέφει ένα αντικείμενο *ElectricityPricesCollection* (βλ. Εικόνα 122).

```
114    private ElectricityPricesCollection readElectricityPrices(String path) {
115        try {
116            InputStream is = getAssets().open(path);
117
118            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
119            DocumentBuilder builder = factory.newDocumentBuilder();
120
121            Document doc = builder.parse(is);
122
123            NodeList nodeList = doc.getElementsByTagName( s: "DAY");
124
125
126            for (int i = 0; i < nodeList.getLength(); i++){
127                ElectricityPrices ep = new ElectricityPrices();
128                try {
129                    ep.setDate(format.parse(nodeList.item(i).getChildNodes().item( i: 1).getTextContent()));
130                } catch (ParseException e){
131                    e.printStackTrace();
132                }
133
134                for (int j = 3; j < 51; j+=2){
135                    ep.addPrice(Float.parseFloat(nodeList.item(i).getChildNodes().item(j).getChildNodes().item( i: 3).getTextContent()));
136                }
137                if (ep.checkDay()){
138                    epc.addDay(ep);
139                }
140            }
141        } catch (SAXException | ParserConfigurationException | IOException e1) {
142            e1.printStackTrace();
143        }
144        return epc;
145    }
```

Εικόνα 122. Συνάρτηση readElectricityPrices

Ένα δείγμα ενός τέτοιου αρχείου XML φαίνεται παρακάτω (βλ. Εικόνα 123).

electricity_price_log.xml

```xml
<?xml version="1.0" ?>
<ELECTRICITY_PRICES>
   <DAY>
      <DATE>2022-01-01</DATE>
      <POINT>
         <POSITION>0</POSITION>
         <VALUE>0.103</VALUE>
      </POINT>
      <POINT>
         <POSITION>1</POSITION>
         <VALUE>0.096</VALUE>
      </POINT>
      <POINT>
         <POSITION>2</POSITION>
         <VALUE>0.093</VALUE>
      </POINT>
      <POINT>
         <POSITION>3</POSITION>
         <VALUE>0.088</VALUE>
      </POINT>
      <POINT>
         <POSITION>4</POSITION>
         <VALUE>0.084</VALUE>
      </POINT>
      <POINT>
         <POSITION>5</POSITION>
         <VALUE>0.083</VALUE>
      </POINT>
      <POINT>
         <POSITION>6</POSITION>
         <VALUE>0.085</VALUE>
      </POINT>
      <POINT>
         <POSITION>7</POSITION>
         <VALUE>0.088</VALUE>
      </POINT>
      <POINT>
         <POSITION>8</POSITION>
         <VALUE>0.091</VALUE>
```

Εικόνα 123. Αρχείο ιστορικού τιμών ηλεκτρικής ενέργειας

ΣΤ. Κλάση ElecAPI

Η τρέχουσα τιμή του ηλεκτρικού ρεύματος θεωρητικά διαβάζεται απευθείας από τον πάροχο ηλεκτρική ενέργειας μέσω ενός API. Επειδή κάτι τέτοιο δεν είναι διαθέσιμο στην Ελλάδα δημιουργούμε την προσομοιωτική κλάση *ElecAPI*, η οποία με την συνάρτηση *getPrice* θα επιστρέφει τιμή από μία λίστα ενδεικτικών τιμών, με βάση την ώρα της ημέρας (βλ. Εικόνα 124).

ElecAPI.java

```java
package com.ntua.algorithm.utilities;

import java.util.Calendar;

8 usages
public class ElecAPI {
    1 usage
    final float[] sample_prices = {0.103f, 0.096f, 0.093f, 0.088f, 0.084f, 0.083f,
            0.085f, 0.088f, 0.091f, 0.1f, 0.11f, 0.117f,
            0.118f, 0.113f, 0.102f, 0.101f, 0.101f, 0.107f,
            0.117f, 0.119f, 0.119f, 0.116f, 0.109f, 0.104f};
    3 usages
    public float getPrice(){
        Calendar c = Calendar.getInstance();
        int hour = c.get(Calendar.HOUR_OF_DAY);
        return sample_prices[hour];
    }
}
```

Εικόνα 124. Κλάση ElecAPI

Ζ. Συνάρτηση readDecisionHistory και αρχείο algorithm_history.xml

Η συνάρτηση αυτή διαβάζει το αρχείο *algorithm_history.xml* και επιστρέφει μία λίστα με όλες τις καταγραφές εκτέλεσης του αλγορίθμου που περιλαμβάνονται στο αρχείο αυτό (βλ. Εικόνα 125).

```
91 @    private ArrayList<String> readDecisionHistory() {
92          ArrayList<String> decisionHistory = new ArrayList<String>();
93          try {
94              InputStream is = getAssets().open( fileName: "algorithm_history.xml");
95
96              DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
97              DocumentBuilder builder = factory.newDocumentBuilder();
98
99              Document doc = builder.parse(is);
100
101             NodeList nodeList = doc.getElementsByTagName( s: "ACTION");
102
103             for (int i = 0; i < nodeList.getLength(); i++){
104                 decisionHistory.add(nodeList.item(i).getTextContent());
105             }
106         } catch (SAXException | ParserConfigurationException | IOException e1) {
107             e1.printStackTrace();
108         }
109         return decisionHistory;
110     }
111 }
```

Εικόνα 125. Συνάρτηση readDecisionHistory

Ένα δείγμα ενός τέτοιου αρχείου XML φαίνεται παρακάτω (βλ. Εικόνα 126).

algorithm_history.xml

```
1  <DECISIONS>
2      <ACTION>22:00 - EXPECTING SHOWER IN THE NEXT HOUR. TURNING WATER HEATER ON NOW.</ACTION>
3      <ACTION>23:00 - There is hot water left in the tank. Current water temperature: 47.3</ACTION>
4      <ACTION>00:00 - No considerable shower probability from 00 to 01</ACTION>
5      <ACTION>01:00 - No considerable shower probability from 01 to 02</ACTION>
6      <ACTION>02:00 - No considerable shower probability from 02 to 03</ACTION>
7      <ACTION>03:00 - No considerable shower probability from 03 to 04</ACTION>
8  </DECISIONS>
```

Εικόνα 126. Αρχείο καταγραφών των τελευταίων εκτελέσεων του αλγορίθμου