ΠΑΝΕΠΙΣΤΗΜΙΟ
ΙΩΑΝΝΙΝΩΝ

**ΣΧΟΛΗ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

**ΠΜΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΔΙΚΤΥΩΝ**

**ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ**

# ΣΥΣΤΗΜΑ ΜΕΤΑΔΟΣΗΣ ΚΑΙ ΔΙΑΧΕΙΡΙΣΗΣ ΣΗΜΑΤΩΝ ΑΠΟ ΔΙΕΠΑΦΕΣ ΕΓΚΕΦΑΛΟΥ-ΥΠΟΛΟΓΙΣΤΗ

Ανδρέας Ζορπίδης

Επιβλέπων: Γιαννακέας Νικόλαος

Επίκουρος Καθηγητής

Άρτα, Αύγουστος 2023

# SIGNAL TRANSMISSION AND MANAGEMENT SYSTEM FOR BRAIN-COMPUTER INTERFACES

**Εγκρίθηκε από τριμελή εξεταστική επιτροπή**

Άρτα, 25/08/2023

## ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Επιβλέπων καθηγητής

   Νικόλαος Γιαννακέας,

   Επίκουρος Καθηγητής

2. Μέλος επιτροπής

   Αλέξανδρος Τζάλλας,

   Αναπληρωτής Καθηγητής

3. Μέλος επιτροπής

   Καρβέλης Πέτρος,

   Επίκουρος Καθηγητής

Ο Προϊστάμενος του Τμήματος

Ευριπίδης Γλαβάς,

Καθηγητής Α’ Βαθμίδας

Υπογραφή

## Δήλωση μη λογοκλοπής

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

Ζορπίδης Ανδρεάς

Υπογραφή

# ΕΥΧΑΡΙΣΤΙΕΣ

Αρχικά θα ήθελα να ευχαριστήσω όλους τους καθηγητές του Τμήματος Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Ιωαννίνων για τις χρήσιμες γνώσεις και συμβουλές που μου παρείχαν. Συγκεκριμένα είμαι ευγνώμων για τον μέντορα και επιβλέπον καθηγητή της εργασίας αυτής, τον κύριο Νικόλαο Γιαννακέα ο οποίος με στήριζε και καθοδηγούσε μέχρι την ολοκλήρωση της. Ευχαριστώ επίσης την οικογένειά μου για την αμέριστη στήριξη και συμπαράσταση που μου παρείχε καθ' όλη τη διάρκεια των σπουδών μου.

# ΠΕΡΙΛΗΨΗ

Η παρούσα εργασία απευθύνεται κυρίως σε αναγνώστες που είναι ήδη εξοικειωμένοι με τη χρήση διεπαφών εγκεφάλου-υπολογιστή (BCI), και με τις τεχνολογίες που χρησιμοποιούνται από αυτές. Πιο συγκεκριμένα, το πρώτο μέρος της εργασίας αφορά την έρευνα που πραγματοποιήθηκε γύρω από τις πιο γνωστές συσκευές-διεπαφές εγκεφάλου-υπολογιστή στην ακαδημαϊκή κοινότητα, τα πρωτόκολλα επικοινωνίας που χρησιμοποιούν, τις πιο ευρέως διαδεδομένες μεθόδους μετάδοσης των σημάτων τους, και τον ρόλο του υπολογιστικού νέφους σε όλα αυτά με στόχο να λυθεί το πρόβλημα της ζωντανής απομακρυσμένης μετάδοσης των σημάτων μέσω διαδικτύου και να γίνει πιο εύκολη η διαχείρισή τους. Το δεύτερο μέρος της εργασίας, αφορά το πείραμα, δηλαδή τον σχεδιασμό και την υλοποίηση ενός proof of concept (POC) πληροφοριακού συστήματος με χρήση τεχνολογιών διαδικτύου, το οποίο φαίνεται να λύνει το πρόβλημα αυτό και αποτελείται από μία εφαρμογή Full-Stack στο υπολογιστικό νέφος η οποία λειτουργεί ως μεσάζων κατά τη διάρκεια μετάδοσης των σημάτων και μπορεί να τα καταγράφει σε συνεδρίες, καθώς και έναν two-way communication client ο οποίος είναι σε θέση να μεταδίδει, να λαμβάνει, να αποθηκεύει και να αναπαριστά μεταδόσεις σημάτων από τους υπολογιστές των χρηστών. Η εφαρμογή υπολογιστικού νέφους είναι υλοποιημένη σε γλώσσα προγραμματισμού JavaScript και κάνει χρήση τεχνολογιών κυρίως Node.js, Express.js για τη δημιουργία ενός REST-API και WebSockets για την λήψη και μετάδοση των σημάτων, ενώ ο client είναι υλοποιημένος επίσης σε γλώσσα προγραμματισμού JavaScript και κάνει χρήση Electron.js για τη δημιουργία της desktop εφαρμογής και UDP, OSC, WebSockets για την μετάδοση των σημάτων. Στο τέλος της εργασίας, βρίσκονται τα συμπεράσματα και γίνεται μια αναπαράσταση όλων των πιθανών σεναρίων σε εικόνες.

**Λέξεις-κλειδιά:** Διεπαφές Εγκεφάλου-Υπολογιστή, Υπολογιστικό Νέφος, Μετάδοση Σημάτων, Μετάδοση Δεδομένων, Full-Stack Development, JavaScript, WebSockets, UDP, OSC, Σχεδιασμός και Ανάπτυξη.

# ABSTRACT

The present thesis is primarily targeted at readers who are already familiar with brain-computer interfaces (BCI) and the technologies used by them. Specifically, the first part of the thesis focuses on research conducted on the most well-known brain-computer interface devices within the academic community, their communication protocols, widely adopted signal transmission methods, and the role of cloud computing. The objective is to solve the issue of real-time remote signal transmission over the internet and facilitate their management. The second part of the thesis describes an experiment, namely the design and implementation of a proof of concept (POC) informational system using internet technologies, which appears to resolve the problem. The system consists of a Full-Stack application deployed on the cloud, serving as an intermediary during the signal transmission, capable of recording signal sessions. Additionally, it includes a two-way communication client that can transmit, receive, store, and reproduce signal transmissions from users' computers. The cloud-based application is developed in JavaScript, primarily utilizing technologies like Node.js and Express.js to create a REST-API, as well as WebSockets for signal reception and transmission. On the other hand, the client is developed in JavaScript, utilizing Electron.js to create a desktop-suite application and UDP, OSC and WebSockets for signal transmission. Finally, the thesis presents findings and a representation of all the potential scenarios in images.

**Keywords:** Brain-Computer Interfaces, Cloud Computing, Signal Transmission, Data Transmission, Full-Stack Development, JavaScript, WebSockets, UDP, OSC, Design and Development.

# ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

# ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

# ΚΑΤΑΛΟΓΟΣ ΔΙΑΓΡΑΜΜΑΤΩΝ/ΕΙΚΟΝΩΝ

# 1 Εισαγωγή

## 1.1 Οι διεπαφές εγκεφάλου-υπολογιστή

### 1.1.1 Επισκόπηση των διεπαφών εγκεφάλου-υπολογιστή

Η διεπαφή εγκεφάλου-υπολογιστή (BCI), επίσης γνωστή ως διεπαφή εγκεφάλου-μηχανής (BMI) ή άμεση νευρωνική διεπαφή, είναι μια τεχνολογία που δημιουργεί μια άμεση οδό επικοινωνίας μεταξύ του εγκεφάλου και εξωτερικών συσκευών, όπως υπολογιστές ή προσθετικά, χωρίς την ανάγκη για τις παραδοσιακές νευρομυϊκές οδούς. Τα BCI επιτρέπουν αμφίδρομη επικοινωνία: μπορούν να αποκωδικοποιήσουν τα σήματα του εγκεφάλου για να δώσουν εντολή σε εξωτερικές συσκευές και επίσης να κωδικοποιήσουν εξωτερικές αισθητηριακές πληροφορίες που θα γίνουν αντιληπτές από τον εγκέφαλο. Τα BCI έχουν τη δυνατότητα να φέρουν επανάσταση σε διάφορους τομείς, συμπεριλαμβανομένων των ιατρικών εφαρμογών, της υποστηρικτικής τεχνολογίας, της επικοινωνίας, του παιχνιδιού και άλλων.

#### 1.1.1.1 Στοιχεία των BCI

- **Αισθητήρες**: Τα BCI χρησιμοποιούν διάφορους αισθητήρες για την ανίχνευση και τη μέτρηση της νευρικής δραστηριότητας. Η ηλεκτροεγκεφαλογραφία (EEG), η μαγνητοεγκεφαλογραφία (MEG), η λειτουργική μαγνητική τομογραφία (fMRI) και τα ενδοφλοιώδη ηλεκτρόδια είναι συνήθεις τύποι αισθητήρων.
- **Επεξεργασία σήματος**: Τα συλλεγόμενα νευρικά σήματα υποβάλλονται σε επεξεργασία για την εξαγωγή σχετικών πληροφοριών. Οι τεχνικές μηχανικής μάθησης διαδραματίζουν κρίσιμο ρόλο στην αποκωδικοποίηση αυτών των σημάτων και στη μετάφρασή τους σε εντολές με νόημα.
- **Αλγόριθμοι αποκωδικοποίησης**: Οι προηγμένοι αλγόριθμοι αποκωδικοποιούν νευρωνικά μοτίβα για να κατανοήσουν την πρόθεση του χρήστη. Αυτοί οι αλγόριθμοι ερμηνεύουν μοτίβα στην εγκεφαλική δραστηριότητα που σχετίζονται με διαφορετικές εργασίες, κινήσεις ή προθέσεις.

- **Εξωτερικές συσκευές**: Τα BCI διασυνδέονται με εξωτερικές συσκευές όπως υπολογιστές, ρομποτικούς βραχίονες, προσθετικά, συστήματα εικονικής πραγματικότητας ή ακόμα και άλλους ανθρώπους.

#### 1.1.1.2 Τύποι BCI

- **Επεμβατικά BCI**: Αυτά περιλαμβάνουν την τοποθέτηση ηλεκτροδίων απευθείας στον εγκεφαλικό ιστό, παρέχοντας σήματα υψηλής ποιότητας αλλά απαιτώντας χειρουργική εμφύτευση. Χρησιμοποιούνται για ακριβή έλεγχο και συχνά θεωρούνται για ιατρικές εφαρμογές.
- **Μη επεμβατικά BCI**: Δεν απαιτούν χειρουργικές επεμβάσεις και συνήθως μετρούν την εγκεφαλική δραστηριότητα έξω από το κρανίο. Το EEG, το fMRI και το MEG είναι παραδείγματα. Είναι πιο φιλικά προς το χρήστη, αλλά μπορεί να έχουν χαμηλότερη ποιότητα σήματος.

### 1.1.2 Εφαρμογές και οφέλη

Παρακάτω αναλύονται οι βασικές εφαρμογές των διεπαφών εγκεφάλου-υπολογιστή. (Javaid, Adeel. (2013). Brain-Computer Interface. SSRN Electronic Journal. 10.2139/ssrn.2386900.)

- **Ιατρικές εφαρμογές**: Τα BCI μπορούν να βοηθήσουν τα άτομα με παράλυση ή κινητικές αναπηρίες να ανακτήσουν την επικοινωνία και τον έλεγχο της κίνησης. Ερευνώνται για παθήσεις όπως το σύνδρομο lock-in, οι τραυματισμοί του νωτιαίου μυελού και οι νευροεκφυλιστικές ασθένειες.
- **Υποστηρικτική τεχνολογία**: Τα BCI επιτρέπουν στους χρήστες να ελέγχουν υπολογιστές, ρομποτικά μέλη ή άλλες συσκευές χρησιμοποιώντας τις σκέψεις τους. Αυτή η τεχνολογία βελτιώνει την ποιότητα ζωής των ατόμων με αναπηρία.
- **Επικοινωνία**: Τα BCI μπορούν να προσφέρουν ένα νέο κανάλι επικοινωνίας για άτομα με σοβαρές διαταραχές ομιλίας ή καταστάσεις όπως το ALS.

- **Νευροαποκατάσταση**: Τα BCI βοηθούν στην αποκατάσταση μετά το εγκεφαλικό, προάγοντας τη νευρική πλαστικότητα μέσω θεραπειών που βασίζονται στην εγκεφαλική δραστηριότητα.
- **Γνωστική Ενίσχυση**: Τα BCI θα μπορούσαν ενδεχομένως να ενισχύσουν τη μνήμη, τη μάθηση και τις γνωστικές λειτουργίες, αν και αυτός ο τομέας εξακολουθεί να είναι σε μεγάλο βαθμό πειραματικός.
- **Παιχνίδι και ψυχαγωγία**: Τα BCI μπορούν να προσφέρουν καθηλωτικές εμπειρίες σε περιβάλλοντα παιχνιδιών και εικονικής πραγματικότητας, επιτρέποντας στους χρήστες να ελέγχουν χαρακτήρες ή ενέργειες χρησιμοποιώντας τις σκέψεις τους.

### 1.1.3 Προκλήσεις και προβληματισμοί

- **Ποιότητα και αξιοπιστία σήματος**: Τα μη επεμβατικά BCI συχνά υποφέρουν από χαμηλότερη ποιότητα σήματος σε σύγκριση με τις επεμβατικές μεθόδους, επηρεάζοντας την ακρίβεια της αποκωδικοποίησης.
- **Προβλήματα ηθικής και ιδιωτικότητας**: Τα BCI εγείρουν ανησυχίες σχετικά με το απόρρητο των νευρωνικών δεδομένων και την πιθανή κακή χρήση αυτών των δεδομένων.
- **Ασφάλεια και μακροπρόθεσμα εμφυτεύματα**: Τα επεμβατικά BCI απαιτούν χειρουργικές επεμβάσεις, αυξάνοντας τους κινδύνους και τις επιπλοκές. Η μακροπρόθεσμη σταθερότητα και ασφάλεια του εμφυτεύματος είναι σημαντικά ζητήματα.
- **Καμπύλη μάθησης**: Οι χρήστες πρέπει να μάθουν πώς να διαμορφώνουν τη δραστηριότητα του εγκεφάλου τους για να ελέγχουν αποτελεσματικά τις εξωτερικές συσκευές μέσω BCI.
- **Ερμηνεία της νευρικής δραστηριότητας**: Η ακριβής αποκωδικοποίηση πολύπλοκων νευρικών μοτίβων παραμένει μια σημαντική πρόκληση λόγω της περίπλοκης φύσης της εγκεφαλικής δραστηριότητας.

Τα BCI διαθέτουν τεράστιες δυνατότητες για την ενίσχυση των ανθρώπινων ικανοτήτων και τη βελτίωση της ζωής των ατόμων με αναπηρία. Ωστόσο, η έρευνα και η ανάπτυξη συνεχίζονται για την αντιμετώπιση τεχνικών, ηθικών και πρακτικών

προκλήσεων προτού τα BCI γίνουν ευρέως προσβάσιμα και ενσωματωθούν στην καθημερινή ζωή.

### 1.1.4 Πλεονεκτήματα

Οι διεπαφές εγκεφάλου-υπολογιστή (BCI) προσφέρουν μια σειρά από πιθανά οφέλη σε διάφορους τομείς και εφαρμογές. Εδώ είναι μερικά από τα βασικά πλεονεκτήματα των BCI. (Ramadan, Rabie & Refat, Samah & Elshahed, Marwa & Ali, Rasha. (2015). Basics of Brain Computer Interface. Intelligent Systems Reference Library. 74. 31-50. 10.1007/978-3-319-10978-7_2.)

- **Βελτιωμένη επικοινωνία για άτομα με ειδικές ανάγκες**: Τα BCI παρέχουν ένα νέο κανάλι επικοινωνίας για άτομα με σοβαρές κινητικές αναπηρίες, όπως εκείνα με σύνδρομο lock-in ή παράλυση. Αυτό τους επιτρέπει να επικοινωνούν με τους άλλους, να εκφράζουν τις σκέψεις τους και να συμμετέχουν πληρέστερα στις κοινωνικές αλληλεπιδράσεις.
- **Αποκατάσταση της κινητικότητας**: Τα BCI μπορούν να επιτρέψουν στα άτομα με παράλυση να ελέγχουν τις ρομποτικές προσθετικές ή εξωσκελετούς, επιτρέποντάς τους να ανακτήσουν κάποιο βαθμό κινητικότητας και ανεξαρτησίας.
- **Βελτιωμένη ποιότητα ζωής**: Τα BCI μπορούν να βελτιώσουν σημαντικά την ποιότητα ζωής των ατόμων με αναπηρίες αποκαθιστώντας την ικανότητά τους να αλληλεπιδρούν με τον κόσμο και να εκτελούν καθημερινές εργασίες.
- **Νευροαποκατάσταση**: Τα BCI μπορούν να διευκολύνουν τη νευροαποκατάσταση μετά από τραυματισμούς όπως εγκεφαλικά επεισόδια. Προσφέρουν στοχευμένες θεραπείες που προάγουν τη νευρική πλαστικότητα και βοηθούν στην ανάκτηση των κινητικών λειτουργιών.
- **Υποστηρικτική τεχνολογία**: Τα BCI μπορούν να χρησιμεύσουν ως προηγμένες υποστηρικτικές τεχνολογίες, επιτρέποντας στους χρήστες να ελέγχουν υπολογιστές, έξυπνες οικιακές συσκευές και άλλες τεχνολογίες χρησιμοποιώντας τις σκέψεις τους. Αυτό μπορεί να αυξήσει την προσβασιμότητα και την ανεξαρτησία.
- **Αποτελεσματική αλληλεπίδραση ανθρώπου-μηχανής**: Τα BCI έχουν τη δυνατότητα να δημιουργούν απρόσκοπτες και αποτελεσματικές αλληλεπιδράσεις μεταξύ ανθρώπων και μηχανών, ιδιαίτερα σε περιβάλλοντα υψηλής πίεσης όπου οι παραδοσιακές μέθοδοι εισαγωγής μπορεί να είναι μη πρακτικές.

- **Έρευνα για τη λειτουργία του εγκεφάλου**: Τα BCI συμβάλλουν στην κατανόηση της λειτουργίας του εγκεφάλου και της νευρικής δραστηριότητας. Η μελέτη των εγκεφαλικών σημάτων σε διάφορα πλαίσια μπορεί να παρέχει πληροφορίες για τις γνωστικές διαδικασίες, τους μηχανισμούς μάθησης και τις εγκεφαλικές διαταραχές.
- **Θεραπεία για νευρολογικές διαταραχές**: Τα BCI προσφέρουν πολλά υποσχόμενους τρόπους για τη θεραπεία νευρολογικών διαταραχών όπως η επιληψία και η νόσος του Πάρκινσον. Θα μπορούσαν να παρέχουν στοχευμένες θεραπείες με βάση την παρακολούθηση της εγκεφαλικής δραστηριότητας σε πραγματικό χρόνο.
- **Συσκευές ελεγχόμενες από το μυαλό**: Τα BCI ανοίγουν δυνατότητες για έλεγχο συσκευών και τεχνολογίας μόνο με σκέψεις, κάτι που θα μπορούσε να φέρει επανάσταση σε τομείς όπως το παιχνίδι, η εικονική πραγματικότητα και η ψυχαγωγία.
- **Γνωστική Ενίσχυση**: Αν και βρίσκονται ακόμη σε πειραματικά στάδια, τα BCI έχουν δυνατότητες ενίσχυσης των γνωστικών λειτουργιών, της μνήμης και της μάθησης, οδηγώντας σε βελτιωμένες νοητικές ικανότητες.
- Προσαρμοστικές διεπαφές: Τα BCI μπορούν να προσαρμοστούν στην ψυχική κατάσταση του χρήστη, επιτρέποντας στις συσκευές να ανταποκρίνονται κατάλληλα με βάση παράγοντες όπως η προσοχή, το άγχος ή τα επίπεδα εστίασης.
- Μελλοντικές καινοτομίες: Καθώς οι BCI συνεχίζουν να εξελίσσονται, ενδέχεται να επιτρέψουν εντελώς νέες δυνατότητες που δεν μπορούμε ακόμη να φανταστούμε πλήρως, οδηγώντας δυνητικά σε καινοτομίες σε τομείς όπως η εκπαίδευση, η δημιουργικότητα και η επίλυση προβλημάτων.

### 1.1.5 Δημοφιλείς οικογένειες συσκευών

- **Muse** (Muse S – 4 ch., **Muse 2** – 4 ch., Muse (original) – 4 ch.)
- **OpenBCI** (Cyton – 8-16 ch., Ganglion – 4 ch., Cyton+Daisy – 8-16 ch., Ultrasound – 4 ch.)
- **Emotiv** (**EPOC X** – 14 ch., EPOC+ - 14 ch., Insight – 5 ch., **EPOC Flex** – 32 ch.)

- NeuroSky (MindWave Mobile – 1 ch., MindWave Mobile 2 – 1 ch., Dev Edition – 8 ch.)
- g.Tec (g.USBamp – 16-64 ch., g.Nautilus – 32-128 ch., g.HEADstage – 8-32 ch, g.LADYbird – 16-64 ch., gHlamp – 256 ch.)
- BrainVision (BrainAmp – 16-128 ch., LiveAmp – 8-64 ch., R-Net – 32-138 ch., actiCHamp – 256 ch.)
- BioSemi (ActiveTwo – 32-512 ch., ActiveTwo Touch – 31-512 ch., EEG Caps – 32-128 ch.)
- Neurobit Systems (Optima+ - 16-32 ch., Optima Sport – 8-24 ch., Optima 4/8/20/32 – 4-31 ch.)

Η **ακαδημαϊκή κοινότητα** αλλά και οι **ερασιτέχνες**, τα τελευταία χρόνια φαίνεται να έχουν δείξει αρκετό **ενδιαφέρον για** τις σειρές προϊόντων **Muse, OpenBCI και Emotiv λόγω** του **χαμηλού κόστους** τους και της **εύκολης διαθεσιμότητάς** τους.

Το Εργαστήριο Αλληλεπίδρασης Ανθρώπου Υπολογιστή (HCILAB) του Τμήματος Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Ιωαννίνων διέθετε στον εξοπλισμό του τις συσκευές Muse 2, EPOC X και EPOC Flex, οπότε πάρθηκε η απόφαση να εστιαστεί η έρευνα στα οικοσυστήματα αυτών των σειρών και να αξιοποιηθούν αυτές οι συσκευές.

Στο πειραματικό σύστημα που παρουσιάζεται στο Β΄ μέρος, έγιναν μετρήσεις με χρήση της συσκευής Muse 2 και σύνδεσή της με τον Η/Υ με χρήση Bluetooth.

### 1.1.6 Πρωτόκολλα & μέθοδοι επικοινωνίας

Τα ηλεκτρικά σήματα του εγκεφάλου συλλέγονται με χρήση ηλεκτροδίων και μετατρέπονται σε αναλογικά ηλεκτρικά σήματα από τους ενισχυτές των EEG συσκευών.

Έπειτα, τα αναλογικά αυτά σήματα μετατρέπονται σε ψηφιακά από έναν analog-to-digital converter (ADC) εντός της συσκευής. Αυτή η διαδικασία, μετατρέπει τις συνεχείς αναλογικές κυματομορφές σε διακριτά ψηφιακά δείγματα.

Μετά τη ψηφιοποίηση, η εκάστοτε συσκευή εφαρμόζει (όχι πάντα) φίλτρα, υποδειγματοληψία ή κάποιου είδους επεξεργασία για τη βελτίωση της ποιότητας του σήματος.

Τέλος, τα δεδομένα αποστέλλονται στον ελεγκτή/διεπαφή επικοινωνίας της συσκευής όπως USB, Bluetooth, Wi-Fi, Ethernet ή κάτι άλλο ανάλογα τη συσκευή, με χρήση συνήθως σειριακής επικοινωνίας RS-232΄ή UART.

Η **συσκευή που λαμβάνει τα δεδομένα** όπως ένα κινητό τηλέφωνο ή ένας ηλεκτρονικός υπολογιστής, πρέπει με κάποιο τρόπο να **τα κάνει διαθέσιμα** προς επεξεργασία ή προβολή **στο περιβάλλον του λειτουργικού της συστήματος**.

Σε αυτό το στάδιο, οι περισσότερες εταιρείες φαίνεται να χρησιμοποιούν δικές τους τεχνικές και **ιδιόκτητο (proprietary) λογισμικό** αποκλείοντας την άμεση πρόσβαση στα δεδομένα. Με πολύ σπάνιες περιπτώσεις όπου παρέχεται ζωντανή **ροή δεδομένων σε πρωτόκολλα LSL ή OSC**.

Γι’ αυτό το λόγο, η ερευνητική κοινότητα έχει δημιουργήσει “clients” ανοιχτού κώδικα για να λύσει το πρόβλημα αυτό.

**Τι είναι το LSL (Lab Streaming Layer);**

Είναι ένα πρωτόκολλο επικοινωνίας ανοιχτού κώδικα που αναπτύχθηκε από το SCCN (Swartz Center for Computational Neuroscience) στο Πανεπιστήμιο της Καλιφόρνια στο Σαν Ντιέγκο.

Έχει σχεδιαστεί για να **διευκολύνει** τη **μετάδοση ροών δεδομένων** χρονοσειρών **σε πραγματικό χρόνο**, όπως του EEG, από διάφορες συσκευές εγγραφής προς πολλαπλές εφαρμογές πελάτη (clients) που εκτελούνται **στον ίδιο ή διαφορετικούς υπολογιστές**. Χρησιμοποιείται ευρέως στην έρευνα νευροεπιστήμης και σε εφαρμογές διεπαφής εγκεφάλου-υπολογιστή (BCI). (LabStreamingLayer, n.d.)

Προσφέρει έναν τυποποιημένο και αποτελεσματικό τρόπο ανταλλαγής δεδομένων μεταξύ συσκευών και εφαρμογών λογισμικού, επιτρέποντας την απρόσκοπτη ενοποίηση και συγχρονισμό δεδομένων σε διαφορετικά ερευνητικά εργαλεία και πλατφόρμες.

**Τι είναι το OSC (Open Sound Control);**

Είναι ένα πρωτόκολλο που **χρησιμοποιείται** συνήθως στους τομείς της **μουσικής**, των **πολυμέσων** και της **διαδραστικής τέχνης**, αλλά έχει βρει εφαρμογές και στη ροή δεδομένων της **νευροεπιστήμης** και του EEG.

Το OSC έχει σχεδιαστεί για έλεγχο και επικοινωνία σε πραγματικό χρόνο μεταξύ συσκευών λογισμικού και υλικού.

Βασίζεται σε μια απλή μορφή μηνύματος που μπορεί να μεταφέρει διάφορους τύπους δεδομένων, καθιστώντας το ευέλικτο για διαφορετικές εφαρμογές.

**Λειτουργεί** σε **διαφορετικά επίπεδα μεταφοράς**, όπως **UDP** και **TCP/IP**, και επιτρέπει αμφίδρομη επικοινωνία, καθιστώντας το κατάλληλο για διαδραστικές εφαρμογές.

Στο πλαίσιο του EEG, το OSC χρησιμοποιείται συχνά για τη μετάδοση δεδομένων, μηνυμάτων της συσκευής ή διάφορων δεικτών σε πραγματικό χρόνο.

### 1.1.7 Λογισμικά προβολής και επεξεργασίας δεδομένων

Παρακάτω, είναι μια λίστα από μερικά λογισμικά (ανοιχτού και κλειστού κώδικα) τα οποία **αξιοποιούν** τα πρωτόκολλα **LSL** και **OSC**, και ως ένα βαθμό επιτρέπουν την **ζωντανή προβολή της τοπικής ροής δεδομένων**, την **αποθήκευση**, την **επεξεργασία** και την αναπαράσταση δεδομένων **από συνεδρίες EEG**.

- OpenVibe
- Neuromore

- EmotivePRO
- Muse Direct
- BCI2000
- MATLAB με EEGLAB και BCILAB
- BioEra
- Βιβλιοθήκες σε γλώσσα Python, όπως NumPy, SciPy.

## 1.2 Το υπολογιστικό νέφος

### 1.2.1 Βασικά χαρακτηριστικά

Το cloud computing είναι μια τεχνολογία που παρέχει πρόσβαση κατ' απαίτηση σε μια κοινή δεξαμενή υπολογιστικών πόρων μέσω του Διαδικτύου. Προσφέρει ένα ευρύ φάσμα υπηρεσιών, όπως υπολογιστική ισχύ, αποθήκευση, βάσεις δεδομένων, δικτύωση, λογισμικό και πολλά άλλα. Τα βασικά χαρακτηριστικά του υπολογιστικού νέφους συχνά συνοψίζονται χρησιμοποιώντας τον «Ορισμό Υπολογιστικού Νέφους NIST» που παρέχεται από το Εθνικό Ινστιτούτο Προτύπων και Τεχνολογίας (NIST). (Birje, Mahantesh & Challagidad, Praveen & Goudar, R.H. & Tapale, Manisha. (2017). Cloud computing review: Concepts, technology, challenges and security. International Journal of Cloud Computing. 6. 32. 10.1504/IJCC.2017.083905.) Αυτά τα χαρακτηριστικά είναι:

- Αυτοεξυπηρέτηση κατ' απαίτηση: Οι χρήστες μπορούν να παρέχουν και να διαχειρίζονται υπολογιστικούς πόρους, όπως επεξεργαστική ισχύ, αποθήκευση και δικτύωση, χωρίς να απαιτείται ανθρώπινη αλληλεπίδραση με τον πάροχο υπηρεσιών. Αυτό επιτρέπει επεκτασιμότητα και ευελιξία στην κατανομή των πόρων.
- Ευρεία πρόσβαση στο δίκτυο: Οι υπηρεσίες Cloud είναι προσβάσιμες μέσω του Διαδικτύου από διάφορες συσκευές, όπως φορητούς υπολογιστές, smartphone, tablet και επιτραπέζιους υπολογιστές. Αυτή η προσβασιμότητα επιτρέπει την απομακρυσμένη πρόσβαση και τη συνεργασία σχεδόν από οπουδήποτε.
- Συγκέντρωση πόρων: Οι πάροχοι cloud χρησιμοποιούν μοντέλα πολλαπλών μισθωτών για τη συγκέντρωση πόρων υπολογιστών και την εξυπηρέτηση πολλών πελατών ταυτόχρονα. Οι πόροι εκχωρούνται και εκχωρούνται δυναμικά με βάση τη ζήτηση, βελτιστοποιώντας τη χρήση των πόρων και την αποτελεσματικότητα.
- Ταχεία ελαστικότητα: Οι πόροι του cloud μπορούν να κλιμακωθούν γρήγορα προς τα πάνω ή προς τα κάτω με βάση τις απαιτήσεις φόρτου εργασίας. Αυτή η ελαστικότητα επιτρέπει στους χρήστες να χειρίζονται διαφορετικά επίπεδα χρήσης χωρίς να αντιμετωπίζουν σημαντική υποβάθμιση της απόδοσης.

- Μετρημένη υπηρεσία: Η χρήση του cloud μετράται και οι χρήστες χρεώνονται με βάση την πραγματική κατανάλωση πόρων τους. Αυτό το pay-as-you-go μοντέλο επιτρέπει τη οικονομικά αποδοτική χρήση και αποφεύγει την υπερβολική παροχή πόρων.

Αυτά τα χαρακτηριστικά επιτρέπουν συλλογικά στο cloud computing να παρέχει διάφορα μοντέλα ανάπτυξης και μοντέλα υπηρεσιών:

### 1.2.2 Μοντέλα υπηρεσιών

- Υποδομή ως υπηρεσία (IaaS): Παρέχει εικονικούς υπολογιστικούς πόρους, συμπεριλαμβανομένων εικονικών μηχανών, αποθήκευσης και στοιχείων δικτύου. Οι χρήστες έχουν τον έλεγχο των λειτουργικών συστημάτων και των εφαρμογών που εκτελούν στην παρεχόμενη υποδομή.
- Πλατφόρμα ως υπηρεσία (PaaS): Προσφέρει μια πλατφόρμα και ένα περιβάλλον στους προγραμματιστές να δημιουργούν, να αναπτύσσουν και να διαχειρίζονται εφαρμογές χωρίς να ανησυχούν για τις υποκείμενες λεπτομέρειες υποδομής. Το PaaS περιλαμβάνει συνήθως εργαλεία ανάπτυξης, συστήματα διαχείρισης βάσεων δεδομένων και περιβάλλοντα χρόνου εκτέλεσης.
- Λογισμικό ως υπηρεσία (SaaS): Παραδίδει εφαρμογές λογισμικού μέσω Διαδικτύου με συνδρομή. Οι χρήστες μπορούν να έχουν πρόσβαση και να χρησιμοποιούν αυτές τις εφαρμογές χωρίς να χρειάζεται να τις εγκαταστήσουν ή να τις συντηρήσουν τοπικά.

(Al-Jabri, Ibrahim. (2014). The Perceptions of Adopters and Non-adopters of Cloud Computing: Application of Technology-Organization-Environment Framework.)

Το cloud computing έχει γίνει μια θεμελιώδης τεχνολογία για επιχειρήσεις, ιδιώτες και οργανισμούς, δίνοντάς τους τη δυνατότητα να έχουν πρόσβαση και να αξιοποιούν ισχυρούς υπολογιστικούς πόρους χωρίς να χρειάζονται μεγάλες επενδύσεις σε υλικό και υποδομή.

### 1.2.3 Μοντέλα ανάπτυξης

- Δημόσιο Cloud: Οι πόροι του Cloud ανήκουν και λειτουργούν από τρίτους παρόχους υπηρεσιών και διατίθενται στο ευρύ κοινό μέσω του Διαδικτύου. Οι χρήστες μοιράζονται πόρους με άλλους πελάτες, επωφελούμενοι από οικονομίες κλίμακας και αποφεύγοντας την ανάγκη διαχείρισης υποδομής.
- Ιδιωτικό Cloud: Οι πόροι του cloud χρησιμοποιούνται αποκλειστικά από έναν οργανισμό. Μπορούν να φιλοξενηθούν εσωτερικά ή εξωτερικά και προσφέρουν μεγαλύτερο έλεγχο και προσαρμογή, κάτι που είναι χρήσιμο για οργανισμούς με συγκεκριμένες απαιτήσεις ασφάλειας ή συμμόρφωσης.
- Υβριδικό Cloud: Ένας συνδυασμός δημόσιων και ιδιωτικών cloud, που επιτρέπει την κοινή χρήση δεδομένων και εφαρμογών μεταξύ τους. Αυτό το μοντέλο προσφέρει ευελιξία και βελτιστοποίηση για ποικίλους φόρτους εργασίας και ευαισθησίες δεδομένων.

## 1.3 Το πρόβλημα

Ενώ **έχει βρεθεί** από την ερευνητική κοινότητα η **λύση** για την εύκολη **μετάδοση** της **ροής** των **δεδομένων τοπικά** εντός της συσκευής π.χ. Η/Υ., προκειμένου να αξιοποιεί ο χρήστης τα λογισμικά που αναφέρθηκαν για επεξεργασία και οπτικοποίηση των δεδομένων, κανένα λογισμικό ή βιβλιοθήκη ανοιχτού κώδικα από αυτά που αναφέρθηκαν, δεν επιτρέπει την ζωντανή απομακρυσμένη μετάδοση των ροών δεδομένων από συνεδρίες EEG.

Όσα παρέχουν αυτή τη λειτουργία, όπως τα EmotivPPO και Muse Direct, είναι **κλειστού κώδικα, απαιτούν συνδρομή** και **αποτελούν τμήματα από μεγαλύτερα πληροφοριακά συστήματα** που κάνουν χρήση τεχνολογιών διαδικτύου και του **υπολογιστικού νέφους**.

Επιπροσθέτως, βρέθηκαν ελάχιστες πληροφορίες και δημοσιεύσεις που αναφέρονται στο πρόβλημα αυτό, δίχως να παρουσιάζουν μια ολοκληρωμένη μέθοδο ή λύση.

## 1.4 Πιθανές λύσεις

Από την έρευνα που πραγματοποιήθηκε, **βρέθηκαν δύο πιθανές** και σχετικά απλές **λύσεις** που να **αξιοποιούν** τα **πρωτόκολλα LSL** ή **OSC**.

1. Υλοποίηση απομακρυσμένης πρόσβασης με τη **μέθοδο port forwarding** η οποία απαιτεί σύνθετες ρυθμίσεις και από τις δύο πλευρές (αποστολέα-παραλήπτη) και δεν αποτελεί την ασφαλέστερη επιλογή διότι εκθέτει ένα μέρος του εσωτερικού δικτύου στο διαδίκτυο.

2. Υλοποίηση **συστήματος** που αποτελείται από μια **εφαρμογή στο υπολογιστικό νέφος** και λειτουργεί ως **μεσάζων** κατά τη διάρκεια της επικοινωνίας και ένα **λογισμικό τύπου client** στους ηλεκτρονικούς υπολογιστές των δύο πλευρών (αποστολέα-παραλήπτη) το οποίο είναι σε θέση είτε να λαμβάνει τοπικές ροές δεδομένων από LSL ή OSC και να τις προωθεί στο υπολογιστικό νέφος, είτε να λαμβάνει ροές δεδομένων από το υπολογιστικό νέφος και να τις προωθεί τοπικά σε LSL ή OSC.

# 2 Μεθοδολογία

## 2.1 Αξιολόγηση πρωτοκόλλων & μεθόδων

### 2.1.1 Μεθοδολογία Α

Αρχικά, έγιναν δοκιμές με τη **μεθοδολογία port forwarding**. Καθώς το LSL δεν βασίζεται στην άμεση επικοινωνία δικτύου όπως τα τυπικά πρωτόκολλα δικτύου, ούτε χρησιμοποιεί εξωτερικές διευθύνσεις IP, η μεθοδολογία port forwarding δεν είναι εφικτή με αυτό. Σε αντίθεση με το LSL, το **OSC** λειτουργεί μέσω τυπικής επικοινωνίας δικτύου ΚΑΙ χρησιμοποιεί το πρωτόκολλο UDP για την αποστολή και λήψη μηνυμάτων μεταξύ συσκευών. Έτσι, έπρεπε:

1. να βρεθεί η διεύθυνση IP του Η/Υ του αποστολέα της ροής δεδομένων στον οποίο ήταν συνδεδεμένη η EEG συσκευή,
2. να απενεργοποιηθεί το τοίχος προστασίας του λειτουργικού συστήματος,
3. να ενεργοποιηθεί η λειτουργία port forwarding από τις ρυθμίσεις του δρομολογητή (router) για την θύρα εκπομπής της συσκευής και από τη πλευρά του παραλήπτη να δηλωθούν στο πρόγραμμα της επιλογής του η διεύθυνση και η θύρα του αποστολέα.

Εδώ, αξίζει να σημειωθεί ότι εκτός από το θέμα ασφαλείας που προκύπτει, ενδεχομένως να υπάρχουν προβλήματα με τη σύνδεση λόγω ορίων από τον πάροχο διαδικτύου (ISP) ή της μορφολογίας του εκάστοτε εσωτερικού δικτύου.

### 2.1.2 Μεθοδολογία Β

Για την δεύτερη μεθοδολογία, έπρεπε αρχικά να **εξεταστεί ο τρόπος μετάδοσης** της ροής δεδομένων **στο υπολογιστικό νέφος**. Είτε γίνει μετατροπή και αναμετάδοση μιας LSL, είτε μιας OSC ροής δεδομένων, το κόστος θα είναι μικρό γιατί η χρήση τους γίνεται τοπικά. Αλλά, για λόγους ασφαλείας, η επικοινωνία πρέπει να είναι αμφίδρομη. Μέρος της εξέτασης αυτής και των δοκιμών, πραγματοποιήθηκαν στα πλαίσια του έργου (ΚΕΕΠΗ | Epirus XR Center) και αφορούν τη χρήση τεχνολογιών όπως:

- **WebSockets** (TCP-based μόνο. Αμφίδρομη επικοινωνία.)
- **Sockets** (UDP-based ή TCP-based, εξαρτάται από την υλοποίηση. Αμφίδρομη επικοινωνία.)
- **WebRTC** (UDP-based ή TCP-based, εξαρτάται από το ίδιο. Αμφίδρομη επικοινωνία.)
- HTTP Long Polling (HTTP/TCP-based μόνο.)
- Webhooks (HTTP/TCP-based μόνο.)

Η **ακεραιότητα των δεδομένων** είναι **ζωτικής σημασίας** για τα **EEG** δεδομένα επειδή επηρεάζει άμεσα την **ακρίβεια** και την **αξιοπιστία** των καταγεγραμμένων σημάτων εγκεφαλικών κυμάτων.

Εφόσον, είναι **γνωστός** ο **ρυθμός δειγματοληψίας** της ροής δεδομένων που στέλνει η EEG συσκευή, μπορούμε να δώσουμε **προτεραιότητα** στην **ακεραιότητα** των δεδομένων, **έναντι** της **ταχύτητας μετάδοσης**.

Επομένως, συγκρίνοντας τα πρωτόκολλα TCP και UDP, θα μπορούσαμε να πούμε ότι ένα **TCP-based πρωτόκολλο** θα είναι **ιδανικότερο** για αυτή την εφαρμογή καθώς διασφαλίζει την ακριβή και διατεταγμένη παράδοση των πακέτων δεδομένων. Οι μηχανισμοί επιβεβαίωσης και αναμετάδοσης του TCP, εγγυώνται ότι τα δεδομένα λαμβάνονται σωστά, ελαχιστοποιώντας τον κίνδυνο απώλειας ή καταστροφής δεδομένων κατά τη μετάδοση. Ωστόσο, μπορεί να υπάρξει ελαφρώς υψηλότερη καθυστέρηση λόγω των μηχανισμών που αναφέρθηκαν παραπάνω.

*Πίνακας 1 Σύγκριση πρωτοκόλλων αμφίδρομης επικοινωνίας (TCP-based μόνο) ως προς την υλοποίηση.*

| Μέθοδος | Υλοποίηση | Απώλειες Πακέτων | Καθυστερήσεις | Κατάλληλο για πραγματικό χρόνο |
|---|---|---|---|---|
| **WebSockets** | Δημιουργία ενός WebSocket server εντός της εφαρμογής νέφους και μίας εφαρμογής πελάτη. | Ελάχιστες | Μικρές | Πολύ Καλό |
| **Sockets** | Δημιουργία ενός TCP socket server εντός της εφαρμογής νέφους και μίας εφαρμογής πελάτη. | Ελάχιστες | Μέτριες | Καλό |
| **WebRTC** | Δημιουργία signaling server και στις δύο πλευρές. | Καλή διαχείριση | Μικρές | Εξαιρετικό |

*Πίνακας 2 Σύγκριση πρωτοκόλλων αμφίδρομης επικοινωνίας (TCP-based μόνο) ως προς την χρήση.*

| Μέθοδος | Κύρια Χρήση | Παραδείγματα Χρήσης | Servers | Browsers |
|---|---|---|---|---|
| WebSockets | Full-duplex επικοινωνία μέσω μίας TCP σύνδεσης. Κατάλληλα για δημιουργία καναλιών επικοινωνίας σε πραγματικό χρόνο μεταξύ web browser και server. | Εφαρμογές συνομιλίας (chat) σε πραγματικό χρόνο, παιχνίδια για πολλούς παίκτες, εφαρμογές αμφίδρομης επικοινωνίας χαμηλής καθυστέρησης. | Ναι | Ναι |
| Sockets | Κατάλληλο για τη δημιουργία αξιόπιστης, διατεταγμένης και ελεγχόμενης μετάδοσης δεδομένων μεταξύ δύο συσκευών σε ένα δίκτυο. | Μεταφορές αρχείων, εργαλεία απομακρυσμένης διαχείρισης, εφαρμογές όπου είναι απαραίτητος ο έλεγχος της επικοινωνίας. | Ναι | Όχι |
| WebRTC | Peer-to-peer επικοινωνία σε πραγματικό χρόνο (ήχος, βίντεο, δεδομένα) απευθείας μεταξύ προγραμμάτων περιήγησης. | Τηλεδιάσκεψη, φωνητικές κλήσεις, ζωντανή ροή, διαδικτυακά παιχνίδια, επικοινωνία από browser σε browser. | Ναι (ως διαχειριστής) | Ναι |

**Και τα τρία πρωτόκολλα φαίνονται να είναι ιδανικοί υποψήφιοι** για αυτή την υλοποίηση σύμφωνα με τις καθυστερήσεις, τις απώλειες πακέτων και την καταλληλότητά τους **για μετάδοση δεδομένων σε πραγματικό χρόνο**.

Ωστόσο, για λόγους ευκολίας και ασφάλειας, θα ήταν **ιδανικό** το επιλεγμένο **πρωτόκολλο** να κάνει **χρήση HTTP** (είτε κατά την εκκίνηση της επικοινωνίας, είτε κατά τη διάρκειά της) **ώστε να είναι εφικτή η αυθεντικοποίηση** του χρήστη **από την ίδια την εφαρμογή στο υπολογιστικό νέφος.**

Στα **WebSockets** και το **WebRTC** μπορεί να γίνει η **αυθεντικοποίηση με παρόμοιο τρόπο** κατά την εκκίνηση της επικοινωνίας με χρήση HTTP handshake και JSON Web Tokens (JWT), ενώ, δυστυχώς τα TCP Sockets δεν παρέχουν αυτή τη δυνατότητα.

Η **υλοποίηση** ενός **WebSocket server** και η διαχείρισή του εντός μιας Full-Stack εφαρμογής νέφους είναι **ευκολότερη.**

Εξίσου, η **υλοποίηση** μιας **εφαρμογής πελάτη (client)** με χρήση **WebSocket** είναι **ευκολότερη μιας και υποστηρίζεται από servers και browsers** μέσω των **ίδιων βιβλιοθηκών.**

Για το λόγο αυτό, επιλέχτηκαν τα WebSockets για τις υλοποιήσεις αυτού του πειράματος.

Ενώ, για λόγους εξοικείωσης και ευκολίας από τη χρήση μιας γλώσσας σε ολόκληρο το σύνολο «stack» των εφαρμογών, επιλέχθηκε η γλώσσα προγραμματισμού JavaScript.

## 2.2 Δοκιμές

Για την **επιβεβαίωση** της **λειτουργικότητας** των **WebSocket** για αυτή την εφαρμογή, χρειάστηκε το όλο σύστημα αν αποδομηθεί σε υπο-προβλήματα και να δημιουργηθούν μερικά δοκιμαστικά δείγματα. Όπως:

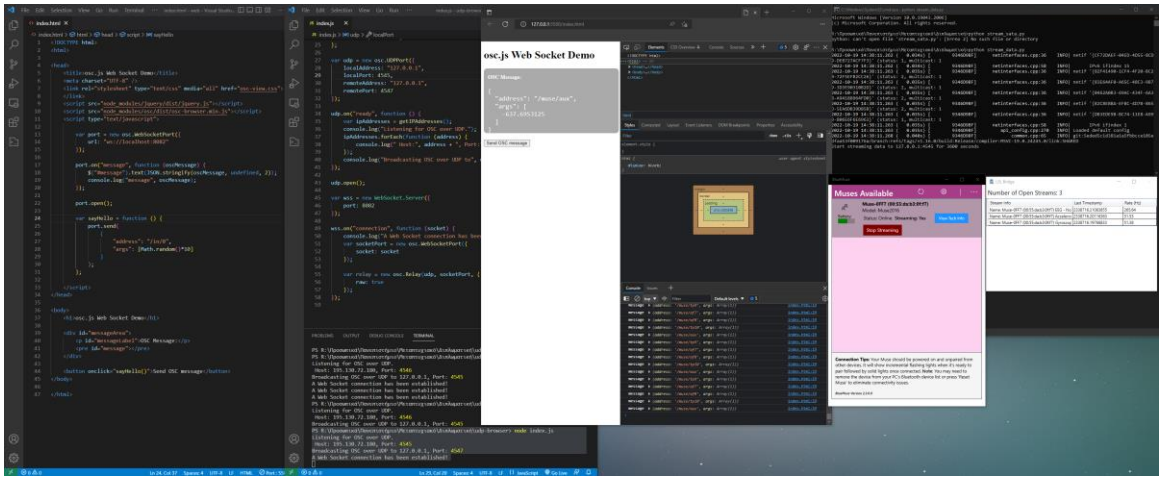
1. **Ανάγνωση τοπικής ροής από LSL ή OSC.**
2. **Μετατροπή και αποστολή δεδομένων μέσω WebSocket.**
3. **Λήψη WebSocket και μετατροπή σε ροή LSL ή OSC.**

Έπειτα, χρειάστηκε να βρεθεί και να κατανοηθεί ο τρόπος λειτουργίας αυτών των δύο πρωτοκόλλων και τέλος να γίνει **αναζήτηση** για τις **κατάλληλες βιβλιοθήκες** ή αλλιώς «**wrappers**» για αυτά ώστε να υλοποιηθούν λύσεις για τα υπο-προβλήματα αυτά πιο εύκολα.

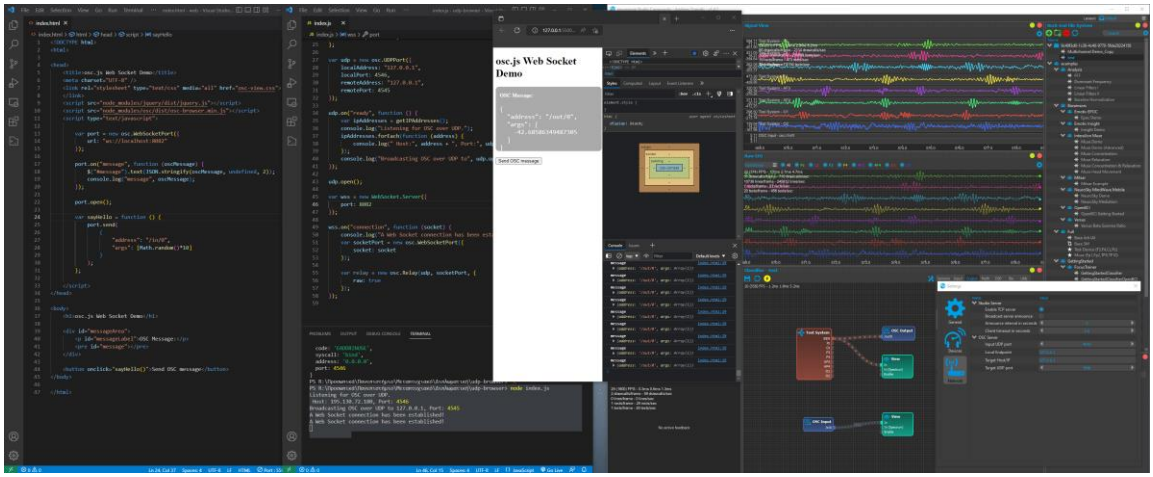
Δυστυχώς, **δεν βρέθηκαν βιβλιοθήκες ή «wrappers»** για το πρωτόκολλο **LSL** σε γλώσσα προγραμματισμού **JavaScript**. Βρέθηκαν παλιές προσπάθειες (2018) για τη δημιουργία ενός «wrapper» από τη κοινότητα ο οποίος είναι ακόμη σε μη λειτουργικό στάδιο, ενώ επίσης, έγινε **προσπάθεια** για τη **δημιουργία** ενός **νέου δίχως επιτυχία**. Η διαδικασία ήταν πολύ χρονοβόρα και οι οδηγίες σύνταξης του πρωτοκόλλου δύσκολες να ακολουθηθούν ή ελλείπεις.

Έπειτα, έγινε εστίαση στο **«γεφύρωμα»** του **OSC** με το **WebSocket** για το οποίο **υπήρχαν** μερικές **βιβλιοθήκες** και η διαδικασία ήταν σχετικά απλή εφόσον το OSC κάνει χρήση του UDP ή TCP για τη μεταφορά των δεδομένων. Πιο συγκεκριμένα, σε χρήση από EEG συσκευές, γίνεται χρήση UDP.

Έτσι, στις πρώτες δοκιμές, έγινε χρήση των βιβλιοθηκών **osc.js** από Colin Clark για Node.js και Browser για την μετατροπή των πακέτων δεδομένων σε JSON αντικείμενα και την αποστολή τους, και **ws** του Node.js για εσωτερική χρήση από το ίδιο για λήψη.




*Εικόνα 1 Στιγμιότυπο δοκιμής Α.*




*Εικόνα 2 Στιγμιότυπο δοκιμής Β.*

## 2.3 Περιγραφή συστήματος

Εφόσον οι **δοκιμές μετατροπής** της **OSC/UDP** ροής δεδομένων **σε WebSockets** πραγματοποιήθηκαν **με επιτυχία**, το **επόμενο βήμα** ήταν να γίνει η συλλογή των απαιτήσεων (επιχειρησιακές διαδικασίες, επιχειρησιακοί κανόνες, λειτουργικές και μη απαιτήσεις, λεξικό δεδομένων και περιπτώσεις χρήσεις) και ο **σχεδιασμός** της **αρχιτεκτονικής** του συστήματος.

Όπως αναφέρθηκε στην αρχή της παρουσίασης, το ιδανικό σύστημα θα πρέπει να αποτελείται από μια εφαρμογή στο υπολογιστικό νέφος και έναν two-way communication client, με την εφαρμογή στο νέφος να λειτουργεί ως μεσάζων για τους clients και ως διαχειριστής των EEG συνεδριών.

Συνοπτικά, ο χρήστης θα πρέπει μέσω της εφαρμογής νέφους να μπορεί:

- να **ορίζει EEG συνεδρίες** μέσω του λογαριασμού του,
- να **αποκτά κλειδιά αυθεντικοποίησης** για τον ίδιο και τον πελάτη του και
- να **αποθηκεύει** τις **ροές δεδομένων** από τις συνεδρίες **στο νέφος** για μελλοντική χρήση/προσομοίωση αναμετάδοσης.

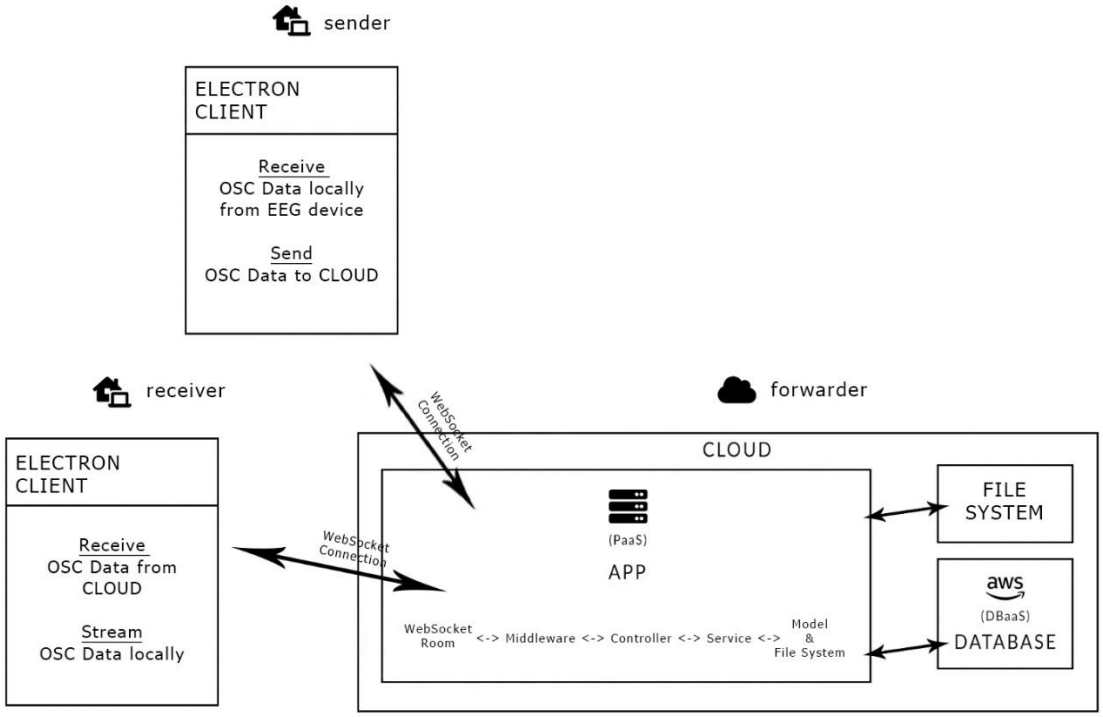
Ενώ, μέσω του client θα πρέπει να μπορούν και οι δύο πλευρές:

- να **ορίζουν** τα **κλειδιά αυθεντικοποίησης** της επικοινωνίας,
- να **ορίζουν** τα **στοιχεία** (διεύθυνση IP και θύρα) της **εφαρμογής** στο νέφος,
- να **ορίζουν** τον **ρυθμό δειγματοληψίας**,
- να **αποθηκεύουν** και να **αναμεταδίδουν ροές δεδομένων τοπικά** στον Η/Υ τους μέσω OSC/UDP,
- να **λαμβάνουν ροές δεδομένων** σε **πραγματικό χρόνο τοπικά** μέσω OSC/UDP,
- να τις **προωθούν μέσω WebSocket** προς την εφαρμογή νέφους και το ανάποδο.

## 2.4 Αρχιτεκτονική συστήματος

Ουσιαστικά, στο εσωτερικό της, η **εφαρμογή νέφους αποτελείται από** ένα **WebSocket server** και έναν **HTTP server** ο οποίος είναι υλοποιημένος με **MVC αρχιτεκτονική**, παρέχει **REST-API** για τις λειτουργίες του και είναι **υπεύθυνος για** το **Front-End** καθώς και για την **διαχείριση** του **WebSocket server**.

Μέσω του διαχειριστικού στο νέφος, όταν ο χρήστης **προγραμματίζει** μια **συνεδρία**, τότε **δημιουργείται** αυτόματα και το αντίστοιχο **WebSocket Room** στο οποίο συνδέονται οι δύο πλευρές.




*Εικόνα 3 Αρχιτεκτονική συστήματος*

## 2.5 Εργαλεία

Για την ανάπτυξη των εφαρμογών χρησιμοποιήθηκαν τα παρακάτω εργαλεία:

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

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

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

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

**Google Chrome** (Wikipedia, χ.χ.) - Είναι πρόγραμμα περιήγησης στο Διαδίκτυο που αναπτύσσεται από την Google και χρησιμοποιεί τη μηχανή απεικόνισης WebKit. Χρησιμοποιήθηκε ως περιηγητής αλλά και για τα εργαλεία ανάπτυξης που προσφέρει.

**Chrome DevTools** (Google, χ.χ.) – Είναι ένα σύνολο εργαλείων για προγραμματιστές διαδικτύου, ενσωματωμένα στον περιηγητή Google Chrome. Χρησιμοποιήθηκε για την εύρεση και επίλυση σφαλμάτων του Front-end

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

**HeidiSQL** (Wikipedia, χ.χ.) – Είναι ένα δωρεάν εργαλείο ανοιχτού κώδικα για τη διαχείριση βάσεων δεδομένων MySQL. Χρησιμοποιήθηκε για την τελική δημοσίευση του σχήματος της βάσης δεδομένων.

# 3 Υλοποιήσεις

## 3.1 Εφαρμογή νέφους (server)

### 3.1.1 Απαιτήσεις

#### 3.1.1.1 Επιχειρησιακές διαδικασίες

Η καταγραφή των λειτουργικών διαδικασιών είναι ο κατάλληλος τρόπος για να βρεθούν οι λειτουργικές απαιτήσεις της εφαρμογής και οι περιπτώσεις χρήσης. Αυτές οι λειτουργικές διαδικασίες, που είναι επίσης γνωστές ως επιχειρησιακές διαδικασίες, βρίσκονται στα επόμενα υποκεφάλαια. Οι παρακάτω επιχειρησιακές διαδικασίες έχουν κριθεί απαραίτητες για την εφαρμογή:

**Αυθεντικοποίηση**

- Σύνδεση χρήστη
- Αποσύνδεση χρήστη
- Εγγραφή χρήστη

**Χρήστες**

- Εμφάνιση διαχειριστικού περιβάλλοντος
- Εμφάνιση προφίλ χρήστη
- Επεξεργασία ρυθμίσεων προφίλ χρήστη

**Μετρήσεις**

- Προγραμματισμός νέας μέτρησης χρήστη και δημιουργία κλειδιών API
- Εμφάνιση καταλόγου μετρήσεων χρήστη
- Διαγραφή μέτρησης χρήστη
- Επεξεργασία στοιχείων μέτρησης χρήστη

#### 3.1.1.2 Επιχειρησιακοί κανόνες

Συντομογραφία br: business-rule

*Πίνακας 3 Επιχειρησιακοί κανόνες*

| **Ταυτότητα / Όνομα** | **Περιγραφή** |
|---|---|
| br.usage | Η εφαρμογή επιτρέπει την χρήση της από πολλούς χρήστες και τις συσκευές των πελατών τους. |
| br.measurement | Κάθε μέτρηση αντιστοιχεί σε ένα μόνο χρήστη και τον πελάτη του. |
| br.measurement.apikey | Η εφαρμογή παρέχει εξουσιοδότηση σε κάθε χρήστη και τον πελάτη του για την αποστολή δεδομένων σε αυτή με ένα κλειδί για τον καθένα αντίστοιχα. Αυτά τα δύο κλειδιά αντιστοιχούν σε μία μόνο μέτρηση. |
| br.measurement.file | Κάθε μέτρηση αντιστοιχεί σε ένα αρχείο στον διακομιστή το οποίο περιέχει όλα τα δεδομένα που έχουν σταλθεί από τη συσκευή του πελάτη του χρήστη. Πρόσβαση στο αρχείο έχει μόνο ο χρήστης με την ολοκλήρωση της αποστολής των δεδομένων από τον πελάτη του. |

#### 3.1.1.3 Μη λειτουργικές απαιτήσεις

Συντομογραφία nfr: non-functional requirements

*Πίνακας 4 Μη λειτουργικές απαιτήσεις*

| **Ταυτότητα / Όνομα** | **Περιγραφή** |
|---|---|
| nfr.design.responsive | Οι διεπαφές της εφαρμογής θα πρέπει να εμφανίζονται σωστά σε κινητά, tablet και υπολογιστές με διάφορες διαστάσεις οθονών. |
| nfr.development.db | Η εφαρμογή θα έχει βάση δεδομένων MySQL. |
| nfr.development.language | Η εφαρμογή θα αναπτυχθεί σε JavaScript. |
| nfr.development.server | Ο διακομιστής της εφαρμογής θα υλοποιηθεί με Express.js. |
| nfr.rendering | Η εφαρμογή θα κάνει Server-Side |

| | |
|---|---|
| | Rendering. |
| nfr.development.view | Οι προβολές τις εφαρμογής θα αναπτυχθούν με χρήση Handlebars.js. |
| nfr.development.view.engine | Τα view engine της εφαρμογής θα είναι το HBS για Express.js. |
| nfr.communication | Ο διακομιστής θα δέχεται μηνύματα από συσκευές με WebSockets. |
| nfr.userinterface | Το ταμπλό διαχείρισης θα υλοποιηθεί με χρήση πρότυπου AdminLTE. |

#### 3.1.1.4 Λειτουργικές απαιτήσεις

Συντομογραφία fr: functional requirements

*Πίνακας 5 Λειτουργικές απαιτήσεις*

| **Ταυτότητα / Όνομα** | **Περιγραφή** |
|---|---|
| fr.authentication.signup | Η εφαρμογή θα επιτρέπει στον μη εγγεγραμμένο χρήστη να εγγραφτεί. |
| fr.authentication.sign | Η εφαρμογή θα επιτρέπει στον μη συνδεδεμένο χρήστη να συνδεθεί. |
| fr.authentication.logout | Η εφαρμογή θα επιτρέπει στον συνδεδεμένο χρήστη να αποσυνδεθεί. |
| fr.user.profile | Η εφαρμογή θα επιτρέπει στον συνδεδεμένο χρήστη να δει τα προσωπικά του στοιχεία. |
| fr.user.profile.settings | Η εφαρμογή θα επιτρέπει στον συνδεδεμένο χρήστη να επεξεργαστεί τα προσωπικά του στοιχεία. |
| fr.user.session | Η συνεδρία θα περιλαμβάνει τις εξής λειτουργίες:<br>Αρχικά:<br>• Σύνδεση<br>Στη συνέχεια:<br>• Εμφάνιση ταμπλό χρήση<br>Η συνεδρία θα τερματίζεται εφόσον ο χρήστης επιλέξει αποσύνδεση. |
| fr.user.measurement.displayAll | Όταν ο συνδεδεμένος χρήστης επιλέξει την εμφάνιση του καταλόγου μετρήσεων, η εφαρμογή θα εμφανίζει όλες τις μετρήσεις του χρήστη. |
| fr.user.measurement.add | Η εφαρμογή θα επιτρέπει στον συνδεδεμένο χρήστη να προγραμματίσει νέες μετρήσεις, να τους αναθέσει στοιχεία όπως όνομα, περιγραφή, διάρκεια και ρυθμό δειγματοληψίας. |
| fr.user.measurement.delete | Η εφαρμογή θα επιτρέπει στον συνδεδεμένο |

| | |
|---|---|
| | χρήστη να διαγράψει μετρήσεις του. |
| fr.user.measurements.key.create | Εξουσιοδοτημένες συσκευές θα μπορούν να στείλουν νέες μετρήσεις. Η εφαρμογή θα δημιουργεί δύο κλειδιά API και θα τα εμφανίζει στον χρήστη όταν ο χρήστης προγραμματίζει μια νέα μέτρηση. |
| fr.user.measurements.session.create | Η εφαρμογή θα ενημερώνει τον WebSocket server και θα δημιουργεί ένα νέο room id για κάθε μέτρηση που προγραμματίζει ο χρήστης. |
| fr.user.measurements.createFile | Κατά τον προγραμματισμό μιας μέτρησης, θα δημιουργείται ένα αντικείμενο μέτρησης στη βάση δεδομένων για αυτή τη μέτρηση με όλα τα απαραίτητα στοιχεία της. Η ροή δεδομένων όμως της μέτρησης δε θα αποθηκεύεται στη βάση δεδομένων αλλά σε ένα αρχείο εντός του συστήματος (file system) σύμφωνα με τον μοναδικό κωδικό αναγνώρισης που της αντιστοιχεί. |

Επιπλέον, έχουν υλοποιηθεί οι λειτουργικές απαιτήσεις για τους χρήστες με ρόλο/δικαιώματα διαχειριστή, αλλά δεν θα αναλυθούν περαιτέρω σε αυτήν την εργασία.

#### 3.1.1.5 Λεξικό δεδομένων

*Πίνακας 6 Λεξικό δεδομένων*

| **Δεδομένο** | **Περιγραφή** | **Συνιστώσες ή τύπος** | **Μήκος** | **Τιμές** |
|---|---|---|---|---|
| Δεδομένα σύνδεσης | | = username<br>+ password | | |
| username | Username χρήστη | string | 32 | |
| password | Κωδικός χρήστη | string | 60 | |

| | | | | |
|---|---|---|---|---|
| Δεδομένα εγγραφής | | = username<br>+ email<br>+ password | | |
| email | Email χρήστη | string | 100 | |
| Δεδομένα προφίλ χρήστη | | = userid<br>+ username<br>+ userrole<br>+ email<br>+ firstname<br>+ lastname<br>+ password | | |
| user id | Μοναδικός κωδικός αναγνώρισης χρήστη | integer | 11 | auto-increment |
| user role | Ρόλος χρήστη | string | 32 | |
| firstname | Όνομα χρήστη | string | 60 | |
| lastname | Επώνυμο χρήστη | string | 60 | |
| Δεδομένα μέτρησης | | = username<br>+ measurement id<br>+ measurement name<br>+ measurement description<br>+ created date<br>+ updated date<br>+ last connection date<br>+ status<br>+ sample rate<br>+ schedule date<br>+ duration<br>+ token sender<br>+ token receiver | | |
| measuremen id | ID μέτρησης | uuid | | UUIDv4 |
| measurement name | Όνομα μέτρησης | string | 64 | |
| measurement description | Περιγραφή μέτρησης | string | 200 | |
| created date | Τύπος συσκευής | date | | current timestamp |
| updated date | ID κλειδιού API | date | | |
| last connection | Τελευταία σύνδεση στη μέτρηση | date | | |

| | | | | |
|---|---|---|---|---|
| date | | | | |
| status | Κατάσταση μέτρησης | tinyint | | 0 ή 1 |
| sample rate | Ρυθμός δειγματοληψίας μέτρησης | float | | 0 |
| schedule date | Προγραμματισμένη ημερομηνία και ώρα μέτρησης | date | | |
| duration | Διάρκεια μέτρησης (σε λεπτά) | integer | | 1 μέχρι 60 |
| token sender | Κλειδί αποστολέα (πελάτη) | string | 512 | |
| token receiver | Κλειδί παραλήπτη (χρήστη) | string | 512 | |

#### 3.1.1.6 Περιπτώσεις χρήσεις (use cases)

##### 3.1.1.6.1 Εγγραφή χρήστη

**Περίπτωση χρήσης:** Εγγραφή χρήστη

**Ταυτότητα:** uc.authentication.signup

**Δράστης:** Χρήστης

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

**Βασική ροή:**

Β0α: Ο δράστης ζητά να κάνει εγγραφή.

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

Β1α: Ο δράστης δίνει τα στοιχεία του.

Β1β: Η εφαρμογή

Β1β1: ελέγχει τα στοιχεία και εφόσον είναι έγκυρα

Β1β2: τα καταχωρεί και

Β1β3: τον συνδέει.

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

**Εναλλακτικές ροές:**

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

#### 3.1.1.6.2 Σύνδεση χρήστη

**Περίπτωση χρήσης:** Σύνδεση χρήστη

**Ταυτότητα:** uc.authentication.sign

**Δράστης:** Χρήστης

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

**Βασική ροή:**

Β0α: Ο δράστης ζητά να συνδεθεί.

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

Β1α: Ο δράστης δίνει τα στοιχεία του.

Β1β: Η εφαρμογή

Β1β1: ελέγχει τα στοιχεία και εφόσον είναι έγκυρα

Β1β2: τον συνδέει.

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

**Εναλλακτικές ροές:**

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

#### 3.1.1.6.3 Αποσύνδεση χρήστη

**Περίπτωση χρήσης:** Αποσύνδεση χρήστη

**Ταυτότητα:** uc.authentication.logout

**Δράστης:** Χρήστης

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

**Βασική ροή:**

Β0α: Ο δράστης ζητά να αποσυνδεθεί.

Β0β: Η εφαρμογή τον αποσυνδέει.

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

**Εναλλακτικές ροές:** καμία

#### 3.1.1.6.4 Εμφάνιση του προφίλ χρήστη

**Περίπτωση χρήσης:** Εμφάνιση του προφίλ χρήστη

**Ταυτότητα:** uc.user.profile

**Δράστης:** Χρήστης

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

**Βασική ροή:**

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

Β0β: Η εφαρμογή εμφανίζει το προφίλ

**Εναλλακτικές ροές:** καμία.

#### 3.1.1.6.5 Επεξεργασία του προφίλ χρήστη

**Περίπτωση χρήσης:** Επεξεργασία του προφίλ χρήστη

**Ταυτότητα:** uc.user.profile.settings

**Δράστης:** Χρήστης

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

**Βασική ροή:**

Β0α: Ο δράστης ζητά να επεξεργαστεί τα προσωπικά του στοιχεία

Β0β: Η εφαρμογή

Β0β1: εμφανίζει τα υπάρχοντα στοιχεία του χρήστη στα πεδία και

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

Β1α: Ο δράστης αλλάζει τα στοιχεία και τα καταχωρεί.

Β1β: Η εφαρμογή

Β1β1: ελέγχει τα στοιχεία και εφόσον είναι έγκυρα

Β1β2: καταχωρεί τα νέα στοιχεία και

Β1β3: εμφανίζει το προφίλ του.

**Εναλλακτικές ροές:**

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

#### 3.1.1.6.6 Εμφάνιση του ταμπλό διαχείρισης

**Περίπτωση χρήσης:** Εμφάνιση του ταμπλό διαχείρισης

**Ταυτότητα:** uc.admin.index

**Δράστης:** Χρήστης

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

**Βασική ροή:**

Β0α: Ο δράστης ζητά να δει το ταμπλό διαχείρισης.

Β0β: Η εφαρμογή εμφανίζει ο ταμπλό διαχείρισης

Β0β1: και εμφανίζει τα τελευταία δεδομένα.

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

**Εναλλακτικές ροές:** καμία.

#### 3.1.1.6.7 Προγραμματισμός νέας μέτρησης

**Περίπτωση χρήσης:** Προγραμματισμός νέας μέτρησης

**Ταυτότητα:** uc.measurement.add

**Δράστης:** Χρήστης

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

**Βασική ροή:**

Β0α: Ο δράστης ζητά να προγραμματίσει μια νέα μέτρηση.

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

Β1α: Ο δράστης δίνει τα στοιχεία της μέτρησης.

Β1β: Η εφαρμογή

Β1β1: ελέγχει τα δεδομένα και εφόσον είναι έγκυρα

Β1β2: καταχωρεί τη νέα μέτρηση

Β1β3: εμφανίζει τα στοιχεία και τα κλειδιά API της μέτρησης.

**Εναλλακτικές ροές:**

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

#### 3.1.1.6.8 Εμφάνιση του καταλόγου μετρήσεων

**Περίπτωση χρήσης:** Εμφάνιση του καταλόγου μετρήσεων

**Ταυτότητα:** uc.measurments.displayAll

**Δράστης:** Χρήστης

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

**Βασική ροή:**

Β0α: Ο δράστης ζητά να δει το κατάλογο μετρήσεων.

Β0β: Η εφαρμογή εμφανίζει το κατάλογο μετρήσεων.

**Εναλλακτικές ροές:** καμία.

#### 3.1.1.6.9 Διαγραφή μέτρησης

**Περίπτωση χρήσης:** Διαγραφή μέτρησης

**Ταυτότητα:** uc.measurment.delete

**Δράστης:** Χρήστης

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

**Βασική ροή:**

Β0α: Ο δράστης ζητά να διαγράψει μια μέτρηση.

Β0β: Η εφαρμογή

Β0β1: διαγράφει τη μέτρηση και

Β0β2: εμφανίζει τον κατάλογο των μετρήσεων

**Εναλλακτικές ροές:** καμία.

#### 3.1.1.6.10 Εκτέλεση προγραμματισμένης μέτρησης

**Περίπτωση χρήσης:** Εκτέλεση προγραμματισμένης μέτρησης

**Ταυτότητα:** uc.measurement.add

**Δράστης:** Συσκευή χρήστη και συσκευή πελάτη

**Αρχικές συνθήκες:**

Α0: Έχει εγγραφτεί ο χρήστης και

Α0β1: έχει προγραμματίσει τη μέτρηση και

Α0β2: έχει αντιγράψει το πρώτο κλειδί API και

Α0β3: έχει δώσει το δεύτερο κλειδί API στον πελάτη του.

Α1: Η συσκευή του χρήστη έχει πρόσβαση στο διαδίκτυο.

Α1α: Ο χρήστης έχει αντιγράψει το πρώτο κλειδί API στον client της συσκευής του.

Α1β: Ο χρήστης έχει θέσει τον client σε λειτουργία λήψης μηνυμάτων.

Α2: Η συσκευή του πελάτη έχει πρόσβαση στο διαδίκτυο.

Α3: Ο πελάτης έχει λάβει το δεύτερο κλειδί από τον χρήστη.

**Βασική ροή:**

Β1α: Ο πελάτης αντιγράφει το δεύτερο κλειδί API στον client της συσκευής του.

Β1β: Ο πελάτης θέτει τον client σε λειτουργία αποστολής μηνυμάτων

Β1γ: Η εφαρμογή

Β1γ1: ελέγχει το κλειδί API του πελάτη και αν είναι έγκυρο,

Β1γ2: δέχεται τα μηνύματα από τον client του πελάτη και

Β1γ3: καταγράφει σε αρχείο όλα τα μηνύματα.

**Εναλλακτικές ροές:**

Ε1: Αν στο βήμα Β1γ1 το κλειδί API δεν είναι έγκυρο, τότε η εφαρμογή επιστρέφει μήνυμα λάθους.

#### 3.1.1.6.11 Λήψη ολοκληρωμένης μέτρησης

**Περίπτωση χρήσης:** Λήψη ολοκληρωμένης μέτρησης

**Ταυτότητα:** uc.measurement.download

**Δράστης:** Χρήστης

**Αρχικές συνθήκες:**

Α0: Η μέτρηση έχει ολοκληρωθεί.

Α1: Η κατάσταση του χρήστη είναι «συνδεδεμένος».

Α2α: Ο δράστης έχει ζητήσει να δει κατάλογο μετρήσεων.

Α2β: Η εφαρμογή έχει εμφανίσει τον κατάλογο μετρήσεων.

**Βασική ροή:**

Β0: Ο δράστης επιλέγει την μέτρηση που επιθυμεί και ζητά να τη κατεβάσει.

Β1: Η εφαρμογή επιστρέψει το επιλεγμένο αρχείο.

**Εναλλακτικές ροές:** καμία.

#### 3.1.1.6.12 Περιπτώσεις χρήσης διαχειριστή

Οι περιπτώσεις χρήσης του διαχειριστή είναι παρόμοιες με αυτές που αναφέρθηκαν παραπάνω για τον χρήστη, έτσι δεν θα γίνει περαιτέρω ανάλυση σε αυτήν την εργασία. Η κύρια διαφορά είναι ότι ο διαχειριστής έχει πρόσβαση σε καταλόγους όλων των πόρων όλων των χρηστών και, φυσικά, έχει όλα τα δικαιώματα (επεξεργασίας/διαγραφής).

### 3.1.2 Αρχιτεκτονική

Από τις απαιτήσεις προκύπτουν τα μέρη που περιλαμβάνει η εφαρμογή:

- Διεπαφή χρήστη
- Επιχειρησιακή λογική
- Μοντέλο δεδομένων

Τα τρία αυτά μέρη αντιστοιχούν στην αρχιτεκτονική MVC, η οποίο είναι ένα πρότυπο σχεδιασμού λογισμικού που χρησιμοποιείται κυρίως για την ανάπτυξη εφαρμογών λογισμικού με γραφική διεπαφή χρήστη (GUI). Πιο αναλυτικά, στην αρχιτεκτονική MVC, ο κώδικας χωρίζεται σε τρεις βασικές επιμέρους συνιστώσες:

- Μοντέλο (Model) για το μοντέλο δεδομένων. Αναπαριστά τα δεδομένα της εφαρμογής και παρέχει λειτουργίες για την επεξεργασία και την ανάκτησή τους.
- Προβολή (View) για τη διεπαφή χρήστη. Αναπαριστά την εμφάνιση των δεδομένων στον χρήστη, δηλαδή το UI (User Interface).
- Ελεγκτής (Controller) για την επιχειρησιακή λογική. Διαχειρίζεται τις ενέργειες του χρήστη και τις εισόδους στο σύστημα, επιτρέποντας στον χρήστη να επικοινωνεί με το μοντέλο και την προβολή.

Στο πρότυπο αυτό, η εφαρμογή διαιρείται σε τρία διασυνδεδεμένα μέρη, ώστε να διαχωριστεί η παρουσίαση της πληροφορίας στον χρήστη από τη μορφή που έχει αποθηκευτεί στο σύστημα. Επίσης, βοηθά στην αποσύνθεση της εφαρμογής σε μικρότερα, πιο επιμέρους τμήματα, διευκολύνοντας έτσι τη συντήρηση και την ανάπτυξη του κώδικα, κάτι που επιτρέπει στους προγραμματιστές να εργάζονται παράλληλα σε διαφορετικά τμήματα της εφαρμογής, χωρίς να παρεμβαίνει ο ένας στο έργο του άλλου.

Για την **επιχειρησιακή λογική**, προτιμήθηκε η χρήση του αρχιτεκτονικού στυλ **REST**, το οποίο βασίζεται στην αρχιτεκτονική MVC. Ειδικότερα, προτιμήθηκε η υλοποίηση ενός REST-API (Back-end) σε έναν διακομιστή/περιβάλλον Node.js για να διαφοροποιηθεί η επιχειρησιακή λογική από τα υπόλοιπα μέρη. Το πλεονέκτημα από αυτήν την επιλογή είναι η ευκολία συντήρησης του κώδικα, η δυνατότητα χρήσης της

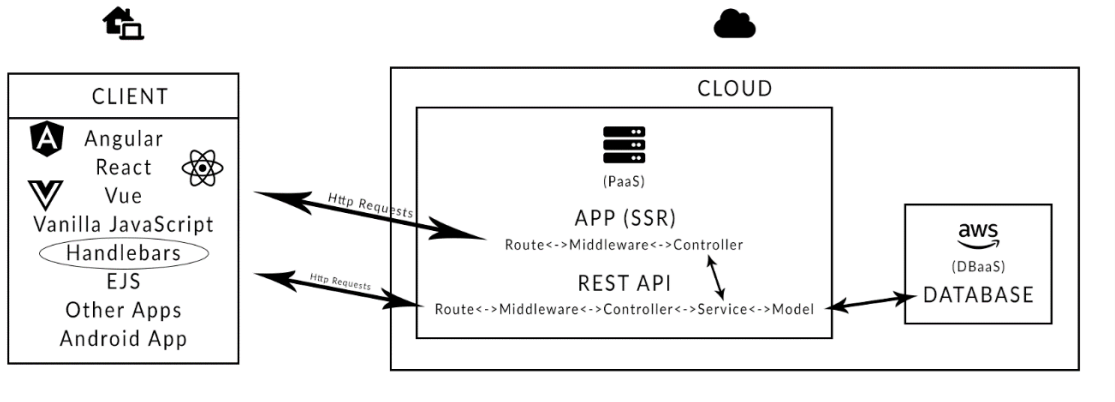
επιχειρησιακής λογικής από άλλα λογισμικά, η δυνατότητα χρήσης πολλών τεχνολογιών για τη διεπαφή χρήστη (Front-end) και η άμεση επικοινωνία των διάφορων συσκευών με την επιχειρησιακή λογική. Επιπλέον, η εγκατάσταση/δημοσίευση του διακομιστή REST-API μπορεί να γίνει ξεχωριστά από τη διεπαφή, βοηθώντας στη μείωση του φόρτου και παρέχοντας τη δυνατότητα αναβάθμισης των πόρων (scaling) του διακομιστή στο μέλλον. Γίνεται περαιτέρω ανάλυση στο επόμενο υποκεφάλαιο.

Για την **άμεση επικοινωνία** μεταξύ των συσκευών των χρηστών για την λήψη των δεδομένων των μετρήσεων, προτιμήθηκε η υλοποίηση ενός **WebSocket Server** με διαχείριση συνεδριών και δωματίων, του οποίου η λειτουργία γίνεται παράλληλα με τον HTTP Server που παρέχει το REST API στους Clients.

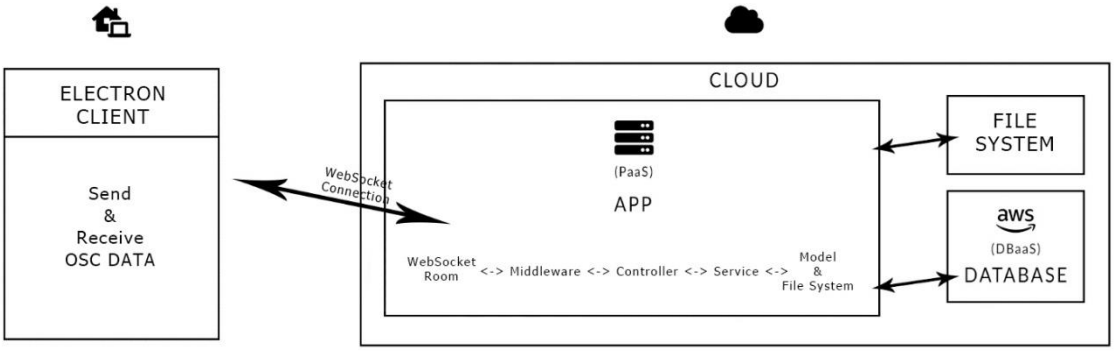
Για το **μοντέλο δεδομένων** χρησιμοποιήθηκε ένας διακομιστής MySQL μιας και είναι ιδανικός για σχεσιακές βάσεις δεδομένων.

Για τη **διεπαφή χρήστη (Front-end),** υπήρχαν πολλές επιλογές. Έπρεπε να γίνει επιλογή μεταξύ Server-Side Rendering (SSR) και Client-Side Rendering (CSR) λαμβάνοντας υπόψη τη χρησιμότητα κάθε τεχνολογίας, τα πλεονεκτήματά τους και το χρόνο ανάπτυξης. Η τάση στη παρούσα εποχή είναι τα Single-Page Apps (SPAs) όπου η σελίδα HTML δημιουργείται δυναμικά στην πλευρά του πελάτη Client, δηλαδή CSR. Τεχνολογίες που χρησιμοποιούνται συχνά για την υλοποίηση SPAs περιλαμβάνουν Angular, React, Vue, Vanilla JavaScript κλπ. Από την άλλη, η υλοποίηση μίας διεπαφής SSR είναι πιο εύκολη και γρήγορη καθώς μπορεί να υλοποιηθεί εντός της ίδιας εφαρμογής και να κάνει άμεση χρήση των πόρων και του API. Οι τεχνολογίες που χρησιμοποιούνται για υλοποίηση SSR με Node.js περιλαμβάνουν Handlebars, JES, Pug και άλλα. Ωστόσο, για λόγους εξοικείωσης, εξοικονόμησης χρόνου και πόρων, για την κατασκευή του προτύπου εμφάνισης, των διαδρομών και των προβολών χρησιμοποιήθηκε κυρίως Handlebars. Για την ανάκτηση δεδομένων χρησιμοποιήθηκε η βιβλιοθήκη Axios στον Client σε JavaScript, όπως γίνεται συνήθως και στις υλοποιήσεις με React. Αποτέλεσμα αυτού είναι ότι η σελίδα αποστέλλεται με το αρχικό πρότυπο στον Client και περιέχει ελάχιστες πληροφορίες σχετικά με τον χρήστη. Τα υπόλοιπα δεδομένα της σελίδας ενημερώνονται μέσω των αιτημάτων που γίνονται στο REST-API από τον πελάτη.

Παρακάτω, ακολουθούν φωτογραφίες από διάφορες πιθανές χρήσης της αρχιτεκτονικής της εφαρμογής.




*Εικόνα 4 Αρχιτεκτονική εφαρμογής νέφους (επικοινωνία με Front-end)*




*Εικόνα 5 Αρχιτεκτονική εφαρμογής νέφους (επικοινωνία με WebSocket Client)*

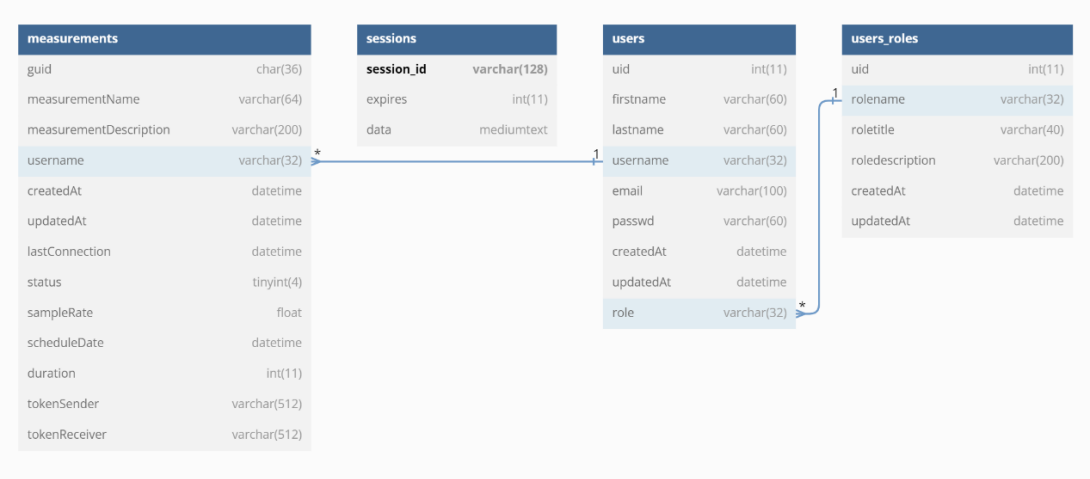
### 3.1.3 Ανάλυση και σχεδιασμός βάσης δεδομένων

Από τις απαιτήσεις προκύπτουν οι παρακάτω **οντότητες** με τα σχετικά δεδομένα τους (δεν αναφέρονται ξανά εδώ). Περαιτέρω ανάλυση της βάσης δεδομένων και των μοντέλων των οντοτήτων γίνεται στο υποκεφάλαιο του Back-end (REST-API):

- Χρήστης
- Ρόλος χρήστη
- Μέτρηση
- Συνεδρία

Και οι **συσχετίσεις** τους αντίστοιχα:

- Κάθε χρήστης ανήκει σε μία συνεδρία. Κάθε συνεδρία έχει ένα χρήστη.
- Κάθε χρήστης έχει ένα ρόλο (user ή admin).
- Κάθε ρόλος ανήκει σε πολλούς χρήστες.
- Κάθε μέτρηση ανήκει σε ένα χρήστη.
- Κάθε χρήστης έχει πολλές μετρήσεις.
- Κάθε μέτρηση έχει δύο κλειδιά API.
- Κάθε κλειδί API ανήκει σε μία μέτρηση.
- Κάθε αρχείο μέτρησης ανήκει σε μία μέτρηση.
- Κάθε μέτρηση έχει ένα αρχείο μέτρησης.




*Εικόνα 6 Σχήμα βάσης δεδομένων(Διάγραμμα EER)*

### 3.1.4 Εργαλεία και τεχνολογίες

Εργαλεία, επίσης γνωστά ως βιβλιοθήκες, εξαρτήσεις, πακέτα ή μονάδες, είναι σύνολα λειτουργιών και μεθόδων που χρησιμοποιούνται σε μια εφαρμογή. Κάποια από τα εργαλεία που χρησιμοποιήθηκαν στην ανάπτυξη του back-end και του front-end είναι τα παρακάτω. Σημείωση: Η αναφορά τους γίνεται εδώ από κοινού καθώς και οι δύο πλευρές υλοποιούνται σε JavaScript και τα περισσότερα εργαλεία είναι συμβατά και με τις δύο.

**Express.js** (Expressjs.com, χ.χ.) – Express.js είναι ένα framework για την πλατφόρμα του Node.js που χρησιμοποιείται για τη δημιουργία εφαρμογών διαδικτύου, κινητών εφαρμογών και APIs. Το Express.js παρέχει μια σειρά από χρήσιμα εργαλεία για την ανάπτυξη της εφαρμογής σας, όπως διαχείριση διαδρομών (routes) και middleware, που μπορούν να χρησιμοποιηθούν για την επεξεργασία των HTTP requests. Η χρήση του Express.js απλοποιεί σημαντικά την ανάπτυξη εφαρμογών σε Node.js, καθώς παρέχει μια σειρά από πρότυπα και εργαλεία για την επικοινωνία με τον πελάτη, τη διαχείριση σφαλμάτων και άλλες λειτουργίες.

**Express-Validator** (Express-Validator, χ.χ.) - Το Express-Validator είναι ένα middleware για το Express.js που χρησιμοποιεί λειτουργίες validator και sanitizer του validator.js μια βιβλιοθήκη που παρέχει λειτουργίες επικύρωσης δεδομένων. Βοηθά στην διασφάλιση ότι τα δεδομένα που λαμβάνονται από την πλευρά του πελάτη είναι στη σωστή μορφή και πληρούν τα αναμενόμενα κριτήρια. Αυτό έχει ως αποτέλεσμα την αποτροπή σφαλμάτων και βελτιώνει τη συνολική ασφάλεια της εφαρμογής. Παρέχει ένα απλό και ευέλικτο API, κάτι που το καθιστά εύκολο στη χρήση και προσαρμογή του.

**Express-Session** (tutorialspoint, χ.χ.) - Το Express-Session είναι ένα middleware για το Express.js που χρησιμοποιείται για τη δημιουργία και διαχείριση συνεδριών (sessions). Παρέχει στους προγραμματιστές ένα απλό τρόπο για να δημιουργήσουν και να διαχειριστούν μοναδικά session IDs για κάθε χρήστη, να αποθηκεύουν δεδομένα συνεδρίας και να ελέγχουν το χρόνο λήξης της συνεδρίας. Επίσης, χρησιμοποιείται συχνά σε εφαρμογές διαδικτύου που απαιτούν πιστοποίηση και εξουσιοδότηση του χρήστη.

**Express-MySQL-Session** (chill117, χ.χ.) – Το Express-MySQL-Session είναι ένα middleware για το Express.js που χρησιμοποιείται για τη διαχείριση των συνεδριών (sessions) των χρηστών με χρήση μιας βάσης δεδομένων MySQL. Βασίζεται στο Express-Session για την αποθήκευση συνεδριών σε βάση δεδομένων MySQL.

**Express-JWT** (express-jwt, χ.χ.) – Το Express-JWT παρέχει στο Express.js ένα middleware για την επικύρωση των αιτημάτων HTTP μέσω κλειδιών JWT (JSON Web Tokens) με χρήση της βιβλιοθήκης jsonwebtoken. Επίσης παρέχει τη δυνατότητα αποθήκευσης των δεδομένων του αποκωδικοποιημένου κλειδιού σε ένα αντικείμενο.

**Sequelize ORM** – Το Object Relational Mapping (ORM) (Kadwill, χ.χ.) είναι μια τεχνική που χαρτογραφεί τα αντικείμενα του λογισμικού σε πίνακες της βάσης δεδομένων. Η Sequelize ORM (Sequelize, χ.χ.) είναι μια δημοφιλής βιβλιοθήκη για το Node.js, η οποία επιτρέπει στους προγραμματιστές να αλληλεπιδρούν με σχεσιακές βάσεις δεδομένων χρησιμοποιώντας αντικείμενα JavaScript. Είναι promise-based, υποστηρίζει σχεσιακές βάσεις δεδομένων και διαλέκτους όπως PostgreSQL, MySQL, MariaDB, SQLite και MSSQL. Μπορεί να χρησιμοποιηθεί ακόμη και για την απευθείας δημιουργία εντολών (queries). Ένα πλεονέκτημα της χρήσης ενός τέτοιου ORM είναι κυρίως η γρήγορη δημιουργία ερωτημάτων χωρίς τον προγραμματιστή να χρειάζεται να γνωρίζει το διάλεκτο της βάσης δεδομένων και δευτερευόντως η προστασία από επιθέσεις SQL Injection.

**JSONWebToken** – Το JSON Web Token, είναι μια βιβλιοθήκη που υλοποιεί λειτουργίες ανοιχτού πρότυπου (RFC 7519) JSON Web Token (JWT) (JWT.IO, χ.χ.). Χρησιμοποιώντας το JWT, οι πληροφορίες του χρήστη αποθηκεύονται σε μια συμβολοσειρά, η οποία είναι γνωστή ως κλειδί JWT Token, και μπορούν να επαληθευτούν επειδή υπογράφονται ψηφιακά και κρυπτογραφούνται με ένα δημόσιο ή ιδιωτικό κλειδί χρησιμοποιώντας RSA, ECDSA ή HMAC. Μαζί με αυτές τις πληροφορίες αποθηκεύονται οι ημερομηνίες δημιουργίας και λήξης, που ορίζουν τη διάρκεια ζωής τους, κάτι που είναι πολύ χρήσιμο για ένα stateless REST API καθώς δεν υπάρχει συνεδρία για να αποθηκευτούν προσωρινά αυτές οι πληροφορίες. Το JWT δημιουργείται από το REST API και αποθηκεύεται στο front-end όπου έχει οριστεί.

**Bcrypt** (Bcrypt, χ.χ.) – Το Bcrypt είναι μια συνάρτηση κατακερματισμού κωδικών πρόσβασης που επιτρέπει την ασφαλή αποθήκευση τους. Χρησιμοποιεί μια τεχνική που

ονομάζεται key stretching, η οποία προσθέτει μια επιπλέον στρώση πολυπλοκότητας στη διαδικασία κατακερματισμού και καθιστά δυσκολότερο για τους χάκερ να μαντέψουν τους κωδικούς πρόσβασης. Το Bcrypt είναι μια δημοφιλής επιλογή για την αποθήκευση κωδικών πρόσβασης σε εφαρμογές ιστού και είναι διαθέσιμο σε πολλές γλώσσες προγραμματισμού.

**Dotenv** (Dotenv, n.d.) – Το Dotenv είναι ένα module για εφαρμογές Node.js χωρίς εξαρτήσεις, που φορτώνει μεταβλητές περιβάλλοντος από ένα αρχείο .env εντός του φακέλου της εφαρμογής. Αυτό διευκολύνει τη διαχείριση ευαίσθητων πληροφοριών, όπως κλειδιά API ή κωδικούς πρόσβασης βάσης δεδομένων, χωρίς να τα αποκαλύπτει στην κώδικα.

**Cors** (Cors, n.d.) – Το Cors είναι ένα middleware του Express.js που επιτρέπει στις εφαρμογές να αποδέχονται αιτήσεις από διαφορετικές προελεύσεις. Αυτό είναι χρήσιμο όταν πρέπει να εκθέτεται το API σε διαφορετικά domains ή subdomains από αυτό της εφαρμογής.

**Body-Parser** (Body-Parser, n.d.) - Το Body-Parser είναι μια βιβλιοθήκη για το Node.js που μεταφέρει τα σώματα (bodies) των εισερχόμενων αιτημάτων σε middleware μέσω της ιδιότητας req.body.

**Cookie-Parser** (Cookie-Parser, χ.χ.) – Το Cookie-Parser είναι μια βιβλιοθήκη για το Node.js που μεταφέρει τις επικεφαλίδες (headers) των Cookie σε middleware μέσω της ιδιότητας req.cookies. Χρησιμοποιείται στο REST-API για την ανάγνωση JWT Token από το Front-end.

**Hbs** (hbs, n.d.) – Το HBS είναι μια βιβλιοθήκη για το Express.js που επιτρέπει τη χρήση του handlebars.js σαν view engine. Χρησιμοποιείται για τη δημιουργία SSR προβολών.

**Passport** (Passport, χ.χ.) – Το Passport είναι ένα middleware συμβατό με το Express.js και χρησιμοποιείται για αυθεντικοποίηση στο Node.js. Παρέχει ένα εύκολο στη χρήση πλαίσιο για την υλοποίηση διαφόρων στρατηγικών αυθεντικοποίησης όπως το OAuth, το OpenID αλλά και για τοπική αυθεντικοποίηση. Παρέχει επίσης υποστήριξη συνεδρίας, καθιστώντας εύκολη τη διαχείριση των συνεδριών χρήστη και της εξουσιοδότησης.

**AdminLTE** (AdminLTE, χ.χ.) – Το AdminLTE είναι ένα δημοφιλές πρότυπο (responsive html template) ανοιχτού κώδικα για τη κατασκευή διαχειριστικών πάνελ που έχει κατασκευαστεί πάνω στο Bootstrap. Παρέχει μια ανταποκρίσιμη και προσαρμόσιμη διεπαφή χρήστη με μια ποικιλία στοιχείων και προσθέτων για τη δημιουργία εφαρμογών ιστού. Περιλαμβάνει διάφορες σελίδες, όπως σελίδες σύνδεσης, εγγραφής, σφάλματος και πίνακα ελέγχου (dashboard), καθώς και πολλά στοιχεία χρήστη, όπως κουμπιά, φόρμες, πίνακες, γραφήματα κλπ.

**Axios** (Axios, χ.χ.) – Το Axios είναι μια βιβλιοθήκη JavaScript για τη διευκόλυνση της αλληλεπίδρασης των client με servers μέσω του πρωτοκόλλου HTTP. Είναι promise based και είναι συμβατό με περιηγητές και Node.js. Χρησιμοποιείται στο Front-end για την ανάκτηση δεδομένων από το REST-API.

**WS** – Το WS είναι μια βιβλιοθήκη για το Node.js που παρέχει μια υλοποίηση του πρωτοκόλλου WebSocket. Παρέχει έναν εύκολο τρόπο για τη δημιουργία τόσο του client όσο και του server.

### 3.1.5 Ανάλυση Back-end (REST-API)

#### 3.1.5.1 Αρχιτεκτονική

REST (Representational State Transfer) (RESTfulAPI.net, χ.χ.) είναι ένα στυλ αρχιτεκτονικής για την ανάπτυξη εφαρμογών ιστού. Τα REST APIs επιτρέπουν στους πελάτες να αλληλεπιδρούν με τον διακομιστή μέσω αιτημάτων HTTP χρησιμοποιώντας τις βασικές μεθόδους HTTP όπως GET, POST, PUT και DELETE. Τα αιτήματα αποστέλλονται σε μια διεύθυνση URL και ο διακομιστής απαντάει με μια απόκριση HTTP που περιέχει τα δεδομένα που ζητήθηκαν. Τα REST APIs είναι ανεξάρτητα από το γραφικό περιβάλλον και μπορούν να χρησιμοποιηθούν σε οποιαδήποτε γλώσσα προγραμματισμού που υποστηρίζει τις βασικές μεθόδους HTTP.

Κάθε πόρος προσδιορίζεται από URIs και καθολικά αναγνωριστικά. Το REST χρησιμοποιεί διάφορες παραστάσεις για να αντιπροσωπεύσει ένα πόρο όπως απλό κείμενο, JSON, XML αλλά το JSON είναι το πιο δημοφιλές. (RESTfulAPI.net, χ.χ.)

**Μέθοδοι HTTP**

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

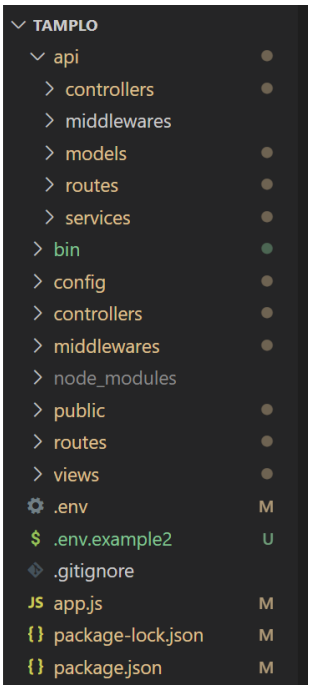
- **GET** – Για την παροχή πρόσβασης σε ένα πόρο μόνο για ανάγνωση.
- **POST** – Για την δημιουργία ενός νέου πόρου.
- **DELETE** – Για την διαγραφή ενός πόρου.
- **PUT** – Για την ενημέρωση ενός υπάρχοντος πόρου ή τη δημιουργία ενός νέου πόρου.

#### 3.1.5.2 Δομικά μέρη

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

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

Επίσης οι βιβλιοθήκες που χρησιμοποιεί βρίσκονται στον φάκελο **node_modules** και παράλληλα μπορεί να δανείζεται υπηρεσίες από «κοινόχρηστα» modules του φακέλου **config**.




*Εικόνα 7 Δομικά μέρη Back-end REST API*

Μια συνήθης πρακτική στη δημιουργία των REST APIs είναι η χρήση των παραπάνω χωρίς υπηρεσίες. Αυτό σημαίνει ότι οι υπηρεσίες ενσωματώνονται στους ελεγκτές. Ωστόσο, αυτό θεωρήθηκε κακή πρακτική σε αυτήν την εφαρμογή, καθώς ο ίδιος κώδικας χρησιμοποιούνταν συχνά από πολλούς ελεγκτές. Η διάκριση των υπηρεσιών από τους ελεγκτές είχε επίσης οφέλη στη δημιουργία ενδιάμεσου λογισμικού και στο Server-Side Rendering (SSR), καθώς και αυτά πλέον μπορούσαν να χρησιμοποιήσουν αυτές τις υπηρεσίες. Ποιο αναλυτικά:

**Κεντρικό αρχείο εφαρμογής:**

- Συνδέει όλα τα τμήματα της εφαρμογής.
- Περιέχει δεδομένα και επιχειρησιακή λογική.

**Διαδρομές (routes):**

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

**Ελεγκτές (controllers):**

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

**Υπηρεσίες (services):**

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

**Μοντέλα (models):**

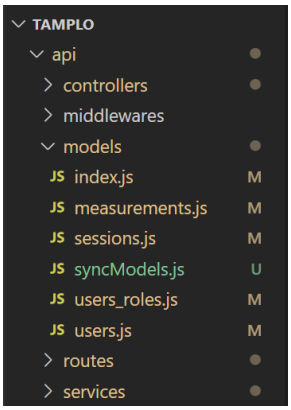
- Δεν περιέχουν επιχειρησιακή λογική (εκτός αν θεωρηθεί αναγκαίο για μια λειτουργία).
- Καθένα αντιστοιχεί σε ένα πίνακα της βάσης δεδομένων.
- Εκτελούν λειτουργίες CRUD στη βάση δεδομένων.
- Δέχονται δεδομένα από τις υπηρεσίες.
- Επιστρέφουν δεδομένα.

**Ενδιάμεσο λογισμικό (middleware):**

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

### 3.1.5.3 Μοντέλα οντοτήτων (models)

Για τη δημιουργία του σχήματος της βάσης δεδομένων χρησιμοποιήθηκε η βιβλιοθήκη Sequelize ORM. Πιο αναλυτικά, αρχικά γίνεται η υλοποίηση των μοντέλων με τη μορφή ανεξάρτητων κλάσεων ES6 CJS module ακολουθώντας τα πρότυπα που αναγνωρίζει η Sequelize ORM. Το καθένα μοντέλο έχει ως όνομα αρχείου το όνομα της οντότητας που αντιστοιχεί και κατάληξη .js όπως φαίνεται παρακάτω. Έπειτα γίνεται είναι η δημιουργία ενός «κεντρικού» index αρχείου το οποίο εισάγει και εξάγει όλα τα μοντέλα αλλά περιέχει επίσης και τις συσχετίσεις τους. Το όφελος από αυτό είναι ότι δε χρειάζεται να εισαχθούν μεμονωμένα τα μοντέλα στον κώδικά όπου αυτά χρειάζονται αλλά εισάγεται το index.js αρχείο και έπειτα καλούνται τα μοντέλα μέσω αυτού. Τέλος, η Sequelize ORM είναι υπεύθυνη για την μετατροπή των μοντέλων αυτών σε κώδικα SQL με τον οποίο δημιουργεί το τελικό σχήμα της βάσης δεδομένων.




*Εικόνα 8 Δημιουργία μοντέλων οντοτήτων με Sequelize ORM*

*Δείγματα κώδικα θα βρείτε στα παραρτήματα Α.1, Α.2, και Α.3.*

#### 3.1.5.4 Διαδρομές (routes)

Όπως αναφέρεται στην αρχιτεκτονική, το REST API λαμβάνει αιτήματα HTTP (χρησιμοποιώντας μεθόδους όπως GET, POST, PUT, DELETE, κλπ.) και στη συνέχεια εκτελεί λειτουργίες CRUD (Δημιουργία, Ανάγνωση, Ενημέρωση, Διαγραφή) στη βάση δεδομένων. Για την τροποποίηση των δεδομένων μιας οντότητας στη βάση δεδομένων, πρέπει να υπάρχουν διαδρομές (routes) που αντιστοιχούν σε αυτές τις λειτουργίες CRUD.

*Πίνακας 7 HTTP Μέθοδοι σε CRUD Λειτουργίες*

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

##### 3.1.5.4.1 Ανάλυση διαδρομών

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

*Πίνακας 8 Διαδρομές Back-end (REST-API)*

| Διαδρομή | Μέθοδος | Περιγραφή |
|---|---|---|
| /measurements | GET | Επιστρέφει λίστα με όλες τις μετρήσεις. |
| /measurements/count | GET | Επιστρέφει τον αριθμό όλων των μετρήσεων. |
| /users | GET | Επιστρέφει λίστα με όλους τους χρήστες. |
| /users/signup | POST | Δημιουργεί νέο χρήστη. |
| /users/login | POST | Επιστρέφει ένα JWT Token με την επιτυχή επαλήθευση των στοιχείων του χρήστη. |
| /users/count | GET | Επιστρέφει τον αριθμό όλων των χρηστών. |
| /users/sessions | GET | Επιστρέφει τον αριθμό |

| | | |
|---|---|---|
| | | όλων των ενεργών συνεδριών των χρηστών. |
| /users/:username | GET | Επιστρέφει τα στοιχεία ενός χρήστη. |
| /users/:username | PUT | Αλλάζει τα στοιχεία του χρήστη. |
| /users/:username | DELETE | Διαγράφει έναν χρήστη. |
| /users/:username/role | PUT | Αλλάζει τον ρόλο του χρήστη. |
| /users/:username/security | PUT | Αλλάζει τον κωδικό πρόσβασης του χρήστη. |
| /users/:username/measurements | GET | Επιστρέφει λίστα με όλες τις μετρήσεις ενός χρήστη. |
| /users/:username/measurements | POST | Δημιουργεί μία νέα μέτρηση σε ένα χρήστη. |
| /users/:username/measurements/:measurement | GET | Επιστρέφει τα στοιχεία μιας μέτρησης ενός χρήστη. |
| /users/:username/measurements/:measurement | DELETE | Διαγράφει μία μέτρηση ενός χρήστη. |

Ορισμένες από τις παραπάνω διαδρομές έχουν τα ακόλουθα χαρακτηριστικά:

- Είναι εμφωλευμένες (nested), δηλαδή η διαδρομή ενός αντικειμένου μπορεί να περιλαμβάνει ή να αναφέρει ένα άλλο.
- Περιέχουν παραμέτρους διεύθυνσης URL/path, δηλαδή μεταβλητές που αφορούν ένα αντικείμενο ή μια οντότητα στη διεύθυνση URL.
- Περιέχουν παραμέτρους σώματος, π.χ. όπου POST Request, το σώμα πρέπει να είναι τύπου JSON και να περιέχει ένα αντικείμενο ή να έχει τη μορφή πίνακα που περιέχει αντικείμενα.

Για παράδειγμα, η εφαρμογή μπορεί να λαμβάνει δεδομένα από οποιαδήποτε συσκευή μπορεί να λειτουργήσει ως πελάτης, δηλαδή:

1. να έχει πρόσβαση στο διαδίκτυο,
2. να μπορεί να κάνει HTTP POST αιτήματα χρησιμοποιώντας Authorization Header για την αποστολή Bearer Token και

3. να μπορεί να στέλνει Body τύπου JSON (το Body πρέπει να έχει τη μορφή πίνακα με αντικείμενα).

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

*Δείγματα κώδικα θα βρείτε στο παράρτημα Α.4*

#### 3.1.5.5 Ελεγκτές (controllers)

Σύμφωνα με τις παραπάνω διαδρομές, δημιουργήθηκαν οι ελεγκτές και οι υπηρεσίες τους. Κάθε ελεγκτής και ιδιαίτερα οι μέθοδοί του είναι υπεύθυνοι για τον έλεγχο και την επικύρωση των δεδομένων που εισέρχονται αλλά και για την παροχή της σωστής απάντησης (response) που αποστέλλεται με κάθε request, ενώ η απάντηση πρέπει πάντα να συνοδεύεται από ένα κωδικό κατάστασης HTTP (HTTP Status Code).

Το HTTP καθορίζει αυτούς τους κωδικούς και αυτοί διακρίνονται σε πέντε κατηγορίες:

- **1xx**: Ενημερωτικό - Αυτοί οι κωδικοί κατάστασης υποδεικνύουν μια προσωρινή απόκριση.
- **2xx**: Επιτυχία - Αυτή η κλάση κωδικών κατάστασης υποδεικνύει ότι ο διακομιστής αποδέχτηκε με επιτυχία την αίτηση του υπολογιστή-πελάτη.
- **3xx**: Ανακατεύθυνση - Το πρόγραμμα περιήγησης του υπολογιστή-πελάτη θα πρέπει να πραγματοποιήσει περαιτέρω ενέργειες για να ικανοποιήσει την αίτηση.
- **4xx**: Σφάλμα προγράμματος πελάτη - Προκύπτει σφάλμα και ο υπολογιστής-πελάτης ενδέχεται να αντιμετωπίζει κάποιο πρόβλημα.
- **5xx**: Σφάλμα διακομιστή - Η αίτηση δεν μπορεί να ολοκληρωθεί από το διακομιστή επειδή προέκυψε σφάλμα.

*Δείγματα κώδικα θα βρείτε στα παραρτήματα Α.5 και Α.6.*

#### 3.1.5.6 Υπηρεσίες (services)

*Δείγματα κώδικα θα βρείτε στο παράρτημα Α.7.*

#### 3.1.5.7 Ενδιάμεσο λογισμικό (middleware)

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

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

Η βιβλιοθήκη JSONWebToken χρησιμοποιήθηκε ειδικότερα για τις λειτουργίες αυθεντικοποίησης και εξουσιοδότησης χρηστών και συσκευών. Το JWT Token δημιουργείται κατά την εγγραφή ή την αυθεντικοποίηση του χρήστη στις διαδρομές /users/signup και /users/login αντίστοιχα και αποθηκεύεται στον Client του χρήστη ή της συσκευής. Ενώ για την αυθεντικοποίηση των μετρήσεων δημιουργούνται δύο API Keys από τον χρήστη μέσω της εφαρμογής κατά τον προγραμματισμό μίας νέας μέτρησης.

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

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

Σε αυτήν την εφαρμογή, η χρήση ενός Cookie με την ιδιότητα httpOnly θεωρήθηκε ασφαλέστερη. Ένα Cookie με αυτήν την ιδιότητα δεν είναι προσβάσιμο στο JavaScript

Document.cookie API και αποστέλλεται μόνο στον διακομιστή. Αυτή η προφύλαξη βοηθά στην αποφυγή επιθέσεων τύπου cross-site scripting (XSS). Ωστόσο, στο middleware authorize.js έγινε χρήση της βιβλιοθήκης Express-JWT και υλοποίηση κώδικα που δέχεται το JWT Token του χρήστη από τρείς τοποθεσίες όπως:

1. το Authorization Header του HTTP request σαν Bearer Token,
2. μέσω παραμέτρου στη διεύθυνση URL,
3. μέσω Cookie με την ιδιότητα httpOnly.

*Δείγματα κώδικα θα βρείτε στο παράρτημα Α.8.*

### 3.1.6 Ανάλυση WebSocket server

Στη παρούσα υλοποίηση ο WebSocket server αποτελεί το σημαντικότερο κομμάτι της εφαρμογής νέφους για την έρευνα αυτή. Ο WebSocket server που υλοποιήθηκε λειτουργεί παράλληλα με τον HTTP Server εντός του Node.js και έτσι είναι έχει τη δυνατότητα να χρησιμοποιεί τις υπηρεσίες, τους ελεγκτές, το ενδιάμεσο λογισμικό και τα μοντέλα του REST-API απευθείας. Κάτι που καθιστά την υλοποίηση πιο γρήγορη στην ανάπτυξή της αλλά και πιο ασφαλή.

Πιο αναλυτικά, δημιουργήθηκε ένας WebSocket server ο οποίος δέχεται συνδέσεις στην ίδια διεύθυνση IP και port με τον HTTP server. Εφαρμόζει έλεγχο ταυτοποίησης των συνδέσεων αυτών με χρήση JWT token και όταν αυτές είναι έγκυρες δημιουργεί νέα WebSocket rooms για τους χρήστες που πρόκειται να συνδεθούν σε αυτά. Ταυτόχρονα βρίσκει την μέτρηση στην οποία αντιστοιχεί το συγκεκριμένο WebSocket room και θέτει όριο στο χρόνο της σύνδεσης όπως αυτό έχει οριστεί από τον χρήστη.

Τέλος, με την δημιουργία μιας σύνδεσης με έναν χρήστη, δημιουργείται αυτόματα ένα αρχείο στον server στο οποίο αποθηκεύονται όλα τα μηνύματα της συνεδρίας. Με την ολοκλήρωση της συνεδρίας, ο χρήστης που προγραμμάτισε την μέτρηση, μπορεί να κατεβάσει το αρχείο που αντιστοιχεί σε αυτή.

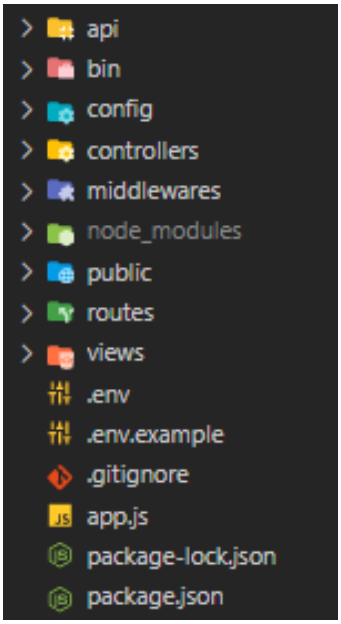
*Δείγματα κώδικα θα βρείτε στο παράρτημα Α.9.*

### 3.1.7 Ανάλυση Front-end

#### 3.1.7.1 Δομικά μέρη

Όπως φαίνεται από την αρχιτεκτονική, το Front-end αποτελείται από 6 βασικά τμήματα:

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




*Εικόνα 9 Δομικά μέρη Front-end*

Όπως και το Back-end, έτσι και το Front-end κάνει χρήση ενός κεντρικού αρχείου για την εφαρμογή καθώς και βιβλιοθήκες/modules/εξαρτήσεις που αποθηκεύονται στον φάκελο **node_modules,** και υπηρεσίες από «κοινόχρηστα» modules του φακέλου **config**. Επίσης, σε αυτήν την αρχιτεκτονική, επειδή πρόκειται για Server-Side Rendering (SSR) το Front-end μπορεί να χρησιμοποιεί υπηρεσίες απευθείας από το

REST API όταν αυτό είναι απαραίτητο, προσθέτοντας ασφάλεια σε ορισμένα μέρη της εφαρμογής επιταχύνοντας ταυτόχρονα την ανάπτυξή της.

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

**Προβολές (views):**

- Περιλαμβάνουν τμήματα από τη δομή (template) της σελίδας που εμφανίζεται στον Client.
- Χρησιμοποιούνται από τον μηχανισμό προβολής (view engine) που έχει οριστεί στον διακομιστή όπως το HBS σε αυτήν την υλοποίηση.
- Χρησιμοποιούνται για την προβολή σελίδων από τη πλευρά του διακομιστή (Server-Side Rendering).

**Δημόσιος φάκελος αρχείων (public)**

- Το Express.static είναι ένα ενδιάμεσο λογισμικό του Express Framework το οποίο χρησιμοποιείται για την προβολή στατικών αρχείων, όπως εικόνες, αρχεία CSS και αρχεία JavaScript.
- Ο φάκελος static περιλαμβάνει διάφορα στατικά αρχεία που είναι απαραίτητα για τον πελάτη ώστε να προβάλει σωστά το Front-end.

#### 3.1.7.2 Διαδρομές (routes)

Κάθε διαδρομή του Front-end που διαχειρίζεται HTTP request με μέθοδο GET αντιστοιχεί είτε σε μια προβολή (όπου η μέθοδος της διαδρομής λειτουργεί ως ελεγκτής και χρησιμοποιεί απευθείας μια προβολή) είτε σε τουλάχιστον έναν ελεγκτή που, στη συνέχεια, χρησιμοποιεί μια προβολή.

#### 3.1.7.2.1 Ανάλυση διαδρομών

Βάσει των λειτουργικών απαιτήσεων, δημιουργήθηκαν οι παρακάτω διαδρομές.

*Πίνακας 9 Διαδρομές Front-end*

| Διαδρομή | Μέθοδος | Περιγραφή |
|---|---|---|
| / | GET | Διαδρομή προβολής της αρχικής σελίδας. |
| /login | GET | Διαδρομή προβολής της σελίδας σύνδεσης χρήστη. |
| /login | POST | Διαδρομή που δέχεται τα δεδομένα σύνδεσης χρήστη. |
| /signup | GET | Διαδρομή προβολής της σελίδας εγγραφής χρήστη. |
| /signup | POST | Διαδρομή που δέχεται τα δεδομένα εγγραφής χρήστη. |
| /logout | GET | Διαδρομή προβολής της σελίδας αποσύνδεσης χρήστη. |
| /admin | GET | Διαδρομή προβολής της σελίδας διαχείρισης (ταμπλό) χρήστη. |
| /admin/measurements | GET | Διαδρομή προβολής της σελίδας διαχείρισης μετρήσεων χρήστη. |
| /admin/profile | GET | Διαδρομή προβολής της σελίδας ρυθμίσεων λογαριασμού και στοιχείων χρήστη. |
| /admin/super | GET | Διαδρομή προβολής της σελίδας διαχείρισης (ταμπλό) διαχειριστή. |
| /admin/super/measurements | GET | Διαδρομή προβολής της σελίδας διαχείρισης μετρήσεων διαχειριστή. |
| /admin/super/users | GET | Διαδρομή προβολής της σελίδας διαχείρισης χρηστών διαχειριστή. |

*Δείγματα κώδικα θα βρείτε στα παραρτήματα Α.10 και Α.11.*

#### 3.1.7.3 Ελεγκτές (controllers)

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

*Δείγματα κώδικα θα βρείτε στο παράρτημα Α.12.*

#### 3.1.7.4 Ενδιάμεσο λογισμικό (middleware)

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

*Δείγματα κώδικα θα βρείτε στο παράρτημα Α.13.*

#### 3.1.7.5 Προβολές (views)

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

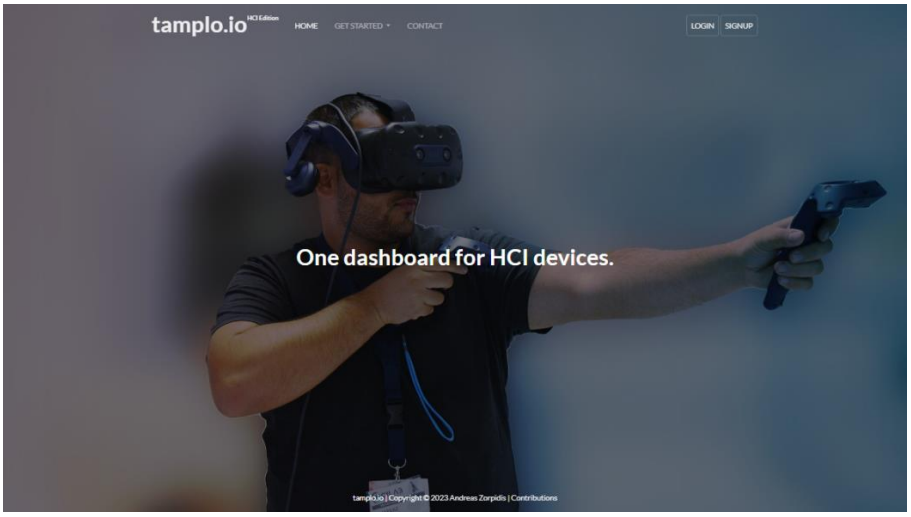
Αρχικά, μετατράπηκε η δομή του προτύπου AdminLTE ώστε να είναι συμβατή με το HBS view engine και το Handlebars.js (δηλαδή έγινε ο χωρισμός του σε κομμάτια «partials» τύπου .hbs) και μεταφέρθηκαν οι στατικοί φάκελοι και βιβλιοθήκες του στον φάκελο δημόσιου υλικού (public) ώστε οι σελίδες να μπορούν να έχουν πρόσβαση σε αυτά κατά τη διαδικασία του rendering. Στη συνέχεια, με χρήση στοιχείων (components) από το AdminLTE, κατασκευάστηκαν οι προβολές που αντιπροσωπεύουν τις σελίδες/διαδρομές του Front-end, και υλοποιήθηκαν Promise-based μέθοδοι σε JavaScript με χρήση της βιβλιοθήκης Axios για την ανάκτηση (fetching) των δεδομένων κάθε σελίδας.

*Δείγματα κώδικα θα βρείτε στο παράρτημα Α.14.*

### 3.1.8 Φωτογραφίες από την εφαρμογή

#### 3.1.8.1 Αρχική σελίδα

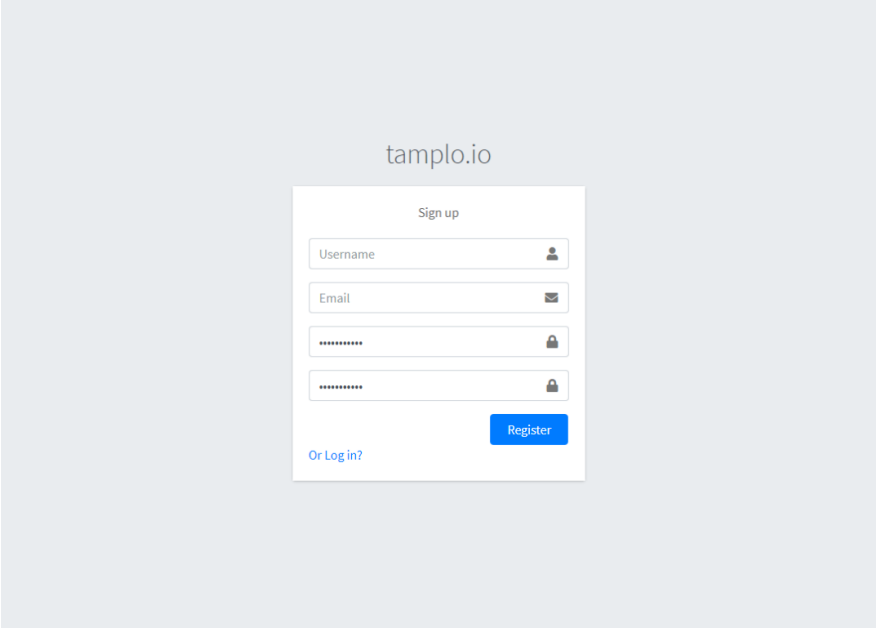
Προβολή: index.hbs




*Εικόνα 10 Αρχική σελίδα*

#### 3.1.8.2 Σελίδα εγγραφής

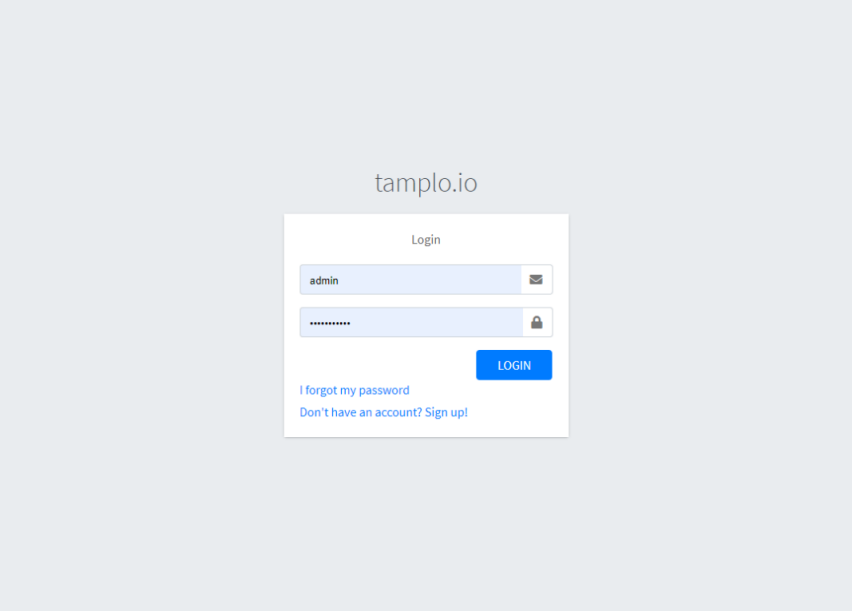
Προβολή: signup.hbs




*Εικόνα 11 Σελίδα εγγραφής (χρήστη)*

### 3.1.8.3 Σελίδα σύνδεσης

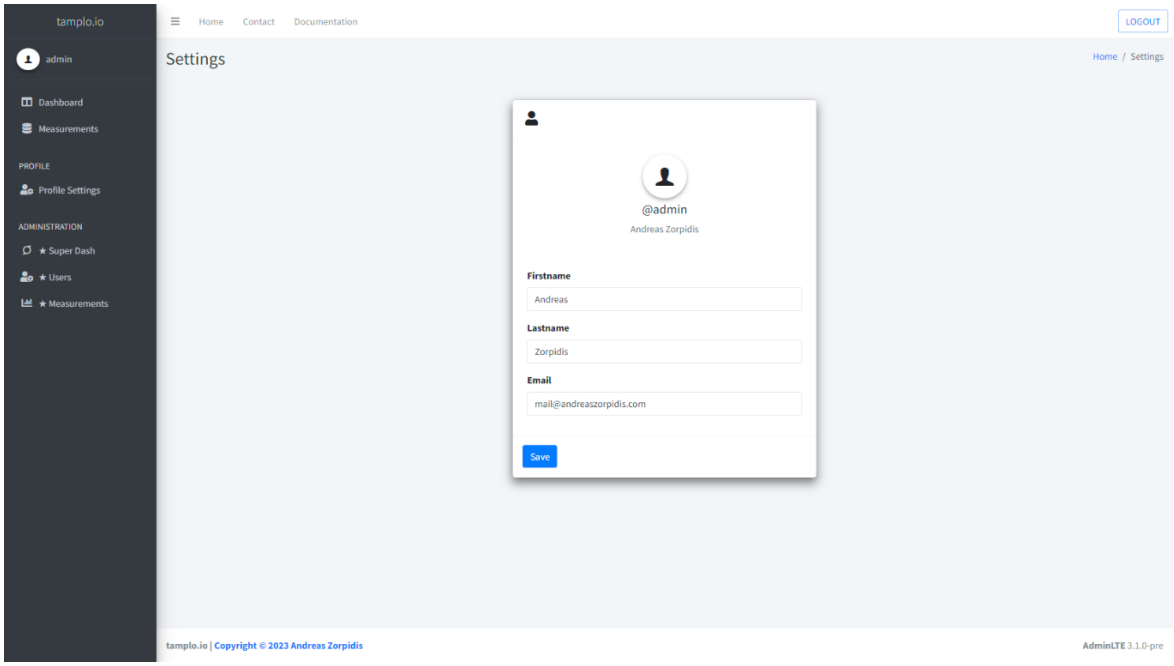
Προβολή: sign.hbs




*Εικόνα 12 Σελίδα σύνδεσης (χρήστη)*

### 3.1.8.4 Σελίδα ρυθμίσεων προφίλ χρήστη

Προβολή: admin/profile.hbs

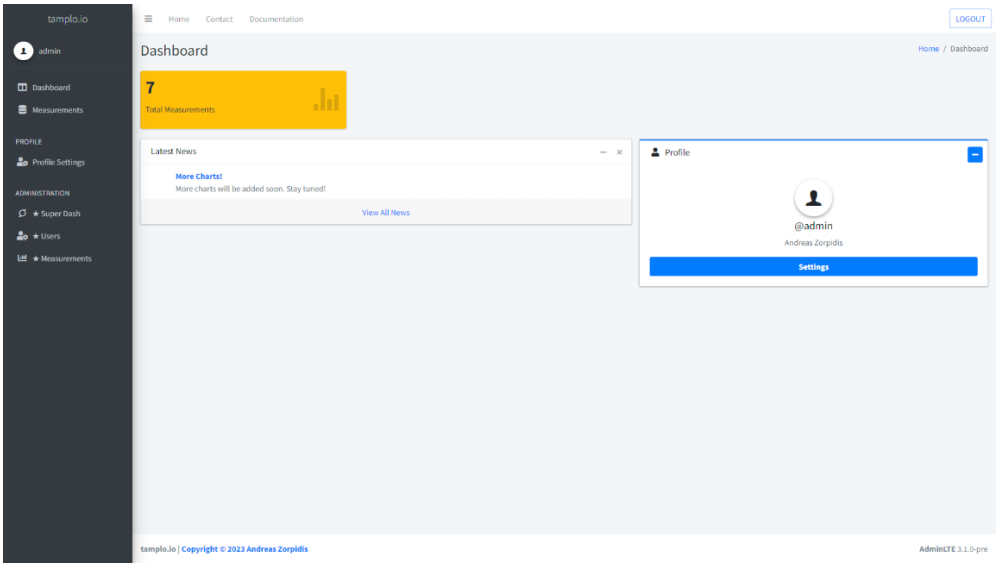



*Εικόνα 13 Σελίδα ρυθμίσεων προφίλ (χρήστη)*

### 3.1.8.5 Σελίδες χρήστη

#### 3.1.8.5.1 Αρχικό ταμπλό

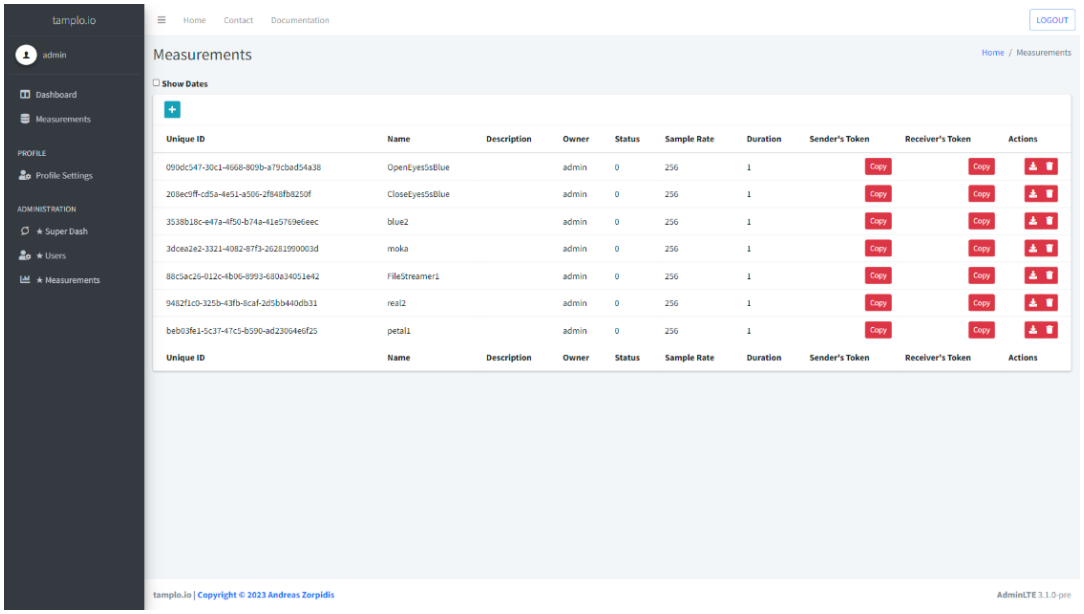
Προβολή: admin/index.hbs




*Εικόνα 14 Αρχικό ταμπλό (χρήστη)*

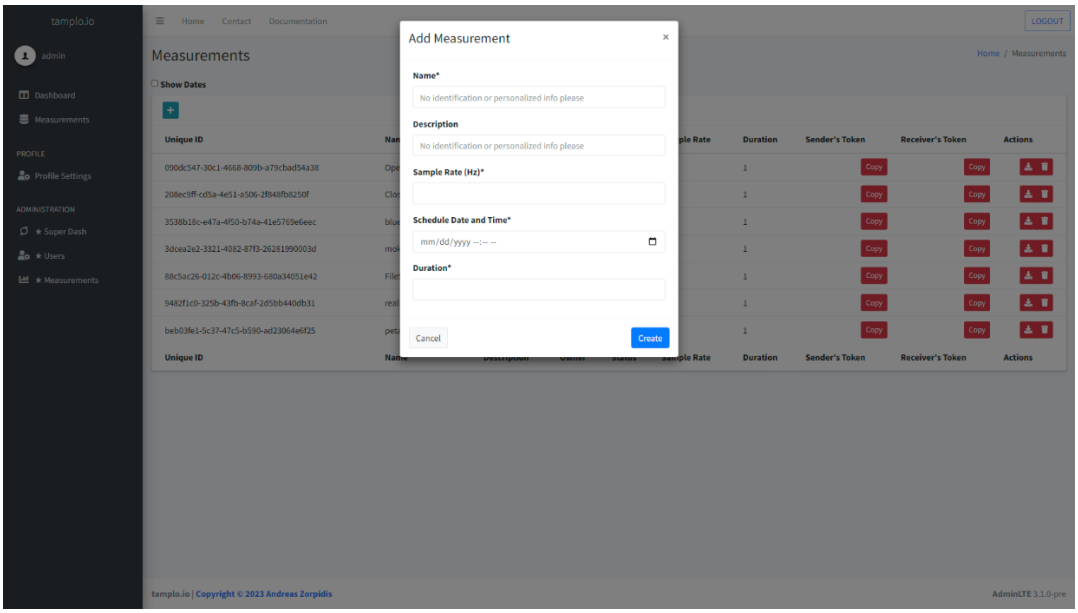
#### 3.1.8.5.2 Μετρήσεις

Προβολή: admin/measurements.hbs




*Εικόνα 15 Σελίδα μετρήσεων (χρήστη)*

#### 3.1.8.5.3 Προσθήκη μέτρησης

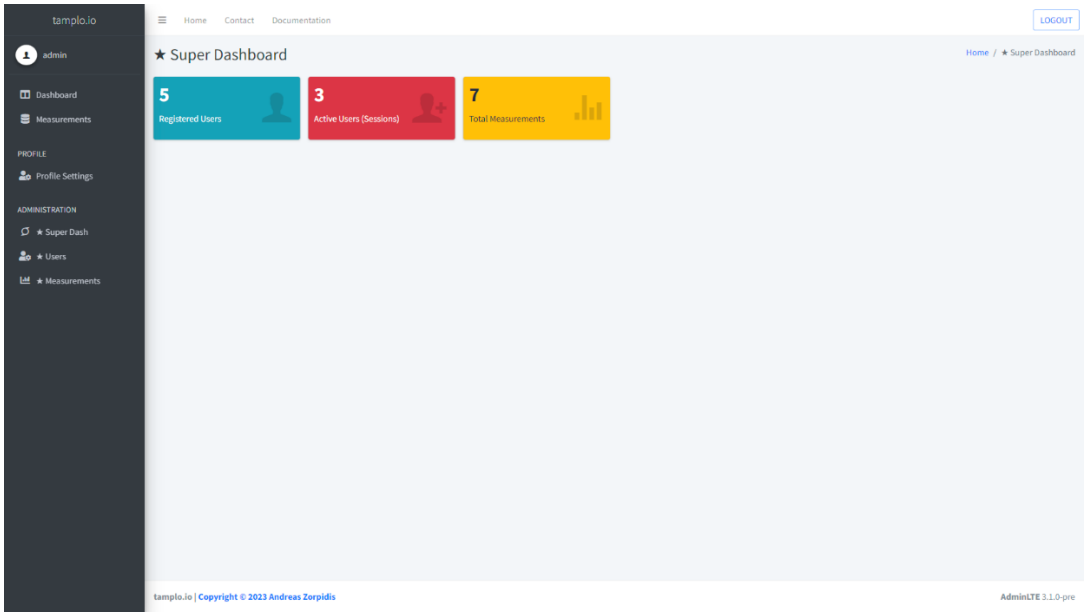



*Εικόνα 16 Pop-up modal για τη προσθήκη μέτρησης*

### 3.1.8.6 Σελίδες διαχειριστή

#### 3.1.8.6.1 Αρχικό ταμπλό

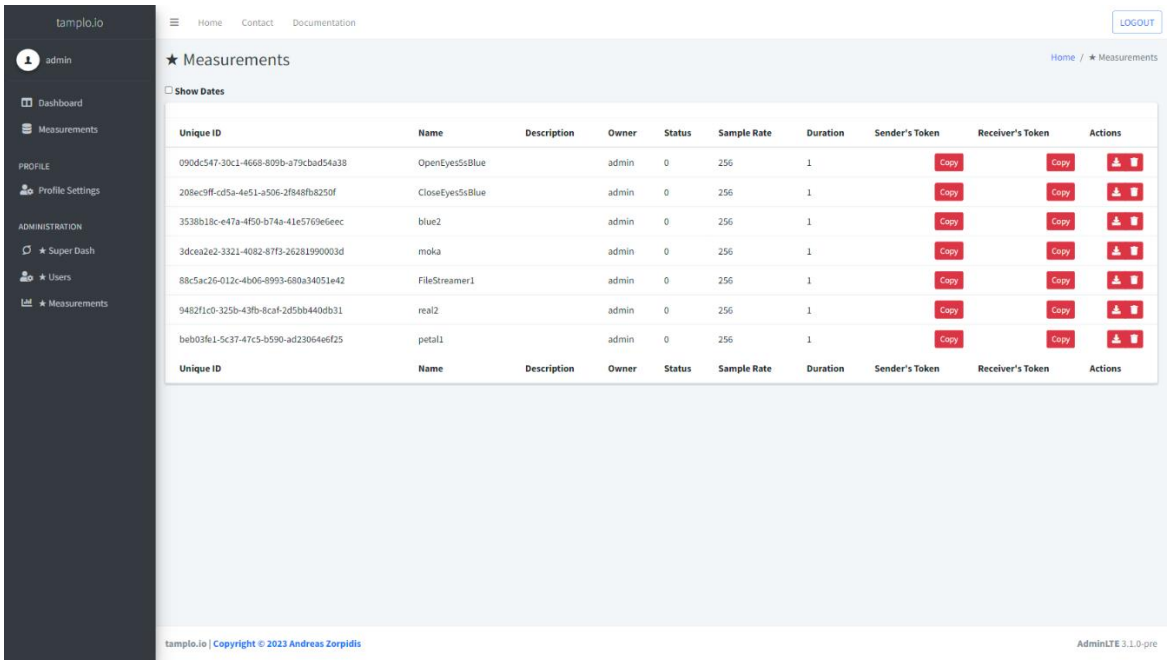
Προβολή: admin/super/index.hbs




*Εικόνα 17 Αρχικό ταμπλό (διαχειριστή)*

### 3.1.8.6.2 Μετρήσεις

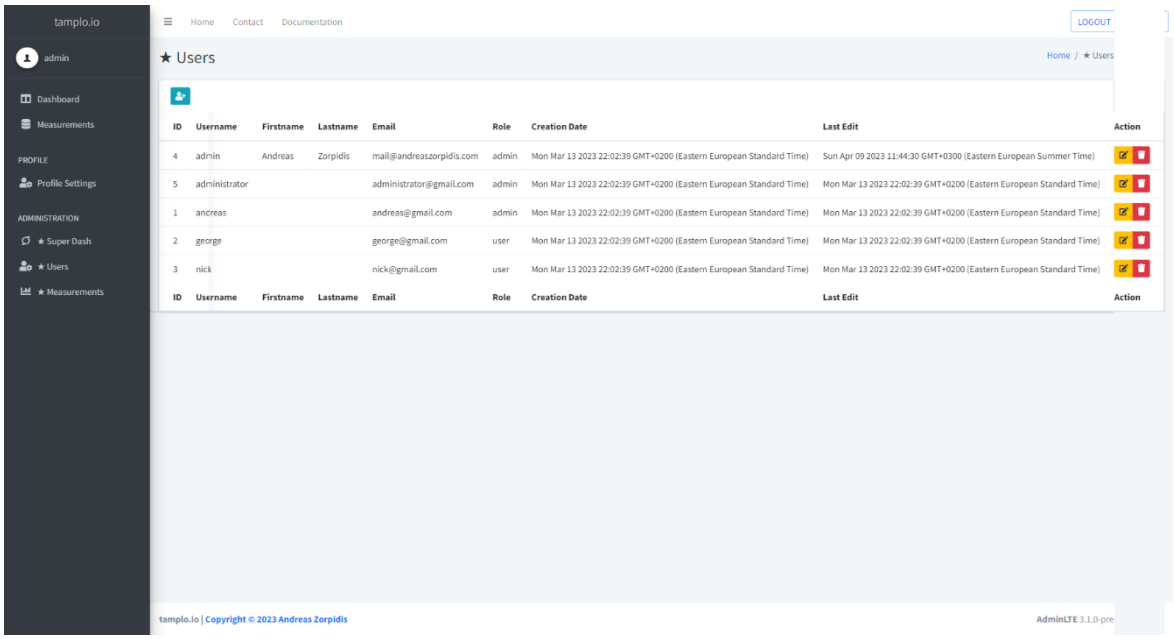
Προβολή: admin/super/measurements.hbs




*Εικόνα 18 Σελίδα διαχείρισης μετρήσεων (διαχειριστή)*

### 3.1.8.6.3 Χρήστες

Προβολή: admin/super/users.hbs




*Εικόνα 19 Σελίδα διαχείρισης χρηστών (διαχειριστή)*

## 3.2 Πολυχρηστική εφαρμογή (client)

### 3.2.1 Απαιτήσεις

#### 3.2.1.1 Επιχειρησιακές διαδικασίες

Όπως αναφέρθηκε και στην αντίστοιχη ενότητα για την εφαρμογή νέφους, η καταγραφή των λειτουργικών διαδικασιών είναι ο κατάλληλος τρόπος για να βρεθούν οι λειτουργικές απαιτήσεις της εφαρμογής και οι περιπτώσεις χρήσης. Αυτές οι λειτουργικές διαδικασίες, που είναι επίσης γνωστές ως επιχειρησιακές διαδικασίες, βρίσκονται στα επόμενα υποκεφάλαια. Οι παρακάτω επιχειρησιακές διαδικασίες έχουν κριθεί απαραίτητες για την εφαρμογή:

**Εξουσιοδότηση**

- Επαλήθευση συνεδρίας χρήστη

**Χρήστες**

- Εμφάνιση διαχειριστικού περιβάλλοντος

**Μετρήσεις**

- Λήψη μέτρησης μέσω UDP/OSC
- Λήψη μέτρησης μέσω WebSocket/OSC
- Αποθήκευση μέτρησης σε αρχείο
- Ανάγνωση μέτρησης από αρχείο
- Εκπομπή μέτρησης σε UDP/OSC
- Εκπομπή μέτρησης σε WebSocket/OSC

#### 3.2.1.2 Επιχειρησιακοί κανόνες

Συντομογραφία br: business-rule

*Πίνακας 10 Επιχειρησιακοί κανόνες*

| **Ταυτότητα / Όνομα** | **Περιγραφή** |
|---|---|
| br.usage | Η εφαρμογή επιτρέπει την χρήση της από έναν μόνο χρήστη ανά συσκευή. |
| br.functionality | Η εφαρμογή επιτρέπει στον χρήστη να επιλέξει ανάμεσα σε λειτουργίες λήψης, αποστολής, ανάγνωσης και αποθήκευσης μετρήσεων με χρήση UDP/OSC ή WebSocket/OSC. |
| br.functionality.receive<br>br.functionality.send | Η εφαρμογή επιτρέπει στον χρήστη να παρέχει τα δικά του στοιχεία IP/Port για τις λειτουργίες της λήψης και αποστολής. |
| br.functionality.receive.apikey<br>br.functionality.send.apikey | Η εφαρμογή παρέχει τη δυνατότητα επαληθευμένης επικοινωνίας για κάθε λήψη ή αποστολή μέτρησης. |
| br.functionality.receive.log<br>br.functionality.send.log | Η εφαρμογή επιτρέπει στον χρήστη να καταγράφει δεδομένα για όλους τους τρόπους επικοινωνίας. |
| br.measurement.file | Η εφαρμογή επιτρέπει στον χρήστη να καταγράφει την μέτρηση τοπικά στον υπολογιστή του σε αρχείο. |
| br.error | Η εφαρμογή εμφανίζει μηνύματα λάθους όταν κάποια λειτουργία δε μπορεί να πραγματοποιηθεί. |

#### 3.2.1.3 Μη λειτουργικές απαιτήσεις

Συντομογραφία nfr: non-functional requirements

*Πίνακας 11 Μη λειτουργικές απαιτήσεις*

| **Ταυτότητα / Όνομα** | **Περιγραφή** |
|---|---|
| nfr.design.responsive | Οι διεπαφές της εφαρμογής θα πρέπει να εμφανίζονται σωστά σε κινητά, tablet και υπολογιστές με διάφορες διαστάσεις οθονών. |
| nfr.compatibility | Η εφαρμογή θα πρέπει να είναι λειτουργική σε Windows και Mac. |
| nfr.development.language | Η εφαρμογή θα αναπτυχθεί σε JavaScript. |
| nfr.development.technology | Η εφαρμογή θα αναπτυχθεί με χρήση της τεχνολογίας Electron.js |
| nfr.communication | Η εφαρμογή θα δέχεται και θα στέλνει μηνύματα από με UDP/OSC και WebSocket/OSC. |
| nfr.userinterface | Το γραφικό περιβάλλον της εφαρμογής θα |

| | |
|---|---|
| | υλοποιηθεί με Bootstrap. |

#### 3.2.1.4 Λειτουργικές απαιτήσεις

Συντομογραφία fr: functional requirements

*Πίνακας 12 Λειτουργικές απαιτήσεις*

| **Ταυτότητα / Όνομα** | **Περιγραφή** |
|---|---|
| fr.gui | Η εφαρμογή θα παρέχει ένα γραφικό περιβάλλον. |
| fr.functionalityOption | Η εφαρμογή θα επιτρέπει στον χρήστη να επιλέξει ανάμεσα σε λειτουργίες λήψης, αποστολής, ανάγνωσης και αποθήκευσης μετρήσεων με χρήση UDP/OSC ή WebSocket/OSC. |
| fr.input | Η εφαρμογή θα επιτρέπει στον χρήστη να παρέχει τα δικά του στοιχεία IP/Port για τις λειτουργίες της λήψης και αποστολής. |
| fr.apikey | Η εφαρμογή θα παρέχει στον χρήστη τη δυνατότητα επαληθευμένης επικοινωνίας με χρήση token. |
| fr.log | Η εφαρμογή θα επιτρέπει στον χρήστη να καταγράφει δεδομένα για όλους τους τρόπους επικοινωνίας. |
| fr.savetofile | Η εφαρμογή θα επιτρέπει στον χρήστη να καταγράφει την μέτρηση τοπικά στον υπολογιστή του σε αρχείο. |
| fr.savetofile.path | Η εφαρμογή θα επιτρέπει στον χρήστη να επιλέξει τη τοποθεσία αποθήκευσης των αρχείων καταγραφής. |
| fr.gui.log | Η εφαρμογή θα δίνει τη δυνατότητα στο χρήστη να εμφανίζει τα μηνύματα στο γραφικό περιβάλλον. |
| fr.error.handling<br>fr.error.log | Η εφαρμογή θα διαχειρίζεται τυχόν σφάλματα και θα εμφανίζει σχετικά μηνύματα στο γραφικό περιβάλλον. |

#### 3.2.1.5 Λεξικό δεδομένων

*Πίνακας 13 Λεξικό δεδομένων*

| **Δεδομένο** | **Περιγραφή** |
|---|---|
| electron | Αντικείμενο του Electron.js για την υλοποίηση εφαρμογών για υπολογιστές με χρήση HTML, CSS, JavaScript. |
| app | Κύριο αντικείμενο της εφαρμογής που παρέχει τον έλεγχο του κύκλου ζωής της καθώς και τη διαχείριση του παραθύρου. |
| BrowserWindow | Κλάση του Electron που παρέχει τη δυνατότητα ελέγχου των παραθύρων. |
| ipcMain | Κλάση του Electron που παρέχει ένα τρόπο επικοινωνίας μεταξύ της κύριας διεργασίας με τη διεργασία προβολής. |

| | |
|---|---|
| osc | Αντικείμενο της βιβλιοθήκης OSC για την λήψη και αποστολή OSC μηνυμάτων. |
| WebSocket | Αντικείμενο του Node.js για τη δημιουργία επικοινωνιών με WebSocket. |
| fs | Αντικείμενο του Node.js που παρέχει λειτουργίες για το σύστημα αρχείων. |
| path | Αντικείμενο του Node.js που παρέχει λειτουργίες για το σύστημα μονοπατιών του συστήματος αρχείων. |
| mainWindow | Μεταβλητή που κάνει αναφορά στη κύρια διεργασία. |
| saveToPath | Μεταβλητή που κρατά το μονοπάτι αποθήκευσης του αρχείου μιας μέτρησης. |
| createWindow() | Συνάρτηση δημιουργίας κυρίως παραθύρου. |
| app.on('ready', createWindow) | Event handler που καλείται από το κύριο αντικείμενο της εφαρμογής όταν αυτό είναι έτοιμο να δημιουργήσει παράθυρα. |
| app.on('window-all-closed', ...) | Event handler που καλείται όταν όλα τα παράθυρα της εφαρμογής έχουν κλείσει. |
| app.on('activate', ...) | Event handler που καλείται όταν η εφαρμογή έχει ενεργοποιηθεί αλλά δεν έχουν ανοίξει ακόμα παράθυρα. |
| ipcMain.on('start-app', ...) | Event handler που καλείται όταν η διεργασία προβολής στέλνει το μήνυμα “start-app” στη κύρια διεργασία. |
| directoryPath | Μεταβλητή που κρατά το όνομα του μονοπατιού αποθήκευσης των αρχείων των μετρήσεων. |
| year, month, day, hour, minute, file_name | Μεταβλητές που κρατάνε την ημερομηνία και ώρα για χρήση στο όνομα του JSON αρχείου κάθε μέτρησης. |
| filename | Μεταβλητή που κρατά το όνομα του αρχείου μιας μέτρησης. |
| option | Μεταβλητή που κρατά την επιλογή λειτουργικότητας της εφαρμογής που έχει επιλέξει ο χρήστης. |
| logData | Μεταβλητή που καθορίζει αν θα εμφανιστούν μηνύματα στο γραφικό περιβάλλον. |
| receiveOnly | Μεταβλητή που καθορίζει αν θα γίνει ξανά αποστολή των μηνυμάτων που έχουν ληφθεί. |
| udpAddress, udpPort | Μεταβλητές που κρατάνε τα στοιχεία IP, port για τη σύνδεση UDP/OSC. |
| wsAddress, wsPort | Μεταβλητές που κρατάνε τα στοιχεία IP, port για τη σύνδεση WebSocket/OSC. |

#### 3.2.1.6 Περιπτώσεις χρήσεις (use cases)

##### 3.2.1.6.1 Ανάγνωση και αναμετάδοση σημάτων από UDP/OSC σε WebSocket/OSC

**Περίπτωση χρήσης:** Ανάγνωση και αναμετάδοση σημάτων από UDP/OSC σε WebSocket/OSC

**Ταυτότητα:** uc.udpToWeb

**Δράστης:** Χρήστης

**Αρχικές συνθήκες:** Ο χρήστης έχει εκτελέσει την εφαρμογή και έχει εμφανιστεί το γραφικό περιβάλλον στην οθόνη του.

**Βασική ροή:**

Β0α: Ο δράστης επιλέγει από το μενού τη λειτουργία UDP/OSC->WebSocket/OSC.

Β1α: Ο δράστης δίνει τα στοιχεία της σύνδεσης UDP (IP,Port) και WebSocket (IP, Port) αντίστοιχα. Επίσης, δίνει ένα token εάν επιθυμεί επαληθευμένη επικοινωνία στη σύνδεση με WebSocket.

Β1β: Η εφαρμογή

Β1β1: πραγματοποιεί σύνδεση στη διεύθυνση UDP και τη κρατάει ενεργή,

Β1β2: πραγματοποιεί σύνδεση στη διεύθυνση WebSocket (προωθεί το token στον διακομιστή εάν αυτό υπάρχει) και αφού επιτραπεί η σύνδεση με το διακομιστή,

Β1β3: ξεκινά να προωθεί τα μηνύματα που λαμβάνει.

**Εναλλακτικές ροές:**

Ε1: Αν στο βήμα Β1β1 τα δεδομένα δεν είναι έγκυρα ή η σύνδεση δεν είναι επιτυχής, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β1α.

Ε2: Αν στο βήμα Β1β2 τα δεδομένα δεν είναι έγκυρα ή η σύνδεση δεν είναι επιτυχής, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β1α.

Ε3: Αν στο βήμα Β1β3 τα μηνύματα δεν έχουν έγκυρη μορφή, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β1α.

#### 3.2.1.6.2 Αποθήκευση σημάτων από UDP/OSC σε αρχείο

**Περίπτωση χρήσης:** Αποθήκευση σημάτων από UDP/OSC σε αρχείο

**Ταυτότητα:** uc.udpToFile

**Δράστης:** Χρήστης

**Αρχικές συνθήκες:** Ο χρήστης έχει εκτελέσει την εφαρμογή και έχει εμφανιστεί το γραφικό περιβάλλον στην οθόνη του.

**Βασική ροή:**

Β0α: Ο δράστης επιλέγει από το μενού τη λειτουργία UDP/OSC->WebSocket/OSC.

Β1α: Ο δράστης δίνει τα στοιχεία της σύνδεσης UDP (IP,Port) και επιλέγει τις λειτουργίες “Save data-stream to file” και “Receive/Read only | Don’t connect to endpoint”.

Β1β: Η εφαρμογή

Β1β1: πραγματοποιεί σύνδεση στη διεύθυνση UDP και τη κρατάει ενεργή,

Β1β2: ξεκινά να αποθηκεύει τα μηνύματα που λαμβάνει στο αρχείο.

**Εναλλακτικές ροές:**

Ε1: Αν στο βήμα Β1β1 τα δεδομένα δεν είναι έγκυρα ή η σύνδεση δεν είναι επιτυχής, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β1α.

Ε2: Αν στο βήμα Β1β2 τα δεδομένα δεν είναι έγκυρα ή υπάρξει σφάλμα στο σύστημα αρχείων, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β1α.

#### 3.2.1.6.3 Αποθήκευση σημάτων από WebSocket/OSC σε αρχείο

**Περίπτωση χρήσης:** Αποθήκευση σημάτων από WebSocket/OSC σε αρχείο

**Ταυτότητα:** uc.webToFile

**Δράστης:** Χρήστης

**Αρχικές συνθήκες:** Ο χρήστης έχει εκτελέσει την εφαρμογή και έχει εμφανιστεί το γραφικό περιβάλλον στην οθόνη του.

**Βασική ροή:**

Β0α: Ο δράστης επιλέγει από το μενού τη λειτουργία WebSocket/OSC->UDP/OSC.

Β1α: Ο δράστης δίνει τα στοιχεία της σύνδεσης WebSocket (IP, Port) και ένα token εάν επιθυμεί επαληθευμένη επικοινωνία στη σύνδεση με WebSocket. Επίσης, επιλέγει τις λειτουργίες “Save data-stream to file” και “Receive/Read only | Don’t connect to endpoint”.

Β1β: Η εφαρμογή

Β1β1: πραγματοποιεί σύνδεση στη διεύθυνση WebSocket (προωθεί το token στον διακομιστή εάν αυτό υπάρχει) και αφού επιτραπεί η σύνδεση με το διακομιστή τη κρατάει ενεργή,

Β1β2: ξεκινά να αποθηκεύει τα μηνύματα που λαμβάνει στο αρχείο.

**Εναλλακτικές ροές:**

Ε1: Αν στο βήμα Β1β1 τα δεδομένα δεν είναι έγκυρα ή η σύνδεση δεν είναι επιτυχής, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β1α.

Ε2: Αν στο βήμα Β1β2 τα δεδομένα δεν είναι έγκυρα ή υπάρξει σφάλμα στο σύστημα αρχείων, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β1α.

#### 3.2.1.6.4 Ανάγνωση και αναμετάδοση σημάτων από WebSocket/OSC σε UDP/OSC

**Περίπτωση χρήσης:** Ανάγνωση και αναμετάδοση σημάτων από WebSocket/OSC σε UDP/OSC

**Ταυτότητα:** uc.webToUdp

**Δράστης:** Χρήστης

**Αρχικές συνθήκες:** Ο χρήστης έχει εκτελέσει την εφαρμογή και έχει εμφανιστεί το γραφικό περιβάλλον στην οθόνη του.

**Βασική ροή:**

Β0α: Ο δράστης επιλέγει από το μενού τη λειτουργία WebSocket/OSC-> UDP/OSC.

Β1α: Ο δράστης δίνει τα στοιχεία της σύνδεσης UDP (IP,Port) και WebSocket (IP, Port) αντίστοιχα. Επίσης, δίνει ένα token εάν επιθυμεί επαληθευμένη επικοινωνία στη σύνδεση με WebSocket.

Β1β: Η εφαρμογή

Β1β1: πραγματοποιεί σύνδεση στη διεύθυνση WebSocket (προωθεί το token στον διακομιστή εάν αυτό υπάρχει) και αφού επιτραπεί η σύνδεση με το διακομιστή τη κρατάει ενεργή,

Β1β2: πραγματοποιεί σύνδεση στη διεύθυνση UDP και

Β1β3: ξεκινά να προωθεί τα μηνύματα που λαμβάνει.

**Εναλλακτικές ροές:**

Ε1: Αν στο βήμα Β1β1 τα δεδομένα δεν είναι έγκυρα ή η σύνδεση δεν είναι επιτυχής, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β1α.

Ε2: Αν στο βήμα Β1β2 τα δεδομένα δεν είναι έγκυρα ή η σύνδεση δεν είναι επιτυχής, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β1α.

Ε3: Αν στο βήμα Β1β3 τα μηνύματα δεν έχουν έγκυρη μορφή, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β1α.

#### 3.2.1.6.5 Ανάγνωση και αναμετάδοση σημάτων από αρχείο σε UDP/OSC

**Περίπτωση χρήσης:** Ανάγνωση και αναμετάδοση σημάτων από αρχείο σε UDP/OSC

**Ταυτότητα:** uc.fileToUdp

**Δράστης:** Χρήστης

**Αρχικές συνθήκες:**

Α0α: Ο χρήστης έχει εκτελέσει την εφαρμογή και έχει εμφανιστεί το γραφικό περιβάλλον στην οθόνη του.

Α1α: Ο χρήστης έχει στη κατοχή του εάν αρχείο μέτρησης.

**Βασική ροή:**

Β0α: Ο δράστης επιλέγει από το μενού τη λειτουργία OSC File->UDP/OSC.

Β1α: Ο δράστης δίνει τα στοιχεία της σύνδεσης UDP (IP,Port). Δίνει το ρυθμό δειγματοληψίας και επιλέγει από το γραφικό περιβάλλον το αρχείο που θέλει να κάνει εκπομπή.

Β1β: Η εφαρμογή

Β1β1: πραγματοποιεί σύνδεση στη διεύθυνση UDP και τη κρατάει ενεργή,

Β1β2: ξεκινά την ανάγνωση του αρχείου σύμφωνα με το ρυθμό δειγματοληψίας που έχει δοθεί και

Β1β3: ξεκινά να προωθεί τα μηνύματα.

**Εναλλακτικές ροές:**

Ε1: Αν στο βήμα Β1β1 τα δεδομένα δεν είναι έγκυρα ή η σύνδεση δεν είναι επιτυχής, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β1α.

Ε3: Αν στο βήμα Β1β2 τα μηνύματα δεν έχουν έγκυρη μορφή στο αρχείο, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β1α.

#### 3.2.1.6.6 Ανάγνωση και αναμετάδοση σημάτων από αρχείο σε WebSocket/OSC

**Περίπτωση χρήσης:** Ανάγνωση και αναμετάδοση σημάτων από αρχείο σε WebSocket/OSC

**Ταυτότητα:** uc.fileToWeb

**Δράστης:** Χρήστης

**Αρχικές συνθήκες:**

Α0α: Ο χρήστης έχει εκτελέσει την εφαρμογή και έχει εμφανιστεί το γραφικό περιβάλλον στην οθόνη του.

Α1α: Ο χρήστης έχει στη κατοχή του εάν αρχείο μέτρησης.

**Βασική ροή:**

Β0α: Ο δράστης επιλέγει από το μενού τη λειτουργία OSC File->WebSocket/OSC.

Β1α: Ο δράστης δίνει τα στοιχεία της σύνδεσης WebSocket (IP, Port) και ένα token εάν επιθυμεί επαληθευμένη επικοινωνία στη σύνδεση με WebSocket. Δίνει το ρυθμό δειγματοληψίας και επιλέγει από το γραφικό περιβάλλον το αρχείο που θέλει να κάνει εκπομπή.

Β1β: Η εφαρμογή

Β1β1: πραγματοποιεί σύνδεση στη διεύθυνση WebSocket (προωθεί το token στον διακομιστή εάν αυτό υπάρχει) και αφού επιτραπεί η σύνδεση με το διακομιστή τη κρατάει ενεργή,

Β1β2: ξεκινά την ανάγνωση του αρχείου σύμφωνα με το ρυθμό δειγματοληψίας που έχει δοθεί και

Β1β3: ξεκινά να προωθεί τα μηνύματα.

**Εναλλακτικές ροές:**

Ε1: Αν στο βήμα Β1β1 τα δεδομένα δεν είναι έγκυρα ή η σύνδεση δεν είναι επιτυχής, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β1α.

Ε3: Αν στο βήμα Β1β2 τα μηνύματα δεν έχουν έγκυρη μορφή στο αρχείο, η εφαρμογή εμφανίζει μήνυμα λάθους και επιστρέφει στο βήμα Β1α.

### 3.2.2 Εργαλεία και τεχνολογίες

**Electron.js** (Electron.js, n.d.)– Το Electron.js είναι μια JavaScript βιβλιοθήκη για το Node.js που χρησιμοποιείται για την ανάπτυξη cross-platform εφαρμογών.

**WS** (WS, n.d.)– Το WS είναι μια βιβλιοθήκη για το Node.js που παρέχει μια υλοποίηση του πρωτοκόλλου WebSocket. Παρέχει έναν εύκολο τρόπο για τη δημιουργία τόσο του client όσο και του server.

**OSC.js** (από τον Colin Clark) – Είναι μια JavaScript βιβλιοθήκη που χρησιμοποιείται για την αποστολή και λήψη μηνυμάτων μέσω του πρωτοκόλλου OSC.

**Fs (του Node.js)** – Είναι μέρος του Node.js που παρέχει λειτουργίες για την πρόσβαση και χειρισμό αρχείων και φακέλων.

**Path (του Node.js)** – Είναι μέρος του Node.js που παρέχει λειτουργίες για την επεξεργασία και χειρισμό ονομάτων αρχείων και φακέλων.

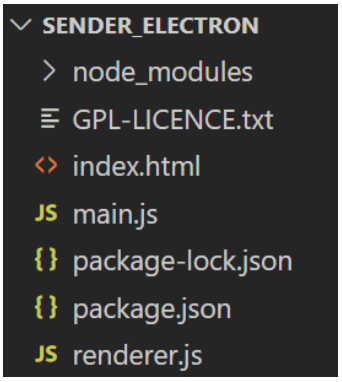
### 3.2.3 Αρχιτεκτονική και δομικά μέρη

Μια τυπική αρχιτεκτονική μιας εφαρμογής Electron.js περιλαμβάνει δύο βασικές διεργασίες:

- Κύρια διεργασία (main process) (main.js)
- Διεργασία απεικόνισης (render process) (renderer.js)

Η κύρια διεργασία είναι υπεύθυνη για τη δημιουργία του παραθύρου της εφαρμογής, την επικοινωνία με το λειτουργικό σύστημα και τις διαδικασίες του, και τη διαχείριση των βασικών πόρων του συστήματος. Η κύρια διεργασία εκτελεί τον κώδικα της εφαρμογής και μπορεί να επικοινωνεί με τη διεργασία απεικόνισης μέσω του μηχανισμού IPC (Inter-Process Communication).

Η διεργασία απεικόνισης είναι υπεύθυνη για την απόδοση του περιεχομένου στο παράθυρο της εφαρμογής. Κάθε παράθυρο της εφαρμογής διαθέτει τη δική του διεργασία απεικόνισης. Η διεργασία απεικόνισης εκτελείται στο περιβάλλον του Chromium και μπορεί να χρησιμοποιεί τις δυνατότητες των HTML, CSS και JavaScript για τη δημιουργία του περιεχομένου του παραθύρου.




*Εικόνα 20 Δομικά μέρη πολυχρηστικής εφαρμογής*

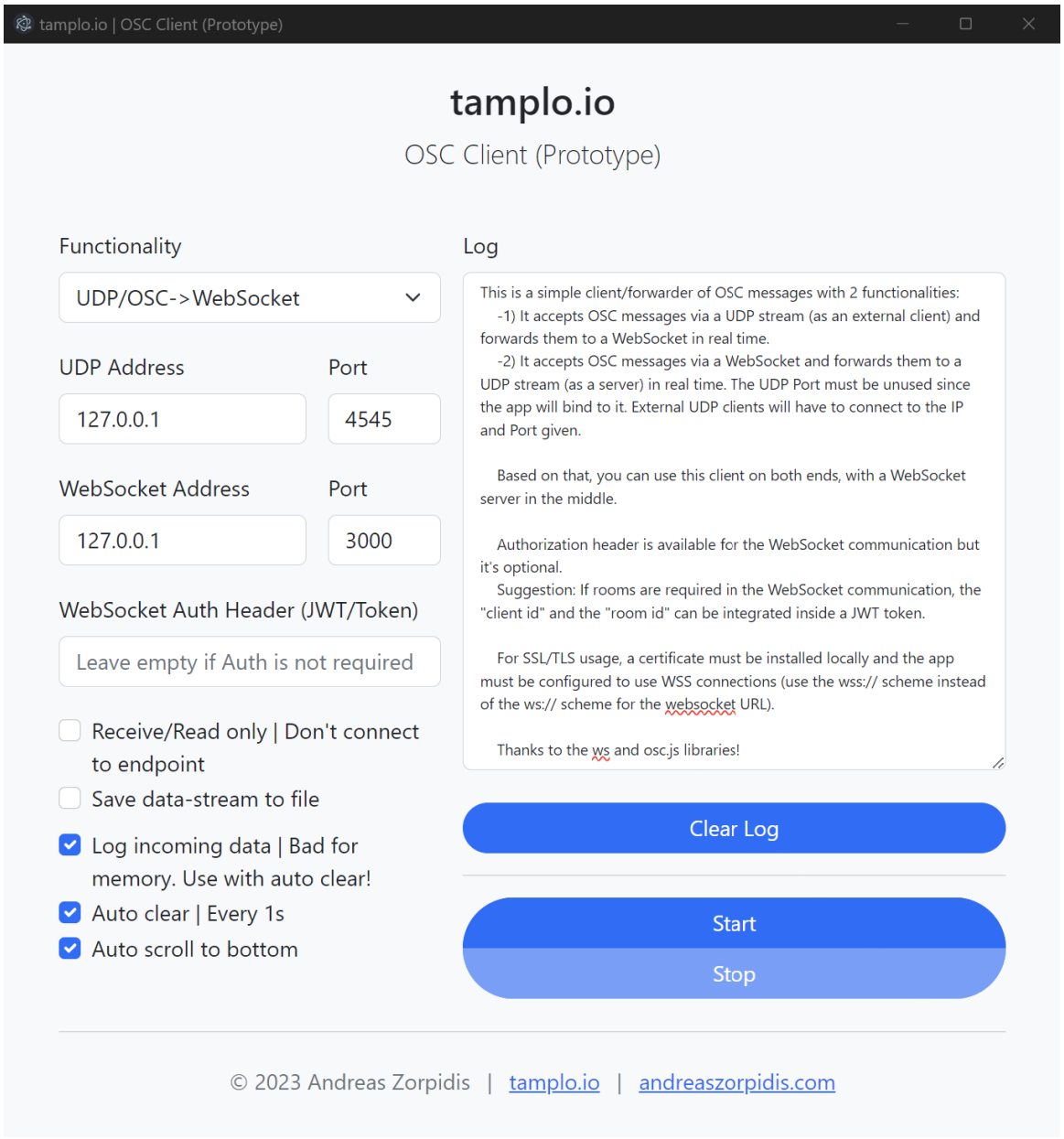
Το αρχείο index.html αποτελεί την κύρια σελίδα της εφαρμογής και φορτώνεται στη διεργασία απεικόνισης. Μέσα σε αυτό μπορεί να γίνει χρήση τεχνολογιών όπως CSS,

JavaScript για τον σχεδιασμό του γραφικού περιβάλλοντος της εφαρμογής και της διαχείρισης των εισαγόμενων δεδομένων αντίστοιχα.

Πιο αναλυτικά, η κύρια σελίδα δημιουργείται με χρήση του αντικειμένου BrowserWindow κατά την κλήση της μεθόδου createWindow() στο αντικείμενο app όταν ενεργοποιείται το ready event. Το αντικείμενο mainWindow επίσης τίθεται δημόσια διαθέσιμο σαν μεταβλητή ώστε να είναι εφικτή η πρόσβαση σε αυτό από τη κύρια διεργασία.

Η κύρια διεργασία επίσης, χρησιμοποιεί το αντικείμενο ipcMain για να παρακολουθεί την ενεργοποίηση τυχόν event που αποστέλλονται από την διεργασία απεικόνισης. Στη συγκεκριμένη υλοποίηση, παρακολουθεί και αναμένει το μήνυμα start-app το οποίο περιέχει διάφορες παραμέτρους που αποστέλλονται από τη διεργασία προβολής. Επιπροσθέτως, η κύρια διεργασία περιέχει την λογική της εφαρμογής για τη διαχείριση των επικοινωνιών UDP και WebSocket, καθώς και τις μεθόδους για την αποθήκευση και αναμετάδοση των μηνυμάτων.

### 3.2.4 Φωτογραφία από την εφαρμογή




*Εικόνα 21 Φωτογραφία της πολυχρηστικής εφαρμογής*

# 4 Δοκιμές και αποτελέσματα

Πραγματοποιήθηκαν με επιτυχία δοκιμές για όλα τα πιθανά σενάρια χρήσης του συστήματος, όπως:

1. Πλήρες σενάριο απομακρυσμένης μετάδοσης

- Προγραμματισμός συνεδρίας
- Σύνδεση συσκευής
- Τοπική αναμετάδοση και μετατροπή σημάτων
- Ανάγνωση και αναμετάδοση σημάτων από UDP/OSC σε WebSocket/OSC
- Ανάγνωση και αναμετάδοση σημάτων από WebSocket/OSC σε UDP/OSC
- Οπτικοποίηση και επεξεργασία σημάτων με χρήση Neuromore

2. Σενάρια αναπαραγωγής σημάτων/προσομοίωσης συσκευής

- Ανάγνωση και αναπαραγωγή αρχείου σε UDP/OSC
- Ανάγνωση και αναπαραγωγή αρχείου σε WebSocket/OSC

3. Σενάρια αλυσιδωτής χρήσης των εργαλείων

- Αναπαραγωγή αρχείου σε UDP/OSC και αναμετάδοση σημάτων σε WebSocket/OSC
- Αναπαραγωγή αρχείου σε WebSocket/OSC και αναμετάδοση σημάτων σε UDP/OSC
- Αναπαραγωγή αρχείου σε UDP/OSC και αναμετάδοση σημάτων σε WebSocket/OSC
- Αναπαραγωγή αρχείου σε WebSocket/OSC και αναμετάδοση σημάτων σε UDP/OSC

# 5 Συμπεράσματα

## 5.1 Ανασκόπηση των κύριων αποτελεσμάτων και συμπερασμάτων της έρευνας

Κατά τη διάρκεια των δοκιμών της εμπειρικής έρευνας αυτής, έγινε ζωντανή μετάδοση και ταυτόχρονη δημιουργία 2 αρχείων με γραμμικά σύνολα δεδομένων (datasets), στα οποία σε κάθε γραμμή αποθηκευόταν ένα μεμονωμένο μήνυμα τύπου OSC όπως αυτό το έστελνε η ίδια η συσκευή. Για τις μετρήσεις χρησιμοποιήθηκε η συσκευή Muse 2 στις οποίες ανά τακτά χρονικά διαστήματα (κάθε 5 δευτερόλεπτα) ο χρήστης άνοιγε ή έκλεινε τα μάτια του αντίστοιχα, συνολικά για 1 λεπτό για κάθε σύνολο δεδομένων.

Έπειτα, πραγματοποιηθήκαν αναπαραστάσεις μετάδοσης των παραπάνω συνόλων δεδομένων με χρήση του client και ξανά αποθήκευση των ροών δεδομένων στο νέφος και στον client του παραλήπτη. Αφετέρου, έγινε σύγκριση των νέων αποθηκευμένων ροών δεδομένων με τα αρχικά σύνολα δεδομένων και δεν παρατηρήθηκε απώλεια δεδομένων, εξαιρουμένων των περιπτώσεων του ανθρώπινου λάθους λόγω καθυστερημένης (μη συγχρονισμένης) έναρξης της συνεδρίας και από τις δύο πλευρές. Ενώ εμπειρικά, δεν παρατηρήθηκαν εμφανείς καθυστερήσεις στην επικοινωνία, ωστόσο συνιστάται η περαιτέρω μελέτη τους.

Επομένως, ο συνδυασμός των πρωτοκόλλων WebSocket και OSC φαίνεται να είναι μια ιδανική επιλογή για την ζωντανή μετάδοση ροών δεδομένων από διεπαφές εγκεφάλου-υπολογιστή, ιδιαίτερα με τη βοήθεια του υπολογιστικού νέφους, ενώ η αποθήκευσή τους και η διαχείρισή τους γίνεται πιο εύκολη με τη χρήση OSC.

## 5.2 Σύγκριση με παρόμοια συστήματα

Κατά τη περίοδο της έρευνας, δε βρέθηκαν ολοκληρωμένες λύσεις ζωντανής απομακρυσμένης μετάδοσης ροών δεδομένων από διεπαφές εγκεφάλου-υπολογιστή μέσω διαδικτύου, παρά μόνο τμηματικά πειράματα από την κοινότητα τα οποία

επέτρεπαν την χρήση των EEG συσκευών ως συσκευές εισόδου απευθείας σε περιηγητές, δηλαδή για τοπική χρήση. Ωστόσο, όσον αφορά το κομμάτι της αποθήκευσης των ροών δεδομένων στο υπολογιστικό νέφος, το λογισμικό κλειστού κώδικα EmotivPRO φάνηκε να παρέχει μια ολοκληρωμένη και εύχρηστη λύση.

Οπότε, σύμφωνα με τα παραπάνω, δεν μπορεί να πραγματοποιηθεί κάποια ουσιαστική σύγκριση με κάποιο άλλο σύστημα.

## 5.3 Περιγραφή των συνεισφορών της έρευνας στον τομέα της διεπαφής εγκεφαλικού υπολογιστή

Η παρούσα έρευνα συνιστά ένα σημαντικό βήμα προς την εξέλιξη του τομέα των διεπαφών εγκεφάλου-υπολογιστή προσφέροντας ένα δείγμα από νέες προοπτικές και λύσεις για την αξιοποίηση των δυνατοτήτων των BCIs χρησιμοποιώντας νέες τεχνολογίες. Προωθεί την τεχνολογική και επιστημονική πρόοδο προς την κατεύθυνση της πρακτικής χρηστικότητας και ευρύτερης υιοθέτησης των BCIs σε πραγματικά σενάρια.

Οι κύριες συνεισφορές είναι οι ακόλουθες:

- Απομακρυσμένη παρακολούθηση και διαχείριση εγκεφαλικής δραστηριότητας μέσω διαδικτύου.
- Αποτελεσματική μετάδοση των σημάτων ανάμεσα στο νέφος και τον πελάτη.
- Ολοκληρωμένο σύστημα αποθήκευσης, αναπαραγωγής και διαχείρισης σημάτων εγκεφαλικής δραστηριότητας.

Η έρευνά που πραγματοποιήθηκε, επιχειρεί να καλύψει μία κενή περιοχή στον τομέα των διεπαφών εγκεφάλου-υπολογιστή προσφέροντας μια λύση που δεν είχε εξερευνηθεί εκτενώς προηγουμένως. Αυτή η συνεισφορά συμβάλλει στην ενίσχυση της συνοχής του τομέα και επιτρέπει την περαιτέρω ανάπτυξη και βελτίωση των τεχνολογιών.

Οι παραπάνω συνεισφορές ανοίγουν το δρόμο για μελλοντική έρευνα και ανάπτυξη προηγμένων διεπαφών εγκεφάλου-υπολογιστή που θα εξυπηρετούν τις ανάγκες της κοινότητας και της κοινωνίας συνολικά.

## 5.4 Προοπτικές

Μελλοντικά, το σύστημα μπορεί να βελτιωθεί εύκολα όσον αφορά το κομμάτι της ασφάλειας με ενσωμάτωση σύγχρονων μεθόδων κρυπτογράφησης των ροών δεδομένων και αποθήκευσης των δεδομένων, ενώ μπορεί να προστεθεί και η δυνατότητα μετατροπής των ροών δεδομένων σε LSL για λόγους ευχρηστίας.

# ΠΑΡΑΡΤΗΜΑ

Εργαλεία που χρησιμοποιήθηκαν ή δημιουργήθηκαν για αυτή την εργασία καθώς και οπτικοακουστικό υλικό που δημιουργήθηκε κατά την εξέλιξη των ερευνών βρίσκονται στους συνδέσμους https://andreaszorpidis.com/ και https://github.com/AndreasZorpidis.

## Α. Δείγματα κώδικα εφαρμογής νέφους (server)

### Α.1 Δημιουργία νέας σύνδεσης με τη βάση δεδομένων

Για τη δημιουργία νέων συνδέσεων του συστήματος με τη βάση δεδομένων δημιουργήθηκε ένα νέο module με ονομασία sequelize.js στον φάκελο config όπως φαίνεται παρακάτω.

```
// config/sequelize.js

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

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

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

```
// app.js

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

## Α.2 Κεντρικό αρχείο συγκέντρωσης μοντέλων και δημιουργίας συσχετίσεων (index.js)

Ενδεικτικά, στο πρώτο τμήμα του παρακάτω κώδικα γίνεται η εισαγωγή των απαραίτητων εξαρτήσεων για το Sequelize. Στο δεύτερο τμήμα γίνεται η συγκέντρωση όλων των μοντέλων του σχήματος της βάσης δεδομένων. Στο τρίτο τμήμα γίνεται η δημιουργία των συσχετίσεών των μοντέλων και τέλος ο κώδικας εξάγει τα μοντέλα ώστε να μπορούν να χρησιμοποιηθούν σε άλλα μέρη της εφαρμογής.

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

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

// Εισαγωγή μοντέλων
const User = require('../../api/models/users')(sequelize, DataTypes);
const UserRole = require('../../api/models/users_roles')(sequelize,
DataTypes);
const Measurement = require('../../api/models/measurements')(sequelize,
DataTypes);

// Δημιουργία συσχετίσεων
User.belongsTo(UserRole, { as: "userrole", foreignKey: 'role', onDelete:
'CASCADE', onUpdate: 'CASCADE' });
UserRole.hasMany(User, { as: "users", foreignKey: 'role', onDelete:
'CASCADE', onUpdate: 'CASCADE' });
Measurement.belongsTo(User, { as: "user", foreignKey: 'username' });
User.hasMany(Measurement, { as: "measurements", foreignKey: 'username' });

// Εξαγωγή μοντέλων
module.exports = {
    User,
    UserRole,
    Measurement
}
```

## Α.3 Δημιουργία μοντέλου μετρήσεων (measurements)

```
// api/models/measurements.js

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

class measurements extends Sequelize.Model {
  static init(sequelize, DataTypes) {
    super.init({
      guid: {
        type: DataTypes.UUID,
        allowNull: false,
        primaryKey: true,
        defaultValue: Sequelize.UUIDV4
      },
      measurementName: {
        type: DataTypes.STRING(64),
        allowNull: false
      },
      measurementDescription: {
        type: DataTypes.STRING(200),
        allowNull: true
      },
      username: {
        type: DataTypes.STRING(32),
        allowNull: false,
        defaultValue: "demo",
        primaryKey: true,
        references: {
          model: {
            tableName: 'users',
          },
          key: 'username'
        }
      },
      createdAt: {
        type: DataTypes.DATE,
        allowNull: false,
        defaultValue: sequelize.fn('current_timestamp')
      },
      updatedAt: {
        type: DataTypes.DATE,
        allowNull: true
      },
      lastConnection: {
```

```
        type: DataTypes.DATE,
        allowNull: true
      },
      status: {
        type: DataTypes.TINYINT,
        allowNull: false,
        defaultValue: 0
      },
      sampleRate: {
        type: DataTypes.FLOAT,
        allowNull: false,
        defaultValue: 0
      },
      scheduleDate: {
        type: DataTypes.DATE,
        allowNull: false
      },
      duration: {
        type: DataTypes.INTEGER,
        allowNull: false,
        validate: {
          min: 1,
          max: 60
        }
      },
      tokenSender: {
        type: DataTypes.STRING(512),
        allowNull: true
      },
      tokenReceiver: {
        type: DataTypes.STRING(512),
        allowNull: true
      }
    }, {
      sequelize,
      tableName: 'measurements'
    });
    return measurements;
  }
}
```

## Α.4 Δημιουργία διαδρομών users.js

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

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

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

// Εισαγωγή βιβλιοθηκών
const Role = require('../middlewares/roles');

// Εισαγωγή βιβλιοθηκών
const authorize = require('../middlewares/authorize');
const authorizeMe = require('../middlewares/authorize_me');

// Δημιουργία διαδρομών

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

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

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

// Προβολή αριθμού εγγεγραμμένων χρηστών
router.get('/count', authorize(Role.Admin),
Controller.User.getAllUsersCount);

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

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

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

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

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

// Εξαγωγή των διαδρομών στο express router
const measurementsRouter = require('./measurements');
router.use('/:username/measurements', measurementsRouter);

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

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

### A.5 Μέθοδοι του ελεγκτή measurements.js

```
// api/controllers/measurements.js

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

// Εξαγωγή μεθόδων ελεγκτή
module.exports = {

    async createMeasurement(req, res, next) {
        try {
            // Validate Request (check if req.body is empty)
            if (Object.keys(req.body).length === 0 && req.body.constructor
=== Object) {
                res.status(400).json({
                    message: "Content can not be empty!"
                });
            } else {
                req.checkBody('measurement_name', 'The measurement name must
be between 4-100 characters long.').len(4, 64);
                req.checkBody('measurement_name', 'The measurement name
field cannot be empty.').notEmpty();
                req.checkBody('measurement_sample_rate', 'The measurement
sample rate field cannot be empty.').notEmpty();
                req.checkBody('measurement_schedule_time', 'The measurement
schedule date field cannot be empty.').notEmpty();
                req.checkBody('measurement_duration', 'The measurement
duration must be between 1-60 minutes.').len(0, 2);
```

```
                req.checkBody('measurement_duration', 'The measurement
duration field cannot be empty.').notEmpty();


                // Set errors if found
                const errors = req.validationErrors();

                if (errors) {
                    console.log(`errors: ${JSON.stringify(errors)}`);

                    // Loop through the errors and remove some attributes
                    var arrayLength = errors.length;
                    for (var i = 0; i < arrayLength; i++) {
                        delete errors[i].location;
                        delete errors[i].value;
                    };

                    // Display the error
                    res.status(401).json({
                        message: "Update failed",
                        errors: errors
                    });

                } else {
                    // Assign input data to variables
                    let _username = req.params.username; // from json input
                    let _measurement_name = req.body.measurement_name; //
from path parameter
                    let _measurement_description =
req.body.measurement_description; // from json input
                    let _measurement_sample_rate =
req.body.measurement_sample_rate; // from json input
                    let _measurement_schedule_time =
req.body.measurement_schedule_time;
                    let _measurement_duration =
req.body.measurement_duration;

                    const created_measurement =
Service.Measurement.createMeasurement(_username, _measurement_name,
_measurement_description, _measurement_sample_rate,
_measurement_schedule_time, _measurement_duration);

                    if (created_measurement) {
                        res.range({
                            first: req.range.first,
                            last: req.range.last,
                            length: 1
                        });
```

```
                        res.status(200).json({ message: "Measurement added"
});
                    }
                    else {
                        res.status(404).json({ message: "The measurement
wasn't added" });
                    }
                };
            }
        }
        catch (e) {
            console.log(e);
            res.status(500).json(e);
        }
    },

    async getMeasurement(req, res, next) {
        try {
            let username = req.params.username;
            let measurementid = req.params.measurement;

            const measurement = await
Service.Measurement.getMeasurement(username, measurementid);

            if (measurement) {
                res.range({
                    first: req.range.first,
                    last: req.range.last,
                    length: measurement.length
                });
                res.status(200).json(measurement);
            }
            else {
                res.status(404).json({ message: "Measurement not found" });
            }
        }
        catch (e) {
            console.log(e);
            res.status(500).json(e);
        }
    },


    async downloadMeasurement(req, res, next) {
        try {
            let username = req.params.username;
            let measurementid = req.params.measurement;

            const path = require('path');
```

```
            // Construct the path to the JSON file on the server
            const filePath = path.join(__dirname, '../..', 'bin',
'datasets', measurementid + '.json');
            const filePathStr = String(filePath);
            // Set the appropriate response headers
            res.setHeader('Content-disposition', 'attachment; filename=' +
measurementid + '.json');
            res.setHeader('Content-type', 'application/json');

            console.log(filePathStr)
            // Use the res.download() method to download the JSON file
            res.download(filePathStr);
        }
        catch (e) {
            console.log(e);
            res.status(500).json(e);
        }
    },

    async getAllMeasurements(req, res, next) {
        try {

            const measurements = await
Service.Measurement.getAllMeasurements();

            if (measurements) {
                res.range({
                    first: req.range.first,
                    last: req.range.last,
                    length: measurements.length
                });
                res.status(200).json(measurements);
            }
            else {
                res.status(404).json({ message: "Measurements not found" });
            }
        }
        catch (e) {
            console.log(e);
            res.status(500).json(e);
        }
    },

    async getAllMeasurementsCount(req, res, next) {
        try {
            const measurements_count = await
Service.Measurement.getAllMeasurementsCount();
            if (measurements_count) {
```

```
                res.range({
                    first: req.range.first,
                    last: req.range.last,
                    length: measurements_count.length
                });
                res.status(200).json(measurements_count);
            }
            else {
                return res.status(404).json({ message: "Devices not found"
});
            }
        }
        catch (e) {
            console.log(e);
            res.status(500).json(e);
        }
    },

    async getAllMeasurementsOfUser(req, res, next) {
        try {
            let username = req.params.username;

            const measurements = await
Service.Measurement.getAllMeasurementsOfUser(username);

            if (measurements) {
                res.range({
                    first: req.range.first,
                    last: req.range.last,
                    length: measurements.length
                });
                res.status(200).json(measurements);
            }
            else {
                res.status(404).json({ message: "Measurements not found" });
            }
        }
        catch (e) {
            console.log(e);
            res.status(500).json(e);
        }
    },

    async deleteMeasurement(req, res, next) {
        try {
            let username = req.params.username;
            let measurementid = req.params.measurement;
```

```
            const deleted_measurement = await
Service.Measurement.deleteMeasurement(username, measurementid);

            if (deleted_measurement) {
                res.range({
                    first: req.range.first,
                    last: req.range.last,
                    length: 1
                });
                res.status(200).json({ message: "Measurement deleted" });
            }
            else {
                res.status(404).json({ message: "Measurement not found" });
            }
        }
        catch (e) {
            console.log(e);
            res.status(500).json(e);
        }
    }
}
```

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

### Α.6 Παράδειγμα «κεντρικού» index αρχείου ελεγκτών

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

// Εισαγωγή ελεγκτών
const User = require('./users');
const Measurement = require('./measurements');

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

## Α.7 Απόσπασμα με μεθόδους της υπηρεσίας measurements.js

```
// api/services/measurements.js

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

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

module.exports = {

    async generateToken(_roomid, _clienttype, notbefore, expiresIn) {
        try {
            const now = Math.floor(Date.now() / 1000);
            const notBeforeInSeconds = Math.floor(new
Date(notbefore).getTime() / 1000);
            const expInSeconds = notBeforeInSeconds + (expiresIn * 60) +
1200; // +20 minutes delay for the users
            const token = jwt.sign(
                {
                    roomid: _roomid,
                    clienttype: _clienttype,
                    iat: now,
                    nbf: notBeforeInSeconds,
                    exp: expInSeconds
                },
                process.env.JWT_KEY
            );
            return token;
        }
        catch (err) {
            console.log(err);
        }
    },

    async createMeasurement(_username, _measurement_name,
_measurement_description, _measurement_sample_rate,
_measurement_schedule_time, _measurement_duration) {

        const measurement = {
            "measurementName": _measurement_name,
            "measurementDescription": _measurement_description,
            "username": _username,
            "sampleRate": _measurement_sample_rate,
            "scheduleDate": _measurement_schedule_time,
            "duration": _measurement_duration
        };
```

```
        try {
            const data = await Model.Measurement.create(measurement);
            const measurementId = data.guid; // get the ID of the created
measurement
            const tokenSender = await this.generateToken(measurementId,
"sender", _measurement_schedule_time, _measurement_duration);
            const tokenReceiver = await this.generateToken(measurementId,
"receiver", _measurement_schedule_time, _measurement_duration);
            // update the measurement with the generated tokens
            await Model.Measurement.update({
                tokenSender,
                tokenReceiver
            }, {
                where: {
                    guid: measurementId
                }
            });
            return data;
        } catch (err) {
            console.log(err);
            /*   return err; */
        }
    },

    getAllMeasurements() {
        return Model.Measurement.findAll({})
            .then(data => {
                return data;
            })
            .catch(err => {
                console.log(err);
                /* return err; */
            });
    },

    getAllMeasurementsOfUser(_username) {
        return Model.Measurement.findAll({ where: { username: _username } })
            .then(data => {
                return data;
            })
            .catch(err => {
                console.log(err);
                /* return err; */
            });
    },

    async getAllMeasurementsCount() {
        const counter = await this.getAllMeasurements();
        return counter.length;
```

```
    },


    async getAllMeasurementsCountOfUser(_username) {
        const counter = await this.getAllMeasurementsOfUser(_username);
        return counter.length;
    },


    deleteMeasurement(_username, _measurementid) {
        return Model.Measurement.destroy({ where: { username: _username,
guid: _measurementid } })
            .then((data) => {
                return data;
            })
            .catch(err => {
                console.log(err);
                /*   return err; */
            });
    },

    getMeasurement(_username, _measurementid) {
        return Model.Measurement.findAll({ where: { username: _username,
guid: _measurementid } })
            .then((data) => {
                return data[0];
            })
            .catch(err => {
                console.log(err);
                /*   return err; */
            });
    },

    getMeasurementById(_measurementid) {
        return Model.Measurement.findAll({ where: { guid: _measurementid }
})
            .then((data) => {
                return data[0];
            })
            .catch(err => {
                console.log(err);
                /*   return err; */
            });
    }

}
```

## Α.8 Απόσπασμα με μεθόδους του authorize.js

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

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

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

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

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

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

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

            // Εξουσιοδότηση βάση ρόλου
            (req, res, next) => {
                if (roles.length && !roles.includes(req.user.role)) {
                    return res.status(403).json({ message: 'Unauthorized'
 });
                }
                next();
            }
        ];
    } catch (e) {
        console.log(e);
        res.status(500).send(e);
    }
}

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

## A.9 Απόσπασμα με μεθόδους του WebSocket server

```
// bin/websocketServer.js

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

// Εισαγωγή εξαρτήσεων
const jwt = require('jsonwebtoken');
const fs = require('fs');
const path = require('path');

const WebSocket = require('ws');
const wss = new WebSocket.Server({ noServer: true });

console.log(`WebSocket server listening on HTTP server's port`);


// Καταγραφή ενεργών δωματίων
```

```
const rooms = {};

// Μέθοδος που δέχεται νέες συνδέσεις
wss.on('connection', async (ws, req) => {
  console.log("WebSocket connection made!");

  // Εξαγωγλη του JWT token από τα headers του request
  const authHeader = req.headers.authorization;

  if (authHeader) {
    const token = authHeader.split(' ')[1];

    // Επαλήθευση του JWT token με βάση το user ID
    try {
      const decoded = jwt.verify(token, process.env.JWT_KEY);
      console.log(decoded);

      const { roomid, clienttype } = decoded;

      const measurement = await
Service.Measurement.getMeasurementById(roomid);

      // Αν πρόκειται για τον πρώτο χρήστη αυτού του δωματίου, τότε
καταγράφεται
      if (!rooms[roomid]) {
        rooms[roomid] = { users: [], messages: [] };
      }

      // Προσθήκη του χρήστη στο δωμάτιο
      const userId = clienttype;
      ws.roomId = roomid;
      ws.userId = userId;
      rooms[roomid].users.push(userId);

      // Δημιουργία αρχείου JSON για τοπική αποθήκευση των μηνυμάτων της
συνεδρίας
      const filename = path.join(__dirname + "/datasets", `${roomid}.json`);
      const writeStream = fs.createWriteStream(filename);

      if (userId === 'sender') {
        console.log(userId)

        // Τερματισμός της συνεδρίας μετά από Χ λεπτά
        const timeout = setTimeout(() => {
          closeRoom(timeout, roomid, writeStream);
        }, measurement.duration * 60 * 1000);
      }

      ws.on('message', message => {
```

```
        if (message) {
          const messageObject = JSON.parse(message);
          if (Object.keys(messageObject).length > 0) {

            // Καταγραφή νέου μηνύματος στο αρχείο
            writeStream.write(`${message}\n`);
          }
        }

      });

      // Τερματισμός συνεδρίας
      ws.on('close', () => {

        if (rooms[roomid]) { // Έλεγχος ύπατξης του δωματίου

          // Αφαίρεση χρήστη από το δωμάτιο
          const index = rooms[roomid].users.indexOf(userId);
          if (index > -1) {
            rooms[roomid].users.splice(index, 1);
          }

          // Διαγραφή του δωματίου όταν δεν είναι κανένας χρήστης
συνδεδεμένος σε αυτό
          if (rooms[roomid].users.length === 0) {
            delete rooms[roomid];
            writeStream.end();
            console.log('WebSocket connection closed.');
          }
        }

      });

      // Διαχείριση σφαλμάτων
    } catch (error) {
      if (error instanceof jwt.TokenExpiredError) {
        console.log('JWT token expired.');
        ws.send(JSON.stringify({ error: 'JWT token expired.' }));
        ws.close();
      } else if (error instanceof jwt.JsonWebTokenError) {
        console.log('Invalid JWT token.');
        ws.send(JSON.stringify({ error: 'Invalid JWT token.' }));
        ws.close();
      } else {
        console.log('Unknown JWT error.');
        console.log(error);

        ws.send(JSON.stringify({ error: 'Unknown JWT error.' }));
```

```
        ws.close();
      }
    }
  } else {
    console.log('JWT token missing.');
    ws.send(JSON.stringify({ error: 'JWT token missing.' }));
    ws.close();
  }

  // Remove all the users from the room and close it.
  function closeRoom(timeout, roomid, writeStream) {

    if (rooms[roomid]) { // Check if room exists
      console.log('trigger****************************************');
      // Remove all users from the room
      rooms[roomid].users.forEach((userId) => {
        wss.clients.forEach((client) => {
          if (client.readyState === WebSocket.OPEN && client.roomId ===
roomid && client.userId === userId) {
            client.close();
          }
        });
      });

      // Remove the room and close the write stream
      delete rooms[roomid];
      writeStream.end();
      console.log('Room closed.');

      // Clear the timeout
      clearTimeout(timeout);

    }
  }


});

// Μέθοδος διαχείρισης σφαλμάτων
wss.on('error', (event) => {
  console.log('WebSocket error: ' + error.message);
})

// Εξαγωγή web socket server
module.exports = wss;
```

## Α.10 Παράδειγμα δημιουργίας διαδρομών index.js

(Με χρήση ελεγκτή που χρησιμοποιεί υπηρεσίες)

```
// routes/index.js

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


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

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

// Δημιουργία διαδρομών

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

// Μέθοδος διαδρομής σελίδας σύνδεσης
router.get('/login', auth.hideFromAuthenticated,
Controller.Users.userLoginPage);

// Μέθοδος διαδρομής που δέχεται τα δεδομένα σύνδεσης
router.post('/login', auth.hideFromAuthenticated,
Controller.Users.userLoginAuthentication, Controller.Users.generateToken);

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

// Μέθοδος διαδρομής που δέχεται τα δεδομένα εγγραφής
router.post('/signup', auth.hideFromAuthenticated,
Controller.Users.userSignup, Controller.Users.generateToken);

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

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

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

## Α.11 Απόσπασμα με μεθόδους των διαδρομών admin.js

(Με απευθείας χρήση υπηρεσίας)

```
// routes/admin.js

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

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

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

// Μέθοδος διαχείρισης ρόλου χρήστη
function getRole(_role) {
    if (_role === "admin" || _role === "Admin") {
        return true;
    } else {
        return null;
    }
    console.log(_role);
};

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

// Μέθοδος διαδρομής /admin/measurements
router.get('/measurements', auth.authenticationCheck, async (req, res) => {
    console.log("Admin Dashboard: Measurements Page");
    const measurements = await
Service.Measurement.getAllMeasurementsOfUser(req.user.username);
    res.render('admin/measurements', { // Απάντηση/επιστροφή με προβολή
        title: "Measurements", //Πεδίο που εισέρχεται στη προβολή
        username: req.user.username, //Πεδίο που εισέρχεται στη προβολή
        user_id: req.user.id, //Πεδίο που εισέρχεται στη προβολή
        admin: getRole(req.user.role), //Πεδίο που εισέρχεται στη προβολή
        data: measurements //Πεδίο που εισέρχεται στη προβολή
    });
```

```
});

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

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

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

### A.12 Απόσπασμα με μεθόδους του ελεγκτή admin.js

```
// controllers/users.js

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

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

// Μέθοδος εγγραφής χρήστη
exports.userSignup = function (req, res, next) {
    // Έλεγχος πεδίων
    req.checkBody('username', 'Username field cannot be empty.').notEmpty();
    req.checkBody('username', 'Username must be between 4-15 characters long
and have no spaces.').len(4, 15);
    req.checkBody('email', 'The email you entered is invalid, please try
again.').isEmail();
    req.checkBody('email', 'Email address must be between 4-100 characters
long, please try again.').len(4, 100);
    req.checkBody('password', 'Password must be between 8-100 characters
long.').len(8, 100);
```

```
    req.checkBody('password', "Password must include one lowercase
character, one uppercase character, a number, and a special
character.").matches(/^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?!.* )(?=.*[^a-zA-Z0-
9]).{8,}$/, "i");
    req.checkBody('conf_password', 'Password must be between 8-100
characters long.').len(8, 100);
    req.checkBody('conf_password', 'Passwords do not match, please try
again.').equals(req.body.password);

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

    if (errors) { // Εμφάνιση των σφαλμάτων αν υπάρχουν
        console.log(`errors: ${JSON.stringify(errors)}`);
        res.render('signup', {
            title: 'Registration Error',
            errors: errors
        });
    } else { // Συνέχεια εγγραφής
        const username = req.body.username;
        const email = req.body.email;
        const password = req.body.password;
        const conf_password = req.body.conf_password;

        const saltRounds = 10;

        // Κατακερματισμός του κωδικού του χρήστη
        bcrypt.hash(password, saltRounds, function (err, hash) {
            // Εισαγωγή του χρήστη στην βάση δεδομένων
            db.query("INSERT INTO  users (username, email, passwd) VALUES
(?, ?, ?)", [username, email, hash],
                function (error, results, fields) {

                    // Έλεγχος σφαλμάτων από τη βάση δεδομένων
                    if (error) {
                        console.log(error);
                        if (error.errno == 1062) {
                            return res.render('signup', {
                                title: 'Registration Error',
                                error: "Username or email are already in
use."
                            });
                        } else {
                            throw error;
                        }
                    }

                    // Δημιουργία νέας συνεδρίας χρήστη στη βάση δεδομένων
                    db.query('SELECT LAST_INSERT_ID() as id', function
(error, results, fields) {
```

```
                        if (error) throw error;
                        const id = results[0];
                        req.login({ id, username }, function (err) {
                            next();

                        })
                    });
                })
        });
    }
};


// Μέθοδος αυθεντικοποίησης χρήστη
exports.userLoginAuthentication = passport.authenticate('local.login', {
    /*     successRedirect: '/admin', */ //Disabled because I added the
authentication as a middleware and then redirect to generateToken service
    failureRedirect: '/login',
    failureFlash: true // optional, see text warning
});

// Μέθοδος προβολής σελίδας σύνδεσης χρήστη
exports.userLoginPage = function (req, res, next) {
    res.render('sign', {
        title: 'Login',
        // https://stackoverflow.com/questions/26403853/node-js-
authentication-with-passport-how-to-flash-a-message-if-a-field-is-missi
        error: req.flash('error')   // Invalid credentials message in login
page
    });
};

// Μέθοδος προβολής σελίδας εγγραφής χρήστη
exports.userSignupPage = function (req, res, next) {
    res.render('signup', {
        title: 'Sign up'
    });
};

// Μέθοδος αποσύνδεσης χρήστη
exports.userLogout = function (req, res) {
    req.logout();
    req.session.destroy();
    res.clearCookie(process.env.JWT_COOKIE_NAME); // Destroy jwt token
cookie
    res.redirect('/');
};
```

```
// Μέθοδος δημιουργίας httpOnly Cookie για την αποθήκευση του JWT Token
κλειδιού του χρήστη
exports.generateToken = async function (req, res, next) {
    const token = await Service.User.generateToken(req.user.username);
    res.cookie(process.env.JWT_COOKIE_NAME, token, {
        expires: new Date(Date.now() +
parseInt(process.env.JWT_COOKIE_LIFE)),
        secure: process.env.NODE_ENV === 'production' ? true : false, // set
to true if your using https
        httpOnly: true // You can't access these tokens in the client's
javascript
    })
        .redirect('/admin');
};
```

### Α.13 Απόσπασμα με μεθόδους του ενδιάμεσου λογισμικού auth.js

```
// middlewares/auth.js

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

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

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

// Δημιουργία νέας στρατηγικής αυθεντικοποίησης για τη βιβλιοθήκη Passport
passport.use('local.login', new LocalStrategy(
    function (username, password, done) {

        //Grab the password from database
        db.query('SELECT UID, PASSWD, ROLE FROM users WHERE USERNAME = ?',
[username],
            function (err, results, fields) {
                if (err) {
```

```
                    done(err);
                };
                if (results.length === 0) {
                    // http://www.passportjs.org/docs/configure/
                    return done(null, false, { message: 'Authentication
failed!' });
                } else {
                    const hash = results[0].PASSWD.toString();

                    bcrypt.compare(password, hash, function (err, response)
{
                        if (response === true) {
                            const id = results[0].ID;
                            const role = results[0].ROLE;
                            // Get user's id and insert into session (sign
him in)
                            return done(null, { id, username, role });
                        } else {
                            return done(null, false, { message:
'Authentication failed!' });
                        }
                    });
                }
            })
    }
));

// Μέθοδος ελέγχου συνεδρίας χρήστη
module.exports.authenticationCheck = (req, res, next) => {
    console.log(`req.session.passport.user:
${JSON.stringify(req.session.passport)}`);
    if (req.isAuthenticated()) return next();
    res.redirect('/login');
}

// Μέθοδος απόκρυψης σελίδων από μη αυθεντικοποιημένους χρήστες
module.exports.hideFromAuthenticated = (req, res, next) => {
    if (req.isAuthenticated()) return res.redirect('/');
    next();
};
```

## Α.14 Απόσπασμα με μεθόδους της προβολής measurements.hbs

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

        <script>

            // Μέθοδος διαγραφής μέτρησης
            async function deleteMeasurement(_measurementid) {
                const username = "{{ username }}";
                await axios.delete(api_url + '/users/' + username +
'/measurements/' + _measurementid, { withCredentials: true })
                    .then(response => {
                        location.reload();
                    })
                    .catch((error) => {
                        console.log('error ' + error.response.data);
                    });
            }

            // Μέθοδος προγραμματισμού νέας μέτρησης
            async function addMeasurement() {
                // Αφαίρεση προηγούμενων σφαλμάτων
                $('#modal_default_add_error').html('');

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

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

                    // Εισαγωγή δεδομένων
                    const username = "{{ username }}";
                    const measurementname =
add_form.add_measurement_name.value;
                    const measurementdescription =
add_form.add_measurement_desc.value;
                    const measurementsamplerate =
add_form.add_measurement_sample_rate.value;
                    const measurementscheduledate =
add_form.add_measurement_schedule_date.value;
                    const measurementduration =
add_form.add_measurement_duration.value;


                    // Δημιουργία αντικειμένου δεδομένων για αποστολή
                    const postdata = {
                        "measurement_name": measurementname,
                        "measurement_description": measurementdescription,
```

```
                        "measurement_sample_rate": measurementsamplerate,
                        "measurement_schedule_time":
measurementscheduledate,
                        "measurement_duration": measurementduration
                    };
                    await axios.post(api_url + '/users/' + username +
'/measurements', postdata, { withCredentials: true })
                        .then(response => {
                            location.reload();
                        })
                        .catch((error) => {
                            //console.log(error.response);
                            $('#modal_default_add_error').html('<div
class="alert alert-danger"><pre>' + JSON.stringify(error.response.data,
null, 2) + '</pre></div>');
                        });
                })
            }

            // Μέθοδος λήψης ολοκληρωμένης μέτρησης
            async function downloadMeasurement(_measurementid) {
                const username = "{{ username }}";
                await axios.get(api_url + '/users/' + username +
'/measurements/download/' + _measurementid, { withCredentials: true,
responseType: 'blob' })
                    .then(response => {
                        const url = window.URL.createObjectURL(new
Blob([response.data]));
                        const link = document.createElement('a');
                        link.href = url;
                        link.setAttribute('download', _measurementid +
'.json');
                        document.body.appendChild(link);
                        link.click();
                    })
                    .catch((error) => {
                        console.log('error ' + error.response.data);
                    });
            }

        </script>
```

## Β. Δείγματα κώδικα πολυχρηστικής εφαρμογής (client)

### Β.1 Περιγραφή

Ξεκινώντας από τη κύρια διεργασία που περιέχει και την λογική της εφαρμογής:

- Στην αρχή του κώδικα γίνεται η εισαγωγή των βιβλιοθηκών και δήλωση των αντικειμένων electron, app, BrowserWindow, ipcMain, osc, WebSocket, fs, και path καθώς και δήλωση των δημόσιων μεταβλητών mainWindow, udp, ws, saveToPath, και saveToFileStream.
- Η μέθοδος createWindow δημιουργεί ένα νέο παράθυρο με βάση τις ιδιότητες που έχουν οριστεί μέσα σε αυτή και φορτώνει το γραφικό περιβάλλον (index.html).
- Το αντικείμενο app χρησιμοποιείται για την διαχείριση διαφόρων events όπως τα ready, window-all-closed και activate σύμφωνα με τις οδηγίες του Electron.js.
- Το αντικείμενο ipcMain χρησιμοποιείται για την επικοινωνία με την διεργασία προβολής και αναμένει συγκεκριμένα μηνύματα από αυτή όπως τις επιλογές του χρήστη.
- Η μέθοδος που διαχειρίζεται τα μηνύματα start-app δέχεται τα ορίσματα του χρήστη δημιουργεί συνδέσεις UDP ή WebSocket σύμφωνα με τις επιλογές του χρήστη και αν έχει οριστεί κάποιο μονοπάτι τότε δημιουργεί ένα νέο φάκελο όπου αποθηκεύει τα μηνύματα από τις συνεδρίες επικοινωνίας.
- Το αντικείμενο udp χρησιμοποιείται για τη δέσμευση ενός udp port τοπικά στη συσκευή το οποίο αναμένει μηνύματα τύπου OSC.
- Το αντικείμενο ws χρησιμοποιείται για τη δημιουργία μιας WebSocket σύνδεσης σε μια συγκεκριμένη διεύθυνση IP.
- Αν είναι ενεργοποιημένη η λειτουργία της αποθήκευση των μηνυμάτων, τότε αποθηκεύονται σε αρχείο.
- Αν είναι ενεργοποιημένη η λειτουργία της καταγραφής των μηνυμάτων, τότε εμφανίζονται στο γραφικό περιβάλλον.
- Η μέθοδος closeConnections διαχειρίζεται τις ενέργειες που πρέπει να γίνουν για τον τερματισμό των επικοινωνιών.
- Στο τέλος του κώδικα, η μέθοδος που διαχειρίζεται τα μηνύματα τερματισμού της εφαρμογής, τερματίζει τις επικοινωνίες.

**Απόσπασμα κώδικα main.js**

```
const electron = require("electron");
const { app, BrowserWindow, ipcMain } = electron;
const osc = require('osc');
const WebSocket = require('ws');
const fs = require('fs');
const path = require('path');

let mainWindow;
let udp;
let ws;

// Exposed globally to the closeConnections function
let saveToPath;
let saveToFileStream;

function createWindow() {
  mainWindow = new BrowserWindow({
    width: 790,
    height: 840,
    webPreferences: {
      nodeIntegration: true,
      contextIsolation: false,
      enableRemoteModule: true,
    },
    autoHideMenuBar: true,
  });

  mainWindow.loadFile('index.html');

  mainWindow.on('closed', () => {
    mainWindow = null;
    closeConnections();
  });
}

app.on('ready', createWindow);

app.on('window-all-closed', function () {
  if (process.platform !== 'darwin') {
    app.quit();
  }
});

app.on('activate', function () {
  if (mainWindow === null) {
```

```
    createWindow();
  }
});

// Listen for "start-app" message from renderer process
ipcMain.on('start-app', function (event, token, option, logData,
receiveOnly, udpAddress, udpPort, wsAddress, wsPort, file, saveTo) {

  saveToPath = saveTo;

  if (saveToPath !== '') {
    // Create the directory if it doesn't exist
    const directoryPath = path.join(saveToPath, 'datasets');
    if (!fs.existsSync(directoryPath)) {
      fs.mkdirSync(directoryPath, { recursive: true });
    }

    // Create the filename based on the current date and time
    const now = new Date();
    const year = now.getFullYear();
    const month = String(now.getMonth() + 1).padStart(2, '0');
    const day = String(now.getDate()).padStart(2, '0');
    const hour = String(now.getHours()).padStart(2, '0');
    const minute = String(now.getMinutes()).padStart(2, '0');
    const file_name = `${year}-${month}-${day}-${hour}-${minute}`;

    // Create the file and write to it
    const filename = path.join(directoryPath, `${file_name}.json`);
    saveToFileStream = fs.createWriteStream(filename);
  }

  // Functionality UDP/OSC->WebSocket
  if (option === "udpToWeb") {...}

  // Functionality WebSocket->UDP/OSC
  if (option === "webToUdp") {... }

  // Functionality File->UDP/OSC
  if (option === "fileToUdp") {... }

  // Functionality File->WebSocket
  if (option === "fileToWeb") {... }

});

function closeConnections() {
  if (udp) {
    udp.close();
    udp = null;
```

```
  }
  if (ws) {
    ws.close();
    ws = null;
  }
  if (saveToPath !== '') {
    saveToFileStream.end();
  }
  mainWindow.webContents.send('button-state', "reset");
}

ipcMain.on('stop-app', () => {
  closeConnections();
});
```

# ΒΙΒΛΙΟΓΡΑΦΙΑ


AdminLTE. (χ.χ.). *AdminLTE*. Ανάκτηση από https://github.com/ColorlibHQ/AdminLTE

Al-Jabri, Ibrahim. (2014). The Perceptions of Adopters and Non-adopters of Cloud Computing: Application of Technology-Organization-Environment Framework. (χ.χ.).

Axios. (χ.χ.). *Axios*. Ανάκτηση από https://github.com/axios/axios

Bcrypt. (χ.χ.). *Bcrypt*. Ανάκτηση από https://github.com/defuse/php-encryption/blob/master/docs/Cryptographically-Secure-Password-Storage.md

Birje, Mahantesh & Challagidad, Praveen & Goudar, R.H. & Tapale, Manisha. (2017). Cloud computing review: Concepts, technology, challenges and security. International Journal of Cloud Computing. 6. 32. 10.1504/IJCC.2017.083905. (χ.χ.).

Body-Parser. (χ.χ.). *Body-Parser*. Ανάκτηση από https://www.npmjs.com/package/body-parser

chill117. (χ.χ.). *express-mysql-session*. Ανάκτηση από https://github.com/chill117/express-mysql-session

Chris S. Crawford and Juan E. Gilbert. 2019. Brains and Blocks: Introducing Novice Programmers to Brain-Computer Interface Application Development. ACM Trans. Comput. Educ. 19, 4, Article 39 (December 2019), 27 pages. https://doi.org/10.1145/3335815. (χ.χ.).

Cookie-Parser. (χ.χ.). *Cookie-Parser*. Ανάκτηση από https://www.npmjs.com/package/cookie-parser

Cors. (χ.χ.). *Cors*. Ανάκτηση από https://www.npmjs.com/package/cors

Dotenv. (χ.χ.). *Dotenv*. Ανάκτηση από https://www.npmjs.com/package/dotenv

Electron.js. (χ.χ.). Ανάκτηση από https://www.electronjs.org/docs

Express. (χ.χ.). *Serving static files in Express*. Ανάκτηση από https://expressjs.com/en/starter/static-files.html

Express. (χ.χ.). *Using template engines with Express*. Ανάκτηση από https://expressjs.com/en/guide/using-template-engines.html

express-jwt. (χ.χ.). *express-jwt*. Ανάκτηση από https://www.npmjs.com/package/express-jwt

Express-Validator. (χ.χ.). *Getting Started*. Ανάκτηση από https://express-validator.github.io/docs/

Garrett Flynn, Joshua Brewster, Dong Song, and Marientina Gotsis. 2023. Developing Brain-Computer Interfaces with Everyone. In Proceedings of the 16th International Conference on PErvasive Technologies Related to Assistive Environments (PETRA '23). Associ. (χ.χ.).

Google. (χ.χ.). *Chrome DevTools*. Ανάκτηση από https://developers.google.com/web/tools/chrome-devtools

hbs. (χ.χ.). *hbs*. Ανάκτηση από https://www.npmjs.com/package/hbs

Javaid, Adeel. (2013). Brain-Computer Interface. SSRN Electronic Journal. 10.2139/ssrn.2386900. (χ.χ.).

JWT.IO. (χ.χ.). *Introduction to JSON Web Tokens*. Ανάκτηση από https://jwt.io/introduction/

Kadwill, T. (χ.χ.). *Using Sequelize ORM with Node.js and Express*. Ανάκτηση από https://stackabuse.com/using-sequelize-orm-with-nodejs-and-express/

LabStreamingLayer. (χ.χ.). Ανάκτηση από https://github.com/sccn/labstreaminglayer: https://github.com/sccn/labstreaminglayer

Microsoft Ignite. (χ.χ.). *Βασικά στοιχεία Git και GitHub για το Docs*. Ανάκτηση από https://docs.microsoft.com/el-gr/contribute/git-github-fundamentals

Node.js. (χ.χ.). *What is npm*. Ανάκτηση από https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/

Passport. (χ.χ.). *Passport*. Ανάκτηση από https://github.com/jaredhanson/passport

Pierce Stegman, Chris Crawford, and Jeff Gray. 2018. WebBCI: An Electroencephalography Toolkit Built on Modern Web Technologies. In Augmented Cognition: Intelligent Technologies: 12th International Conference, AC 2018, Held as Part of HCI International 20. (χ.χ.).

Postman. (χ.χ.). *Postman*. Ανάκτηση από https://www.postman.com/

Ramadan, Rabie & Refat, Samah & Elshahed, Marwa & Ali, Rasha. (2015). Basics of Brain Computer Interface. Intelligent Systems Reference Library. 74. 31-50. 10.1007/978-3-319-10978-7_2. (χ.χ.).

RESTfulAPI.net. (χ.χ.). *RESTfulAPI.net*. Ανάκτηση από https://restfulapi.net/

Sequelize. (χ.χ.). *Sequelize ORM*. Ανάκτηση από https://sequelize.org/

Sequelize-Auto. (χ.χ.). *Sequelize-Auto*. Ανάκτηση από https://www.npmjs.com/package/sequelize-auto

tutorialspoint. (χ.χ.). *ExpressJS - Sessions*. Ανάκτηση από https://www.tutorialspoint.com/expressjs/expressjs_sessions.htm

Visual Studio Code. (χ.χ.). *Visual Studio Code FAQ*. Ανάκτηση από https://code.visualstudio.com/docs/supporting/faq

Wikipedia. (χ.χ.). *Google Chrome*. Ανάκτηση από https://el.wikipedia.org/wiki/Google_Chrome

Wikipedia. (χ.χ.). *HeidiSQL*. Ανάκτηση από https://en.wikipedia.org/wiki/HeidiSQL

Wikipedia. (χ.χ.). *Node.js*. Ανάκτηση από https://el.wikipedia.org/wiki/Nodejs

WS. (χ.χ.). *WS*. Ανάκτηση από https://github.com/websockets/ws

Expressjs.com. (χ.χ.). *Node.js - Express Framework*. Ανάκτηση από https://expressjs.com/