ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αρχιτεκτονικές και εφαρμογές δικτύων προσωπικής περιοχής στο πεδίο του αθλητισμού

Χάρης Ραζής

161072

Παναγιώτης Καρκαζής Επίκουρος Καθηγητής
Απόστολος Αναγνωστόπουλος ΕΤΕΠ

Διπλωματική εργασία υποβληθείσα στο Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

του Πανεπιστημίου Δυτικής Αττικής

ΑΘΗΝΑ, ΟΚΤΩΜΒΡΙΟΣ 2021

Πανεπιστήμιο Δυτικής Αττικής, Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Χάρης Ραζής

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
Αρχιτεκτονικές και εφαρμογές δικτύων προσωπικής περιοχής στο πεδίο του αθλητισμού

ΧΑΡΗΣ ΡΑΖΗΣ
Α.Μ. 161072

Εισηγητής:
Παναγιώτης Καρκαζής

Εξεταστική Επιτροπή:

Παναγιώτης Καρκαζής,
Επίκουρος Καθηγητής

Panagiotis Karkazis
Digitally signed by Panagiotis Karkazis
Date: 2021.10.25 12:09:54 +03'00'

Αθανάσιος Βουλόδημος,
Επίκουρος Καθηγητής

Athanasios Voulodimos
Digitally signed by Athanasios Voulodimos
Date: 2021.10.26 19:23:23 +03'00'

Ελένη – Αικατερίνη Λελίγκου,
Αναπληρώτρια Καθηγήτρια

Eleni Aikaterini Leligkou
Digitally signed by Eleni Aikaterini Leligkou
Date: 2021.10.31 21:52:41 +02'00'

Ημερομηνία εξέτασης 19/10/2021

Η έγκριση της διπλωματικής εργασίας δεν υποδηλώνει την αποδοχή των γνωμών του συγγραφέα.

Κατά τη συγγραφή τηρήθηκαν οι αρχές της ακαδημαϊκής δεοντολογίας.

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος Ραζής Χάρης του Ανδρέα, με αριθμό μητρώου 161072 φοιτητής του Πανεπιστημίου Δυτικής Αττικής της Σχολής Μηχανικών του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών, δηλώνω υπεύθυνα ότι:

«Είμαι συγγραφέας αυτής της διπλωματικής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος. Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

Ο Δηλών

ΠΕΡΙΛΗΨΗ

Αρχιτεκτονικές και εφαρμογές δικτύων προσωπικής περιοχής στο πεδίο του αθλητισμού

Χάρης Ραζής

Αντικείμενο της διπλωματικής αυτής εργασίας είναι η μελέτη της αρχιτεκτονικής των δικτυακών εφαρμογών Body Area Networks (BAN) και Personal Area Networks (PAN) με στόχο την βελτιστοποίηση των επιδόσεων των αθλητών, μέσω την ενίσχυση των εξατομικευμένων πρακτικών προπόνησης. Στο πλαίσιο την διπλωματικής θα μελετηθούν αντικείμενα, όπως η συλλογή δεδομένων για την αναγνώριση της φυσικής στάσης του σώματος του αθλητή, καθώς και ο σχεδιασμός έξυπνων υπηρεσιών IoT με έμφαση τον αθλητισμό. Τέλος θα σχεδιαστεί και θα υλοποιηθεί μια εφαρμογή «e-sport» η οποία θα συλλέγει και θα αναλύει δεδομένα θέσης του σώματος του αθλητή σε πραγματικό χρόνο.

Λέξεις κλειδιά:

Διαδίκτυο των Αντικειμένων, Δίκτυα Προσωπικής Περιοχής, Docker, Node-js, Arduino, IMU

ABSTRACT

Personal Area Networks applications on sports

Haris Razis

The object of this dissertation is the study of the architecture of Body Area Networks (BAN) and Personal Area Networks (PAN), and their applications, with the aim of optimizing the performance of athletes, by enhancing personalized training practices. In the context of the thesis, subjects that will be studied include the collection of data for the recognition of the physical posture of the athlete's body, as well as the design of smart IoT services with an emphasis on sports. Finally, an "e-sport" application will be designed and implemented which will collect and analyze data on the athlete's body position in real-time.

Keywords

Internet of Things, Personal Area Networks, Docker, Node-js, Arduino, IMU

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ ΚΑΙ ΕΙΚΟΝΩΝ

ΣΥΝΤΟΜΕΥΣΕΙΣ - ΑΚΡΟΝΥΜΙΑ

IoT – Internet of Things

IMU – Inertial Measurement Unit

IEEE – Institute of Electrical and Electronics Engineers

BAN – Body Area Networks

PAN – Personal Area Networks

M2M – Machine to Machine

NFC – Near Field Communication

UWB – Ultra Wideband

LPWAN – Low Power Wide Area Networking

BPSK – Binary Phase Shifting Key

FTTx – Fiber To The x

xDSL – x Digital Subscriber Line

PLC - Power Line Consumption

SNR - Signal to Noise Ratio

SDN – Software Defined Networking

NFV - Network functions virtualization

QoS – Quality of Service

Mbps - Megabits per second

MK – Master Key

PTK - Pairwise Temporal Key

GTK – Group Temporal Key

PS – Personal Server

AP – Access Point

GHz - Gigahertz

Kbps – Kilobits per second

HART - Highway Addressable Remote Transducer

6LoWPAN - IPv6 over Low-Power Wireless Personal Area Networks

DOF – Degree of Freedom

MEMS - Microelectromechanical Systems

ECG - Electrocardiography

PPG – Photoplethysmography

GPS – Global Positioning System

EY – Ernst & Young Global Limited

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΙΣΑΓΩΓΗ

Σε έναν κόσμο όπου οι αισθητήρες, οι έξυπνες συσκευές και η ολοένα αυξανόμενη πρόσβασή στο διαδίκτυο βρίσκουν τρόπο σε κάθε πτυχή της ζωής μας, το διαδίκτυο των αντικειμένων καλείται να περιγράψει την στενή σχέση του φυσικού με τον ψηφιακό κόσμο. Σύμφωνα με την ***IEEE*** οι συνδεδεμένες ***IoT*** συσκευές στο διαδίκτυο υπολογίζονται στις 50 δισεκατομμύρια, με προβλέψεις για την επόμενη δεκαετία να τις τοποθετούν στο φάσμα των 125 δισεκατομμυρίων. Το ***IoT*** έχει πλέον βιομηχανικές, εμπορικές και καταναλωτικές λύσεις, από έναν απλό αυτοματισμό στο σπίτι μέχρι τα πολύπλοκα μοντέλα προγνωστικής ποιότητας στον βιομηχανικό τομέα.

Δεν είναι δύσκολο λοιπόν, κάποιος να καταλάβει ότι η κοινωνία θα γαλουχηθεί από μια σειρά έξυπνων συστημάτων που θα περικλείουν την ζωή μας, στο χώρο εργασίας, στα σχολεία και στο ίδιο μας το σπίτι. Τα συστήματα αυτά θα συνδέονται και θα συνεργάζονται για να προσφέρουν πληροφορία και υπηρεσίες στον χρήστη, δημιουργώντας έτσι ένα έξυπνο περιβάλλον, δίνοντας υπολογιστική ισχύ και δυνατότητα επικοινωνίας σε καθημερινά αντικείμενα.

Αντικείμενο αυτής της διπλωματικής είναι η μελέτη των διαδικτυακών εφαρμογών ***Body Area Network*** (***BAN)*** και ***Personal Area Network (PAN)*** με στόχο την βελτιστοποίηση των επιδόσεων των αθλητών. Αναλυτικότερα στο κεφάλαιο 1 θα περιγράφουν οι τωρινές τεχνολογίες ***BAN*** και ***PAN*** ώστε να αποκτήσουμε το κατάλληλο υπόβαθρο για να εμβαθύνουμε περισσότερο στα επόμενα κεφάλαια. Στην συνέχεια στο κεφάλαιο 2, θα εξεταστούν οι τεχνολογίες και αισθητήρες για την συλλογή δεδομένων των αθλητών. Στο κεφάλαιο 3, θα μελετηθεί η περίπτωση του ***Vello***, μιας εφαρμογής που με την χρήση του ***IoT*** μεταμόρφωσε την ποδηλασία. Στο κεφάλαιο

4 θα αναλυθεί η ***Ichnaea***, μια εφαρμογή που συλλεγεί και αναλύει δεδομένα θέσης σώματος του αθλητή σε πραγματικό χρόνο. Τέλος το κεφάλαιο 5 είναι η κατακλείδα της διπλωματικής, με τα αποτελέσματα και μια σύντομη αναφορά σε μελλοντικές προβλέψεις.

ΔΙΚΤΥΑ ΠΡΟΣΩΠΙΚΗΣ ΠΕΡΙΟΧΗΣ

1.1 ΕΙΣΑΓΩΓΗ ΣΤΑ ΔΙΚΤΥΑ

Το ***Machine to Machine*** αποτελεί μια απευθείας επικοινωνία συσκευής με συσκευή η οποία μπορεί να γίνει ενσύρματα η ασύρματα [1]. Τέτοιου είδους επικοινωνίες είναι άρρηκτα συνδεδεμένες με το ***IoT*** αφού δεν χρειάζεται η αλληλεπίδραση ανθρώπου στην ανταλλαγή πληροφορίας. Μερικές από τις τεχνολογίες που εμπίπτουν σε αυτή την κατηγορία είναι τα εξής [2]:

- ***5G*** που προσφέρει αυξημένο ρυθμό δεδομένων, δυνατότητα υποστήριξης τεράστιου αριθμού συσκευών και πολύ χαμηλή απόκριση σε σχέση με την προηγουμένη γενιά δικτυών κινητής.
- ***Near Field Communication*** το οποίο είναι διάσημο για την ευκολία χρήσης και το μηδενικό στήσιμο.
- ***Ultra Wideband*** με δυνατότητες υψηλού ***bandwidth***, μεγάλης ακρίβειας ακόμα και σε εξωτερικούς χώρους με χαμηλή κατανάλωση ενέργειας.

Εξαιρετικά σημαντικό χαρακτηριστικό που πρέπει να έχει μια τεχνολογία με εφαρμογές στο ***IoT*** είναι η χαμηλή κατανάλωση ενέργειας. Για αυτόν τον λόγο πρωτόκολλα ***Low Power Wide Area Networking*** προτιμώνται για το χαμηλό κόστος δικτύου και το μεγάλο εύρος κάλυψης. Μπορεί είτε να χρησιμοποιηθεί απευθείας σύνδεση με κάποιο ***base station*** που είναι υπεύθυνο να δρομολογήσει την κίνηση στους ***backend server***, είτε να χρησιμοποιηθεί κάποιο τοπικό ***getaway*** που θα γεφυρώνει το ***LPWAN*** σε κάποια τεχνολογία μικρής συχνότητας. Μερικά από αυτά είναι [3]:

- ***SigFox*** το οποίο χρησιμοποιείται για ***uplink*** σε αισθητήρες με πολύ βασικές λειτουργίες και χρησιμοποιεί ***Binary Phase Shifting Key*** με πολύ χαμηλό ρυθμό μετάδοσης.
- ***Weightless***, μια σειρά από 3 ***LPWAN Open Standard*** με διαφορετικά χαρακτηριστικά και σενάρια χρήσης.
- ***LoRa***, μια τεχνική διαμόρφωσης που επιτρέπει την μετάδοση σε μεγάλες αποστάσεις με χαμηλή κατανάλωση ενέργειας.

Ακόμα χρησιμοποιούνται ενσύρματες τεχνολογίες για την επικοινωνία ***M2M***, όπως [2]:

- ***FTTx***, αδιαμφισβήτητα η καλύτερη επιλογή για ενσύρματη επικοινωνία αλλά εξαιρετικά ακριβή η δημιουργία καινούργιας υποδομής που θα την υποστηρίζει.
- ***xDSL*** το οποίο προσφέρει την καλύτερη αναλογία κόστους και ταχύτητας. Είναι η πιο διαδεδομένη ενσύρματη τεχνολογία αυτή την στιγμή.
- ***PLC*** όπου χρησιμοποιεί τις υπάρχουσες υποδομές μεταφοράς ηλεκτρικού ρεύματος αλλά έχει πολύ υψηλό ***SNR***.

Η επιλογή του τύπου τεχνολογίας (ασύρματη ή ενσύρματη) γίνεται με γνώμονα τον τύπο της εφαρμογής, δηλαδή αν θα είναι σταθερή, με περιορισμένη κινητικότητα, ή φορητή. Ακόμα είναι σημαντικό να σημειώσουμε ότι μεγάλο ρόλο έχουν παίξει και τεχνολογίες σαν το ***SDN*** και το ***NVF***. Στο ***SDN*** η διαχείριση του δικτύου αποσυνδέεται απλό τις υποκείμενες συσκευές δικτύου και ενσωματώνεται σε ένα στοιχείο λογισμικού. Αυτό προσφέρει ευελιξία, κεντρικοποιημένο έλεγχο, μειωμένη πολυπλοκότητα, και ευκολία χρήσης [2]. Το ***NVF*** χρησιμοποιεί εικονικοποίηση για να παρέχει δικτυακές λειτουργίες χωρίς να χρειάζεται επιπλέον υλικό και έτσι η πολυπλοκότητα και το κόστος μειώνονται [2].

1.2.1 ΔΙΚΤΥΑ BAN

Τα ***Body Area Networks*** είναι δίκτυα αποτελούμενα από φορετές (wearables) συσκευές, οι οποίες μπορεί να είναι εμφυτευμένες στο σώμα, τοποθετημένες πάνω στο σώμα η μια μικρή συσκευή που μπορεί να κρατηθεί στα χεριά η και να τοποθετηθεί σε μια τσέπη, τσάντα. Οι κύριες εφαρμογές τέτοιων συσκευών είναι ιατρικού σκοπού όπου μπορούν να παρέχουν ζωντανά δεδομένα από τον ασθενή, συνήθως στην μορφή ενός εμφυτεύματος. Ένα τέτοιο παράδειγμα είναι η αντλία ινσουλίνης που όταν ανιχνεύει μια αύξηση στα επίπεδα της ινσουλίνης, τότε αυτόματα εγχέει την σωστή δοσολογία για να επιστρέψει στα φυσιολογικά επίπεδα. Ακόμα υπάρχουν συσκευές ικανές να ανιχνεύσουν μια επερχόμενη καρδιακή προσβολή και να ειδοποιήσουν τόσο τον ασθενή όσο και το κοντινότερο νοσοκομείο [2].

Το τελευταίο πρότυπο για τα ***Wireless Body Area Network*** είναι το ***802.15.6*** και προσφέρει χαμηλή κατανάλωση ενέργειας, μικρού μήκους εμβέλεια, ***QoS,*** και αξιόπιστη ασύρματη επικοινωνία με ρυθμούς μεταφοράς μέχρι 10 Mbps. Είναι συχνό να υπάρχει αλληλεπίδραση των δικτυών αυτών με το διαδίκτυο, άλλες ασύρματες τεχνολογίες, ή κινητά δίκτυα [4].

1.2.2 ΑΣΦΑΛΕΙΑ ΣΤΑ BAN

Η ασφάλεια έχει επίσης βελτιωθεί από προηγούμενες γενιές, με το πρωτόκολλο να υποστηρίζει 3 επίπεδα ασφάλειας [5]:

- ***Unsecured Communication Level***, το χαμηλότερο επίπεδο ασφάλειας όπου τα δεδομένα μεταφέρονται σε ακάλυπτα πλαίσια (***frames***).
- ***Authentication Level***, το μεσαίο επίπεδο ασφάλειας οπού τα δεδομένα μεταφέρονται με ασφαλή έλεγχο ταυτότητας

- ***Authentication and Encryption***, το μεγαλύτερο επίπεδο ασφάλειας, όπου τα δεδομένα μεταφέρονται με ασφαλή έλεγχο ασφάλειας, και σε κρυπτογραφημένα πλαίσια.

Η επιλογή του επιπέδου ασφάλειας γίνεται κάθε φορά κατά την διαδικασία σύνδεσης. Στην περίπτωση της μονοεκπομπής (***unicast***) ένα παλιό η ένα καινούργιο ***Master Key*** ενεργοποιείται για την ασφαλή επικοινωνία, και στην συνέχεια ένα ***Pairwise Temporal Key*** δημιουργείται για αυτή την συνέδρια. Στην πολυεκπομπή (***multicast***) δημιουργείται ένα ***Group Temporal Key*** το οποίο διανέμεται με την μέθοδο της μονοεκπομπης που περιεγράφηκε πριν. Σύμφωνα με τον ***IEEE***, υπάρχουν μερικές ανοιχτές προτάσεις για την βελτίωση της ασφάλειας στα ***WBAN***.

1.2.3 ΑΡΧΙΤΕΚΤΟΝΙΚΗ BAN

Δεδομένου ότι τα ***BAN*** αφορούν την ασύρματη επικοινωνία μέσα η γύρω από ένα ανθρώπινο σώμα, μπορούμε να χωρίσουμε την αρχιτεκτονική επικοινωνιών σε 3 επίπεδα [5]:

- ***Intra-WBAN***. Σε αυτό το επίπεδο εμπίπτουν όλες οι συσκευές μέσα και γύρω από το σώμα σε ακτίνα 2 μέτρων συμπεριλαμβανομένου του ***Personal Server***. Αφού συλλεχθούν τα δεδομένα προωθούνται στο ***PS***, συνήθως μια κινητή συσκευή, και στην συνέχεια προωθούνται σε ένα ***Access Point*** που βρίσκεται στο δεύτερο επίπεδο.
- ***Inter-WBAN***. Το δεύτερο επίπεδο αφορά την επικοινωνία ενός ***PS*** μεταξύ ενός ή πολλών ***AP***. Σκοπός αυτού του επιπέδου είναι να προσφέρει συνδεσιμότητα στα ***WBAN*** δίκτυα με τον έξω κόσμο, καθιστώντας τα εύκολα στην χρήση από υπηρεσίες η εφαρμογές τρίτων. Σε αυτό το σημείο μπορούμε να χωρίσουμε το δεύτερο επίπεδο σε δυο υποκατηγορίες:
    - ***Infrastructure based architecture*** όπου προσφέρει κεντρικοποιημένο έλεγχο με μεγαλύτερο εύρος ζώνης.

- ***Ad-hoc based architecture*** όπου πολλαπλά ***AP*** χρησιμοποιούνται με σκοπό να δημιουργήσουν ένα πλέγμα, και έτσι να αυξήσουν την ζώνη κάλυψης και να επιτρέψουν μεγαλύτερη κινητικότητα.

- ***Beyond-WBAN***. Το τρίτο και τελευταίο επίπεδο έχει σχεδιαστεί για μητροπολιτική χρήση όπου ένα ***getaway*** χρησιμοποιείται για να ενώσει το επίπεδο 2 με αυτό. Αυτό το επίπεδο σχεδιάζεται με γνώμονα την εκάστοτε εφαρμογή και την λειτουργικότητα που θέλει να παρέχει.

1.2.4 ΑΣΥΡΜΑΤΕΣ ΤΕΧΝΟΛΟΓΙΕΣ

Τα δίκτυα ***WBAN*** εμπλέκουν διάφορες τεχνολογίες στα επίπεδα που περιγράφονται στο 1.2.2. Αυτές οι τεχνολογίες μπορεί να είναι προ υπάρχουσες που έχουν προσαρμοστεί για την χρήση αυτή η εξειδικευμένες για τα δίκτυα ***WBAN***. Παρακάτω θα κάνουμε μια σύντομη αναφορά σε μερικές από αυτές [2][6].

1.2.4.1 BLUETOOTH

Το ***Bluetooth*** είναι μια ασύρματη τεχνολογία ανταλλαγής δεδομένων μικρού εύρους η οποία λειτουργεί στην μπάντα των 2.4GHz, με μέγιστο ρυθμό μεταφοράς τα 3 Mbps. Μια συσκευή μπορεί να επικοινωνήσει με άλλες 7, δημιουργώντας έτσι ένα ***ad-hoc*** δίκτυο το οποίο ονομάζεται ***piconet*** [7].

1.2.4.2 BLUETOOTH LOW ENERGY (BLE)

Το ***BLE*** αποτελεί μια παράγωγο του κλασσικού ***Bluetooth*** που στοχεύει σε εφαρμογές που η εξοικονόμηση ενέργειας είναι βασικός παράγοντας. Δεν είναι συμβατό με το ***Bluetooth*** και έχει μειωμένο ρυθμό μεταφοράς, μέχρι 1Mbps. Λειτουργεί στην ιδιά μπάντα, και έχει μειωμένη απόκριση αφού χρησιμοποιεί λιγότερα κανάλια για να συζεύξει συσκευές, μειώνοντας έτσι και τον χρόνο σύζευξης από μερικά δευτερόλεπτα σε μερικά χιλιοστά του δευτερολέπτου [8].

1.2.4.3 ZIGBEE

Το ***ZigBee*** είναι μια ευρέως διαδεδομένη τεχνολογία σε εφαρμογές που απαιτείται χαμηλή κατανάλωση ενέργειας. Διαθέτει ***sleep mode*** που επιτρέπει να κλείνει τον πομπό του όποτε δεν χρειάζεται. Βέβαια έχει δυο βασικά μειονεκτήματα, τον χαμηλό ρυθμό μεταφοράς των 250Kbps που το καθιστά άχρηστο για ζωντανές εφαρμογές μεγάλης κλίμακας καθώς και παρεμβολές αφού λειτουργεί στην μπάντα των 2.4GHz [9].

1.2.4.4 ULTRA WIDE BAND (UWB)

Το ***UWB*** είναι μια ασύρματη τεχνολογία που χρησιμοποιείται για χαμηλής ενέργειας, μικρού εύρους και υψηλής ταχύτητας μεταφορά δεδομένων στην μπάντα των 3.1-10.6GHz. Είναι μια αξιόπιστη μέθοδος με ρυθμούς μεταφοράς μέχρι και 480Mbps [10].

1.2.4.5 ANT

Το ***ANT*** είναι ένα πρωτόκολλο το οποίο δημιουργήθηκε από την ***Garmin*** με βασικό πεδίο εφαρμογής τον αθλητισμό και τους αισθητήρες υγείας. Έχει χαμηλή κατανάλωση ενέργειας,

ρυθμούς μεταφοράς μέχρι 1Mbps και λειτουργεί στην μπάντα των 2.4GHz. Υπάρχει επίσης το ***ANT+*** το οποίο είναι μια λειτουργία που μπορεί να προστεθεί στο βασικό ***ANT***, και χρησιμοποιείται σε αισθητήρες παλμών, βαθόμετρα, ζυγαριές και αλλά [11].

1.3.1 ΔΙΚΤΥΑ PAN

Τα ***Personal Area Networks*** είναι δίκτυα που αφορούν συσκευές που καλύπτουν την περιοχή που καταλαμβάνει ένα άτομο, τέτοιες συσκευές δηλαδή μπορεί να είναι ένας προσωπικός υπολογιστής, ένα κινητό τηλέφωνο η ένα ποντίκι υπολογιστή, βιομετρικοί αισθητήρες. Το εύρος κάλυψης τους είναι μέχρι μερικά μέτρα, και η σύνδεση μπορεί να είναι ενσύρματη η ασύρματη. Πολλές τεχνολογίες που χρησιμοποιούνται στα δίκτυα ***BAN*** είναι κατάλληλες να εφαρμοστούν και στα δίκτυα ***PAN***. Για αυτόν τον λόγο θα παραλειφθεί η ανάλυση τεχνολογιών που έχουν προηγουμένως αναφερθεί. Πιο συγκεκριμένα θα αναφερθούμε στο πρότυπο ***802.15.4*** το οποίο περιγράφει τα ***LRWPAN***, δίκτυα τα οποία έχουν ευρεία εφαρμογή στο πεδίο του ***IoT***.

Το πρότυπο αυτό ορίζει τα χαρακτηριστικά του ***Physical*** και ***MAC*** επιπέδου. Τα προτερήματα του είναι η ευκολία εγκατάστασης, η αξιόπιστη μεταφορά δεδομένων, ενώ ταυτόχρονα διατηρεί χαμηλό κόστος και μεγάλη διάρκεια μπαταρίας. Είναι σημαντικό να σημειώσουμε ότι επιτρέπει μεγάλη ευελιξία στην στοίβα πρωτοκόλλου, πράγμα που το καθιστά ιδανικό δομικό στοιχείο για πρωτόκολλα που αξιοποιούν τα υψηλοτέρα επίπεδα όπως το ***ZigBee*** ή το ***6LoWPAN***.

1.3.2 ΤΟΠΟΛΟΓΙΕΣ 802.15.4

Σύμφωνα με την ***IEEE*** ορίζονται 2 τοπολογίες για τέτοιου είδους δίκτυα. Κάθε συσκευή στο δίκτυο αυτό, ανεξάρτητου τοπολογίας, έχει ένα μοναδικό αναγνωριστικό το οποίο ονομάζεται ***Extended Address***. Πιο συγκεκριμένα [2][12]:

- ***Star topology***. Στην τοπολογία αστέρα κάθε συσκευή επικοινωνεί με έναν κεντρικό ελεγκτή ο οποίος ονομάζεται ***PAN Coordinator***. Αυτή η τεχνική περιορίζει την απόσταση σε 1 ***hop***.
- ***Peer to Peer topology***. Στην τοπολογία ***P2P*** ο ***PAN Coordinator*** παραμένει, αλλά οι συσκευές έχουν την δυνατότητα να επικοινωνήσουν η μια με την άλλη εφόσον είναι η μια στο εύρος της άλλης. Αυτό επιτρέπει να σχηματιστούν πιο πολύπλοκες τοπολογίες όπως είναι η ***Mesh*** τοπολογία και να καταργηθεί το όριο του 1 ***hop***.

Η επιλογή τοπολογίας έχει να κάνει με τις απαιτήσεις της εφαρμογής. Είναι επίσης σημαντικό να διευκρινίσουμε ότι ο σχηματισμός του δικτύου γίνεται από μεγαλύτερα επίπεδα και δεν περιγράφεται από αυτό το πρότυπο.

1.3.3 PHYSICAL ΚΑΙ MAC ΕΠΙΠΕΔΟ

Το πρότυπο ***802.15.4*** ορίζει 2 επίπεδα, ένα τουλάχιστον φυσικό επίπεδο το οποίο περιέχει τον πομπό και τον χαμηλού επιπέδου έλεγχο, και το ***MAC*** επίπεδο που επιτρέπει σε υψηλοτέρα επίπεδα να έχουν πρόσβαση στο φυσικό κανάλι και την μεταφορά του. Τα υψηλοτέρα επίπεδα είναι το επίπεδο δικτύου και το επίπεδο εφαρμογής τα οποία δεν περιγράφονται από αυτό το πρότυπο, αλλά τα αναλαμβάνει το εκάστοτε πρωτόκολλο το οποίο έχει χτιστεί πάνω στο ***802.15.4*** [4]. Αναλυτικότερα:

- ***Physical***. Σε αυτό το επίπεδο προσφέρονται 3 συχνότητες, η 2450MHz, η 915MHz και η 868MHz, αντίστοιχα με ρυθμό δεδομένων 250kbps, 40kbps, 20kbps. Η πιο διάσημη είναι η μπάντα των 2.4GHz καθώς επιτρέπεται η χρήση της παγκοσμίως. Βέβαια είναι σε συζήτηση η αποδοχή καινούργιων συχνοτήτων για το πρότυπο αυτό, στο εύρος των 300-800MHz.
- ***MAC***. Το υποεπίπεδο ***MAC*** προσφέρει 2 υπηρεσίες την ***Mac Data Service*** (η ***MAC Common Port Layer - MCPS***) που επιτρέπει την μεταφορά ***MAC Protocol Data Units*** (***MPDUs***), και το ***MAC Management Service*** που επιτρέπει τον έλεγχο λειτουργιών.

Να σημειωθεί ότι το ***802.15.4*** μπορεί να διαχειριστεί μεγάλο αριθμό συσκευών αλλά δεν προσφέρεται κανένα QoS.

1.3.4 ΑΣΥΡΜΑΤΕΣ ΤΕΧΝΟΛΟΓΙΕΣ

Μερικές από τις τεχνολογίες που χρησιμοποιούν το 802.15.4 είναι:

- ***WirelessHART***, δουλεύει στην μπάντα των 2.4 και έχει συγχρονισμένη, αυτό-οργανωμένη και αυτό-θεραπευτική αρχιτεκτονική πλέγματος. Έχει βασιστεί στο πρότυπο ***HART*** το οποίο είναι ένα υβριδικό βιομηχανικό πρότυπο (αναλογικό και ψηφιακό) [14].
- ***MiWi***, είναι ένα πρότυπο το οποίο χρησιμοποιείται για αυτοματισμούς στο σπίτι η σε βιομηχανικό περιβάλλον και υποστηρίζει την μπάντα των 2.4GHz και την sub-GHz [15].
- ***6LoWPAN***, είναι ένα ανοιχτό πρότυπο το οποίο χρησιμοποιεί την έκδοση ***IPv6*** αντί ***IPv4***. Παρόλο που αρχικά σχεδιάστηκε να υποστηρίζει μόνο την μπάντα των 2.4GHz, πλέον έχει διευρυνθεί και επιτρέπει και την χρήση sub-GHz [16].

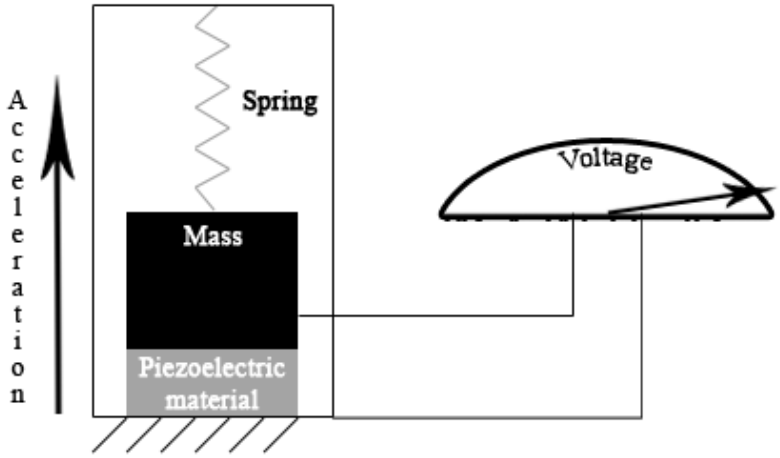
ΑΙΣΘΗΤΗΡΕΣ ΣΥΛΛΟΓΗΣ ΔΕΔΟΜΕΝΩΝ

2.1 ΕΙΣΑΓΩΓΗ ΣΤΟΥΣ ΑΙΣΘΗΤΗΡΕΣ

Ένας αισθητήρας είναι μια συσκευή ικανή να ανιχνεύσει αλλαγές στο περιβάλλον, φυσικά φαινόμενα όπως είναι η αλλαγή θερμοκρασίας, και να το μετατρέψει σε ηλεκτρικό σήμα το οποίο μπορούμε να αξιοποιήσουμε. Αντίστοιχα υπάρχουν και οι ενεργοποιητές (actuators), οι οποίοι ερμηνεύουν ένα ηλεκτρικό σήμα σε μια μηχανική κίνηση, όπως θα ήταν το άνοιγμα και το κλείσιμο μιας βαλβίδας. Αυτά τα δυο είναι αδιάσπαστο κομμάτι του ***IoT***, και είναι αυτά που του δίνουν την επαφή με τον πραγματικό κόσμο [17]. Στο συγκεκριμένο κεφάλαιο θα εξετάσουμε τους αισθητήρες, και πιο συγκεκριμένα τους αισθητήρες με εφαρμογές στον αθλητισμό.

2.2 ΕΠΙΤΑΧΥΝΣΙΟΜΕΤΡΟ

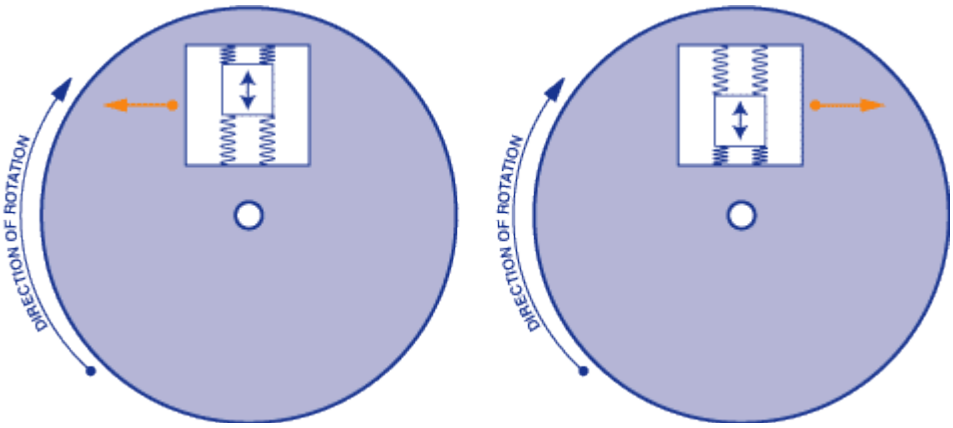
Το επιταχυνσιόμετρο είναι ο πιο κοινός αισθητήρας που μπορεί να βρεθεί σε μια συσκευής. Τα επιταχυνσιόμετρα μπορούν να ανιχνεύσουν την επιτάχυνση ενός αντικειμένου, και συνήθως μετράνε τις αλλαγές της ταχύτητας σε 3 κατευθύνσεις, βέβαια αυτό εξαρτάται ανάλογα την εφαρμογή του και τον τύπο. Μπορούν να χρησιμοποιηθούν για διάφορους σκοπούς, από την παρακολούθηση του ύπνου μέχρι και σε αεροσκάφη για τον προσδιορισμό τοποθεσίας. Στις αθλητικές εφαρμογές μπορεί να χρησιμοποιηθεί για να ανιχνεύσει την κίνηση, μαζί με την επιτάχυνση και την κατεύθυνση.



Εικόνα 2.1 Εσωτερικό Πιεζοηλεκτρικού επιταχυνσιόμετρου από ***Sparkfun***.

2.3 ΓΥΡΟΣΚΟΠΙΟ

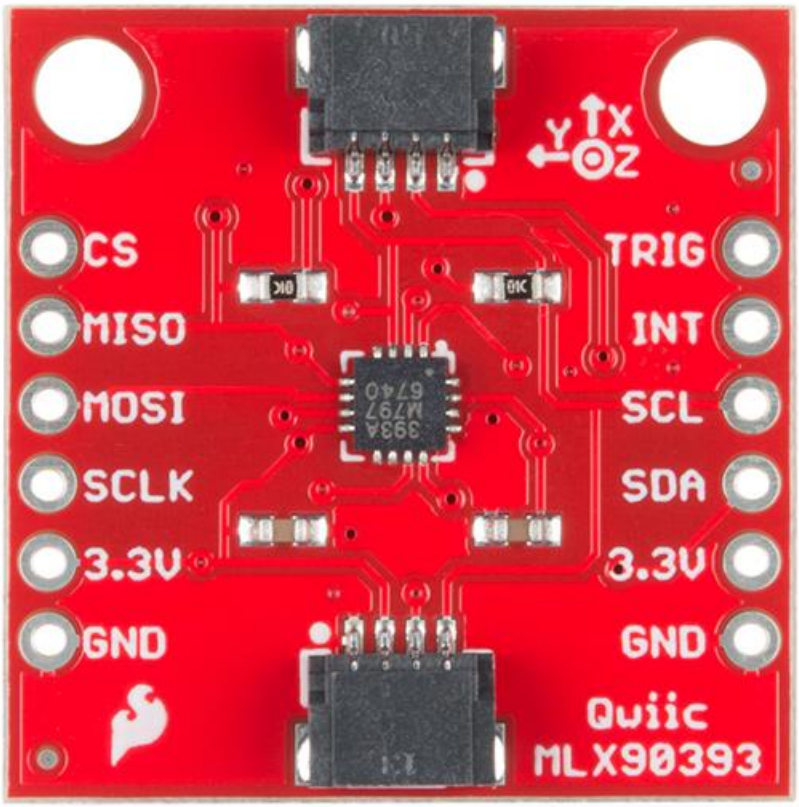
Άλλη μια κοινή συσκευή είναι το γυροσκόπιο. Το γυροσκόπιο μοιάζει στο επιταχυνσιόμετρο αλλά διαφέρει στο γεγονός ότι μετράει μόνο τις γωνιακές επιταχύνσεις. Υπάρχουν 3 τύποι γυροσκοπίων, μηχανικά, με ρουλεμάν αερίου, και οπτικά [18]. Τα κλασικά γυροσκόπια (μηχανικά και ρουλεμάν αερίου) έχουν έναν περιστρεφόμενο δίσκο ο οποίος είναι ελεύθερος να προσανατολιστεί μόνος του. Τα οπτικά γυροσκόπια λειτουργούν διαφορετικά, όπου δύο πηνία οπτικών ινών περιστρέφονται σε εναλλασσόμενες κατευθύνσεις και παρακολουθείται η διαφορά απόστασης μεταξύ τους (***Sagnac Effect***).



*Εικόνα 2.2 Εσωτερικό **MEMS** γυροσκόπιου από **Sparkfun**.*

2.4 ΜΑΓΝΗΤΟΜΕΤΡΑ

Το μαγνητόμετρο μετρά την μαγνητική διπολική ροπή ή το μαγνητικό Πεδίο (***Hall Effect***). Ένα τέτοιο παράδειγμα είναι η πυξίδα, που μετρά το μαγνητικό πεδίο της γης. Έχουν ευρύ πεδίο χρήσης, από ανιχνευτές μετάλλων μέχρι και σε δορυφόρους για να μετρηθεί το μέγεθος και η κατεύθυνση του μαγνητικού πεδίου ενός πλανήτη.

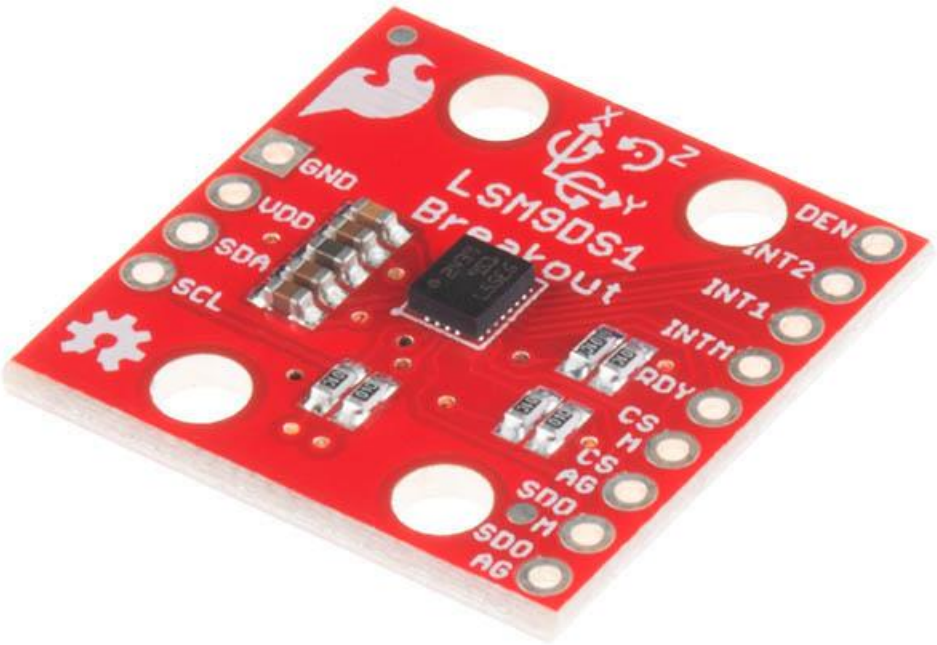


Εικόνα 2.3 **Triple Axis** *μαγνητόμετρο από* **Sparkfun**.

2.5 IMU

Οι αισθητήρες ***IMU*** αποτελούνται από επιταχυνσιόμετρα, γυροσκόπια, και μερικές φορές μαγνητόμετρα. Το επιταχυνσιόμετρο χρησιμοποιείται για να ανιχνεύσει την γραμμική επιτάχυνση, το γυροσκόπιο ανιχνεύει τον ρυθμό περιστροφής, και το μαγνητόμετρο για σημείο αναφοράς. Τα ***IMU*** κατηγοριοποιούνται βάση του ***Degree Of Freedom***, όπου είναι ο αριθμός των ανεξάρτητων παραμέτρων σε ένα σύστημα. Τα συστήματα 2 αισθητήρων έχουν ***6-DOF***, ενώ τα συστήματα 3 αισθητήρων έχουν ***9-DOF*** [19][20]. Σε γενικές γραμμές όσο μεγαλύτερος είναι ο αριθμός των

DOF, τόσο πιο ακριβές είναι το δείγμα. Οι ξεχωριστές μετρήσεις από τους αισθητήρες ενός ***IMU***, συνδυάζονται μέσω φίλτρων και αλγορίθμων ώστε να εξαλειφθεί ο θόρυβος και η μετατόπιση που μπορεί να έχουν οι αισθητήρες. Οι πρώτες εφαρμογές των ***IMU*** ήταν σε αεροσκάφη την δεκαετία του 1930, αλλά σήμερα έχουν εφαρμογές σε ιατρικές συσκευές, συστήματα πλοήγησης, αθλήματα, και πολλά ακόμα [21]. Τα περισσότερα σύγχρονα IMU είναι τύπου ***MEMS***, που είναι φθηνοί σε παραγωγή μεγάλης κλίμακας.



Εικόνα 2.4 ***9-DOF Sparkfun IMU*** *με* επιταχυνσιόμετρο, γυροσκόπιο, και μαγνητόμετρο.

2.6 ΠΑΛΜΟΓΡΑΦΟΙ

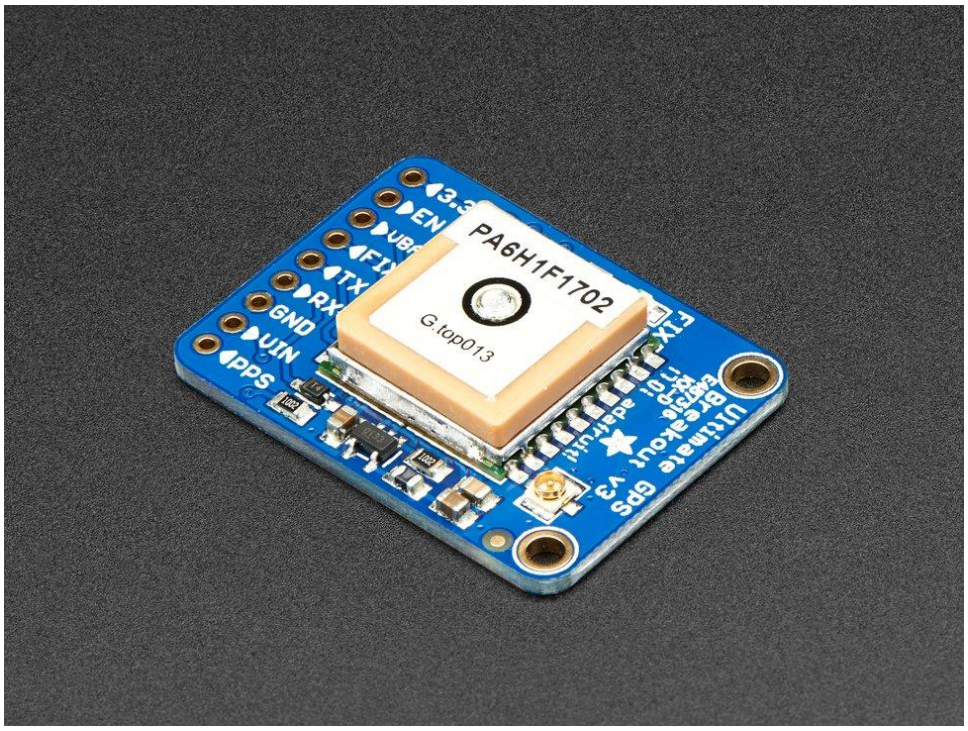
Υπάρχουν 2 είδη παλμογράφων, ηλεκτρικοί και οπτικοί. Οι αισθητήρες ***ECG***, που μετρούν το βίο-δυναμικό που παράγεται από ηλεκτρικά σήματα που ελέγχουν την επέκταση και τη συστολή των καρδιακών θαλάμων, και οι αισθητήρες ***PPG*** που χρησιμοποιούν τεχνολογία βασισμένη στο φως για τη μέτρηση του όγκου του αίματος που ελέγχεται από την αντλία της καρδιάς [19]. Η πρώτη κατηγορία συναντάτε συνήθως σε ιατρικές συσκευές ή στις κλασικές ζώνες παλμογράφων που χρησιμοποιούν σε πολλά αθλήματα, ενώ η δεύτερη κατηγορία συναντάτε πιο συχνά σε ρολόγια και κινητά τηλεφωνά. Οι οπτικοί αισθητήρες τείνουν να είναι πιο ανακριβείς, και να αργούν πολλές φορές να ανιχνεύσουν τις απότομες αλλαγές στους καρδιακούς παλμούς.

Εικόνα 2.5 Ζώνη παλμογράφος **Garmin,** *που τοποθετείται στο στήθος.*

2.7 GPS

Το ***GPS*** είναι ένα δορυφορικό σύστημα ραδιοπλοήγησης, ικανό να παρέχει την γεωγραφική τοποθεσία με ακρίβεια 30 εκατοστών μαζί με πληροφορίες χρόνου, σε έναν δέκτη ο οποίος είναι συνδεμένος με τουλάχιστον 4 τέτοιους δορυφόρους [19]. Ο δέκτης μπορεί να υπολογίσει την θέση

του και την ώρα από δεδομένα που λαμβάνει από τους δορυφόρους, οι οποίοι είναι εξοπλισμένοι με απόλυτα συγχρονισμένα μεταξύ τους ατομικά ρολόγια καθώς και την ακριβή τοποθεσία τους. Αν και αρχικά χρησιμοποιούταν αποκλειστικά από τον αμερικάνικο στρατό, την δεκαετία του 1980 έγινε δημόσια διαθέσιμο [22]. Στον αθλητισμό έχει ευρεία εφαρμογή καθώς επιτρέπει στους αθλητές να καταγράφουν τις δραστηριότητες τους με μεγάλη ακρίβεια, πράγμα που επιτρέπει την στοχεύμενη και εξατομικευμένη προπόνηση. Όλες οι σύγχρονες συσκευές, όπως κινητά και ρολόγια, έρχονται εξοπλισμένες με ***GPS***.



Εικόνα 2.6 ***Adafruit GPS*** *66 καναλιών με ρυθμό ανανέωσης 10****Hz****.*

2.8 ΒΑΤΟΜΕΤΡΟ

Το βατόμετρο (***Power Meter***) είναι μια συσκευή που εφαρμόζει σε ποδήλατο, και μετράει την απόδοση ισχύος του αναβάτη. Τα περισσότερα βατόμετρα χρησιμοποιούν μετρητές πίεσης για να μετρήσουν την ροπή και σε συνδυασμό με την γωνιακή ταχύτητα μπορούν να υπολογίσουν την δύναμη που εφαρμόζεται σε ***w*att**. Τοποθετούνται στα πετάλια, στην μεσαία τριβή, στην αράχνη, ή στο κέντρο του πίσω τροχού. Χρησιμοποιείται κυρίως κατά την διάρκεια της προπόνησης ώστε ο αθλητής να μπορεί να προπονηθεί στις κατάλληλες ζώνες με ακρίβεια κατά την διάρκεια της διαλειματικής προπόνησης. Επίσης βοηθάει να εξαλείψει ανακρίβειες που παρατηρούνται μόνο με την χρήση παλμογράφου. Για παράδειγμα όταν ο αθλητής είναι κουρασμένος οι παλμοί τείνουν να αποκλίνουν από τους παλμούς που θα είχε για την ιδιά στάθμη δύναμης αλλά ξεκούραστος. Έχει επίσης βρεθεί από έρευνες ότι η δύναμη που εφαρμόζεται μπορεί να χρησιμοποιηθεί σε μαθηματικά μοντέλα και συνεπώς να προβλέψουν με ακρίβεια την ταχύτητα [23].

Εικόνα 2.7 Βατόμετρο μεσαίας τριβής ***Rotor INpower***.

VELON - ΜΕΛΕΤΗ ΠΕΡΙΠΤΩΣΗΣ

3.1 ΤΙ ΕΙΝΑΙ ΤΟ VELON

Το ***Velon*** αποτελεί μια πρωτοποριακή εφαρμογή που έχει μεταμορφώσει τον επαγγελματικό χώρο της αγωνιστικής ποδηλασίας. Χρησιμοποιεί τους ήδη υπάρχοντες αισθητήρες του ποδήλατου και του αθλητή, για να μεταφέρουν σε πραγματικό χρόνο δεδομένα που έχουν να κάνουν με την ταχύτητα, την δύναμη, τον ρυθμό πεταλιάς και τους καρδιακούς παλμούς. Στην συνέχεια αυτά χρησιμοποιούνται μέσω μιας διαδικτυακής εφαρμογής, στην ζωντανή κάλυψη αγώνων, και σαν γραφήματα. Το μεγαλύτερο κομμάτι των ομάδων στην επαγγελματική ποδηλασία συνεργάζεται με το ***Velon***, πιο συγκεκριμένα 11 από τις 19 ομάδες, προσφέροντας μια καινούργια όψη του αθλήματος στους θεατές.

3.2 ΤΟ ΠΡΟΒΛΗΜΑΤΙΚΟ ΜΟΝΤΕΛΟ ΤΗΣ ΠΟΔΗΛΑΣΙΑΣ

Η επαγγελματική ποδηλασία έχει ένα πλούσιο πρόγραμμα αγώνων, αποτελούμενο από συντόμους, νεοσύστατους αγώνες σαν το ***Hammer Series*** μέχρι τους ιστορικούς και επίπονους σαν τον γύρο της Γαλλίας. Παρόλο το μεγάλο κοινό και την παγκόσμια πλέον τηλεθέαση αντιμετωπίζει σοβαρό πρόβλημα νομισματοποίησης. Σύμφωνα με το ***Ernst & Young Global Limited (EY)*** [24] το ποδήλατο έχει εμπορικό εισόδημα μόλις 600εκ. Δολάρια, εν αντιθέσει με το ***NFL*** και τις πρώτες 5 ομάδες ευρωπαϊκού ποδοσφαίρου που έχουν εισόδημα 8.2 και 16.8 εκατομμύρια δολάρια αντίστοιχα. Το πρόβλημα αυτό ανέρχεται στο γεγονός ότι οι αγώνες αντοχής είναι μεγάλοι σε σκέλος, φτάνοντας μέχρι και 305 χιλιόμετρα στην περίπτωση του ***Milano-Sanremo***. Αυτό σημαίνει ότι ο θεατής μπορεί να βρεθεί να περιμένει πολλές ώρες στην άκρη του δρόμου απλά για να δει το πελατών να περνάει στιγμιαία από μπροστά του. Συνεπώς όχι

μόνο οι θεατές δεν παρακολουθούν όλον τον αγώνα, αλλά και οι οργανώσεις δεν μπορούν να εκμεταλλευτούν έσοδα από εισιτήρια όπως θα μπορούσαν στην περίπτωση ενός ποδοσφαιρικού αγώνα, μέσα σε ένα ελεγχόμενο γήπεδο. Όλα τα παραπάνω σε συνδυασμό με ένα μοντέλο πληρωμής των ομάδων μόνο από χορηγούς, καταλήγει να δημιουργεί μια πολύ εύθραυστη αγορά.

Το ***Velon*** εκμεταλλεύεται την πληθώρα δεδομένων που παράγει ένας αθλητής. Πιο συγκεκριμένα, με τους ήδη υπάρχοντες αισθητήρες του αθλητή μαζί με στοιχεία εδάφους και τοποθεσίας μπορεί πλέον να κατασκευαστεί το προφίλ του κάθε αθλητή, οδηγώντας σε ελκυστικά στοιχεία για τον θεατή. Δημιουργήθηκε λοιπόν μια ***IoT*** λύση που συλλεγεί και μεταφέρει τον τεράστιο όγκο δεδομένων σε πραγματικό χρόνο.

3.3 ΣΥΝΕΧΟΜΕΝΗ ΠΑΡΑΚΟΛΟΥΘΗΣΗ ΑΘΛΗΤΩΝ

Το ποδήλατο αποτελεί μια ιδιαίτερη περίπτωση καθώς πολλές φορές οι αθλητές βρίσκονται σε περιοχές χωρίς δυνατότητα σύνδεσης (π.χ. ***LTE***). Ο σκοπός της ***Velon*** είναι να χρησιμοποιήσει τους αισθητήρες συμβατούς με ***ANT+*** και το ***GPS*** για συλλογή δεδομένων, και μετά να τα προωθήσει με χρήση τοπικών τηλεφωνικών δικτύων. Σε περίπτωση που αυτό δεν είναι δυνατό, τότε χρησιμοποιεί έναν ***buffer*** για τα δεδομένα και τα μεταδίδει με την επανασύνδεση το δίκτυο [25].

3.4 ΤΟ ΥΛΙΚΟ

Η συσκευή διαθέτει ***GPS***, ***ANT+***, ***bluetooth*** και επιταχυνσιόμετρο ενώ χρησιμοποιείται μια μικρή ***3D*** εκτυπωμένη θήκη για να στεγάσει όλον τον εξοπλισμό, που τοποθετείται σε ένα αεροδυναμικό σημείο του ποδήλατου, συνήθως κάτω από την σέλα. Η συσκευή αυτή εξελίσσεται

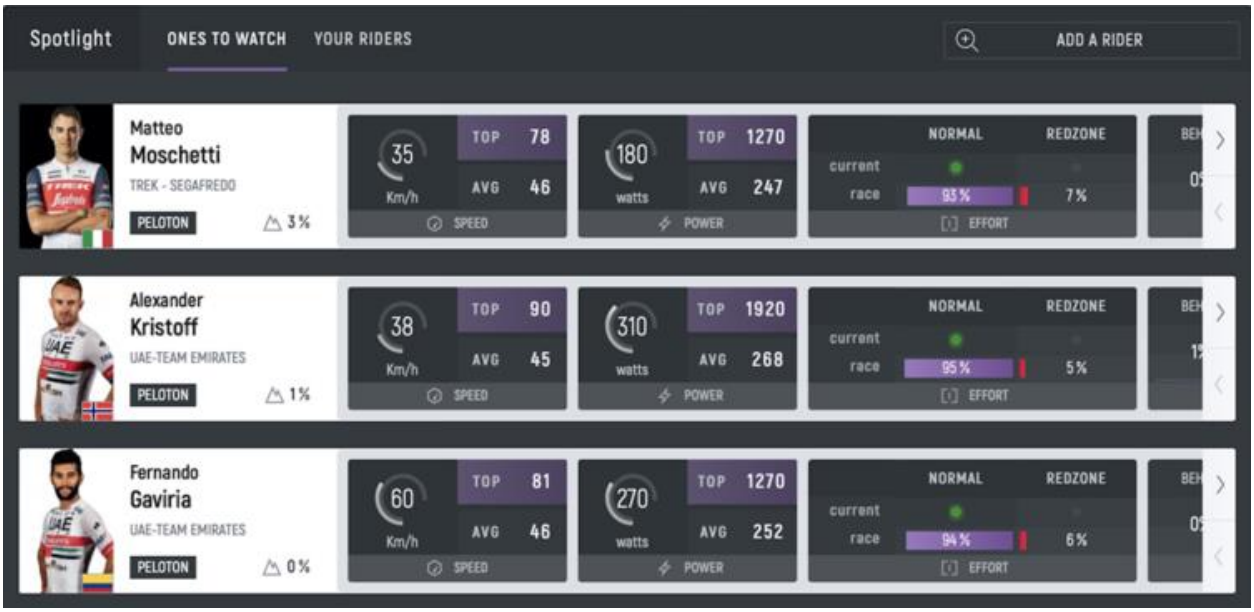
διαρκώς, αλλάζοντας σε σχήμα και δυνατότητες. Η ενσωματωμένη μπαταρία είναι ικανή για 12 ώρες λειτουργίας και η συνδεσιμότητα γίνεται με ***LTE*** στα τοπικά δίκτυα [26].

Εικόνα 3.1 Συσκευή εντοπισμού **Velon** *τοποθετημένη σε σέλα ποδήλατου.*

3.5 ΔΙΚΤΥΑΚΗ ΕΦΑΡΜΟΓΗ

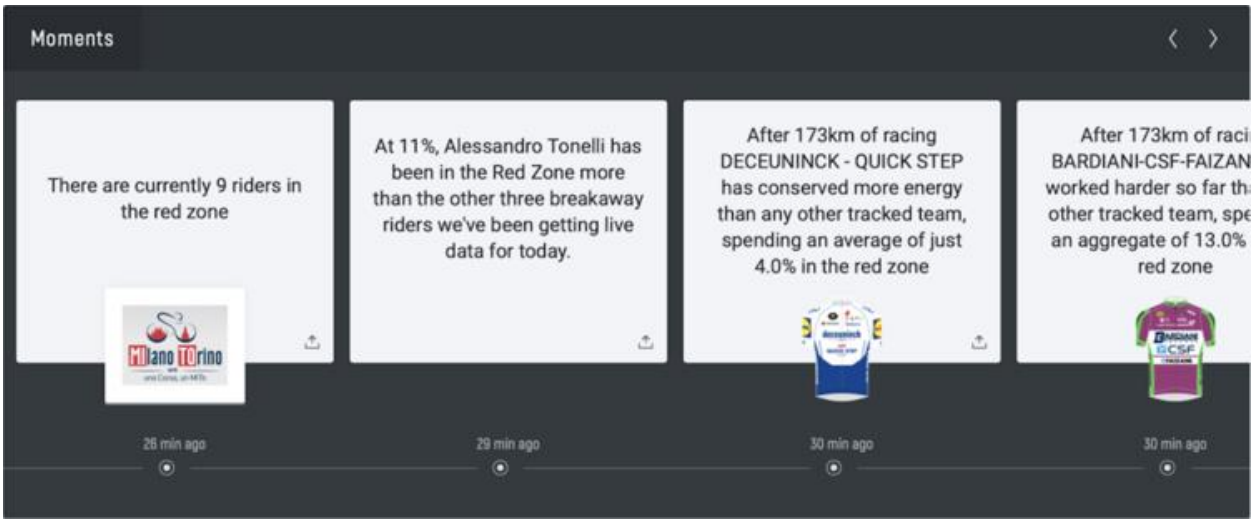
Ο χρήστης μπορεί να αλληλοεπιδράσει με την εφαρμογή, λαμβάνοντας δεδομένα από τους αγώνες ανά ένα δευτερόλεπτο. Αναλυτικότερα, όπως φαίνεται και στην εικόνα 3.2 ο χρήστης

μπορεί να επιλέξει μέχρι 10 αναβάτες για σύγκριση, και να δει σε πραγματικό χρόνο την τοποθεσία και τα δεδομένα τους [27].



Εικόνα 3.2 Συνοπτική απεικόνιση των αθλητών στην εφαρμογή **Velon**.

Ακόμα ένα χαρακτηριστικό, σύμφωνα με το ***Velon***, είναι τα ***moments***, μια μετάδοση με διάφορες στιγμές από τον αγώνα όπως είναι ο αναβάτης με την μεγαλύτερη μέγιστη ταχύτητα η την περισσότερη ώρα στο "κόκκινο".



Εικόνα 3.3 Τα **moments** *στο* **dashboard** *του* **Velon**.

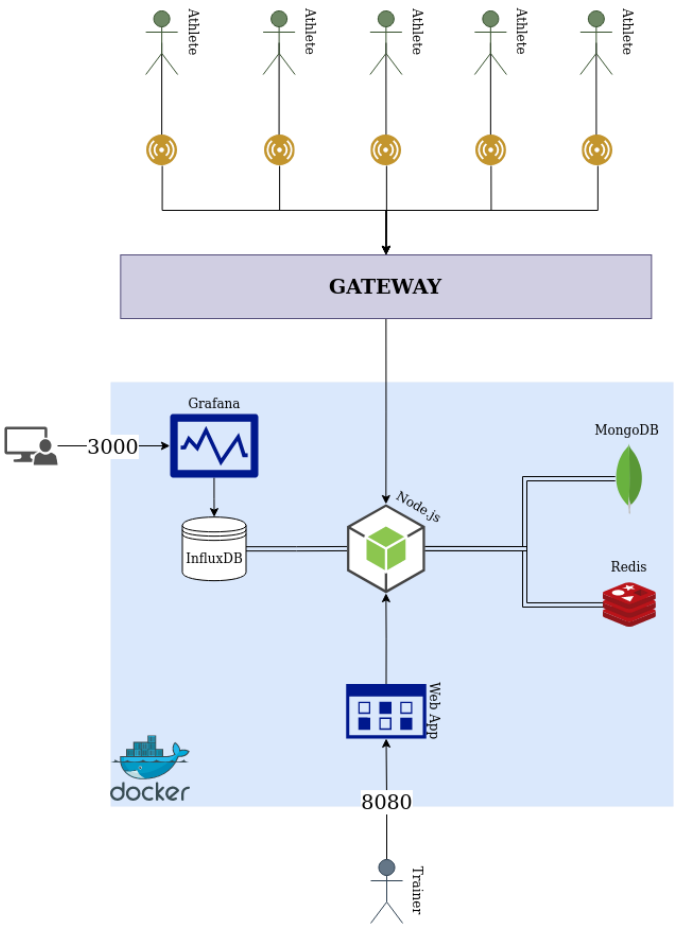
3.6 ΠΩΣ ΤΟ IoT ΑΛΛΑΞΕ ΤΗΝ ΠΟΔΗΛΑΣΙΑ

Καταλήγουμε λοιπόν στο συμπέρασμα ότι το ***IoT*** έχει μεταμορφώσει το μοντέλο της ποδηλασίας. Εταιρίες σαν το ***Velon*** και την ***Dimension Data*** επιτρέπουν στις ομάδες να εξασφαλίσουν ένα επιπλέον οικονομικό εισόδημα, και στους θεατές να βρίσκονται συνέχεια μέσα στον αγώνα. Όμως, το ***IoT*** στο ποδήλατο δεν σταματάει εκεί. Ένα ακόμα τρανό παράδειγμα είναι η συνεργασία της γυναικείας αμερικάνικης ομάδας πίστας με την ***IBM***. Πιο συγκεκριμένα μετράνε το επίπεδο οξυγόνου στους μυς, το παραγόμενο ***έργο*** και αρκετές ακόμα μετρικές, καθώς χρησιμοποιούν και έξυπνα γυαλιά [28]. Αυτό επιτρέπει στους προπονητές να συνεργάζονται με τους αθλητές σε πραγματικό χρόνο στην προσπάθεια να βελτιστοποιήσουν την απόδοσή της ομάδας. Το ***IoT*** συνεχίζει πέρα από την ποδηλασία, αθλήματα από το μπέιζμπολ μέχρι το ποδόσφαιρο προσπαθούν να βελτιώσουν τους αθλητές τους, να τους προστατέψουν από τραυματισμούς και να προσελκύσουν περισσοτέρους θαυμαστές.

ΕΦΑΡΜΟΓΗ E-SPORT

4.1 ICHNAEA

Στο πλαίσιο της διπλωματικής αυτής, σχεδιάστηκε και μια εφαρμογή τύπου '***e-sport***'. Η εφαρμογή αυτή ονομάζεται ΙΧΝΑΙΗ (***ICHNAEA***), η οποία ήταν μια γυναικά τιτάνας, θεότητα του εντοπισμού και της παρακολούθησης. Δεδομένου ότι η τάση των διαδικτυακών εφαρμογών κινείται προς ***microservices*** αρχιτεκτονικές, μαζί με ***Frontends*** που αποτελούνται από ***single-page Applications***, η ***ICHNAEA*** ενσωμάτωσε τεχνολογίες βασιζόμενες σε ***cloud native*** προσεγγίσεις όπως είναι τα ***Docker Containers***.



Εικόνα 4.1.1 Σχεδιάγραμμα αρχιτεκτονικής της ***ICHNAEA***.

Το παραπάνω σχεδιάγραμμα είναι μια επισκόπηση της αρχιτεκτονικής της ***ICHNAEA,*** και απεικονίζει τις βασικές τεχνολογίες και τις μεταξύ τους σχέσεις. Στις επόμενες υποενότητες θα αναλυθούν οι τεχνολογίες αυτες, η αρχιτεκτονική, καθώς και όλες οι επιλογές που έγιναν κατά την ανάπτυξη της εφαρμογής. Στο τέλος θα προσομοιώσουμε μερικά σενάρια για να δείξουμε την λειτουργία της εφαρμογής. Όλος ο πηγαίος κώδικας βρίσκεται στο ***repository xrazis/ichnaea*** στο ***github***. Στο παράρτημα αναλύονται μερικές έννοιες που θα συναντήσουμε στις παρακάτω ενότητες, όπως οι γωνίες ***Euler***.

4.2 SERVER

Ο ***Server*** της ***ICHNAEA*** είναι φτιαγμένος σε ***Node.js*** [29], ένα ασύγχρονο ***event-driven runtime*** κατάλληλο για επεκτάσιμες δικτυακές εφαρμογές. Το ***Node*** είναι βασισμένο σε δυο βιβλιοθήκες ανοιχτού κώδικα, στην ***V8 Javascript Engine*** [30] της ***Google***, και στην ***libuv*** [31] μια βιβλιοθήκη ***CPP*** η οποία αναλαμβάνει τις υποκείμενες ενέργειες του συστήματος. Είναι σημαντικό να κατανοήσουμε τι ακριβώς σημαίνει ***non-blocking I/O*** προτού συνεχίσουμε την ανάλυση.

Κατά την εκκίνηση του προγράμματος, ένα ***thread*** κατανέμεται στο ***event loop*** και μέσω αυτού εκτελείται ο κώδικας. Αυτό δεν σημαίνει ότι όλες οι λειτουργίες του προγράμματος στηρίζονται σε αυτό το ένα ***thread***. Για παράδειγμα οι λειτουργίες ***crypto*** χρησιμοποιούν ένα ***thread pool***, αποτελούμενο από συνολικά 4 ***thread***. Η απόφαση του τι θα περάσει στο ***thread pool*** γίνεται στην ***CPP*** μεριά του ***Node***. Βέβαια για μερικές λειτουργίες του λειτουργικού συστήματος, όπως είναι ένα ***HTTP Request***, το ***Node*** και οι υποκείμενες βιβλιοθήκες του δεν είναι ικανές να τις εκτελέσουν οπότε αυτές ανατίθενται στο λειτουργικό σύστημα.

Για την δημιουργία του ***API*** χρησιμοποιήθηκε το ***Express***, ένα ***Node.js framework*** κατάλληλο για εφαρμογές δικτύου και ***API***. Ο ορισμός ενός ***HTTP Route*** είναι απλός και σύντομος.

```
router.get('/api/athletes',
    requireAuth,
    async (req, res) => {
        const athletes = await Athlete.find();
        res.send(athletes);
    });
```

Απόσπασμα κώδικα από /server/routes/athletes.js

Στο παραπάνω απόσπασμα ορίζουμε το ***route /api/athletes*** το οποίο μας επιστρέφει όλους τους αθλητές. Με μόλις 6 γραμμές κώδικα δημιουργήσαμε ένα καινούργιο ***HTTP Route***, που

μπορούμε να χτυπήσουμε στην διαδρομή ***backend:8000/api/athletes***. Παρατηρούμε ότι πριν την κύρια ***function*** έχουμε ένα ***middleware***, το ***requireAuth***. Αυτό το ***middleware*** εξάγεται από ένα άλλο αρχείο και το χρησιμοποιούμε σαν ένα ενδιάμεσο βήμα κώδικα. Δηλαδή θα εκτελείται πάντα πριν την κύρια ***function***, και σε περίπτωση αποτυχίας του δεν θα προχωράει το ***request***.

```
module.exports = {
requireAuth(req, res, next) {
    if (!req.isAuthenticated())
        return res.redirect('/auth/login');
    next();
}
```

Απόσπασμα κώδικα από /server/middlewares/middleware.js

Είναι αναγκαίο όμως να περιορίσουμε τα ***request*** που δεχόμαστε στα ***endpoint***, και έτσι ως μέτρο ασφαλείας χρησιμοποιούμε το πακέτο ***express-rate-limit***. Στο παρακάτω απόσπασμα το πρώτο όρισμα είναι ο χρόνος σε ***ms***, ενώ το δεύτερο είναι τα ***request*** που επιτρέπονται ανά ***IP*** στο χρονικό διάστημα ***windowMs***.

```
app.use(rateLimit({
    windowMs: 15 * 60 * 1000,
    max: 100
}));
```

Απόσπασμα κώδικα από /server/index.js

Για να διατηρήσουμε την συνέδρια του χρήστη χρησιμοποιούμε το ***express-session*** μαζί με το ***cookie-session***. Με αυτά το ***middleware*** όλα τα στοιχεία της συνεδρίας μένουν στον server, ενώ ο χρήστης αναλαμβάνει μόνο το ***Session ID***. Παρακάτω ορίζουμε το κλειδί με το οποίο υπογράφεται και πιστοποιείται η τιμή του ***cookie***, καθώς και την μέγιστη ηλικία ενός ***cookie***.

```
app.use(
    require('cookie-session')({
        keys: [session_secret],
        maxAge: 30 * 24 * 60 * 60 * 1000
    })
);
```

Απόσπασμα κώδικα από /server/index.js

Πλέον που μπορούμε να διατηρήσουμε την συνεδρία του χρήστη πρέπει να προστατεύσουμε μερικές διαδρομές, καθώς δεν πρέπει να είναι διαθέσιμα στους χρήστες χωρίς εξουσιοδότηση. Για παράδειγμα ένας μη εγγεγραμμένος χρήστης δεν πρέπει να έχει πρόσβαση στην λίστα των αθλητών. Θα εγκαταστήσουμε ένα ***middleware*** αυθεντικοποίησης, το ***Passport***, το οποίο θα διαχειρίζεται όλα αυτά μέσω 'στρατηγικών'. Για τις ανάγκες της ***ICHNAEA*** θα χρησιμοποιήσουμε μόνο την τοπική στρατηγική, που μας επιτρέπει να αποθηκεύσουμε και να αυθεντικοποιήσουμε τον χρήστη τοπικά. Υπάρχουν βέβαια πληθώρα στρατηγικών ακόμα, όπως ***OAuth*** η ***Openid***. Για να μπορούμε να έχουμε επίμονες συνεδρίες, πρέπει πρώτα να σειριοποιήσουμε τον χρήστη στο ***session*** και στην συνέχεια να αποσειριοποιούμε σε κάθε ***request***.

```
passport.serializeUser(function (user, done) {
    done(null, user.id);
});

passport.deserializeUser(function (id, done) {
    User.findById(id, function (err, user) {
        done(err, user);
    });
});
```

Απόσπασμα κώδικα από /server/services/passport.js

Μετά ορίζουμε τις 2 στρατηγικές, την ***local_register*** και την ***local_login***. Παρακάτω είναι η ***local_login***.

```
passport.use('local_login', new LocalStrategy({
  usernameField: 'username',
  passwordField: 'password'
  },(username, password, done) => {
    User.findOne({username: username})
    .then(user => {
      if (!user) {
        return done(null, false, {message: 'User does not exist!'});
      } else {
        bcrypt.compare(password, user.password, (err, isMatch) => {
          if (err) throw err;

            if (isMatch) {
              const lastLogin = Date.now()
```

```
              User.updateOne(user._id, {lastLogin: lastLogin})
              return done(null, user);
            } else {
              return done(null, false, {message: 'Wrong password'});
            }
          });
      }
    })
    .catch(err => {
      return done(null, false, {message: err});
     });
}));
```

Απόσπασμα κώδικα από /server/services/passport.js

Στην στρατηγική της σύνδεσης, δεχόμαστε 2 ορίσματα, το όνομα του χρήστη και τον κωδικό. Στην συνέχεια αναζητούμε αυτόν τον χρήστη στην βάση δεδομένων, και ανάλογα την απάντηση επιστρέφουμε είτε τον χρήστη και γίνεται ***serialized***, είτε το αντίστοιχο μήνυμα λάθους. Πρέπει όμως να δεχόμαστε συγκεκριμένα πράγματα για είσοδο, δηλαδή δεν γίνεται ένα email να μην περιέχει το '**@**'. Χρησιμοποιούμε για αυτόν τον λόγο το ***Joi***, ένα ***middleware*** που πραγματοποιεί επικύρωση αντικειμένου. Όπως παρατηρούμε, ένας χρήστης που εγγράφεται πρέπει να έχει ένα όνομα χρήστη και κωδικό τύπου ***string***, και ένα ***email*** που είναι προφανώς τύπου ***email***.

```
const userRegSchema = {
    body: {
        username: Joi.string().required(),
        email: Joi.string().email().required(),
        password: Joi.string().required(),
    }
};
```

Απόσπασμα κώδικα από /server/schemas/joi.js

Έχοντας πλέον θεμελιώσει την βασική λειτουργία της εφαρμογής με την διασύνδεση του χρήστη και το ***API***, ήρθε η ώρα να εξετάσουμε πως θα γίνεται η σύγχρονη μεταφορά δεδομένων από τον ***Client*** (συσκευή αθλητή), στο ***backend***, και τέλος στο ***Frontend***. Υπήρξε αρκετή σκέψη και προβληματισμός, εξετάστηκαν λύσεις όπως είναι το ***MQTT*** και τα ***Native Websockets***, αλλά

τελικά η βέλτιστη λύση για την ***ICHNAEA*** κατέληξε να είναι το ***Socket.io***. Το προαναφερθέν είναι ένα περικάλυμμα του ***Websocket API*** που επιτρέπει αμφίδρομη, ***event-based*** επικοινωνία μεταξύ του ***Client*** και του ***Server***. Η αρχική σύνδεση γίνεται με ***HTTP Long Polling***, και στην συνέχεια αναβαθμίζεται σε ***Websocket,*** αν είναι δυνατόν. Αυτό γίνεται γιατί ενδιάμεσα μπορεί να υπάρχουν ***proxies***, ***firewalls***, ***load balancers*** η οτιδήποτε άλλο που μπορεί να καθυστερήσουν την σύνδεση με ***Websockets*** και συνεπώς προσφέροντας μια κακή εμπειρία στον χρήστη. Στην περίπτωση μας καλούμαστε να το χρησιμοποιήσουμε μαζί με το ***Express***, και για να γίνει αυτό πρέπει να περάσουμε τον ***Express Server*** σαν όρισμα στην δημιουργία ενός ***HTTP Server*** όπως βλέπουμε στο απόσπασμα παρακάτω.

```
const app = require('express')();
const server = require('http').createServer(app);
```

Απόσπασμα κώδικα από /server/index.js

Στο τέλος του αρχείου δεν καλούμε την ***function .listen()*** στο ***Express*** αλλά στον ***HTTP Server*** που μόλις δημιουργήσαμε. Αυτό το βήμα είναι ιδιαίτερα σημαντικό, καθώς αν εφαρμόσουμε την ***.listen()*** στο ***app instance***, τότε το ***Express*** θα δημιουργήσει έναν καινούργιο ***server***!

```
server.listen(PORT, () => console.log(`Server listening on port
${PORT}!`));
```

Απόσπασμα κώδικα από /server/index.js

Αφού έχει γίνει η αρχικοποίηση του ***server***, ξεκινάμε με το να αναγνωρίζουμε κάθε καινούργια σύνδεση. Στο απόσπασμα παρακάτω γίνεται ακριβώς αυτό. Για κάθε καινούργια σύνδεση έχουμε ένα καινούργιο στιγμιότυπο socket που ενθυλακώνει αρκετά χρήσιμα γνωρίσματα.

```
io.on('connection', socket => {
    ...
});
```

Απόσπασμα κώδικα από /server/services/socket.js

Για παράδειγμα από το αντικείμενο ***socket*** μπορούμε να αναγνωρίσουμε τα συμβάντα σαν την αναβάθμιση από ***Long Pooling*** σε ***Websocket***, και να εκτυπώσουμε το ***Socket ID***.

```
socket.conn.on('upgrade', () =>
    console.log(`Client with id: ${socket.id} upgraded to
${socket.conn.transport.name}!`)
);
```

Απόσπασμα κώδικα από /server/services/socket.js

Αυτό είναι ένα αφελές παράδειγμα χωρίς κάποια πραγματική χρησιμότητα πέρα από ***logging***. Στην περίπτωση της εφαρμογής μας θέλουμε να έχουμε πάντα το τελευταίο ***Socket ID*** του ***client*** αποθηκευμένο στην βάση δεδομένων. Να σημειώσω ότι το ***Socket ID*** δημιουργείται εκ νέου σε κάθε σύνδεση, και είναι ένα τυχαίο αναγνωριστικό 20 χαρακτήρων. Άρα σε ένα ***subscribe event*** κάνουμε τα ακόλουθα:

```
try {
    if (subscribe === 'clients') {
        client = await Athlete.findOne({id}).populate('_trainer');

        sub.on('message', async (channel, msg) => {
            if (String(client._trainer._id) === JSON.parse(msg))
                client._trainer = await User.findOne({_id:
client._trainer});
        });

        sub.subscribe('updateSocketID');

        if (client) {
            await Athlete.findOneAndUpdate({id}, {socketID});
            return;
        }

        await saveAthlete(id, socketID);
```

```
    } else if (subscribe === 'dashboard') {
        client = await User.findOne({id});

        pub.publish('updateSocketID', JSON.stringify(client._id));

        await User.findOneAndUpdate({id}, {socketID});
    }
} catch (e) {
    console.log(e);
}
```

Απόσπασμα κώδικα από /server/services/socket.js

Τι γίνεται όμως στην περίπτωση που η σύνδεση του ***Dashboard*** γίνει μετά από τον ***Client,*** είτε γίνει επανασύνδεση του? Πλέον ο ***Client*** θα στέλνει δεδομένα σε ένα παλιό ***Socket ID*** το οποίο δεν χρησιμοποιείται πια. Πρέπει λοιπόν να μπορούμε να ενημερώνουμε δυναμικά το ***Socket ID*** στις συσκευές που είναι τύπου ***'clients'***. Δεν γίνεται όμως να χτυπάμε συνεχώς την βάση δεδομένων κάθε φορά που θέλουμε να στείλουμε δεδομένα, καθώς θα δημιουργήσει υπερβολικό φόρτο όταν μιλάμε για δεκάδες συσκευές. Για να το επιτεύξουμε αυτό χρησιμοποιούμε την δυνατότητα ***publish-subscribe*** του ***Redis***. Κάνουμε ***publish*** ένα ***topic*** κάθε φορά που έχουμε μια καινούργια σύνδεση ***Dashboard***. Μετά στο αντίστοιχο ***Socket Instance*** των ***Client*** κάνουμε ***subscribe*** σε αυτό το ***topic***, και κάθε φορά που έχουμε μήνυμα ελέγχουμε αν το ***id*** του αθλητή ταιριάζει με το ***id*** του προπονητή. Αν αυτό ταιριάζει τότε ανακτούμε τα καινούργια δεδομένα από την βάση.

Ένα ακόμα σημαντικό βήμα που πρέπει να κάνουμε είναι να τυλίξουμε τις επικίνδυνες διαδράσεις με την βάση δεδομένων σε ένα ***try-catch block***. Για παράδειγμα αν ένας αθλητής δεν υπάρχει στην βάση δεδομένων, τότε θα πετάξει ***error***. Η σωστή αντιμετώπιση με το ***try-catch*** θα φροντίσει ότι έχουμε τόσο το κατάλληλο μήνυμα στο ***logging***, όσο και την ομαλή συνέχεια του προγράμματος. Ο ***Server*** υποστηρίζει την σύγχρονη μεταφορά δεδομένων από τον ***Client***, στο ***Backend*** και τέλος στο ***Frontend***. Η λογική αυτή έχει ως εξής:

```
socket.on('data', async data => {
    if (client?._trainer) {
        iWrite(data);
        io.volatile.to(client._trainer.socketID).emit('console', data);
    }
});
```

Απόσπασμα κώδικα από /server/services/socket.js

1. Ακούμε για ένα ***event*** 'data' το οποίο στέλνει μόνο ο ***Client***.
2. Ελέγχουμε αν ο συγκεκριμένος αθλητής έχει προπονητή.
3. Γράφουμε τα δεδομένα στην ***Influx***.
4. Στέλνουμε τα δεδομένα μόνο στον προπονητή του.

Τα παραπάνω αποτελούν τον ***Server*** της ***ICHNAEA***. Δεν έχουν περιληφθεί τα πάντα εδώ καθώς θα ξεφευγε γρηγορά το μήκος της διπλωματικής. Αναλύθηκαν τα κυριά πακέτα που χρησιμοποιούνται από το ***NPM Registry*** καθώς και μερικές αποφάσεις που χρειάστηκε να παρθούν κατά την διάρκεια της ανάπτυξης. Μπορεί κάνεις πολύ ευκολά να εξερευνήσει τον υπόλοιπο κώδικα στο ***repository*** που είναι διαθέσιμος.

4.3 DATABASES

Στην ***ICHNAEA*** χρησιμοποιούνται 3 βάσεις δεδομένων, η καθεμιά για ξεχωριστό λόγο. Παρακάτω αναλύονται οι λόγοι αυτοί, καθώς και το πως χρησιμοποιούνται στο πρότζεκτ αυτό.

4.3.1 MongoDB

Η ***MongoDB*** [32] είναι μια ***NoSQL Document Database***. Οι ***NoSQL*** βάσεις δεδομένων παρέχουν έναν διαφορετικό τρόπο αποθήκευσης και ανάκτησης δεδομένων από τις κλασσικές σχεσιακές βάσεις δεδομένων. Αυτό δεν σημαίνει ότι δεν μπορεί να υπάρχει κάποιου είδους

σχεσιακών δεδομένων, απλά είναι δομημένα διαφορετικά, σε μια κοινή δομή δεδομένων αντί για διαφορετικά ***tables***. Πιο συγκεκριμένα η ***Mongo*** αποθηκεύει δεδομένα σε ***Document*** με ***key-value pairs***, παρόμοια με το ***JSON***. Ο τύπος ενός ***value*** μπορεί να είναι ***string***, ***number***, ***boolean***, ***array***, ή ***object***, πράγμα το οποίο θυμίζει πολύ τα αντικείμενα, και συνεπώς βοηθάει στην αποδοτικότητα του προγραμματιστή [33][34].

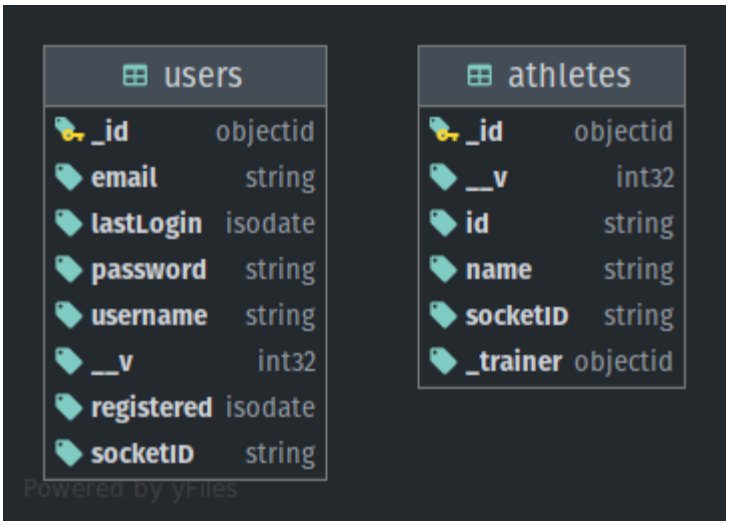
Για την εφαρμογή e-sport χρησιμοποιούμε δυο μοντέλα, ένα για τον χρήστη που θα συνδέεται στο ***dashboard*** και ένα για τον αθλητή. Τα δυο μοντέλα αυτά έχουν μερικά βασικά κλειδιά όπως ***username*** και ***email*** για τον ***User***, και το ***name***, ***_trainer*** για τον ***Athlete***. Παρακάτω είναι τα 2 αυτά μοντέλα.

```
const userSchema = new Schema({
    username: String,
    email: String,
    password: String,
    registered: Date,
    lastLogin: Date,
    socketID: String,
});
```

Απόσπασμα κώδικα από /server/models/User.js

```
const AthleteSchema = new Schema({
    id: String,
    socketID: String,
    name: String,
    _trainer: {type: Schema.Types.ObjectId, ref: 'User'},
});
```

Απόσπασμα κώδικα από /server/models/Athlete.js



Εικόνα 4.3.1 Στιγμιότυπο μοντέλων **MongoDB** *από* ***Jetbrains Webstorm IDE***.

Με το πεδίο ***_trainer*** κρατάμε μια αναφορά στον προπονητή του αθλητή, δηλαδή το ***_id*** του προπονητή. Έτσι μπορούμε πολύ εύκολα να χρησιμοποιήσουμε μεθόδους όπως είναι η ***.populate()*** για να ανακτήσουμε όλα τα στοιχεία του χρήστη.

Για να διατηρήσουμε τον κώδικα καθαρό και να μπορούμε να χρησιμοποιούμε παρόμοια αποσπάσματα σε πολλαπλά σημεία, εξάγουμε μερικά ***actions*** σε δικό τους αρχείο. Για παράδειγμα η αποθήκευση ενός αθλητή είναι ένα συχνό ***action***, οπότε το αρχικοποιούμε στο ξεχωριστό αρχείο ***mongo_actions.js***.

```
saveAthlete = async (id, socketID) => {
    const newAthlete = new Athlete({
        id: id,
        socketID: socketID,
        name: faker.name.findName(),
    })

    try {
        await newAthlete.save();
    } catch (err) {
        console.log(err);
    }
}
```

Απόσπασμα κώδικα από /server/actions/mongo_actions.js

Παρατηρήστε ότι η ανάκτηση δεδομένων χρειάζεται μόλις μερικά χιλιοστά του δευτερολέπτου!

```
[2021-07-23 12:28:43] Connected
ichnaea> use ichnaea
[2021-07-23 12:28:44] completed in 85 ms
ichnaea> db.getCollection("athletes")
           .find({})
           .limit(21)
[2021-07-23 12:28:44] 1 row retrieved starting from 1 in 356 ms (execution:
216 ms, fetching: 140 ms)
```

Απόσπασμα **terminal.**

4.3.2 InfluxDB

Η ***InfluxDB*** [35] είναι μια βάση δεδομένων χρονοσειρών. Τέτοιες βάσεις δεδομένων έχουν δημιουργηθεί ειδικά για χειρισμό δεδομένων τα οποία έχουν χρονική σήμανση. Η ειδοποιός διαφορά των TSDB με τις κλασικές βάσεις δεδομένων, είναι ότι δεν χρειάζονται πολύπλοκες σχέσεις με άλλους πίνακες και δεν απαιτείται η αποθήκευση των δεδομένων για πάντα. Αυτό σημαίνει ότι μπορούν να ρυθμιστούν να διαγράφουν τακτικά δεδομένα που έχουν λήξει. Η ***Influx*** αποτελεί πρωτοπόρο σε τέτοιου τύπου βάσεων δεδομένων και αυτό οφείλεται κυρίως στο γεγονός ότι έχει σχεδιαστεί αποκλειστικά για αυτό. Τα δεδομένα μπορούν να έχουν ακρίβεια ***nanosecond***, πράγμα που τις καθιστά ιδανικές για επιστημονικές η οικονομικές εφαρμογές [36][37].

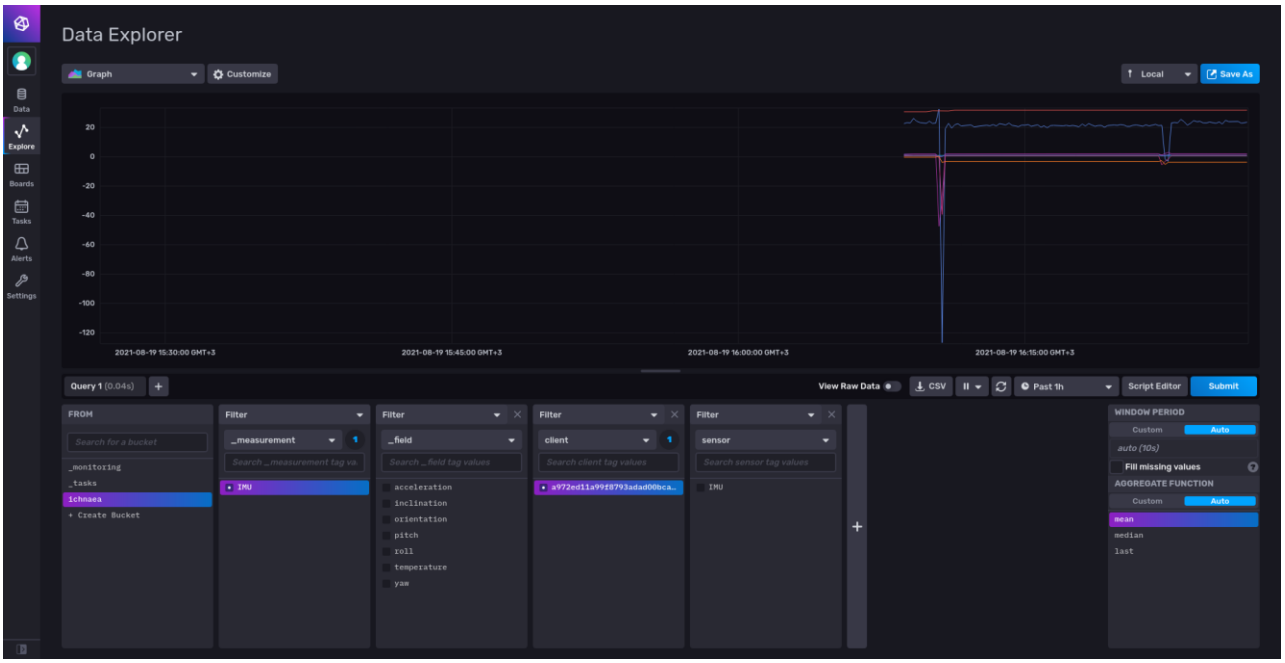
Στην ***ICHNAEA*** χρησιμοποιούμε την ***Influx*** για να αποθηκεύσουμε τις μετρήσεις των αθλητών. Για αυτό είναι υπεύθυνος ο server που αποθηκεύει τις εισερχόμενες μετρήσεις από το ***socket***. Πάλι, όπως και στην ***Mongo***, έχουμε ξεχωρίσει τα ***action*** σε έναν δικό τους φάκελο. Για παράδειγμα στο παρακάτω απόσπασμα κώδικα εκτελούμε ένα ***Query***.

```
iQuery = (query) => {
    return queryApi
        .collectRows(query)
        .then((result) => {
```

```
            return result;
        })
        .catch((err) => {
            return [{Error: `Error occurred: ${err}`}];
        });
}
```

Απόσπασμα κώδικα από /server/actions/influx_actions.js

Η ***Influx*** έχει ένα πλούσιο ***dashboard*** το οποίο έχουμε αφήσει εκτεθειμένο στην πόρτα 8086 μέσω του ***docker-compose***. Η σύνδεση σε αυτό γίνεται με το όνομα χρήστη και το συνθηματικό που έχουμε ορίσει στο αρχείο ***influx-variables.env***. Εκεί τα δεδομένα απεικονίζονται σε πίνακες, γραφήματα, μετρητές, και πολλά ακόμα. Έχει ακόμα, μια πληθώρα από φίλτρα, δυνατότητες ειδοποίησης, καθώς και δυνατότητα για παρακολούθηση σε ζωντανό χρόνο.



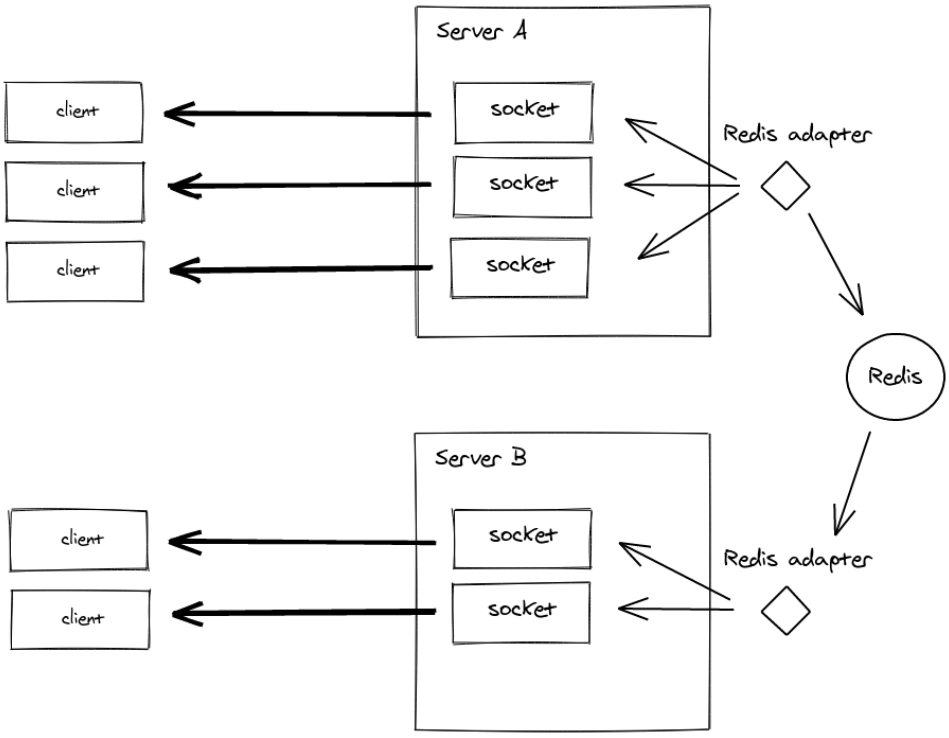
*Εικόνα 4.3.2 Στιγμιότυπο **Data Explorer** από **Influx Dashboard**.*

4.3.3 Redis

Η ***Redis*** [38] είναι ένα λογισμικό ανοιχτού κώδικα που αποθηκεύει δεδομένα στην μνήμη, και χρησιμοποιείται σαν κατανεμημένη βάση δεδομένων, ***cache***, ή και ***message broker*** μεταξύ πολλών άλλων. Από προεπιλογή η ***Redis*** αποθηκεύει στιγμιότυπα του συνόλου δεδομένων στον δίσκο σε ένα διάδικό αρχείο, ανά προκαθορισμένο χρονικό διάστημα. Το στιγμιότυπο αυτό δεν είναι ιδιαίτερα ανθεκτικό, και αυτό σημαίνει ότι στην περίπτωση που το μηχάνημα σβήσει ή η διεργασία πεθάνει, τότε τα δεδομένα θα χαθούν.

Στην ***ICHNAEA*** χρησιμοποιούμε την ***Redis*** σαν μηχανισμό ***Publish-Subscribe.*** Στον μηχανισμό αυτόν ο αποστολέας, δηλαδή ο ***publisher***, στέλνει ένα μήνυμα σε κανάλι. Αυτό το μήνυμα το λαμβάνουν όσοι είναι εγγεγραμμένοι σε αυτό το κανάλι, δηλαδή οι ***subscribers***. Δεν υποστηρίζεται η δυνατότητα να σταλούν κάποιου είδους προσωπικά μηνύματα. Κανένα από τα δυο μέλη δεν έχει επίγνωση για την ύπαρξη του αλλού, και αυτό επιτρέπει μεγαλύτερη ευελιξία στον σχεδιασμό.

Αρχικά το χρησιμοποιούμε για να ανανεώσουμε το ***Socket ID*** όπως προαναφέρθηκε στην ενότητα 4.2, και μετά σαν προσαρμογέα στο ***socket.io***. Αυτό το χρησιμοποιούμε για την περίπτωση που έχουμε πολλαπλούς ***server*** όταν κάνουμε ***scale-up***, και χρειαστεί να κάνουμε ***broadcast*** σε πολλαπλούς ***client*** που βρίσκονται σε διαφορετικούς ***server.***



Εικόνα 4.3.3 Σχεδιάγραμμα **Redis Adapter** *από* **socket.io documentation**.

4.4 WEB

Το ***Dashboard*** της ***ICHNAEA*** είναι ένα ***Vue.js SPA*** [39] φτιαγμένο με ***Typescript***. Το ***Vue.js*** είναι μια βιβλιοθήκη ανοιχτού κώδικα κατάλληλη για την δημιουργία ***UI*** και ***SPA***. Αποτελεί ένα από τα τρία μεγάλα ***frontend framework***, και έχει μια επεκτάσιμη βιβλιοθήκη με προηγμένες δυνατότητες όπως είναι το ***routing*** και το ***state management***. Τα τελευταία δυο προσφέρονται σαν ξεχωριστά προτζεκτ, αλλά συνεχίζουν να διατηρούνται από την επίσημη κοινότητα. Η απόφαση να γραφτεί σε ***Typescript*** έγινε συνειδητά, καθώς παραμένει η βασική λειτουργικότητα της ***Javascript***, απλά προστίθενται ***Types*** και έχουμε πλέον μια πιο ρητά ορισμένη γλώσσα με ***type-safety***. Η συμβατότητα της ***Vue*** και των παράγωγων της είναι σχεδόν

καθολική με την ***Typescript***. Χρησιμοποιήσαμε το ***NPM*** πακέτο της ***Vue***, και έτσι μπορούμε να εκμεταλλευτούμε όλο το οικοσύστημα του ***Node***.

Ένα από τα πρώτα πράγματα που θα χρησιμοποιήσει κανείς στο ***Vue*** είναι το ***Declarative Rendering***. Αυτό επιτρέπει στα δεδομένα να είναι ***reactive*** και να αλλάζουν τα στοιχεία στο ***DOM*** χωρίς να φορτώνει όλη η σελίδα από την αρχή.

```
<p class="is-size-4 is-bold">&nbspWhats up, {{ user.username }}</p>
<p class="is-size-6 is-bold">Last login on {{ date }}</p>
```

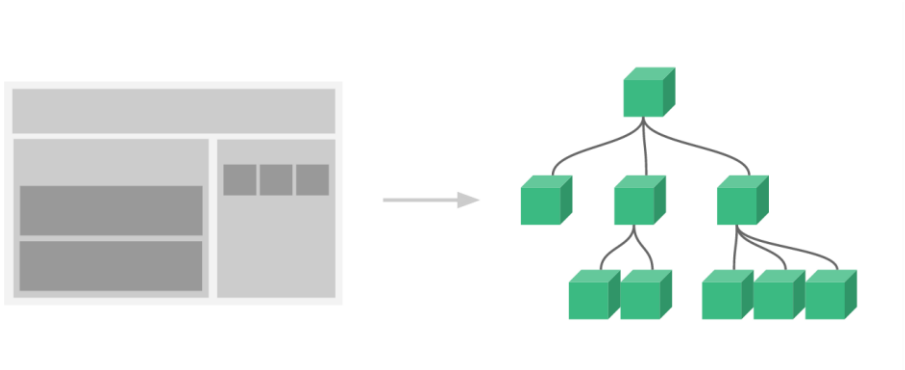
Απόσπασμα κώδικα από /web/src/views/DashboardHome.vue

Στο παραπάνω απόσπασμα βλέπουμε ένα τέτοιο παράδειγμα από την εφαρμογή μας. Τα δυο αυτά πεδία τα παίρνουμε από το ***store*** κατά την δημιουργία της σελίδας. Μπορούμε επίσης να εισάγουμε λογική, όπως είναι ένα ***if statement*** η ένα ***loop***.

```
<article class="tile is-child box">
        <p class="title">System status</p>
        <p v-if="this.$store.getters.server_status" class="subtitle">
                <i class="fas fa-check-circle"></i> All systems operational
        </p>
        <p v-else class="subtitle">
                <i class="fas fa-times-circle"></i> Something does not seem
right
        </p>
</article>
```

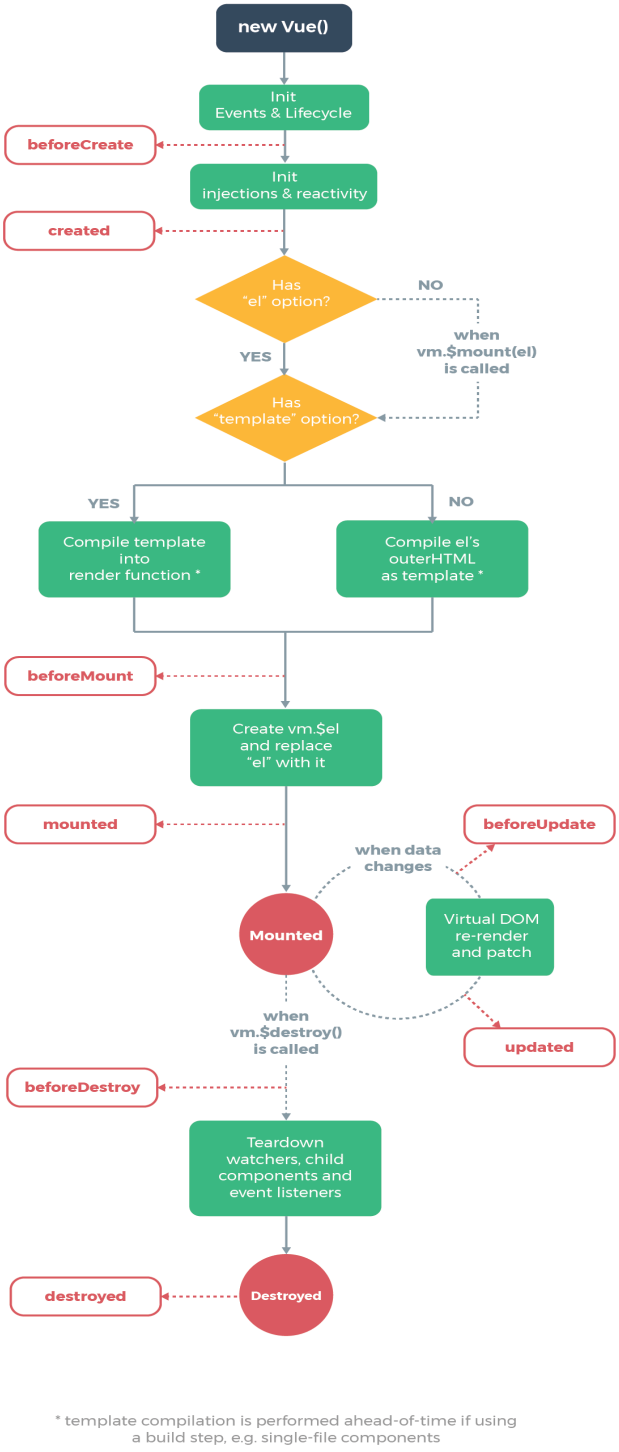
Απόσπασμα κώδικα από /web/src/views/DashboardHome.vue

Μια ακόμα σημαντική έννοια που πρέπει να καταλάβουμε είναι το ***Component Composition***. Με αυτόν τον τρόπο μπορούμε να χωρίσουμε το ***frontend*** σε μικρότερα, αυτόνομα, και επαναχρησιμοποιήσιμα ***component***.



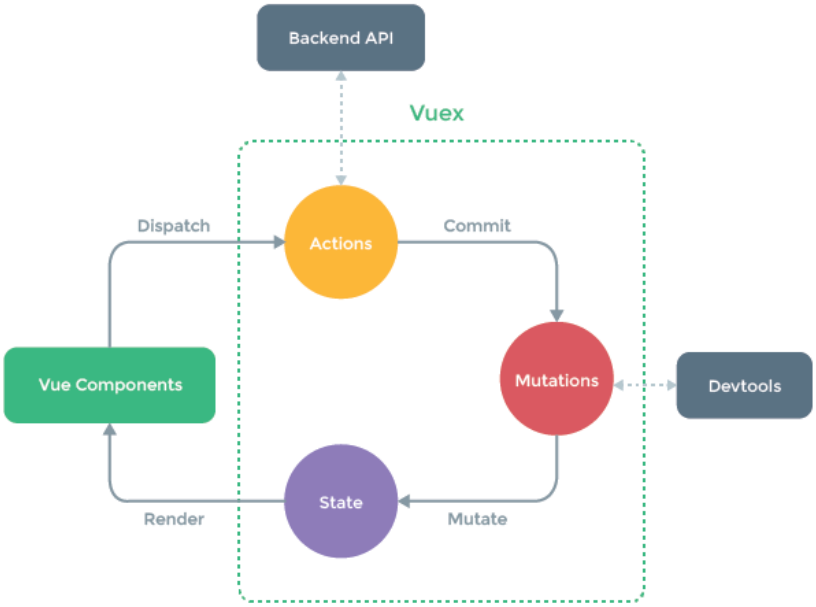
Εικόνα 4.4.1 Σχεδιάγραμμα **Component Composition** *από* ***Vue.js documentation***.

Ένα τέτοιο παράδειγμα στην ***ICHNAEA*** είναι το ***Navbar***, οπού έχει μεταφερθεί σε ένα δικό του αυτόνομο αρχείο. Μια ακόμα σημαντική έννοια είναι το ***Lifecycle Hooks***. Κάθε ***component*** περνάει από μια σειρά από βήματα αρχικοποίησης από το ***template compilation*** μέχρι την ανανέωση του ***Virtual DOM*** με καινούργια δεδομένα. Είναι ιδιαίτερα χρήσιμο καθώς επιτρέπει στον χρήστη να προσθέσει κώδικα σε διαφορετικά στάδια της εφαρμογή. Στην εφαρμογή μας το χρησιμοποιούμε εκτεταμένα, όπως για παράδειγμα να αναγνωρίσουμε πότε μια σελίδα είναι στο στάδιο ***created*** ώστε να κάνουμε τις κατάλληλες κλήσεις στο ***API*** και στην συνέχεια να απεικονίσουμε τα δεδομένα αυτά.



Εικόνα 4.4.2 Σχεδιάγραμμα **Lifecycle** *από* ***Vue.js documentation.***

Όσο μεγαλώνουν και γίνονται πιο πολύπλοκες οι δικτυακές εφαρμογές, τόσο πιο δύσκολο είναι να κρατήσει κάνεις μια μοναδική πηγή αλήθειας για τα δεδομένα του. Για αυτόν τον λόγο χρησιμοποιούμε το ***Vuex*** [40], ένα πρότυπο διαχείρισης κατάστασης το οποίο υποστηρίζεται από την επίσημη κοινότητα. Έτσι σημαντικά και επαναχρησιμοποιούμενα δεδομένα βρίσκονται σε ένα κεντρικό σημείο, οπού όλα τα ***Views*** έχουν πρόσβαση.



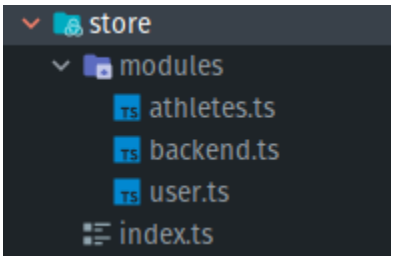
Εικόνα 4.4.3 Σχεδιάγραμμα **Lifecycle** *από* ***Vuex documentation.***

Ένα τέτοιο παράδειγμα στην ***ICHNAEA*** είναι όταν ο χρήστης προσπαθεί να συνδεθεί στην εφαρμογή. Πρώτα καλούμε την δράση ***user_login*** για να χτυπήσουμε το ***API*** στο κατάλληλο ***route***. Αν ο χρήστης υπάρχει και η απάντηση από τον ***server*** είναι επιτυχής τότε κάνουμε αλλάζουμε την κατάσταση του χρήστη αποθηκεύοντας τις πληροφορίες που γυρίσαν τοπικά. Στο παρακάτω απόσπασμα φαίνεται το ***action***, ***mutation***, ***state*** που καλούνται όταν ο χρήστης προσπαθεί να συνδεθεί.

```
// state
const state = () => ({
    user: <UserInterface>{},
    userStatus: false,
    err: <Error>{},
});
...
// mutation
auth_success(state: userState, user: UserInterface) {
    state.user = user;
    delete state.user['password'];
    state.userStatus = true;
}
...
// action
user_login({commit}, user: UserInterface) {
        return new Promise((resolve, reject) => {
            axios({
                method: 'POST',
                url: '/auth/login',
                data: qs.stringify({...user})
            })
                .then((resp: AxiosResponse) => {
                    commit('auth_success', resp.data.user);
                    resolve(resp);
                })
                .catch((err: Error) => {
                    commit('auth_error', err);
                    reject(err);
                });
        });
}
...
```

Απόσπασμα κώδικα από /web/ src/store/modules/user.ts

Έχουμε χωρίσει επίσης το ***Vuex Store*** σε μικρά, λογικά, και αυτόνομα ***component***, όπου το καθένα περιέχει διαφορετικές ενέργειες. Το ***store*** είναι και αυτό γραμμένο σε ***Typescript***.



Εικόνα 4.4.4 Στιγμιοτυπο **File Tree** *από* ***Jetbrains Webstorm IDE***

Όπως προαναφέρθηκε η ***ICHNAEA*** είναι ένα ***Single Page Application***. Μια ***SPA*** γράφει δυναμικά δεδομένα στην σελίδα από τον ***server***, αντί να φορτώνει εξολοκλήρου από την αρχή όπως θα έκανε μια παραδοσιακή σελίδα. Έτσι η δρομολόγηση μεταξύ των σελίδων δε γίνεται πλέον στον ***server***, αλλά στην μεριά του ***client***. Αυτό ονομάζεται ***client-side routing*** και απαιτεί μια ακόμα βιβλιοθήκη, το ***Vue-router*** [41]. Μπορούμε να ορίσουμε ένα απλού επιπέδου ***route*** με τον παρακάτω τρόπο.

```
const routes: Array<RouteRecordRaw> = [
     ...
        {
            path: '/login',
            name: 'Login',
            component: Login
        },
     ...
    ]
```

Απόσπασμα κώδικα από /web/ src/router/index.ts

Στην συνέχεια μπορούμε να προηγηθούμε στην συγκεκριμένη διαδρομή με δύο τρόπους. Είτε με ένα ***HTML element***, το ***router-link*** το οποίο είναι παρόμοιο με το ***<a>*** αλλά για αποκλειστική χρήση με το ***Vue-router***, είτε από κάποια μέθοδο με την συντόμευση ***this.$router.***

```
user_login() {
  this.$store.dispatch('user_login', {username: this.username, password:
this.password})
      .then(() => this.$router.push({name: 'DashboardHome', params:
{username: this.username}}))
      .catch(() => {
        this.msg =
this.$store.getters.user_err.response.data.errors.message || 'Something
went wrong!'
      });
}
```

Απόσπασμα κώδικα από web/src/views/Login.vue

Στο παραπάνω απόσπασμα βλέπουμε πως χρησιμοποιείται ο δεύτερος τρόπος για την εσωτερική ανακατεύθυνση. Παρομοίως από κάτω είναι το ***HTML element***.

```
<router-link :to="{ name: 'Model', params: { id: athlete._id }}">
```

```
  <span class="icon is-medium has-background-primary has-text-white mr-1">
    <i class="fa fa-lg fa-male"></i>
  </span>
</router-link>
```

Απόσπασμα κώδικα από web/src/views/Athletes.vue

Το οικοσύστημα της ***Vue*** είναι απίστευτα πλούσιο με πολλές επιλογές παραμετροποίησης. Για αυτόν τον λόγο εξετάστηκαν μόνο τα σημαντικά σημεία των τριών μεγάλων βιβλιοθηκών που χρησιμοποιούμε. Είναι ένα συνεχώς αναπτυσσόμενο οικοσύστημα, οπότε είναι καλύτερο κανείς να συμβουλευτεί την τελευταία έκδοση των εγγράφων της ***Vue***.

Έχοντας πλέον αναλύσει τα βασικά στοιχεία της δικτυακής εφαρμογής, ήρθε η ώρα να εξετάσουμε το πως θα χειριζόμαστε τα δεδομένα ενός αθλητή. Από την στιγμή που χρησιμοποιούμε το ***Component Composition*** και η εφαρμογή μας είναι χωρισμένη σε μικρότερες λογικές μονάδες, δεν χρειάζεται να δεχόμαστε δεδομένα των αθλητών σε όλες τις σελίδες. Αντί αυτού δεχόμαστε τα δεδομένα θέσης του αθλητή στην δικιά του σελίδα, δηλαδή στα ***/athletes/:id/table***, ***athletes/:id/char***t, ***athletes/:id/model,*** όπου ***:id*** το μοναδικό αναγνωριστικό του κάθε αθλητή. Με αυτόν τον τρόπο αποφεύγουμε να υπερφορτώσουμε το ***frontend*** με αχρείαστα δεδομένα, φροντίζοντας ότι ο προπονητής παίρνει κάθε φορά μόνο τα δεδομένα του αθλητή που κοιτάει. Στο παρακάτω απόσπασμα κώδικα από την σελίδα ***Model*** ο χρήστης εγγράφετε στο κανάλι ***console*** του ***Socket.io*** όταν βρισκόμαστε στο στάδιο ***mounted()***, και απογράφετε όταν βρισκόμαστε στο στάδιο ***beforeUnmount()*** για να μην μείνει κάποια σύνδεση ανοιχτή όταν ο χρήστης απομακρυνθεί από την σελίδα.

```
mounted() {
    this.$store.dispatch('athlete_getOne',
this.$route.params.id).then((res) => this.athlete = res.data);
    this.$socket.client.on('console', this.parseData);
    this.initScene();
  },
 beforeUnmount() {
    this.$socket.client.off('console', this.parseData);
```

```
  }
```

Απόσπασμα κώδικα από /web/src/views/Model.vue

Τα εισερχόμενα δεδομένα δεν αποθηκεύονται κάπου τοπικά, αλλά ανανεώνουν δυναμικά το μοντέλο απεικόνισης του αθλητή. Κάθε φορά που δεχόμαστε δεδομένα από τον server χρησιμοποιούμε την μέθοδο ***parseData()*** για να μετατρέψουμε τις μετρήσεις ***pitch***, ***roll***, και ***yaw*** σε ακτίνια από μοίρες, και στην συνέχεια σε τετραδόνια με την μέθοδο που διατίθεται από τη βιβλιοθήκη ***Three.js*** [42] την οποία χρησιμοποιούμε για τις ***3D*** απεικονίσεις.

```
parseData(data) {
  const {temperature, pitch, roll, yaw} = data;
  this.temperature = temperature;
  // Euler angles in three.js are in rads
  this.pitch = pitch * this.toRads;
  this.roll = roll * this.toRads;
  this.yaw = yaw * this.toRads;
  if (this.loadedModel) {
    this.head.quaternion.setFromEuler(new Euler(this.roll, this.pitch,
this.yaw));
  }
}
```

Απόσπασμα κώδικα από /web/src/views/Model.vue

Ο λόγος που υπολογίζουμε σε γωνίες ***Euler*** και μετατρέπουμε σε ***Quaternion*** στο ***frontend***, είναι γιατί θέλουμε να έχουμε αποθηκευμένη μια μέτρηση που είναι πιο εύκολο να απεικονιστεί με νόημα σε ένα γράφημα. Για παράδειγμα μπορούμε να χρησιμοποιήσουμε τα γραφήματα που προσφέρονται στο ***dashboard*** της ***influx*** για να αναπαραστήσουμε τα ***pitch***, ***roll***, και ***yaw***, τρεις μονάδες μέτρησης που είναι πιο εύκολο να κατανοήσει κανείς. Η μετατροπή σε ***Quaternion*** γίνεται για να αποφύγουμε το ***Gimbal Lock*** που προκύπτει με την χρήση των γωνιών ***Euler***. Η μετατροπή είναι αρκετά απλή όπως παρατηρούμε και στην βιβλιοθήκη ***Three.js***. Το απόσπασμα παρακάτω είναι ένα μέρος της μεθόδου ***.setFromEuler()***.

```
const cos = Math.cos;
const sin = Math.sin;

const c1 = cos( x / 2 );
```

```
const c2 = cos( y / 2 );
const c3 = cos( z / 2 );

const s1 = sin( x / 2 );
const s2 = sin( y / 2 );
const s3 = sin( z / 2 );

this._x = s1 * c2 * c3 + c1 * s2 * s3;
this._y = c1 * s2 * c3 - s1 * c2 * s3;
this._z = c1 * c2 * s3 + s1 * s2 * c3;
this._w = c1 * c2 * c3 - s1 * s2 * s3;
```

Απόσπασμα κώδικα από web/node_modules/three/src/math/Quaternion.js

Για την αναπαράσταση χρησιμοποιούμε μια βασική σκηνή με ένα ***FBX*** μοντέλο τοποθετημένο στην μέση. Με την αρχικοποίηση του μοντέλου κρατάμε αναφορές στα κομμάτια του σώματος που είναι απαραίτητα για την αναπαράσταση ώστε αργότερα να αλλάξουμε τις συντεταγμένες τους, όπως στο απόσπασμα παρακάτω που τρέχει όταν το μοντέλο έχει πλέον φορτωθεί στην σκηνή.

```
onLoad(object) {
  // Add the skeleton to the scene
  const skeleton = new SkeletonHelper(object);
  skeleton.visible = true;
  this.$refs.scene.add(skeleton);

  // Get all the necessary body parts for the animation
  this.head = object.getObjectByName('mixamorig1Head');
  this.leftArm = object.getObjectByName('mixamorig1LeftArm');
  this.rightArm = object.getObjectByName('mixamorig1RightArm');
  this.leftForeArm = object.getObjectByName('mixamorig1LeftForeArm');
  this.rightForeArm = object.getObjectByName('mixamorig1RightForeArm');
  this.leftLeg = object.getObjectByName('mixamorig1RightLeg');
  this.rightLeg = object.getObjectByName('mixamorig1LeftLeg');
  this.leftUpLeg = object.getObjectByName('mixamorig1RightUpLeg');
  this.rightUpLeg = object.getObjectByName('mixamorig1LeftUpLeg');

  // Add shadows
  object.traverse(child => {
    if (child.isMesh) {
      child.castShadow = true;
      child.receiveShadow = true;
    }
  });
```

```
  this.loadedModel = true;
}
```

Απόσπασμα κώδικα από /web/src/views/Model.vue

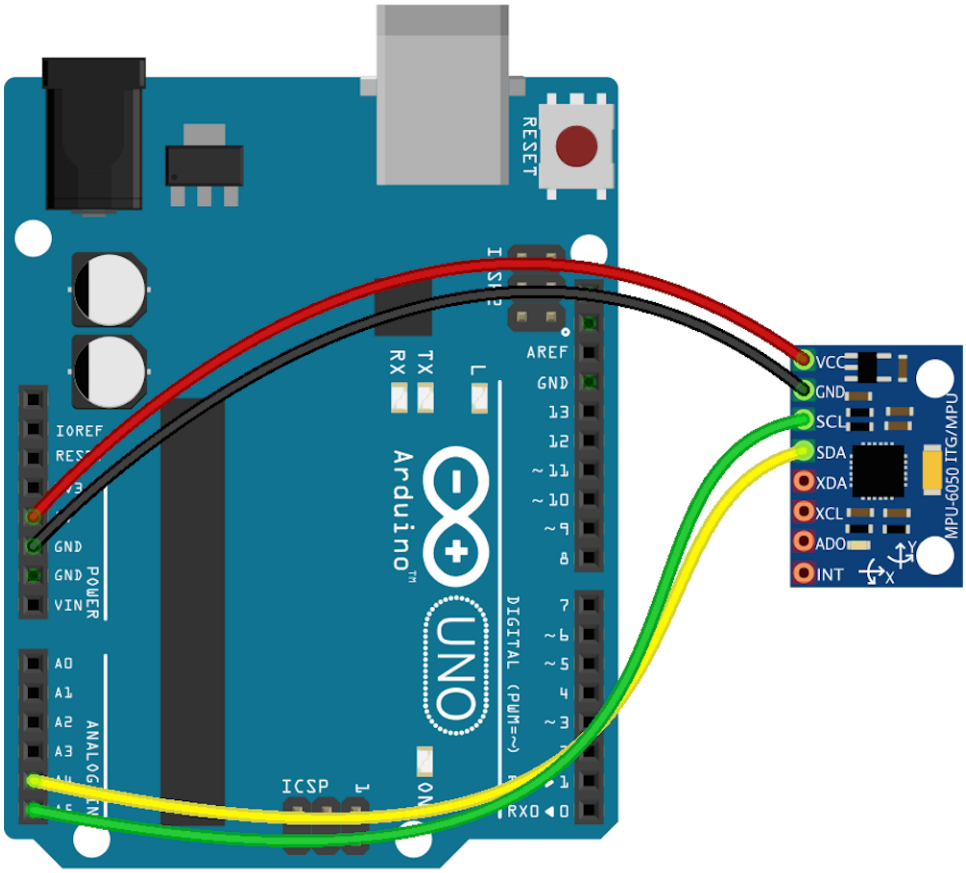
Αυτά είναι τα βασικά σημεία της ***web*** εφαρμογής. Υπάρχουν ακόμα μερικά ***views*** που δεν έχουν αναλυθεί και θεωρείται περιττό. Η μεθοδολογία και λογική που ακολουθήθηκε είναι παρόμοια με τα προαναφερθέν, οπότε δεν υπάρχει λόγος να επαναληφθούν. Ο λόγος που προστέθηκαν, για παράδειγμα το ***Profile.vue*** όπου ο χρήστης έχει δυνατότητα αλλαγής των στοιχείων του**,** είναι για να γίνει η εφαρμογή όσο πιο ρεαλιστική και κοντά σε μια υπηρεσία παραγωγής γίνεται.

4.5 CLIENT

Για να ανιχνεύσουμε και να απεικονίσουμε την κίνηση ενός αθλητή χρησιμοποιούμε έναν αισθητήρα ***IMU*** συνδεδεμένο σε ένα ***Arduino Microcontroller*** [43]. Στην συνέχεια τα δεδομένα συλλέγονται από την εφαρμογή ***client*** και κατόπιν αποστέλνονται στον server. Έχουμε αναλύσει εκτενώς τι γίνεται στην μεριά του ***server***, οπότε σε αυτή την ενότητα θα επικεντρωθούμε αποκλειστικά στην υλοποίηση της ***client*** εφαρμογής.

Η εφαρμογή του ***client*** είναι και αυτή φτιαγμένη με ***Node.js***. Για την αμφίδρομη και σύγχρονη επικοινωνία με τον ***server*** χρησιμοποιούμε το ***npm*** πακέτο ***socket.io-client***, ειδικά σχεδιασμένο για ***client*** εφαρμογές. Ακόμα για την συνδεσιμότητα και την συλλογή δεδομένων χρησιμοποιούμε την βιβλιοθήκη ***Johnny-five*** [44], κατάλληλη για ***Robotics*** και ***IoT*** σε ***Javascript***. Η εφαρμογή είναι ***containerized*** σε ***Docker*** (περισσότερα στην επόμενη ενότητα) οπότε μπορεί να τρέξει είτε σε υπολογιστή με μια από τις κλασσικές αρχιτεκτονικές επεξεργαστή, είτε σε ***Raspberry*** [45] καθώς το ***image*** του ***Node.js*** υποστηρίζει και ***ARM*** αρχιτεκτονικές.

Το πρώτο βήμα είναι η σύνδεση του μικροελεγκτή με τον αισθητήρα ***IMU***. Αυτό είναι αρκετά απλό και απεικονίζεται στο επόμενο διάγραμμα.



Εικόνα 4.5.1 Σχεδιάγραμμα σύνδεσης **IMU** *με* **Arduino** *από* **Johnny-five.**

Συνεχίζουμε λοιπόν με την σύνδεση και συλλογή δεδομένων της ***host*** συσκευής με το ***Arduino***. Πρώτα συνδέουμε το ***Arduino*** σε μια ***USB*** θύρα, πριν ακόμα εκκινήσουμε το ***client*** πρόγραμμα. Τα ***Arduino*** έρχονται με το λογισμικό ***StandardFirmata*** προ εγκατεστημένο, αυτό που χρειάζεται και η βιβλιοθήκη ***Johnny-five*** για την επικοινωνία με την πλακέτα. Σε κάθε άλλη περίπτωση είναι πολύ απλό κανείς να φορτώσει το λογισμικό αυτό, με το να κατεβάσει το ***Arduino IDE*** και να το επιλέξει από την λίστα παραδειγμάτων των ***Firmata***.

```
try {
    board = new Board({port: '/dev/ttyACM0', repl: false, debug: false});
```

```
} catch (e) {
    console.log('Arduino is probably not connected! Please connect device
and restart program...');
    process.exit(1);
}
```

Απόσπασμα κώδικα από /client/services/socket.js

Πρώτα αρχικοποιούμε ένα στιγμιότυπο της κλάσης ***board*** οπού αντιπροσωπεύει την φυσική μας πλακέτα, με ορίσματα την πόρτα, την ρητή απενεργοποίηση του ***REPL*** (***Read-eval-print loop***) και της εξόδου αποσφαλμάτωσης. Κανένα από τα παραπάνω ορίσματα δεν είναι αναγκαία, και η παράληψη τους δεν θα προκαλέσει κάποιου είδους ανώμαλης συμπεριφοράς. Στην συνέχεια περιμένουμε για ένα ***ready event*** από την πλακέτα όπως στο απόσπασμα παρακάτω.

```
board.on('ready', () => {
    console.log('Board is ready!');

    const imu = new IMU({
        controller: 'MPU6050',
        freq: 100
    });

    imu.on('change', () => socket.volatile.emit('data', parseData(imu)));
});
```

Απόσπασμα κώδικα από /client/services/socket.js

Όταν πλέον έχουμε συνδεθεί στον μικροελεγκτή τότε μπορούμε να αρχικοποιήσουμε ένα καινούργιο στιγμιότυπο της κλάσης ***IMU***, με ορίσματα τον τύπο του ***IMU*** και την συχνότητα δειγματοληψίας σε ***HZ***. Τώρα πλέον λαμβάνουμε μετρήσεις ανά 0.1 δευτερόλεπτα, και πρώτού τα στείλουμε στον server τα διαμορφώνουμε κατάλληλα για να αποκτήσουν κάποιου είδους νόημα.

```
function parseData(imu) {
    const {temperature, accelerometer, gyro} = imu;

    // Get pitch, roll, yaw from gyro
    pitch += (gyro.rate.x / gyroSens) * samplingInterval;
    roll -= (gyro.rate.y / gyroSens) * samplingInterval;
    yaw += (gyro.rate.z / gyroSens) * samplingInterval;
```

```
    // Only use accelerometer when forces are ~1g
    if (accelerometer.acceleration > -1 && accelerometer.acceleration < 2)
{
        pitch =
            0.98 * pitch +
            0.02 * accelerometer.pitch;

        roll =
            0.98 * roll +
            0.02 * accelerometer.roll;
    }

    // Filter out noise (a small tremor appears with too many fraction
digits)
    pitch = toFixed(pitch);
    roll = toFixed(roll);
    yaw = toFixed(yaw);

    return {
        pointName: 'IMU',
        uuid: id,
        temperature: temperature.celsius,
        pitch,
        roll,
        yaw,
        acceleration: imu.accelerometer.acceleration,
        inclination: imu.accelerometer.inclination,
        orientation: imu.accelerometer.orientation,
    }
}
```

Απόσπασμα κώδικα από /client/actions/imu_actions.js

Η βιβλιοθήκη ***Johnny-five*** δεν μας παρέχει μόνο με τα ωμά δεδομένα των αισθητήρων, αλλά κάνει και μερικούς υπολογισμούς όπως είναι η επιτάχυνση ή η κλίση. Έχουμε στην διάθεση μας επίσης και τις 3 γωνίες ***Euler***, όπως υπολογίζονται ξεχωριστά από κάθε αισθητήρα. Υπάρχει όμως ένα πρόβλημα με αυτές τις μετρήσεις. Τα γυροσκόπια αν και παρέχουν αρκετά ακριβείς μετρήσεις, τείνουν να έχουν απόκλιση με την πάροδο του χρόνου ακόμα και όταν είναι ακίνητα. Άρα τα γυροσκόπια είναι χρήσιμα για να υπολογισμούς σε σύντομα χρονικά διαστήματα, και λόγω της απόκλισης είναι αναξιόπιστα στο πέρασμα του χρόνου. Από την άλλη τα επιταχυνσιόμετρα

δεν έχουν κάποια απόκλιση στην πάροδο του χρόνου, αλλά οι μετρήσεις έχουν πολύ περισσότερο θόρυβο, καθιστώντας τα ακατάλληλα για βραχυπρόθεσμες μετρήσεις.

Συμπεραίνοντας λοιπόν ότι τα γυροσκόπια είναι πιο ακριβές βραχυπροθέσμα και ότι τα επιταχυνσιόμετρα είναι πιο ακριβές μακροπρόθεσμα, μπορουμε να σχεδιάσουμε ένα φίλτρο χαμηλής διέλευσης για το επιταχυνσιόμετρο και υψηλής διέλευσης για το γυροσκόπιο. Αυτό είναι το λεγόμενο συμπληρωματικό φίλτρο, δηλαδή μια γραμμική παρεμβολή μεταξύ του προσανατολισμού που εκτιμάται από το γυροσκόπιο και αυτό που υπολογίζεται από το επιταχυνσιόμετρο. Αυτό μπορεί να εκφραστεί σε μια απλή του μορφή ως εξής:

```
angle = a * (angle + gyroscopeData * dt) + 1-a * accelerometerData
```

Το ποσοστό του ***a*** συνήθως κυμαίνεται από 0.95 μέχρι 0.98 αλλά εξαρτάται από την εφαρμογή του ***IMU***. Με αυτόν τον τρόπο παίρνουμε τα καλυτέρα χαρακτηριστικά από τους 2 αισθητήρες, χωρίς τον θόρυβο η την απόκλιση. Όπως φαίνεται στο παραπάνω απόσπασμα, χρησιμοποιούμε ποσοστά 98%-2% για το γυροσκόπιο και το επιταχυνσιόμετρο αντίστοιχα. Λαμβάνουμε υπόψιν τις μετρήσεις του επιταχυνσιόμετρου μόνο όταν βρισκόμαστε σε μια σχετικά σταθερή κατάσταση, δηλαδή η επιτάχυνση να είναι κοντά στο 1G.

Υπάρχουν μερικά ακόμα φίλτρα που μπορούν να εφαρμοστούν για να συνδυάσουν τις μετρήσεις από γυροσκόπια, επιταχυνσιόμετρα, και μαγνητόμετρα, όπως είναι τα φίλτρα ***Kalman*** και ***Madgwick***. Προτιμήθηκε το συμπληρωματικό φίλτρο λόγω της απλότητας του και της χαμηλής υπολογιστικής ισχύς που χρειάζεται, καθιστώντας το ιδανικό για ενσωματωμένα συστήματα.

4.6 DEPLOYMENT

Φτάνοντας πλέον στο κρίσιμο σημείο της υλοποίησης, υπήρξε αρκετή σκέψη για το πως θα τρέχει η εφαρμογή. Οι κλασικές ***Node.js*** εφαρμογές τρέχουν με την εντολή ***npm run <script>***, οπού στην θέση του ***<script>*** μπαίνει το επιθυμητό ***script*** από το ***package.json***. Αν και αυτή η λύση είναι αρκετή για μικρές και απλές εφαρμογές, είναι μακριά από το ιδανικό όσο γίνονται πιο πολύπλοκες οι εφαρμογές και οι μεταξύ τους σχέσεις. Ένα εργαλείο που μπορεί να καταργήσει επαναλαμβανόμενες διαδικασίες διαμόρφωσης είναι το ***Docker*** [46]. Το ***Docker*** αποτελεί ένα λογισμικό ανοιχτού κώδικα για την αυτοματοποίηση των εφαρμογών ως φορητά, αυτόνομα ***container*** τα οποία εκτελούνται στο νέφος η στις τοπικές εγκαταστάσεις. Είναι μια άμεση αντικατάσταση του μέχρι τώρα παραδοσιακού τρόπου ***deployment*** με ***VM***, και έρχονται με μερικές αλλαγές.

Το ***Docker*** δεν έχει την ανάγκη για ***hypervisor*** η ξεχωριστά λειτουργικά συστήματα μέσα σε κάθε ***VM***, πάρα μόνο ένα λειτουργικό σύστημα πάνω στο οποίο τρέχει το ίδιο το ***Docker***. Αυτός είναι ο βασικός λόγος που προσφέρονται τόσο υψηλές ταχύτητες εκκίνησης των υπηρεσιών, με το μειονέκτημα βέβαια της απομόνωσης που θα πρόσφερε ένα ***VM***. Ένα ακόμα σημαντικό πλεονέκτημα για τους προγραμματιστές είναι ότι το περιβάλλον παραμένει το ίδιο σε όλα τα μηχανήματα, ενώ είναι εύκολο και γρήγορο να τρέξεις μια εφαρμογή με έναν αξιόπιστο και αναπαράξιμο τρόπο. Όλες οι ρυθμίσεις μπορούν να γράφουν σε ένα ***yaml*** αρχείο (στην περίπτωση που χρησιμοποιηθεί το ***docker-compose***) το οποίο έρχεται με τα πλεονεκτήματα του ***IaC*** (***Infrastructure as Code)***. Έτσι όλα τα ***pipeline*** επιταχύνονται χωρίς λάθη, από το τοπικό περιβάλλον μέχρι την παραγωγή.

Για αυτούς του λόγους στην Ιχναίη, τόσο κατά την διάρκεια της ανάπτυξης, όσο και στην τελική της μορφή, χρησιμοποιήθηκε το ***Docker*** σε συνδυασμό με το ***docker-compose***.

Αναλυτικότερα μέσα σε κάθε υποφάκελο στο ***repository*** υπάρχει ένα ***dockerfile***. Το ***dockerfile*** είναι ένα αρχείο με οδηγίες του πως να φτιάξει το **docker** ένα ***image***. Υπάρχουν αρκετά ορίσματα που μπορούν να χρησιμοποιηθούν στο αρχείο αυτό, τα οποία είναι εκτός του σκοπού της διπλωματικής αυτής και είναι καλύτερο κάνεις να συμβουλευτεί το ***documentation*** του ιδίου του ***docker***.

```
FROM node:14-alpine3.10
WORKDIR /usr/src/app/server
COPY package*.json ./
RUN npm install
EXPOSE 8000
CMD [ "npm", "run" ]
```

Απόσπασμα κώδικα από /server/Dockerfile

Αναλυτικότερα, κάθε αρχείο πρέπει να ξεκινάει από την εντολή ***FROM***, όπου ορίζει το βασικό ***image*** το οποίο από προεπιλογή την τραβάει από το ***docker hub***. Ορίζουμε στην συνέχεια τον κατάλογο όπου οι επόμενες εντολές θα εκτελεστούν. Αν δεν οριστεί ρητά τότε θα οριστεί αυτόματα από το ***Docker***. Η επόμενη εντολή αντιγράφει ένα η παραπάνω αρχεία στο ***container***, στην περίπτωση της ***ICHNAEA*** που χρησιμοποιούμε το ***Node.js***, αντιγράφουμε τα αρχεία ***package.json*** και ***package-lock.json***, απαραίτητα για να εγκαταστήσουμε τα ***node_modules***. Αφού έχουμε αντιγράψει τα απαραίτητα αρχεία από το προηγούμενο βήμα, τρέχουμε πλέον την απαραίτητη εντολή που θα εγκαταστήσει τα ***node_modules***. Εκθέτουμε την πόρτα ***8000***, με το προκαθορισμένο πρωτόκολλο ***TCP***. Να σημειωθεί ότι η εντολή αυτή δεν κάνει την πόρτα ***8000*** διαθέσιμη εκτός του ***container*** και πρέπει να χρησιμοποιηθεί ένα ξεχωριστό ***flag*** για να είναι πρόσβαση από τον έξω κόσμο, κατά την διάρκεια εκτέλεσης του ***container***. Τέλος εκτελούμε την εντολή ***npm run***, που είναι συντομογραφία για την εντολή ***npm run run***.

Τα υπόλοιπα ***Dockerfile*** αρχεία κινούνται στις ιδίες γραμμές οπότε είναι περιττό να τα αναλύσουμε και αυτά. Πρέπει όμως να σημειωθεί ότι συνήθως χρησιμοποιούνται διαφορετικά

Dockerfile για ανάπτυξη και διαφορετικά για παραγωγή. Μπορούν για αυτό τον λόγο βέβαια να χρησιμοποιηθούν τα ***multistage builds*** που εξαλείφουν την ανάγκη για πολλαπλά ***Dockerfile***, και παράγουν μικρότερα και πιο βελτιστοποιημένα ***images***. Οι παραπάνω 6 γραμμές είναι αρκετές για να φτιάξουμε ένα ***image*** του ***server***. Υπάρχουν δυο επιλογές για το πως θα εκτελέσουμε το ***image*** αυτό μετά την δημιουργία του. Είτε θα το εκτελέσουμε ως μια αυτόνομη οντότητα είτε σε συσχετισμό με τα υπόλοιπα ***container***.

Για τις ανάγκες του ***project*** αυτού χρησιμοποιήσαμε το ***docker-compose***. Με το ***docker-compose*** δίνεται η δυνατότητα να αφαιρέσεις τις ρυθμίσεις σε ένα ***yaml*** αρχείο, να συσχετίσεις όλες τις απαραίτητες λειτουργίες μεταξύ τους, και τέλος να το τρέξεις με μια απλή εντολή. Κοινώς με το ***docker-compose*** μπορούμε να ελέγξουμε όλο τον κύκλο της εφαρμογής.

```
version: "3.9"
services:
  redis:
    image: redis:latest
    container_name: "redis"
    restart: always
    networks:
      - backend
  influx:
    image: influxdb:latest
    container_name: "influx"
    restart: always
    env_file:
      - influx-variables.env
    volumes:
      - ./database/influx:/var/lib/influxdb2
      - ./database/influx:/etc/influxdb2
    networks:
      - backend
      - monitor
  grafana:
    image: grafana/grafana:latest
    container_name: "grafana"
    restart: always
    ports:
      - "3000:3000"
    networks:
```

```
      - monitor
    depends_on:
      - influx
  mongo:
    image: mongo:latest
    container_name: "mongo"
    restart: always
    ports:
      - "27017:27017"
    volumes:
      - ./database/mongo:/data/db
      - ./database/mongo:/data/configdb
    networks:
      - backend
    env_file:
      - mongo-variables.env
  backend:
    build: ./server
    container_name: "backend"
    ports:
      - "8000:8000"
      - "9229:9229"
    volumes:
      - ./server:/usr/src/app/server
      - /usr/src/app/server/node_modules
    networks:
      - backend
      - frontend
    depends_on:
      - redis
      - influx
      - mongo
  frontend:
    build: ./web
    container_name: "frontend"
    ports:
      - "8080:8080"
    volumes:
      - ./web:/usr/src/app/web
      - /usr/src/app/web/node_modules
    networks:
      - frontend
    depends_on:
      - backend
  test_client:
    build: ./client
    container_name: "test_client"
    volumes:
      - ./client:/usr/src/app/client
```

```
      - /usr/src/app/client/node_modules
    networks:
      - backend
    depends_on:
      - backend
networks:
  backend:
  frontend:
  monitor:
volumes:
  mongodb:
  influx:
```

Απόσπασμα κώδικα από docker-compose.yml

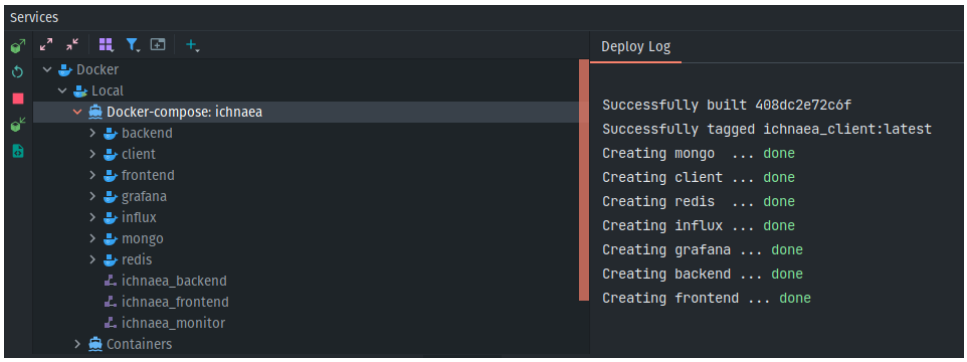
Στην κορυφή κάθε αρχείου ορίζουμε την έκδοση του ***compose*** που θέλουμε. Αυτό έχει πλέον καταργηθεί και δεν είναι αναγκαίο μετα την έκδοση ***docker 19.03+***. Μετά στο σημείο των υπηρεσιών ορίζουμε ρητά την κάθε υπηρεσία της εφαρμογής μας. Πρώτα δίνουμε ένα όνομα στην κάθε υπηρεσία και μετά την μόνη αναγκαία επιλογή διαμόρφωσης η οποία είναι η τοποθεσία του ***dockerfile***. Στην συνέχεια μας δίνεται μια πληθώρα από επιλογές για την παραμετροποίηση της εφαρμογής μας. Για παράδειγμα μπορούμε να ορίσουμε το ***restart policy*** η το όνομα του ***container*** για να αποφύγουμε τα τυχαία ονόματα που δίνει το ***docker*** και δυσκολεύουν την διαχείριση μετέπειτα. Ακόμα, έχουμε προσδιορίσει μερικά ***enviroment files***, με την επιλογή ***env_file***. Αυτό είναι ιδιαίτερα χρήσιμο για να αρχικοποιήσουμε μερικές μεταβλητές στις υπηρεσίες ***influx*** και ***mongo.***

Δυο όμως επιλογές είναι ιδιαίτερα χρήσιμες για την περίπτωση μας. Αρχικά τα ***networks***, μια σειρά από δίκτυα που επιτρέπουν την δημιουργία πολύπλοκων εφαρμογών. Για παράδειγμα στην Ιχνάη, η διασύνδεση των βάσεων δεδομένων με το ***backend*** είναι απομονωμένα από την υπόλοιπη εφαρμογή, όπως και η διασύνδεση του ***backend*** με το ***frontend***. Η αναφορά σε μια άλλη υπηρεσία μέσα στην εφαρμογή, γίνεται πλέον με το όνομα της υπηρεσίας που έχουμε δώσει. Δηλαδή αντί να προσδιορίσουμε την ***IP*** διεύθυνση της ***Influxdb***, χρησιμοποιούμε το όνομα της

υπηρεσίας από το ***docker-compose***, ***"influx"***. Να σημειωθεί ότι αν δεν ορίσουμε ρητά τι είδους δικτύωμα θέλουμε το ***docker*** επιλεγεί ***bridge*** και ***overlay*** δίκτυο, σε ***single host*** και ***swarm*** αντίστοιχα.

Η δεύτερη τέτοια επιλογή παραμετροποίησης είναι τα ***volumes***. Τα ***volumes*** δίνουν την δυνατότητα σε μια υπηρεσία να διαβάζει και να αποθηκεύει δεδομένα στον ***host***. Είναι μια ιδιαίτερα χρήσιμη επιλογή για βάσεις δεδομένων καθώς επιτρέπουν την εξωτερική αποθήκευση, το οποίο σημαίνει ότι τα δεδομένα δεν χάνονται όταν το ***container*** σβήσει η επανεκκινηθεί, και κάνει πολύ πιο εύκολο ένα ***backup strategy***.

Για να τρέξουμε τις υπηρεσίες μας το μόνο που κάνουμε είναι να χρησιμοποιήσουμε το ***docker-compose CLI***, χρησιμοποιώντας την επιλογή ***up*** . Αντίστοιχα, αντί να χρησιμοποιήσουμε την επιλογή ***up,*** χρησιμοποιούμε το ***down*** για να σταματήσουμε τις υπηρεσίες.



*Εικόνα 4.6.1 Εντολή για να 'σηκώσουμε' τις υπηρεσίες μέσα απο **Jetbrains Webstorm IDE**.*

Στον παρακάτω πίνακα είναι οι υπηρεσίες και οι ανάλογες πόρτες που είναι εκτεθειμένες εκτός του ***docker***.

SERVICE	PORT	USAGE
InfluxDB	**{ip}:8086**	**Web UI**
Grafana	**{ip}:3000**	**Web UI**
MongoDB	**{ip}:27017**	**Shell**
Backend	**{ip}:8000**	**Socket and API**

Backend	**{ip}:9229**	**Debugger**
Frontend	**{ip}:8080**	**Web UI**

*Πίνακας 4.1 με εκτεθειμένες πόρτες από την **ICHNAEA**.*

Στην θέση της ***{ip}***, αντικαθιστάτε με ***localhost*** αν τρέχει στο τοπικό μηχάνημα, ή με την ***ip*** του ***server***.

4.7 ΔΟΚΙΜΕΣ

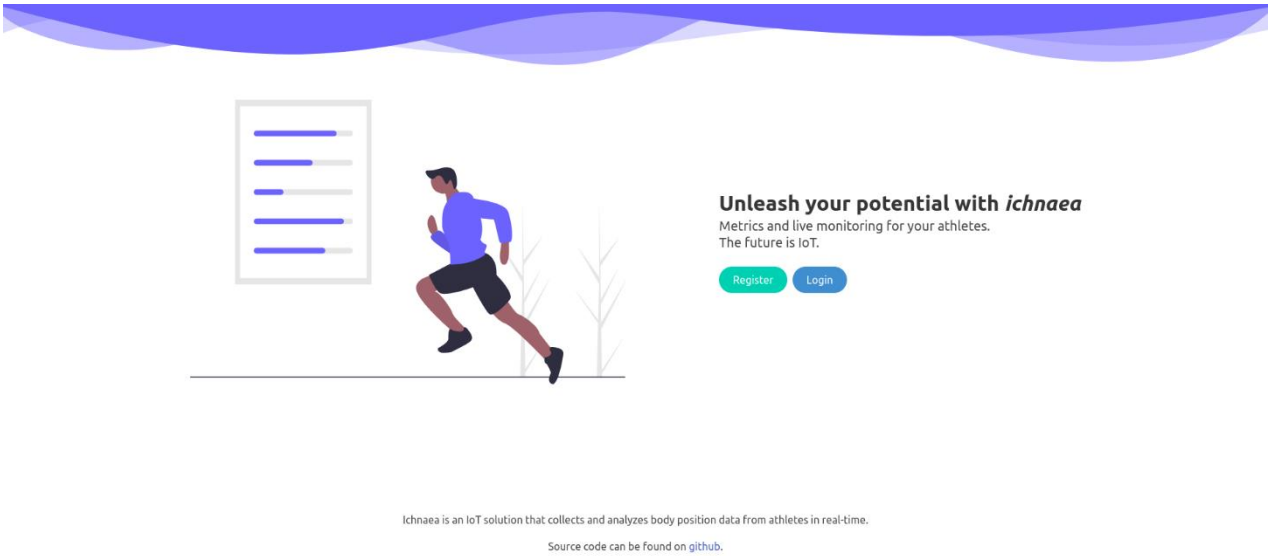
Για να τρέξουμε την ***ICHNAEA*** πρέπει πρώτα να κατεβάσουμε τον πηγαίο κώδικα από το ***github*** [47]. Αυτό είναι απλό και γρήγορο με την εντολή ***git clone*** όπως παρακάτω. Προϋποθέτει να υπάρχει το ***git*** [48] εγκατεστημένο στο σύστημα.

```
git clone git@github.com:xrazis/ichnaea.git
```

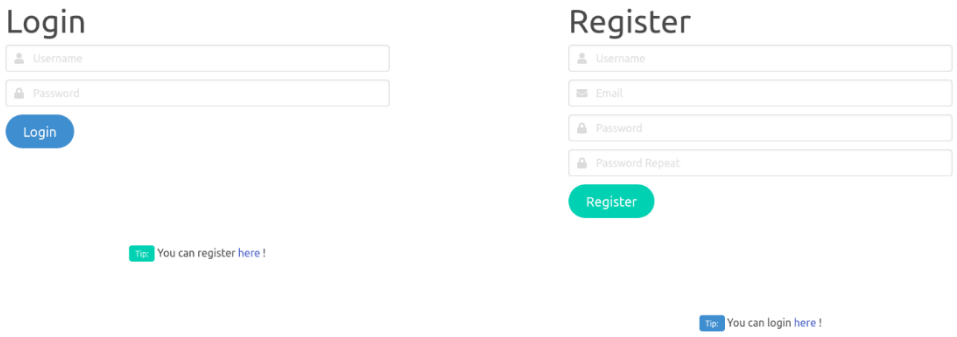
Όταν ολοκληρωθεί το ***clone*** μεταβαίνουμε στον κύριο φάκελο της εφαρμογής και τρέχουμε την παρακάτω εντολή για να εκτελέσουμε το ***docker-compose*** αρχείο.

```
docker-compose -f ./docker-compose.yml up –d –build
```

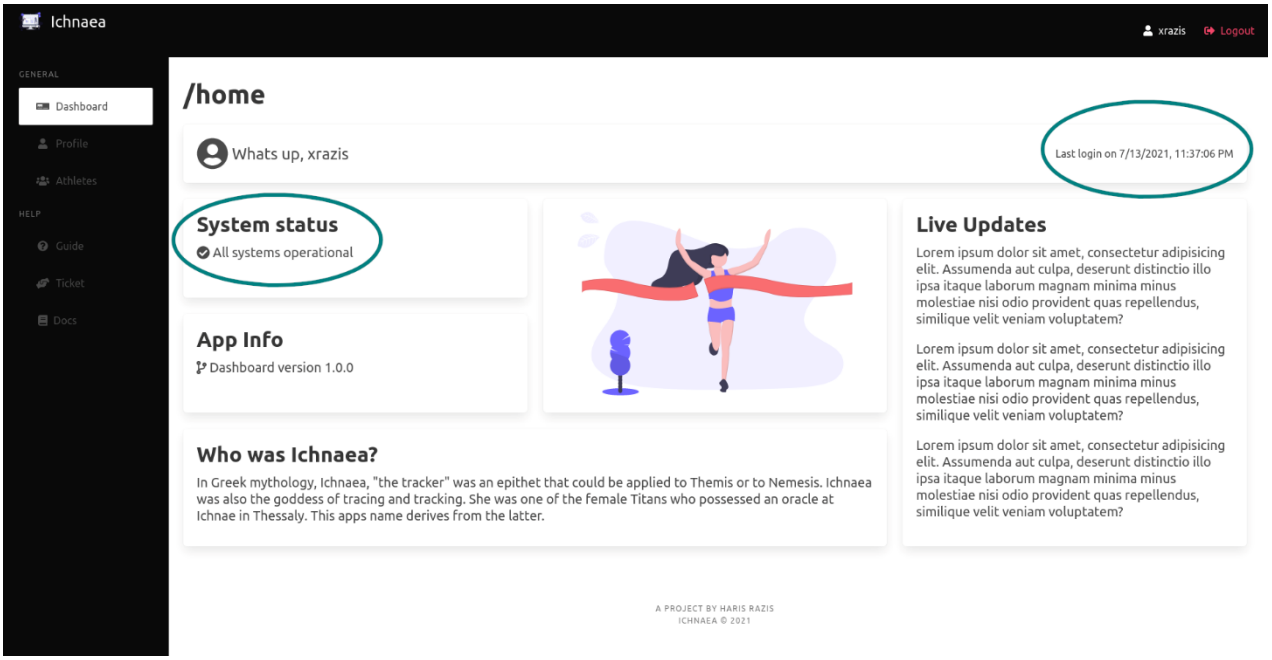
Σε λίγα μόλις λεπτά όλες οι υπηρεσίες θα είναι έτοιμες και μπορουμε πλέον να έχουμε πρόσβαση στο ***dashboard*** της ***ICHNAEA*** στο localhost:8080/. Εκεί έχουμε την επιλογή να εγγράφουμε ή να συνδεθούμε με έναν ήδη υπάρχον χρήστη. Ανεξαρτήτως τι επιλέξουμε, αν το ***http response status*** είναι 200, θα μεταφερθούμε στο ***dashboard***. Εκεί η πρώτη σημαντική πληροφορία που παίρνουμε είναι το ***Server Status***, όπου είναι η επιτυχής ή όχι σύνδεση με ***socket*** στον ***server***.



Εικόνα 4.7.1 Στιγμιότυπο εφαρμογής από ***Ichnaea Home.vue.***

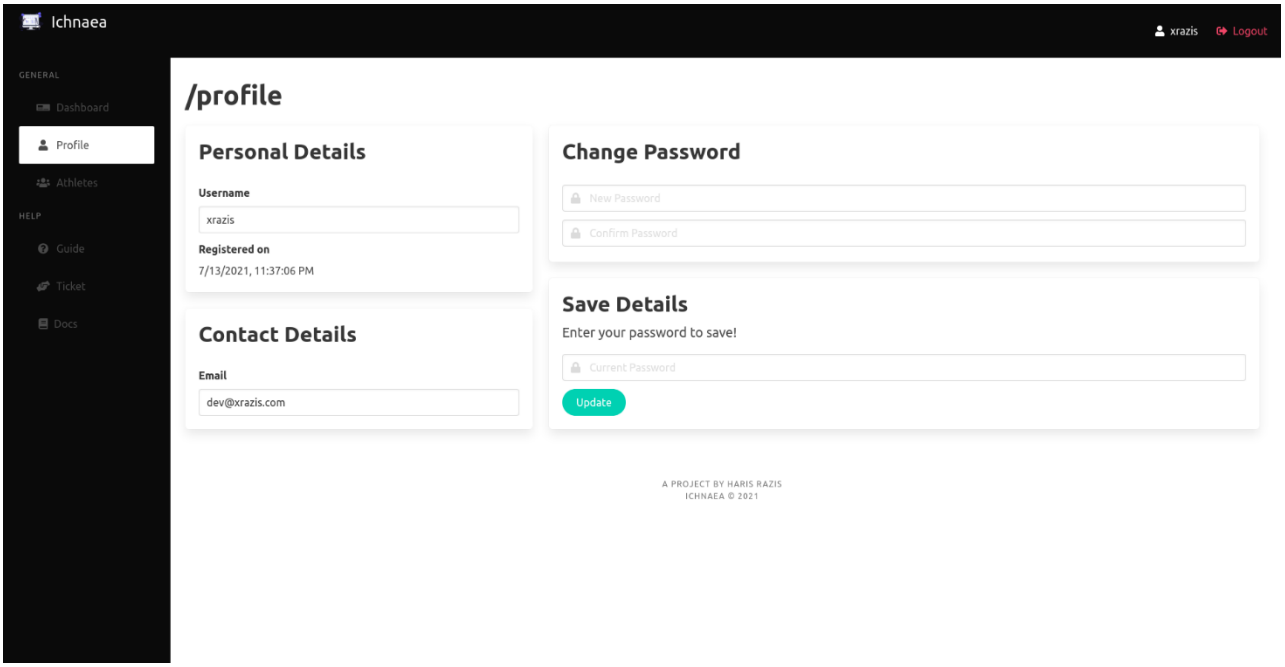


Εικόνα 4.7.2 και 4.7.3 Στιγμιότυπα εφαρμογής από ***Ichnaea Login.vue*** και ***Register.vue*** αντίστοιχα.

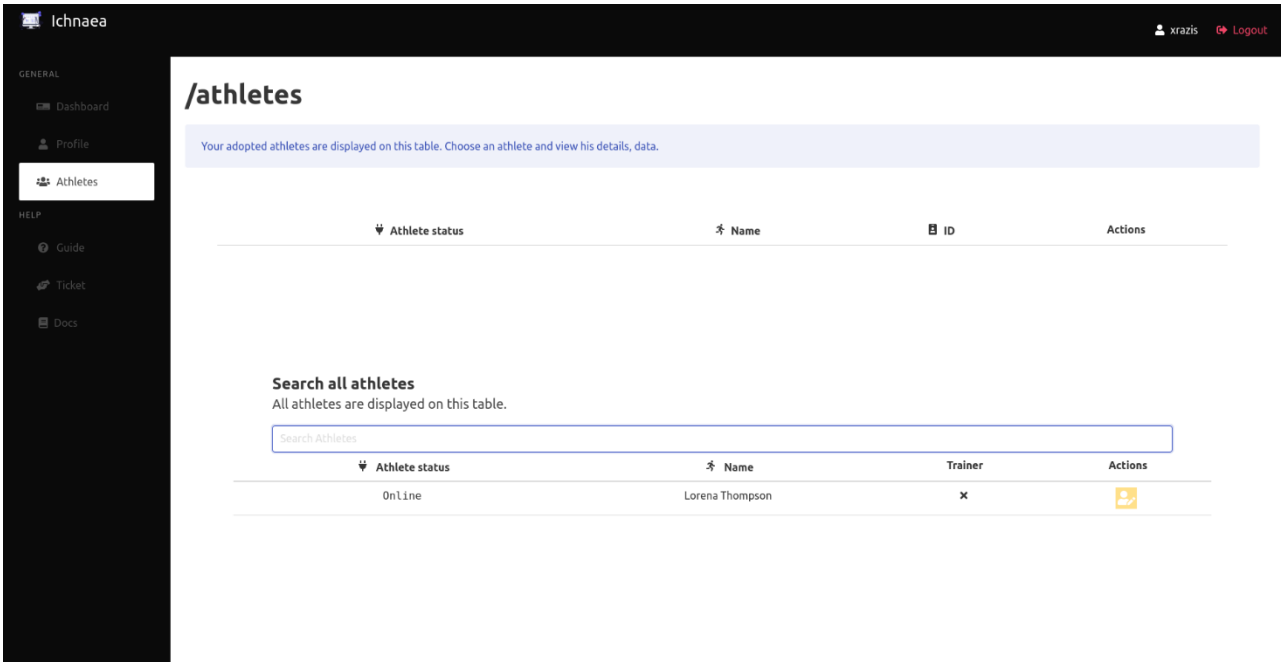


Εικόνα 4.7.4 Στιγμιότυπο εφαρμογής από ***DashboardHome.vue.***

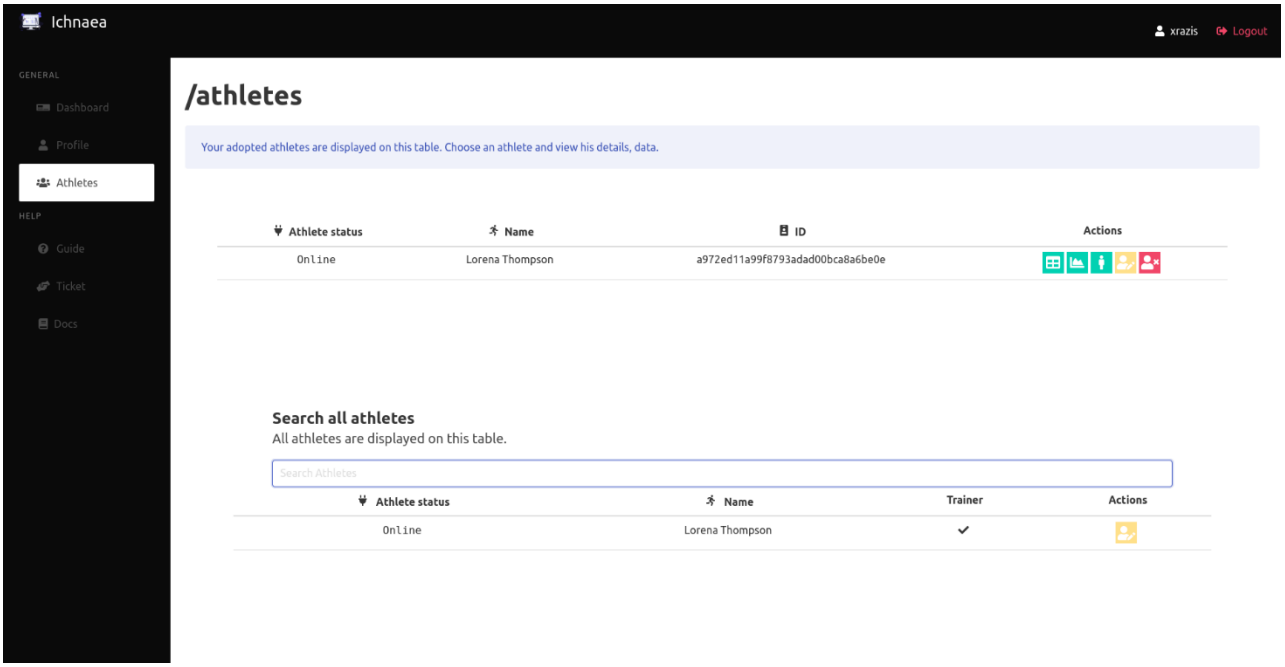
Υπάρχει ακόμα η δυνατότητα να επεξεργαστεί ο χρήστης το προφίλ του, καθώς και των αθλητών του. Πως συσχετίζουμε όμως τους αθλητές με τον προπονητή? Αφού ανακατευθύνουμε στην σελίδα ***Athletes*** από το ***sidemenu***, μας εμφανίζονται 2 λίστες. Η πρώτη λίστα είναι οι αθλητές που θεωρούνται δικοί μας και η δεύτερη είναι όλοι οι διαθέσιμοι αθλητές ανεξάρτητου προπονητή.



Εικόνα 4.7.5 Στιγμιότυπο εφαρμογής από ***Profile.vue****, δυνατότητα επεξεργασίας προφίλ.*

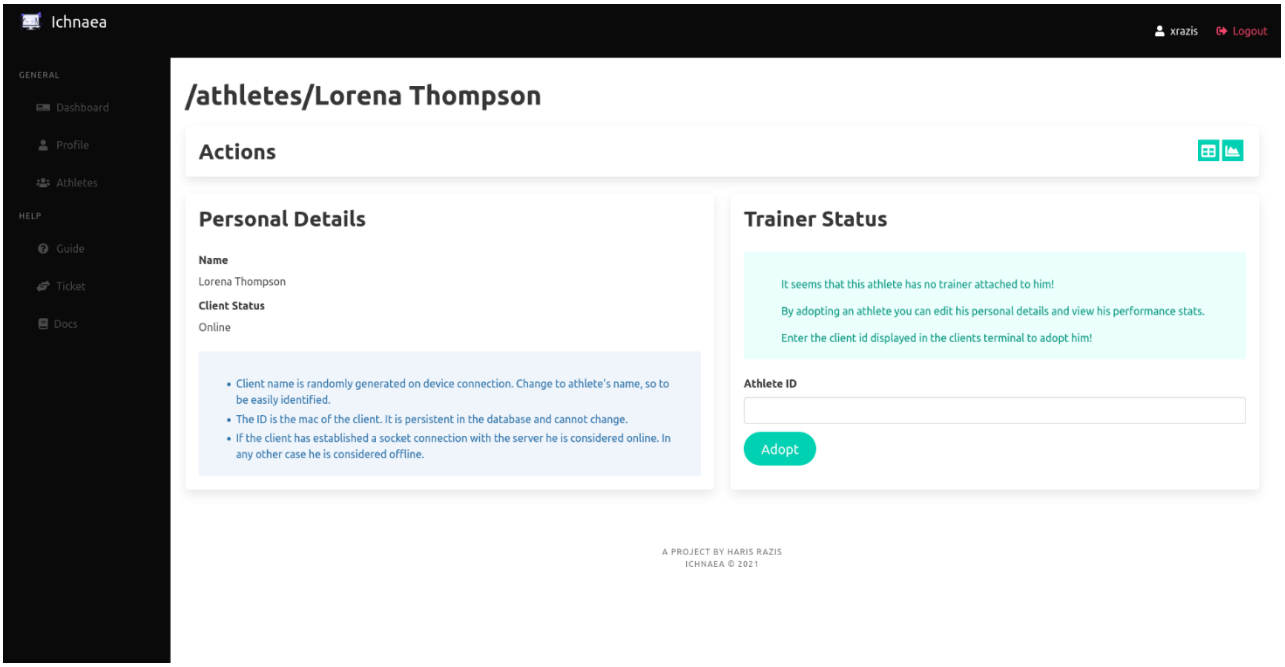


Εικόνα 4.7.6 Στιγμιότυπο εφαρμογής από ***Athletes.vue****, περίπτωση που δεν έχουμε κάποιον αθλητή υπό την αιγίδα μας.*

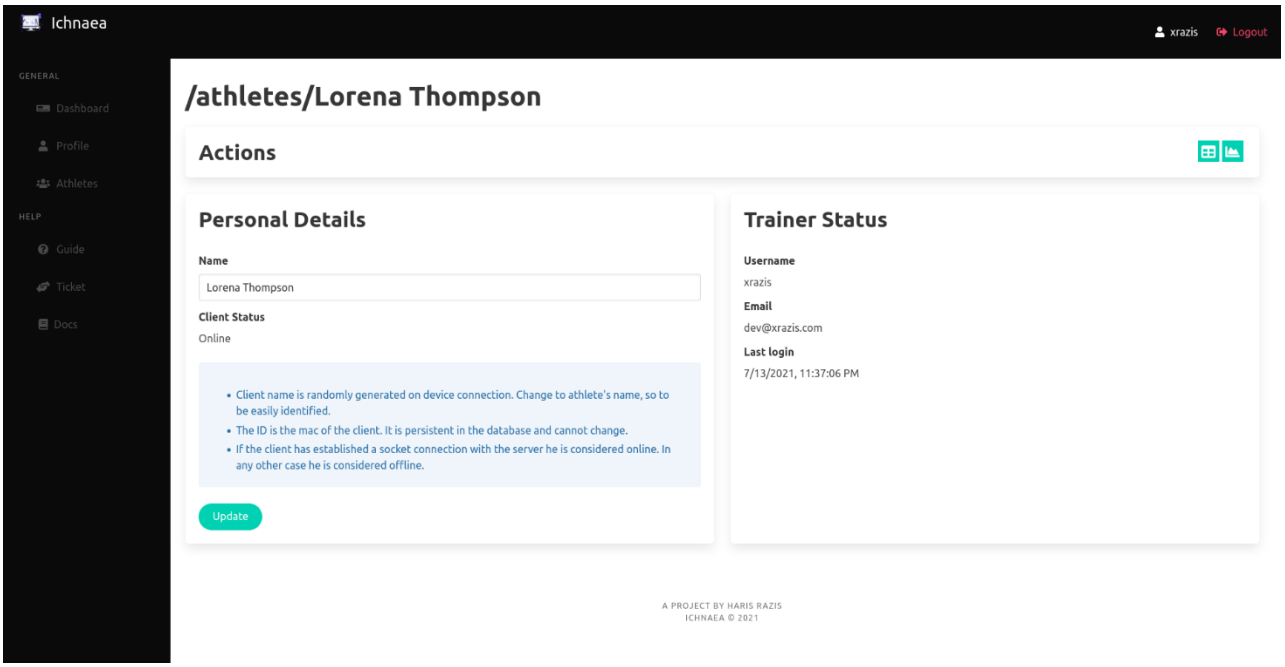


Εικόνα 4.7.7 Στιγμιότυπο εφαρμογής από ***Athletes.vue***, περίπτωση που έχουμε υιοθέτηση κάποιον αθλητή.

Ο αθλητής μπορεί να υιοθετηθεί με το μοναδικό ID που παράγεται κατά την διάρκεια εκκίνησης του προγράμματος ***client***. Είναι ένα ***md5 hash*** της ***MAC*** διεύθυνσης του μηχανήματος που τρέχει το πρόγραμμα. Έτσι μπορουμε να συσχετίσουμε τα δεδομένα του ***client*** κάθε φορά που συνδέεται, και να έχουμε ένα ιστορικό στις βάσεις μας.

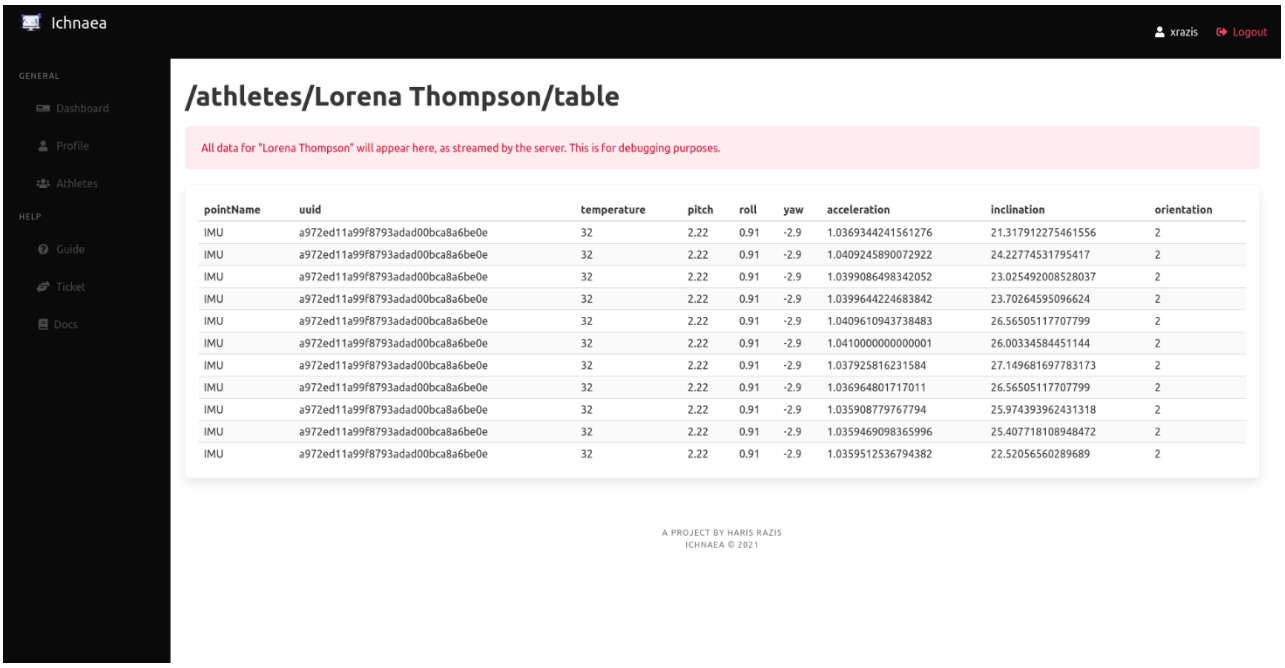


Εικόνα 4.7.8 Στιγμιότυπο εφαρμογής από ***Athlete.vue***, περίπτωση που ο αθλητής δεν έχει κάποιον προπονητή.



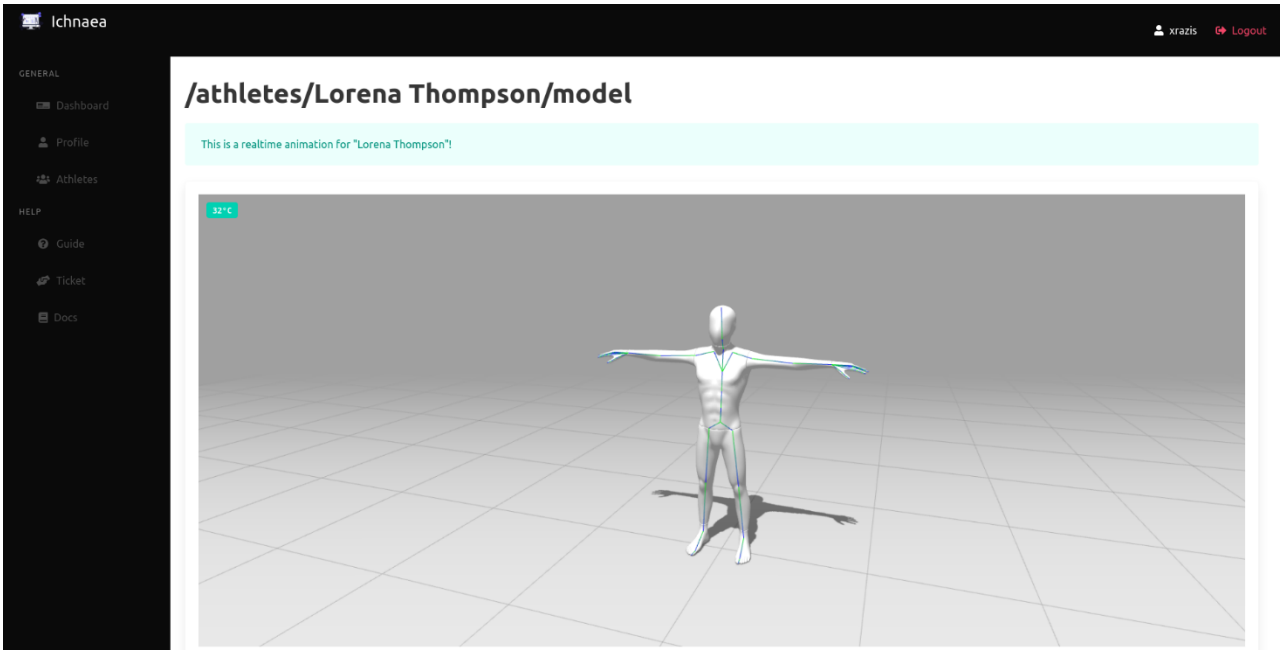
Εικόνα 4.7.9 Στιγμιότυπο εφαρμογής από ***Athlete.vue***, περίπτωση που ο αθλητής έχει προπονητή.

Από την στιγμή που ένας αθλητής έχει προπονητή δεν μπορεί κάποιος άλλος προπονητής να αλλάξει τα στοιχεία του ή να δει τα δεδομένα του. Μπορεί βέβαια ο ήδη υπάρχον προπονητής να τον διαγράψει από τους δικούς του αθλητές και μετά να έχει κάποιος άλλος την δυνατότητα να τον υιοθετήσει.



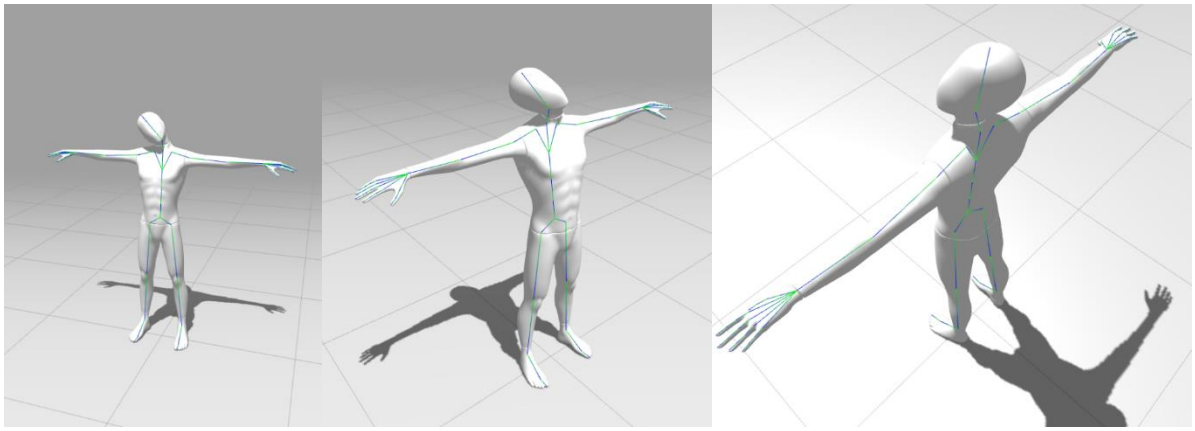
Εικόνα 4.7.10 Στιγμιότυπο εφαρμογής από ***Table.vue.***

Το πρώτο σημαντικό ***view*** που έχει να κάνει με τον αθλητή είναι το ***Table***. Εδώ βλέπουμε τα ωμά δεδομένα, σε πραγματικό χρόνο, όπως τα στέλνει ο ***server***. Ο σκοπός της σελίδας αυτής είναι για αποσφαλμάτωση και για την βεβαίωση της σωστής λειτουργίας του αισθητήρα. Η σελίδα αυτή εμφανίζει όλα τα ***key-value pairs*** από το ***object*** του ***socket***, άρα δεν χρειάζεται να πειράξουμε κάτι στην περίπτωση που αλλάξουμε κάτι ***server-side***.



Εικόνα 4.7.11 Στιγμιότυπο εφαρμογής από ***Model.vue.***

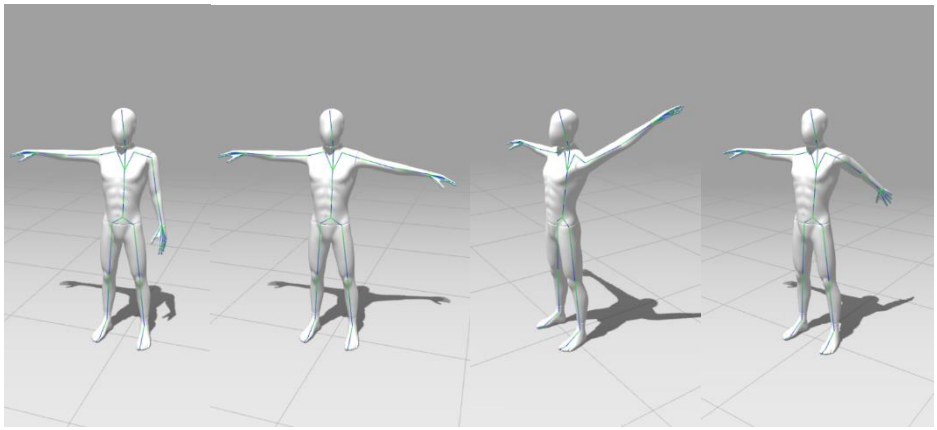
Το επόμενο ***view*** που είναι διαθέσιμο για τον αθλητή είναι το ***Model***. Εδώ χρησιμοποιούμε ένα FBX μοντέλο όπως περιεγράφηκε σε προηγούμενες ενότητες για να απεικονίσουμε την κίνηση ενός αθλητή. Επειδή εργαζόμαστε με ***invesense MPU-6050*** [49] και δεν έχει μαγνητόμετρο, οι μετρήσεις για το ***yaw*** θα έχουν απόκλιση με την πάροδο του χρόνου. Ξεκινάμε με την τοποθέτηση του μικροελεγκτή στο κεφάλι και κάνουμε πρώτα μια μικρή κίνηση δεξιά, κατόπιν προς τα πίσω, και τέλος περιστροφή προς τα αριστερά.



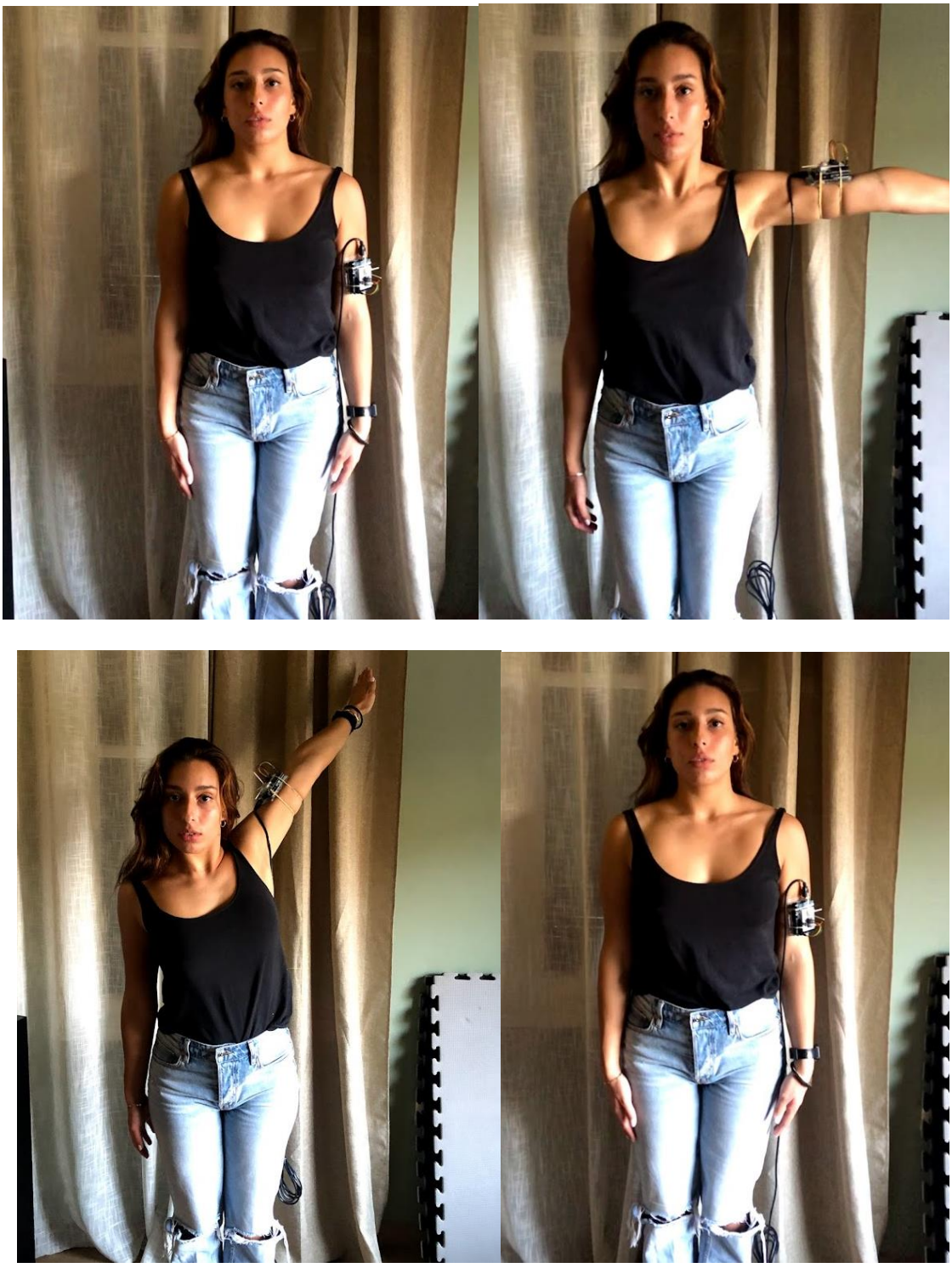
Εικόνα 4.7.12, 4.7.13, και 4.7.14 Στιγμιότυπα εφαρμογής από ***Model.vue***, μετακίνηση κεφαλιού.

Εικόνα 4.7.15, 4.7.16, και 4.7.17 Φωτογραφία από πραγματικό μοντέλο που πραγματοποιεί την μετακίνηση κεφαλιού.

Παρατηρούμε λοιπόν ότι το μοντέλο αντικατοπτρίζει με ακρίβεια τις 3 αυτές κινήσεις. Δηλαδή αυτό που κάναμε είναι να περιστρέφουμε πρώτα στον ***x*** άξονα, μετά στον ***y***, και τέλος στον ***z*** ή αλλιώς ***roll***, ***pitch***, ***yaw***. Στην συνέχεια αλλάζουμε τον αισθητήρα, αυτή την φορά στο χέρι, για να αντικατοπτριστούμε την κίνηση όλου του χεριού στο μοντέλο. Αυτή την φορά θα κάνουμε μια πιο ακαθόριστη κίνηση, ξεκινώντας από καθετή στάση του χεριού διπλά στο πόδι και καταλήγοντας πίσω, πάνω αριστερά έχοντας ακολουθήσει μια ημικυκλική κατεύθυνση. Εκεί θα περιμένουμε λίγη ώρα και στην συνέχεια θα επιστρέψουμε στην αρχική θέση.

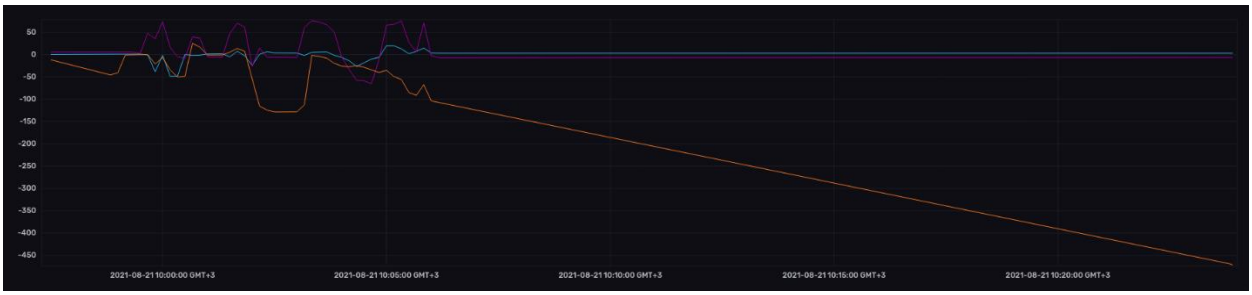


Εικόνα 4.7.18, 4.7.19, 4.7.20, και 4.7.21 Στιγμιότυπα εφαρμογής από ***Model.vue***, μετακίνηση χεριού.



Εικόνα 4.7.22, 4.7.23, 4.7.24, και 4.7.25 Φωτογραφία από πραγματικό μοντέλο που πραγματοποιεί την μετακίνηση χεριού.

Πάλι όπως και πριν, οι κινήσεις σχεδιάζονται στην οθόνη όπως θα έπρεπε μέχρι το πάνω σημείο. Εκεί όσο περισσότερη ώρα περιμένουμε τόσο μεγαλύτερη θα είναι η απόκλιση της γωνίας ***yaw***, και συνεπώς όταν επιστρέψουμε στην αρχική θέση δεν θα αντικατοπτρίζεται η πραγματικότητα. Να σημειωθεί ότι λόγω της έλλειψης μαγνητόμετρου, το ***yaw*** υπολογίζεται βάση της απόκλισης από την αρχική θέσης του γυροσκοπίου και όχι σε σχέση με τον μαγνητικό βορρά.



Εικόνα 4.7.26 Στιγμιότυπο από ***Influx Dashboard***, αναπαράσταση σε γράφημα του ***gyro drift*** κατά την μετακίνηση του χεριού.

Σε ένα γράφημα φαίνεται ξεκάθαρα το ***gyro drift*** που προκύπτει όταν δεν χρησιμοποιούμε άλλο σημείο αναφοράς. Παρατηρείστε την πορτοκαλί γραμμή (***yaw***) που έχει μια σταθερή απόκλιση με την πάροδο του χρόνου.

5 ΣΥΜΠΕΡΑΣΜΑΤΑ - ΜΕΛΛΟΝΤΙΚΕΣ ΠΡΟΒΛΕΨΕΙΣ ΚΑΙ ΒΕΛΤΙΩΣΕΙΣ

Στην διπλωματική αυτή εργασία μελετήθηκαν αντικείμενα όπως αισθητήρες και πρωτόκολλα δικτυών με κύρια εφαρμογή το Διαδίκτυο των Αντικειμένων. Ακόμα εξετάστηκαν περιπτώσεις όπου το IoT έχει εφαρμογές στον αθλητισμό, βοηθώντας στην αναβίωση και την νομισματοποίηση ενός αθλήματος. Τέλος αναπτύχθηκε η ***ICHNAEA***, μια εφαρμογή σχεδιασμένη

σε τεχνολογίες αιχμής, ακολουθώντας πρότυπα και βέλτιστες πρακτικές όπως θα γινόταν σε μια εφαρμογή παραγωγής.

Μια αλλαγή που θα μπορούσε σίγουρα να γίνει στο μέλλον είναι η βελτίωση των αισθητήρων. Με αυτό εννοώ την χρήση ενός μικρότερου μικροελεγκτή, όπως ένα ***arduino nano***, όπου θα μπορούσε να επικολληθεί ο ***IMU,*** μαζί με μια μονάδα ***Bluetooth,*** και μια μικρή μπαταριά, δημιουργώντας έτσι έναν ασύρματο αισθητήρα κίνησης. Έτσι θα ήταν πιο εύκολο να χρησιμοποιηθούν πολλοί αισθητήρες για να αναπαραστήσουν την κίνηση του σώματος με ακρίβεια, συνδεμένοι με ένα ***Raspberry*** που θα δρα σαν ***getaway*** για τις συσκευές. Η εφαρμογή έχει σχεδιαστεί με τέτοιον τρόπο που με ελάχιστες αλλαγές μπορεί να φιλοξενήσει διαφορετικούς αισθητήρες, οπότε καλό θα ήταν επίσης να αναβαθμιστεί ο αισθητήρας ***IMU*** σε κάποιον με ***9-DOF***.

Καταλήγουμε λοιπόν στο συμπέρασμα ότι το μέλλον είναι ***IoT***. Αρκεί κανείς να κοιτάξει γύρω του και να δει τον ολοένα αυξανόμενο αριθμό έξυπνων συσκευών, ήδη τα περισσότερα σπίτια έχουν μια πληθώρα από έξυπνες συσκευές.

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Wikipedia contributors, Machine to machine, (2021, August 18), retrieved from https://en.wikipedia.org/w/index.php?title=Machine_to_machine&oldid=1037400016
2. Pethuru Raj, Anupama C. Raman, (2017), The Internet of things: enabling technologies, platforms, and use cases
3. IEEE, (2015), IEEE Standard for Low-Rate Wireless Networks

4. Ullah, Sana, Manar Mohaisen, and Mohammed A. Alnuem., (April 2013), "A Review of IEEE 802.15.6 MAC, PHY, and Security Specifications." International Journal of Distributed Sensor Networks.
5. Rim Negraa, Imen Jemilia, Abdelfettah Belghitha, (2013), Wireless Body Area Networks: Applications and technologies
6. Paolo Baronti, Prashant Pillai, Vince W.C. Chook, Stefano Chessa, Alberto Gotta, Y. Fun Hu (2007), Wireless sensor networks: A survey on the state of the art and the 802.15.4 and ZigBee standards
7. Bluetooth SIG Proprietary, (2016), Bluetooth Core Specification v5.0
8. BLE, (2021, August 30), retrieved from https://www.bluetooth.com/learn-about-bluetooth/tech-overview/
9. ZigBee, ZigBee Specification, (2021, August 30), retrieved from https://web.archive.org/web/20130627172453/http://www.zigbee.org/Specifications/ZigBee/FAQ.aspx
10. Kshetrimayum, R. (2009). "An introduction to UWB communication systems". IEEE Potentials.
11. ANT, (2014),ANT Message Protocol and Usage, Rev 5.1, retrieved from https://www.thisisant.com/developer/resources/downloads/#documents_tab
12. Electronic Notes, IEEE 802.15.4 Standard: a tutorial / primer, (2021, August 18), retrieved from https://www.electronics-notes.com/articles/connectivity/ieee-802-15-4-wireless/basics-tutorial-primer.php
13. Samaneh Movassaghi, Mehran Abolhasan, Justin Lipman, David Smith, Abbas Jamalipour, Wireless Body Area Networks: A Survey

14. Devan, P. Arun Mozhi; Hussin, Fawnizu Azmadi; Ibrahim, Rosdiazli; Bingi, Kishore; Khanday, Farooq Ahmad (January 2021). "A Survey on the Application of WirelessHART for Industrial Process Monitoring and Control"
15. MiWi™ Protocol, (2021, August 30), retrieved from https://www.microchip.com/en-us/products/wireless-connectivity/sub-ghz/miwi-protocol
16. Nandakishore Kushalnagar, Gabriel Montenegro, Christian Peter Pii Schumacher, (August 2007), IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals, retrieved from https://www.rfc-editor.org/rfc/rfc4919.html
17. Cristian González García, Daniel Meana-Llorián, B. Cristina Pelayo G-Bustelo, and Juan Manuel Cueva Lovelle, A review about Smart Objects, Sensors, and Actuators
18. James Croxon, Types of Gyroscopes, (2021, August 18), retrieved from https://sciencing.com/types-gyroscopes-7329313.html
19. Gobinath Aroganam, Nadarajah Manivannan and David Harrison, (April 2019), Review on Wearable Technology Sensors Used in Consumer Sport Applications
20. Norhafizan Ahmad, Raja Ariffin Raja Ghazilla, and Nazirah M. Khairi, (2013 December), Reviews on Various Inertial Measurement Unit (IMU) Sensor Applications
21. Wikipedia contributors, Inertial measurement unit, (2021, May 28), retrieved from https://en.wikipedia.org/w/index.php?title=Inertial_measurement_unit&oldid=1025660823
22. Wikipedia contributors, Global Positioning System, (2021, August 23), Retrieved from https://en.wikipedia.org/w/index.php?title=Global_Positioning_System&oldid=1040232563

23. *Martin, J.C., Milliken, D.L., Cobb, J.E., McFadden, K.L., & Coggan, A. (1998). Validation of a Mathematical Model for Road Cycling Power. Journal of applied biomechanics, 14 3, 276-291.*

24. EY, How to use IoT and data to transform the economics of a sport, (2021, August 17), retrieved from https://go.ey.com/33MSVIT

25. Velon, Velon data: How it works, (2021, August 17), retrieved from https://www.velon.cc/how-it-works

26. Ray Maker, A Look At Velon's Live Race Tracking System, (2021, August 17), retrieved from https://www.dcrainmaker.com/2018/01/velon-live-tracking-system.html

27. Velon, Velon Live Race Centre, (2021, August 17), retrieved from https://www.velon.cc/news/2020/8/6/velon-live-race-centre

28. Amy Bennett and Rhonda Edwards, (2021, August 18), retrieved from https://www.ibm.com/blogs/internet-of-things/usa-cycling-data/

29. Node.js, (2021, August 30), retrieved from https://nodejs.org/en/

30. V8, (2021, August 30), retrieved from https://v8.dev/

31. Libuv, (2021, August 30), retrieved from https://libuv.org/

32. MongoDB, (2021, August 30), retrieved from https://www.mongodb.com/

33. MongoDB, What is NoSQL?, (2021, August 18), retrieved from https://www.mongodb.com/nosql-explained

34. MongoDB, Advantages of NoSQL Databases, (2021, August 18), retrieved from https://www.mongodb.com/nosql-explained/advantages

35. InfluxDB, (2021, August 30), retrieved from https://www.influxdata.com/

36. InfluxData, Time series database (TSDB) explained, (2021, August 18), retrieved from https://www.influxdata.com/time-series-database/
37. InfluxData, Data Layout and Schema Design Best Practices for InfluxDB, (2021, August 18), retrieved from https://www.influxdata.com/blog/data-layout-and-schema-design-best-practices-for-influxdb/
38. Redis, (2021 August 30), retrieved from https://redis.io/
39. Vue.js, (2021, August 30), retrieved from https://v3.vuejs.org/
40. Vuex, (2021, August 30), retrieved from https://next.vuex.vuejs.org/
41. Vue Router, (2021, August 30), retrieved from https://next.router.vuejs.org/
42. Three.js, (2021, August 30), retrieved from https://threejs.org/
43. Arduino, (2021, August 30), retrieved from https://www.arduino.cc/
44. Johnny-five, (2021, August 30), retrieved from http://johnny-five.io/
45. Raspberry Pi, (2021, August 30), Retrieved from https://www.raspberrypi.org/
46. Docker, (2021, August 30), retrieved from https://www.docker.com/
47. Github, (2021, August 30), retrieved from https://github.com/
48. Git, (2021, August 30), retrieved from https://git-scm.com/
49. Invesense MPU-6050, (2021, August 30), retrieved from https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf
50. Weisstein, Eric W. "Euler Angles.", (2021, August 18), retrieved from *MathWorld*--A Wolfram Web Resource. https://mathworld.wolfram.com/EulerAngles.html
51. Wikipedia contributors, Euler angles, (2021, August 18), retrieved from https://en.wikipedia.org/w/index.php?title=Euler_angles&oldid=1029550074

52. Weisstein, Eric W. "Quaternion.", (2021, August 18), retrieved from *MathWorld*--A Wolfram Web Resource. https://mathworld.wolfram.com/Quaternion.html

53. Wikipedia contributors, Quaternion, (2021, August 18), retrieved from https://en.wikipedia.org/w/index.php?title=Quaternion&oldid=1038494327

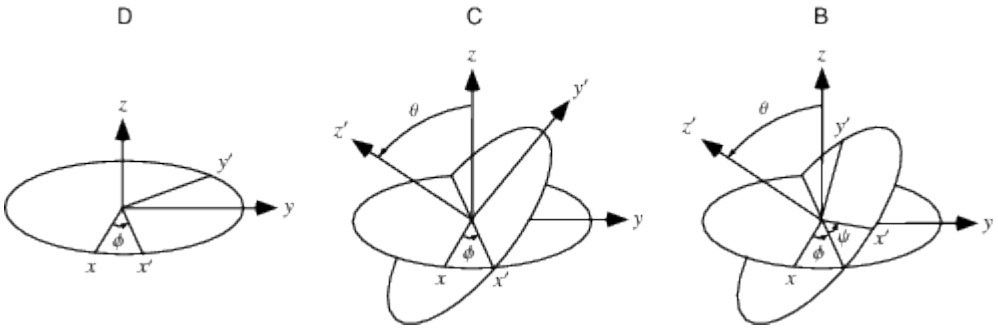
54. Wikipedia contributors, Gimbal lock (2021, January 17), retrieved from https://en.wikipedia.org/w/index.php?title=Gimbal_lock&oldid=1000885298

ΠΑΡΑΡΤΗΜΑ Α

ΜΑΘΗΜΑΤΙΚΕΣ ΕΝΝΟΙΕΣ

EULER ANGLES

Οι τρείς γωνίες ***Euler*** περιγράφουν την κίνηση ενός άκαμπτου σώματος σε σχέση με ένα σταθερό σύστημα συντεταγμένων [50][51]. Σε μηχανικές εφαρμογές τείνουν να αναφέρονται σαν ***pitch***, ***roll***, και ***yaw*** οπού είναι η μετατόπιση στον **y**, **x**, και **z** άξονα αντίστοιχα.



Εικόνα Α.1 Αναπαράσταση ***Euler Agles*** *απο* ***Mathworld***.

QUATERNION

Τα τετραδόνια είναι μια επέκταση των μιγαδικών αριθμών και αντιπροσωπεύονται από την μορφή

$$a + bi + cj + dk$$

οπού τα ***i, j, k*** είναι τα βασικά τετραδόνια, και τα a, b, c, d είναι πραγματικοί αριθμοί [52][53].

IMU

GIMBAL LOCK

Το ***Gimbal Lock*** είναι όταν χάνεται ένας βαθμός ελευθέριας (***DOF***) από ένα τρισδιάστατο μηχανισμό και κλειδώνεται σε δυο διαστάσεις. Αυτό συμβαίνει όταν 2 από τα 3 ***gimbal*** βρίσκονται παράλληλα και συνεπώς οποιαδήποτε τους κίνηση έχει το ίδιο αποτέλεσμα στο σύστημα [54]. Για την αποφυγή του πρέπει να προσεγγιστεί το πρόβλημα διαφορετικά. Τα ***quaternion*** είναι από τις πιο διάσημες λύσεις.