ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
ΣΧΟΛΗ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ ΝΕΦΟΥΣ ΓΙΑ ΤΗΝ ΦΙΛΟΞΕΝΙΑ ΚΑΙ ΟΠΤΙΚΟΠΟΙΗΣΗ ΔΕΔΟΜΕΝΩΝ ΑΠΟ ΙΟΤ ΣΥΣΚΕΥΕΣ

Ανδρέας Ζορπίδης

Επιβλέπων: Γιαννακέας Νικόλαος,
Επίκουρος Καθηγητής

Άρτα, Σεπτέμβριος 2020

ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

ΣΧΟΛΗ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ ΝΕΦΟΥΣ ΓΙΑ ΤΗΝ ΦΙΛΟΞΕΝΙΑ ΚΑΙ ΟΠΤΙΚΟΠΟΙΗΣΗ ΔΕΔΟΜΕΝΩΝ ΑΠΟ ΙΟΤ ΣΥΣΚΕΥΕΣ

Ανδρέας Ζορπίδης

Επιβλέπων: Γιαννακέας Νικόλαος,
Επίκουρος Καθηγητής

Άρτα, Σεπτέμβριος 2020

DESIGN AND DEVELOPMENT OF A CLOUD APPLICATION FOR HOSTING AND VISUALIZATION OF DATA FROM IOT DEVICES

Εγκρίθηκε από τριμελή εξεταστική επιτροπή

Άρτα, 30/09/2020

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Επιβλέπων καθηγητής

   Νικόλαος Γιαννακέας,

   Επίκουρος Καθηγητής

2. Μέλος επιτροπής

   Αλέξανδρος Τζάλλας,

   Επίκουρος Καθηγητής

3. Μέλος επιτροπής

   Ιωάννης Τσούλος

   Αναπληρωτής Καθηγητής

Ο Προϊστάμενος του Τμήματος

Ευριπίδης Γλαβάς,

Καθηγητής Α' Βαθμίδας

Υπογραφή

Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα πτυχιακή εργασία είναι εξ ολοκλήρου αποτέλεσμα δικής μου ερευνητικής εργασίας, δεν αποτελεί προϊόν αντιγραφής ούτε προέρχεται από ανάθεση σε τρίτους. Όλες οι πηγές που χρησιμοποιήθηκαν (κάθε είδους, μορφής και προέλευσης) για τη συγγραφή της περιλαμβάνονται στη βιβλιογραφία.

Ζορπίδης Ανδρέας

Υπογραφή

ΕΥΧΑΡΙΣΤΙΕΣ

Αρχικά θα ήθελα να ευχαριστήσω όλους τους καθηγητές του Τμήματος Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Ιωαννίνων για τις χρήσιμες γνώσεις και συμβουλές που μου παρείχαν. Συγκεκριμένα είμαι ευγνώμων για τον μέντορα και επιβλέπον καθηγητή της εργασίας αυτής, τον κύριο Νικόλαο Γιαννακέα ο οποίος με στήριζε και καθοδηγούσε μέχρι την ολοκλήρωση της. Ευχαριστώ επίσης τους καθηγητές Αλέξανδρο Τζάλλα, Μάρκο Τσίπουρα, Χαριλόγη Βασίλειο και τον κοσμήτορα της σχολής Ευριπίδη Γλαβά για την άψογη και επιμορφωτική συνεργασία τους στις διάφορες δραστηριότητες του τμήματος, καθώς και τον κύριο Μάριο Μάντακα για την πολύτιμη συμβολή του στα μαθήματα της κατεύθυνσης λογισμικού. Τέλος, οφείλω ένα μεγάλο ευχαριστώ στην οικογένειά μου για την αμέριστη συμπαράσταση και κατανόηση που έδειξε καθ' όλη τη διάρκεια των σπουδών μου.

ΠΕΡΙΛΗΨΗ

Η παρούσα εργασία αφορά αναγνώστες που είναι ήδη έμπειροι σε σχεδιασμό πληροφοριακών συστημάτων και έχουν βασικές γνώσεις σε τεχνολογίες διαδικτύου. Έχει ως σκοπό την εξοικείωση στον σχεδιασμό και την υλοποίηση ενός ολοκληρωμένου πληροφοριακού συστήματος δηλαδή μιας Full-Stack εφαρμογής στο υπολογιστικό νέφος υλοποιημένη σε γλώσσα προγραμματισμού JavaScript. Πιο συγκεκριμένα αφορά την αποθήκευση και την οπτικοποίηση δεδομένων κυρίως από αισθητήρες IoT συσκευών διάφορων χρηστών, επομένως η εφαρμογή χειρίζεται δεδομένα και εκτελεί λειτουργίες CRUD. Στο πρώτο κεφάλαιο γίνεται μια αναφορά στις διάφορες έννοιες, τεχνολογίες, αρχιτεκτονικές και μεθόδους που χρησιμοποιούν συχνά οι προγραμματιστές διαδικτύου στις μέρες μας για την υλοποίηση τέτοιων εφαρμογών. Έπειτα ακολουθεί ένα τμήμα σχετικά με τις IoT συσκευές που επικρατούν σήμερα και τα πρωτόκολλα επικοινωνίας που χρησιμοποιούν και τέλος αναφέρονται μερικά παρόμοια πληροφοριακά συστήματα καθώς και οι λόγοι της υλοποίησης αυτής της εφαρμογής. Ύστερα γίνεται ανάλυση των απαιτήσεων της εφαρμογής όπου αναπτύσσονται οι επιχειρησιακές διαδικασίες, οι επιχειρησιακοί κανόνες, οι μη λειτουργικές απαιτήσεις, οι λειτουργικές απαιτήσεις, το λεξικό δεδομένων και οι περιπτώσεις χρήσης. Μετά ακολουθεί η ανάλυση της αρχιτεκτονικής, ο σχεδιασμός της βάσης δεδομένων σε SQL και μία σύντομη αναφορά στα εργαλεία που χρησιμοποιήθηκαν. Στο ίδιο κεφάλαιο γίνεται ο σχεδιασμός και ανάπτυξη του REST-API ως Back-end της εφαρμογής με χρήση Node.js, Express.js, Sequelize ORM και άλλων γνωστών πακέτων όπως Axios, (JWT) jsonwebtoken, bcrypt, κλπ. και η υλοποίηση ενός (SSR) Front-End με χρήση Bootstrap, Handlebars.js, Charts.js Axios και JWT Authentication. Τέλος, γίνεται η προετοιμασία, η υποβολή της εφαρμογής στο υπολογιστικό νέφος και δοκιμές με IoT συσκευές σε πραγματικές συνθήκες.

Λέξεις-κλειδιά: Υπολογιστικό Νέφος, IoT, Full-Stack, JavaScript, Σχεδιασμός και Ανάπτυξη, Οπτικοποίηση Δεδομένων

ABSTRACT

The present thesis is dedicated to readers who are already experienced in information systems design and have a basic knowledge of the internet technologies. It aims to familiarize you with the design and development process of a complete solution, a Full-Stack application in the cloud developed in JavaScript programming language. More specifically, it is about hosting and visualization of data sent mainly from sensors of IoT devices of various users, therefore the application will be handling data and perform CRUD operations. The first chapter provides an overview of the various concepts, technologies, architectures and methods that are often used by web developers today to implement such applications. This is followed by a section about the current most used IoT devices and the communication protocols they use, and a section about some similar application platforms as well as the reasons for developing this application. Then the software requirements are analyzed where the business processes, business rules, non-functional requirements, functional requirements, data dictionary and use cases are developed. This is followed by an analysis of the architecture, the design of the SQL database and a brief overview of the tools used. The same chapter includes sections about the design and development of a REST-API Back-end using Node.js, Express.js, Sequelize ORM and other known packages such as Axios, (JWT) jsonwebtoken, bcrypt, etc. and the development of a (SSR) Front-End using Bootstrap, Handlebars.js, Charts.js Axios and JW. Finally, the application is prepared, deployed in the cloud and tests with IoT devices in real conditions are being done.

Keywords: Cloud Computing, IoT, Full-Stack, JavaScript, Design and Development, Data Visualization

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

1 Εισαγωγή

1.1 Το υπολογιστικό νέφος (Cloud) στις μέρες μας

Σύμφωνα με το Εθνικό Ινστιτούτο Προτύπων και Τεχνολογίας (NIST - National Institute of Standards and Technology) «Το υπολογιστικό νέφος είναι ένα μοντέλο που επιτρέπει τη συνεχή και εύκολη πρόσβαση σε ένα κοινόχρηστο σύνολο διαμορφώσιμων υπολογιστικών πόρων (π.χ. δίκτυα, διακομιστές, αποθηκευτικοί χώροι, εφαρμογές, και υπηρεσίες) τους οποίους μπορεί να παρέχει με ελάχιστη προσπάθεια διαχείρισης ή αλληλεπίδρασης από το πάροχο της υπηρεσίας. Αυτό το μοντέλο αποτελείται από πέντε βασικά χαρακτηριστικά, τρία μοντέλα υπηρεσιών και τέσσερα μοντέλα ανάπτυξης» Μετάφραση από [1] (σελίδα 2).

1.1.1 Βασικά χαρακτηριστικά

Κατά παραγγελία αυτοεξυπηρέτηση: Οι καταναλωτές είναι σε θέση να βοηθήσουν τον εαυτό τους και να αποφασίσουν σε ποιες υπηρεσίες θα εγγραφούν και πόσο να επενδύσουν και χωρίς να απαιτείται ανθρώπινη αλληλεπίδραση με κάθε πάροχο από τις υπηρεσίες αυτές. [2]

Πανταχού παρούσα πρόσβαση στο δίκτυο: Τα συστήματα υπολογιστικού νέφους εξαρτώνται από την υποδομή του διαδικτύου και ως εκ τούτου παρέχουν υπηρεσίες εφόσον υπάρχει σύνδεση στο διαδίκτυο (π.χ. ένα στέλεχος με έδρα τις ΗΠΑ μπορεί να εκτελέσει τους ρόλους του κατά τη διάρκεια επαγγελματικών ταξιδιών, αποκτώντας πρόσβαση στους διαδικτυακούς πόρους της εταιρείας του που φιλοξενούνται στην Ιρλανδία μέσω της σύνδεσης στο Διαδίκτυο στη Σιγκαπούρη). [2]

Ευρεία πρόσβαση στο δίκτυο: Οι δυνατότητες διατίθενται μέσω του δικτύου και η πρόσβαση μέσω τυπικών μηχανισμών που προωθούν τη χρήση από ετερογενείς λεπτές ή παχιές πλατφόρμες πελατών (π.χ. κινητά τηλέφωνα, tablet, φορητούς υπολογιστές και σταθμούς εργασίας). [2]

Συγκέντρωση πόρων: Η συνδυασμένη υπολογιστική ισχύς μεγάλων ποσοτήτων φυσικών και εικονικών διακομιστών παρέχει μια οικονομικά αποδοτική συγκέντρωση πόρων. Οι

λύσεις πολλαπλών λειτουργιών επέτρεψαν σε αρκετούς οργανισμούς να μοιράζονται τους ίδιους πόρους υπολογιστικού νέφους χωρίς να ανησυχούν για τη διαρροή δεδομένων στα λογικά όρια του άλλου. [2]

Γρήγορη ελαστικότητα: Τα συστήματα υπολογιστικού νέφους αξιοποιούν τεχνολογίες όπως η εικονικοποίηση του διακομιστή και του συστήματος αποθήκευσης προκειμένου να μπορούν να καλύψουν τη γρήγορη άνοδο και μείωση του φόρτου χρήσης και ζήτησης των υπηρεσιών. (π.χ. μια νέα επιχείρηση που αναμένει 10.000 πελάτες θα είναι σε θέση να χειριστεί ένα απροσδόκητο φορτίο 1 εκατομμυρίου πελατών χωρίς να ανησυχεί για την ανάγκη αγοράς ή δημιουργίας νέων διακομιστών σε σύντομο χρονικό διάστημα). Η ελαστικότητα αυτή βελτιώνει επίσης τη χρήση των πόρων του υπολογιστικού νέφους. [2]

Μετρημένη υπηρεσία: Τα συστήματα υπολογιστικού νέφους ελέγχουν και βελτιστοποιούν αυτόματα τη χρήση πόρων, αξιοποιώντας ένα σύστημα μέτρησης - πληρωμής ανάλογα τον τύπο της υπηρεσίας που χρησιμοποιείται, κάτι που προσφέρει διαφάνεια τόσο στον πάροχο όσο και στον καταναλωτή. [2]

1.1.2 Μοντέλα υπηρεσιών

Software as a Service (SaaS): Στο μοντέλο «λογισμικό ως υπηρεσία» παρέχεται στο καταναλωτή η δυνατότητα να χρησιμοποιεί τις εφαρμογές του παρόχου που εκτελούνται σε μία υποδομή υπολογιστικού νέφους. Οι εφαρμογές είναι προσβάσιμες από διάφορες συσκευές πελατών είτε μέσω μίας λεπτής διεπαφής όπως ένας περιηγητής ιστού (π.χ. μέσω ηλεκτρονικού ταχυδρομείου) είτε μέσω μίας διεπαφής προγράμματος. Ο καταναλωτής δεν διαχειρίζεται ούτε ελέγχει την υποκείμενη υποδομή υπολογιστικού νέφους, συμπεριλαμβανομένων των δικτύων, διακομιστών, λειτουργικών συστημάτων, αποθηκευτικών συστημάτων ή ακόμη και μεμονωμένων δυνατοτήτων των εφαρμογών, με πιθανή εξαίρεση ορισμένων ρυθμίσεων της εφαρμογής που αφορούν τον χρήστη της εφαρμογής. Μετάφραση από [1] (σελίδα 2).

Platform as a Service (PaaS): Στο μοντέλο «πλατφόρμα ως υπηρεσία» παρέχεται στο καταναλωτή η δυνατότητα να εγκαθιστά/δημοσιεύει στις υποδομές του υπολογιστικού νέφους εφαρμογές που δημιουργήθηκαν από τον ίδιο ή αποκτήθηκαν και που δημιουργήθηκαν χρησιμοποιώντας γλώσσες προγραμματισμού, βιβλιοθήκες, υπηρεσίες

και εργαλεία που υποστηρίζονται από το πάροχο. Ο καταναλωτής δεν διαχειρίζεται ή ελέγχει την υποκείμενη υποδομή υπολογιστικού νέφους, συμπεριλαμβανομένων των δικτύων, διακομιστών, λειτουργικών συστημάτων, αποθηκευτικών συστημάτων, αλλά έχει τον έλεγχο των δικών του εφαρμογών και πιθανόν ρυθμίσεων που αφορούν το περιβάλλον φιλοξενίας των εφαρμογών αυτών. Μετάφραση από [1] (σελίδα 2).

Infrastructure as a Service (IaaS): Στο μοντέλο «υποδομή ως υπηρεσία» παρέχεται στο καταναλωτή η δυνατότητα επεξεργασίας, αποθήκευσης, δικτύωσης και άλλων θεμελιωδών υπολογιστικών πόρων όπου μπορεί να αναπτύξει και να εκτελέσει αυθαίρετο λογισμικό, το οποίο μπορεί να περιλαμβάνει λειτουργικά συστήματα και εφαρμογές. Ο καταναλωτής δεν διαχειρίζεται ή ελέγχει την υποκείμενη υποδομή υπολογιστικού νέφους αλλά έχει τον έλεγχο των λειτουργικών συστημάτων, της αποθήκευσης, των εφαρμογών που έχουν αναπτυχθεί και πιθανώς περιορισμένο έλεγχο επιλεγμένων στοιχείων δικτύωσης (π.χ. τείχη προστασίας). Μετάφραση από [1] (σελίδα 3).

1.1.3 Μοντέλα ανάπτυξης

Private cloud: Στο «ιδιωτικό νέφος» η υποδομή παρέχεται για αποκλειστική χρήση από ένα μονάχα οργανισμό που περιλαμβάνει πολλούς καταναλωτές(π.χ. επιχειρηματικές μονάδες). Μπορεί να ανήκει, να διοικείται και να διαχειρίζεται από τον οργανισμό, έναν τρίτο ή κάποιο συνδυασμό αυτών. Μετάφραση από [1] (σελίδα 3).

Community cloud: Στο «νέφος κοινότητας» η υποδομή παρέχεται για αποκλειστική χρήση από μία συγκεκριμένη κοινότητα καταναλωτών από οργανώσεις που έχουν κοινά ενδιαφέροντα (π.χ. αποστολή, απαιτήσεις ασφαλείας, θέματα πολιτικής και συμμόρφωσης). Μπορεί να ανήκει, να ελέγχεται και να διαχειρίζεται από μία ή περισσότερες οργανώσεις της κοινότητας, έναν τρίτο ή κάποιο συνδυασμό αυτών. Μετάφραση από [1] (σελίδα 3).

Public cloud: Το «δημόσιο νέφος» παρέχεται για ελεύθερη χρήση από το ευρύ κοινό. Μπορεί να ανήκει, να ελέγχεται και να διαχειρίζεται από μία επιχείρηση, ένα ακαδημαϊκό ή κυβερνητικό οργανισμό ή κάποιο συνδυασμό αυτών. Μετάφραση από [1] (σελίδα 3).

Hybrid cloud: Στο «υβριδικό νέφος» η υποδομή αποτελείται από τον συνδυασμό δύο ή περισσότερων μοντέλων ανάπτυξης. Μετάφραση από [1] (σελίδα 3).

1.2 Το διαδίκτυο των πραγμάτων (IoT) στην εποχή μας

Το Διαδίκτυο των πραγμάτων (IoT) περιγράφει το δίκτυο φυσικών αντικειμένων - "πράγματα" - που είναι ενσωματωμένα με αισθητήρες, λογισμικό και άλλες τεχνολογίες με σκοπό τη σύνδεση και την ανταλλαγή δεδομένων με άλλες συσκευές και συστήματα μέσω του διαδικτύου. Αυτές οι συσκευές κυμαίνονται από συνηθισμένα οικιακά αντικείμενα έως εξελιγμένα βιομηχανικά εργαλεία. Μετάφραση από [3]

Τα τελευταία χρόνια, το IoT έχει γίνει μια από τις πιο σημαντικές τεχνολογίες του 21ου αιώνα. Τώρα που μπορούμε να συνδέσουμε καθημερινά αντικείμενα - συσκευές κουζίνας, αυτοκίνητα, θερμοστάτες, οθόνες μωρών - στο Διαδίκτυο μέσω ενσωματωμένων συσκευών, είναι δυνατή η απρόσκοπτη επικοινωνία μεταξύ ανθρώπων, διαδικασιών και πραγμάτων. Μετάφραση από [3]

Μέσω υπολογιστών χαμηλού κόστους, το υπολογιστικό νέφος και τα κινητά τηλέφωνα, οι συσκευές αυτές μπορούν να μοιράζονται και να συλλέγουν δεδομένα με ελάχιστη ανθρώπινη παρέμβαση. Στη σημερινή εποχή, τα ψηφιακά συστήματα μπορούν να καταγράφουν, να παρακολουθούν και να προσαρμόζουν κάθε αλληλεπίδραση μεταξύ συνδεδεμένων πραγμάτων. Ο φυσικός κόσμος συναντά τον ψηφιακό κόσμο και συνεργάζονται. Μετάφραση από [3]

1.2.1 Χρήσεις και οφέλη

Η ικανότητα του IoT να παρέχει πληροφορίες από αισθητήρες καθώς και να επιτρέπει την επικοινωνία μεταξύ συσκευών οδηγεί σε ένα ευρύ φάσμα εφαρμογών και οφελών. Παρακάτω είναι μερικές από τις πιο δημοφιλείς εφαρμογές του. Αποσπάσματα μεταφρασμένα από [3]

- Δημιουργία νέων αποδόσεων στον τομέα των κατασκευών μέσω της παρακολούθησης με χρήση μηχανημάτων και μέσω της παρακολούθησης ποιότητας προϊόντων. Τα μηχανήματα μπορούν να παρακολουθούνται συνεχώς και να αναλύονται ώστε να βεβαιώνεται η λειτουργία τους εντός των απαιτούμενων ανοχών. Τα προϊόντα μπορούν επίσης να

παρακολουθούνται σε πραγματικό χρόνο για τον εντοπισμό και την αντιμετώπιση ελαττωμάτων στη ποιότητα.

- Βελτίωση στην παρακολούθηση και προστασία των περιουσιακών στοιχείων. Η παρακολούθηση επιτρέπει στις επιχειρήσεις να προσδιορίζουν γρήγορα την τοποθεσία των περιουσιακών τους στοιχείων και να διασφαλίσουν ότι τα περιουσιακά στοιχεία υψηλής αξίας προστατεύονται από κλοπή.
- Χρήση φορητών (wearable) συσκευών για την παρακολούθηση και την ανάλυση της ανθρώπινης υγείας και των περιβαλλοντικών συνθηκών. Τα φορητά IoT επιτρέπουν στους ανθρώπους να κατανοήσουν καλύτερα τη δική τους υγεία και επιτρέπουν στους γιατρούς να παρακολουθούν εξ αποστάσεως τους ασθενείς. Αυτή η τεχνολογία επιτρέπει επίσης στις εταιρείες να παρακολουθούν την υγεία και την ασφάλεια των υπαλλήλων τους, κάτι που είναι ιδιαίτερα χρήσιμο για τους εργαζόμενους που απασχολούνται σε επικίνδυνες συνθήκες.
- Ενεργοποίηση αλλαγών σε επιχειρηματικές διαδικασίες. Ένα παράδειγμα αυτού είναι η χρήση συσκευών IoT για την παρακολούθηση της υγείας των απομακρυσμένων μηχανών και την δημιουργία κλήσεων για συντήρηση. Η δυνατότητα παρακολούθησης απομακρυσμένων μηχανών επιτρέπει επίσης νέα επιχειρηματικά μοντέλα προϊόντων ως υπηρεσία, όπου οι πελάτες δεν χρειάζεται πλέον να αγοράζουν ένα προϊόν, αλλά να πληρώνουν για τη χρήση του.

1.2.2 Δημοφιλείς οικογένειες συσκευών

Το Διαδίκτυο των πραγμάτων εξελίσσεται με γρήγορους ρυθμούς κυρίως εξαιτίας της ανάπτυξης της τεχνολογίας και της μαζικής παραγωγής ηλεκτρονικών εξαρτημάτων. Αυτό έχει επίσης ως αποτέλεσμα την μεγάλη διαθεσιμότητά και με μικρό κόστος. Πλέον είναι ευρέως διαθέσιμα IoT kits και συσκευές ανάπτυξης που συνδυάζουν ισχυρούς μικροελεγκτές χαμηλής κατανάλωσης, επεξεργαστές που παρέχουν ασύρματη επικοινωνία και άλλα εξαρτήματα όπως αισθητήρες σε ένα πακέτο, έτοιμο για προγραμματισμό.

Υπάρχουν διαθέσιμες διάφορες διαμορφώσεις, από all-in-one ολοκληρωμένα που τροφοδοτούνται από μπαταρίες και επικοινωνούν μέσω Bluetooth σε υπολογιστές μεγέθους

πιστωτικής κάρτας που τροφοδοτούνται μέσω USB και περιέχουν αυτόνομα ολοκληρωμένα για επικοινωνία μέσω WiFi.

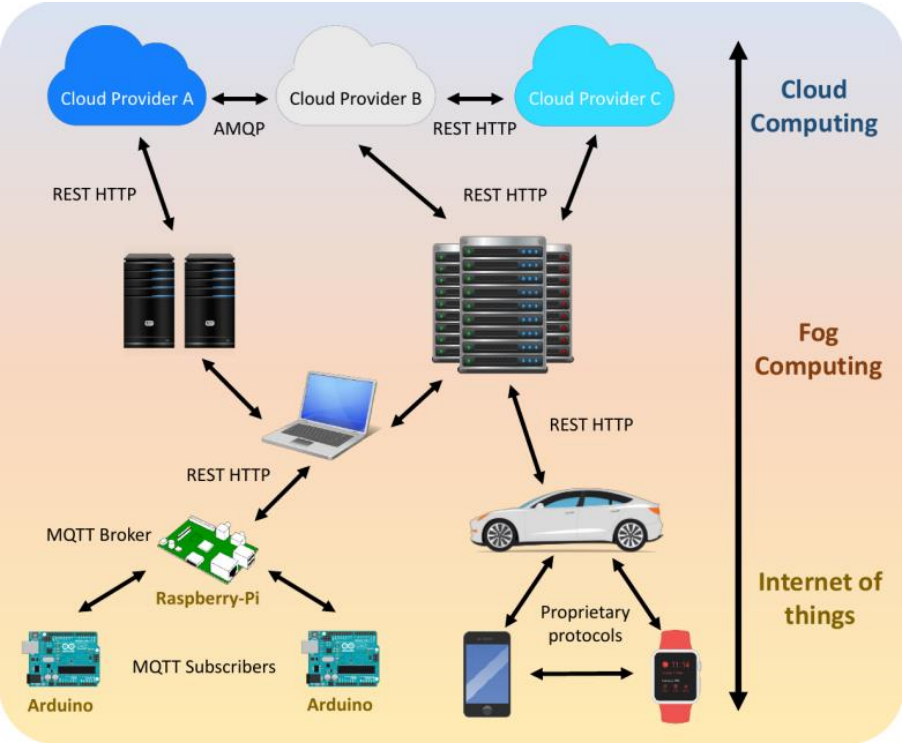
Μερικές συσκευές και οικογένειες συσκευών που χρησιμοποιούνται συχνά στην εποχή μας για την ανάπτυξη IoT συσκευών είναι οι παρακάτω:

- Arduino
- Respberry Pi
- NodeMCU
- Wemos/Lolin
- ESP32
- ESP8266
- LoRa
- BeagleBoard
- Nvidia Jetson
- Odroid
- RAK
- Intel Edison
- Particle
- Pycom
- Udoo
- Adafruit FONA
- Adafruit Feather
- Tessel
- Mediatek LinkIt

1.2.3 Μέθοδοι αποστολής και συλλογής δεδομένων

Κατά την λήψη, αποστολή και συλλογή των δεδομένων, τα δεδομένα συχνά διανύουν τρία επίπεδα αρχιτεκτονικών (εξαρτάται και από την εκάστοτε υλοποίηση) όπως Cloud Computing, Fog Computing, Internet of Things και μεσολαβούν διάφορα πρωτόκολλα επικοινωνίας. Παρακάτω αναφέρονται μερικά από αυτά μαζί με αποσπάσματα μεταφρασμένα από το δημοσίευμα (A Survey of Communication Protocols for Internet of

Things and Related Challenges of Fog and Cloud Computing Integration των JASENKA DIZDAREVIĆ, FRANCISCO CARPIO, ADMELA JUKAN, XAVI MASIP-BRUIN).



Εικόνα 1 IoT και Fog σε Cloud [4] (σελίδα 3)

1.2.3.1 REST HTTP

Αυτό το πρωτόκολλο είναι το βασικό πρωτόκολλο μοντέλου πελάτη-διακομιστή που χρησιμοποιείται από το διαδίκτυο και το πιο συμβατό με την υπάρχουσα υποδομή δικτύου. Προς το παρόν, η πιο ευρέως αποδεκτή έκδοση αυτού του πρωτοκόλλου είναι η HTTP/1.1. Στο πρωτόκολλο HTTP, ένας πελάτης στέλνει ένα αίτημα/μήνυμα σε ένα διακομιστή και ο διακομιστής επιστρέφει ένα μήνυμα απόκρισης που περιέχει τον πόρο που ζητήθηκε σε περίπτωση που το αίτημα έγινε δεκτό. Τα τελευταία χρόνια, το HTTP συσχετίστηκε με το αρχιτεκτονικό στυλ REST για τη δημιουργία API με μεγάλη επιτυχία και γι' αυτό το λόγο έχουν γίνει προσπάθειες ενσωμάτωσης αυτού του αρχιτεκτονικού στυλ σε συστήματα που βασίζονται στο Internet of Things. Ο συνδυασμός του πρωτοκόλλου HTTP με REST είναι χρήσιμος καθώς οι συσκευές μπορούν να κάνουν τα δεδομένα τους εύκολα διαθέσιμα μέσω βασικών λειτουργιών (δημιουργίας, ανάγνωσης, ενημέρωσης και διαγραφής) γνωστών ως CRUD. Σύμφωνα με αυτήν τη χαρτογράφηση, οι λειτουργίες για τη δημιουργία, ενημέρωση, ανάγνωση και διαγραφή πόρων αντιστοιχούν στις μεθόδους HTTP POST,

GET, PUT και DELETE, αντίστοιχα. Για τους προγραμματιστές, το γεγονός ότι το REST δημιουργεί μια χαρτογράφηση αυτών των λειτουργιών CRUD με μεθόδους HTTP, σημαίνει ότι μπορούν εύκολα να δημιουργήσουν ένα μοντέλο REST για διαφορετικές συσκευές IoT. Η παρουσίαση των δεδομένων δεν είναι προκαθορισμένη και ως εκ τούτου, ο τύπος είναι αυθαίρετος, με τα πιο κοινά να είναι JSON και XML. Στις περισσότερες περιπτώσεις, το IoT χρησιμοποιεί JSON μέσω HTTP. Μετάφραση από [4] (σελίδα 6).

1.2.3.2 MQTT

Το MQTT είναι ένα από τα ελαφριά πρωτόκολλα ανταλλαγής μηνυμάτων που ακολουθούν το παράδειγμα δημοσίευσης-εγγραφής, το οποίο το καθιστά μάλλον κατάλληλο για συσκευές περιορισμένης χρήσης πόρων και για μη ιδανικές συνθήκες σύνδεσης δικτύου, όπως με χαμηλό εύρος ζώνης και υψηλό λανθάνοντα χρόνο. Το MQTT κυκλοφόρησε από την IBM, με την τελευταία έκδοση MQTT v3.1 που υιοθετήθηκε για το IoT από το OASIS. Λόγω της απλότητάς του και της πολύ μικρής κεφαλίδας μηνυμάτων σε σύγκριση με άλλα πρωτόκολλα ανταλλαγής μηνυμάτων, συνιστάται συχνά ως λύση στην επικοινωνία του IoT. Το MQTT λειτουργεί πάνω από το πρωτόκολλο μεταφοράς TCP, το οποίο διασφαλίζει την αξιοπιστία του. Σε σύγκριση με άλλα αξιόπιστα πρωτόκολλα, όπως το HTTP, και χάρη στην ελαφρύτερη κεφαλίδα του, το MQTT διαθέτει πολύ χαμηλότερες απαιτήσεις ισχύος, καθιστώντας το μία από τις πιο εξέχουσες λύσεις πρωτοκόλλου σε περιορισμένα περιβάλλοντα. Μετάφραση από [4] (σελίδα 9).

1.2.3.3 CoAP

Αυτό το πρωτόκολλο σχεδιάστηκε από την ομάδα Constrained RESTful Environments (CoRE) του IETF για χρήση σε περιορισμένες συσκευές με περιορισμένες δυνατότητες επεξεργασίας. Παρόμοιο με το HTTP, ένα από τα πιο καθοριστικά χαρακτηριστικά του είναι η χρήση αρχιτεκτονικής REST. Με αυτήν τη δυνατότητα, το CoAP υποστηρίζει το παράδειγμα αιτήματος / απόκρισης όπως το REST HTTP, και ειδικά σε περιορισμένα περιβάλλοντα. Το CoAP θεωρείται ένα ελαφρύ πρωτόκολλο, επομένως οι κεφαλίδες, οι μέθοδοι και οι κωδικοί κατάστασης είναι όλοι κωδικοποιημένοι σε δυαδικό, μειώνοντας έτσι την επιβάρυνση του πρωτόκολλο σε σύγκριση με πολλά πρωτόκολλα. Τρέχει επίσης σε λιγότερο περίπλοκο πρωτόκολλο μεταφοράς UDP αντί για TCP, μειώνοντας περαιτέρω την επιβάρυνση. Όταν ένας πελάτης CoAP στέλνει ένα ή περισσότερα αιτήματα CoAP στον

διακομιστή και λαμβάνει την απάντηση, αυτή η απάντηση δεν αποστέλλεται μέσω μιας προηγούμενης σύνδεσης, αλλά ανταλλάσσεται ασύγχρονα μέσω μηνυμάτων CoAP. Η τιμή που καταβάλλεται για αυτήν τη μείωση είναι η αξιοπιστία. Μετάφραση από [4] (σελίδα 8).

1.2.3.4 AMQP

Το AMQP είναι ένα ανοιχτού τύπου πρωτόκολλο που ακολουθεί το πρότυπο δημοσίευσης-εγγραφής, όπως ορίζεται από το OASIS, σχεδιασμένο να επιτρέπει τη διαλειτουργικότητα μεταξύ ενός ευρέος φάσματος διαφορετικών εφαρμογών και συστημάτων, ανεξάρτητα από τα εσωτερικά τους σχέδια. Αρχικά αναπτύχθηκε για ανταλλαγή μηνυμάτων επιχειρήσεων με την ιδέα να προσφέρει μια μη ιδιόκτητη λύση που μπορεί να διαχειριστεί μεγάλο αριθμό ανταλλαγών μηνυμάτων που θα μπορούσαν να συμβούν σε σύντομο χρονικό διάστημα σε ένα σύστημα. Αυτό το χαρακτηριστικό διαλειτουργικότητας AMQP είναι σημαντικό καθώς επιτρέπει σε διαφορετικές πλατφόρμες, που είναι υλοποιημένες σε διαφορετικές γλώσσες προγραμματισμού, να ανταλλάσσουν μηνύματα, τα οποία ίσως είναι ιδιαίτερα χρήσιμα σε ετερογενή συστήματα. Μετάφραση από [4] (σελίδια 12).

1.2.3.5 DDS

Το DDS είναι ένα πρότυπο διαλειτουργικότητας για δεδομένα σε πραγματικό χρόνο και χρησιμοποιεί ένα μοντέλο αλληλεπίδρασης δημοσίευσης-εγγραφής, όπως ορίζεται από την Object Management Group (OMG). Σε αντίθεση με ορισμένα πρωτόκολλα εγγραφής δημοσίευσης, το DDS είναι αποκεντρωμένο και βασίζεται σε επικοινωνία peer-to-peer. Στο DDS, οι εκδότες και οι συνδρομητές μπορούν να επικοινωνούν ως ομότιμοι μέσω του διαύλου δεδομένων, επιτρέποντας την ασύγχρονη ανταλλαγή δεδομένων με βάση τα ενδιαφέροντά τους. Το γεγονός ότι δεν υπάρχει broker μειώνει επίσης την πιθανότητα αστοχίας του συστήματος, καθιστώντας ένα σύστημα πιο αξιόπιστο. Οι δύο πλευρές επικοινωνίας δεν είναι συνδεδεμένες μεταξύ του και ένας εκδότης μπορεί να δημοσιεύει δεδομένα ακόμη και αν δεν υπάρχουν ενδιαφερόμενοι συνδρομητές. Η χρήση δεδομένων είναι ουσιαστικά ανώνυμη, καθώς οι εκδότες δεν ρωτούν ποιος καταναλώνει τα δεδομένα τους. Μετάφραση από [4] (σελίδα 11).

1.2.3.6 XMPP

Το XMPP είναι ένα ανοιχτού τύπου πρωτόκολλο ανταλλαγής μηνυμάτων που έχει επισημοποιηθεί από το IETF και σχεδιάστηκε αρχικά για ανταλλαγή άμεσων μηνυμάτων και ανταλλαγή μηνυμάτων μεταξύ εφαρμογών. Πρόκειται για ένα πρωτόκολλο που βασίζεται σε κείμενο, βασισμένο σε Extensible Markup Language (XML) που εφαρμόζει αλληλεπίδραση πελάτη-διακομιστή και δημοσίευσης-εγγραφής, που εκτελείται μέσω TCP. Στο IoT έχει σχεδιαστεί για να επιτρέπει στους χρήστες να στέλνουν μηνύματα σε πραγματικό χρόνο. Το XMPP επιτρέπει στις εφαρμογές άμεσων μηνυμάτων να επιτυγχάνουν όλες τις βασικές δυνατότητες, συμπεριλαμβανομένου του ελέγχου ταυτότητας, της κρυπτογράφησης και της συμβατότητας με άλλα πρωτόκολλα. Μετάφραση από [4] (σελίδα 13).

1.3 Cloud και IoT

Το IoT αρχίζει να μεταμορφώνει το τρόπο που γίνονται καθημερινές εργασίες. Στις μέρες μας το IoT αποτελείται από απλές καθημερινές συσκευές, μικρούς υπολογιστές, οχήματα και φυσικά αντικείμενα που έχουν ενσωματωμένα ηλεκτρονικά κυκλώματα, αισθητήρες, λογισμικό και συνδεσιμότητα με το διαδίκτυο που τους επιτρέπουν να συλλέγουν, να στέλνουν και να λαμβάνουν δεδομένα. Ο τεράστιος αυτός όγκος δεδομένων (Big Data) που δημιουργείται καθημερινά από το IoT δημιουργεί φόρτο στις υποδομές του δικτύου και αυτό αναγκάζει τις εταιρείες να βρουν λύσεις για την ελαχιστοποίησή του αλλά και λύσεις στο πρόβλημα της μεταφοράς του μεγάλου όγκου δεδομένων. Η συνεργασία του υπολογιστικού νέφους με το IoT παρέχει νέες λύσεις στις εταιρείες σχετικά με το πρόβλημα αυτό αλλά και αρκετά επιπλέον οφέλη. Μερικά από αυτά είναι τα παρακάτω:

- Αποτελεσματικός τρόπος συλλογής δεδομένων από μη ελεγχόμενες και αυτόνομες συσκευές.
- Αποθήκευση, ανάλυση, οπτικοποίηση και διαχείριση δεδομένων από μεγάλο αριθμό συσκευών.
- Άμεση πρόσβαση σε δεδομένα ανεξάρτητα από τη κατάσταση της σύνδεσης των συσκευών.
- Πρόσβαση σε δεδομένα σε πραγματικό χρόνο από απόσταση.
- Απομακρυσμένος έλεγχος της φυσικής κατάστασης των συσκευών.
- Καταγραφή λειτουργιών.
- Επεκτασιμότητα δεδομένων.

1.4 Παρόμοια πληροφοριακά συστήματα

Έπειτα από εκτενή έρευνα, βρέθηκαν αρκετά παρόμοια πληροφοριακά συστήματα και υπηρεσίες. Μερικά από αυτά που ξεχώρισαν αναφέρονται παρακάτω:

ThingsBoard

Το ThingsBoard [5] είναι μία πλατφόρμα IoT ανοιχτού κώδικα που παρέχει λειτουργίες συλλογής, επεξεργασίας και οπτικοποίησης δεδομένων αλλά και διαχείριση συσκευών. Επίσης, παρέχει συνδεσιμότητα μέσω πρωτοκόλλων IoT όπως MQTT, CoAP και HTTP και υποστηρίζει εγκαταστάσεις στο υπολογιστικό νέφος αλλά και τοπικές.

Grafana

Το Grafana [6] είναι ένα λογισμικό ανοιχτού κώδικα, πιο συγκεκριμένα μια πλατφόρμα για την οπτικοποίηση και ανάλυση δεδομένων. Δεν αποθηκεύει δεδομένα αλλά παρέχει τη δυνατότητα σύνδεσης με βάσεις δεδομένων.

Thinger.io

Το Thinger.io [7] είναι μία πλατφόρμα υπολογιστικού νέφους που παρέχει εργαλεία για πρωτότυπες δοκιμές, κλιμάκωση και διαχείριση συνδεδεμένων συσκευών. Δεν αποθηκεύει δεδομένα, ούτε συνδέεται με κάποια βάση δεδομένων, οι συσκευές συνδέονται σε αυτό μέσω του δικού του API και χρησιμοποιείται για την οπτικοποίηση της μετάδοσης των δεδομένων.

Microsoft Azure IoT Suite

Το Azure Internet of Things [8] είναι μία συλλογή από υπηρεσίες υπολογιστικού νέφους που διαχειρίζεται η Microsoft και συνδέουν, παρακολουθούν και ελέγχουν IoT συσκευές. Πρόκειται για μία λύση IoT που αποτελείται από μία ή περισσότερες υπηρεσίες Back-end που φιλοξενούνται στο υπολογιστικό νέφος.

Google Cloud IoT Platform

Το Google Cloud IoT [9] είναι μία συλλογή εργαλείων για τη σύνδεση, την επεξεργασία, την αποθήκευση και την ανάλυση δεδομένων στο υπολογιστικό νέφος. Η πλατφόρμα αποτελείται από επεκτάσιμες, πλήρως διαχειριζόμενες υπηρεσίες υπολογιστικού νέφους.

AWS IoT Platform

Το AWS IoT Core [10] είναι μία πλατφόρμα που επιτρέπει τη σύνδεση συσκευών με τις υπηρεσίες AWS, την ασφαλή αποθήκευση δεδομένων και την επεξεργασία/ενέργεια βάσει των δεδομένων των συσκευών. Επίσης επιτρέπει στις εφαρμογές να αλληλοεπιδρούν με συσκευές ακόμα και όταν είναι εκτός σύνδεσης.

Salesforce IoT Cloud

Το Salesforce IoT Cloud [11] είναι μία πλατφόρμα για την αποθήκευση και επεξεργασία δεδομένων IoT. Χρησιμοποιεί το Thunder engine για κλιμάκωση και επεξεργασία σε πραγματικό χρόνο και παρέχει μία συλλογή από εργαλεία ανάπτυξης εφαρμογών, γνωστή ως Lightning.

Kaa IoT Platform

Το Kaa [12] είναι μία πλατφόρμα, ενδιάμεσο λογισμικό ανοιχτού κώδικα που χρησιμοποιείται για την εφαρμογή ολοκληρωμένων λύσεων IoT, συνδεδεμένων εφαρμογών και έξυπνων προϊόντων. Προσφέρει μια σειρά από λειτουργίες IoT για επιχειρήσεις που μπορούν εύκολα να συνδεθούν και να χρησιμοποιηθούν για την υλοποίηση πολλών περιπτώσεων χρήσης IoT. Οι δυνατότητες της πλατφόρμας περιλαμβάνουν διαχείριση συσκευών, συλλογή δεδομένων, διαχείριση διαμόρφωσης, ανταλλαγή μηνυμάτων και άλλα.

Thingspeak IoT Platform

Το ThingSpeak [13] είναι μία πλατφόρμα αναλύσεων IoT που επιτρέπει τη συγκέντρωση, την οπτικοποίηση και την ανάλυση ζωντανών ροών δεδομένων στο υπολογιστικό νέφος.

Παρέχει άμεση απεικόνιση των δεδομένων που δημοσιεύονται από τις συσκευές και επιτρέπει την ανάλυση και επεξεργασία τους την στιγμή της δημοσίευσης με χρήση κώδικα MATLAB.

DeviceHive

Το DeviceHive [14] είναι μία δωρεάν ανοιχτού κώδικα πλατφόρμα με δυνατότητες συλλογής, επεξεργασίας, ανάλυσης και οπτικοποίησης δεδομένων καθώς και διαχείρισης συσκευών.

Mainflux

Το Mainflux [15] είναι μία δωρεάν ανοιχτού κώδικα πλατφόρμα ανεπτυγμένη σε Go. Δέχεται συνδέσεις από χρήστες και συσκευές μέσω διάφορων πρωτοκόλλων επικοινωνίας όπως (HTTP, MQTT, WebSocket, CoAP). Χρησιμοποιείται ως ενδιάμεσο λογισμικό σε IoT για τη κατασκευή πολύπλοκων IoT λύσεων.

1.5 Σύνοψη εργασίας

Κατά τη διάρκεια της έρευνας, παρατηρήθηκε πως πολλά από τα παρόμοιά πληροφοριακά συστήματα όπως τα IBM Watson IoT Platform, Cisco IoT Cloud Connect, GE Predix IoT Platform και Oracle IoT Platform πρόσφεραν κυρίως βιομηχανικές enterprise λύσεις, ως αποτέλεσμα η πρόσβασή τους στο πλαίσιο της έρευνας και των δοκιμών ήταν δύσκολη ή και αδύνατη. Άλλα ενώ έδειχναν να προσφέρουν μία ολοκληρωμένη λύση αποθήκευσης και οπτικοποίησης των δεδομένων, εν τέλη πρόσφεραν μόνο τη ζωντανή αναπαράστασή τους με μορφή γραφημάτων ή πρόσφεραν «σουίτες» με εργαλεία για την ενσωμάτωσή τους σε άλλες εφαρμογές. Επίσης τα περισσότερα ακολουθούσαν τα μοντέλα υπηρεσιών Cloud Computing PaaS (πλατφόρμα ως υπηρεσία) και SaaS (λογισμικό ως υπηρεσία) και είχαν πολύ μεγάλες υποδομές κάτι που περιόριζε την προσαρμοστικότητα, ενώ μερικά από αυτά δεν παρείχαν τη δυνατότητα εγκατάστασης σε υπολογιστικούς πόρους του χρήστη ή τοπικά. Έτσι κρίθηκε απαραίτητη η υλοποίηση μιας εφαρμογής που θα μπορεί να εγκατασταθεί σε υπολογιστικούς πόρους του χρήστη είτε τοπικά είτε στο υπολογιστικό νέφος και που θα παρέχει ένα πυρήνα με χρήσιμες λειτουργίες οι οποίες θα μπορούν να επεκταθούν μελλοντικά. Μερικές από αυτές τις λειτουργίες είναι η εγγραφή και διαχείριση χρηστών, η υποστήριξη IoT συσκευών με διάφορα πρωτόκολλα επικοινωνίας (για αρχή HTTP) και η αποθήκευση και η οπτικοποίηση δεδομένων (για αρχή με γραφήματα). Στο επόμενο κεφάλαιο λοιπόν γίνεται η ανάλυση και η παρουσίαση της εφαρμογής που υλοποιήθηκε στα πλαίσια της εργασίας αυτής.

2 Η εφαρμογή

2.1 Απαιτήσεις

2.1.1 Επιχειρησιακές διαδικασίες εφαρμογής

Ο κατάλληλος τρόπος για να βρεθούν οι λειτουργικές απαιτήσεις της εφαρμογής και οι περιπτώσεις χρήσης είναι η καταγραφή των επιχειρησιακών διαδικασιών δηλαδή λειτουργιών. [16] (σελίδα 103). Οι επιχειρησιακές διαδικασίες που κρίθηκαν απαραίτητες για την εφαρμογή είναι οι παρακάτω.

Αυθεντικοποίηση

- Σύνδεση χρήστη
- Αποσύνδεση χρήστη
- Εγγραφή χρήστη

Χρήστες

- Εμφάνιση ταμπλό διαχείρισης
- Εμφάνιση προφίλ χρήστη
- Επεξεργασία ρυθμίσεων προφίλ χρήστη

Συσκευές

- Εμφάνιση καταλόγου συσκευών χρήστη
- Επεξεργασία στοιχείων συσκευής
- Διαγραφή συσκευής

Αισθητήρες

- Εμφάνιση καταλόγου αισθητήρων χρήστη
- Επεξεργασία στοιχείων αισθητήρα
- Διαγραφή αισθητήρα

Κλειδιά API

- Εμφάνιση καταλόγου κλειδιών API χρήστη

- Επεξεργασία στοιχείων κλειδιών API
- Διαγραφή κλειδιών API

Μετρήσεις

- Προσθήκη μετρήσεων συσκευής χρήστη
- Εμφάνιση μετρήσεων συσκευής χρήστη

2.1.2 Επιχειρησιακοί κανόνες

Συντομογραφία br: business-rule

Ταυτότητα / Όνομα	**Περιγραφή**
br.usage	Η εφαρμογή επιτρέπει την χρήση της από πολλούς χρήστες και συσκευές ταυτόχρονα.
br.device	Κάθε συσκευή αντιστοιχεί σε ένα μόνο χρήστη.
br.device.type	Κάθε συσκευή αντιστοιχεί σε ένα τύπο (http ή mqtt).
br.sensor	Κάθε αισθητήρας αντιστοιχεί σε ένα μόνο χρήστη και μία μόνο συσκευή.
br.sensor.type	Κάθε αισθητήρας αντιστοιχεί σε ένα τύπο (θερμοκρασίας, υγρασίας κλπ)
br.apikey	Η εφαρμογή παρέχει εξουσιοδότηση σε κάθε συσκευή για αποστολή δεδομένων σε αυτή με ένα κλειδί API. Κάθε κλειδί API αντιστοιχεί σε ένα μόνο χρήστη και μία μόνο συσκευή.
br.apikey.status	Κάθε κλειδί API μπορεί να ενεργοποιείται και να απενεργοποιείται από τον χρήστη.
br.apikey.rights	Κάθε κλειδί API μπορεί να αντιστοιχεί σε δικαιώματα που ορίζει ο χρήστης.

br.measurement	Κάθε μέτρηση αντιστοιχεί σε ένα μόνο χρήστη και μία μόνο συσκευή ενώ περιέχει τιμές από τους αισθητήρες που αντιστοιχούν σε αυτή τη συσκευή.

Πίνακας 1 Επιχειρησιακοί κανόνες

2.1.3 Μη λειτουργικές απαιτήσεις

Συντομογραφία nfr: non-functional requirements

Ταυτότητα / Όνομα	**Περιγραφή**
nfr.design.responsive	Οι διεπαφές της εφαρμογής θα πρέπει να εμφανίζονται σωστά σε κινητά, tablet και υπολογιστές με διάφορες διαστάσεις οθονών.
nfr.development.db	Η εφαρμογή θα έχει βάση δεδομένων MySQL.
nfr.development.language	Η εφαρμογή θα αναπτυχθεί σε JavaScript.
nfr.development.server	Ο διακομιστής της εφαρμογής θα υλοποιηθεί με Express.js.
nfr.rendering	Η εφαρμογή θα κάνει Server-Side Rendering.
nfr.development.view	Οι προβολές τις εφαρμογής θα αναπτυχθούν με χρήση Handlebars.js.
nfr.development.view.engine	Τα view engine της εφαρμογής θα είναι το HBS για Express.js.
nfr.userinterface	Το ταμπλό διαχείρισης θα υλοποιηθεί με χρήση πρότυπου AdminLTE.

Πίνακας 2 Μη λειτουργικές απαιτήσεις

2.1.4 Λειτουργικές απαιτήσεις

Συντομογραφία fr: functional requirements

Ταυτότητα / Όνομα	**Περιγραφή**
fr.authentication.signup	Η εφαρμογή θα επιτρέπει στον μη εγγεγραμμένο χρήστη να εγγραφτεί.
fr.authentication.sign	Η εφαρμογή θα επιτρέπει στον μη συνδεδεμένο χρήστη να συνδεθεί.
fr.authentication.logout	Η εφαρμογή θα επιτρέπει στον συνδεδεμένο χρήστη να αποσυνδεθεί.
fr.user.profile	Η εφαρμογή θα επιτρέπει στον συνδεδεμένο χρήστη να δει τα προσωπικά του στοιχεία.
fr.user.profile.settings	Η εφαρμογή θα επιτρέπει στον συνδεδεμένο χρήστη να επεξεργαστεί τα προσωπικά του στοιχεία.
fr.user.session	Η συνεδρία θα περιλαμβάνει τις εξής λειτουργίες: Αρχικά: • Σύνδεση Στη συνέχεια: • Εμφάνιση ταμπλό χρήση Η συνεδρία θα τερματίζεται εφόσον ο χρήστης επιλέξει αποσύνδεση.
fr.user.device.displayAll	Όταν ο συνδεδεμένος χρήστης επιλέξει την εμφάνιση του καταλόγου συσκευών, η εφαρμογή θα εμφανίζει όλες τις συσκευές του χρήστη.
fr.user.device.add	Η εφαρμογή θα επιτρέπει στον συνδεδεμένο χρήστη να προσθέσει νέες συσκευές, να τους αναθέσει στοιχεία όπως όνομα και περιγραφή και να επιλέξει τον τύπο τους.
fr. user.device.edit	Η εφαρμογή θα επιτρέπει στον συνδεδεμένο χρήστη να επεξεργαστεί τα στοιχεία των συσκευών του.
fr.user.device.delete	Η εφαρμογή θα επιτρέπει στον συνδεδεμένο χρήστη να διαγράψει συσκευές του. Με τη διαγραφή μιας

	συσκευής θα διαγράφονται οι αισθητήρες, τα κλειδιά API και οι μετρήσεις που σχετίζονται με αυτή.
fr.user.device.measurements	Όταν ο συνδεδεμένος χρήστης επιλέξει την σελίδα μίας συσκευής, η εφαρμογή θα εμφανίζει γραφήματα για κάθε τύπο αισθητήρα από τον οποίο υπάρχει μέτρηση και σε κάθε γράφημα θα εμφανίζει κάθε αισθητήρα (ίδιου τύπου) με διαφορετικό χρώμα. Η ενημέρωση των γραφημάτων θα γίνεται κάθε 2 δευτερόλεπτα.
fr.user.device.measurements.add	Εξουσιοδοτημένες συσκευές θα μπορούν να στείλουν νέες μετρήσεις για όσους αισθητήρες τους αντιστοιχούν και οι οποίοι είναι σε λειτουργία.
fr.user.sensor.displayAll	Όταν ο συνδεδεμένος χρήστης επιλέξει την εμφάνιση του καταλόγου αισθητήρων, η εφαρμογή θα εμφανίζει όλους τους αισθητήρες του χρήστη.
fr.user.sensor.add	Η εφαρμογή θα επιτρέπει στον συνδεδεμένο χρήστη να προσθέσει νέους αισθητήρες στις συσκευές του, να τους αναθέσει στοιχεία όπως όνομα και περιγραφή και να επιλέξει τον τύπο τους.
fr.user.sensor.edit	Η εφαρμογή θα επιτρέπει στον συνδεδεμένο χρήστη να επεξεργαστεί τα στοιχεία των αισθητήρων του.
fr.user.sensor.delete	Η εφαρμογή θα επιτρέπει στον συνδεδεμένο χρήστη να διαγράψει αισθητήρες του. Με τη διαγραφή ενός αισθητήρα θα διαγράφονται οι τιμές από τις μετρήσεις που σχετίζονται με αυτόν.
fr.user.apikey.displayAll	Όταν ο συνδεδεμένος χρήστης επιλέξει την εμφάνιση του καταλόγου κλειδιών API, η εφαρμογή θα εμφανίζει όλα τα κλειδιά API του χρήστη.
fr.user.apikey.add	Η εφαρμογή θα επιτρέπει στον συνδεδεμένο χρήστη να προσθέσει νέα κλειδιά API στις συσκευές του, να τους αναθέσει στοιχεία όπως όνομα, περιγραφή και να επιλέξει τα δικαιώματα τους.

fr.user.apikey.edit	Η εφαρμογή θα επιτρέπει στον συνδεδεμένο χρήστη να επεξεργαστεί τα στοιχεία των αισθητήρων του.
fr.user.apikey.delete	Η εφαρμογή θα επιτρέπει στον συνδεδεμένο χρήστη να διαγράψει κλειδιά API του.

Πίνακας 3 Λειτουργικές απαιτήσεις

Επιπλέον υλοποιήθηκαν οι λειτουργικές απαιτήσεις για τους χρήστες με ρόλο/δικαιώματα διαχειριστή αλλά δε θα γίνει περαιτέρω ανάλυσή τους σε αυτή την εργασία.

2.1.5 Λεξικό δεδομένων

Δεδομένο	**Περιγραφή**	**Συνιστώσες ή τύπος**	**Μήκος**	**Τιμές**
Δεδομένα σύνδεσης		= username + password		
username	Username χρήστη	string	32	
password	Κωδικός χρήστη	string	60	
Δεδομένα εγγραφής		= username + email + password		
email	Email χρήστη	string	100	
Δεδομένα προφίλ χρήστη		= username + user role + email + firstname + lastname		
user role	Ρόλος χρήστη	string	32	user
firstname	Όνομα χρήστη	string	60	
lastname	Επώνυμο χρήστη	string	60	

Δεδομένα συσκευής		= device id + device name + device description + device type		
device id	ID συσκευής	int	11	
device name	Όνομα συσκευής	string	64	
device description	Περιγραφή συσκευής	string	200	
device type	Τύπος συσκευής	string	32	http_device
Δεδομένα αισθητήρα		= sensor id + sensor name + sensor description + sensor type + sensor latitude + sensor longitude + device id		
sensor id	ID αισθητήρα	int	11	
sensor name	Όνομα αισθητήρα	string	32	
sensor description	Περιγραφή αισθητήρα	string	200	
sensor type	Τύπος αισθητήρα	string	32	
sensor latitude	Γεωγραφικό πλάτος αισθητήρα	decimal	10,8	0

sensor longitude	Γεωγραφικό μήκος αισθητήρα	decimal	111,8	0
Δεδομένα κλειδιού API		= apikey id + apikey name + apikey description + device id + apikey permissions + apikey status		
apikey id	ID κλειδιού API	int	11	
apikey name	Όνομα κλειδιού API	string	64	
apikey description	Περιγραφή κλειδιού API	string	200	
apikey permissions	Δικαιώματα κλειδιού API	string	32	all
apikey status	Κατάσταση κλειδιού API	tinyint	1	
Δεδομένα μέτρησης		= username + device id + + measurement id + sensor id + measurement sensor value		
measurement id	ID Μέτρησης	int	11	
measurement sensor value	Τιμή μέτρησης του αισθητήρα	float		0

Πίνακας 4 Λεξικό δεδομένων

2.1.6 Περιπτώσεις χρήσης (use cases)

2.1.6.1 Εγγραφή χρήστη

Περίπτωση χρήσης: Εγγραφή χρήστη

Ταυτότητα: uc.authentication.signup

Δράστης: Χρήστης

Αρχικές συνθήκες: Η κατάσταση του χρήστη είναι «ασύνδετος» και «μη εγγεγραμμένος».

Βασική ροή:

Β0α: Ο δράστης ζητά να κάνει εγγραφή.

Β0β: Η εφαρμογή ζητά από τον δράστη να καταχωρήσει τα στοιχεία του.

Β1α: Ο δράστης δίνει τα στοιχεία του.

Β1β: Η εφαρμογή

Β1β1: ελέγχει τα στοιχεία και εφόσον είναι έγκυρα

Β1β2: τα καταχωρεί και

Β1β3: τον συνδέει.

Τελική συνθήκη μετά τη βασική ροή: Η κατάσταση του χρήστη είναι «εγγεγραμμένος και «συνδεδεμένος».

Εναλλακτικές ροές:

Ε1: Αν στο βήμα Β1β1 τα στοιχεία δεν είναι έγκυρα, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β0β.

2.1.6.2 Σύνδεση χρήστη

Περίπτωση χρήσης: Σύνδεση χρήστη

Ταυτότητα: uc.authentication.sign

Δράστης: Χρήστης

Αρχικές συνθήκες: Η κατάσταση του χρήστη είναι «ασύνδετος».

Βασική ροή:

Β0α: Ο δράστης ζητά να συνδεθεί.

Β0β: Η εφαρμογή ζητά από τον δράστη να καταχωρήσει τα στοιχεία του.

Β1α: Ο δράστης δίνει τα στοιχεία του.

Β1β: Η εφαρμογή

Β1β1: ελέγχει τα στοιχεία και εφόσον είναι έγκυρα

Β1β2: τον συνδέει.

Τελική συνθήκη μετά τη βασική ροή: Η κατάσταση του χρήστη είναι «συνδεδεμένος».

Εναλλακτικές ροές:

Ε1: Αν στο βήμα Β1β1 τα στοιχεία δεν είναι έγκυρα, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β0β.

2.1.6.3 Αποσύνδεση χρήστη

Περίπτωση χρήσης: Αποσύνδεση χρήστη

Ταυτότητα: uc.authentication.logout

Δράστης: Χρήστης

Αρχικές συνθήκες: Η κατάσταση του χρήστη είναι «συνδεδεμένος».

Βασική ροή:

Β0α: Ο δράστης ζητά να αποσυνδεθεί.

Β0β: Η εφαρμογή τον αποσυνδέει.

Τελική συνθήκη μετά τη βασική ροή: Η κατάσταση του χρήστη είναι «ασύνδετος».

Εναλλακτικές ροές: καμία

2.1.6.4 Εμφάνιση του προφίλ χρήστη

Περίπτωση χρήσης: Εμφάνιση του προφίλ χρήστη

Ταυτότητα: uc.user.profile

Δράστης: Χρήστης

Αρχικές συνθήκες: Η κατάσταση του χρήστη είναι «συνδεδεμένος».

Βασική ροή:

Β0α: Ο δράστης ζητά να δει το προφίλ του με τα προσωπικά του στοιχεία.

Β0β: Η εφαρμογή εμφανίζει το προφίλ

Εναλλακτικές ροές: καμία.

2.1.6.5 Επεξεργασία του προφίλ χρήστη

Περίπτωση χρήσης: Επεξεργασία του προφίλ χρήστη

Ταυτότητα: uc.user.profile.settings

Δράστης: Χρήστης

Αρχικές συνθήκες: Η κατάσταση του χρήστη είναι «συνδεδεμένος».

Βασική ροή:

Β0α: Ο δράστης ζητά να επεξεργαστεί τα προσωπικά του στοιχεία

Β0β: Η εφαρμογή

Β0β1: εμφανίζει τα υπάρχοντα στοιχεία του χρήστη στα πεδία και

Β0β2: ζητά από τον δράστη να τα αλλάξει και να τα καταχωρήσει.

Β1α: Ο δράστης αλλάζει τα στοιχεία και τα καταχωρεί.

Β1β: Η εφαρμογή

Β1β1: ελέγχει τα στοιχεία και εφόσον είναι έγκυρα

Β1β2: καταχωρεί τα νέα στοιχεία και

Β1β3: εμφανίζει το προφίλ του.

Εναλλακτικές ροές:

Ε1: Αν στο βήμα Β1β1 τα δεδομένα δεν είναι έγκυρα, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β0β.

2.1.6.6 Εμφάνιση του ταμπλό διαχείρισης

Περίπτωση χρήσης: Εμφάνιση του ταμπλό διαχείρισης

Ταυτότητα: uc.admin.index

Δράστης: Χρήστης

Αρχικές συνθήκες: Η κατάσταση του χρήστη είναι «συνδεδεμένος».

Βασική ροή:

Β0α: Ο δράστης ζητά να δει το ταμπλό διαχείρισης.

Β0β: Η εφαρμογή εμφανίζει ο ταμπλό διαχείρισης

Β0β1: και εμφανίζει τα τελευταία δεδομένα.

Τελική συνθήκη μετά τη βασική ροή: Επαναφόρτωση του βήματος Β0β1 μετά από 2 δευτερόλεπτα.

Εναλλακτικές ροές: καμία.

2.1.6.7 Προσθήκη νέας συσκευής

Περίπτωση χρήσης: Προσθήκη νέας συσκευής

Ταυτότητα: uc.device.add

Δράστης: Χρήστης

Αρχικές συνθήκες: Η κατάσταση του χρήστη είναι «συνδεδεμένος».

Βασική ροή:

Β0α: Ο δράστης ζητά να προσθέσει μια νέα συσκευή.

Β0β: Η εφαρμογή ζητά από τον δράστη να καταχωρήσει τα δεδομένα της συσκευής.

Β1α: Ο δράστης δίνει τα δεδομένα της συσκευής.

Β1β: Η εφαρμογή

Β1β1: ελέγχει τα δεδομένα και εφόσον είναι έγκυρα

Β1β2: καταχωρεί τη νέα συσκευή

Β1β3: εμφανίζει τον κατάλογο συσκευών.

Εναλλακτικές ροές:

Ε1: Αν στο βήμα Β1β1 τα δεδομένα δεν είναι έγκυρα, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β0β.

2.1.6.8 Εμφάνιση του καταλόγου συσκευών

Περίπτωση χρήσης: Εμφάνιση του καταλόγου συσκευών

Ταυτότητα: uc.devices.displayAll

Δράστης: Χρήστης

Αρχικές συνθήκες: Η κατάσταση του χρήστη είναι «συνδεδεμένος».

Βασική ροή:

Β0α: Ο δράστης ζητά να δει το κατάλογο συσκευών.

Β0β: Η εφαρμογή εμφανίζει το κατάλογο συσκευών.

Εναλλακτικές ροές: καμία.

2.1.6.9 Επεξεργασία συσκευής

Περίπτωση χρήσης: Επεξεργασία συσκευής

Ταυτότητα: uc.device.edit

Δράστης: Χρήστης

Αρχικές συνθήκες: Η κατάσταση του χρήστη είναι «συνδεδεμένος».

Βασική ροή:

Β0α: Ο δράστης ζητά να επεξεργαστεί τα δεδομένα μιας συσκευή.

Β0β: Η εφαρμογή

Β0β1: εμφανίζει τα υπάρχοντα δεδομένα της συσκευής στα πεδία και

Β0β2: ζητά από τον δράστη να τα αλλάξει και να τα καταχωρήσει.

Β1α: Ο δράστης αλλάζει τα δεδομένα της συσκευής και τα καταχωρεί.

Β1β: Η εφαρμογή

Β1β1: ελέγχει τα δεδομένα και εφόσον είναι έγκυρα

Β1β2: καταχωρεί τα νέα δεδομένα και

Β1β3: εμφανίζει τον κατάλογο συσκευών.

Εναλλακτικές ροές:

Ε1: Αν στο βήμα Β1β1 τα δεδομένα δεν είναι έγκυρα, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β0β.

2.1.6.10 Διαγραφή συσκευής

Περίπτωση χρήσης: Διαγραφή συσκευής

Ταυτότητα: uc.device.delete

Δράστης: Χρήστης

Αρχικές συνθήκες: Η κατάσταση του χρήστη είναι «συνδεδεμένος».

Βασική ροή:

Β0α: Ο δράστης ζητά να διαγράψει μια συσκευή.

Β0α: Η εφαρμογή

Β0α1: διαγράφει τη συσκευή και

Β0α2: εμφανίζει τον κατάλογο των συσκευών

Εναλλακτικές ροές: καμία

2.1.6.11 Προσθήκη νέου αισθητήρα

Περίπτωση χρήσης: Προσθήκη νέου αισθητήρα

Ταυτότητα: uc.sensor.add

Δράστης: Χρήστης

Αρχικές συνθήκες: Η κατάσταση του χρήστη είναι «συνδεδεμένος».

Βασική ροή:

Β0α: Ο δράστης ζητά να προσθέσει ένα νέο αισθητήρα.

Β0β: Η εφαρμογή ζητά από τον δράστη να καταχωρήσει τα δεδομένα του αισθητήρα.

Β1α: Ο δράστης δίνει τα δεδομένα του αισθητήρα.

Β1β: Η εφαρμογή

Β1β1: ελέγχει τα δεδομένα και εφόσον είναι έγκυρα

Β1β2: καταχωρεί το νέο αισθητήρα

Β1β3: εμφανίζει τον κατάλογο αισθητήρων.

Εναλλακτικές ροές:

Ε1: Αν στο βήμα Β1β1 τα δεδομένα δεν είναι έγκυρα, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β0β.

2.1.6.12 Εμφάνιση του καταλόγου αισθητήρων

Περίπτωση χρήσης: Εμφάνιση του καταλόγου αισθητήρων

Ταυτότητα: uc.sensors.displayAll

Δράστης: Χρήστης

Αρχικές συνθήκες: Η κατάσταση του χρήστη είναι «συνδεδεμένος».

Βασική ροή:

Β0α: Ο δράστης ζητά να δει το κατάλογο αισθητήρων.

Β0β: Η εφαρμογή εμφανίζει το κατάλογο αισθητήρων.

Εναλλακτικές ροές: καμία.

2.1.6.13 Επεξεργασία αισθητήρα

Περίπτωση χρήσης: Επεξεργασία αισθητήρα

Ταυτότητα: uc.sensor.edit

Δράστης: Χρήστης

Αρχικές συνθήκες: Η κατάσταση του χρήστη είναι «συνδεδεμένος».

Βασική ροή:

Β0α: Ο δράστης ζητά να επεξεργαστεί τα δεδομένα ενός αισθητήρα.

Β0β: Η εφαρμογή

Β0β1: εμφανίζει τα υπάρχοντα δεδομένα του αισθητήρα στα πεδία και

Β0β2: ζητά από τον δράστη να τα αλλάξει και να τα καταχωρήσει.

Β1α: Ο δράστης αλλάζει τα δεδομένα του αισθητήρα και τα καταχωρεί.

Β1β: Η εφαρμογή

Β1β1: ελέγχει τα δεδομένα και εφόσον είναι έγκυρα

Β1β2: καταχωρεί τα νέα δεδομένα και

Β1β3: εμφανίζει τον κατάλογο αισθητήρων.

Εναλλακτικές ροές:

Ε1: Αν στο βήμα Β1β1 τα δεδομένα δεν είναι έγκυρα, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β0β.

2.1.6.14 Διαγραφή αισθητήρα

Περίπτωση χρήσης: Διαγραφή αισθητήρα

Ταυτότητα: uc.sensor.delete

Δράστης: Χρήστης

Αρχικές συνθήκες: Η κατάσταση του χρήστη είναι «συνδεδεμένος».

Βασική ροή:

Β0α: Ο δράστης ζητά να διαγράψει ένα αισθητήρα.

Β0α: Η εφαρμογή

Β0α1: διαγράφει τον αισθητήρα και

Β0α2: εμφανίζει τον κατάλογο των αιοσθητήρων

Εναλλακτικές ροές: καμία

2.1.6.15 Προσθήκη νέου κλειδιού API

Περίπτωση χρήσης: Προσθήκη νέου κλειδιού API

Ταυτότητα: uc.apikey.add

Δράστης: Χρήστης

Αρχικές συνθήκες: Η κατάσταση του χρήστη είναι «συνδεδεμένος».

Βασική ροή:

Β0α: Ο δράστης ζητά να προσθέσει ένα νέο κλειδί API.

Β0β: Η εφαρμογή ζητά από τον δράστη να καταχωρήσει τα δεδομένα του κλειδιού API

Β1α: Ο δράστης δίνει τα δεδομένα του κλειδιού API.

Β1β: Η εφαρμογή

Β1β1: ελέγχει τα δεδομένα και εφόσον είναι έγκυρα

Β1β2: καταχωρεί το νέο κλειδί API

Β1β3: εμφανίζει τον κατάλογο κλειδιών API.

Εναλλακτικές ροές:

Ε1: Αν στο βήμα Β1β1 τα δεδομένα δεν είναι έγκυρα, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β0β.

2.1.6.16 Εμφάνιση του καταλόγου κλειδιών API

Περίπτωση χρήσης: Εμφάνιση του καταλόγου κλειδιών API

Ταυτότητα: uc.apikey.displayAll

Δράστης: Χρήστης

Αρχικές συνθήκες: Η κατάσταση του χρήστη είναι «συνδεδεμένος».

Βασική ροή:

Β0α: Ο δράστης ζητά να δει το κατάλογο κλειδιών API.

Β0β: Η εφαρμογή εμφανίζει το κατάλογο κλειδιών API.

Εναλλακτικές ροές: καμία.

2.1.6.17 Επεξεργασία κλειδιού API

Περίπτωση χρήσης: Επεξεργασία κλειδιού API

Ταυτότητα: uc.apikey.edit

Δράστης: Χρήστης

Αρχικές συνθήκες: Η κατάσταση του χρήστη είναι «συνδεδεμένος».

Βασική ροή:

Β0α: Ο δράστης ζητά να επεξεργαστεί τα δεδομένα ενός κλειδιού API.

Β0β: Η εφαρμογή

Β0β1: εμφανίζει τα υπάρχοντα δεδομένα του κλειδιού API στα πεδία και

Β0β2: ζητά από τον δράστη να τα αλλάξει και να τα καταχωρήσει.

Β1α: Ο δράστης αλλάζει τα δεδομένα του κλειδιού API και τα καταχωρεί.

Β1β: Η εφαρμογή

Β1β1: ελέγχει τα δεδομένα και εφόσον είναι έγκυρα

Β1β2: καταχωρεί τα νέα δεδομένα και

Β1β3: εμφανίζει τον κατάλογο κλειδιών API..

Εναλλακτικές ροές:

Ε1: Αν στο βήμα Β1β1 τα δεδομένα δεν είναι έγκυρα, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β0β.

2.1.6.18 Διαγραφή κλειδιού API

Περίπτωση χρήσης: Διαγραφή κλειδιού API

Ταυτότητα: uc.apikey.delete

Δράστης: Χρήστης

Αρχικές συνθήκες: Η κατάσταση του χρήστη είναι «συνδεδεμένος».

Βασική ροή:

Β0α: Ο δράστης ζητά να διαγράψει ένα κλειδί API.

Β0α: Η εφαρμογή

Β0α1: διαγράφει το κλειδί API και

Β0α2: εμφανίζει τον κατάλογο των κλειδιών API.

Εναλλακτικές ροές: καμία

2.1.6.19 Προσθήκη μετρήσεων συσκευής

Περίπτωση χρήσης: Προσθήκη μετρήσεων συσκευής

Ταυτότητα: uc.measurement.add

Δράστης: Συσκευή

Αρχικές συνθήκες:

Α0: Η συσκευή έχει πρόσβαση στο διαδίκτυο.

Α1: Έχει εγγραφτεί ο χρήστης της συσκευής και

Α1β1: έχει καταχωρήσει τη συσκευή και

Α1β2: έχει καταχωρήσει αισθητήρες στη συσκευή και

Α1β3: έχει δημιουργήσει ένα τουλάχιστον κλειδί API για τη συσκευή με δικαιώματα εγγραφής.

Βασική ροή:

Β0α: Η συσκευή ζητά να καταχωρήσει στην εφαρμογή μία μέτρηση.

Β0β: Η εφαρμογή

Β1β1: ελέγχει τα δεδομένα και εφόσον είναι έγκυρα

Β1β2: καταχωρεί τη νέα μέτρηση

Εναλλακτικές ροές:

Ε1: Αν στο βήμα Β1β1 τα δεδομένα δεν είναι έγκυρα, η εφαρμογή επιστρέφει μήνυμα λάθους.

2.1.6.20 Προβολή μετρήσεων συσκευής

Περίπτωση χρήσης: Προβολή μετρήσεων συσκευής

Ταυτότητα: uc.measurement.display

Δράστης: Χρήστης

Αρχικές συνθήκες: Η κατάσταση του χρήστη είναι «συνδεδεμένος».

Βασική ροή:

Β0α: Ο δράστης ζητά να εμφανιστεί η σελίδα μιας συσκευής.

Β0β: Η εφαρμογή εμφανίζει τη σελίδα και

Β0β1: δημιουργεί ένα νέο γράφημα για κάθε διαθέσιμο τύπο αισθητήρα της συσκευής

Β0β2: εμφανίζει τις τελευταίες 10 τιμές για κάθε αισθητήρα και προβάλει κάθε αισθητήρα του ίδιου τύπου με διαφορετικό χρώμα.

Τελική συνθήκη μετά τη βασική ροή: Επαναφόρτωση του βήματος Β0β1 και Β0β2 μετά από 2 δευτερόλεπτα.

Εναλλακτικές ροές: καμία.

2.1.6.21 Περιπτώσεις χρήσης διαχειριστή

Οι περιπτώσεις χρήσεις του διαχειριστή είναι όμοιες με τις παραπάνω που αφορούν τον χρήστη οπότε δε θα γίνει περαιτέρω ανάλυση σε αυτή την εργασία. Η κύρια διαφορά είναι ότι ο διαχειριστής έχει πρόσβαση σε καταλόγους όλων των πόρων όλων των χρηστών και φυσικά έχει όλα τα δικαιώματα (επεξεργασίας/διαγραφής).

2.2 Αρχιτεκτονική

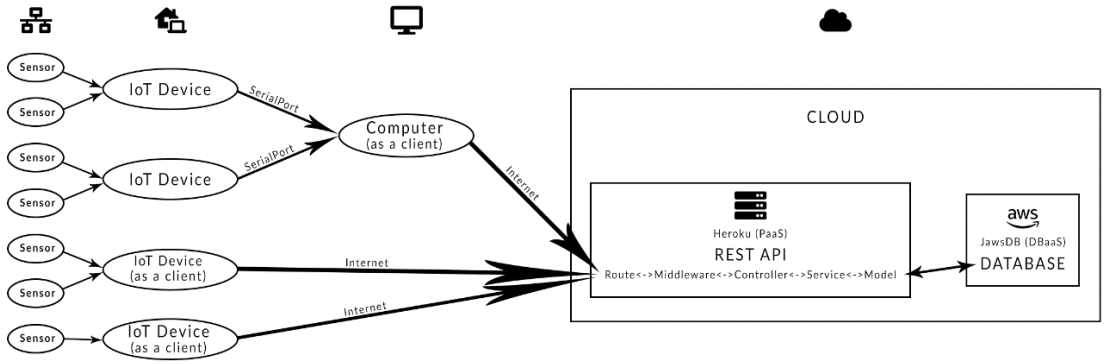
Από τις απαιτήσεις προκύπτουν τα μέρη που περιλαμβάνει η εφαρμογή:

- Διεπαφή χρήστη
- Επιχειρησιακή λογική
- Μοντέλο δεδομένων

Τα τρία αυτά μέρη αντιστοιχούν στα επίπεδα του μοντέλου αρχιτεκτονικής λογισμικού MVC, δηλαδή Model (Μοντέλο) για το μοντέλο δεδομένων, View (Προβολή) για τη διεπαφή χρήστη, Controller (Ελεγκτής) για την επιχειρησιακή λογική. Το MVC χρησιμοποιείται για τη δημιουργία περιβαλλόντων αλληλεπίδρασης χρήστη. Στο μοντέλο αυτό η εφαρμογή διαιρείται σε τρία διασυνδεδεμένα μέρη ώστε να διαχωριστεί η παρουσίαση της πληροφορίας στον χρήστη από την μορφή που έχει αποθηκευτεί στο σύστημα. Το κύριο μέρος του μοντέλου είναι το αντικείμενο Model το οποίο διαχειρίζεται την ανάκτηση/αποθήκευση των δεδομένων στο σύστημα. Το αντικείμενο View χρησιμοποιείται μόνο για να παρουσιάζεται η πληροφορία στον χρήστη (π.χ. με γραφικό τρόπο). Το τρίτο μέρος είναι ο Controller ο οποίος δέχεται την είσοδο και στέλνει εντολές στο αντικείμενο Model και στο View. [17]

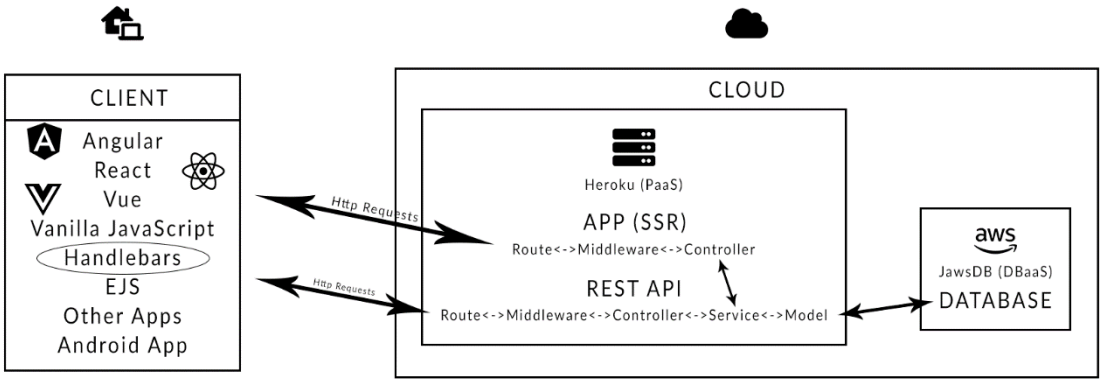
Για την **επιχειρησιακή λογική** κρίθηκε ως καλή πρακτική η χρήση του αρχιτεκτονικού στυλ REST το οποίο βασίζεται στην αρχιτεκτονική MVC. Πιο συγκεκριμένα την υλοποίηση ενός REST-API (Back-end) σε διακομιστή/περιβάλλον Node.js για την διαφοροποίησή της επιχειρησιακής λογικής από τα υπόλοιπα μέρη. Το πλεονέκτημα από αυτό είναι η εύκολη συντήρηση του κώδικα, η δυνατότητα χρήσης της επιχειρησιακής λογικής από άλλα λογισμικά, η δυνατότητα χρήσης πολλών τεχνολογιών για τη διεπαφή χρήστη (Front-end) και η άμεση επικοινωνία των συσκευών (IoT Device) με την επιχειρησιακή λογική. Επίσης μπορεί να γίνει ξεχωριστά η εγκατάσταση/δημοσίευση του διακομιστή REST-API από τη διεπαφή, το οποίο βοηθά στην μείωση του φόρτου και παρέχει τη δυνατότητα αναβάθμισής των πόρων (scaling) του διακομιστή μελλοντικά.

Για το **μοντέλο δεδομένων** χρησιμοποιήθηκε ένας διακομιστής MySQL μιας και είναι ιδανικός για σχεσιακές βάσεις δεδομένων.



Εικόνα 2 Αρχιτεκτονική (Εφαρμογής-IoT Συσκευών)

Για τη **διεπαφή χρήστη** (Front-end) οι επιλογές ήταν πολλές. Έπρεπε να γίνει επιλογή ανάμεσα σε Server-Side Rendering (SSR) και Client-Side Rendering (CSR) και να υπολογισθούν η χρησιμότητα κάθε τεχνολογίας, τα πλεονεκτήματα κάθε μίας αλλά και ο χρόνος ανάπτυξης. Η τάση στην εποχή μας είναι τα Single-Page Apps ή αλλιώς SPAs εν συντομία στα οποία η σελίδα HTML δημιουργείται δυναμικά στον πελάτη Client, δηλαδή CSR. Τεχνολογίες που χρησιμοποιούνται για την υλοποίηση SPA είναι Angular, React, Vue, σκέτη (vanilla) JavaScript κλπ. Αφ’ ετέρου η υλοποίηση μίας διεπαφής SSR είναι πιο εύκολη και γρήγορη καθώς μπορεί να γίνει άμεση χρήση των υπηρεσιών και λειτουργιών του REST-API αλλά και η χρήση των πόρων συστήματος μπορεί να είναι κοινή. Τεχνολογίες που χρησιμοποιούνται για SSR με Node.js είναι Handlebars, JES, Pug κλπ. Παρόλα αυτά, για λόγους εξοικείωσης, εξοικονόμησης χρόνου και πόρων έγινε χρήση Handlebars για SSR κυρίως για τη κατασκευή του πρότυπου εμφάνισης, των διαδρομών και των εκλεκτών των προβολών ενώ για την ανάκτηση δεδομένων έγινε χρήση της βιβλιοθήκης Axios στον Client σε JavaScript όπως γίνεται συχνά σε υλοποιήσεις με React. Το αποτέλεσμα από αυτό είναι ότι στέλνεται η σελίδα με το αρχικό πρότυπο στον πελάτη Client συμπεριλαμβάνοντας πολύ ελάχιστες πληροφορίες σχετικά με τον χρήστη και τα υπόλοιπα δεδομένα της σελίδας ενημερώνονται μετά από τον Client μέσω των request που γίνονται στο REST-API.



Εικόνα 3 Αρχιτεκτονική (Εφαρμογής-Client)

2.3 Βάση δεδομένων

2.3.1 Ανάλυση και σχεδιασμός

Από τις απαιτήσεις προκύπτουν οι παρακάτω **οντότητες** με τα δεδομένα τους (δεν αναφέρονται ξανά εδώ για λόγους συντομίας):

- Χρήστης
- Ρόλος χρήστη
- Συσκευή
- Τύπος συσκευής
- Αισθητήρας
- Τύπος αισθητήρα
- Κλειδί API
- Δικαιώματα κλειδιού API
- Μέτρηση
- Δεδομένο μέτρησης
- Συνεδρία

Και οι **συσχετίσεις** τους αντίστοιχα:

- Κάθε χρήστης έχει ένα ρόλο (user ή admin). Κάθε ρόλος ανήκει σε πολλούς χρήστες.

- Κάθε συσκευή ανήκει σε ένα χρήστη. Κάθε χρήστης έχει πολλές συσκευές.
- Κάθε συσκευή έχει ένα τύπο. Κάθε τύπος συσκευής ανήκει σε πολλές συσκευές.

- Κάθε αισθητήρας ανήκει σε ένα χρήστη. Κάθε χρήστης έχει πολλούς αισθητήρες.
- Κάθε αισθητήρας ανήκει σε μία συσκευή. Κάθε συσκευή έχει πολλούς αισθητήρες.
- Κάθε αισθητήρας έχει ένα τύπο. Κάθε τύπος αισθητήρα ανήκει σε πολλούς αισθητήρες.

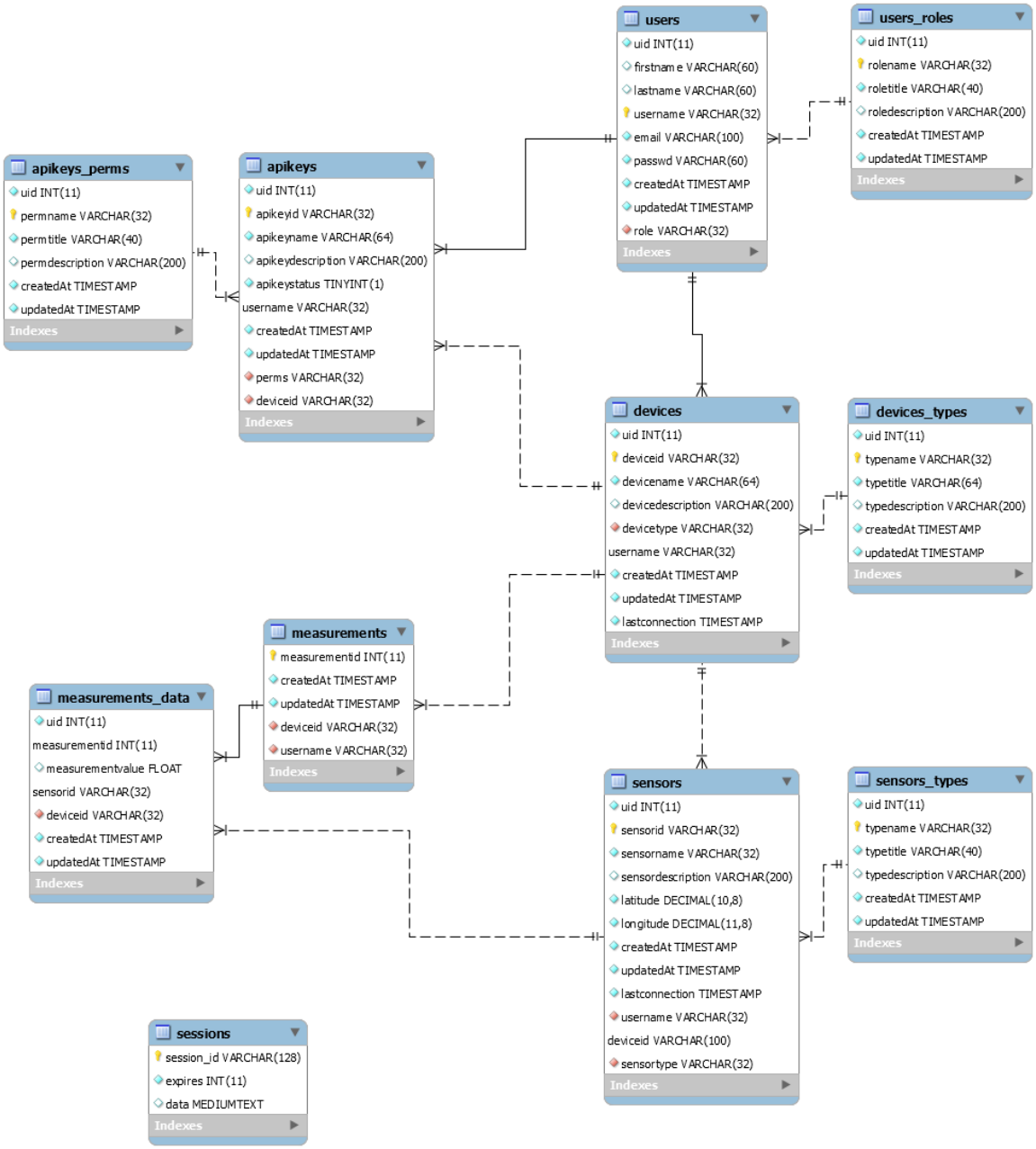
- Κάθε κλειδί API ανήκει σε ένα χρήστη. Κάθε χρήστης έχει πολλά κλειδιά API.
- Κάθε κλειδί API ανήκει σε μία συσκευή. Κάθε συσκευή έχει πολλά κλειδιά API.

- Κάθε κλειδί API έχει μία επιλογή δικαιωμάτων. Κάθε επιλογή δικαιωμάτων ανήκει σε πολλά κλειδιά API.

- Κάθε μέτρηση ανήκει σε ένα χρήστη. Κάθε χρήστης έχει πολλές μετρήσεις.
- Κάθε μέτρηση ανήκει σε μια συσκευή. Κάθε συσκευή έχει πολλές μετρήσεις.

- Κάθε δεδομένο μέτρησης ανήκει σε μία μέτρηση. Κάθε μέτρηση έχει πολλά δεδομένα μέτρησης.
- Κάθε δεδομένο μέτρησης ανήκει σε μία συσκευή. Κάθε συσκευή έχει πολλά δεδομένα μέτρησης.
- Κάθε δεδομένο μέτρησης ανήκει σε ένα αισθητήρα. Κάθε αισθητήρας έχει πολλά δεδομένα μέτρησης.

- Κάθε χρήστης ανήκει σε μία συνεδρία. Κάθε συνεδρία έχει ένα χρήστη.



Εικόνα 4 Σχήμα βάσης δεδομένων(Διάγραμμα EER)

2.3.2 Υλοποίηση

Ο παρακάτω κώδικας SQL περιέχει εντολές που δημιουργούν το σχήμα (schema) στη βάση δεδομένων.

```
/*
* Name: users_roles
* Description: User roles
*/
CREATE TABLE users_roles (
    uid INT(11) NOT NULL UNIQUE AUTO_INCREMENT,
    rolename VARCHAR(32) NOT NULL UNIQUE,
    roletitle VARCHAR(40) NOT NULL,
    roledescription VARCHAR(200),
    createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTA
MP,
    updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTA
MP,
    PRIMARY KEY (rolename)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

/*
* Name: users
* Description: Entity table of Users
* Comment: Only the username is stored for security reasons.
*          Each device is authenticated via an api key, the existance of a
matching api key and by the existance and match of their devices.
*/
CREATE TABLE users (
    uid INT(11) NOT NULL UNIQUE AUTO_INCREMENT,
    firstname VARCHAR(60),
    lastname VARCHAR(60),
    username VARCHAR(32) NOT NULL UNIQUE,
    email VARCHAR(100) NOT NULL UNIQUE,
    passwd VARCHAR(60) NOT NULL,
    createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTA
MP,
    updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTA
MP,
    role VARCHAR(32) NOT NULL DEFAULT 'user',
    PRIMARY KEY (username),
    FOREIGN KEY (role) REFERENCES users_roles(rolename)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

/*
```

```
* Name: devices_types
* Description: Device types with their descriptions
*/
CREATE TABLE devices_types (
    uid INT(11) NOT NULL UNIQUE AUTO_INCREMENT,
    typename VARCHAR(32) NOT NULL UNIQUE,
    typetitle VARCHAR(64) NOT NULL,
    typedescription VARCHAR(200),
    createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTA
MP,
    updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTA
MP,
    PRIMARY KEY (typename)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

/*
* Name: devices
* Description: Entity table of devices
*/
CREATE TABLE devices (
    uid INT(11) NOT NULL UNIQUE AUTO_INCREMENT,
    deviceid VARCHAR(32) NOT NULL,
    devicename VARCHAR(64) NOT NULL,
    devicedescription VARCHAR(200),
    devicetype VARCHAR(32) NOT NULL DEFAULT 'http_device',
    username VARCHAR(32) NOT NULL,
    createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTA
MP,
    updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTA
MP,
    lastconnection TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TI
MESTAMP,
    PRIMARY KEY (username, deviceid),
    FOREIGN KEY (username) REFERENCES users(username)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (devicetype) REFERENCES devices_types(typename)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

/*
* Name: apikeys_perms
* Description: apikeys_perms
*/
CREATE TABLE apikeys_perms (
    uid INT(11) NOT NULL UNIQUE AUTO_INCREMENT,
    permname VARCHAR(32) NOT NULL UNIQUE,
    permtitle VARCHAR(40) NOT NULL,
```

```
    permdescription VARCHAR(200),
    createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTA
MP,
    updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTA
MP,
    PRIMARY KEY (permname)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

/*
* Name: apikeys
* Description: Table that stores access apikeys for users and devices
* Comment: JWT apikey contains apikeyid and username
*/
CREATE TABLE apikeys (
    uid INT(11) NOT NULL UNIQUE AUTO_INCREMENT,
    apikeyid VARCHAR(32) NOT NULL,
    apikeyname VARCHAR(64) NOT NULL,
    apikeydescription VARCHAR(200),
    apikeystatus TINYINT(1) NOT NULL DEFAULT 1,
    apikey VARCHAR(60) NOT NULL UNIQUE,
    username VARCHAR(32) NOT NULL,
    createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTA
MP,
    updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTA
MP,
    PRIMARY KEY (username, apikeyid),
    FOREIGN KEY (username) REFERENCES users(username)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    perms VARCHAR(32) NOT NULL DEFAULT 'all',
    FOREIGN KEY (perms) REFERENCES apikeys_perms(permname)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    deviceid VARCHAR(32) NOT NULL,
    FOREIGN KEY (username, deviceid) REFERENCES devices(username, deviceid)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

/*
* Name: sensors_types
* Description: Sensor types with their descriptions
*/
CREATE TABLE sensors_types (
    uid INT(11) NOT NULL UNIQUE AUTO_INCREMENT,
    typename VARCHAR(32) NOT NULL UNIQUE,
    typetitle VARCHAR(40) NOT NULL,
    typedescription VARCHAR(200),
```

```
    createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTA
MP,
    updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTA
MP,
    PRIMARY KEY (typename)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

/*
* Name: sensors
* Description: Entity table of sensors
*/
CREATE TABLE sensors (
    uid INT(11) NOT NULL UNIQUE AUTO_INCREMENT,
    sensorid VARCHAR(32) NOT NULL,
    sensorname VARCHAR(32) NOT NULL,
    sensordescription VARCHAR(200),
    latitude DECIMAL(10, 8) NOT NULL DEFAULT '0',
    longitude DECIMAL(11, 8) NOT NULL DEFAULT '0',
    createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTA
MP,
    updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTA
MP,
    lastconnection TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TI
MESTAMP,
    username VARCHAR(32) NOT NULL,
    deviceid VARCHAR(100) NOT NULL,
    sensortype VARCHAR(32) NOT NULL,
    PRIMARY KEY (deviceid, sensorid),
    FOREIGN KEY (username, deviceid) REFERENCES devices(username, deviceid)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (sensortype) REFERENCES sensors_types(typename)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

/*
* Name: measurements
* Description: Table that stores the measurements sent by the devices. Sets
 a unique id and a timestamp for each device id and username combination
*/
CREATE TABLE measurements (
    measurementid INT(11) NOT NULL UNIQUE AUTO_INCREMENT,
    createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTA
MP,
    updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTA
MP,
    PRIMARY KEY (measurementid),
    UNIQUE KEY (measurementid, createdAt),
```

```
    deviceid VARCHAR(32) NOT NULL,
    username VARCHAR(32) NOT NULL,
    FOREIGN KEY (username, deviceid) REFERENCES devices(username, deviceid)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

/*
* Name: measurements_data
* Description: Table that stores the measurement values sent by the devices
. Sets a unique id and a for each measurement and sensor id combination
*/
CREATE TABLE measurements_data (
    uid INT(11) NOT NULL UNIQUE AUTO_INCREMENT,
    measurementid INT(11) NOT NULL,
    measurementvalue FLOAT NOT NULL DEFAULT 0,
    sensorid VARCHAR(32) NOT NULL,
    deviceid VARCHAR(32) NOT NULL,
    createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTA
MP,
    updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTA
MP,
    PRIMARY KEY (measurementid, sensorid),
    FOREIGN KEY (measurementid) REFERENCES measurements(measurementid)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (deviceid, sensorid) REFERENCES sensors(deviceid, sensorid)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

2.4 Εργαλεία και τεχνολογίες

2.4.1 Εργαλεία ανάπτυξης λογισμικού

Για την ανάπτυξη της εφαρμογής χρησιμοποιήθηκαν τα παρακάτω εργαλεία:

Visual Studio Code – [18] Είναι ένας βελτιωμένος επεξεργαστής κώδικα (IDE) με υποστήριξη λειτουργιών ανάπτυξης όπως ο εντοπισμός σφαλμάτων, η εκτέλεση εργασιών και ο έλεγχος εκδόσεων. Χρησιμοποιήθηκε για την ανάπτυξη και την επεξεργασία κώδικα.

Git Bash [19] – Είναι μια εφαρμογή (κονσόλα) για περιβάλλοντα Microsoft Windows που υποστηρίζει εντολές Git για τη διαχείριση τοπικών αποθετηρίων δεδομένων και επικοινωνία με αποθετήρια όπως το GitHub. Χρησιμοποιήθηκε ως κύρια κονσόλα/γραμμή εντολών.

npm [20] – Το npm, συντομογραφία του Node Package Manager είναι ένα διαδικτυακό αποθετήριο για τη δημοσίευση έργων ανοιχτού κώδικα, για το Node.js. Επίσης είναι ένα βοηθητικό πρόγραμμα γραμμής εντολών με το εν λόγω αποθετήριο που βοηθά στην εγκατάσταση πακέτων, στη διαχείριση εκδόσεων και στη διαχείριση εξαρτήσεων. Μία πληθώρα βιβλιοθηκών και εφαρμογών Node.js δημοσιεύονται στο npm καθημερινά τα οποία μπορούν να εγκατασταθούν με μία εντολή στη γραμμή εντολών. Χρησιμοποιήθηκε για την εγκατάσταση βιβλιοθηκών και εργαλείων.

Node.js [21] - Είναι μια πλατφόρμα ανάπτυξης λογισμικού (κυρίως διακομιστών) χτισμένη σε περιβάλλον JavaScript. Στόχος του Node είναι να παρέχει ένα εύκολο τρόπο δημιουργίας κλιμακωτών διαδικτυακών εφαρμογών. Σε αντίθεση από τα περισσότερα σύγχρονα περιβάλλοντα ανάπτυξης εφαρμογών δικτύων μία διεργασία node δεν στηρίζεται στην πολυνηματικότητα αλλά σε ένα μοντέλο ασύγχρονης επικοινωνίας εισόδου/εξόδου. Χρησιμοποιήθηκε ως διακομιστής της εφαρμογής.

Google Chrome [22] - Είναι πρόγραμμα περιήγησης στο Διαδίκτυο που αναπτύσσεται από την Google και χρησιμοποιεί τη μηχανή απεικόνισης WebKit. Χρησιμοποιήθηκε ως περιηγητής αλλά και για τα εργαλεία ανάπτυξης που προσφέρει.

Chrome DevTools [23] – Είναι ένα σύνολο εργαλείων για προγραμματιστές διαδικτύου, ενσωματωμένα στον περιηγητή Google Chrome. Χρησιμοποιήθηκε για την εύρεση και επίλυση σφαλμάτων του Front-end.

Postman [24] – Είναι ένας API Client που χρησιμοποιείται για τη δημιουργία, δοκιμή και καταγραφή API. Χρησιμοποιήθηκε για την υλοποίηση και τον έλεγχο των διαδρομών της εφαρμογής προσομοιώνοντας HTTP requests.

HeidiSQL [25] – Είναι ένα δωρεάν εργαλείο ανοιχτού κώδικα για τη διαχείριση βάσεων δεδομένων MySQL. Χρησιμοποιήθηκε για την τελική δημοσίευση του σχήματος της βάσης δεδομένων.

2.4.2 Τεχνολογίες – Βιβλιοθήκες

Εργαλεία, ή αλλιώς βιβλιοθήκες (libraries), ή εξαρτήσεις (dependencies) ή πακέτα (packages) ή μονάδες (modules) είναι σύνολα λειτουργιών και μεθόδων που συμπεριλαμβάνονται σε μια εφαρμογή. Μερικά από αυτά που χρησιμοποιήθηκαν στην ανάπτυξη του Back-end και του Front-end είναι τα παρακάτω. Σημείωση: Η αναφορά τους δε χρειάζεται να γίνει ξεχωριστά μιας και οι δύο πλευρές (τα δύο stack όπως συνηθίζεται να λέγονται) είναι υλοποιημένες σε JavaScript και τα περισσότερα εργαλεία είναι συμβατά και με τις δύο.

Express.js [26] – ή απλά Express είναι ένα framework για το Node.js που περιέχει ένα ισχυρό σύνολο εργαλείων για τη κατασκευή εφαρμογών διαδικτύου, κινητών και APIs. Παρέχει εργαλεία για τη κατασκευή διαδρομών (routes), επιτρέπει τη χρήση ενδιάμεσου λογισμικού (middleware) σε HTTP requests κλπ.

Η εγκατάσταση έγινε από τη γραμμή εντολών μέσω του NPM στον φάκελο του project:

```
$ npm install express --save
```

Express-Validator [27] - Το Express-Validator είναι ένα σύνολο μεσαίων λογισμικών του Express.js που χρησιμοποιεί λειτουργίες validator και sanitizer του validator.js μια βιβλιοθήκη που παρέχει λειτουργίες επικύρωσης δεδομένων.

Η εγκατάσταση έγινε από τη γραμμή εντολών μέσω του NPM στον φάκελο του project:

```
$ npm install express-validator --save
```

Express-Session [28] - Το Express-Session είναι ένα σύνολο μεσαίων λογισμικών του Express.js που χρησιμοποιείτε για τη δημιουργία και διαχείριση συνεδριών (sessions).

Η εγκατάσταση έγινε από τη γραμμή εντολών μέσω του NPM στον φάκελο του project:

```
$ npm install express-session --save
```

Express-MySQL-Session [29] – Χρησιμοποιείται από το Express-Session για την αποθήκευση συνεδριών σε βάση δεδομένων MySQL.

Η εγκατάσταση έγινε από τη γραμμή εντολών μέσω του NPM στον φάκελο του project:

```
$ npm install express-mysql-session --save
```

Express-JWT [30] – Το Express-JWT παρέχει στο Express.js ενδιάμεσο λογισμικό για την επικύρωση κλειδιών JWT (JSON Web Tokens) μέσω της βιβλιοθήκης jsonwebtoken. Επίσης παρέχει τη δυνατότητα αποθήκευσης των δεδομένων του αποκωδικοποιημένου κλειδιού σε ένα αντικείμενο.

Η εγκατάσταση έγινε από τη γραμμή εντολών μέσω του NPM στον φάκελο του project:

```
$ npm install express-jwt --save
```

Sequelize ORM – Το Object Relational Mapping (ORM) [31] είναι μια τεχνική που χαρτογραφεί τα αντικείμενα του λογισμικού σε πίνακες της βάσης δεδομένων. Η Sequelize ORM [32] είναι μια αρκετά δημοφιλής βιβλιοθήκη και χρησιμοποιείται παράλληλα με το Node.js. Είναι promise-based, υποστηρίζει σχεσιακές βάσεις δεδομένων και διαλέκτους όπως PostgreSQL, MySQL, MariaDB, SQLite και MSSQL. Μπορεί να χρησιμοποιηθεί

ακόμη και για την απευθείας δημιουργία εντολών (queries). Το πλεονέκτημα από τη χρήση ενός τέτοιου ORM είναι αρχικά η γρήγορη δημιουργία query χωρίς να χρειάζεται να γνωρίζει ο προγραμματιστής τη διάλεκτο της βάσης δεδομένων και σε δεύτερο βαθμό η προστασία από επιθέσεις SQL Injection.

Η εγκατάσταση έγινε από τη γραμμή εντολών μέσω του NPM στον φάκελο του project:

```
$ npm install sequelize --save
```

Sequelize-Auto [33] – Παραδοσιακά, στη Sequelize ORM ο προγραμματιστής πρέπει να κατασκευάσει το μοντέλο της κάθε οντότητας της βάσης δεδομένων. Πρόκειται για μια σχετικά χρονοβόρα διαδικασία και απαιτεί εξοικείωσή με τους ορισμούς μοντέλων της Sequelize ORM. Ιδιαίτερα, όταν υπάρχει μια ήδη έτοιμη βάση δεδομένων ή για ορισμένους λόγους η βάση δεδομένων δεν χρησιμοποιείται απόλυτα από ένα σύστημα ή πραγματοποιούνται αλλαγές κατά τη διάρκεια της ανάπτυξης της εφαρμογής απευθείας στη βάση δεδομένων, είναι επιθυμητό τα μοντέλα οντοτήτων να ενημερώνονται γρήγορα. Η λύση σε αυτό είναι η βιβλιοθήκη Sequelize-Auto με την οποία μπορούμε να κατασκευάσουμε αυτόματα τα μοντέλα οντοτήτων με βάση τη σχεσιακή βάση δεδομένων που υπάρχει ήδη. Η Sequelize-Auto υποστηρίζει διαλέκτους MySQL/MariaDB, Postgres, Sqlite, MSSQL και η χρήση της γίνεται μέσω της γραμμής εντολών με την εντολή `sequelize-auto` η οποία δέχεται σαν ορίσματα τη ζωντανή διεύθυνση της βάσης δεδομένων, το όνομά της, το όνομα χρήστη, port, τοποθεσία δημιουργίας των μοντέλων, τύπο διαλέκτου της βάσης και τη γλώσσα/τύπο μοντέλων.

Η εγκατάσταση έγινε από τη γραμμή εντολών μέσω του NPM στον φάκελο του project:

```
$ npm install sequelize-auto --save
```

JSONWebToken – Είναι μια βιβλιοθήκη που υλοποιεί λειτουργίες ανοιχτού πρότυπου (RFC 7519) JSON Web Token (JWT) [34]. Με τη χρήση του JWT, πληροφορίες που αφορούν ένα χρήστη αποθηκεύονται σε ένα string, επίσης γνωστό ως κλειδί JWT Token, μπορούν να πιστοποιηθούν καθώς υπογράφονται ψηφιακά και κρυπτογραφούνται με ένα κλειδί δημόσιο ή ιδιωτικό με RSA, ECDSA ή HMAC. Μαζί με τις πληροφορίες αυτές αποθηκεύονται οι ημερομηνίες δημιουργίας και τερματισμού που ορίζουν τον χρόνο ζωής τους, κάτι που είναι πολύ χρήσιμο για ένα Stateless REST API καθώς δεν υπάρχει συνεδρία

(session) για να αποθηκευτούν κάπου προσωρινά αυτές οι πληροφορίες. Το JWT δημιουργείται από το REST API και αποθηκεύεται στο Front-end όπου αυτό έχει ορίσει.

Η εγκατάσταση έγινε από τη γραμμή εντολών μέσω του NPM στον φάκελο του project:

```
$ npm install jsonwebtoken --save
```

Bcrypt [35] – Είναι μια βιβλιοθήκη που βοηθά στο hashing ή αλλιώς κατακερματισμό κωδικών.

Η εγκατάσταση έγινε από τη γραμμή εντολών μέσω του NPM στον φάκελο του project:

```
$ npm install bcrypt --save
```

Morgan [36] – Είναι μια βιβλιοθήκη που χρησιμοποιείται για την καταγραφή request στη κονσόλα.

Η εγκατάσταση έγινε από τη γραμμή εντολών μέσω του NPM στον φάκελο του project:

```
$ npm install morgan
```

Nodemon [37] – Είναι ένα εργαλείο για Node.js που κάνει αυτόματα επανεκκίνηση της εφαρμογής όταν γίνουν αλλαγές σε αρχεία της.

Η εγκατάσταση έγινε από τη γραμμή εντολών μέσω του NPM στον φάκελο του project:

```
$ npm install nodemon
```

Dotenv [38] – Είναι ένα εργαλείο χωρίς εξαρτήσεις που φορτώνει μεταβλητές περιβάλλοντος από ένα αρχείο .env εντός του φακέλου της εφαρμογής.
-Η εγκατάσταση έγινε από τη γραμμή εντολών μέσω του NPM στον φάκελο του project:

```
$ npm install dotenv --save
```

Cors [39] – Είναι μια βιβλιοθήκη που παρέχει στο Express.js ενδιάμεσο λογισμικό για τη χρήση CORS με διάφορες επιλογές.

Η εγκατάσταση έγινε από τη γραμμή εντολών μέσω του NPM στον φάκελο του project:

```
$ npm install cors --save
```

Body-Parser [40] **-** Είναι μια βιβλιοθήκη για το Node.js που μεταφέρει τα σώματα (bodies) των εισερχόμενων request σε ενδιάμεσο λογισμικό μέσα στην ιδιότητα req.body.

Η εγκατάσταση έγινε από τη γραμμή εντολών μέσω του NPM στον φάκελο του project:

```
$ npm install body-parser --save
```

Cookie-Parser [41] **-** Είναι μια βιβλιοθήκη για το Node.js που μεταφέρει τις επικεφαλίδες (headers) των Cookie σε ενδιάμεσο λογισμικό μέσα στην ιδιότητα req.cookies. Χρησιμοποιήθηκε στο REST-API για την ανάγνωση JWT Token από το Front-end.

Η εγκατάσταση έγινε από τη γραμμή εντολών μέσω του NPM στον φάκελο του project:

```
$ npm install cookie-parser --save
```

Hbs [42] **-** Είναι μια βιβλιοθήκη για το Express.js που επιτρέπει τη χρήση του handlebars.js σαν view engine. Χρησιμοποιήθηκε για τη δημιουργία SSR προβολών.

Η εγκατάσταση έγινε από τη γραμμή εντολών μέσω του NPM στον φάκελο του project:

```
$ npm install hbs --save
```

Passport [43] **-** Είναι ένα ενδιάμεσο λογισμικό συμβατό με το Express.js που προσθέτει λειτουργίες αυθεντικοποίησης.

Η εγκατάσταση έγινε από τη γραμμή εντολών μέσω του NPM στον φάκελο του project:

```
$ npm install passport --save
```

AdminLTE [44] – Είναι ένα δημοφιλές πρότυπο (responsive html template) ανοιχτού κώδικα για τη κατασκευή πινάκων διαχείρισης, βασισμένο σε Bootstrap 3 και χρησιμοποιεί όλα τα εξαρτήματα του Bootstrap καθώς και πρόσθετα όπως jQuery και Chart.js.

Axios [45] – Είναι ένας promise based HTTP client για περιηγητές και node.js. Χρησιμοποιήθηκε στο Front-end για την ανάκτηση δεδομένων από το REST-API.

Chart.js [46] – Είναι μια JavaScript βιβλιοθήκη ανοιχτού κώδικα που χρησιμοποιείται για την οπτικοποίηση δεδομένων με μορφή γραφιμάτων.

2.5 Ανάλυση Back-end (REST API)

Το REST (Representational State Transfer) είναι ένα αρχιτεκτονικό στυλ που παρουσιάστηκε για πρώτη φορά από τον Roy Fielding το 2000. [47] Αποτελεί ένα σύνολο από αρχές σχεδίασης και βασίζεται σε αρχιτεκτονικά πρότυπα διαδικτύου. Επικεντρώνεται στους πόρους (π.χ. δεδομένα) ενός συστήματος και η μεταβολή της κατάστασης των πόρων του συστήματος περιγράφεται και μεταφέρεται στο σύστημα μέσω του πρωτοκόλλου HTTP από διάφορους clients (ανεξαρτήτως της γλώσσας στην οποία έχουν υλοποιηθεί). [48]

Κάθε πόρος προσδιορίζεται από URIs και καθολικά αναγνωριστικά. Το REST χρησιμοποιεί διάφορες παραστάσεις για να αντιπροσωπεύσει ένα πόρο όπως απλό κείμενο, JSON, XML αλλά το JSON είναι το πιο δημοφιλές. [47]

Μέθοδοι HTTP

Οι παρακάτω μέθοδοι HTTP χρησιμοποιούνται πιο συχνά σε REST βασισμένες αρχιτεκτονικές.

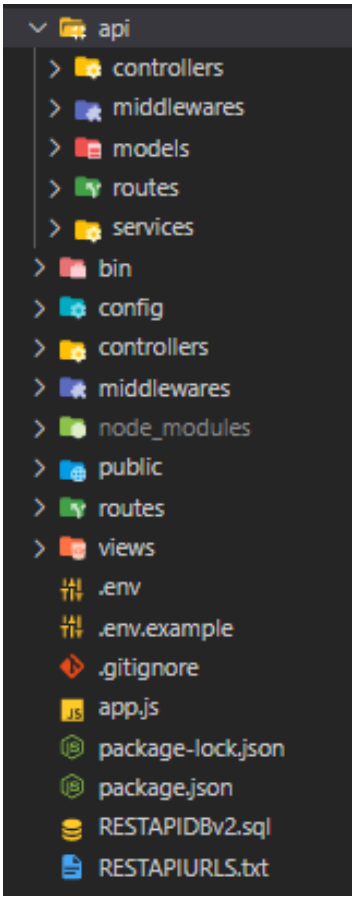
- **GET** – Για την παροχή πρόσβασης σε ένα πόρο μόνο για ανάγνωση.
- **POST** – Για την δημιουργία ενός νέου πόρου.
- **DELETE** – Για την διαγραφή ενός πόρου.
- **PUT** – Για την ενημέρωση ενός υπάρχοντος πόρου ή τη δημιουργία ενός νέου πόρου.

2.5.1 Δομικά μέρη

Όπως φαίνεται από στην αρχιτεκτονική, το Back-end REST API αποτελείται από 6 βασικά κομμάτια:

- το κεντρικό αρχείο της εφαρμογής **app.js**,
- τις διαδρομές (**routes**),
- τους ελεγκτές (**controllers**),
- τις υπηρεσίες (**services**),
- τα μοντέλα (**models**),
- και το ενδιάμεσο λογισμικό (**middleware**).

Επίσης οι βιβλιοθήκες/modules/εξαρτήσεις που χρησιμοποιεί αποθηκεύονται στον φάκελο **node_modules** και δανείζεται υπηρεσίες από «κοινόχρηστα» modules του φακέλου **config**.



Εικόνα 5 Δομικά μέρη Back-end REST API

Μια συνηθισμένη πρακτική στην κατασκευή REST API είναι η χρήση των παραπάνω αλλά χωρίς τις υπηρεσίες. Δηλαδή οι υπηρεσίες να είναι ενσωματωμένες στους ελεγκτές. Αυτό θεωρήθηκε κακή πρακτική στην εφαρμογή αυτή εφόσον γινόταν συχνή χρήση του ίδιου κώδικα από πολλούς ελεγκτές. Η διαφοροποίηση των υπηρεσιών από τους ελεγκτές είχε επίσης όφελος στην δημιουργία του ενδιάμεσου λογισμικού και του Server-Side Rendering (SSR) καθώς πλέον μπορούσαν να χρησιμοποιήσουν και αυτά τις υπηρεσίες αυτές. Ποιο αναλυτικά:

Κεντρικό αρχείο εφαρμογής:

- Συνδέει όλα τα μέρη μεταξύ τους.
- Περιέχει επιχειρησιακή λογική και δεδομένα.

Διαδρομές (routes):

- Δέχονται HTTP requests.
- Χρησιμοποιούν κυρίως ενδιάμεσο λογισμικό και ελεγκτές.
- Δεν περιέχουν επιχειρησιακή λογική.
- Προωθούν δεδομένα από τις παραμέτρους ή το σώμα του request στις στους ελεγκτές.

Ελεγκτές (controllers):

- Διαχειρίζονται τα εισερχόμενα HTTP requests.
- Επιστρέφουν μία απάντηση (response).
- Αποφασίζουν τι υπηρεσίες θα χρησιμοποιήσουν.
- Προωθούν δεδομένα από τις παραμέτρους ή το σώμα του request στις υπηρεσίες.
- Δεν περιέχουν επιχειρησιακή λογική.

Υπηρεσίες (services):

- Δέχονται δεδομένα και επιστρέφουν δεδομένα.
- Περιέχουν την επιχειρησιακή λογική της εφαρμογής.
- Συνεργάζονται με άλλες υπηρεσίες.
- Χρησιμοποιούν τα μοντέλα οντοτήτων.

Μοντέλα (models):

- Δεν περιέχουν επιχειρησιακή λογική (εκτός αν θεωρηθεί αναγκαίο για μια λειτουργία).
- Καθένα αντιστοιχεί σε ένα πίνακα της βάσης δεδομένων.
- Εκτελούν λειτουργίες CRUD στη βάση δεδομένων.
- Δέχονται δεδομένα από τις υπηρεσίες.
- Επιστρέφουν δεδομένα.

Ενδιάμεσο λογισμικό (middleware):

- Χρησιμοποιούνται από τις διαδρομές.
- Συνήθως καλούνται πριν τους ελεγκτές.
- Δέχονται δεδομένα από τις διαδρομές.
- Επιστρέφουν δεδομένα.

- Περιέχουν επιχειρησιακή λογική.

2.5.2 Μοντέλα οντοτήτων (models)

Για τη δημιουργία των μοντέλων οντοτήτων (models) χρησιμοποιήθηκε η βιβλιοθήκη Sequelize ORM. Έπειτα για τη δημιουργία νέων συνδέσεων του συστήματος με τη βάση δεδομένων δημιουργήθηκε ένα νέο module με ονομασία sequelize.js στον φάκελο config όπως φαίνεται παρακάτω,

```
// config/sequelize.js

const { Sequelize } = require('sequelize').Sequelize;

module.exports = new Sequelize(process.env.DB_NAME, process.env.DB_USER, pr
ocess.env.DB_PASS, {
  host: process.env.DB_HOST,
  dialect: process.env.DB_TYPE
});
```

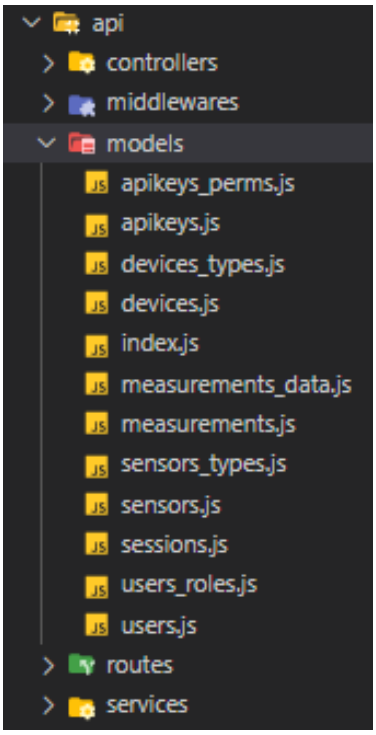
και προστέθηκε κώδικας για τη δοκιμή της σύνδεσης με την έναρξη της εφαρμογής στο αρχείο app.js

```
// app.js

// Sequelize connection test
const sequelize = require('./config/sequelize');
sequelize.authenticate()
    .then(() => console.log(cconsole.tcolor.green, '\n[sequelize] Database
Connection: Connected! \n'))
    .catch(err => console.log(cconsole.tcolor.red, '\n[sequelize] Database
Connection: Failed! \n\n', cconsole.tcolor.white, err, '\n'))
```

Τέλος, για την αυτόματη δημιουργία των μοντέλων έγινε χρήση της παρακάτω εντολής:

```
sequelize-auto -h localhost -d restapitestv2 -u root -p 3306 -o "./models" -e
mysql -l es6
```

Έτσι δημιουργήθηκαν στον φάκελο models τα μοντέλα/κλάσεις της εφαρμογής με τη μορφή ES6 CJS module. Το καθένα έχει ως όνομα αρχείου το όνομα της οντότητας που αντιστοιχεί και κατάληξη .js όπως φαίνεται παρακάτω.



Εικόνα 6 Δημιουργία μοντέλων οντοτήτων με Sequelize-Auto

Παράδειγμα αυτόματου μοντέλου/κλάσης users:

```
// api/models/users.js

const Sequelize = require('sequelize');
module.exports = (sequelize, DataTypes) => {
  return users.init(sequelize, DataTypes);
}

class users extends Sequelize.Model {
  static init(sequelize, DataTypes) {
  super.init({
    uid: {
      autoIncrement: true,
      type: DataTypes.INTEGER(11),
      allowNull: false,
      unique: true
    },
    firstname: {
      type: DataTypes.STRING(60),
      allowNull: true
    },
    lastname: {
      type: DataTypes.STRING(60),
      allowNull: true
    },
```

```
    username: {
      type: DataTypes.STRING(32),
      allowNull: false,
      defaultValue: "demo",
      primaryKey: true
    },
    email: {
      type: DataTypes.STRING(100),
      allowNull: false,
      defaultValue: "demo",
      unique: true
    },
    passwd: {
      type: DataTypes.STRING(60),
      allowNull: false
    },
    createdAt: {
      type: DataTypes.DATE,
      allowNull: false,
      defaultValue: sequelize.fn('current_timestamp')
    },
    updatedAt: {
      type: DataTypes.DATE,
      allowNull: false,
      defaultValue: sequelize.fn('current_timestamp')
    },
    role: {
      type: DataTypes.STRING(32),
      allowNull: false,
      defaultValue: "user",
      references: {
        model: {
          tableName: 'users_roles',
        },
        key: 'rolename'
      }
    }
  }, {
    sequelize,
    tableName: 'users'
  });
  return users;
  }
}
```

Επίσης μια συνηθισμένη πρακτική μετά τη δημιουργία των μοντέλων, είναι η δημιουργία ενός «κεντρικού» index αρχείου το οποίο εισάγει και εξάγει όλα τα μοντέλα αλλά περιέχει επίσης και τις σχέσεις τους. Το όφελος από αυτό είναι ότι δε χρειάζεται να εισαχθούν μεμονωμένα τα μοντέλα στον κώδικά όπου αυτά χρειάζονται αλλά εισάγεται το index.js αρχείο και έπειτα καλούνται τα μοντέλα μέσω αυτού.

Παράδειγμα «κεντρικού» index αρχείου μοντέλων:

```
// api/models/index.js

// Εισαγωγή βιβλιοθηκών
const { Sequelize, Op, Model, DataTypes } = require("sequelize");
const sequelize = require('../../config/sequelize');


// Εισαγωγή μοντέλων
const User = require('../../api/models/users')(sequelize, DataTypes);
const UserRole = require('../../api/models/users_roles')(sequelize, DataTyp
es);
const Device = require('../../api/models/devices')(sequelize, DataTypes);
const DeviceType = require('../../api/models/devices_types')(sequelize, Dat
aTypes);
const Sensor = require('../../api/models/sensors')(sequelize, DataTypes);
const SensorType = require('../../api/models/sensors_types')(sequelize, Dat
aTypes);
const Measurement = require('../../api/models/measurements')(sequelize, Dat
aTypes);
const MeasurementData = require('../../api/models/measurements_data')(seque
lize, DataTypes);
const ApiKey = require('../../api/models/apikeys')(sequelize, DataTypes);
const ApiKeyPerm = require('../../api/models/apikeys_perms')(sequelize, Dat
aTypes);


// Εισαγωγή δημιουργία σχέσεων

/*******USERS**********/
User.belongsTo(UserRole, { as: "userrole", foreignKey: "role" });
UserRole.hasMany(User, { as: "users", foreignKey: "role" });
/**********************/

/*******DEVICES**********/
Device.belongsTo(User, { as: "user", foreignKey: "username" });
User.hasMany(Device, { as: "devices", foreignKey: "username" });

Device.belongsTo(DeviceType, { as: "type", foreignKey: "devicetype" });
DeviceType.hasMany(Device, { as: "devices", foreignKey: "devicetype" });
```

```
/**********************/


/*******API KEYS*********/
ApiKey.belongsTo(User, { as: "user", foreignKey: "username" });
User.hasMany(ApiKey, { as: "apikeys", foreignKey: "username" });

ApiKey.belongsTo(ApiKeyPerm, { as: "perm", foreignKey: "perms" });
ApiKeyPerm.hasMany(ApiKey, { as: "apikeys", foreignKey: "perms" });

ApiKey.belongsTo(Device, { as: "device", foreignKey: "deviceid" });
Device.hasMany(ApiKey, { as: "apikeys", foreignKey: "deviceid" });
/**********************/


/*******SENSORS*********/
Sensor.belongsTo(User, { as: "user", foreignKey: "username" });
User.hasMany(Sensor, { as: "sensors", foreignKey: "username" });

Sensor.belongsTo(Device, { as: "device", foreignKey: "deviceid" });
Device.hasMany(Sensor, { as: "sensors", foreignKey: "deviceid" });

Sensor.belongsTo(SensorType, { as: "type", foreignKey: "sensortype" });
SensorType.hasMany(Sensor, { as: "sensors", foreignKey: "sensortype" });
/**********************/


/*******MEASUREMENTS*******/
Measurement.belongsTo(User, { as: "user", foreignKey: "username" });
User.hasMany(Measurement, { as: "measurements", foreignKey: "username" });

Measurement.belongsTo(Device, { as: "device", foreignKey: "deviceid" });
Device.hasMany(Measurement, { as: "measurements", foreignKey: "deviceid" })
;
/**********************/


/*******MEASUREMENT DATA*******/
MeasurementData.belongsTo(Measurement, { as: "measurement", foreignKey: "me
asurementid" });
Measurement.hasMany(MeasurementData, { as: "data", foreignKey: "measurement
id" });

MeasurementData.belongsTo(Device, { as: "device", foreignKey: "deviceid" })
;
Device.hasMany(MeasurementData, { as: "data", foreignKey: "deviceid" });

MeasurementData.belongsTo(Sensor, { as: "sensor", foreignKey: "sensorid" })
;
Sensor.hasMany(MeasurementData, { as: "data", foreignKey: "sensorid" });
```

```
/**********************/

// Εξαγωγή μοντέλων
module.exports = {
    User,
    UserRole,
    Device,
    DeviceType,
    Sensor,
    SensorType,
    Measurement,
    MeasurementData,
    ApiKey,
    ApiKeyPerm
}
```

2.5.3 Διαδρομές (routes)

Όπως αναφέρθηκε στην εισαγωγή, το REST API δέχεται HTTP requests (με μεθόδους όπως GET, POST, PUT, DELETE, κλπ.) και έπειτα πραγματοποιεί λειτουργίες CRUD (Create, Read, Update, Delete) στη βάση δεδομένων. Για την αλλαγή των δεδομένων μιας οντότητας στη βάση, πρέπει να υπάρχουν διαδρομές (routes) που αντιστοιχούν σε αυτές και τις λειτουργίες CRUD.

HTTP Μέθοδος	CRUD Λειτουργία
POST	CREATE
GET	READ
PUT	UPDATE
DELETE	DELETE

Πίνακας 5 HTTP Μέθοδοι σε CRUD Λειτουργίες

2.5.3.1 Ανάλυση διαδρομών

Για την εφαρμογή αυτή ήταν απαραίτητο να δημιουργηθούν οι παρακάτω διαδρομές/λειτουργίες με βάση τις λειτουργικές απαιτήσεις και τη βάση δεδομένων της εφαρμογής.

Διαδρομή	Μέθοδος	Περιγραφή

/devices	GET	Επιστρέφει λίστα με όλες τις συσκευές.
/devices/count	GET	Επιστρέφει τον αριθμό όλων των συσκευών.
/devices/types	GET	Επιστρέφει λίστα με όλους τους τύπους συσκευών.
/sensors	GET	Επιστρέφει λίστα με όλους τους αισθητήρες.
/sensors/count	GET	Επιστρέφει τον αριθμό όλων των αισθητήρων
/sensors/types	GET	Επιστρέφει λίστα με όλους τους τύπους αισθητήρων.
/apikeys	GET	Επιστρέφει λίστα με όλα τα κλειδιά API των συσκευών.
/apikeys/count	GET	Επιστρέφει τον αριθμό όλων των κλειδιών API των συσκευών.
/measurements	GET	Επιστρέφει λίστα με όλες τις μετρήσεις.
/measurements/count	GET	Επιστρέφει τον αριθμό όλων των μετρήσεων.
/users	GET	Επιστρέφει λίστα με όλους τους χρήστες και τις συσκευές τους.
/users/signup	POST	Δημιουργεί νέο χρήστη.
/users/login	POST	Επιστρέφει ένα JWT Token με την επιτυχή επαλήθευση των στοιχείων του χρήστη.
/users/count	GET	Επιστρέφει τον αριθμό όλων των χρηστών.
/users/sessions	GET	Επιστρέφει τον αριθμό όλων των ενεργών συνεδριών των χρηστών.
/users/:username	GET	Επιστρέφει τα στοιχεία και τις συσκευές ενός χρήστη.
/users/:username	PUT	Αλλάζει τα στοιχεία του χρήστη.
/users/:username	DELETE	Διαγράφει έναν χρήστη.
/users/:username/role	PUT	Αλλάζει τον ρόλο του χρήστη.
/users/:username/devices	GET	Επιστρέφει λίστα με όλες τις συσκευές ενός χρήστη.
/users/:username/devices	POST	Δημιουργεί μία νέα συσκευή σε ένα χρήστη.
/users/:username/devices/:device	GET	Επιστρέφει τα στοιχεία και τους αισθητήρες μίας συσκευής.

/users/:username/devices/:device	PUT	Αλλάζει τα στοιχεία μίας συσκευής ενός χρήστη.
/users/:username/devices/:device	DELETE	Διαγράφει μία συσκευή ενός χρήστη.
/users/:username/devices/:device/sensors	GET	Επιστρέφει μία λίστα με όλους τους αισθητήρες μίας συσκευής ενός χρήστη.
/users/:username/devices/:device/sensors	POST	Δημιουργεί ένα νέο αισθητήρα για μία συσκευή ενός χρήστη.
/:username/devices/:device/sensors/:sensor	GET	Επιστρέφει τα στοιχεία ενός αισθητήρα μίας συσκευής ενός χρήστη.
/:username/devices/:device/sensors/:sensor	PUT	Αλλάζει τα στοιχεία ενός αισθητήρα μίας συσκευής ενός χρήστη.
/:username/devices/:device/sensors/:sensor	DELETE	Διαγράφει έναν αισθητήρα μίας συσκευής ενός χρήστη.
/users/:username/devices/:device/sensors/:sensor/measurements	POST	Δημιουργεί μία νέα μέτρηση για έναν αισθητήρα μίας συσκευής ενός χρήστη.
/users/:username/devices/:device/sensors/:sensor/measurements	GET	Επιστρέφει μία λίστα με τις μετρήσεις ενός αισθητήρα μίας συσκευής ενός χρήστη.
/users/:username/devices/:device/sensors/:sensor/measurements/:measurement	DELETE	Διαγράφει μία μέτρηση ενός αισθητήρα μίας συσκευής ενός χρήστη.
/users/:username/devices/:device/data	POST	Δημιουργεί μία νέα μέτρηση για μία συσκευή ενός χρήστη.
/users/:username/devices/:device/data	GET	Επιστρέφει μία λίστα με όλες τις μετρήσεις μίας συσκευής ενός χρήστη.
/users/:username/devices/:device/data/:measurement	DELETE	Διαγράφει μία μέτρηση μίας συσκευής ενός χρήστη.
/users/:username/apikeys	POST	Δημιουργεί ένα νέο κλειδί API για ένα χρήστη.

/users/:username/apikeys	GET	Επιστρέφει μία λίστα με κλειδιά API ενός χρήστη.
/users/:username/apikeys /:apikey	PUT	Αλλάζει τα στοιχεία ενός κλειδιού API ενός χρήστη.
/users/:username/apikeys /:apikey	DELETE	Διαγράφει ένα κλειδί API ενός χρήστη.

Πίνακας 6 Διαδρομές Back-end (REST-API)

Μερικές από τις παραπάνω διαδρομές έχουν τα εξής χαρακτηριστικά:

- Είναι εμφωλευμένες (nested), δηλαδή η διαδρομή μίας οντότητας μπορεί να περιέχει ή να αναφέρει μία άλλη.
- Περιέχουν παραμέτρους διεύθυνσης (url/path parameters), δηλαδή μεταβλητές που αφορούν ένα αντικείμενο ή οντότητα στη διεύθυνση URL.
- Περιέχουν παραμέτρους σώματος (body parameters), π.χ. όπου POST Request, το Body πρέπει να είναι τύπου JSON και να περιέχει ένα αντικείμενο ή να έχει τη μορφή πίνακα που περιέχει αντικείμενα.

Για παράδειγμα, η εφαρμογή μπορεί να δεχτεί δεδομένα από οποιαδήποτε συσκευή έχει τη δυνατότητα να λειτουργήσει ως Client, δηλαδή:

1. να έχει πρόσβαση στο διαδίκτυο,
2. να μπορεί να κάνει HTTP POST request με χρήση Authorization Header για την αποστολή Bearer Token και
3. να στείλει Body τύπου JSON (το Body πρέπει να έχει τη μορφή πίνακα με αντικείμενα μετρήσεων/τιμές από αισθητήρες).

Οι διαδρομές μπορούν να χρησιμοποιούν μεθόδους από ελεγκτές (controllers), υπηρεσίες (services), μοντέλα (models) και ενδιάμεσο λογισμικό (**middleware**) όπως φαίνεται στο παράδειγμα παρακάτω αλλά καλύτερη πρακτική θεωρείται η χρήση μόνο ελεγκτών και ενδιάμεσου λογισμικού.

2.5.3.2 Παράδειγμα δημιουργίας διαδρομών users.js

```
// api/routes/users.js

// Εισαγωγή βιβλιοθηκών
const express = require('express');
const router = express.Router();

// Εισαγωγή ελεγκτή
const Controller = require('../controllers/index');

// Εισαγωγή ρόλων των χρηστών (ενδιάμεσο λογισμικό)
const Role = require('../middlewares/roles');

// Εισαγωγή  ελεγκτή αυθεντικοποίησης (ενδιάμεσο λογισμικό)
const authorize = require('../middlewares/authorize');
const authorizeMe = require('../middlewares/authorize_me');


// Δημιουργία διαδρομών

// Προβολή λίστας με όλους τους χρήστες
router.get('/', authorize(Role.Admin), Controller.User.getAllUsers);

// Εγγραφή χρήστη
router.post('/signup', Controller.User.signup);

// Σύνδεση χρήστη
router.post('/login', Controller.User.login);

// Προβολή αριθμού εγγεγραμμένων χρηστών
router.get('/count', authorize(Role.Admin), Controller.User.getAllUsersCoun
t);

// Προβολή αριθμού ενεργών χρηστών
router.get('/sessions', authorize(Role.Admin), Controller.User.getAllUsersS
essions);

// Προβολή στοιχείων χρήστη
router.get('/:username', authorize(), authorizeMe(), Controller.User.getUse
r);

// Διαγραφή χρήστη
router.delete('/:username', authorize(), authorizeMe(), Controller.User.del
eteUser);

// Αλλαγή στοιχείων χρήστη
router.put('/:username', authorize(), authorizeMe(), Controller.User.update
UserInfo);
```

```
// Αλλαγή ρόλου χρήστη
router.put('/:username/role', authorize(Role.Admin), authorizeMe(), Control
ler.User.updateUserRole);


// Δήλωση εμφωλευμένων διαδρομών που περιέχουν τη διαδρομή /users και τη πα
ράμετρο username
const devicesRouter = require('./devices');
router.use('/:username/devices', devicesRouter);

const keysRouter = require('./apikeys');
router.use('/:username/apikeys', keysRouter);

// Εξαγωγή των διαδρομών στο express router
module.exports = router;
```

Παρομοίως δημιουργήθηκαν οι διαδρομές και για τις υπόλοιπες οντότητες και λειτουργίες.

2.5.4 Ελεγκτές (controllers)

Με βάση τις παραπάνω διαδρομές, δημιουργήθηκαν οι ελεγκτές και οι υπηρεσίες τους. Ο κάθε ελεγκτής, και πιο συγκεκριμένα οι μέθοδοι του ελεγκτή είναι υπεύθυνοι για τον έλεγχο και την εγκυρότητα των δεδομένων που εισέρχονται αλλά και για την σωστή απάντηση (response) που θα σταλεί στο κάθε request ενώ η απάντηση πρέπει πάντα να συνοδεύεται από ένα κωδικό κατάστασης HTTP (HTTP Status Code).

Το HTTP καθορίζει τους κωδικούς αυτούς και χωρίζονται σε πέντε κατηγορίες [49]:
1xx: Ενημερωτικό - Αυτοί οι κωδικοί κατάστασης υποδεικνύουν μια προσωρινή απόκριση.
2xx: Επιτυχία - Αυτή η κλάση κωδικών κατάστασης υποδεικνύει ότι ο διακομιστής αποδέχτηκε με επιτυχία την αίτηση του υπολογιστή-πελάτη.
3xx: Ανακατεύθυνση - Το πρόγραμμα περιήγησης του υπολογιστή-πελάτη θα πρέπει να πραγματοποιήσει περαιτέρω ενέργειες για να ικανοποιήσει την αίτηση.
4xx: Σφάλμα προγράμματος πελάτη - Προκύπτει σφάλμα και ο υπολογιστής-πελάτης ενδέχεται να αντιμετωπίζει κάποιο πρόβλημα.
5xx: Σφάλμα διακομιστή - Δεν είναι δυνατή η ολοκλήρωση της αίτησης από το διακομιστή επειδή προέκυψε κάποιο σφάλμα.

2.5.4.1 Απόσπασμα με μεθόδους του ελεγκτή apikeys.js

```
// api/controllers/apikeys.js

// Εισαγωγή υπηρεσιών
const Service = require('../services/index');

// Εξαγωγή μεθόδων ελεγκτή
module.exports = {

    // Μέθοδος που επιστρέφει τα κλειδιά API ενός χρήστη
    async getAllApiKeysOfUser(req, res, next) {
        try {
            let _username = req.params.username; // Όρισμα παραμέτρου διεύθ
υνσης url
            const apikeys = await Service.ApiKey.getAllApiKeysOfUser(_usern
ame); // Κλήση μεθόδου από την υπηρεσία ApiKey
            if (apikeys) {
                res.status(201).json(apikeys); // Απάντηση με τη λίστα
            }
            else { res.status(404).json("Api keys not found") } // Απάντηση
 με μήνυμα
        }
        catch (e) {
            console.log(e);
            res.status(500).json(e);
        }
    },

    // Μέθοδος που επιστρέφει όλα τα κλειδιά API όλων των χρηστών
    async getAllApiKeys(req, res, next) {
        try {
            const apikeys = await Service.ApiKey.getAllApiKeys(); // Κλήση
μεθόδου από την υπηρεσία ApiKey
            if (apikeys) {
                res.range({
                    first: req.range.first,
                    last: req.range.last,
                    length: apikeys.length
                });
                res.status(200).json(apikeys); // Απάντηση με τη λίστα
            }
            else { res.status(404).json({ message: "Api Keys not found" })
} // Απάντηση με μήνυμα
        }
        catch (e) {
            console.log(e);
            res.status(500).json(e);
        }
    },
```

```
    // Μέθοδος που ενημερώνει τα στοιχεία ενός κλειδιού API
    async updateApiKey(req, res, next) {
        try {
            // Έλεγχος εγκυρότητας σώματος request
            if (Object.keys(req.body).length === 0 && req.body.constructor
=== Object) {
                res.status(400).json({
                    message: "Content can not be empty!"
                });
            } else {
                // Έλεγχος πεδίων/παραμέτρων σώματος
                req.checkBody('deviceid', 'The device id field must be betw
een 4-100 characters long, please try again.').len(4, 64);
                req.checkBody('deviceid', 'The device id field cannot be em
pty.').notEmpty();
                req.checkBody('perms', 'The permissions field field cannot
be empty.').notEmpty();
                req.checkBody('apikeyname', 'The api key name field must be
 between 4-100 characters long, please try again.').len(4, 64);
                req.checkBody('apikeyname', 'The api key name field field c
annot be empty.').notEmpty();
                req.checkBody('apikeydescription', 'The api key description
 must be between 0-200 characters long, please try again.').len(0, 200);
                req.checkBody('apikeystatus', 'Please insert numbers only.'
).isNumeric();

                const errors = req.validationErrors();

                if (errors) {
                    // Απάντηση με μήνυμα σφάλματος
                    res.status(401).json({
                        message: "Update failed",
                        errors: errors
                    });
                } else {
                    let _username = req.params.username; // Όρισμα παραμέτρ
ου διεύθυνσης url
                    let _apikeyid = req.params.apikey; // Όρισμα παραμέτρου
 διεύθυνσης url
                    let _deviceid = req.body.deviceid; // Όρισμα παραμέτρου
 σώματος
                    let _perms = req.body.perms; // Όρισμα παραμέτρου σώματ
ος
                    let _apikeyname = req.body.apikeyname; // Όρισμα παραμέ
τρου σώματος
                    let _apikeydescription = req.body.apikeydescription; //
 Όρισμα παραμέτρου σώματος
```

```
                    let _apikeystatus = req.body.apikeystatus; // Όρισμα πα
ραμέτρου σώματος
                    const updated_apikey = Service.ApiKey.updateApiKeyInfo(
_username, _apikeyid, _deviceid, _perms, _apikeyname, _apikeydescription, _
apikeystatus); // Κλήση μεθόδου από την υπηρεσία ApiKey
                    if (updated_apikey) {
                        res.range({
                            first: req.range.first,
                            last: req.range.last,
                            length: 1
                        });
                        res.status(200).json({ message: "Api key updated" }
); // Απάντηση με μήνυμα
                    }
                    else {  res.status(404).json({ message: "Api key not fo
und" }) } // Απάντηση με μήνυμα
                }
            }
        }
        catch (e) {
            console.log(e);
            res.status(500).json(e);
        }
    },

    // Μέθοδος που διαγράφει ένα κλειδί API
    async deleteApiKey(req, res, next) {
        try {
            let username = req.params.username; // Όρισμα παραμέτρου διέυθυ
νσης url
            let apikeyid = req.params.apikey; // Όρισμα παραμέτρου διέυθυνσ
ης url
            const deleted_apikey = await Service.ApiKey.deleteApiKey(userna
me, apikeyid); // Κλήση μεθόδου από την υπηρεσία ApiKey
            if (deleted_apikey) {
                res.range({
                    first: req.range.first,
                    last: req.range.last,
                    length: 1
                });
                res.status(200).json({ message: "Api key deleted" }); // Απ
άντηση με μήνυμα
            }
            else { res.status(404).json({ message: "Api key not found" }) }
 // Απάντηση με μήνυμα
        }
        catch (e) {
            console.log(e);
            res.status(500).json(e);
```

```
        }
    }

}
```

Όπως στα μοντέλα, και εδώ είναι συνηθισμένη πρακτική μετά τη δημιουργία των ελεγκτών, η δημιουργία ενός «κεντρικού» index αρχείου το οποίο εισάγει και εξάγει όλους τους ελεγκτές.

2.5.4.2 Παράδειγμα «κεντρικού» index αρχείου ελεγκτών

```
// api/controllers/index.js

// Εισαγωγή ελεγκτών
const User = require('./users');
const Device = require('./devices');
const Sensor = require('./sensors');
const ApiKey = require('./apikeys');
const Measurement = require('./measurements');

// Εξαγωγή μοντέλων
module.exports = {
    User,
    Device,
    Sensor,
    ApiKey,
    Measurement
}
```

2.5.5 Υπηρεσίες (services)

2.5.5.1 Απόσπασμα με μεθόδους της υπηρεσίας apikeys.js

```
// api/services/apikey.js

// Εισαγωγή βιβλιοθηκών
const jwt = require('jsonwebtoken');

// Εισαγωγή μοντέλων
const Model = require('../models/index');

// Εξαγωγή μεθόδων υπηρεσίας
module.exports = {

    // Μέθοδος δημιουργίας κλειδιού API με τη μορφή JWT Token
    async generateToken(_username, _apikeyid) {
```

```
        try {
            const token = jwt.sign(
                {
                    username: _username,
                    apikeyid: _apikeyid
                },
                process.env.JWT_KEY,
                {
                    noTimestamp: true
                }
            );
            return token;
        }
        catch (err) {
            console.log(err);
        }
    },

    // Μέθοδος που επιστρέφει όλα τα κλειδιά API όλων των χρηστών
    getAllApiKeys() {
        return Model.ApiKey.findAll({})  // Κλήση μεθόδου από το μοντέλο Ap
iKey
            .then(async data => {
                // Create the api keys (tokens) on the go since they are no
t stored in db
                for (var i = 0; i < data.length; i++) {
                    let generated_apikey = await this.generateToken(data[i]
.username, data[i].apikeyid);
                    data[i].setDataValue("apikey", generated_apikey);
                    data[i] = data[i].get({ plain: true });
                };
                return data; // Επιστροφή δεδομένων
            })
            .catch(err => {
                console.log(err);
            });
    },

    // Μέθοδος που επιστρέφει τα κλειδιά API ενός χρήστη
    getAllApiKeysOfUser(_username) {
        return Model.ApiKey.findAll({ where: { username: _username } })  //
 Κλήση μεθόδου από το μοντέλο ApiKey
            .then(async data => {
                // Create the api keys (tokens) on the go since they are no
t stored in db
                for (var i = 0; i < data.length; i++) {
                    let generated_apikey = await this.generateToken(_userna
me, data[i].apikeyid);
                    data[i].setDataValue("apikey", generated_apikey);
```

```
                    data[i] = data[i].get({ plain: true });
                };
                return data; // Επιστροφή δεδομένων
            })
            .catch(err => {
                console.log(err);
            });
    },

    // Μέθοδος που ενημερώνει τα στοιχεία ενός κλειδιού API
    updateApiKeyInfo(_username, _apikeyid, _deviceid, _perms, _apikeyname,
_apikeydescription, _apikeystatus) {
        return Model.ApiKey.findOne({ where: { username: _username, devicei
d: _deviceid, apikeyid: _apikeyid } })  // Κλήση μεθόδου από το μοντέλο Api
Key
            .then((data) => {
                if (data) {
                    data.update({ perms: _perms, apikeyname: _apikeyname, a
pikeydescription: _apikeydescription, apikeystatus: _apikeystatus }).then(
                        updatedData => {
                            return true; // Επιστροφή ένδειξης επιτυχής ενη
μέρωσης
                        })
                }
            })
            .catch(err => {
                console.log(err);
            });
    },

    // Μέθοδος που διαγράφει ένα κλειδί API
    deleteApiKey(_username, _apikeyid) {
        return Model.ApiKey.destroy({ where: { username: _username, apikeyi
d: _apikeyid } }) // Κλήση μεθόδου από το μοντέλο ApiKey
            .then((data) => {
                return data; // Επιστροφή δεδομένων
            })
            .catch(err => {
                console.log(err);
            });
    }

}
```

2.5.6 Ενδιάμεσο λογισμικό (middleware)

Τα ενδιάμεσα λογισμικά – μέθοδοι που υλοποιήθηκαν για την εφαρμογή αυτή έχουν τη μορφή module και είναι οι παρακάτω:

- **authorize_device.js** – Για την εξουσιοδότηση των συσκευών.
- **authorize_me.js** – Για τον περιορισμό των χρηστών στις διευθύνσεις url που τους ανήκουν.
- **authorize.js** – Για την εξουσιοδότηση των χρηστών.
- **error_handler.js** – Για τον χειρισμό σφαλμάτων.
- **roles.js** – Για τον έλεγχο των ρόλων των χρηστών.

Πιο συγκεκριμένα για τις λειτουργίες της αυθεντικοποίησης και εξουσιοδότησης των χρηστών και των συσκευών χρησιμοποιήθηκε η βιβλιοθήκη JSONWebToken**.** Το κλειδί που αφορά ένα χρήστη ονομάζεται JWT Token ενώ το κλειδί που αφορά μια συσκευή ονομάζεται API Key. Το JWT Token δημιουργείται κατά την εγγραφή ή την αυθεντικοποίηση του χρήστη στις διαδρομές /users/signup και /users/login αντίστοιχα και αποθηκεύεται στο Front-end όπου αυτό έχει ορίσει. Ενώ το API Key μιας συσκευής δημιουργείται από τον χρήστη μέσω της εφαρμογής.

Στο Front-end, υπάρχουν 4 πιθανές τοποθεσίες αποθήκευσης ενός JWT Token:

- Cookie
- Cookie με την ιδιότητα httpOnly
- LocalStorage
- SessionStorage

Στην εφαρμογή αυτή θεωρήθηκε ασφαλέστερη η χρήση ενός Cookie με την ιδιότητα httpOnly. Ένα Cookie με την ιδιότητα αυτή, δεν είναι προσβάσιμο στο JavaScript Document.cookie API και αποστέλλεται μονάχα στον διακομιστή. Αυτή η προφύλαξη βοηθά στην αποφυγή επιθέσεων τύπου cross-site scripting (XSS) [50]. Παρόλα αυτά, στα middleware authorize_device.js και authorize.js έγινε χρήση της βιβλιοθήκης Express-JWT και υλοποίηση κώδικα που δέχεται το JWT Token του χρήστη από τρείς τοποθεσίες όπως:

1. το Authorization Header του HTTP request σαν Bearer Token,
2. μέσω παραμέτρου στη διεύθυνση URL,,

3. μέσω Cookie με την ιδιότητα httpOnly.

2.5.6.1 Απόσπασμα με μεθόδους του authorize.js

```
// api/middlewares/authorizs.js

// Εισαγωγή βιβλιοθηκών
const jwt = require('express-jwt');

// Μέθοδος που επιστρέφει ένα αντικείμενο με το όνομα του cookie
// Από https://alligator.io/nodejs/express-cookies/
const getAppCookies = (req) => {
    const rawCookies = req.headers.cookie.split('; ');
    const parsedCookies = {};
    rawCookies.forEach(rawCookie => {
        const parsedCookie = rawCookie.split('=');
        parsedCookies[parsedCookie[0]] = parsedCookie[1];
    });
    return parsedCookies;
};

// Μέθοδος εξουσιοδότησης χρήστη
// Βασισμένο σε https://jasonwatmore.com/post/2018/11/28/nodejs-role-based-
authorization-tutorial-with-example-api#user-service-js
// και https://www.npmjs.com/package/express-jwt
function authorize(roles = []) {
    try {
        if (typeof roles === 'string') {
            roles = [roles];
        }
        const _secret = process.env.JWT_KEY; // Ορισμός κρυφου κλειδιού

        return [
            // Αυθεντικοποίηση του JWT token και εισαγωγή των πληροφοριών τ
ου χρήστη στο αντικείμενο (req.user)
            jwt({
                secret: _secret,
                algorithms: ['HS256'],

                // Τοποθεσία κλειδιού
                getToken: function fromHeaderOrQuerystring(req) {
                    // Bearer Token (Authorization Header)
                    if (req.headers.authorization && req.headers.authorizat
ion.split(' ')[0] === 'Bearer') {
                        return req.headers.authorization.split(' ')[1];
                    }
                    // Παράμετρος διεύθυνσης
                    else if (req.query && req.query.token) {
                        return req.query.token;
```

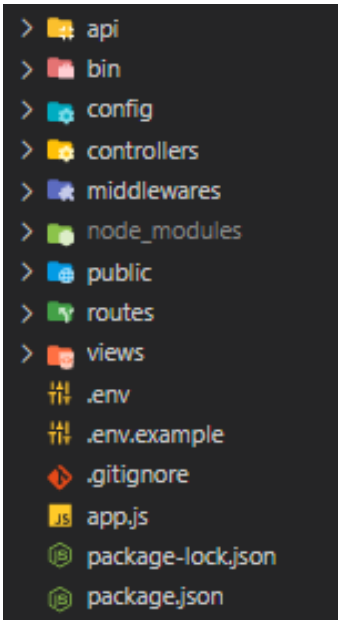
```
                    }
                    // httpOnly cookie
                    else if (getAppCookies(req)['jwt_token']) {
                        return getAppCookies(req)['jwt_token'];
                    }
                    return null;
                }
            }),

            // Εξουσιοδότηση βάση ρόλου
            (req, res, next) => {
                if (roles.length && !roles.includes(req.user.role)) {
                    return res.status(403).json({ message: 'Unauthorized' }
);

                }
                next();
            }
        ];
    } catch (e) {
        console.log(e);
        res.status(500).send(e);
    }
}

// Εξαγωγή module
module.exports = authorize;
```

2.6 Ανάλυση Front-end

2.6.1 Δομικά μέρη

Όπως φαίνεται από στην αρχιτεκτονική, το Front-end αποτελείται επίσης από 6 βασικά κομμάτια:

- το κεντρικό αρχείο της εφαρμογής **app.js**,
- τις διαδρομές (**routes**),
- τους ελεγκτές (**controllers**),
- τις προβολές (**views**),
- τον δημόσιο φάκελο αρχείων (**public**),
- και το ενδιάμεσο λογισμικό (**middleware**).



Εικόνα 7 Δομικά μέρη Front-end

Όπως και το Back-end, χρησιμοποιεί και αυτό το κεντρικό αρχείο της εφαρμογής, βιβλιοθήκες/modules/εξαρτήσεις που αποθηκεύονται στον φάκελο **node_modules,** και υπηρεσίες από «κοινόχρηστα» modules του φακέλου **config**. Επίσης, σε αυτή την αρχιτεκτονική, μιας και πρόκειται για Server-Side Rendering (SSR) το Front-end μπορεί να χρησιμοποιεί υπηρεσίες απευθείας από το REST API όπου αυτό χρειαστεί κάτι που προσθέτει ασφάλεια σε ορισμένα μέρη της εφαρμογής και ταχύτητά στην ανάπτυξή της.

Τα περισσότερα από τα παραπάνω κομμάτια περιγράφηκαν στο τμήμα του REST-API, εκτός από τις προβολές και τον δημόσιο φάκελο αρχείων.

Προβολές (views):

- Περιέχουν κομμάτια από τη δομή (template) της σελίδας που φαίνεται στον Client.
- Χρησιμοποιούνται από το view engine [51] που έχει οριστεί στον διακομιστή όπως το HBS στην υλοποίηση αυτή.
- Χρησιμοποιούνται για την προβολή των σελίδων από τη πλευρά του διακομιστή (Server-Side Rendering).

Δημόσιος φάκελος αρχείων (public)

- Χρησιμοποιείται από το Express.js Framework, πιο συγκεκριμένα από το ενδιάμεσο λογισμικό του Express το express.static [52] για την προβολή στατικών αρχείων. όπως εικόνες, αρχεία CSS και αρχεία JavaScript.
- Περιέχει εικόνες, αρχεία CSS και αρχεία JavaScript. Γενικότερα περιέχει στατικά αρχεία που χρειάζεται ο client για να προβάλει σωστά το Front-end.

2.6.2 Διαδρομές (routes)

Κάθε διαδρομή του Front-end που διαχειρίζεται HTTP request με μέθοδο GET αντιστοιχεί σε μία προβολή (δηλαδή η μέθοδος της κάθε διαδρομής λειτουργεί σαν ελεγκτής και χρησιμοποιεί απευθείας μία προβολή) ή σε τουλάχιστον ένα ελεγκτή που αυτός με τη σειρά του χρησιμοποιεί μία προβολή.

2.6.2.1 Ανάλυση διαδρομών

Με βάση τις λειτουργικές απαιτήσεις δημιουργήθηκαν οι παρακάτω διαδρομές:

Διαδρομή	Μέθοδος	Περιγραφή
/	GET	Αρχική σελίδα.
/login	GET	Σελίδα σύνδεσης χρήστη.
/login	POST	Σελίδα που δέχεται τα δεδομένα σύνδεσης χρήστη.
/signup	GET	Σελίδα εγγραφής χρήστη.

/signup	POST	Σελίδα που δέχεται τα δεδομένα εγγραφής χρήστη.
/logout	GET	Σελίδα αποσύνδεσης χρήστη.
/admin	GET	Σελίδα διαχείρισης (ταμπλό) χρήστη.
/admin/devices	GET	Σελίδα διαχείρισης συσκευών χρήστη.
/admin/devices/:device	GET	Σελίδα προβολής μετρήσεων συσκευής χρήστη.
/admin/sensors	GET	Σελίδα διαχείρισης αισθητήρων χρήστη.
/admin/apikeys	GET	Σελίδα διαχείρισης κλειδιών API χρήστη.
/admin/settings	GET	Σελίδα ρυθμίσεων λογαριασμού χρήστη.
/admin/super	GET	Σελίδα διαχείρισης (ταμπλό) διαχειριστή.
/admin/super/devices	GET	Σελίδα διαχείρισης συσκευών διαχειριστή.
/admin/super/sensors	GET	Σελίδα διαχείρισης αισθητήρων διαχειριστή.
/admin/super/apikeys	GET	Σελίδα διαχείρισης κλειδιών API διαχειριστή.
/admin/super/measurements	GET	Σελίδα διαχείρισης μετρήσεων διαχειριστή.
/admin/super/users	GET	Σελίδα διαχείρισης χρηστών διαχειριστή.

Πίνακας 7 Διαδρομές Front-end

2.6.2.2 Παράδειγμα δημιουργίας διαδρομών index.js

(Με χρήση ελεγκτή που χρησιμοποιεί υπηρεσίες)

```
// routes/index.js

// Εισαγωγή βιβλιοθηκών
const express = require('express');
const router = express.Router();

// Εισαγωγή ενδιάμεσου λογισμικού
const auth = require('../middlewares/auth');

// Εισαγωγή ελεγκτών
const Controller = require('../controllers/index');

// Δημιουργία διαδρομών

// Μέθοδος διαδρομής αρχικής σελίδας
router.get('/', Controller.Home.getPage);

// Μέθοδος διαδρομής /login
```

```
router.get('/login', auth.hideFromAuthenticated, Controller.Users.userLogin
Page);

// Μέθοδος διαδρομής /login
router.post('/login', auth.hideFromAuthenticated, Controller.Users.userLogi
nAuthentication, Controller.Users.generateToken);

// Μέθοδος διαδρομής /signup
router.get('/signup', auth.hideFromAuthenticated, Controller.Users.userSign
upPage);

// Μέθοδος διαδρομής /signup
router.post('/signup', auth.hideFromAuthenticated, Controller.Users.userSig
nup, Controller.Users.generateToken);

// Μέθοδος διαδρομής /logout
router.get('/logout', auth.authenticationCheck, Controller.Users.userLogout
);

// Εισαγωγή διαδρομών από admin.js (ταμπλό διαχείρισης χρήστη)
const adminRouter = require('./admin');
router.use('/admin', adminRouter);

// Εξαγωγή των διαδρομών στο express router
module.exports = router;
```

2.6.2.3 Απόσπασμα με μεθόδους των διαδρομών admin.js

(Με απευθείας χρήση υπηρεσίας)

```
// routes/admin.js

// Εισαγωγή βιβλιοθηκών
const express = require('express');
const router = express.Router();

// Εισαγωγή ενδιάμεσου λογισμικού
const auth = require('../middlewares/auth');

// Εισαγωγή υπηρεσιών
const Service = require('../api/services/index');

// Μέθοδος διαδρομής /admin
router.get('/', auth.authenticationCheck, (req, res) => {
    res.render('admin/index', { // Απάντηση/επιστροφή με προβολή
        title: "Dashboard", // Πεδίο που εισέρχεται στη προβολή
        username: req.user.username, // Πεδίο που εισέρχεται στη προβολή
        user_id: req.user.id, // Πεδίο που εισέρχεται στη προβολή
    });
});
```

```
// Μέθοδος διαδρομής /admin/devices
router.get('/devices', auth.authenticationCheck, async (req, res) => {
    const devices = await Service.Device.getAllDevicesOfUser(req.user.usern
ame); // Χρήση υπηρεσίας Device
    const devicetypes = await Service.Device.getAllDeviceTypes(); // Χρήση
υπηρεσίας Device
    res.render('admin/devices', { // Απάντηση/επιστροφή με προβολή
        title: "Devices", // Πεδίο που εισέρχεται στη προβολή
        username: req.user.username, // Πεδίο που εισέρχεται στη προβολή
        user_id: req.user.id, // Πεδίο που εισέρχεται στη προβολή
        data: devices, // Πεδίο που εισέρχεται στη προβολή
        devicetypes: devicetypes // Πεδίο που εισέρχεται στη προβολή
    });
});

// Μέθοδος διαδρομής /admin/devices/:device
router.get('/devices/:device', auth.authenticationCheck, async (req, res) =
> {
    const devices = await Service.Device.getAllDevicesOfUser(req.user.usern
ame); // Χρήση υπηρεσίας Device
    const devicetypes = await Service.Device.getAllDeviceTypes(); // Χρήση
υπηρεσίας Device
    res.render('admin/device', { // Απάντηση/επιστροφή με προβολή
        title: "Devices / " + req.params.device, // Πεδίο που εισέρχεται στ
η προβολή
        username: req.user.username, // Πεδίο που εισέρχεται στη προβολή
        user_id: req.user.id, // Πεδίο που εισέρχεται στη προβολή
        data: devices, // Πεδίο που εισέρχεται στη προβολή
        devicetypes: devicetypes, // Πεδίο που εισέρχεται στη προβολή
        deviceid: req.params.device // Πεδίο που εισέρχεται στη προβολή
    });
});

// Μέθοδος διαδρομής /admin/settings
router.get('/settings', auth.authenticationCheck, (req, res) => {
    res.render('admin/settings', { // Απάντηση/επιστροφή με προβολή
        title: "Settings", // Πεδίο που εισέρχεται στη προβολή
        username: req.user.username, // Πεδίο που εισέρχεται στη προβολή
        user_id: req.user.id, // Πεδίο που εισέρχεται στη προβολή
    });
});

// Δήλωση εμφωλευμένων διαδρομών (διαχειριστή)
const administratorsRouter = require('./super');
router.use('/super', administratorsRouter);

// Εξαγωγή των διαδρομών στο express router
module.exports = router;
```

2.6.3 Ελεγκτές (controllers)

Με βάση τις παραπάνω διαδρομές, δημιουργήθηκαν ελεγκτές για τις λειτουργίες της αυθεντικοποίησης του χρήστη.

2.6.3.1 Απόσπασμα με μεθόδους του ελεγκτή admin.js

```
// controllers/users.js

// Εισαγωγή βιβλιοθηκών
const db = require('../config/database');
const passport = require('passport');
const bcrypt = require('bcrypt');

// Εισαγωγή υπηρεσιών
const Service = require('../api/services/index');

// Μέθοδος εγγραφής χρήστη
exports.userSignup = function (req, res, next) {

    // Έλεγχος πεδίων
    req.checkBody('username', 'Username field cannot be empty.').notEmpty()
;
    req.checkBody('username', 'Username must be between 4-
15 characters long and have no spaces.').len(4, 15);
    req.checkBody('email', 'The email you entered is invalid, please try ag
ain.').isEmail();
    req.checkBody('email', 'Email address must be between 4-
100 characters long, please try again.').len(4, 100);
    req.checkBody('password', 'Password must be between 8-
100 characters long.').len(8, 100);
    req.checkBody('password', "Password must include one lowercase characte
r, one uppercase character, a number, and a special character.").matches(/^
(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?!.* )(?=.*[^a-zA-Z0-9]).{8,}$/, "i");
    req.checkBody('conf_password', 'Password must be between 8-
100 characters long.').len(8, 100);
    req.checkBody('conf_password', 'Passwords do not match, please try agai
n.').equals(req.body.password);

    const errors = req.validationErrors(); // Συλλογή των σφαλμάτων

    if (errors) { // Εμφάνιση των σφαλμάτων αν υπάρχουν
        res.render('signup', {
            title: 'Registration Error',
            errors: errors
        });
    } else { // Συνέχεια εγγραφής
```

```
        const username = req.body.username;
        const email = req.body.email;
        const password = req.body.password;
        const conf_password = req.body.conf_password;

        const saltRounds = 10;

        bcrypt.hash(password, saltRounds, function (err, hash) { // Κατακερ
ματισμός του κωδικού του χρήστη
            db.query("INSERT INTO  users (username, email, passwd) VALUES (
?, ?, ?)", [username, email, hash], // Εισαγωγή του χρήστη στην βάση δεδομέ
νων
                function (error, results, fields) {
                    if (error) { // Έλεγχος σφαλμάτων από τη βάση δεδομένων
                            throw error;
                    }
                    // Δημιουργία νέας συνεδρίας χρήστη στη βάση δεδομένων
                    db.query('SELECT LAST_INSERT_ID() as id', function (err
or, results, fields) {
                        if (error) throw error;
                        const id = results[0];
                        req.login({ id, username }, function (err) {
                            next();
                        })
                    });
                })
        });
    }
};

// Μέθοδος αποσύνδεσης χρήστη
exports.userLogout = function (req, res) {
    req.logout(); // Αποσύνδεση του χρήστη
    req.session.destroy(); // Καταστροφή της συνεδρίας του χρήστη
    res.clearCookie(process.env.JWT_COOKIE_NAME); // Καταστροφή του httpOnl
y Cookie που περιέχει το JWT Token κλειδί του χρήστη
    res.redirect('/'); // Ανακατεύθνση στην αρχική σελίδα
};

// Μέθοδος δημιουργίας httpOnly Cookie για την αποθήκευση του JWT Token κλε
ιδιού του χρήστη
exports.generateToken = async function (req, res, next) {
    const token = await Service.User.generateToken(req.user.username); // Χ
ρήση της υπηρεσίας User για την παραγωγή νέου κλειδιού
    res.cookie(process.env.JWT_COOKIE_NAME, token, { // Δημιουργία νέου Coo
kie
        expires: new Date(Date.now() + process.env.JWT_COOKIE_LIFE), // Ορι
σμός χρόνου ζωής Cookie
```

```
        secure: process.env.NODE_ENV === 'production' ? true : false, // Ορ
ισμός χρήσης ssl
        httpOnly: true // Ορισμός ιδιότητας httpOnly
    })
        .redirect('/admin'); // Ανακατεύθυνση στο ταμπλό διαχείρισης του χρ
ήστη
};
```

2.6.4 Ενδιάμεσο λογισμικό (middleware)

Το ενδιάμεσο λογισμικό που αναπτύχθηκε για το Front-end της εφαρμογής έχει τη μορφή module και αφορά τις λειτουργίες της συνεδρίας και της αυθεντικοποίησης των χρηστών.

2.6.4.1 Απόσπασμα με μεθόδους του ενδιάμεσου λογισμικού auth.js

```
// middlewares/auth.js

// Εισαγωγή βιβλιοθηκών
const passport = require('passport');
const LocalStrategy = require('passport-local').Strategy;
const db = require('../config/database');
const bcrypt = require('bcrypt');

// Μέθοδος δημιουργίας νέας συνεδρίας χρήστη
passport.serializeUser(function (idAndUsernameAndRole, done) {
    done(null, idAndUsernameAndRole);
});

// Μέθοδος διαγραφής της συνεδρίας του χρήστη
passport.deserializeUser(function (idAndUsernameAndRole, done) {
    done(null, idAndUsernameAndRole);
});

// Δημιουργία νέας στρατηγικής αυθεντικοποίησης για τη βιβλιοθήκη Passport
passport.use('local.login', new LocalStrategy(
    function (username, password, done) {
        db.query('SELECT UID, PASSWD, ROLE FROM users WHERE USERNAME = ?',
[username],
            function (err, results, fields) {
                if (err) {
                    done(err);
                };
                if (results.length === 0) {
                    return done(null, false, { message: 'Authentication fai
led!' });
                } else {
                    const hash = results[0].PASSWD.toString();
```

```
                    bcrypt.compare(password, hash, function (err, response)
 {
                        if (response === true) {
                            const id = results[0].ID;
                            const role = results[0].ROLE;
                            return done(null, { id, username, role });
                        } else {
                            return done(null, false, { message: 'Authentica
tion failed!' });
                        }
                    });
                }
            })
    }
));
```

2.6.5 Προβολές (views)

Για τη κατασκευή των προβολών χρησιμοποιήθηκαν εργαλεία και τεχνολογίες όπως η βιβλιοθήκη HBS σαν view engine του Express.js, το AdminLTE σαν βασικό πρότυπο εμφάνισης των σελίδων διαχείρισης και η βιβλιοθήκη Axios για την ανάκτηση δεδομένων από το Back-end σε πραγματικό χρόνο.

Αρχικά έγινε μετατροπή της δομής του πρότυπου AdminLTE σε συμβατότητα με το HBS view engine και το Handlebars.js (δηλαδή ο χωρισμός του σε κομμάτια «partials» τύπου .hbs) και μεταφορά των στατικών αρχείων και βιβλιοθηκών του στο φάκελο δημόσιου υλικού (public) ώστε οι σελίδες να έχουν πρόσβαση σε αυτά κατά τη διαδικασία του rendering. Έπειτα κατασκευάστηκαν με στοιχεία (components) από το AdminLTE οι προβολές που αντιπροσωπεύουν τις σελίδες/διαδρομές του Front-end και υλοποιήθηκαν Promise-based μέθοδοι σε JavaScript με τη χρήση της βιβλιοθήκης Axios για την ανάκτηση (fetching) των δεδομένων κάθε σελίδας.

2.6.5.1 Απόσπασμα με μεθόδους της προβολής devices.hbs

```
<!— Εισαγωγή της βιβλιοθήκης Axios -->
<script src="/plugins/axios/axios.min.js"></script>

<script>

  // Μέθοδος ανάκτησης δεδομένων από το rest-api
  function getData(api_url, timeout) {
    return axios.get(api_url, { withCredentials: true })
```

```
      .then(response => {
        return response;
      })
      .catch((error) => {
        console.log('Error: ' + error);
        // Αν δε βρεθούν δεδομένα τότε μηδενίζεται η μεταβλητή data
        if (error.response.status === 404) {
          const response = {
            "data": 0
          }
          return response;
        }
      });
  }

  // Δήλωση διαδρομής του rest-api
  const api_url = "{{ sitehost }}/api/v1";

  // Δήλωση διαδρομής του χρήστη και λήψη δεδομένων του χρήστη
  const userData = await getData(api_url + "/users/{{ username }}");

        // Μέθοδος διαγραφής μιας συσκευής
        async function deleteDevice(_username, _deviceid) {
            await axios.delete(api_url + '/users/' + _username + '/devices/
' + _deviceid, { withCredentials: true })
                .then(response => {
                    location.reload();
                })
                .catch((error) => {
                    console.log('error ' + error.response.data);
                });
        }

      // Μέθοδος επεξεργασίας των στοιχείων μιας συσκευής
      async function editDeviceInfo(_username, _deviceid, _devicename, _dev
icedescription) {
            // Αφαίρεση προηγούμενων σφαλμάτων
            $('#modal_default_edit_error').html('');

            // Δήλωση της φόρμας
            const edit_form = document.querySelector('form[data-
form="edit_form"]');

            // Προ-εμφάνιση των δεδομένων στα πεδία
            edit_form.edit_device_name.value = _devicename;
            edit_form.edit_device_desc.value = _devicedescription;

            // Έλεγχος του event αποστολής
            edit_form.addEventListener('submit', async (e) => {
```

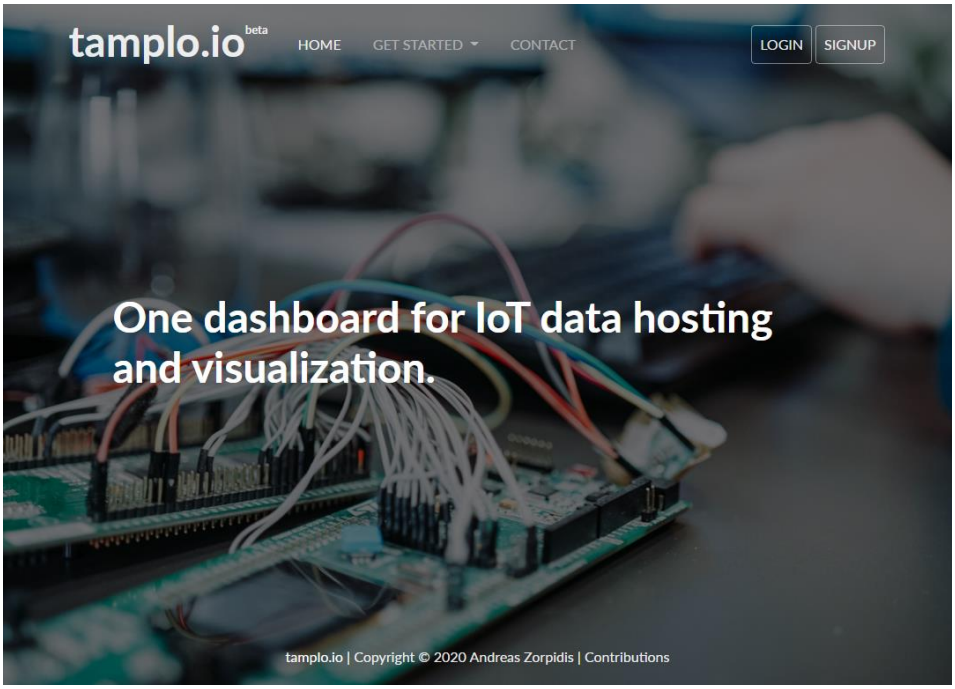
```
                // Απενεργοποίηση της προκαθορισμένης λειτουργίας της
φόρμας
                e.preventDefault();

                // Εισαγωγή δεδομένων
                const devicename = edit_form.edit_device_name.value;
                const devicedescription = edit_form.edit_device_desc.value;

                // Δημιουργία αντικειμένου δεδομένων για αποστολή
                const postdata = {
                    "devicename": devicename,
                    "devicedescription": devicedescription
                };

                await axios.put(api_url + '/users/' + _username + '/devices
/' + _deviceid, postdata, { withCredentials: true })
                    .then(response => {
                        location.reload();
                    })
                    .catch((error) => {
                        $('#modal_default_edit_error').html('<div class="al
ert alert-
danger"><pre>' + JSON.stringify(error.response.data, null, 2) + '</pre></di
v>');
                    });
            })
        }

      // Μέθοδος προσθήκης νέας συσκευής

      async function addDevice() {

            // Αφαίρεση προηγούμενων σφαλμάτων
            $('#modal_default_add_error').html('');

            const add_form = document.querySelector('form[data-
form="add_form"]');

            add_form.addEventListener('submit', async (e) => {
                e.preventDefault();

                // Εισαγωγή δεδομένων
                const deviceid = add_form.add_device_id.value;
                const devicename = add_form.add_device_name.value;
                const devicedescription = add_form.add_device_desc.value;
                const devicetype = add_form.add_device_type.value;
                const username = add_form.add_device_username.value;

                // Δημιουργία αντικειμένου δεδομένων για αποστολή
```

```
                const postdata = {
                    "deviceid": deviceid,
                    "devicename": devicename,
                    "devicedescription": devicedescription,
                    "devicetype": devicetype,
                    "username": username
                };

                await axios.post(api_url + '/users/' + username + '/devices
', postdata, { withCredentials: true })
                    .then(response => {
                        location.reload();
                    })
                    .catch((error) => {
                        $('#modal_default_add_error').html('<div class="ale
rt alert-
danger"><pre>' + JSON.stringify(error.response.data, null, 2) + '</pre></di
v>');
                    });
            })
        }
</script>
```

2.7 Φωτογραφίες από την εφαρμογή

2.7.1 Αρχική σελίδα

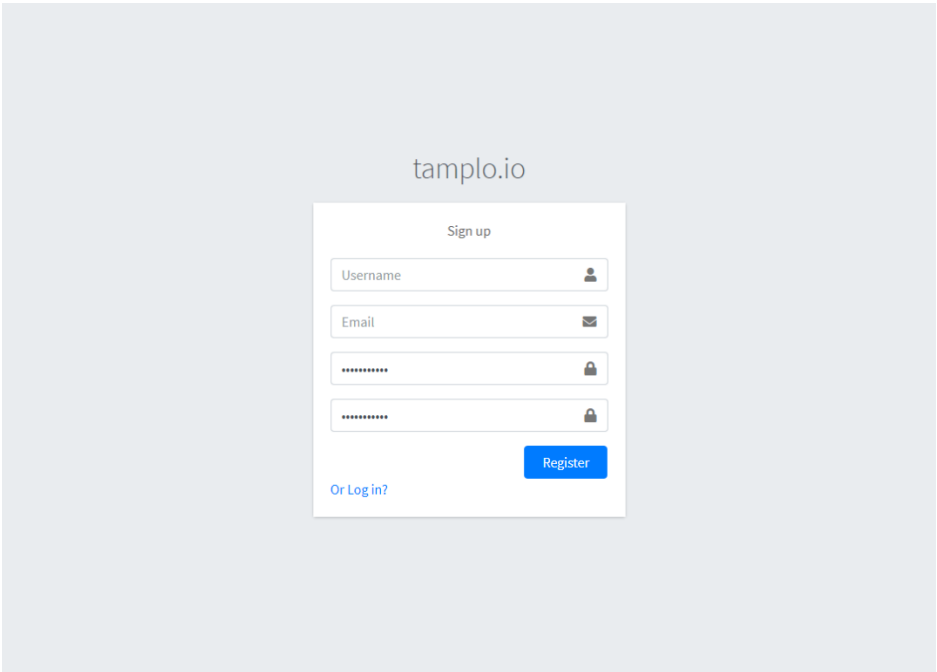
Προβολή: index.hbs



Εικόνα 8 Αρχική σελίδα

2.7.2 Σελίδα εγγραφής

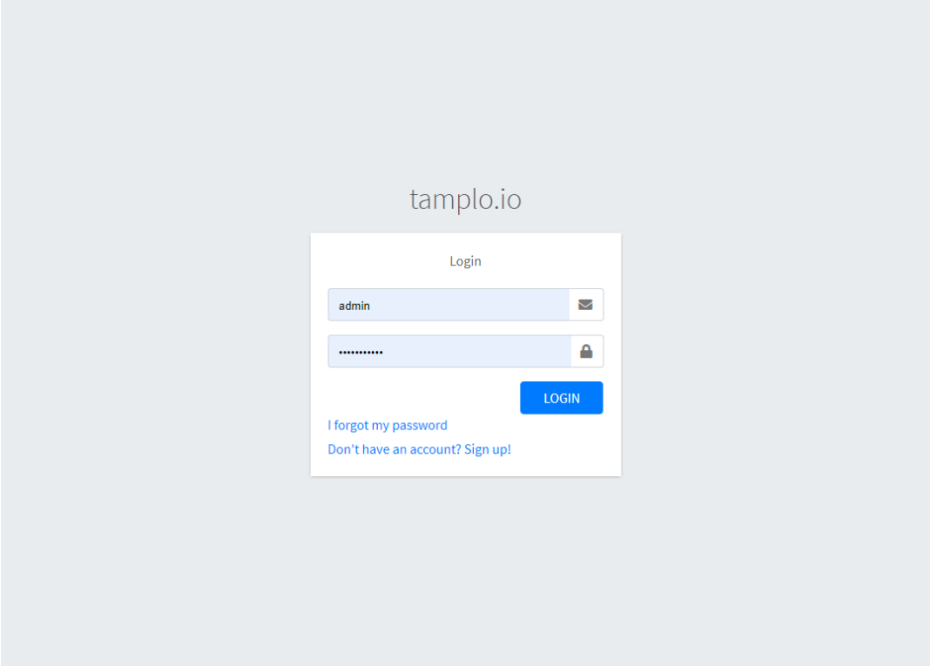
Προβολή: signup.hbs



Εικόνα 9 Σελίδα εγγραφής (χρήστη)

2.7.3 Σελίδα σύνδεσης

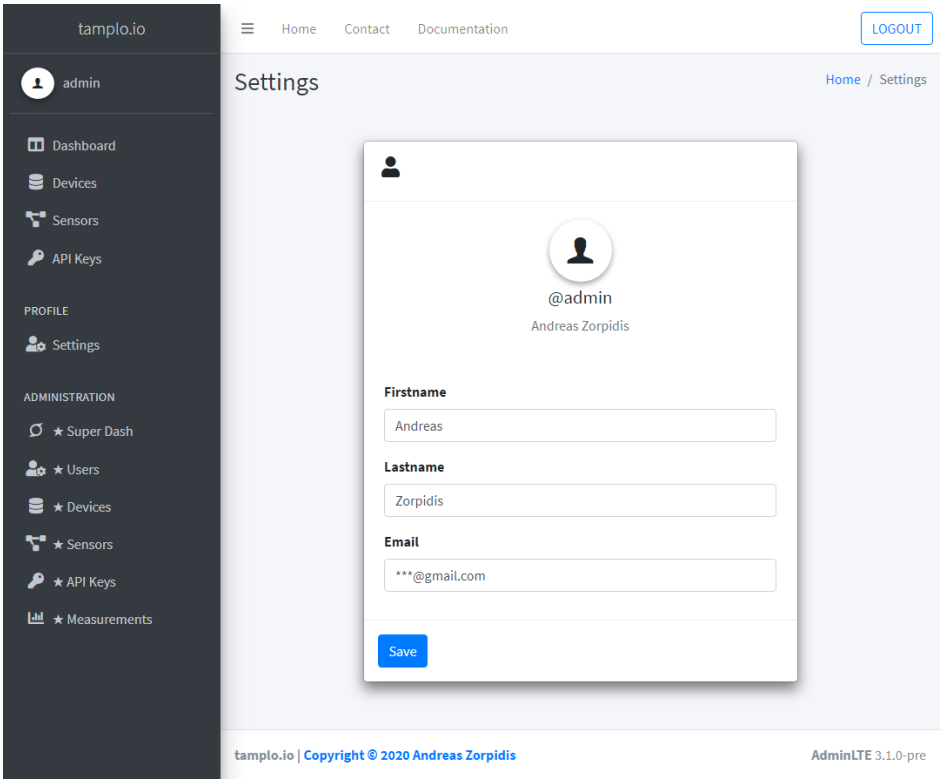
Προβολή: sign.hbs



Εικόνα 10 Σελίδα σύνδεσης (χρήστη)

2.7.4 Σελίδα ρυθμίσεων

Προβολή: admin/settings.hbs

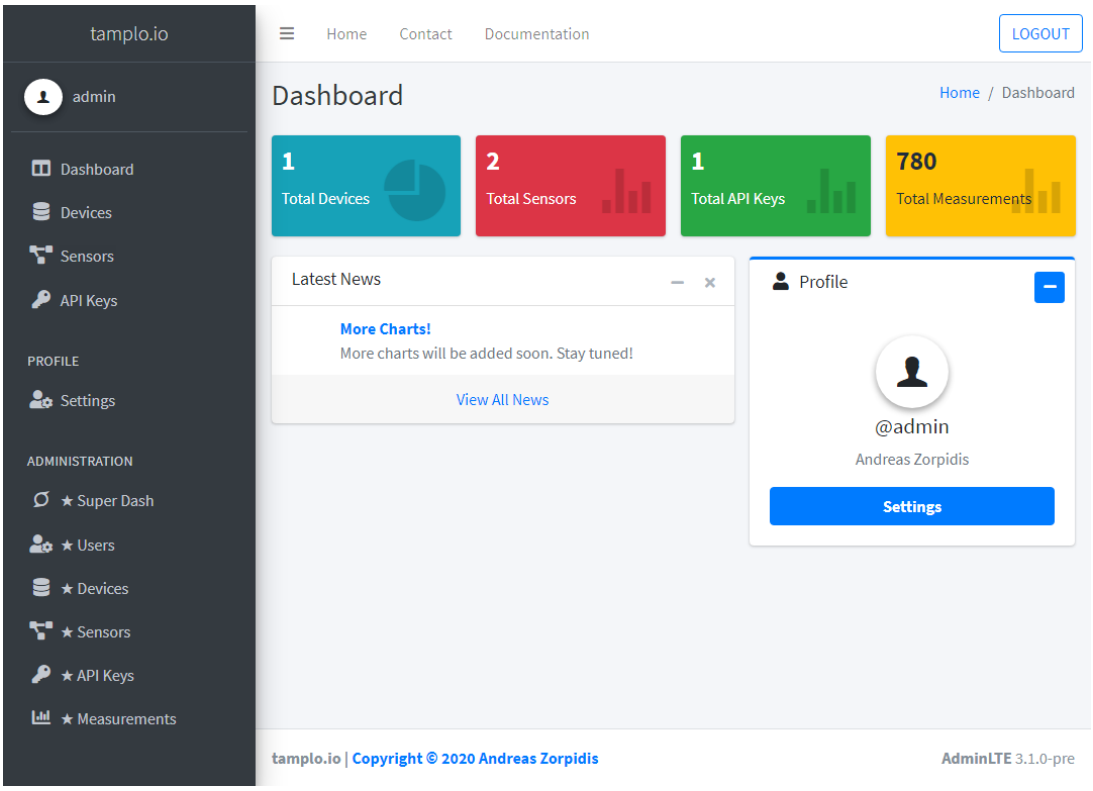


Εικόνα 11 Σελίδα ρυθμίσεων προφίλ (χρήστη)

2.7.5 Σελίδες χρήστη

2.7.5.1 Αρχικό ταμπλό

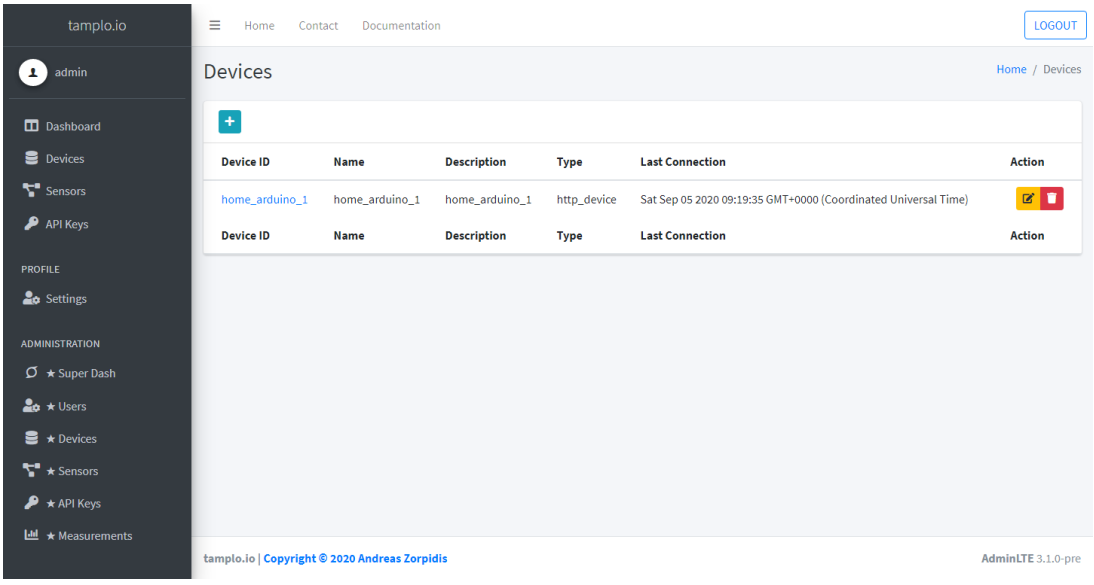
Προβολή: admin/index.hbs



Εικόνα 12 Αρχικό ταμπλό (χρήστη)

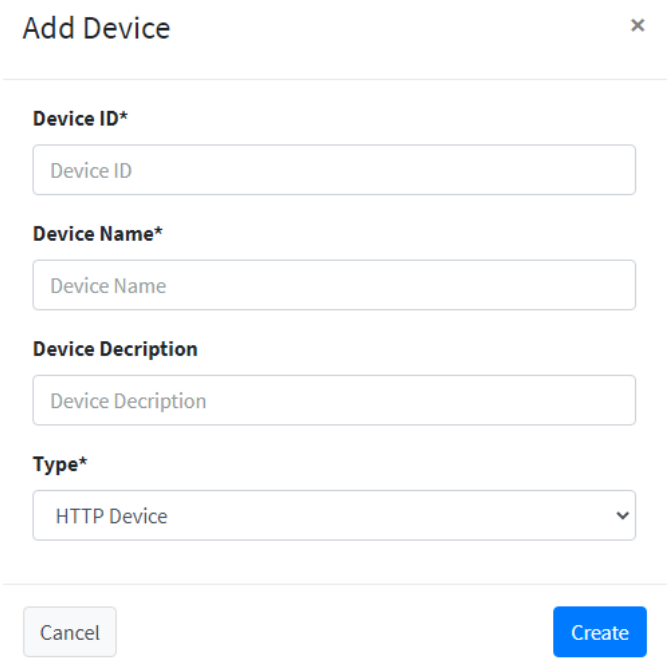
2.7.5.2 Συσκευές

Προβολή: admin/devices.hbs



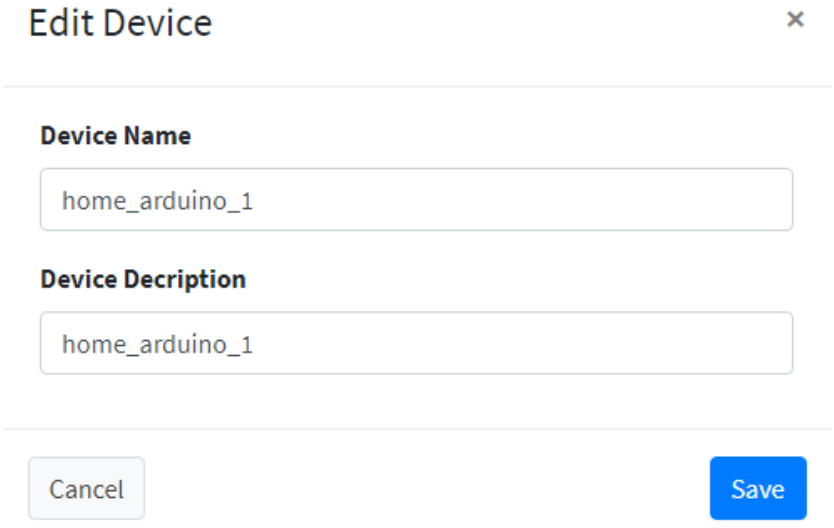
Εικόνα 13 Σελίδα διαχείρισης συσκευών (χρήστη)

2.7.5.2.1 Προσθήκη συσκευής



Εικόνα 14 Pop-up modal για τη προσθήκη συσκευής

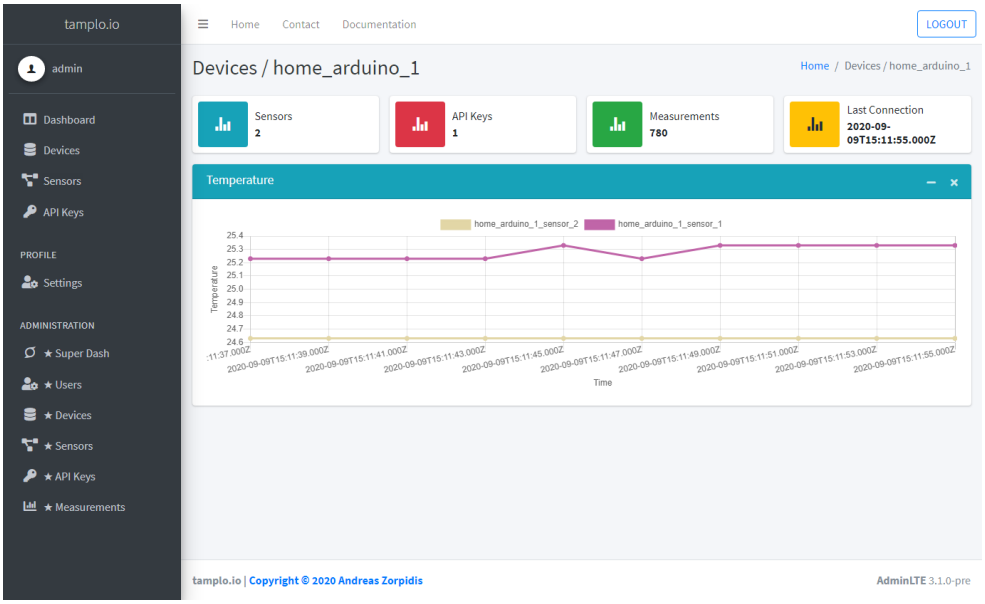
2.7.5.2.2 Επεξεργασία συσκευής



Εικόνα 15 Pop-up modal για την επεξεργασία συσκευής

2.7.5.2.3 Μετρήσεις συσκευής

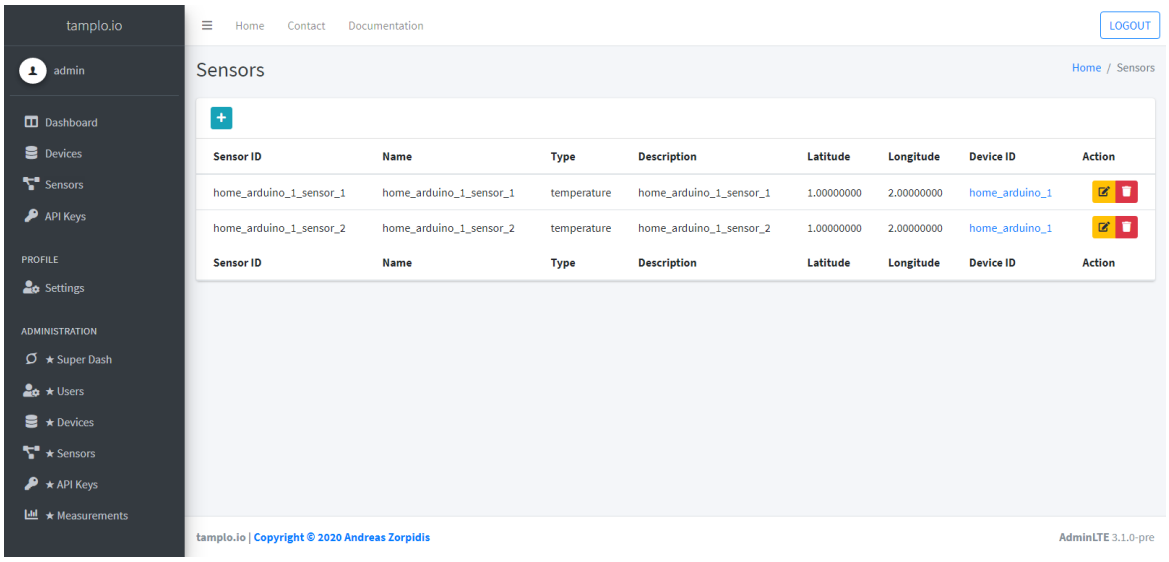
Προβολή: admin/device.hbs



Εικόνα 16 Σελίδα μετρήσεων συσκευής (χρήστη)

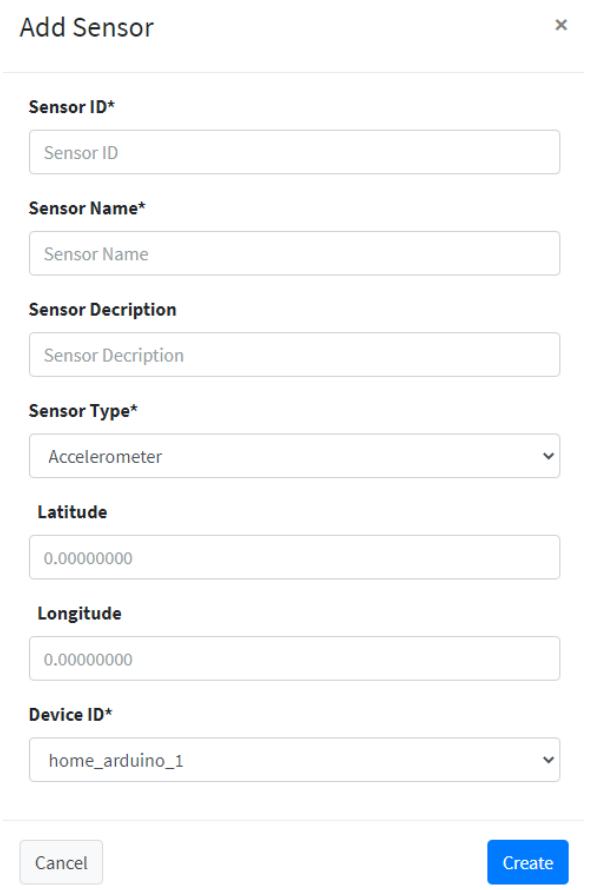
2.7.5.3 Αισθητήρες

Προβολή: admin/sensors.hbs



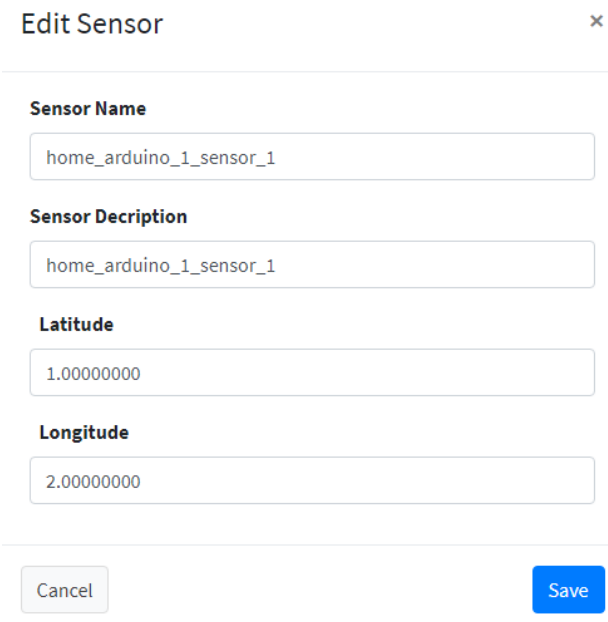
Εικόνα 17 Σελίδα διαχείρισης αισθητήρων (χρήστη)

2.7.5.3.1 Προσθήκη αισθητήρα



Εικόνα 18 Pop-up modal για τη προσθήκη αισθητήρα

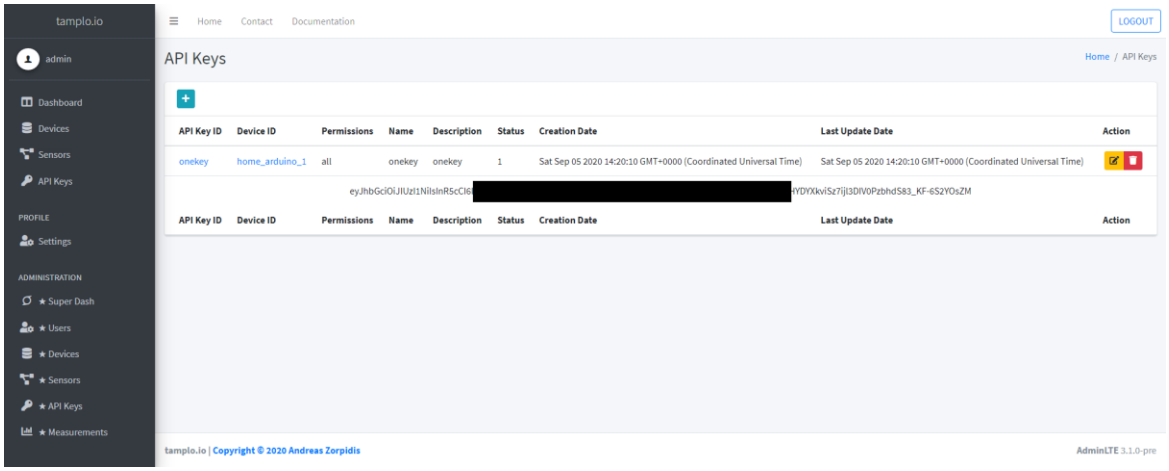
2.7.5.3.2 Επεξεργασία αισθητήρα



Εικόνα 19 Pop-up modal για την επεξεργασία αισθητήρα

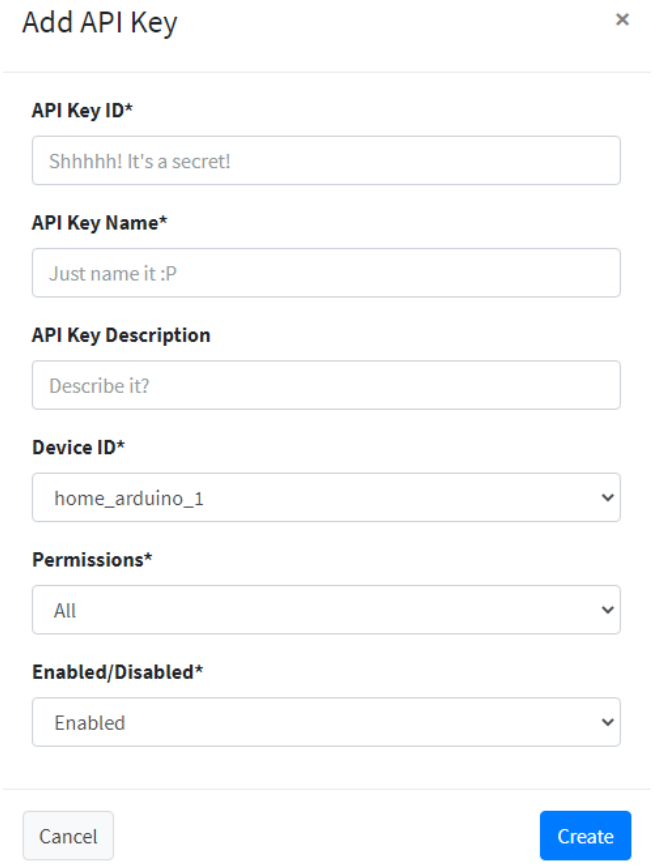
2.7.5.4 Κλειδιά API

Προβολή: apikeys.hbs



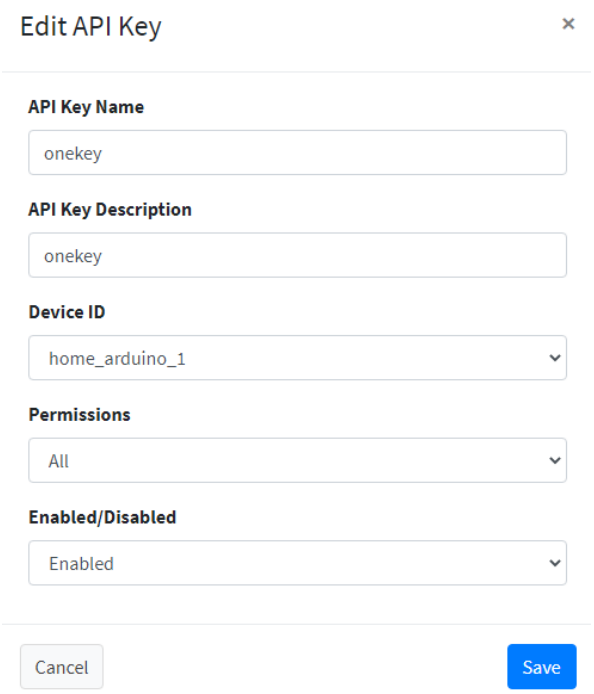
Εικόνα 20 Σελίδα διαχείρισης κλειδιών API (χρήστη)

2.7.5.4.1 Προσθήκη κλειδιού



Εικόνα 21 Pop-up modal για τη προσθήκη κλειδιού API

2.7.5.4.2 Επεξεργασία κλειδιού

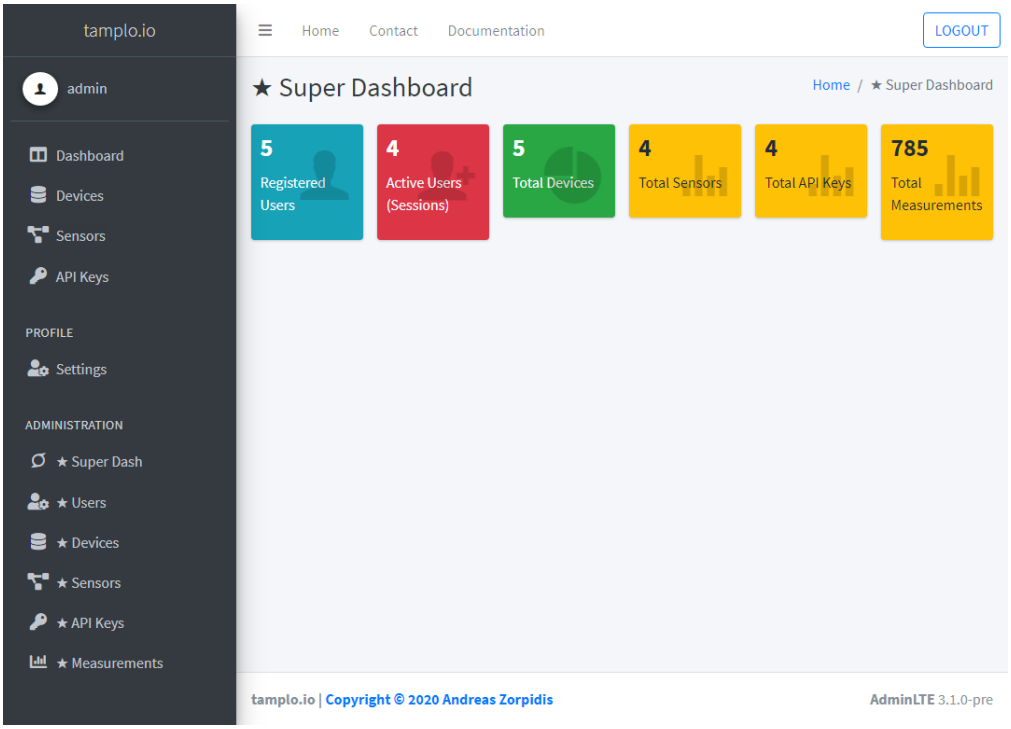


Εικόνα 22 Pop-up modal για την επεξεργασία κλειδιού API

2.7.6 Σελίδες διαχειριστή

2.7.6.1 Αρχικό ταμπλό

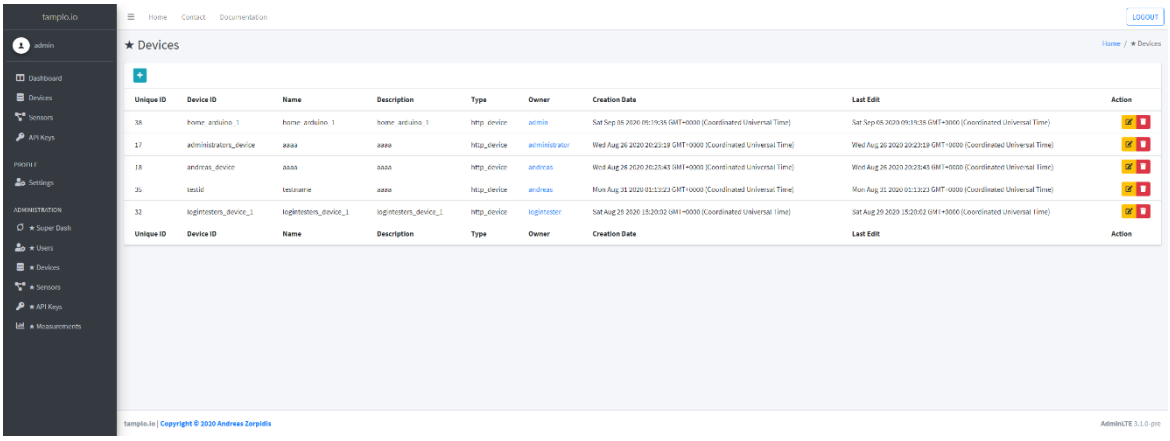
Προβολή: admin/super/index.hbs



Εικόνα 23 Αρχικό ταμπλό (διαχειριστή)

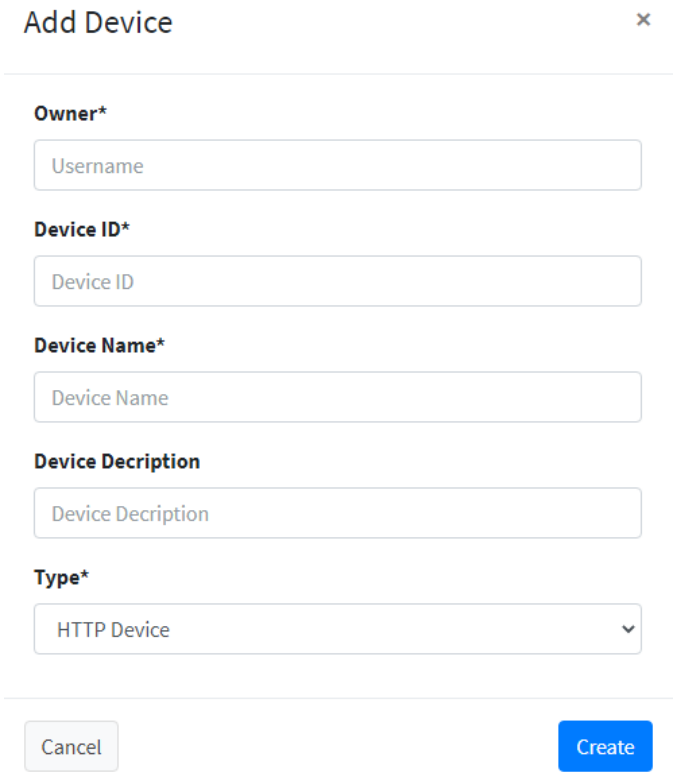
2.7.6.2 Συσκευές

Προβολή: admin/super/devices.hbs



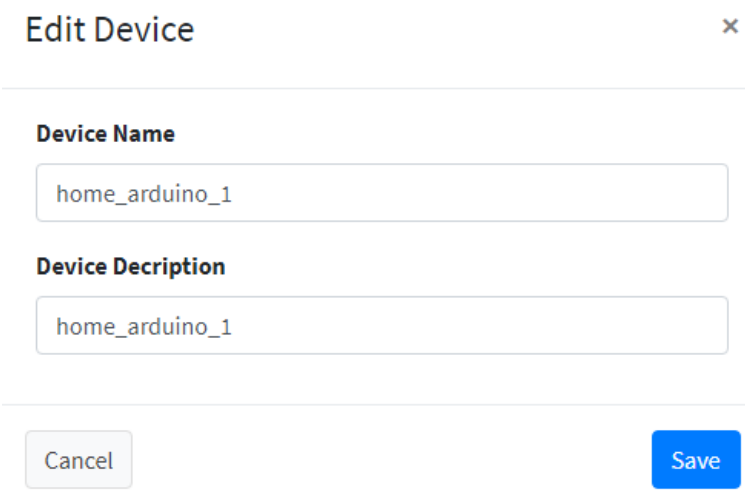
Εικόνα 24 Σελίδα διαχείρισης συσκευών (διαχειριστή)

2.7.6.2.1 Προσθήκη συσκευής



Εικόνα 25 Pop-up modal για τη προσθήκη συσκευής

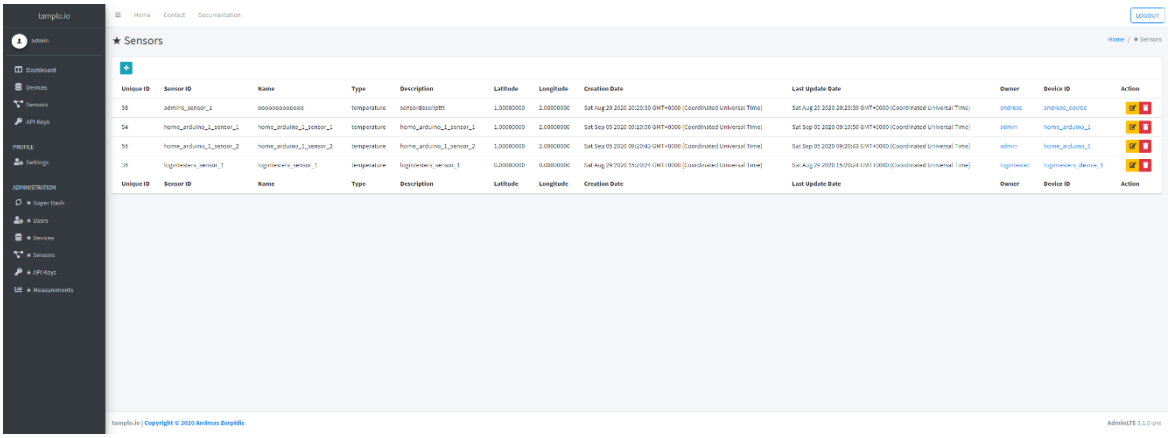
2.7.6.2.2 Επεξεργασία συσκευής



Εικόνα 26 Pop-up modal για την επεξεργασία συσκευής

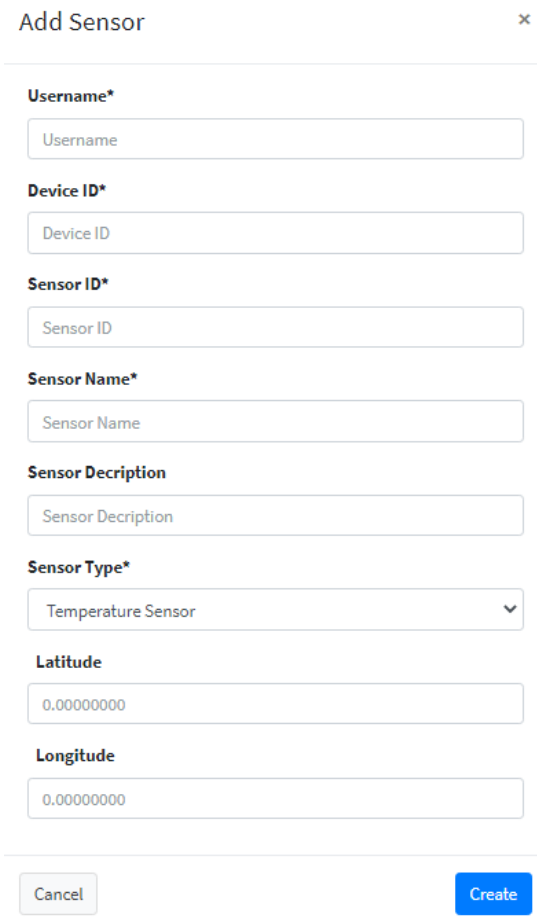
2.7.6.3 Αισθητήρες

Προβολή: admin/super/sensors.hbs



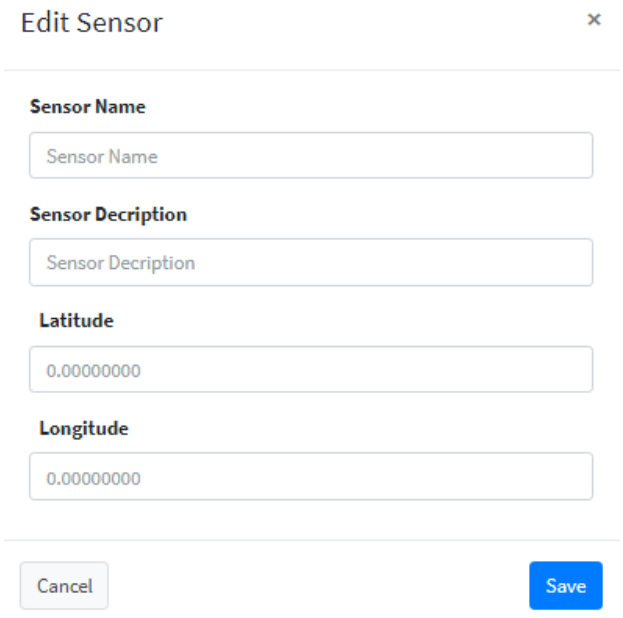
Εικόνα 27 Σελίδα διαχείρισης αισθητήρων (διαχειριστή)

2.7.6.3.1 Προσθήκη αισθητήρα



Εικόνα 28 Pop-up modal για τη προσθήκη αισθητήρα

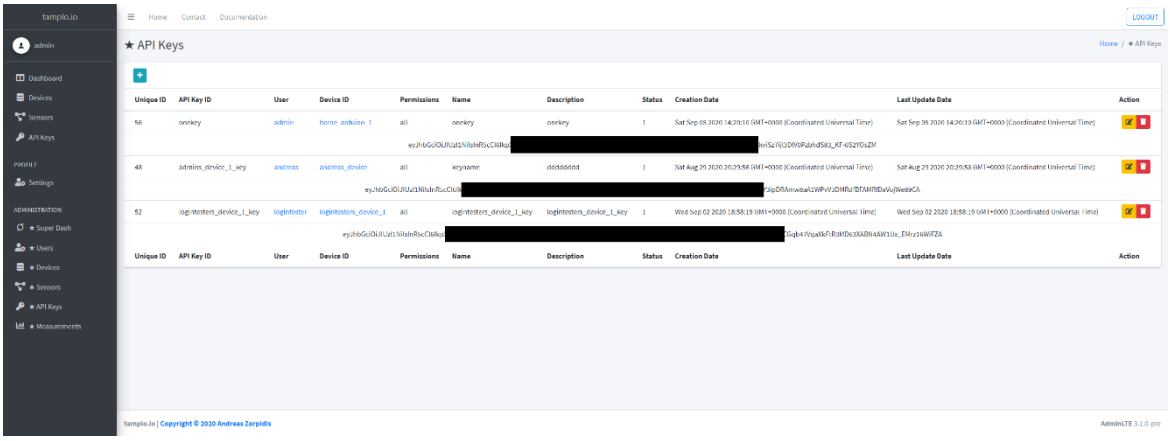
2.7.6.3.2 Επεξεργασία αισθητήρα



Εικόνα 29 Pop-up modal για την επεξεργασία αισθητήρα

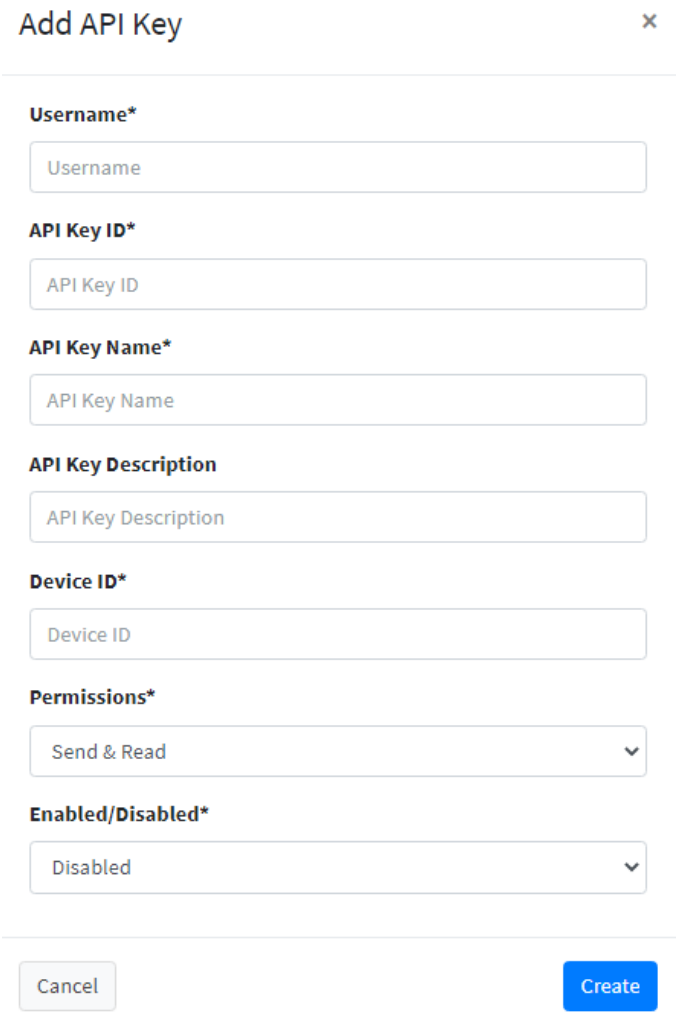
2.7.6.4 Κλειδιά API

Προβολή: admin/super/apikeys.hbs



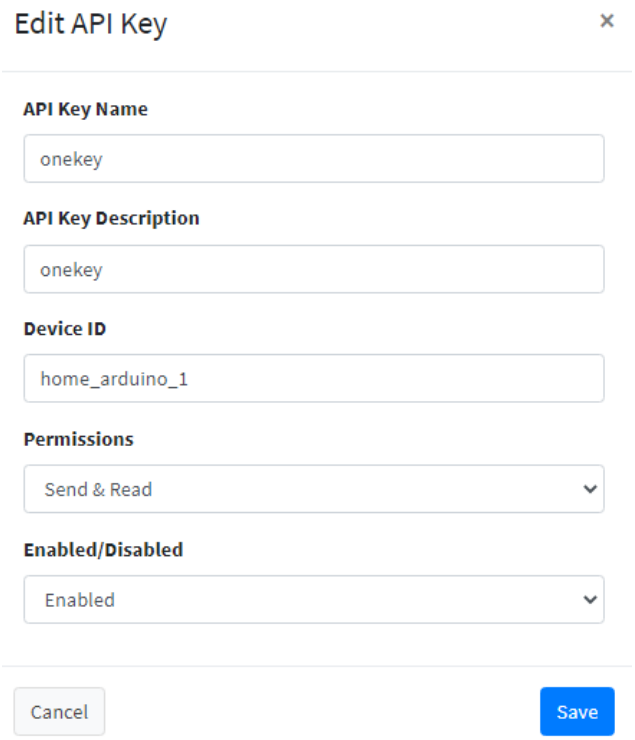
Εικόνα 30 Σελίδα διαχείρισης κλειδιών API (διαχειριστή)

2.7.6.4.1 Προσθήκη κλειδιού



Εικόνα 31 Pop-up modal για τη προσθήκη κλειδιού API

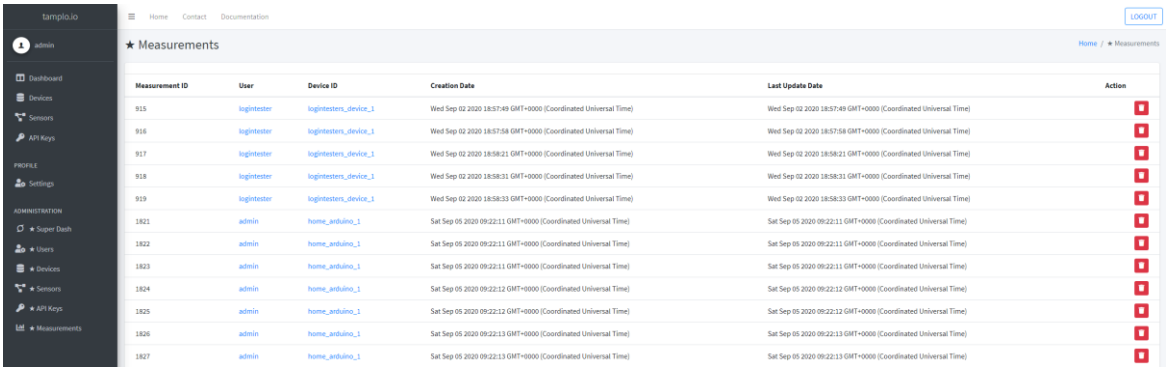
2.7.6.4.2 Επεξεργασία κλειδιού



Εικόνα 32 Pop-up modal για την επεξεργασία κλειδιού API

2.7.6.5 Μετρήσεις

Προβολή: admin/super/measurements.hbs



Εικόνα 33 Σελίδα διαχείρισης μετρήσεων (διαχειριστή)

2.8 Δημοσίευση (deployment) της εφαρμογής στο υπολογιστικό νέφος

2.8.1 Χρήση πλατφόρμας (PaaS) υπολογιστικού νέφους

Η πλατφόρμα ως υπηρεσία (PaaS) [53] είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης και δημοσίευσης εφαρμογών στο υπολογιστικό νέφος με ευέλικτους πόρους. Δηλαδή, όπως το IaaS, το PaaS περιλαμβάνει φυσικό υλικό (διακομιστές, αποθήκευση, δικτύωση), λειτουργικό σύστημα και εργαλεία ανάπτυξης λογισμικού με βάση τις ανάγκες του πληροφοριακού συστήματος. Το PaaS έχει σχεδιαστεί για να υποστηρίζει τον πλήρη κύκλο ζωής της εφαρμογής ιστού: δημιουργία, δοκιμή, ανάπτυξη, διαχείριση και ενημέρωση. Έτσι η δημοσίευση και η συντήρηση των εφαρμογών γίνεται γρήγορα και με χαμηλό κόστος.

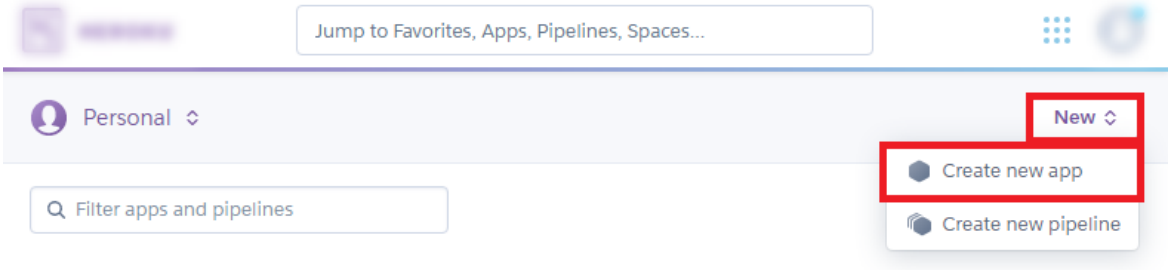
Μερικοί γνωστοί πάροχοι PaaS υπηρεσιών είναι:

- Alibaba Cloud
- Amazon Web Services
- AppScale
- Google App Engine
- Heroku
- IBM Cloud
- Microsoft Azure
- Netlify
- Oracle Cloud

Στην εργασία αυτή έγινε χρήση των υπηρεσιών του Heroku. Το Heroku παρέχει τις υπηρεσίες του δωρεάν σε νέα μικρά projects, παρέχει ένα ολοκληρωμένο documentation, η διαδικασία του deployment είναι εύκολη και μπορεί αργότερα να γίνει scaling (κλιμάκωση) της εφαρμογής σε άλλους υπολογιστικούς πόρους που προσφέρει, γρήγορα και δίχως να βγει εκτός λειτουργίας η εφαρμογή.

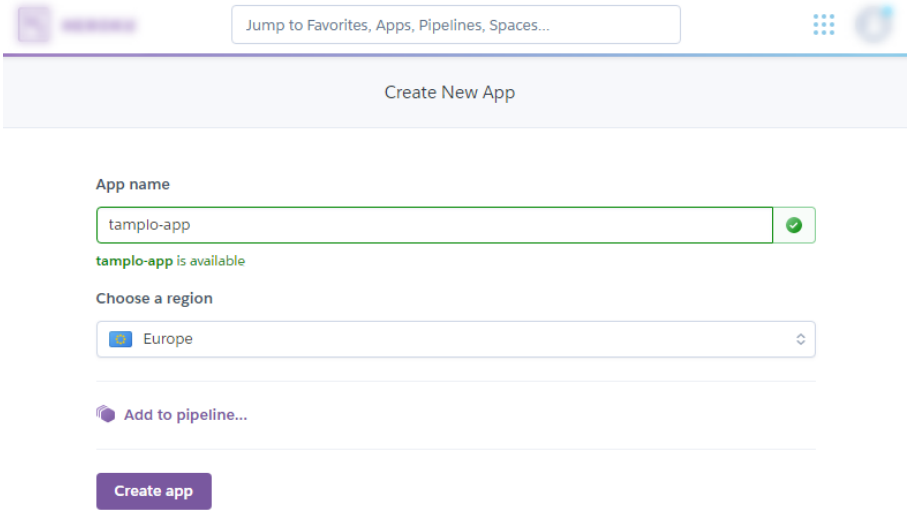
Αφού λοιπόν δημιουργήθηκε ένας προσωπικός λογαριασμός στη πλατφόρμα του Heroku, ακολουθήθηκαν τα παρακάτω βήματα για τη δημιουργία μίας νέας εφαρμογής στη πλατφόρμα:

1) Στη τη σελίδα διαχείρισης (New->Create new app).



Εικόνα 34 Heroku Platform δημιουργία εφαρμογής – βήμα 1

2) Δόθηκε η ονομασία της εφαρμογής και επιλέχθηκε η επιθυμητή περιοχή την οποία αφορά περισσότερο η εφαρμογή.

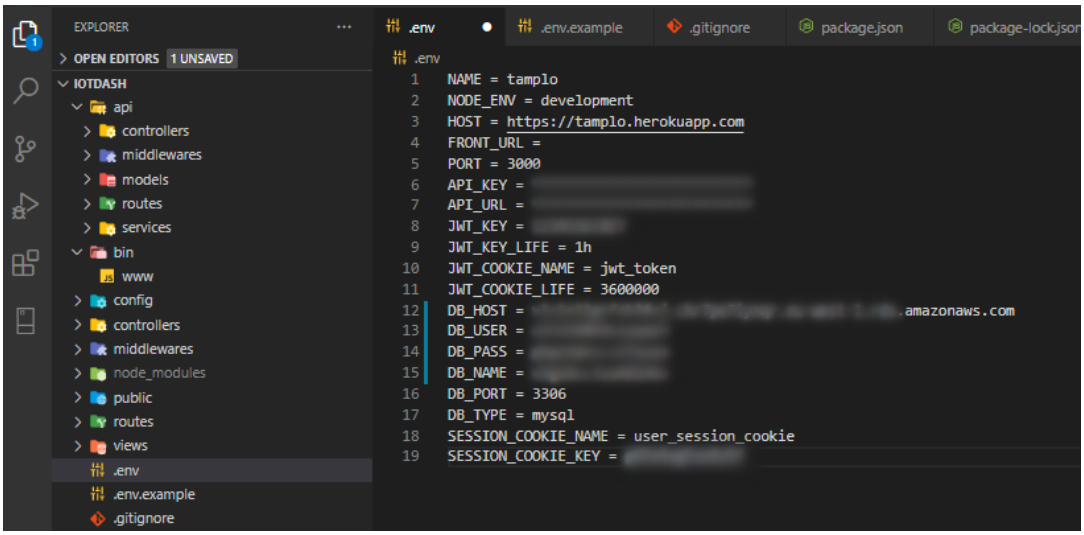


Εικόνα 35 Heroku Platform δημιουργία εφαρμογής - βήμα 2

2.8.2 Προετοιμασία και πρώτη δημοσίευση

Με τη δημιουργία της εφαρμογής στη πλατφόρμα του Heroku, δημιουργήθηκε μία νέα διεύθυνση sub domain του Heroku με βάση το όνομα που δόθηκε (στη περίπτωση αυτή είχε τη μορφή «https://tamplo.herokuapp.com/») για χρήση αποκλειστικά από αυτή την εφαρμογή.

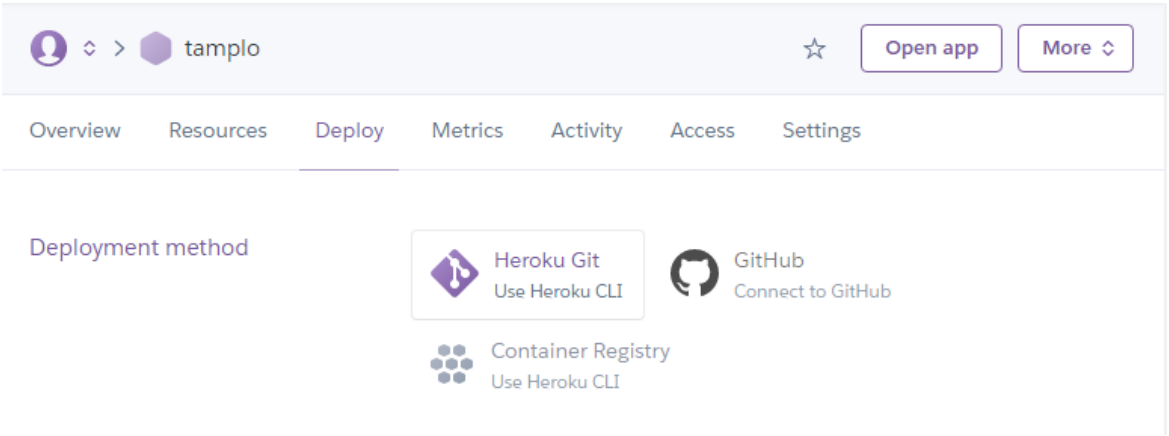
Έπειτα, έπρεπε να ενημερωθούν οι στατικές μεταβλητές της εφαρμογής στο αρχείο app.js και οι μεταβλητές περιβάλλοντος στο αρχείο .env που περιέχουν τη διεύθυνση domain name στην οποία λειτουργεί η εφαρμογή ώστε να περιέχουν αυτό το sub domain.



Εικόνα 36 Ενημέρωση αρχείου μεταβλητών περιβάλλοντος

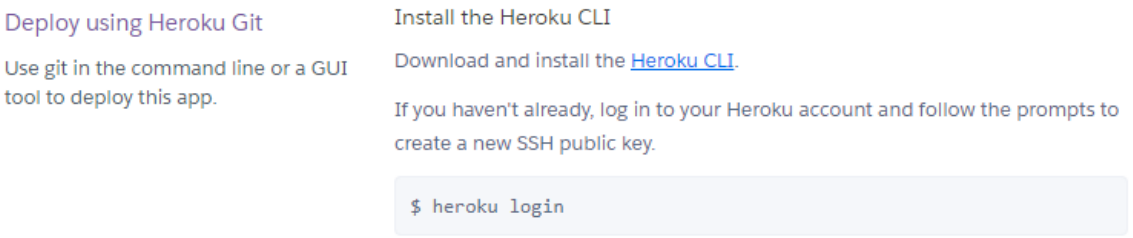
Για να ανέβει στο χώρο του Heroku και να δημοσιευθεί η εφαρμογή έπρεπε να γίνουν επίσης τα παρακάτω βήματα [54] :

1) Να επιλεχθεί μέθοδος δημοσίευσης όπως Heroku Git ή GitHub (σε αυτή τη περίπτωση Heroku Git).



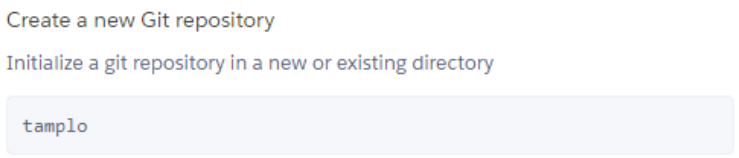
Εικόνα 37 Επιλογή μεθόδου δημοσίευσης

2) Να γίνει εγκατάσταση του Heroku CLI και η σύνδεσή του με τον προσωπικό λογαριασμό του Heroku μέσω της γραμμής εντολών με την εντολή `$ heroku login`



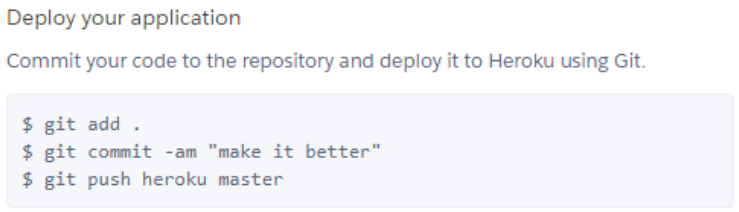
Εικόνα 38 Σύνδεση μέσω Heroku CLI

3) Να γίνει αλλαγή του τρέχοντος καταλόγου εργασίας της γραμμής εντολών στον φάκελο της εφαρμογής.



Εικόνα 39 Φάκελος με τα περιεχόμενα της εφαρμογής

4) Και τέλος η δημοσίευση της μέσω της γραμμής εντολών με τις παρακάτω εντολές.



Εικόνα 40 Εντολές για τη δημοσίευση της εφαρμογής

2.8.3 Δημιουργία βάσης δεδομένων σε υπηρεσία (DBaaS)

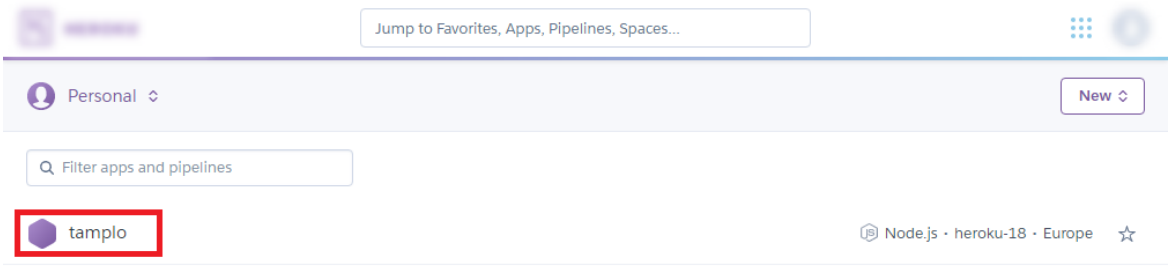
Η Database as a service (DBaaS εν συντομία) [55] είναι μια υπηρεσία υπολογιστικού νέφους που επιτρέπει στους χρήστες να δημιουργούν και να χρησιμοποιούν περιβάλλοντα βάσεων δεδομένων cloud χωρίς να χρειάζεται η αγορά υλικού, η εγκατάσταση λογισμικού και η ρύθμισή του από τους ίδιους για αυτό το σκοπό. Οι υπηρεσίες αυτές συνήθως παρέχουν διάφορα χρηματικά πακέτα με βάση το μέγεθος της βάσης δεδομένων του πελάτη

και τον υπολογιστικό φόρτο που δέχεται η υπηρεσία. Ως αποτέλεσμα, τα πλεονεκτήματα για τον πελάτη μίας τέτοιας υπηρεσίας είναι ευελιξία, η πληθώρα επιλογών και οικονομική εξυπηρέτησή του σε σύντομο χρόνο.

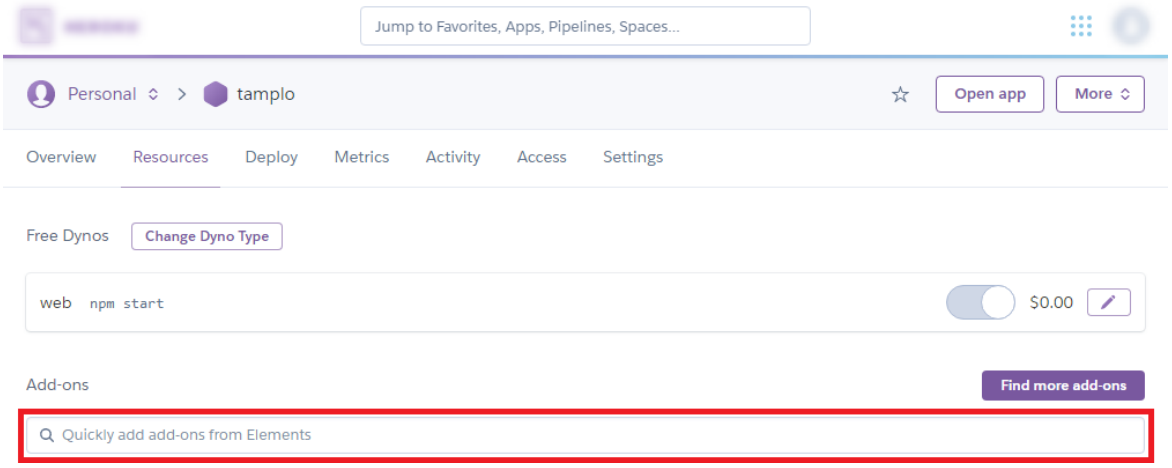
Μερικοί γνωστοί πάροχοι DBaaS υπηρεσιών είναι:

- Oracle
- Amazon (AWS)
- IBM
- Google (Cloud)
- Microsoft (Azure)
- RavenDB
- ClearDB
- JawsDB
- MongoDB Atlas
- Ninox
- Kintone

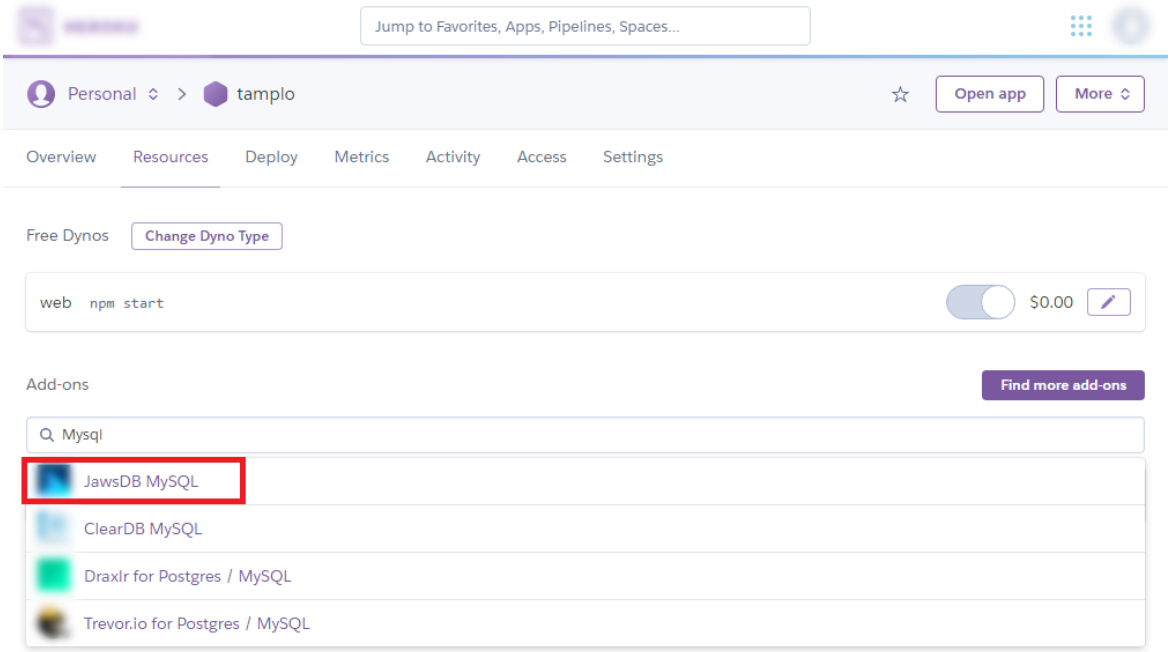
Στην εφαρμογή αυτή έγινε χρήση τη JawsDB η οποία βασίζεται στο DBaaS της Amazon και συνεργάζεται με το Heroku, δηλαδή παρέχεται από το Heroku σαν πρόσθετη λειτουργία [56]. Η προσθήκη της βάσης δεδομένων έγινε εύκολα από τη πλατφόρμα του Heroku κάνοντας αναζήτηση στα πρόσθετα και έπειτα προσθήκη όπως φαίνεται στις φωτογραφίες παρακάτω.



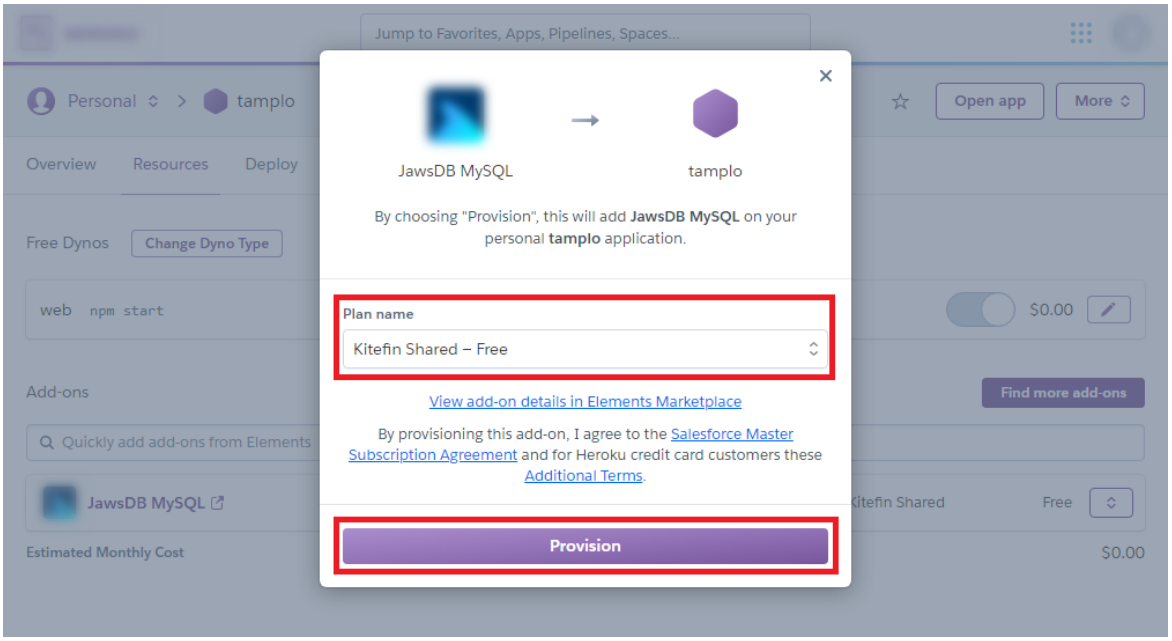
Εικόνα 41 Κεντρική σελίδα διαχείρισης του χρήστη



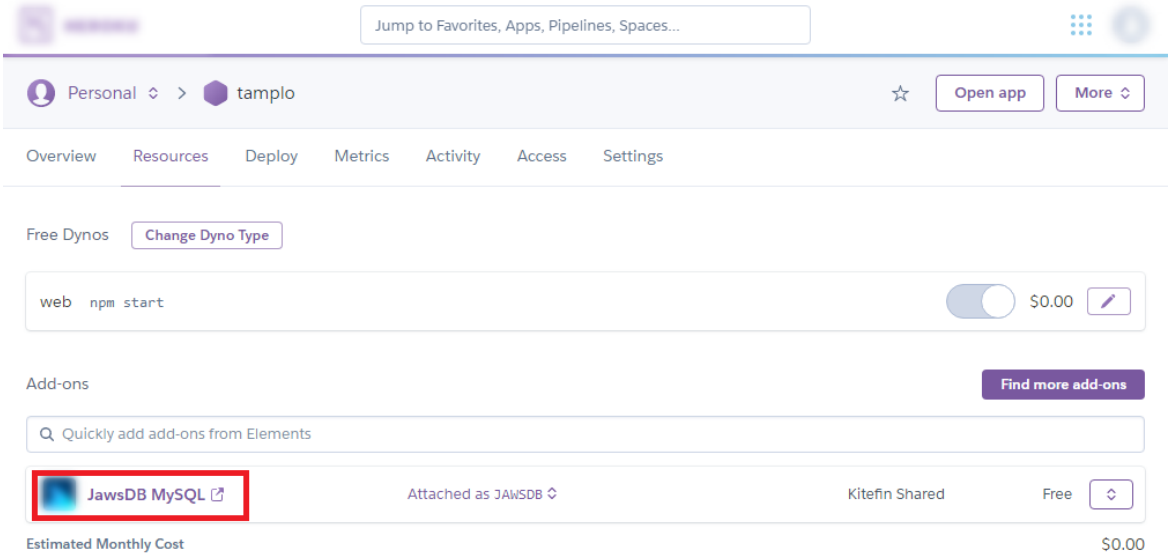
Εικόνα 42 Σελίδα προσθήκης πρόσθετων



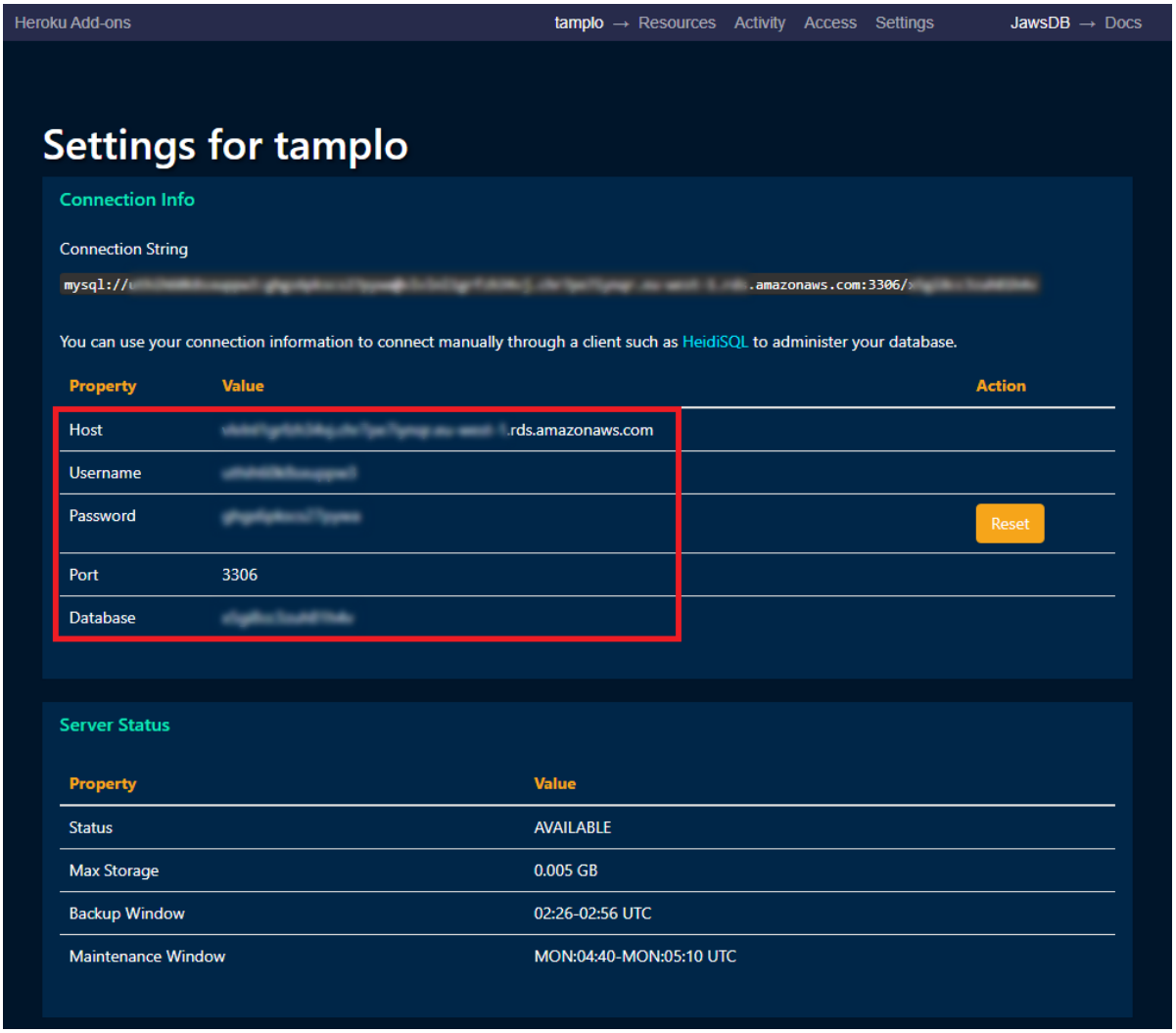
Εικόνα 43 Επιλογή της JawsDB MySQL



Εικόνα 44 Επιλογή πακέτου υπηρεσίας

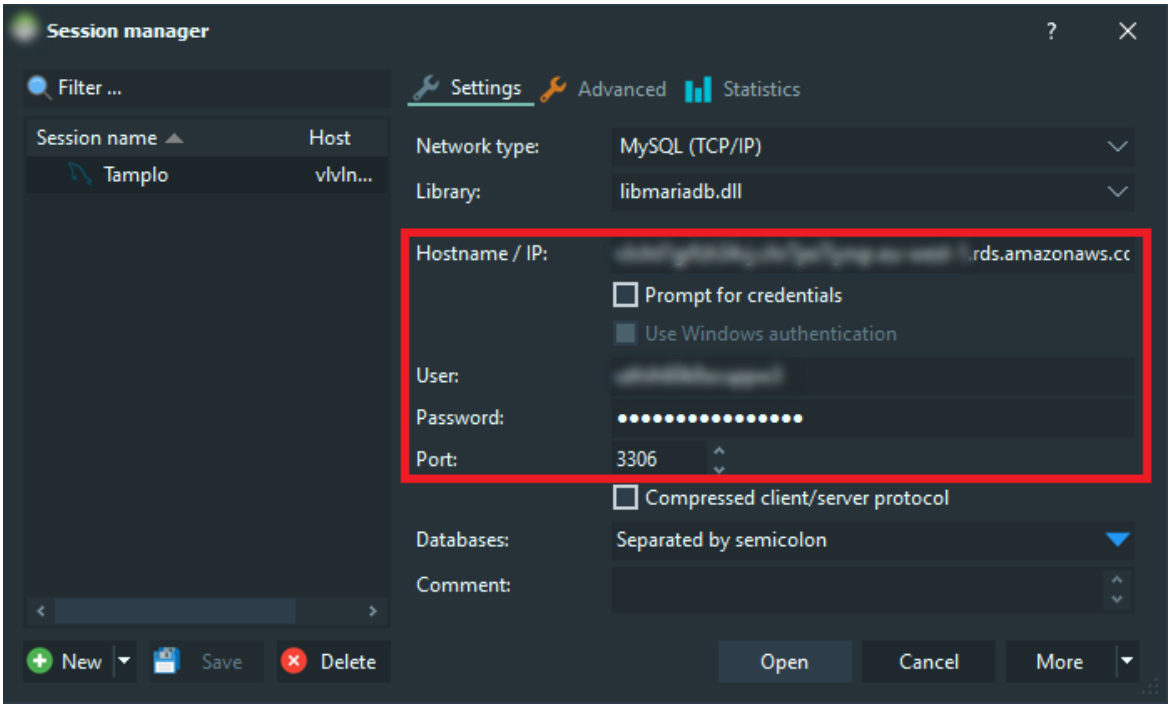


Εικόνα 45 Σελίδα προσθήκης πρόσθετων



Εικόνα 46 Σελίδα στοιχείων βάσης δεδομένων JawsDB

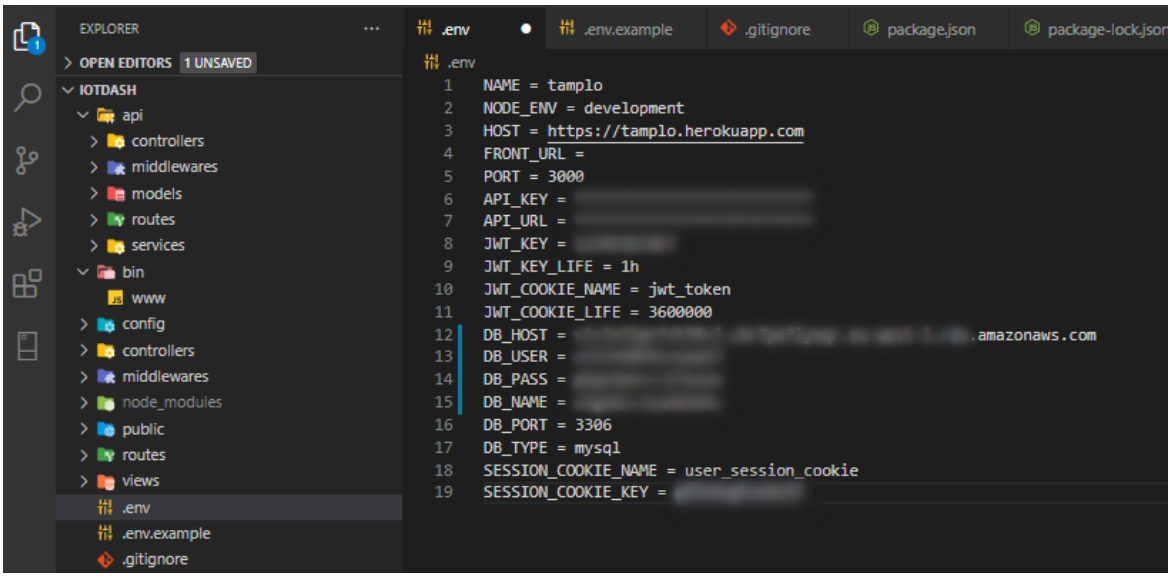
Με τη προσθήκη του πρόσθετου αυτού δημιουργήθηκε μία νέα άδεια βάση δεδομένων και εμφανίστηκαν τα στοιχεία της στη σελίδα (Host, Username, Password, Port, Database). Με τη χρήση ενός client όπως το HeidiSQL έγινε απομακρυσμένη σύνδεση στη βάση δεδομένων και εισαγωγή της βάσης δεδομένων της εφαρμογής όπως φαίνεται παρακάτω.



Εικόνα 47 Παράδειγμα σύνδεσης με HeidiSQL

2.8.4 Τελική δημοσίευση

Για τη τελική δημοσίευση έπρεπε να ενημερωθεί το αρχείο **.env** που περιέχει τη λίστα μεταβλητών περιβάλλοντος της εφαρμογής, να γίνει commit με τις αλλαγές και deploy στο Heroku [54] ώστε να συνδεθεί η εφαρμογή με τη βάση δεδομένων.



Εικόνα 48 Ενημέρωση αρχείου μεταβλητών περιβάλλοντος

```
$ git add .
$ git commit -m "DB Update"
$ git push heroku master
```

Εικόνα 49 Εντολές για ενημέρωση της εφαρμογής στο Heroku

2.8.5 Σύνδεση με domain name

Η διεύθυνση (το domain name) της εφαρμογής στο Heroku είχε τη μορφή «https://tamplo.herokuapp.com/», δηλαδή ήταν ενεργή κάτω από ένα sub domain που δημιούργησε η υπηρεσία του Heroku. Αρκετοί είναι οι λόγοι για τη χρήση ενός tld root domain name και ένας από αυτούς είναι η ασφάλεια. Με ένα κατοχυρωμένο root domain name μπορούν να δημιουργηθούν διευθύνσεις email με βάση αυτό και να χρησιμοποιηθεί πιστοποιητικό SSL με CSR. Επομένως, για την εργασία αυτή κατοχυρώθηκε η ονομασία «tamplo.io» εφόσον ήταν διαθέσιμη και δεν χρησιμοποιούταν η λέξη «tamplo» ή το «tamplo.io» από άλλες όμοιες υπηρεσίες στο διαδίκτυο.

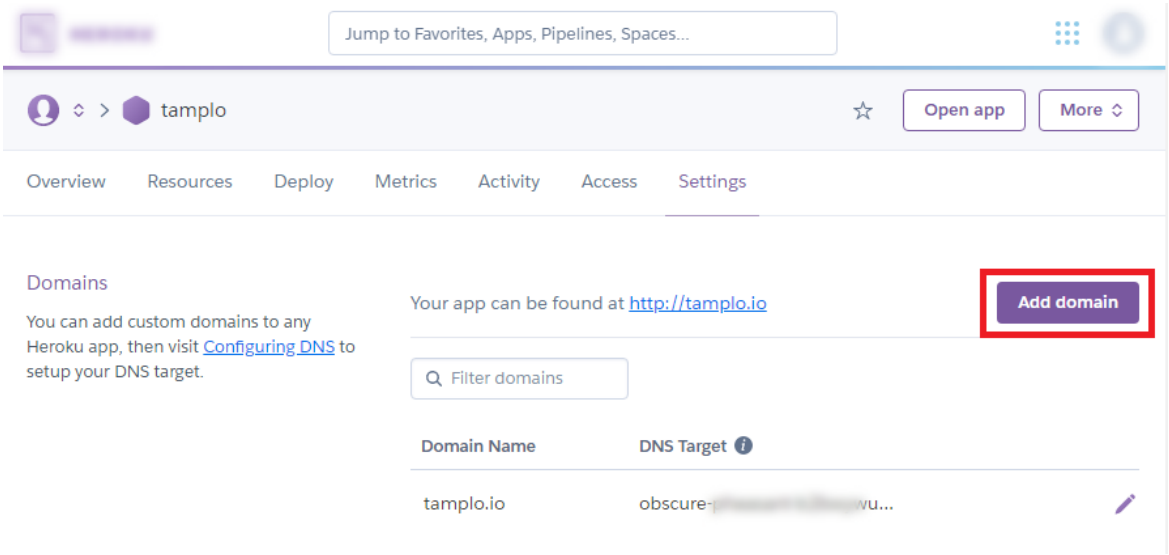
Η προσθήκη του νέου domain name στην εφαρμογή μπορούσε να γίνει με δύο τρόπους:

1) Μέσω της γραμμής εντολών στο Heroku CLI [57].

```
$ heroku domains:add example.com
Adding example.com to ⬢ example-app... done
 ▸    Configure your app's DNS provider to point to the DNS Target
 ▸    whispering-willow-5678.herokudns.com.
 ▸    For help, see https://devcenter.heroku.com/articles/custom-domains

The domain example.com has been enqueued for addition
 ▸    Run heroku domains:wait 'example.com' to wait for completion
```

Εικόνα 50 Heroku CLI προσθήκη domain name

2) Από τον πίνακα ελέγχου της πλατφόρμας στη σελίδα των ρυθμίσεων της εφαρμογής.



Εικόνα 51 Heroku Platform προσθήκη domain name

Με την προσθήκη του domain name, το Heroku επέστεψε ένα DNS Target όπως φαίνεται παρακάτω μαζί με σύντομες οδηγίες. Έπειτα έπρεπε να γίνει η προσθήκη του DNS Target στο DNS του domain name με τη μορφή εγγραφής ALIAS ή ANAME.

Record	Name	Target
`ALIAS` ή `ANAME`	<κενό> ή `@`	`to-dns-target-από-το-heroku.com.`

Εικόνα 52 Heroku DNS Target

Τέλος, έπρεπε να ενημερωθούν οι στατικές μεταβλητές της που περιέχουν τη διεύθυνση domain name στο αρχείο app.js και οι μεταβλητές περιβάλλοντος στο αρχείο .env. Και να γίνει commit/push στο Heroku όπως στη τελική δημοσίευση ώστε να ενημερωθεί η εφαρμογή.

2.8.6 Εγκατάσταση SSL

Με την σύνδεση του domain name, πολλές λειτουργίες της εφαρμογής και κυρίως HTTP requests που κάνει το Front-end στο REST API ήταν λογικό να μη λειτουργούν λόγω των περιορισμών CORS στο αρχείο app.js. Για παράδειγμα όπου είναι αναγκαία η χρήση Cookie για την επικοινωνία με τον διακομιστή και το Cookie έχει τις ιδιότητες Secure και httpOnly, ο διακομιστής δεν είναι δυνατό να λάβει το Cookie. Γι' αυτό το λόγο αλλά και για την ασφάλεια των δεδομένων των χρηστών έπρεπε να γίνει εγκατάσταση

πιστοποιητικού SSL με CSR στη πλευρά του διακομιστή και να δημιουργηθούν υποχρεωτικές ανακατευθύνσεις σε HTTPS σε όλους τους προορισμούς (routes) της εφαρμογής.

Επειδή όμως η διαδικασία αυτή διαφέρει αρκετά από πάροχο σε πάροχο, συνήθως προσφέρονται οδηγίες εγκατάστασης από τους ίδιους με βάση το σύνολο των δικών τους υπηρεσιών και τις αρχές πιστοποίησης (CA: Certification Authority) που συνεργάζονται, οπότε δε θα γίνει περαιτέρω ανάλυση σε αυτή την εργασία.

2.8.7 Scaling

Scaling, αλλιώς κλιμάκωση στα ελληνικά είναι ένα από τα πιο δημοφιλή χαρακτηριστικά του cloud computing και αφορά την ικανότητα αύξησης ή μείωσης των πόρων του πληροφοριακού συστήματος με βάση τις απαιτήσεις του. Στην εργασία αυτή η δημοσίευση ή αλλιώς εγκατάσταση της εφαρμογής έγινε πάνω σε λειτουργικό ενός διακομιστή του Heroku. Το container αυτό όπου τρέχει το Linux λειτουργικό στο Heroku ονομάζεται dyno και προσφέρονται διάφορες εκδόσεις dyno με διαφορετικούς πόρους (π.χ. υπολογιστική ισχύ, ram) κάτι που είναι εξαιρετικά χρήσιμο σε περίπτωση που θεωρηθεί αναγκαία η αναβάθμιση του πληροφοριακού συστήματος αυτού στο μέλλον. Το ίδιο ισχύει και για τη βάση δεδομένων όπου μπορεί να γίνει επιλογή ανάμεσα σε χωρητικότητα, ram, αριθμό συνδέσεων, αντιγράφων ασφαλείας κλπ.

2.9 Δοκιμές σε πραγματικές συνθήκες

Έχοντας

1. δημοσιεύσει επιτυχώς την εφαρμογή στη πλατφόρμα του Heroku,
2. συνδέσει την εφαρμογή με νέο root domain name «tamplo.io» (προαιρετικό) και
3. εγκαταστήσει SSL με CSR (προαιρετικό),

η εφαρμογή είναι πλέον πλήρως λειτουργική και σε κατάσταση «production». Στο πλαίσιο των δοκιμών έγινε η εγγραφή μερικών χρηστών, η δημιουργία συσκευών, ανάθεση αισθητήρων και δημιουργήθηκαν κλειδιά API για τη σύνδεση των συσκευών. Έγινε έλεγχος όλων των διαδρομών (routes) του Front-end μέσω browser και του Back-end REST API με χρήση του εργαλείου Postman.

2.9.1.1 Παράδειγμα για την αποστολή μίας μέτρησης από ένα αισθητήρα μίας συσκευής

Όνομα χρήστη: andreas
Όνομα συσκευής: arduino_1
Όνομα αισθητήρα: arduino_1_sensor_1
Τιμή μέτρησης: 20
Κλειδί API συσκευής: 12345

POST Request: https://tamplo.io/api/v1/users/andreas/devices/arduino_1/data

Header: Authorization

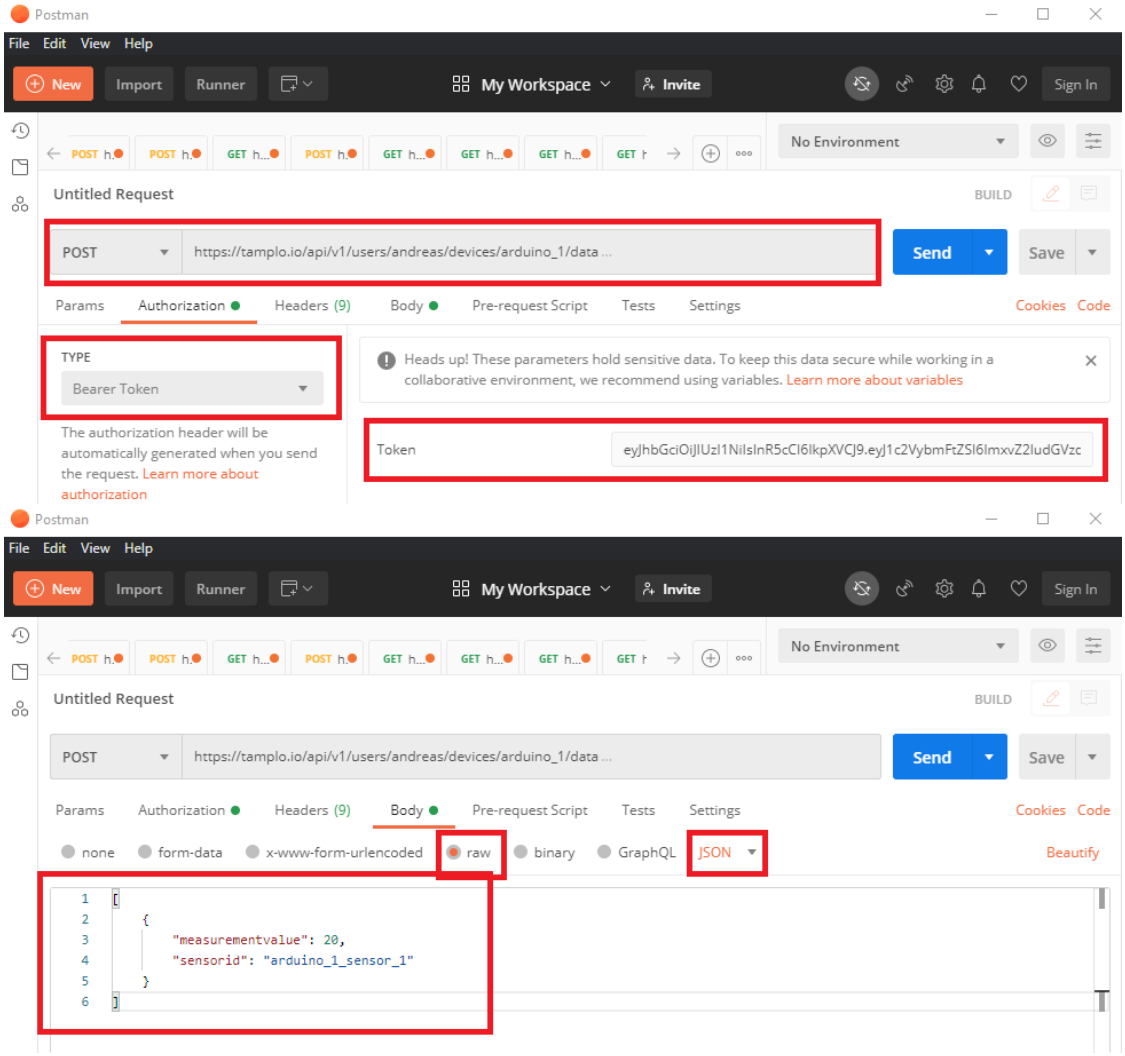
Authorization Type: Bearer Token

Authorization Value: Bearer 12345

Body Type: JSON

Body:

```
[
    {
        "measurementvalue": 20,
        "sensorid": "arduino_1_sensor_1"
    }
]
```



Εικόνα 53 Δοκιμή με Postman

2.9.1.2 Παράδειγμα για την αποστολή μίας μέτρησης από πολλούς αισθητήρες μίας συσκευής

Όνομα χρήστη: andreas
Όνομα συσκευής: arduino_1
Όνομα αισθητήρα 1: arduino_1_sensor_1
Όνομα αισθητήρα 2: arduino_1_sensor_2
Όνομα αισθητήρα 3: arduino_1_sensor_3
Τιμή μέτρησης αισθητήρα 1: 20
Τιμή μέτρησης αισθητήρα 2: 34
Τιμή μέτρησης αισθητήρα 3: 40
Κλειδί API συσκευής: 12345

POST Request: https://tamplo.io/api/v1/users/andreas/devices/arduino_1/data

Header: Authorization

Authorization Type: Bearer Token

Authorization Value: Bearer 12345

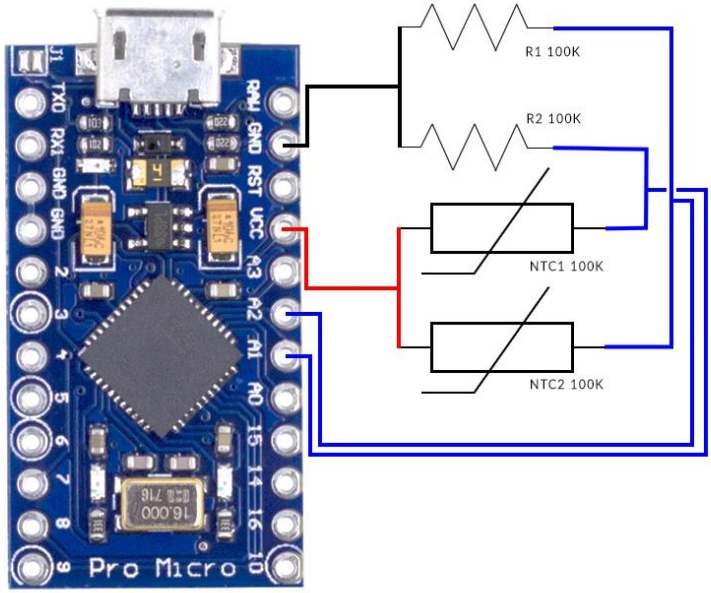
Body Type: JSON

Body:

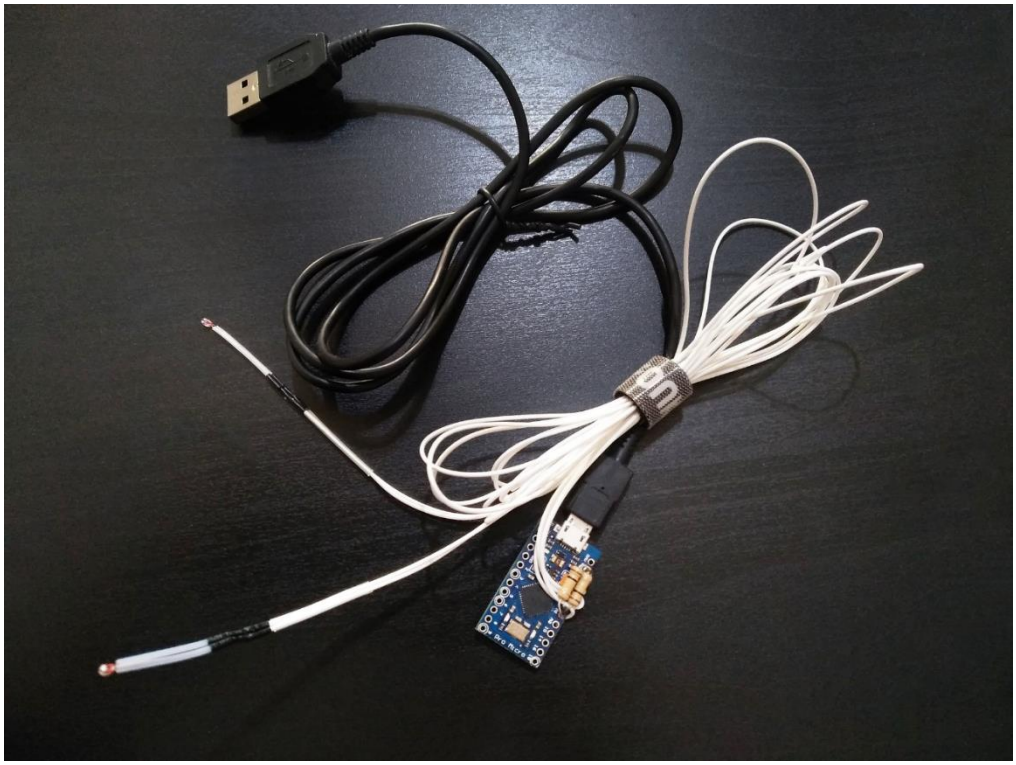
```
[
    {
        "measurementvalue": 20,
        "sensorid": "arduino_1_sensor_1"
    },
    {
        "measurementvalue": 34,
        "sensorid": "arduino_1_sensor_2"
    },
    {
        "measurementvalue": 40,
        "sensorid": "arduino_1_sensor_3"
    }
]
```

2.9.1.3 Κατασκευή Client για Υπολογιστές-Arduino

Επίσης αναπτύχθηκε ένας Client για υπολογιστές για την ανάγνωση και την αποστολή δεδομένων μέσω USB (SerialPort) από Arduino που δεν υποστηρίζουν σύνδεση με το διαδίκτυο. Για λόγους διαθεσιμότητας χρησιμοποιήθηκε ένα Arduino Pro Micro με δύο αισθητήρες θερμοκρασίας (NTC Thermistor) των 100K Ohm και από μία αντίσταση των 100K Ohm το καθένα όπως φαίνεται στο κύκλωμα της εικόνας παρακάτω.



Εικόνα 54 Το κύκλωμα Arduino-Αισθητήρων



Εικόνα 55 Arduino Pro Micro με δύο αισθητήρες θερμοκρασίας

Ο κώδικας που χρησιμοποιήθηκε στο Arduino είναι βασισμένος σε μία πολύ απλή υλοποίηση του CircuitBasics [58] σε Arduino IDE για την μέτρηση και την εμφάνιση των τιμών στη κλίμακα Κελσίου ανά 1 δευτερόλεπτο όπως φαίνεται στον κώδικα παρακάτω.

```
int sensor_1 = A1;
int sensor_2 = A2;
int Vo_s1, Vo_s2;
float R1_s1 = 10000, R1_s2 = 10000;
float logR2_s1, R2_s1, T_s1, Tc_s1, logR2_s2, R2_s2, T_s2, Tc_s2;
float c1_s1 = 1.009249522e-03, c2_s1 = 2.378405444e-
04, c3_s1 = 2.019202697e-07;
float c1_s2 = 1.009249522e-03, c2_s2 = 2.378405444e-
04, c3_s2 = 2.019202697e-07;

void setup() {
Serial.begin(9600);
}

void loop() {

  Vo_s1 = analogRead(sensor_1);
  R2_s1 = R1_s1 * (1023.0 / (float)Vo_s1 - 1.0);
  logR2_s1 = log(R2_s1);
  T_s1 = (1.0 / (c1_s1 + c2_s1*logR2_s1 + c3_s1*logR2_s1*logR2_s1*logR2_s1)
);
  Tc_s1 = T_s1 - 273.15;

  Vo_s2 = analogRead(sensor_2);
  R2_s2 = R1_s2 * (1023.0 / (float)Vo_s2 - 1.0);
  logR2_s2 = log(R2_s2);
  T_s2 = (1.0 / (c1_s2 + c2_s2*logR2_s2 + c3_s2*logR2_s2*logR2_s2*logR2_s2)
);
  Tc_s2 = T_s2 - 273.15;

  Serial.print(Tc_s1);
  Serial.print(" ");
  Serial.println(Tc_s2);

  delay(1000);
}
```

Η ανάπτυξη του Client έγινε επίσης με JavaScript σε Node.js με χρήση των βιβλιοθηκών SerialPort και Axios. Με τη βιβλιοθήκη SerialPort ο Client είναι σε θέση να διαβάζει τις τιμές που στέλνει η συσκευή στον υπολογιστή και με τη βιβλιοθήκη Axios να δημιουργεί

POST Requests όπως το Front-end της εφαρμογής με το Header και Body που είναι επιθυμητό από την εφαρμογή όπως φαίνεται στον κώδικα παρακάτω.

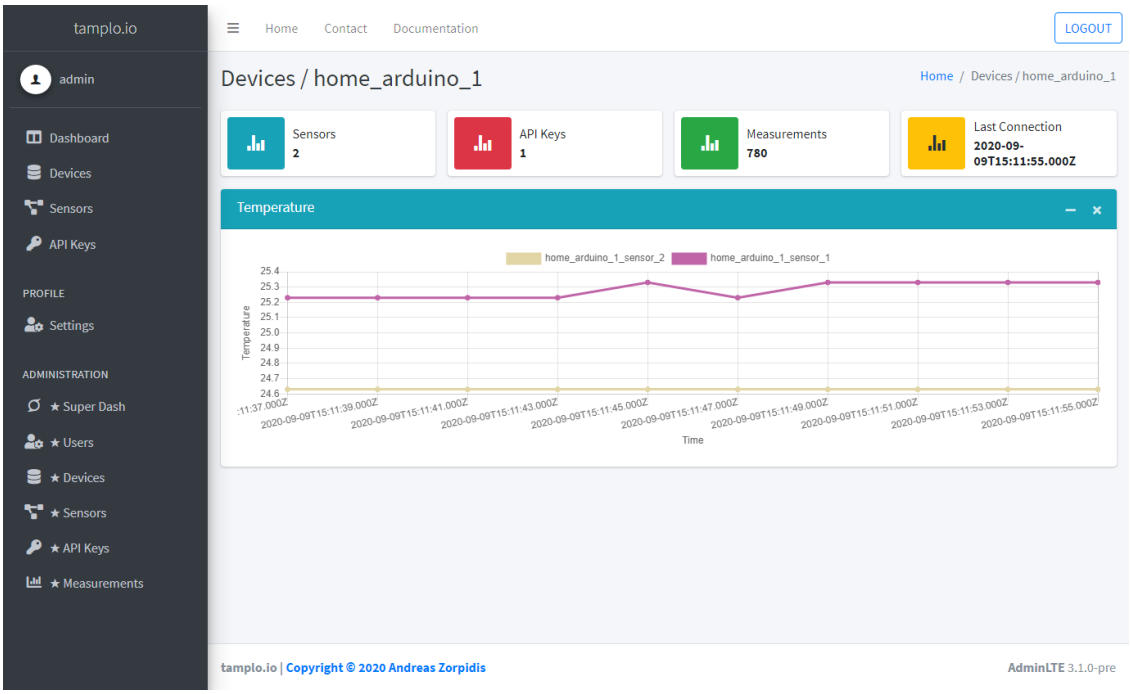
```
const SerialPort = require('serialport');
const Readline = require('@serialport/parser-readline');
const port = new SerialPort('COM16', { baudRate: 9600 });
const axios = require('axios');

const apikey = "12345";
const user = "andreas";
const deviceid = "arduino_1";
const sensorid_1 = "arduino_1_sensor_1";
const sensorid_2 = "arduino_1_sensor_2";
const url = 'https://tamplo.io/api/v1/users/' + user + '/devices/' + device
id + '/data';

const parser = new Readline()
port.pipe(parser)

parser.on('data', (data) => {

    var split_data = data.split(/\s+/);
    var sensor_1 = split_data[0];
    var sensor_2 = split_data[1];
    console.log(sensor_1 + ' ' + sensor_2);

    axios({
        method: 'post',
        url: url,
        headers: { Authorization: `Bearer ${apikey}` },
        data:
            [
                {
                    "measurementvalue": parseFloat(sensor_1),
                    "sensorid": sensorid_1
                },
                {
                    "measurementvalue": parseFloat(sensor_2),
                    "sensorid": sensorid_2
                }
            ]
    });
})
```

Ο παραπάνω κώδικας περιέχει ακριβώς ότι απαιτείται για τη λειτουργία του Client. Πιο αναλυτικά: στην αρχή γίνεται η δήλωση των βιβλιοθηκών, έπειτα αποθήκευση δεδομένων

σε μεταβλητές όπως το κλειδί API της συσκευής και τα στοιχεία που συνθέτουν τον σύνδεσμο αποστολής του POST Request. Έπειτα διαβάζονται οι μετρήσεις κάθε φορά που τις στέλνει το Arduino (έχουν τη μορφή: 00.00 00.00, δηλαδή χωρίζονται με κενό ανάμεσά τους), προβάλλονται στη κονσόλα, δημιουργείται το POST Request, το Header με το Bearer Token (κλειδί API της συσκευής), το Body με τις τιμές και τα ονόματα των αισθητήρων και τέλος ολοκληρώνεται το Request.

2.9.1.4 Προβολή μετρήσεων

Τέλος, η προβολή των δεδομένων έγινε με επιτυχία στη σελίδα της συσκευής. Δημιουργήθηκε αυτόματα ένα νέο γράφημα για τον τύπο των αισθητήρων που χρησιμοποιήθηκαν και κάθε αισθητήρας αναπαραστάθηκε με διαφορετικό χρώμα. Η ενημέρωση των μετρήσεων και των υπόλοιπων πληροφοριών του ταμπλό του χρήστη γινόταν έγκαιρα ανά 1 δευτερόλεπτο ενώ στο γράφημα διατηρούνταν οι τελευταίες 10 μετρήσεις.



Εικόνα 56 Σελίδα μετρήσεων συσκευής

ΠΑΡΑΡΤΗΜΑ

Στα δεύτερο κεφάλαιο της εργασίας αυτής, έγινε μια προσπάθεια παρουσίασης της υλοποίησης μιας εφαρμογής νέφους, των τεχνικών, των εργαλείων και των τεχνολογιών που χρησιμοποιήθηκαν τα οποία προϋποθέτουν καλές γνώσεις γύρω από τις τεχνολογίες του διαδικτύου, καλή γνώση JavaScript και βασικές γνώσεις σχεδιασμού πληροφοριακών συστημάτων. Η παρουσίαση έγινε με συνοπτικό τρόπο ώστε να μην αποσπάται η προσοχή του αναγνώστη και να είναι ευανάγνωστη, κάτι που κρύβει το μέγεθος της υλοποίησης της εφαρμογής και κομμάτια κώδικα που ίσως να ενδιέφεραν κάποιο πιο εξοικειωμένο αναγνώστη. Γι' αυτό το λόγο, ο πλήρης κώδικας της υλοποίησης είναι διαθέσιμος στις διευθύνσεις https://tamplo.io και https://andreaszorpidis.com.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] NIST- National Institute of Standards and Technology, «The NIST Definition of Cloud Computing,» [Ηλεκτρονικό]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf. [Πρόσβαση 09 2020].

[2] Science Direct, «2.1 Perspective 1: essential characteristics,» [Ηλεκτρονικό]. Available: https://www.sciencedirect.com/topics/computer-science/on-demand-self-service. [Πρόσβαση 09 2020].

[3] Oracle, «What Is the Internet of Things (IoT)?,» [Ηλεκτρονικό]. Available: https://www.oracle.com/internet-of-things/what-is-iot.html. [Πρόσβαση 09 2020].

[4] J. DIZDAREVIĆ, F. CARPIO, A. JUKAN και X. MASIP-BRUIN, «A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration,» [Ηλεκτρονικό]. Available: https://arxiv.org/pdf/1804.01747.pdf. [Πρόσβαση 09 2020].

[5] ThingsBoard, «What is ThingsBoard?,» [Ηλεκτρονικό]. Available: https://thingsboard.io/docs/faq/. [Πρόσβαση 09 2020].

[6] Grafana Labs, «What is Grafana?,» [Ηλεκτρονικό]. Available: https://grafana.com/docs/grafana/latest/getting-started/what-is-grafana/. [Πρόσβαση 09 2020].

[7] Thinger.io, «What is Thinger.io?,» [Ηλεκτρονικό]. Available: https://docs.thinger.io/. [Πρόσβαση 09 2020].

[8] Microsoft Azure, «Microsoft Azure IoT Suite – Connecting Your Things to the Cloud,» [Ηλεκτρονικό]. Available: https://azure.microsoft.com/en-in/blog/microsoft-azure-iot-suite-connecting-your-things-to-the-cloud/. [Πρόσβαση 09 2020].

[9] Google Cloud, «Google Cloud IoT,» [Ηλεκτρονικό]. Available: https://cloud.google.com/solutions/iot. [Πρόσβαση 09 2020].

[10] aws, «AWS IoT Core features,» [Ηλεκτρονικό]. Available: https://aws.amazon.com/iot-core/features/. [Πρόσβαση 09 2020].

[11] salesforce, «The Internet of Things Connects Customers to Their World,» [Ηλεκτρονικό]. Available: https://www.salesforce.com/products/platform/best-practices/internet-of-things-connects-customers/ . [Πρόσβαση 09 2020].

[12] Kaa, «Kaa,» [Ηλεκτρονικό]. Available: https://kaaproject.github.io/kaa/docs/v0.10.0/Welcome/. [Πρόσβαση 09 2020].

[13] ThingSpeak, «Learn More About ThingSpeak,» [Ηλεκτρονικό]. Available: https://thingspeak.com/pages/learn_more. [Πρόσβαση 09 2020].

[14] DeviceHive, «DeviceHive,» [Ηλεκτρονικό]. Available: https://devicehive.com/. [Πρόσβαση 09 2020].

[15] Mainflux, «Mainflux,» [Ηλεκτρονικό]. Available: https://github.com/mainflux/mainflux. [Πρόσβαση 09 2020].

[16] Μ. Μάντακας, «Επιχειρησιακές διαδικασίες,» σε *Τεχνολογία Λογισμικού*, Άρτα, Τμήμα Πληροφορικής και Τηλεπικοινωνιών - Πανεπιστήμιο Ιωαννίνων, 2020.

[17] Wikipedia, «Model-view-controller,» [Ηλεκτρονικό]. Available: https://el.wikipedia.org/wiki/Model-view-controller. [Πρόσβαση 09 2020].

[18] Visual Studio Code, «Visual Studio Code FAQ,» [Ηλεκτρονικό]. Available: https://code.visualstudio.com/docs/supporting/faq. [Πρόσβαση 09 2020].

[19] Microsoft Ignite, «Βασικά στοιχεία Git και GitHub για το Docs,» [Ηλεκτρονικό]. Available: https://docs.microsoft.com/el-gr/contribute/git-github-fundamentals. [Πρόσβαση 09 2020].

[20] Node.js, «What is npm,» [Ηλεκτρονικό]. Available: https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/. [Πρόσβαση 09 2020].

[21] Wikipedia, «Node.js,» [Ηλεκτρονικό]. Available: https://el.wikipedia.org/wiki/Nodejs. [Πρόσβαση 09 2020].

[22] Wikipedia, «Google Chrome,» [Ηλεκτρονικό]. Available: https://el.wikipedia.org/wiki/Google_Chrome. [Πρόσβαση 09 2020].

[23] Google, «Chrome DevTools,» [Ηλεκτρονικό]. Available: https://developers.google.com/web/tools/chrome-devtools. [Πρόσβαση 09 2020].

[24] Postman, «Postman,» [Ηλεκτρονικό]. Available: https://www.postman.com/. [Πρόσβαση 08 2020].

[25] Wikipedia, «HeidiSQL,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/HeidiSQL. [Πρόσβαση 08 2020].

[26] tutorialspoint, «Node.js - Express Framework,» [Ηλεκτρονικό]. Available: https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm. [Πρόσβαση 08 2020].

[27] Express-Validator, «Getting Started,» [Ηλεκτρονικό]. Available: https://express-validator.github.io/docs/. [Πρόσβαση 08 2020].

[28] tutorialspoint, «ExpressJS - Sessions,» [Ηλεκτρονικό]. Available: https://www.tutorialspoint.com/expressjs/expressjs_sessions.htm. [Πρόσβαση 08 2020].

[29] Express.js, «express-session,» [Ηλεκτρονικό]. Available: https://expressjs.com/en/resources/middleware/session.html. [Πρόσβαση 08 2020].

[30] express-jwt, «express-jwt,» [Ηλεκτρονικό]. Available: https://www.npmjs.com/package/express-jwt. [Πρόσβαση 08 2020].

[31] T. Kadwill, «Using Sequelize ORM with Node.js and Express,» [Ηλεκτρονικό]. Available: https://stackabuse.com/using-sequelize-orm-with-nodejs-and-express/. [Πρόσβαση 08 2020].

[32] Sequelize, «Sequelize ORM,» [Ηλεκτρονικό]. Available: https://sequelize.org/. [Πρόσβαση 08 2020].

[33] Sequelize-Auto, «Sequelize-Auto,» [Ηλεκτρονικό]. Available: https://www.npmjs.com/package/sequelize-auto. [Πρόσβαση 08 2020].

[34] JWT.IO, «Introduction to JSON Web Tokens,» [Ηλεκτρονικό]. Available: https://jwt.io/introduction/. [Πρόσβαση 08 2020].

[35] Bcrypt, «Bcrypt,» [Ηλεκτρονικό]. Available: https://www.npmjs.com/package/bcrypt. [Πρόσβαση 08 2020].

[36] Morgan, «Morgan,» [Ηλεκτρονικό]. Available: https://www.npmjs.com/package/morgan. [Πρόσβαση 08 2020].

[37] Nodemon, «Nodemon,» [Ηλεκτρονικό]. Available: https://www.npmjs.com/package/nodemon. [Πρόσβαση 08 2020].

[38] Dotenv, «Dotenv,» [Ηλεκτρονικό]. Available: https://www.npmjs.com/package/dotenv. [Πρόσβαση 08 2020].

[39] Cors, «Cors,» [Ηλεκτρονικό]. Available: https://www.npmjs.com/package/cors. [Πρόσβαση 08 2020].

[40] Body-Parser, «Body-Parser,» [Ηλεκτρονικό]. Available: https://www.npmjs.com/package/body-parser. [Πρόσβαση 09 2020].

[41] Cookie-Parser, «Cookie-Parser,» [Ηλεκτρονικό]. Available: https://www.npmjs.com/package/cookie-parser. [Πρόσβαση 08 2020].

[42] hbs, «hbs,» [Ηλεκτρονικό]. Available: https://www.npmjs.com/package/hbs. [Πρόσβαση 09 2020].

[43] Passport, «Passport,» [Ηλεκτρονικό]. Available: http://www.passportjs.org/. [Πρόσβαση 08 2020].

[44] AdminLTE, «AdminLTE,» [Ηλεκτρονικό]. Available: https://github.com/ColorlibHQ/AdminLTE. [Πρόσβαση 08 2020].

[45] Axios, «Axios,» [Ηλεκτρονικό]. Available: https://github.com/axios/axios. [Πρόσβαση 09 2020].

[46] Wikipedia, «Chart.js,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Chart.js. [Πρόσβαση 09 2020].

[47] tutorialspoint, «Node.js - RESTful API,» [Ηλεκτρονικό]. Available: https://www.tutorialspoint.com/nodejs/nodejs_restful_api.htm. [Πρόσβαση 09 2020].

[48] Γ. Αλέξανδρος, «Εισαγωγή στην αρχιτεκτονική δικτυακών υπηρεσιών REST,» [Ηλεκτρονικό]. Available: http://www.users.dpem.tuc.gr/gougousis/rest/. [Πρόσβαση 09 2020].

[49] Microsoft, «Περιγραφή των κωδικών κατάστασης των υπηρεσιών Microsoft Internet Information Services (IIS) 5.0 και 6.0,» [Ηλεκτρονικό]. Available: https://support.microsoft.com/el-gr/help/318380/description-of-microsoft-internet-information-services-iis-5-0-and-6-0. [Πρόσβαση 08 2020].

[50] Mozilla MDN Web Docs, «Using HTTP cookies,» [Ηλεκτρονικό]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies. [Πρόσβαση 08 2020].

[51] Express, «Using template engines with Express,» [Ηλεκτρονικό]. Available: https://expressjs.com/en/guide/using-template-engines.html. [Πρόσβαση 08 2020].

[52] Express, «Serving static files in Express,» [Ηλεκτρονικό]. Available: https://expressjs.com/en/starter/static-files.html. [Πρόσβαση 08 2020].

[53] Microsoft, «What is PaaS? Platform as a service,» [Ηλεκτρονικό]. Available: https://azure.microsoft.com/en-us/overview/what-is-paas/. [Πρόσβαση 09 2020].

[54] Heroku, «Deploying with Git,» [Ηλεκτρονικό]. Available: https://devcenter.heroku.com/articles/git. [Πρόσβαση 08 2020].

[55] IBM Cloud Learn Hub, «DBaaS (Database-as-a-Service),» [Ηλεκτρονικό]. Available: https://www.ibm.com/cloud/learn/dbaas. [Πρόσβαση 09 2020].

[56] Heroku, «JawsDB,» [Ηλεκτρονικό]. Available: https://devcenter.heroku.com/articles/jawsdb. [Πρόσβαση 08 2020].

[57] Heroku, «Custom Domain Names for Apps,» [Ηλεκτρονικό]. Available: https://devcenter.heroku.com/articles/custom-domains. [Πρόσβαση 08 2020].

[58] Scott Campbell - Circuit Basics, «MAKE AN ARDUINO TEMPERATURE SENSOR (THERMISTOR TUTORIAL),» [Ηλεκτρονικό]. Available: https://www.circuitbasics.com/arduino-thermistor-temperature-sensor-tutorial/. [Πρόσβαση 09 2020].

[59] Nodemon, «Nodemon,» [Ηλεκτρονικό]. Available: https://www.npmjs.com/package/nodemon. [Πρόσβαση 08 2020].