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

Εφαρμογή IoT για την υποστήριξη ηλικιωμένων και ατόμων με ειδικές ανάγκες σε PuckJS και κινητά, μέσω ανίχνευσης πτώσης και λειτουργίας κουμπιού πανικού

Διπλωματική Εργασία

Μπρατσιώτης Γεώργιος Γεράσιμος

Επιβλέπων: Παναγιώτης Τσανάκας
Καθηγητής ΕΜΠ

Αθήνα, Σεπτέμβριος 2024

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

Εφαρμογή IoT για την υποστήριξη ηλικιωμένων και ατόμων με ειδικές ανάγκες σε PuckJS και κινητά, μέσω ανίχνευσης πτώσης και λειτουργίας κουμπιού πανικού

Διπλωματική Εργασία

Μπρατσιώτης Γεώργιος Γεράσιμος

Επιβλέπων: Παναγιώτης Τσανάκας
Καθηγητής ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 30η Οκτωβρίου 2024

…….	…….	…….
Π. Τσανάκας	Δ. Σούντρης	Γ. Ματσόπουλος
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2024

Γεώργιος Γεράσιμος Μπρατσιώτης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η παρούσα διπλωματική εργασία επικεντρώνεται στην ανάπτυξη ενός συστήματος ανίχνευσης πτώσεων και πανικού για ηλικιωμένους, άτομα με αναπηρίες και ασθενείς, βασισμένο στη συσκευή Puck.js. Το σύστημα ανιχνεύει πτώσεις χρησιμοποιώντας δεδομένα επιταχυνσιόμετρου και ένα πολυεπίπεδο νευρωνικό δίκτυο (MLP), ενώ ταυτόχρονα λειτουργεί και ως κουμπί πανικού. Όταν ανιχνεύεται πτώση ή ενεργοποιείται το κουμπί πανικού, τα δεδομένα μεταφέρονται μέσω Bluetooth Low Energy (BLE) στη συνοδευτική εφαρμογή, η οποία με τη σειρά της τα προωθεί στον εξυπηρετητή για περαιτέρω επεξεργασία και αποστολή ειδοποιήσεων σε έμπιστες επαφές και φροντιστές.

Το σύστημα διαθέτει επίσης τη δυνατότητα επικοινωνίας μεταξύ συσκευών Puck.js, όταν δεν υπάρχει άμεση σύνδεση με την εφαρμογή. Σε αυτήν την περίπτωση, οι συσκευές Puck.js μεταφέρουν μεταξύ τους τα δεδομένα και τελικά η πρώτη που συνδέεται με τη συνοδευτική εφαρμογή αποστέλλει όλα τα μηνύματα που έχουν συλλεχθεί.

Η υποδομή του backend διαχειρίζεται τις λειτουργίες αυθεντικοποίησης, αποθήκευσης δεδομένων και αποστολής ειδοποιήσεων μέσω email, ενώ χρησιμοποιήθηκαν τόσο δημόσια datasets όσο και προσωπικές μετρήσεις για την εκπαίδευση του νευρωνικού δικτύου. Το σύστημα παρέχει αναγνώριση πτώσεων και άμεση ειδοποίηση προς έμπιστες επαφές, διευκολύνοντας την έγκαιρη παροχή βοήθειας.

Λέξεις Κλειδιά

Νευρωνικά Δίκτυα, Multilayer Perceptron, Bluetooth, ανίχνευση πτώσης, επιταχυνσιόμετρο, κουμπί πανικού, PuckJS, Javascript, Flutter, Sendgrid, εφαρμογή κινητού, ειδοποίηση φροντιστών

Abstract

This thesis focuses on the development of a fall detection and panic button system for the elderly, people with disabilities, and patients, utilizing the Puck.js device. The system detects falls using accelerometer data and a multilayer perceptron (MLP) neural network, while also functioning as a panic button. When a fall is detected or the panic button is pressed, the data is transferred via Bluetooth Low Energy (BLE) to the companion app, which forwards it to the backend for further processing and email notifications to trusted contacts and caregivers.

The system also supports Puck-to-Puck communication when no direct connection to the companion app is available. In such cases, Puck.js devices relay data among themselves, and the first device to connect to the app sends all collected messages.

The backend infrastructure manages authentication, data storage, and email notifications. Both public datasets and personal measurements were used to train the neural network. The system ensures fall detection and immediate notifications to trusted contacts, facilitating timely assistance in emergency situations.

Keywords

Neural Networks, Multilayer Perceptron, Bluetooth, fall detection, accelerometer, panic button, PuckJS, Javascript, Flutter, Sendgrid, smartphone app, caregiver notification

Ευχαριστίες

Με την ολοκλήρωση αυτής της διπλωματικής εργασίας, θα ήθελα να εκφράσω την ειλικρινή μου ευγνωμοσύνη προς όλους εκείνους που με στήριξαν και συνέβαλαν σε αυτήν την πορεία.

Πρώτα απ’ όλα, θα ήθελα να ευχαριστήσω τον κύριο Π. Τσανάκα για την πολύτιμη καθοδήγηση, την αδιάκοπη υποστήριξη και την ουσιαστική συμβολή του σε κάθε στάδιο της εργασίας μου. Οι συμβουλές του και οι τακτικές συναντήσεις μας προκειμένου να παρακολουθεί και να καθοδηγεί την πορεία της εργασίας μου, αποτέλεσαν ακρογωνιαίο λίθο για την επιτυχή ολοκλήρωση αυτής.

Ιδιαίτερες ευχαριστίες οφείλω επίσης στον διδακτορικό φοιτητή, Θ. Παπαρηγόπουλο, για τη συνεχή βοήθεια και την εξαιρετική συνεργασία που είχαμε καθ' όλη τη διάρκεια αυτής της διαδικασίας. Η συνεισφορά του ήταν ανεκτίμητη και η καθοδήγησή του καίρια.

Ευχαριστώ θερμά και τους καθηγητές μου, τον κύριο Δ. Σούντρη και τον κύριο Γ. Ματσόπουλο**,** για τη συμμετοχή τους στην τριμελή επιτροπή και για το χρόνο που αφιέρωσαν στην αξιολόγηση της εργασίας μου.

Επιπρόσθετα, ευχαριστίες αξίζουν στους φίλους και τη σύντροφό μου, που με στήριξαν, με ενθάρρυναν και μου προσέφεραν την αγάπη, την παρέα και τη φιλία τους καθ’ όλη τη διάρκεια αυτής της διαδρομής. Η παρουσία τους υπήρξε ανεκτίμητη και με βοήθησε να ανταπεξέλθω σε κάθε δυσκολία.

Τις βαθύτερες ευχαριστίες και την απέραντη ευγνωμοσύνη μου αξίζουν οι γονείς μου, ο αδελφός μου και η οικογένειά μου, για τη συνεχή τους στήριξη, την ακούραστη ενθάρρυνση και την αμέριστη αγάπη τους. Η πίστη τους σε μένα και η ηθική και υλική τους υποστήριξη ήταν οι πυλώνες που με κράτησαν σταθερό σε αυτή την πορεία. Από τους προαναφερθέντες δεν παραλείπω να ευχαριστήσω τους ανθρώπους που δεν είναι πια δίπλα μας, αλλά η παρουσία τους στη ζωή μου έχει αφήσει ανεξίτηλο σημάδι. Η σκέψη και η μνήμη τους με συντροφεύουν σε κάθε μου βήμα και η επιρροή τους ήταν και θα είναι πάντα μαζί μου, αποτελώντας πηγή έμπνευσης και δύναμης.

Σας ευχαριστώ όλους από καρδιάς.

Στην οικογένειά μου

Περιεχόμενα

Ευρετήριο Εικόνων

Λίστα Ακρωνυμίων

ΣΚΠ:	Σκλήρυνση Κατά Πλάκας
JS:	JavaScript
MLP:	Multilayer Perceptron
NN	Neural Network
ANN	Artificial Neural Network
SNN	Simulated Neural Network
API:	Application Programming Interface
IDE:	Integrated Development Environment
BLE:	Bluetooth Low Energy
PAN:	Personal Area Network
GATT:	Generic Attribute Profile
SIG:	Special Interest Group
NFC:	Near Field Communication
SQL:	Structured Query Language
DB:	Database
RDBMS:	Relational Database Management System
DDL:	Data Definition Language
ORM:	Object-Relational Mapping
DTO:	Data Transfer Object
GHz:	Giga-Hertz
MB:	Mega-Byte
IR:	Infrared
UHF:	Ultra High Frequency
ISM:	Industrial, Scientific and Medical
kB:	Kilo-Byte
iOS:	iPhone Operating System
UI:	User Interface
JSON:	JavaScript Object Notation
JWT:	JSON Web Token
LED:	Light-Emitting Diode
CPU:	Central Processing Unit
RAM:	Random Access Memory
RISC:	Reduced Instruction Set Computer
ARM:	Advanced RISC Machines
SoC:	System-on-chip
IoT:	Internet of Things
PCB:	Printed Circuit Board
PWM:	Pulse-Width Modulation
ADC:	Analog-to-Digital Converter
UART:	Universal Asynchronous Receiver/Transmitter
I2C:	Inter-Integrated Circuit
SPI:	Serial Peripheral Interface
I/O:	Input/Output
NPM:	Node Package Manager
UUID:	Universally Unique Identifier

ID:	identity
SDK:	Software Development Kit
HTTP(S):	Hypertext Transfer Protocol (Secure)
REST:	Representational State Transfer
CRUD:	Create, Read, Update, Delete
SOA:	Service Oriented Architecture
HTML:	Hypertext Markup Language
NumPy:	Numeric Python
DKIM:	DomainKeys Identified Mail
SPF:	Sender Policy Framework
TLS:	Transport Layer Security

Κεφάλαιο 1: Εισαγωγή, Ιστορικό και Κίνητρα

1.1 Νοσηρότητα Πτώσεων και ομάδες υψηλού κινδύνου

Η χρήση τεχνολογιών για την προστασία της υγείας και της ασφάλειας των ηλικιωμένων και ατόμων με αναπηρίες έχει αναδειχθεί ως κρίσιμος τομέας έρευνας και ανάπτυξης, ειδικά λόγω της αυξανόμενης γήρανσης του πληθυσμού σε παγκόσμιο επίπεδο.

Η πτώση ορίζεται ως ένα συμβάν που έχει ως αποτέλεσμα ένα άτομο να καταλήξει ακούσια στο έδαφος, στο πάτωμα ή σε άλλο χαμηλότερο επίπεδο. Οι τραυματισμοί που σχετίζονται με πτώσεις μπορεί να είναι θανατηφόροι ή μη θανατηφόροι, αν και οι περισσότεροι είναι μη θανατηφόροι. Σύμφωνα με τον Παγκόσμιο Οργανισμό Υγείας (WHO), οι πτώσεις είναι η δεύτερη κύρια αιτία θανάτου από ακούσιο τραυματισμό παγκοσμίως. Κάθε χρόνο υπολογίζεται ότι 684.000 άτομα πεθαίνουν από πτώσεις παγκοσμίως, εκ των οποίων πάνω από το 80% βρίσκεται σε χώρες με χαμηλό και μεσαίο εισόδημα, ενώ εκτιμάται ότι λαμβάνουν χώρα επιπλέον 37,3 μη θανάσιμες αλλά σοβαρές πτώσεις που χρήζουν ιατρικής φροντίδας[1].

Οι ενήλικες άνω των 65 ετών αποτελούν την πιο ευάλωτη ομάδα, καθώς το ένα τρίτο των ατόμων ηλικίας άνω των 65 ετών που διαμένουν στην κοινότητα και το 60% των ατόμων σε γηροκομεία και οίκους ευγηρίας υφίστανται πτώσεις κάθε χρόνο. Οι πτώσεις ευθύνονται για πάνω από το 80% των εισαγωγών στο νοσοκομείο λόγω τραυματισμών σε άτομα ηλικίας άνω των 65 ετών. Στις Ηνωμένες Πολιτείες της Αμερικής, το 20–30% των ηλικιωμένων που πέφτουν υφίστανται μέτριους έως σοβαρούς τραυματισμούς, όπως μώλωπες, κατάγματα ισχίου ή τραύματα στο κεφάλι. Στους ηλικιωμένους πληθυσμούς οι πτώσεις αποτελούν μία από τις πιο κοινές αιτίες συνδρόμων χρόνιου πόνου, λειτουργικής ανεπάρκειας και περιορισμών, εξαρθρώσεων, καταγμάτων, αναπηρίας και σοβαρών τραυματισμών στους μαλακούς ιστούς, δημιουργώντας ένα τεράστιο οικονομικό βάρος στο σύστημα υγείας, καθώς και υψηλή θνησιμότητα. Επιπλέον, εκτιμάται ότι περίπου το 50% των ατόμων που υφίστανται πτώση, θα ξαναπέσουν επανειλημμένως, γεγονός που αυξάνει τον κίνδυνο σοβαρών τραυματισμών και την ανάγκη για επαναλαμβανόμενη ιατρική φροντίδα. Η σοβαρότητα των τραυματισμών είναι σημαντικά υψηλότερη για τις γυναίκες, καθώς τα ποσοστά σοβαρών τραυματισμών, όπως τα κατάγματα, μπορεί να είναι περισσότερο από διπλάσια για γυναίκες ηλικίας άνω των 75 ετών σε σύγκριση με τους άνδρες της ίδιας ηλικίας [1], [2], [3].

Η πρόληψη πτώσεων και τραυματισμών δεν είναι εύκολη, καθώς προκαλούνται από έναν συνδυασμό εγγενών βλαβών και αναπηριών (δηλαδή, αυξημένη ευπάθεια πτώσης), με ή χωρίς συνοδευτικούς περιβαλλοντικούς κινδύνους (δηλαδή αυξημένη ευκαιρία πτώσης) [2]. Η αύξηση δηλαδή του κινδύνου πτώσης μπορεί να οφείλεται σε σωματικές, αισθητηριακές και γνωστικές αλλαγές, παρενέργειες φαρμάκων, σωματική αδράνεια και απώλεια ισορροπίας που σχετίζονται με τη γήρανση, σε συνδυασμό με περιβάλλοντα που δεν είναι προσαρμοσμένα στις ανάγκες μιας γηράσκουσας πληθυσμιακής ομάδας[1].

Μία ακόμη πολύ ευπαθής ομάδα είναι τα άτομα που πάσχουν από νευρολογικές ασθένειες, όπως η Σκλήρυνση Κατά Πλάκας (ΣΚΠ). Σε σύγκριση με το ποσοστό

πτώσεων 30% στον γενικό πληθυσμό των ηλικιωμένων που αναφέρεται στη βιβλιογραφία, το 56% των ατόμων με ΣΚΠ υφίστανται πτώσεις τουλάχιστον μία φορά ανά τρίμηνο, με το 37% των οποίων να παρουσιάζει συχνές πτώσεις (δύο περισσότερες σε ένα τρίμηνο). Τα ποσοστά τραυματισμών που αποδίδονται σε πτώσεις σε αυτή την πληθυσμιακή ομάδα κυμαίνονται από 11% έως 58%. Τα άτομα με ΣΚΠ εμφανίζουν μια σειρά από συμπτώματα που αυξάνουν τον κίνδυνο πτώσης, όπως διαταραχές ισορροπίας, γνωστικής λειτουργίας και όρασης [4].

Όπως καταδεικνύουν κάποιες άλλες μελέτες, σημαντικό κίνδυνο πτώσης εμφανίζουν και τα άτομα με νοητική αναπηρία, στα οποία παρατηρείται τριπλάσιο ποσοστό πτώσεων σε σύγκριση με το γενικό πληθυσμό, 12,3% έναντι 4,3%, ενώ ορισμένες μελέτες καταλήγουν σε ποσοστό πτώσεων έως και 34% σε διάστημα 12 μηνών, με μέσο όρο 7,21 πτώσεις ανά άτομο, σε αυτή την πληθυσμιακή ομάδα. Η ευπάθεια αυτή εμφανίζεται ανεξαρτήτως της ηλικίας και αφορά ακόμα και τα νέα άτομα. [5], [6]

1.2 Σημασία της Αμεσότητας στην Ειδοποίηση για παροχή βοήθειας

Η ανάγκη για άμεση ανταπόκριση και ιατρική παρέμβαση μετά από πτώση ή επείγον είναι ζωτικής σημασίας για την πρόληψη σοβαρών επιπλοκών και τη μείωση του κινδύνου θανάτου. Η καθυστέρηση στην παροχή βοήθειας μπορεί να οδηγήσει σε αυξημένα ποσοστά θνησιμότητας, σοβαρές αναπηρίες, και παρατεταμένη νοσηλεία. Παράλληλα, άτομα με αναπηρίες και χρόνιες ασθένειες μπορεί να χρειαστούν άμεση βοήθεια σε περιπτώσεις έκτακτης ανάγκης, καθιστώντας την χρήση ενός κουμπιού πανικού απαραίτητη.

Σύμφωνα με μελέτες, το 80% των πτώσεων συμβαίνουν όταν το άτομο βρίσκεται μόνο του και το 30% των ηλικιωμένων που πέφτουν και δεν μπορούν να σηκωθούν μόνοι τους, παραμένουν στο έδαφος για περισσότερο από μία ώρα πριν λάβουν βοήθεια, μία κατάσταση η οποία συχνά χαρακτηρίζεται με τον όρο «παρατεταμένη κατάκλιση» (long lie). Όπως έχει διαπιστωθεί, η παρατεταμένη κατάκλιση δεν επηρεάζει μόνο τους ηλικιωμένους που έχουν πέσει και δεν μπορούν να σηκωθούν μόνοι τους λόγω της αδυναμίας τους, αλλά και άτομα με περιορισμένη κινητικότητα, η οποία μπορεί να σχετίζεται με διάφορους λόγους [7]. Αυτή η καθυστέρηση σχετίζεται με αυξημένα ποσοστά επιπλοκών, όπως αφυδάτωση, κατακλίσεις (που συχνά επιδεινώνονται από την αναπόφευκτη ακράτεια), υποθερμία, πνευμονία, σηψαιμία, λοιμώξεις, ραβδομυόλυση, σοβαρή μυϊκή αδυναμία και απώλεια συνείδησης, που μπορούν να οδηγήσουν σε αυξημένη διάρκεια νοσηλείας ή ακόμα και σε θάνατο [7], [8], [9], [10]. Ορισμένες μελέτες δείχνουν ότι οι ηλικιωμένοι που παραμένουν αβοήθητοι για πάνω από μία ώρα μετά από πτώση έχουν έως και πενταπλάσια πιθανότητα θανάτου εντός των επόμενων έξι μηνών σε σύγκριση με εκείνους που λαμβάνουν βοήθεια άμεσα, αφού αυτοί που λαμβάνουν άμεση ιατρική φροντίδα μετά από πτώση έχουν 30% υψηλότερες πιθανότητες να επιστρέψουν στην κανονική τους δραστηριότητα σε σύγκριση με εκείνους που παραμένουν αβοήθητοι για μεγάλο χρονικό διάστημα.

Εκτός από τις σοβαρές σωματικές επιπτώσεις, οι εργαζόμενοι στην κοινότητα και οι ψυχολόγοι έχουν αναφέρει ότι τα άτομα που έχουν πέσει είναι ψυχολογικά τραυματισμένα λόγω της κατάστασης. Ιδιαίτερα τα άτομα που δεν λαμβάνουν άμεση βοήθεια μετά από πτώση είναι πιο πιθανό να αναπτύξουν φόβο για μελλοντικές πτώσεις, ο οποίος μπορεί να οδηγήσει σε απομόνωση, μειωμένη σωματική δραστηριότητα και χειροτέρευση της συνολικής τους υγείας και ποιότητας ζωής [7].

1.3 Στόχος της εφαρμογής

Όπως αναλύθηκε παραπάνω, η άμεση παροχή βοήθειας σε έκτακτα περιστατικά και σε περιστατικά πτώσεων είναι ζωτικής σημασία για την αποκατάσταση της υγείας των παθόντων και την αποφυγή των διαφόρων επιπλοκών που συνοδεύουν συνήθως τέτοια γεγονότα. Η τεχνολογία, όπως τα συστήματα ανίχνευσης πτώσεων και τα κουμπιά πανικού, προσφέρει μία λύση, αφού έχει τη δυνατότητα να εξασφαλίσει ότι οι ειδοποιήσεις θα σταλούν άμεσα σε έμπιστες επαφές ή ιατρικές υπηρεσίες, ελαχιστοποιώντας τις καθυστερήσεις και αυξάνοντας τις πιθανότητες επιβίωσης και πλήρους ανάρρωσης. Η πρόοδος στην τεχνολογία αισθητήρων, ειδικά σε φορητές συσκευές όπως το Puck.js που εξετάζεται στο παρόν έργο, επιτρέπει την ανάπτυξη τέτοιων οικονομικά προσιτών και μικρών σε μέγεθος συστημάτων, τα οποία ενσωματώνουν προηγμένους αλγορίθμους για την κατά το δυνατόν ακριβή ανίχνευση πτώσης και την άμεση ανταπόκριση για παροχή βοήθειας. Ο σκοπός λοιπόν της παρούσας εργασίας είναι η ανάπτυξη ενός ολοκληρωμένου συστήματος ανίχνευσης πτώσεων και αποστολής ειδοποιήσεων για ηλικιωμένους, άτομα με αναπηρίες ή ασθενείς, το οποίο θα είναι εύχρηστο, οικονομικό και ενεργειακά αποδοτικό, ώστε να είναι προσιτό από όλους τους χρήστες. Ώστε να μπορεί να υποστηρίξει την ανθρώπινη υγεία αλλά και το σύστημα υγείας, διασφαλίζοντας αμεσότητα στην ειδοποίηση για παροχή βοήθειας σε έκτακτες καταστάσεις και γεγονότα πτώσεων, και άρα μειώνοντας τις επιπλοκές και τη θνησιμότητα που επιφέρουν οι καθυστερήσεις.

1.4 Καινοτομία

Η καινοτομία του παρόντος έργου εντοπίζεται σε τρεις άξονες.

Κατ’ αρχάς, πολλές υπάρχουσες εφαρμογές επικεντρώνονται σε στατιστικές προσεγγίσεις και μαθηματικές φόρμουλες βασισμένες σε κατώφλια, εμφανίζοντας υψηλό ποσοστό ψευδών θετικών [11]. Οι τελευταίες ερευνητικές τάσεις στα συστήματα ανίχνευσης και πρόληψης πτώσεων αξιοποιούν τη Μηχανική Μάθηση, ενώ και κάποιες εταιρείες έχουν αρχίσει να στρέφονται προς εκείνη την κατεύθυνση. Έτσι, η εργασία αυτή κάνει χρήση πολυεπίπεδων νευρωνικών δικτύων (Multilayer Perceptrons - MLPs) για την ανίχνευση πτώσεων. Σε αντίθεση με παραδοσιακές μεθόδους που βασίζονται σε προκαθορισμένα όρια επιτάχυνσης, το νευρωνικό δίκτυο εκπαιδεύεται σε δεδομένα επιτάχυνσης και βελτιώνει την ακρίβεια της ανίχνευσης, μαθαίνοντας μοτίβα που αντιστοιχούν σε πτώσεις. Αυτή η προσέγγιση συνεισφέρει στην αύξηση της αξιοπιστίας του συστήματος, καθώς το δίκτυο μπορεί να αναγνωρίσει πιο σύνθετες και ποικίλες κινήσεις.

Η δεύτερη καινοτόμος πτυχή του συστήματος, είναι η δυνατότητα επικοινωνίας πολλαπλών Puck.js συσκευών μεταξύ τους μέσω Bluetooth. Στην περίπτωση που κάποια συσκευή δεν μπορεί να βρει τη συνοδευτική εφαρμογή για να στείλει την ειδοποίηση, η συσκευή θα μεταδώσει το σήμα σε άλλες κοντινές συσκευές Puck.js, οι οποίες θα αναλάβουν να στείλουν το μήνυμα. Αυτό το χαρακτηριστικό βελτιώνει την αξιοπιστία του συστήματος, καθώς δεν βασίζεται αποκλειστικά στη σύνδεση με την εφαρμογή. Κατά τη βιβλιογραφική έρευνα δεν εντοπίστηκε κάποιο σύστημα που νας λειτουργεί με παρόμοιο τρόπο.

Τέλος, μία ακόμη καινοτομία της εργασίας είναι ότι αξιοποιεί το Puck.js για την ανίχνευση πτώσεων. Το Puck.js είναι μια μικρή και οικονομικά προσιτή φορητή συσκευή, η οποία προσφέρει μια πιο εύκολη και ευέλικτη λύση σε σχέση με παραδοσιακές φορητές συσκευές, δεδομένης της δυνατότητας προγραμματισμού του σε JavaScript μέσω του Espruino Web IDE, η οποία μάλιστα είναι και ανοιχτού κώδικα, σε αντίθεση με άλλες υπάρχουσες λύσεις που στηρίζονται σε συσκευές εταιρειών.

Κεφάλαιο 2: PuckJS και εργαλεία ανάπτυξης

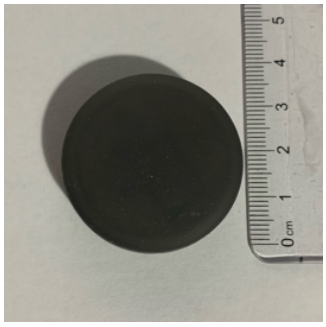
2.1 Το Puck.js

2.1.1 Γενικά

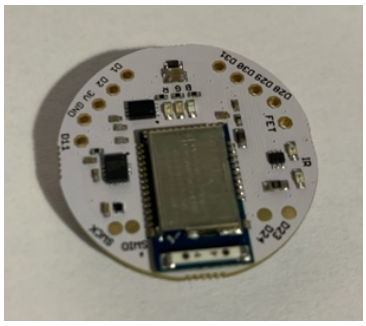
Το Puck.js είναι ένας μικρός, φορητός και προγραμματιζόμενος μικροελεγκτής Bluetooth σχεδιασμένος ειδικά για εφαρμογές στο πλαίσιο του (Internet of Things). Αναπτύχθηκε από τον Gordon Williams, τον δημιουργό της πλατφόρμας ανοιχτού κώδικα Espruino, και υποστηρίζεται από την κοινότητα ανοιχτού κώδικα που έχει δημιουργηθεί γύρω από αυτήν. Είναι εξοπλισμένο με πληθώρα αισθητήρων, όπως επιταχυνσιόμετρο, αισθητήρα φωτός, αισθητήρα θερμοκρασίας, μαγνητόμετρο και απτικό διακόπτη (κουμπί), γεγονός που το καθιστά ιδιαίτερα ευέλικτο για χρήση σε ποικίλες εφαρμογές, όπως η παρακολούθηση κίνησης, η ανίχνευση πτώσεων, και η χρήση ως κουμπί πανικού.

2.1.2 Τεχνικά Χαρακτηριστικά

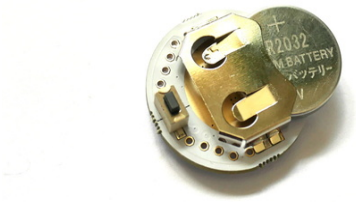
Το Puck.js είναι εξαιρετικά ελαφρύ και μικρό σε διαστάσεις, έχοντας βάρος 17 γραμμάρια (14 γραμμάρια το PCB με την πλαστική θήκη και 3 γραμμάρια η μπαταρία), διάμετρο 36 χιλιοστά και πάχος 12.5 χιλιοστά (με την πλαστική θήκη). Είναι εξοπλισμένο με το MDBT42Q module, το οποίο βασίζεται στο nRF52832 SoC (System on Chip) της Nordic Semiconductor.



Εικόνα 3: Διάμετρος Puck.js με θήκη



Εικόνα 2: Άνω όψη Puck.js, χωρίς θήκη



Εικόνα 1: Κάτω όψη Puck.js, χωρίς θήκη, με μπαταρία

Το MDBT42Q διαθέτει έναν 32-bit ARM Cortex-M4 μικροελεγκτή χρονισμένο στα 64MHz, 64kB μνήμης RAM και 512kB μνήμης flash. Ο Cortex-M4 προσφέρει επεξεργαστική ισχύ με εξαιρετικά χαμηλή κατανάλωση ενέργειας, ιδανική για φορητές συσκευές τροφοδοτούμενες από μπαταρία. Το module υποστηρίζει επίσης πολλαπλές λειτουργίες χαμηλής ισχύος, επιτρέποντας στη συσκευή να λειτουργεί για

μεγάλες χρονικές περιόδους χωρίς την ανάγκη συχνής αλλαγής μπαταρίας. Παράλληλα, ενσωματώνει Bluetooth Low Energy (BLE) και υποστηρίζει την πιο πρόσφατη έκδοση Bluetooth, την 5.0, η οποία προσφέρει βελτιωμένη εμβέλεια, μεγαλύτερη ταχύτητα μετάδοσης δεδομένων, και αυξημένη χωρητικότητα μηνυμάτων σε σχέση με προηγούμενες εκδόσεις. Αυτή η τεχνολογία επιτρέπει στο Puck.js την αξιόπιστη ασύρματη επικοινωνία με άλλες συσκευές όπως smartphones, tablets και υπολογιστές, ενώ ταυτόχρονα προσφέρει μακρά διάρκεια ζωής της μπαταρίας, καθώς το BLE είναι ιδιαίτερα αποδοτικό ενεργειακά. Η συσκευή τροφοδοτείται από μία μπαταρία λιθίου CR2032 των 3V. Τέλος, το MDBT42Q περιλαμβάνει ένα μεγάλο σύνολο ενσωματωμένων περιφερειακών, όπως 12-bit ADC (Analog-to-Digital Converter), για τη μέτρηση αναλογικών σημάτων, PWM (Pulse Width Modulation), για τον έλεγχο εξωτερικών συσκευών όπως κινητήρες και φώτα, I2C, SPI και UART για την επικοινωνία με άλλες συσκευές και αισθητήρες και NFC (Near Field Communication) tag. Τα περιφερειακά αυτά είναι εκτός ενδιαφέροντος του παρόντος έργου και αναφέρονται χάριν πληρότητας [12].

Το Puck.js φέρει, ακόμη, τρία ενσωματωμένα LED, ένα κόκκινο, ένα μπλε και ένα πράσινο. Αυτά τα LED μπορούν να προγραμματιστούν ώστε να ανάβουν με διάφορους τρόπους, παρέχοντας οπτική ανατροφοδότηση για την κατάσταση της συσκευής ή για την αναπαράσταση δεδομένων. Στα πλαίσια του παρόντος οι ενδείξεις μέσω των LED αξιοποιούνται για να δείξουν αν η συσκευή έχει συνδεθεί επιτυχώς μέσω Bluetooth ή αν έχει ανιχνευθεί κάποια συγκεκριμένη κατάσταση, όπως η πτώση.

Το Puck.js περιλαμβάνει επίσης έναν πομπό υπερύθρων (IR), ο οποίος μπορεί να χρησιμοποιηθεί για την αποστολή υπέρυθρων σημάτων σε άλλες συσκευές που ελέγχονται μέσω υπερύθρων, όπως τηλεοράσεις και συστήματα κλιματισμού. Αυτό το χαρακτηριστικό δίνει στο Puck.js τη δυνατότητα να λειτουργήσει ως τηλεχειριστήριο ή να ενσωματωθεί σε αυτοματοποιημένα συστήματα που απαιτούν αποστολή IR εντολών. Περισσότερα σχετικά με αυτό αναφέρονται στο τελευταίο κεφάλαιο του παρόντος, που αφορά τις πιθανές επεκτάσεις[13] .

2.1.3 Ενσωματωμένοι Αισθητήρες

Το Puck.js διαθέτει πληθώρα αισθητήρων, οι οποίοι παρουσιάζονται παρακάτω:

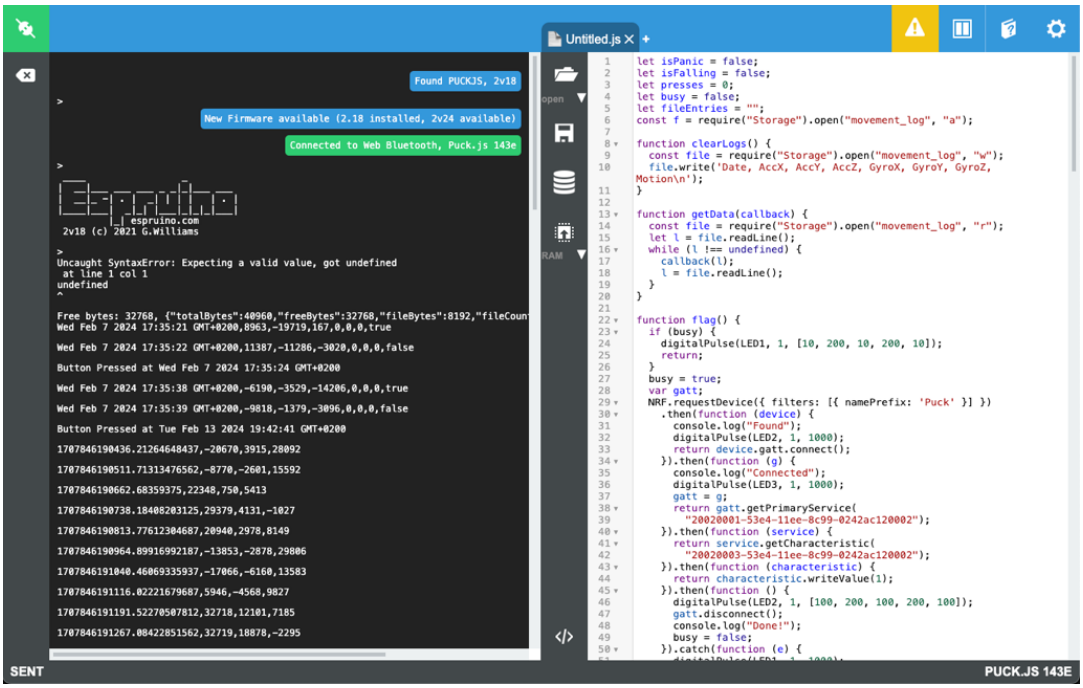
- Επιταχυνσιόμετρο με μέτρηση επιτάχυνσης σε τρεις άξονες (x, y, z) γεγονός που το καθιστά ιδανικό για εφαρμογές που παρακολουθούν τη στάση και τη δραστηριότητα του χρήστη, καθώς και για ανίχνευση πτώσεων.
- Μαγνητόμετρο, το οποίο ανιχνεύει το μαγνητικό πεδίο της Γης, κάτι που μπορεί να χρησιμοποιηθεί για την ανίχνευση μεταβολών στην κατεύθυνση ή την παρουσία μεταλλικών αντικειμένων που επηρεάζουν το μαγνητικό πεδίο.
- Γυροσκόπιο, το οποίο μετράει την γωνιακή ταχύτητα, δηλαδή την ταχύτητα με την οποία περιστρέφεται η συσκευή γύρω από τους τρεις άξονες (x, y, z). Ο συνδυασμός με τα δύο προαναφερθέντα παρέχει πλήρη δεδομένα κίνησης και προσανατολισμού, επιτρέποντας πιο ακριβή και αξιόπιστη παρακολούθηση της κίνησης.

- Αισθητήρα επιπέδου μπαταρίας, ο οποίος μπορεί να αξιοποιηθεί για την παρακολούθηση της κατάστασης της μπαταρίας, και να ενημερώνει τον χρήστη όταν η μπαταρία είναι χαμηλή και χρειάζεται αντικατάσταση, για παράδειγμα μέσω των LED.
- Αισθητήρας Φωτός και Θερμοκρασίας οι οποίοι επιτρέπουν τη μέτρηση και παρακολούθηση των περιβαλλοντικών παραμέτρων.

2.1.4 Προγραμματισμός με JavaScript, ο διερμηνέας Espruino και το Espruino Web IDE

Το Puck.js είναι σχεδιασμένο για να προγραμματίζεται εύκολα και γρήγορα χρησιμοποιώντας τη γλώσσα JavaScript. Αυτή η επιλογή της γλώσσας προγραμματισμού το καθιστά προσιτό τόσο σε έμπειρους προγραμματιστές όσο και σε αρχάριους, καθώς η JavaScript είναι μια από τις πιο διαδεδομένες γλώσσες προγραμματισμού. Χρησιμοποιεί τον διερμηνέα JavaScript Espruino, ο οποίος είναι ένας ανοιχτού κώδικα διερμηνέας JavaScript σχεδιασμένος για να τρέχει σε μικροελεγκτές και είναι πλέον εγκατεστημένος σε δεκάδες χιλιάδες συσκευές και έχει χιλιάδες χρήστες.

Ο Espruino παρέχει ένα κοινό API μεταξύ των συσκευών που τον υποστηρίζουν, δίνοντας τη δυνατότητα άμεσης μεταφοράς του έργου από τη μία στην άλλη. Παράλληλα, παρέχεται το Espruino Web IDE, ένα φιλικό διαδικτυακό περιβάλλον ανάπτυξης που επιτρέπει στους χρήστες να γράφουν, να τροποποιούν και να εκτελούν κώδικα JavaScript από τον φυλλομετρητή τους απευθείας στη συσκευή, παρατηρώντας τα αποτελέσματα σε πραγματικό χρόνο. Το Web IDE μεταφορτώνει τον κώδικα τον στο Puck.js ασύρματα μέσω Bluetooth, χωρίς την ανάγκη για καλώδια και συνδεσμολογία. Υπάρχει η δυνατότητα φόρτωσης τόσο στη Ram όσο και στη flash, επιτρέποντας στο μικροελεγκτή να λειτουργεί αυτόνομα χωρίς να χρειάζεται συνεχής



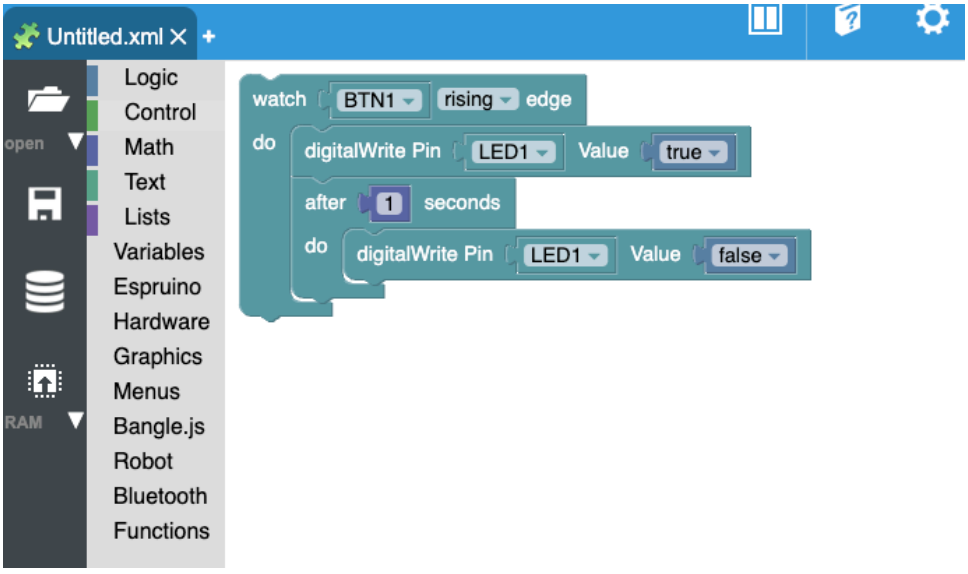
Εικόνα 4: Το Espruino Web IDE

σύνδεση με τον υπολογιστή ή το προγραμματιστικό περιβάλλον. Τέλος, το Web IDE παρέχει στον προγραμματιστή κάποια βασικά εργαλεία αποσφαλμάτωσης. Επισημαίνει συντακτικά σφάλματα και περιλαμβάνει μια ενσωματωμένη κονσόλα εντολών, όπου μπορούν προβληθούν logs (αρχεία καταγραφής ή εντολές εκτύπωσης), εξαιρέσεις και άλλα μηνύματα από τη συσκευή κατά την εκτέλεση του κώδικα. Η κονσόλα διευκολύνει την παρακολούθηση της εκτέλεσης και της απόδοσης του προγράμματος σε πραγματικό χρόνο, καθώς και την εύρεση και επιδιόρθωση σφαλμάτων [14].

Τα προαναφερθέντα προσφέρουν ευκολία στη χρήση, ευελιξία στις δοκιμές και τις τροποποιήσεις, φορητότητα και ταχύτητα στην ανάπτυξη και τον έλεγχο πρωτοτύπων.

2.1.5 Προγραμματισμός με Blockly

Εκτός από τη δυνατότητα απευθείας προγραμματισμού με JavaScript, το Espruino Web IDE υποστηρίζει και τον προγραμματισμό μέσω του Blockly. Το Blockly είναι ένα ανοιχτού κώδικα εργαλείο που επιτρέπει τη δημιουργία κώδικα χρησιμοποιώντας γραφικά στοιχεία (blocks) αντί για γραμμές κειμένου. Μόλις ολοκληρωθεί το πρόγραμμα, ο κώδικας μπορεί να εκτελεστεί απευθείας στη συσκευή μέσω του Espruino Interpreter, ακριβώς όπως και με τον κώδικα JavaScript. Το Blockly είναι ιδιαίτερα χρήσιμο για αρχάριους ή για χρήστες που προτιμούν έναν πιο οπτικό τρόπο προγραμματισμού.



Εικόνα 5: Παράδειγμα προγράμματος Blockly για το Puck.js

2.2 Η γλώσσα προγραμματισμού JavaScript

2.2.1 Βασικά Χαρακτηριστικά

Η JavaScript είναι μια από τις πιο διαδεδομένες και ευέλικτες γλώσσες προγραμματισμού παγκοσμίως. Ο ρόλος της στην ανάπτυξη ιστοσελίδων, η ευελιξία, η ευκολία χρήσης, το εκτενές οικοσύστημα και η μεγάλη κοινότητα υποστήριξης έχουν καταστήσει τη JavaScript την πρώτη επιλογή για πολλούς προγραμματιστές και οργανισμούς [15], [16], [17]. Αν και είναι περισσότερο γνωστή ως η γλώσσα σεναρίων για ιστοσελίδες, για την οποία χρήση αναπτύχθηκε αρχικά, πλέον χρησιμοποιείται επίσης σε πολλά μη-φυλλομετρικά περιβάλλοντα, όπως το Node.js, το Apache CouchDB και το Adobe Acrobat. Η JavaScript είναι μια γλώσσα σεναρίων που βασίζεται σε πρωτότυπα (prototype-based), είναι μονονηματική, δυναμική, με ασθενείς τύπους, έχει συναρτήσεις ως αντικείμενα πρώτης τάξης και υποστηρίζει πολλαπλά προγραμματιστικά παραδείγματα (multi-paradigm): αντικειμενοστρεφή, προστακτικό και συναρτησιακό προγραμματισμό, επιτρέποντας στους προγραμματιστές να επιλέξουν τον προγραμματιστικό στυλ που ταιριάζει καλύτερα στις ανάγκες τους. Τα προαναφερθέντα την καθιστούν αρκετά ευέλικτη και ιδανική για την ανάπτυξη ποικίλων εφαρμογών. Η σύνταξή της είναι επηρεασμένη από τη C. Η JavaScript χρησιμοποιεί πολλά ονόματα και συμβάσεις ονοματοδοσίας από τη Java, αλλά γενικά οι δύο αυτές γλώσσες δε σχετίζονται και έχουν πολύ διαφορετική σημασιολογία. Ένα ακόμη πολύ σημαντικό της χαρακτηριστικό είναι ότι διαθέτει ισχυρές δυνατότητες ασύγχρονου προγραμματισμού μέσω της χρήσης callbacks, promises, και του συντακτικού async/await, που διευκολύνουν την ανάπτυξη εφαρμογών που απαιτούν διαχείριση πολλών διαδικασιών ταυτόχρονα, όπως κλήσεις σε διακομιστές, επεξεργασία δεδομένων σε πραγματικό χρόνο και λειτουργίες εισόδου/εξόδου (I/O operations). Το πρότυπο της γλώσσας κατά τον οργανισμό τυποποίησης ECMA ονομάζεται ECMAscript.

2.2.2 Ιστορική Αναδρομή

Η JavaScript αναπτύχθηκε το 1995 από τον Brendan Eich κατά τη διάρκεια της εργασίας του στη Netscape Communications. Αρχικά δημιουργήθηκε για την προσθήκη διαδραστικότητας στις ιστοσελίδες, ως μια ελαφριά, διερμηνευόμενη γλώσσα προγραμματισμού που θα μπορούσε να εκτελείται απευθείας στον φυλλομετρητή (browser), δηλαδή στην πλευρά του πελάτη (client-side), όπου και απέκτησε μεγάλη επιτυχία. Παρά το γεγονός ότι αρχικά θεωρήθηκε απλή και κατάλληλη μόνο για μικρά σενάρια, η JavaScript εξελίχθηκε ταχύτατα, και με την έλευση του περιβάλλοντος ανάπτυξης Node.js το 2009, μετατράπηκε σε μια πλήρη γλώσσα γενικής ικανή να χρησιμοποιείται τόσο στον πελάτη όσο και στο διακομιστή. Σήμερα χρησιμοποιείται ευρέως σε διάφορες πλατφόρμες, από την ανάπτυξη ιστοσελίδων μέχρι την ανάπτυξη εφαρμογών για κινητές συσκευές και το Internet of Things (IoT).

2.2.3 Πλεονεκτήματα

Επιπρόσθετα των πλεονεκτημάτων περί ευελιξίας που αναφέρθηκαν στην παράγραφο 2.2.1, η JavaScript συνδυάζει δύο ακόμη πολύ ουσιώδη πλεονεκτήματα. Όπως αναφέρθηκε ήδη, μπορεί να χρησιμοποιηθεί σε πολλαπλές πλατφόρμες και περιβάλλοντα, από φυλλομετρητές και διακομιστές έως κινητές συσκευές και μικροελεγκτές όπως το Puck.js. Αυτή η ευελιξία επιτρέπει στους προγραμματιστές να χρησιμοποιούν την ίδια γλώσσα για πολλά διαφορετικά μέρη μιας εφαρμογής, κάνοντας την ανάπτυξη πιο αποδοτική και συνεκτική. Ακόμη, η JavaScript υποστηρίζεται από την μεγαλύτερη κοινότητα προγραμματιστών σε σύγκριση με οποιαδήποτε άλλη γλώσσα [18], [19], με αποτέλεσμα να υπάρχουν χιλιάδες βιβλιοθήκες, εκατοντάδες πλαίσια ανάπτυξης εφαρμογών (frameworks)και πληθώρα οδηγών και παραδειγμάτων, κατά μεγάλο ποσοστό ανοιχτού κώδικα, που διευκολύνουν την ανάπτυξη εφαρμογών. Αυτό σημαίνει ότι για πολλά προβλήματα, ανάγκες ή λειτουργίες με τα οποία έρχεται αντιμέτωπος ο προγραμματιστής, πιθανώς επιλύονται ή υλοποιούνται ήδη μέσω μιας βιβλιοθήκης ή ενός εργαλείου, τα οποία μάλιστα μπορούν να τροποποιηθούν κατά το δοκούν.

2.2.4 Μειονεκτήματα

Παρά τα πολλά της πλεονεκτήματα, η JavaScript εμφανίζει δύο βασικά μειονεκτήματα τα οποία μας απασχολούν στα πλαίσια του παρόντος έργου. Αρχικά, όπως ήδη έχει αναλυθεί, είναι μια γλώσσα δυναμική γλώσσα ασθενών τύπων, που σημαίνει ότι οι μεταβλητές δεν έχουν καθορισμένο τύπο και μπορούν να αλλάζουν τύπο κατά την εκτέλεση. Αν και αυτό προσφέρει ευελιξία, αυξάνει τον κίνδυνο απρόσμενων σφαλμάτων που εμφανίζονται μόνο κατά την εκτέλεση, καθιστώντας δύσκολο τον εντοπισμό και την επίλυση προβλημάτων νωρίς, κατά τη διάρκεια της ανάπτυξης. Το πρόβλημα αυτό, ωστόσο, είναι δυνατόν να αντιμετωπιστεί, όπου αυτό απαιτείται, με χρήση της γλώσσας TypeScript αντί για την JavaScript, η οποία θα παρουσιαστεί στην αμέσως επόμενη υποενότητα. Επιπρόσθετα, σε σύγκριση με άλλες γλώσσες, η JavaScript μπορεί να μην αποδίδει το ίδιο καλά σε εφαρμογές που απαιτούν έντονη χρήση της CPU ή πολύπλοκους υπολογισμούς καθώς γλώσσα δεν είναι σχεδιασμένη για εργασίες υψηλής απόδοσης. Το πρόβλημα αυτό έγινε εμφανές κατά τη διαδικασία εκπαίδευσης του Νευρωνικού Δικτύου, η οποία θα παρουσιαστεί στο Κεφάλαιο 4 του παρόντος.

2.2.5 Η γλώσσα προγραμματισμού TypeScript

Η TypeScript είναι μια δωρεάν και ανοιχτού κώδικα γλώσσα προγραμματισμού υψηλού επιπέδου που αναπτύχθηκε από τη Microsoft και παρουσιάστηκε το 2012. Αποτελεί ένα υπερσύνολο της JavaScript, το οποίο προσθέτει δυνατότητες στατικής τυποποίησης (static typing) με προαιρετικές δηλώσεις τύπων και άλλες σύγχρονες λειτουργίες, οι οποίες διευκολύνουν τη συντήρηση και ανάπτυξη μεγάλων και πολύπλοκων εφαρμογών. Ο κώδικας TypeScript μεταγλωττίζεται σε καθαρή JavaScript, πράγμα που σημαίνει ότι όλα τα προγράμματα JavaScript είναι συντακτικά έγκυρα προγράμματα TypeScript, ενώ μπορεί να τρέξει σε οποιοδήποτε περιβάλλον που υποστηρίζει JavaScript. Όπως αναφέρθηκε στην προηγούμενη υποενότητα, η

TypeScript αναπτύχθηκε ως απάντηση στις προκλήσεις που αντιμετωπίζουν οι προγραμματιστές κατά την ανάπτυξη μεγάλων και σύνθετων εφαρμογών με JavaScript. Παρόλο που η JavaScript είναι εξαιρετικά ευέλικτη και δυναμική, η έλλειψη στατικών τύπων μπορεί να οδηγήσει σε δυσκολίες στην ανάπτυξη και τη συντήρηση. Η TypeScript, με τη στατική της τυποποίηση, επιτρέπει στους προγραμματιστές να προσδιορίζουν τους τύπους δεδομένων από την αρχή, μειώνοντας έτσι τα πιθανά σφάλματα και μεταφέροντάς τα στη φάση της μεταγλώττισης αντί για εκείνη της εκτέλεσης, καθιστώντας έτσι τον κώδικα πιο προβλέψιμο, ευανάγνωστο και διαχειρίσιμο και κάνοντας πλέον εφικτό τον έγκαιρο εντοπισμό σφαλμάτων. Παράλληλα, οι περισσότεροι σύγχρονοι επεξεργαστές κώδικα και IDEs προσφέρουν εκτεταμένη υποστήριξη για TypeScript, συμπεριλαμβανομένων των αυτόματων συμπληρώσεων κώδικα, της εύρεσης αναφορών και της επισήμανσης συντακτικών σφαλμάτων. Αυτά τα χαρακτηριστικά κάνουν την ανάπτυξη με TypeScript πιο ευχάσριστη, αποδοτική, γρήγορη και λιγότερο επιρρεπή σε λάθη [20].

2.3 Το περιβάλλον εκτέλεσης Node.js και οι χρησιμοποιούμενες βιβλιοθήκες

2.3.1 Γενικά

Το Node.js είναι ένα δωρεάν, ανοιχτού κώδικα, ανεξάρτητο πλατφόρμας (cross-platform) περιβάλλον εκτέλεσης JavaScript που επιτρέπει την εκτέλεση κώδικα JavaScript εκτός ενός προγράμματος φυλομετρητή (browser). Αναπτύχθηκε το 2009 από τον Ryan Dahl και βασίζεται στη μηχανή JavaScript V8 της Google, που χρησιμοποιείται και στο Google Chrome. Το Node.js έχει γίνει δημοφιλές για τη δημιουργία ταχέων και επεκτάσιμων διακομιστών (scalable servers), υπηρεσιών ροής (streaming services) και διαδικτυακών εφαρμογών, προσφέροντας δυνατότητες που παραδοσιακά ήταν διαθέσιμες μόνο μέσω γλωσσών προγραμματισμού για την πλευρα του εξυπηρετητή, όπως η PHP, η Ruby, ή η Python. Χρησιμοποιείται επίσης ευρέως για εφαρμογές υπολογιστών (desktop apps), εφαρμογές διαχείρισης δεδομένων, εφαρμογές IoT, εργαλεία γραμμής εντολών και scripts [21].

2.3.2 Βασικά Χαρακτηριστικά- Πλεονεκτήματα

- Ασύγχρονη και Event-Driven Αρχιτεκτονική: Το Node.js βασίζεται σε μια ασύγχρονη, βάσει γεγονότων (event-driven) αρχιτεκτονική, που σημαίνει ότι μπορεί να χειρίζεται πολλαπλά αιτήματα ταυτόχρονα χωρίς να μπλοκάρει τον εξυπηρετητή. Κάθε φορά που διαχειρίζεται ένα συμβάν (event), εκτελεί μία σχετική κλήση επιστροφής (callback), επιτρέποντας τη διαχείριση μεγάλου όγκου δεδομένων με χαμηλή απαίτηση σε υπολογιστική ισχύ.
- Μηχανή JavaScript V8: Ο πυρήνας του Node.js βασίζεται στη μηχανή JavaScript V8 της Google, η οποία είναι γνωστή για την υψηλή της απόδοση και την ταχύτητα εκτέλεσης κώδικα. Αυτή η μηχανή παρέχει εκτέλεση JavaScript κώδικα με πολύ υψηλή ταχύτητα, πράγμαι κρίσιμο για τη διαχείριση εφαρμογών σε πραγματικό χρόνο.
- Non-blocking I/O: Το Node.js χρησιμοποιεί ένα non-blocking I/O μοντέλο, που σημαίνει ότι οι λειτουργίες εισόδου-εξόδου (I/O) δεν μπλοκάρουν την εκτέλεση άλλων λειτουργιών. Αυτό επιτρέπει στο Node.js τη διαχείριση

πολλαπλών αιτημάτων ταυτόχρονα με μεγάλη αποτελεσματικότητα, καθιστώντας το ιδανικό για εφαρμογές που απαιτούν γρήγορη απόκριση και χειρισμό μεγάλου όγκου αιτήσεων, όπως APIs και εφαρμογές πραγματικού χρόνου.

- Ενιαία Γλώσσα για Frontend και Backend: Χρησιμοποιώντας JavaScript τόσο στο frontend όσο και στο backend, το Node.js επιτρέπει στους προγραμματιστές να χρησιμοποιούν την ίδια γλώσσα σε ολόκληρο το σύνολο τεχνολογιών ανάπτυξης. Αυτό απλοποιεί τη διαδικασία ανάπτυξης, διευκολύνοντας την ανταλλαγή κώδικα και τη συνεργασία μεταξύ των ομάδων και μειώνοντας κατά το ήμισυ τον όγκο των απαιτούμενων γνώσεων για την ολοκλήρωση ενός έργου.
- Node Package Manager (NPM): Το Node.js συνοδεύεται από το NPM το οποίο είναι η μεγαλύτερη συλλογή πακέτων στον κόσμο. Το NPM επιτρέπει στους προγραμματιστές να βρίσκουν και να ενσωματώνουν εύκολα βιβλιοθήκες και εργαλεία στα έργα τους, επιταχύνοντας την ανάπτυξη και διευκολύνοντας την επίλυση κοινών προβλημάτων.

2.3.3 Μειονεκτήματα

- Μονονηματικό μοντέλο (Single-Threaded Model): Το Node.js λειτουργεί με ένα μονονηματικό event loop, το οποίο είναι ιδιαίτερα αποδοτικό για λειτουργίες I/O, ωστόσο μπορεί να είναι περιοριστικό για εργασίες που απαιτούν έντονη χρήση της CPU, όπως πολύπλοκοι υπολογισμοί, επεξεργασία δεδομένων ή κρυπτογράφηση. Αυτές οι εργασίες είναι δυνατόν να μπλοκάρουν το event loop, προκαλώντας καθυστερήσεις στην επεξεργασία άλλων αιτημάτων. Παράλληλα, το μονονηματικό μοντέλο οδηγεί σε υποαξιοποίηση των πολυπύρηνων επεξεργαστών. Αν και υπάρχουν λύσεις, όπως η συστοιχία (clustering) και τα νήματα εργατών (worker threads), για την αξιοποίηση πολλαπλών πυρήνων, αυτές οι λύσεις προσθέτουν πολυπλοκότητα στη διαδικασία ανάπτυξης.
- Ανεπαρκής Ωριμότητα Ορισμένων Βιβλιοθηκών: Αν και το οικοσύστημα του Node.js είναι τεράστιο, δεν έχουν όλες οι βιβλιοθήκες και τα πακέτα την ίδια ποιότητα. Κάποιες βιβλιοθήκες μπορεί να μην υποστηρίζονται επαρκώς ή να στερούνται κατάλληλης τεκμηρίωσης. Αυτή η ανωριμότητα στο οικοσύστημα μπορεί να οδηγήσει σε προβλήματα σταθερότητας ή κενά ασφαλείας στις εφαρμογές.
- Αστάθεια API λόγω συχνών ενημερώσεων και αλλαγών που προκαλούν ασυμβατότητα (Breaking Changes): Το Node.js εξελίσσεται γρήγορα, και ενώ αυτό φέρνει νέες δυνατότητες και βελτιώσεις, σημαίνει επίσης ότι τα API μπορούν να αλλάζουν συχνά. Αυτό μπορεί να οδηγήσει σε ασυμβατότητες εκδόσεων στις υπάρχουσες εφαρμογές, απαιτώντας από τους προγραμματιστές να ενημερώνουν και να δοκιμάζουν συχνά τον κώδικά τους.

2.3.4 Το πλαίσιο ανάπτυξης εφαρμογών Express.js

2.3.4.1 Βασικά Χαρακτηριστικά και πλεονεκτήματα

Το Express.js είναι ένα δημοφιλές, ελαφρύ και ευέλικτο πλαίσιο ανάπτυξης (framework) διαδικτυακών εφαρμογών για το Node.js. Αναπτύχθηκε το 2010 από τον TJ Holowaychuk και έχει εξελιχθεί σε ένα από τα πιο χρησιμοποιούμενα frameworks στη JavaScript κοινότητα, λόγω της απλότητας και της ισχύος του. Το Express.js είναι γνωστό για την απλότητά του, αφού παρέχει ένα μινιμαλιστικό πυρήνα, χωρίς περιορισμούς δομής και σχεδίασης (unopinionated) και έναν ιδιαίτερα ευέλικτο μηχανισμό δρομολόγησης (routing). Αποτελεί λοιπόν μια καθαρή και λιτή διεπαφή για τη δημιουργία διακομιστών και το χειρισμό αιτημάτων και απαντήσεων και είναι εύκολο στη μάθηση καις τη χρήση, ακόμη και μη προχωρημένους προγραμματιστές. Παρά την απλότητά του δεν στερείται δυνατοτήτων, όντας πολύ επεκτάσιμο. Η έλλειψη επιβαλλόμενης δομής, σχεδίασης και προεπιλογής εργαλείων από το framework επιτρέπει στους προγραμματιστές να προσθέσουν ακριβώς τις βιβλιοθήκες και τα εργαλεία που χρειάζονται και να ακολουθήσουν αρχιτεκτονική που επιθυμούν για την εφαρμογή τους. Ένα από τα πιο ισχυρά χαρακτηριστικά του Express.js είναι η υποστήριξη για middleware. Τα middlewares είναι λειτουργίες που εκτελούνται κατά τη διάρκεια της επεξεργασίας αιτημάτων στον διακομιστή. Το Express.js επιτρέπει και ενθαρρύνει την αλυσιδωτή χρήση middleware, προσφέροντας έναν εύκολο τρόπο για την προσθήκη λειτουργιών όπως η αυθεντικοποίηση, η διαχείριση σφαλμάτων, η ανάλυση δεδομένων (parsing) και η διαχείριση συνεδριών (sessions). Τέλος, το Express.js συνεργάζεται άψογα με διάφορες τεχνολογίες frontend και μηχανές προτύπων (templating engines), όπως το Pug, το EJS και το Handlebars. Αυτές οι τεχνολογίες επιτρέπουν τη δημιουργία δυναμικών σελίδων HTML από δεδομένα, καθιστώντας το Express.js ιδανικό για την ανάπτυξη fullstack (frontend και backend μαζί) εφαρμογών. Το τελευταίο πλεονέκτημα δεν αξιοποιείται στα πλαίσια του παρόντος αφού το frontend αποτελεί μία εφαρμογή για κινητές συσκευές γραμμένη σε Flutter και αναφέρεται μόνο χάριν πληρότητας [22].

2.3.4.2 Μειονεκτήματα

Τα μεγαλύτερα πλεονεκτήματα του Express.js, δηλαδή η ευελιξία και η ελευθερία επιλογής σε δομή, εργαλεία και επεκτάσεις, συνοψίζουν ουσιαστικά και το βασικό του μειονέκτημα: απαιτεί βαθιά γνώση της JavaScript και των εσωτερικών μηχανισμών του Node.js, για τη δημιουργία σύνθετων εφαρμογών. Επαφίεται στον προγραμματιστή σχεδιάσει και να δομήσει σωστά την εφαρμογή και τα διάφορα μέρη της και να επιλέξει τις καταλληλότερες και ποιοτικότερες βιβλιοθήκες ακόμα και για τις πιο συνηθισμένες περιπτώσεις χρήσεις όπως η αυθεντικοποίηση χρηστών, πράγμα που μπορεί να οδηγήσει σε ριζικά λάθη έναν άπειρο προγραμματιστή.

2.3.5 Η βιβλιοθήκη Kysely

Το Kysely είναι μια σύγχρονη βιβλιοθήκη TypeScript για τη διαχείριση βάσεων δεδομένων, η οποία παρέχει ένα εύχρηστο και ευέλικτο αντικειμενοστραφές API με

ασφάλεια τύπων και δυνατότητες αυτοσυμπλήρωσης για τη δημιουργία SQL ερωτημάτων. Σε αντίθεση με τα ORM (Object-Relational Mapping), το Kysely είναι απλώς ένα ελαφρύ στρώμα αφαίρεσης (abstraction layer) πάνω από την SQL. Δεν προσπαθεί να αποκρύψει το SQL, αλλά αντίθετα το αναδεικνύει, προσφέροντας ένα καθαρό και ασφαλές API που καθιστά τη συγγραφή SQL ερωτημάτων πιο εύκολη, ευανάγνωστη και λιγότερο επιρρεπή σε σφάλματα. Είναι συμβατό με πολλές διαφορετικές βάσεις δεδομένων και μπορεί εύκολα να επεκταθεί από τον ίδιο τον προγραμματιστή, με τη βοήθεια της κοινότητας, ώστε να καλύψει εξειδικευμένα σενάρια χρήσης και συμβατότητας [23].

2.4 SQL, Σχεσιακές Βάσεις Δεδομένων και MariaDB

2.4.1 Structured Query Language (SQL)

Η SQL (Structured Query Language) είναι η καθιερωμένη γλώσσα προγραμματισμού που χρησιμοποιείται για τη διαχείριση και την αλληλεπίδραση με σχεσιακές βάσεις δεδομένων. Είναι ιδιαίτερα χρήσιμη στην επεξεργασία δομημένων δεδομένων, δηλαδή δεδομένων που περιλαμβάνουν σχέσεις μεταξύ οντοτήτων και μεταβλητών. επιτρέποντας στους χρήστες να εκτελούν εντολές για τη δημιουργία, την τροποποίηση, την ανάκτηση και τη διαγραφή δεδομένων μέσα από σχεσιακές βάσεις δεδομένων, καθώς και να διαχειρίζονται τις δομές των ίδιων των βάσεων (όπως πίνακες, όψεις και δείκτες). Η SQL αποτελεί τη ραχοκοκαλιά των συστημάτων διαχείρισης σχεσιακών βάσεων δεδομένων (Relational Database Management System - RDBMS) και είναι απαραίτητη για τη λειτουργία σύγχρονων εφαρμογών που απαιτούν ασφαλή και αποδοτική διαχείριση μεγάλου όγκου δεδομένων.

2.4.2 Σχεσιακές Βάσεις Δεδομένων

Οι βάσεις δεδομένων είναι οργανωμένες συλλογές δεδομένων που αποθηκεύονται με τρόπο που διευκολύνει την ανάκτηση και την ανάλυση τους. Οι σχεσιακές βάσεις δεδομένων οργανώνουν τα δεδομένα σε πίνακες, όπου οι στήλες αντιπροσωπεύουν τα πεδία των δεδομένων, τα οποία χαρακτηρίζονται από έναν τύπο (για παράδειγμα ακέραιος ή συμβολοσειρά) και οι γραμμές αντιπροσωπεύουν διαφορετικές εγγραφές. Οι βάσεις δεδομένων χρησιμοποιούνται ευρέως σε διάφορες εφαρμογές, από ιστότοπους και επιχειρηματικά συστήματα μέχρι μεγάλα πληροφοριακά συστήματα, καθώς προσφέρουν την υποδομή για την ασφαλή αποθήκευση και γρήγορη ανάκτηση δεδομένων.

2.4.3 MariaDB

Η MariaDB είναι ένα δημοφιλές και ισχυρό, ανοικτού κώδικα σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων που βασίζεται στη MySQL και είναι πλήρως συμβατή με αυτό, γεγονός που επιτρέπει την εύκολη μετάβαση από το ένα σύστημα στο άλλο με ελάχιστες ή καθόλου τροποποιήσεις στα υπάρχοντα προγράμματα. Παρέχει επίσης βελτιώσεις απόδοσης και νέες δυνατότητες που δεν υπάρχουν στη MySQL, καθιστώντας την μια δημοφιλή επιλογή για νέες εφαρμογές και αναβαθμίσεις από παλαιότερες εκδόσεις MySQL. Δημιουργήθηκε από κάποιους εκ των αρχικών

προγραμματιστών της MySQL ως παρακλάδι (fork) της τελευταίας λόγω της εξαγοράς της από την Oracle, με σκοπό να διασφαλίσει ότι η βάση δεδομένων θα παραμείνει ελεύθερη και ανοιχτή για χρήση, τροποποίηση και διανομή από την κοινότητα. Η MariaDB είναι σχεδιασμένη για να προσφέρει υψηλές επιδόσεις, ακόμα και σε μεγάλης κλίμακας εφαρμογές. Διαθέτει μηχανισμούς βελτιστοποίησης ερωτημάτων (query optimization) και αποδοτικής διαχείρισης μνήμης, που της επιτρέπουν να επεξεργάζεται μεγάλο όγκο δεδομένων γρήγορα και αποτελεσματικά. Υποστηρίζει μεγάλες βάσεις δεδομένων και μπορεί να διαχειριστεί πολλαπλές συνδέσεις ταυτόχρονα, καθιστώντας την ιδανική για εφαρμογές που απαιτούν υψηλή διαθεσιμότητα και αξιοπιστία. Τέλος, η MariaDB περιλαμβάνει προηγμένα χαρακτηριστικά ασφαλείας, όπως έλεγχο ταυτότητας χρηστών, διαχείριση δικαιωμάτων πρόσβασης, και υποστήριξη για κρυπτογράφηση δεδομένων. Αυτά τα χαρακτηριστικά διασφαλίζουν ότι τα δεδομένα παραμένουν προστατευμένα από μη εξουσιοδοτημένη πρόσβαση.

Τα παραπάνω καθιστούν τη MariaDB αξιόπιστο, ασφαλές και αποδοτικό σύστημα διαχείρισης βάσεων δεδομένων, κατάλληλο για ένα ευρύ φάσμα εφαρμογών, από μικρές προσωπικές ιστοσελίδες έως μεγάλες επιχειρηματικές λύσεις. Με την υποστήριξη της κοινότητας ανοικτού κώδικα και τις ισχυρές δυνατότητες που προσφέρει, η MariaDB αποτελεί μια εξαιρετική επιλογή για την αποθήκευση και διαχείριση δεδομένων σε σύγχρονες εφαρμογές. έχει κερδίσει ευρεία αποδοχή κι έχει επιλεγεί σε πολλές σύγχρονες εφαρμογές, συμπεριλαμβανομένου και του παρόντος έργου για την αποθήκευση και διαχείριση των δεδομένων που συλλέγονται από συσκευές Puck.js [24].

2.5 To Blockly

Το Blockly είναι ένα ανοιχτού κώδικα εργαλείο που επιτρέπει την ανάπτυξη λογισμικού χρησιμοποιώντας οπτική προγραμματιστική γλώσσα, δηλαδή με μεταφορά και τοποθέτηση γραφικών στοιχείων (blocks) αντί για κείμενο κώδικα. Κάθε block αντιπροσωπεύει μια διαφορετική ενέργεια, εντολή ή συνάρτηση, και οι χρήστες μπορούν να τα συνδέουν μεταξύ τους για να δημιουργήσουν πολύπλοκες λογικές δομές και προγράμματα. Μπορεί να χρησιμοποιηθεί για να δημιουργήσει κώδικα σε γλώσσες όπως JavaScript, Lua, Dart, Python, or PHP. καθιστώντας το προγραμματισμό πιο προσβάσιμο σε όσους δεν είναι εξοικειωμένοι με τον κώδικα κειμένου [25].

2.5.1 Πλεονεκτήματα

Το Blockly προσφέρει έναν απλό και φιλικό προς τον χρήστη τρόπο για τη δημιουργία κώδικα. Οι χρήστες απλώς σύρουν και συνδέουν τα blocks για να δημιουργήσουν τα προγράμματά τους, χωρίς να απαιτείται εκτενής γνώση γραπτής προγραμματιστικής γλώσσας, καθιστώντας το ιδανικό για εκπαιδευτικούς σκοπούς, καθώς επιτρέπει στους αρχάριους να μάθουν τις βασικές αρχές του προγραμματισμού χωρίς να αντιμετωπίζουν την πολυπλοκότητα μιας παραδοσιακής γλώσσας. Οι χρήστες μπορούν να πειραματιστούν με διάφορες λειτουργίες και εντολές μέσω των blocks, ενώ παράλληλα μαθαίνουν πώς λειτουργούν οι λογικές ακολουθίες και οι δομές

ελέγχου. Ο προγραμματισμός με το Blockly κάνει την ανάπτυξη εφαρμογών IoT με το Puck.js πιο προσιτή και διασκεδαστική, επιτρέποντας σε χρήστες κάθε επιπέδου εμπειρίας να δημιουργήσουν και να δοκιμάσουν τις ιδέες τους εύκολα και γρήγορα.

2.5.2 Μειονεκτήματα

Το Blockly, ενώ αποτελεί καλή επιλογή για βασικές και μεσαίου επιπέδου εφαρμογές, δεν προσφέρει την ίδια ευελιξία, δύναμη και απόδοση που έχει η γραφή κώδικα απευθείας σε κείμενο. Ορισμένες πολύπλοκες λειτουργίες, σύνθετοι αλγόριθμοι ή προηγμένες τεχνικές προγραμματισμού μπορεί να είναι δύσκολο να υλοποιηθούν χρησιμοποιώντας μόνο blocks. Αυτή η έλλειψη ελέγχου είναι ένα κρίσιμο μειονέκτημα για τις εφαρμογές IoT όπως η παρούσα, διότι αφενός οι πόροι είναι περιορισμένοι κι αφετέρου ο διερμηνέας Espruino εκτελεί τον κώδικα απευθείας από το πηγαίο αρχείο, χωρίς μεταγλώττιση. Η έλλειψη μεταγλώττισης άρα και των βελτιστοποιήσεων που αυτή προσφέρει, καθιστά την αποδοτικότητα του κώδικα πλήρως εξαρτώμενη από το μέγεθος του πηγαίου αρχείου, τις δομές δεδομένων, και τις προγραμματιστικές τεχνικές που έχουν χρησιμοποιηθεί. Τα προαναφερθέντα απαιτούν λεπτομερή γνώση και σχεδίαση του κώδικα, πράγματα στα οποία το Blockly υστερεί. Για αυτούς τους λόγους, στα πλαίσια του παρόντος έργου χρησιμοποιήθηκε αποκλειστικά JavaScript και το Blockly περιγράφεται απλώς ως μία επιπλέον ενδιαφέρουσα και χρήσιμη δυνατότητα του Puck.js.

2.6 Flutter και Dart

2.6.1 Το πλαίσιο ανάπτυξης εφαρμογών Flutter

Το Flutter είναι ένα ανοιχτού κώδικα πλαίσιο ανάπτυξης διεπαφών χρήστη (UI software development kit) που αναπτύχθηκε από την Google. Το Flutter επιτρέπει στους προγραμματιστές να δημιουργούν εφαρμογές πολλαπλών πλατφόρμων (cross-platform) χρησιμοποιώντας έναν ενιαίο κώδικα, ο οποίος μπορεί να τρέχει σε Android, iOS, Fuchsia, Windows, macOS, Linux και web περιβάλλοντα. Πρωτοπαρουσιάστηκε το 2015 και κυκλοφόρησε τον Μάιο του 2017. Το Flutter χρησιμοποιείται εσωτερικά από την Google σε εφαρμογές όπως το Google Pay και το Google Earth, καθώς και από άλλους προγραμματιστές λογισμικού, συμπεριλαμβανομένων των ByteDance και Alibaba. Το κύριο χαρακτηριστικό του Flutter είναι η ικανότητά του να παρέχει εμπειρία χρήστη που προσεγγίζει αυτή των εγγενών εφαρμογών (δηλαδή τις εφαρμογές που είναι γραμμένες στη γλώσσα που προορίζεται για την εκάστοτε πλατφόρμα, όπως η Swift για το iOS ή η Kotlin για το Android) όσον αφορά την απόδοση, παρέχοντας όμως επιπλέον ευελιξία και ταχύτητα στην διαδικασία της ανάπτυξης [26].

2.6.1.1 Βασικά Χαρακτηριστικά και πλεονεκτήματα

- Ενιαίος Κώδικας για Πολλαπλές Πλατφόρμες: Το Flutter επιτρέπει τη συγγραφή ενός κώδικα που μπορεί να τρέξει σε πολλές πλατφόρμες, όπως αναφέρθηκε. Αυτό μειώνει το χρόνο, τους πόρους και τις γνώσεις που

απαιτούνται για την ανάπτυξη και τη συντήρηση εφαρμογών σε διαφορετικές πλατφόρμες.

- Υψηλή Απόδοση: Το Flutter είναι σχεδιασμένο για να παρέχει υψηλή απόδοση, εφάμιλλη των εγγενών εφαρμογών. Χρησιμοποιεί τη δική του μηχανή απεικόνισης, γραμμένη κατά κύριο λόγο σε C++, η οποία επιτρέπει τη δημιουργία γραφικών με γρήγορο και αποδοτικό τρόπο.
- Πλούσια Widgets: Το Flutter προσφέρει μια εκτεταμένη βιβλιοθήκη από προσαρμόσιμα widgets που επιτρέπουν την εύκολη δημιουργία σύγχρονων και ελκυστικών διεπαφών χρήστη (UI). Το widget είναι ο δομικός λίθος σε σε ένα πρόγραμμα Flutter και μπορεί με τη σειρά του να αποτελείται από άλλα widgets. Ένα πρόγραμμα Flutter λοιπόν αποτελεί ουσιαστικά ένα δέντρο από widgets. Ένα widget περιγράφει τη λογική, την αλληλεπίδραση και το σχεδιασμό μιας διεπαφής χρήστη χρησιμοποιώντας το παράδειγμα του δηλωτικού, αντιδραστικού προγραμματισμού (reactive programming).
- Άμεση ανανέωση (Hot Reload): επιτρέπει στους προγραμματιστές να βλέπουν άμεσα τις αλλαγές που κάνουν στον κώδικα, χωρίς να χρειάζεται να επανεκκινήσουν την εφαρμογή. Αυτό επιταχύνει τον κύκλο ανάπτυξης και διευκολύνει την προσαρμογή και την αποσφαλμάτωση του κώδικα.
- Πλούσιο Οικοσύστημα και Υποστήριξη: Το Flutter διαθέτει ένα μεγάλο και ενεργό οικοσύστημα με πολλές βιβλιοθήκες και πακέτα που μπορούν να χρησιμοποιηθούν για την επέκταση των δυνατοτήτων της εφαρμογής. Η κοινότητα του είναι πολύ ενεργή, παρέχοντας συνεχή υποστήριξη και ενημερώσεις.

2.6.1.2 Μειονεκτήματα

Στα πλαίσια του παρόντος έργου τα μειονεκτήματα του Flutter δεν θα μας απασχολήσουν καθώς η συνοδευτική εφαρμογή που δημιουργήθηκε μέσω αυτού είναι μία απλή, μικρή εφαρμογή με μόνο βασικές λειτουργίες και πολύ χαμηλές απαιτήσεις σε πόρους, καθώς επίσης προορίζεται μόνο για κινητές συσκευές Android και iOS, για τις οποίες το Flutter έχει καλή υποστήριξη. Παρόλα αυτά αναφέρονται τα βασικά μειονεκτήματά του εν συντομία χάριν πληρότητας:

- Μεγάλο Μέγεθος Εφαρμογής: Οι εφαρμογές που αναπτύσσονται με το Flutter τείνουν να έχουν μεγαλύτερο μέγεθος σε σύγκριση με τις εγγενείς εφαρμογές.
- Περιορισμένη Υποστήριξη για Πλατφόρμες: Παρόλο που υποστηρίζει πολλές πλατφόρμες, η υποστήριξη για τον ιστό και τους υπολογιστές δεν είναι τόσο ώριμη όσο για τις κινητές συσκευές.
- Μη Εγγενής Απόδοση: Παρόλο που προσφέρει πολύ καλή απόδοση, σε ορισμένες περιπτώσεις η απόδοση των εφαρμογών πληριάζει αλλά μπορεί να μην φτάνει αυτή των εγγενών εφαρμογών, ειδικά όταν πρόκειται για πολύπλοκες διεπαφές ή βαριές διεργασίες που απαιτούν άμεση πρόσβαση στους πόρους της συσκευής.

2.6.2 Η γλώσσα προγραμματισμού Dart

Η Dart είναι μια γλώσσα προγραμματισμού που σχεδιάστηκε από τους Lars Bak και Kasper Lund και αναπτύχθηκε από την Google. Μπορεί να χρησιμοποιηθεί για την ανάπτυξη εφαρμογών για τον ιστό και κινητές συσκευές, καθώς και για διακομιστές και εφαρμογές επιφάνειας εργασίας. Είναι μια γλώσσα αντικειμενοστραφής, βασισμένη σε κλάσεις, με ισχυρούς τύπους, με συλλογή απορριμμάτων (garbage collection) και σύνταξη εμπνευσμένη από τη C#. Μπορεί να μεταγλωττίζεται σε γλώσα μηχανής, JavaScript ή WebAssembly. Υποστηρίζει διεπαφές (interfaces), μίξεις (mixins), αφηρημένες κλάσεις, υλοποιημένες γενικές κλάσεις (reified generics), κληρονομικότητα (inheritance) και συμπερασμό τύπων (type inference), ιδιότητες που καθιστούν τον κώδικα πιο αναγνώσιμο, οργανωμένο και επαναχρησιμοποιήσιμο. Η Dart προσφέρει ισχυρή υποστήριξη για ασύγχρονο προγραμματισμό, μέσω της χρήσης futures και streams, με συντακτικό παρόμοιο με τα promises και το async/await της JavaScript. Αυτή η δυνατότητα είναι ιδιαίτερα χρήσιμη για τη διαχείριση λειτουργιών που μπορεί να διαρκέσουν χρόνο, όπως οι κλήσεις δικτύου, χωρίς να μπλοκάρει η κύρια ροή της εφαρμογής. Τέλος, παρουσιάζει ευέλικτη μεταγλώττιση καθώς μπορεί να μεταγλωττίζεται τόσο σε εγγενή κώδικα (ARM, x86) όσο και σε JavaScript για να τρέχει σε προγράμματα περιήγησης. Αυτό σημαίνει ότι οι εφαρμογές που γράφονται σε Dart μπορούν να εκτελούνται σε σχεδόν οποιαδήποτε συσκευή [27].

2.7 Η πλατφόρμα Firebase

2.7.1 Γενικά

Η Firebase είναι μια πλατφόρμα ανάπτυξης εφαρμογών που παρέχεται από την Google και προσφέρει ένα πλήρες σύνολο εργαλείων για την κατασκευή, τη βελτίωση και τη διαχείριση εφαρμογών ιστού και κινητών συσκευών. Η Firebase παρέχει μια σειρά από υπηρεσίες όπως βάσεις δεδομένων σε πραγματικό χρόνο, αποθήκευση αρχείων, ανάλυση χρηστών και διαχείριση ειδοποιήσεων. Ένα από τα πιο σημαντικά εργαλεία που προσφέρει είναι το Firebase Authentication [28].

2.7.2 Firebase Authentication

Η Firebase Authentication είναι μια υπηρεσία που επιτρέπει στους προγραμματιστές να ενσωματώσουν μηχανισμούς αυθεντικοποίησης (authentication) σε εφαρμογές ιστού και κινητών συσκευών. Η αυθεντικοποίηση είναι μια κρίσιμη διαδικασία που επιτρέπει στους χρήστες να δημιουργούν λογαριασμούς, να συνδέονται στην εφαρμογή και να έχουν πρόσβαση σε εξατομικευμένες λειτουργίες και δεδομένα. Ένα από τα βασικότερα χαρακτηριστικά της υπηρεσίας είναι τα έτοιμα API και SDK που παρέχει, τα οποία εξασφαλίζουν την εύκολη και γρήγορη ενσωμάτωση της υπηρεσίας σε εφαρμογές Android, iOS και Web, επιτρέποντας την εύκολη διαχείριση χρηστών και την αυθεντικοποίηση. Ένα ακόμη πολύ σημαντικό της χαρακτηριστικό είναι η υποστήριξη ποικίλων τρόπων αυθεντικοποίησης. Συγκεκριμένα, υποστηρίζονται οι εξής τρόποι:

- Ηλεκτρονικό ταχυδρομείο και Κωδικός (Email and Password): Η πιο συνηθισμένη μέθοδος αυθεντικοποίησης που επιτρέπει στους χρήστες να δημιουργούν λογαριασμούς χρησιμοποιώντας τη διεύθυνση email τους και έναν κωδικό πρόσβασης.
- OAuth πάροχοι: Υποστηρίζει σύνδεση μέσω τρίτων παρόχων όπως Google, Facebook, Twitter, GitHub.
- Αυθεντικοποίηση μέσω Τηλεφώνου (Phone Authentication): Υποστήριξη αυθεντικοποίησης μέσω αριθμού τηλεφώνου με χρήση SMS για επαλήθευση.
- Ανώνυμη Αυθεντικοποίηση (Anonymous Authentication): Παρέχει τη δυνατότητα στους χρήστες να χρησιμοποιούν προσωρινά την εφαρμογή χωρίς να συνδεθούν, επιτρέποντας την εύκολη μετάβαση σε πλήρη αυθεντικοποίηση αργότερα.
- Πάροχος Προσαρμοσμένων Διακριτικών (Custom Token Provider): Ένας τέτοιος πάροχος επιτρέπει στον προγραμματιστή να χρησιμοποιήσει το δικό του μηχανισμό αυθεντικοποίησης και να δημιουργήσει προσαρμοσμένα (custom) JWT tokens, τα οποία στη συνέχεια η Firebase Authentication μπορεί να επαληθεύσει (verify). Με άλλα λόγια παρέχεται στον προγραμματιστή η δυνατότητα χρήσης του δικού του backend ως σύστημα ελέγχου ταυτότητας (για παράδειγμα, μια δική του βάση δεδομένων χρηστών) και στη συνέχεια να δημιουργήσει ένα custom token, το οποίο η Firebase θα χρησιμοποιήσει για να επαληθεύσει την ταυτότητα του χρήστη. Αυτός είναι ο τρόπος που αξιοποιείται στα πλαίσια του παρόντος, όπως θα αναλυθεί στο Κεφάλαιο 3.

Τέλος, αξίζει να τονιστεί ότι η Firebase Authentication προσφέρει μια ασφαλή πλατφόρμα αυθεντικοποίησης που συμμορφώνεται με βέλτιστες πρακτικές ασφαλείας, συμπεριλαμβανομένης της κρυπτογράφησης κωδικών πρόσβασης και της επαλήθευσης ταυτότητας χρηστών [29].

2.8 Η πλατφόρμα Sendgrid

2.8.1 Βασικά Χαρακτηριστικά- Πλεονεκτήματα

Η SendGrid είναι μια πλατφόρμα νέφους (cloud platform) που επιτρέπει την αποστολή μηνυμάτων ηλεκτρονικού ταχυδρομείου (emails) εύκολα και αξιόπιστα μέσω του API της. Χρησιμοποιείται ευρέως για την αποστολή τόσο συναλλακτικών emails (π.χ., επιβεβαιώσεις λογαριασμών, ειδοποιήσεις) όσο και για μαζικές καμπάνιες email marketing. Παρέχει εργαλεία παρακολούθησης, ασφαλείς διασυνδέσεις και υψηλή ταχύτητα παράδοσης. Τα κύρια χαρακτηριστικά της πλατφόρμας είναι τα ακόλουθα:

- Εύκολη Ενσωμάτωση μέσω API: Παρέχει έτοιμα API και SDKs για πολλές γλώσσες προγραμματισμού (Node.js, Python, Java, κ.λπ.), που διευκολύνουν την αποστολή emails από εφαρμογές.

- Αξιοπιστία και Υψηλό Ποσοστό Παράδοσης: Η SendGrid χρησιμοποιεί τεχνολογίες όπως το DomainKeys Identified Mail (DKIM) και το Sender Policy Framework (SPF) για να αυξήσει την ασφάλεια και να βελτιώσει την παράδοση των emails, αποφεύγοντας τα φίλτρα ανεπιθύμητης/ενοχλητικής αλληλογραφίας (spam).
- Πολλαπλοί Τρόποι Αποστολής: Υποστηρίζει τόσο τα συναλλακτικά emails (π.χ., ειδοποιήσεις πτώσης, ενημερώσεις λογαριασμού) όσο και τις μαζικές καμπάνιες.
- Αναφορές και Παρακολούθηση: Παρέχει λεπτομερή αναφορά σχετικά με την κατάσταση των emails, όπως αν έχει γίνει η παράδοση, αν έχουν ανοιχτεί, ή αν έχουν υπάρξει προβλήματα κατά την αποστολή (bounces).
- Ασφάλεια: Υποστηρίζει κρυπτογράφηση μέσω Transport Layer Security (TLS), προσφέροντας ασφαλή μετάδοση των emails.

Τα πλεονεκτήματα της Sendgrid συνοψίζονται λοιπόν στην αξιοπιστία, αφού εξασφαλίζει υψηλά ποσοστά παράδοσης και ασφάλεια δεδομένων, στην Ικανότητα αποστολής μεγάλου όγκου emails χωρίς δυσκολία και στις λεπτομερείς αναφορές και τα στατιστικά σχετικά με την κατάσταση των emails .

2.8.2 Μειονεκτήματα

Παρά τα πολλά πλεονεκτήματα, η SendGrid έχει και ορισμένα μειονεκτήματα που πρέπει να ληφθούν υπόψη [30]:

- Κόστος: Η SendGrid προσφέρει ένα δωρεάν πλάνο, αλλά όταν αυξάνεται ο όγκος των email, το κόστος μπορεί να γίνει σημαντικό, ειδικά αν στέλνεις πολλά emails σε τακτική βάση. Το μοντέλο τιμολόγησης βασίζεται στον αριθμό των emails που αποστέλλονται και μπορεί να είναι ακριβό για εφαρμογές με μεγάλες λίστες παραληπτών.
- Δυσκολία Παραμετροποίησης: Παρά το γεγονός ότι παρέχει εύκολη ενσωμάτωση μέσω API, η πλήρης παραμετροποίηση της SendGrid, ειδικά για μαζικές καμπάνιες ή σύνθετες συναλλαγές, μπορεί να απαιτεί εξειδικευμένες γνώσεις και χρόνο για να βελτιστοποιηθεί σωστά.
- Αυστηρή Πολιτική Καταπολέμησης Spam: Η αυστηρή πολιτική καταπολέμησης του spam είναι ένα θετικό χαρακτηριστικό, αλλά σε κάποιες περιπτώσεις μπορεί να οδηγήσει σε ακούσιο φιλτράρισμα ή περιορισμό του λογαριασμού σου, ειδικά αν αποστέλλονται πολλά emails που αναπηδούν (bounces) ή οδηγούν σε αναφορές ως spam.

Στα πλαίσια του παρόντος έργου θα μας απασχολήσει μόνο το τρίτο από αυτά, όπως θα εξηγηθεί στο Κεφάλαιο 3.

2.9 Bluetooth

2.9.1 Γενικά

Το Bluetooth είναι ένα πρότυπο ασύρματης τεχνολογίας μικρής εμβέλειας που χρησιμοποιείται για την ανταλλαγή δεδομένων μεταξύ σταθερών και κινητών συσκευών σε μικρές αποστάσεις και για τη δημιουργία προσωπικών δικτύων περιοχής (Personal Area Networks - PANs), χωρίς την ανάγκη για καλωδιακή σύνδεση μεταξύ των συσκευών και χωρίς την ανάγκη για μια κεντρική συσκευή, όπως ένας δρομολογητής ή ένα σημείο πρόσβασης. Έχει σχεδιαστεί κυρίως για χαμηλή κατανάλωση ενέργειας και στην πιο ευρέως χρησιμοποιούμενη λειτουργία του, η ισχύς μετάδοσης περιορίζεται στα 2,5 milliwatt, προσφέροντας πολύ μικρή εμβέλεια έως 10 μέτρα. Χρησιμοποιεί ραδιοκύματα εξαιρετικά υψηλής συχνότητας (Ultra High Frequency - UHF) στις ζώνες ISM (industrial, scientific, and medical), από 2,402 GHz έως 2,48 GHz. Χρησιμοποιείται ευρέως σε κινητές συσκευές, ακουστικά, πληκτρολόγια και άλλες συσκευές που απαιτούν ασύρματη επικοινωνία ως εναλλακτική στις ενσύρματες συνδέσεις, καθώς και για την ανταλλαγή αρχείων μεταξύ κοντινών συσκευών. Επειδή οι συσκευές χρησιμοποιούν ένα σύστημα επικοινωνίας μέσω ραδιοκυμάτων (broadcast), δεν χρειάζεται να βρίσκονται σε οπτική επαφή μεταξύ τους. Ωστόσο, μια σχεδόν οπτική ασύρματη διαδρομή (quasi-optical wireless path) πρέπει να είναι εφικτή.

2.9.2 Bluetooth Low Energy (BLE)

Το Bluetooth Low Energy (BLE), επίσης γνωστό ως Bluetooth Smart, είναι μια παραλλαγή του Bluetooth που έχει σχεδιαστεί για να καταναλώνει πολύ χαμηλή ενέργεια, καθιστώντας το ιδανικό για εφαρμογές και συσκευές που απαιτούν παρατεταμένη διάρκεια ζωής της μπαταρίας. Το BLE χρησιμοποιεί την ίδια ζώνη συχνοτήτων και προσφέρει παρόμοια εμβέλεια με το κλασικό Bluetooth αλλά με σημαντικά μικρότερη κατανάλωση ενέργειας, κάτι που το καθιστά ιδανικό για φορητές συσκευές, αισθητήρες και άλλες συσκευές IoT, όπου η εξοικονόμηση ενέργειας είναι κρίσιμη.

Το BLE προσφέρει δύο βασικούς τύπους επικοινωνίας:

- Μετάδοση (Advertising): Σε αυτήν τη λειτουργία, η συσκευή μεταδίδει δεδομένα χωρίς να απαιτείται η δημιουργία μιας σταθερής σύνδεσης. Αυτό επιτρέπει στις συσκευές να ανακαλύπτονται εύκολα από άλλες οι οποίες αναζητούν συγκεκριμένες υπηρεσίες.
- Λειτουργία Σύνδεσης (Connected Mode): Σε αυτήν τη λειτουργία, δύο συσκευές συνδέονται μεταξύ τους για αμφίδρομη ανταλλαγή δεδομένων. Αυτός ο τρόπος είναι χρήσιμος όταν απαιτείται συνεχής και αξιόπιστη επικοινωνία μεταξύ των συσκευών.

2.9.2.1 Generic Attribute Profile (GATT)

Το BLE είναι σχεδιασμένο να υποστηρίζει εφαρμογές που απαιτούν διαλειτουργικότητα μεταξύ διαφόρων συσκευών, εξασφαλίζοντας παράλληλα την ασφάλεια των δεδομένων. Μια συσκευή BLE περιέχει έναν πίνακα δεδομένων που ονομάζεται Πίνακας Χαρακτηριστικών, ο οποίος μπορεί να προσπελαστεί από άλλες συνδεδεμένες συσκευές με διάφορους δυνατούς τρόπους . Αυτός ο πίνακας δεδομένων και οι τρόποι με τους οποίους μπορεί να αξιοποιηθεί εμπίπτουν σε μια τεχνική περιοχή του Bluetooth που ονομάζεται το Γενικό Προφίλ Χαρακτηριστικών (Generic Attribute Profile – GATT). Το GATT αποτελεί μία προδιαγραφή της συσκευής, που περιγράφει τη δομή των δεδομένων και των υπηρεσιών που παρέχονται από τη συσκευή αυτή. Με άλλα λόγια, το προφίλ Bluetooth καθορίζει τον τρόπο με τον οποίο μια συσκευή εμφανίζεται σε άλλες συσκευές όσον αφορά τα χαρακτηριστικά της και τις λειτουργίες που μπορεί να εκτελέσει και τον τρόπο τον τρόπο με τον οποίο άλλες συσκευές μπορούν να επικοινωνούν μαζί της.

Το GATT αποτελείται από τριών ειδών στοιχεία, τις Υπηρεσίες (Services), τις Χαρακτηριστικές (Characteristics) και τους Περιγραφείς (Descriptors), οργανωμένα σε μια ιεραρχία με τις Υπηρεσίες στην κορυφή και τις Περιγραφές στη βάση. Οι Υπηρεσίες περιέχουν υποχρεωτικά μία ή περισσότερες Χαρακτηριστικές. Μία Χαρακτηριστική μπορεί να διαθέτει κανέναν ή προαιρετικά κάποιους Περιγραφείς [31].

2.9.2.2 Υπηρεσίες

Μια Υπηρεσία είναι μία ομαδοποίηση για λογικά συσχετισμένα στοιχεία δεδομένων Bluetooth. Αυτά τα στοιχεία δεδομένων ονομάζονται στην πραγματικότητα Χαρακτηριστικές. Μια Υπηρεσία μπορεί να θεωρηθεί ως ο ιδιοκτήτης των Χαρακτηριστικών που περιέχει. Συχνά, μια Υπηρεσία αντιπροσωπεύει ένα συγκεκριμένο χαρακτηριστικό (π.χ. ένα υλικό χαρακτηριστικό) μιας συσκευής, όπως τα κουμπιά ή ένα συγκεκριμένο αισθητήρα. Ένα παράδειγμα Υπηρεσίας που ορίζεται από το Bluetooth SIG (Special Interest Group) είναι η Υπηρεσία Πληροφοριών Συσκευής (Device Information Service), η οποία, όπως υποδηλώνει το όνομα, είναι ένας περιέκτης για διάφορα στοιχεία πληροφοριών σχετικά με τη συσκευή, όπως ο κατασκευαστής και ο σειριακός αριθμός της.

2.9.2.3 Χαρακτηριστικές

Οι Χαρακτηριστικές είναι στοιχεία δεδομένων που σχετίζονται με μια συγκεκριμένη εσωτερική κατάσταση της συσκευής (για παράδειγμα το τρέχον επίπεδο μπαταρίας) ή ίσως με κάποια κατάσταση του περιβάλλοντος, την οποία η συσκευή μπορεί να μετρήσει χρησιμοποιώντας έναν αισθητήρα (για παράδειγμα τη θερμοκρασία περιβάλλοντος). Μερικές φορές, τα Χαρακτηριστικά αντιπροσωπεύουν δεδομένα διαμόρφωσης, όπως η επιθυμητή συχνότητα κάποιας μέτρησης. Σε οποιαδήποτε από αυτές τις περιπτώσεις, ο τρόπος με τον οποίο μια συσκευή μπορεί να εκθέσει τέτοια δεδομένα σε άλλες συσκευές για να τα χρησιμοποιήσουν μέσω Bluetooth είναι καταστήσει διαθέσιμα ως Χαρακτηριστικές. Ένα παράδειγμα Χαρακτηριστικής που έχει οριστεί από το Bluetooth SIG είναι η Συμβολοσειρά Σειριακού Αριθμού, η οποία

περιέχεται στην υπηρεσία Πληροφοριών Συσκευής. Οι Χαρακτηριστικές περιέχουν τα εξής μέρη: έναν τύπο, μια τιμή, τις ιδιότητες και τις άδειες.

Ο τύπος είναι μια τιμή UUID που υποδεικνύει σε ποιον συγκεκριμένο τύπο Χαρακτηριστικού ανήκει η Χαρακτηριστική. Η τιμή είναι η τιμή του σχετικού στοιχείου δεδομένων κατάστασης.

Οι ιδιότητες ορίζουν τι μπορεί να κάνει μια άλλη συσκευή με τη Χαρακτηριστική μέσω Bluetooth όσον αφορά διάφορες καθορισμένες λειτουργίες, όπως ανάγνωση, εγγραφή ή ειδοποίηση (read, write, notify). Η ανάγνωση μίας Χαρακτηριστικής σημαίνει μεταφορά της τρέχουσας τιμής της από τον Πίνακα Χαρακτηριστικών στη συνδεδεμένη συσκευή μέσω Bluetooth. Η εγγραφή επιτρέπει στη συνδεδεμένη συσκευή να αλλάξει αυτήν την τιμή στον πίνακα κατάστασης. Οι ειδοποιήσεις είναι ένας ειδικός τύπος μηνύματος, το οποίο μια συσκευή μπορεί να στείλει σε μια άλλη κάθε φορά που η τιμή της σχετικής Χαρακτηριστικής αλλάζει ή ίσως περιοδικά, ελεγχόμενη από ένα χρονοδιακόπτη.

Οι Άδειες έχουν σχέση με την ασφάλεια και περιγράφουν περαιτέρω τις συνθήκες ασφαλείας που πρέπει να πληρούνται πριν δοθεί πρόσβαση για ανάγνωση ή εγγραφή στη Χαρακτηριστική.

2.9.2.4 Περιγραφείς

Οι Περιγραφείς περιέχουν μεταδεδομένα που είτε εμπλουτίζουν τις λεπτομέρειες που σχετίζονται με την Χαρακτηριστική στην οποία ανήκει ο Περιγραφέας είτε επιτρέπουν τη διαμόρφωση μιας συμπεριφοράς που περιλαμβάνει αυτή τη Χαρακτηριστική. Για παράδειγμα, τα μηνύματα ειδοποίησης ενεργοποιούνται ή απενεργοποιούνται χρησιμοποιώντας μια ειδική περιγραφή που ονομάζεται Περιγραφέας Διαμόρφωσης Χαρακτηριστικής Πελάτη.

2.10 Η γλώσσα προγραμματισμού Python

Η Python είναι μια δυναμική, υψηλού επιπέδου, γενικού σκοπού, πολυπλατφορμική γλώσσα προγραμματισμού που έχει κερδίσει τεράστια δημοτικότητα τα τελευταία χρόνια, κυρίως λόγω της απλότητας και της ευελιξίας της. Είναι γνωστή για τη σαφή και λιτή σύνταξή της, καθώς και το τεράστιο οικοσύστημα βιβλιοθηκών που παρέχει, χαρακτηριστικά που την καθιστούν εύκολη στην εκμάθηση και στη χρήση. Η Python αποτελεί μια από τις πιο δημοφιλείς γλώσσες προγραμματισμού για ένα ευρύ φάσμα εφαρμογών, από την ανάπτυξη λογισμικού έως την επιστήμη δεδομένων και τη μηχανική μάθηση. Η Python είναι ιδιαίτερα γνωστή για τις δυνατότητές της στην ανάλυση δεδομένων, χάρη σε ισχυρές βιβλιοθήκες όπως το NumPy και το Pandas [32].

2.10.1 Η βιβλιοθήκη NumPy

Η NumPy (Numerical Python) είναι μια βιβλιοθήκη που επιτρέπει την αποδοτική δημιουργία και διαχείριση πολυδιάστατων πινάκων (arrays) και την εκτέλεση

αριθμητικών υπολογισμών υψηλής ταχύτητας. Παρέχει εργαλεία για την εκτέλεση μαθηματικών πράξεων, γραμμικής άλγεβρας, και στατιστικών υπολογισμών. Είναι η βάση πάνω στην οποία χτίζονται πολλές άλλες βιβλιοθήκες ανάλυσης δεδομένων στην Python, καθιστώντας την ένα βασικό εργαλείο για την επεξεργασία μεγάλων συνόλων αριθμητικών δεδομένων [33].

2.10.2 Η βιβλιοθήκη Pandas

Η Pandas είναι μια βιβλιοθήκη που χτίστηκε πάνω στο NumPy και ειδικεύεται στην ανάλυση και διαχείριση δεδομένων. Παρέχει πιο εξειδικευμένα εργαλεία και δομές για την ανάλυση και διαχείριση δεδομένων, όπως DataFrames και Series, που διευκολύνουν την οργάνωση και τον χειρισμό μεγάλων συνόλων δεδομένων. Η Pandas χρησιμοποιείται ευρέως για την εισαγωγή, καθαρισμό και ανάλυση δεδομένων, και προσφέρει εργαλεία για την εύκολη εκτέλεση λειτουργιών όπως φιλτράρισμα, ομαδοποίηση και συγχώνευση δεδομένων [34].

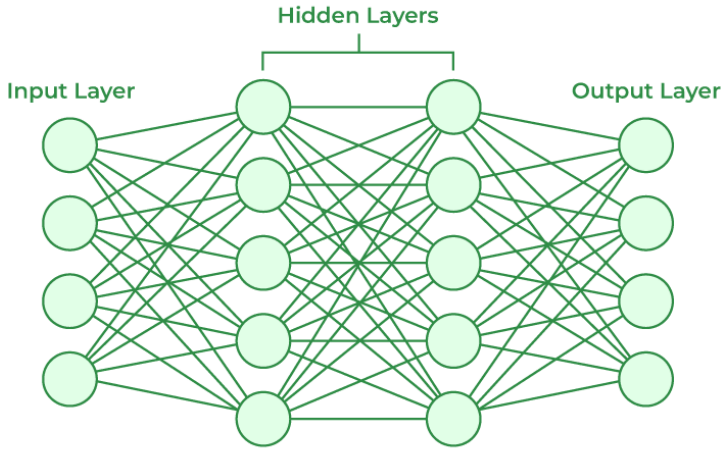
2.11 Νευρωνικά Δίκτυα

2.11.1 Γενικά

Ένα Νευρωνικό Δίκτυο (Neural Network – NN, επίσης γνωστό ως Τεχνητό Νευρωνικό Δίκτυο – Artificial Neural Network – ANN ή ως Προσομοιωμένο Νευρωνικό Δίκτυο – Simulated Neural Network – SNN) είναι ένα αλγοριθμικό μοντέλο Μηχανικής Μάθησης που λαμβάνει αποφάσεις με τρόπο εμπνευσμένο από τη λειτουργία του ανθρώπινου εγκεφάλου, χρησιμοποιώντας διαδικασίες που μιμούνται τον τρόπο με τον οποίο οι βιολογικοί νευρώνες συνεργάζονται για να αναγνωρίσουν φαινόμενα, να ζυγίσουν επιλογές και να καταλήξουν σε συμπεράσματα.

Ένα ANN αποτελείται από συνδεδεμένες μονάδες ή κόμβους που ονομάζονται τεχνητοί νευρώνες, οι οποίοι προσομοιώνουν προσεγγιστικά τους νευρώνες ενός βιολογικού εγκεφάλου. Αυτοί συνδέονται με ακμές, οι οποίες προσομοιώνουν τις συνάψεις στον εγκέφαλο και οργανώνονται σε στρώματα ή αλλιώς επίπεδα (layers) κόμβων:

- ένα στρώμα εισόδου (input layer), από το οποίο εισέρχονται τα δεδομένα στο δίκτυο. Κάθε κόμβος (νευρώνας) σε αυτό το επίπεδο αντιπροσωπεύει ένα χαρακτηριστικό του συνόλου δεδομένων.
- ένα ή περισσότερα κρυφά στρώματα (hidden layers), τα οποία εκτελούν τους υπολογισμούς και τις μετασχηματίσεις στα δεδομένα εισόδου. Το πλήθος των κρυφών στρωμάτων εξαρτάται από την πολυπλοκότητα του προβλήματος.
- και ένα στρώμα εξόδου (output layer), το οποίο παράγει το τελικό αποτέλεσμα ή πρόβλεψη του μοντέλου, είτε πρόκειται για μια κατηγορία (στην περίπτωση της κατηγοριοποίησης - classification) είτε για μια συνεχή τιμή (στην περίπτωση της παλινδρόμησης - regression)



Εικόνα 6: Τυπική Δομή ενός Νευρωνικού Δικτύου με δύο κρυφά στρώματα

Κάθε τεχνητός νευρώνας λαμβάνει σήματα από συνδεδεμένους νευρώνες, τα επεξεργάζεται και στέλνει ένα σήμα σε άλλους συνδεδεμένους νευρώνες. Το "σήμα" είναι ένας πραγματικός αριθμός, και η έξοδος κάθε νευρώνα υπολογίζεται μέσω μιας μη γραμμικής συνάρτησης του αθροίσματος των εισόδων του, η οποία ονομάζεται συνάρτηση ενεργοποίησης (activation function). Η ισχύς του σήματος σε κάθε σύνδεση καθορίζεται από έναν συντελεστή βάρους (weight), ο οποίος προσαρμόζεται κατά τη διαδικασία μάθησης.

Τα νευρωνικά δίκτυα βασίζονται σε δεδομένα για να μάθουν και να βελτιώσουν την ακρίβειά τους με την πάροδο του χρόνου μέσω μιας διαδικασίας που ονομάζεται εκπαίδευση (training). Αφού ρυθμιστούν για μέγιστη ακρίβεια, είναι ισχυρά εργαλεία στην επιστήμη των υπολογιστών και την τεχνητή νοημοσύνη, επιτρέποντας την ταξινόμηση και την ομαδοποίηση δεδομένων με υψηλή ταχύτητα. Τα νευρωνικά δίκτυα χρησιμοποιούνται ευρέως για εφαρμογές που αφορούν την αναγνώριση προτύπων, την προγνωστική μοντελοποίηση, την ανάλυση δεδομένων και την επίλυση προβλημάτων στην τεχνητή νοημοσύνη. Μπορούν να μαθαίνουν από την εμπειρία και να εξάγουν συμπεράσματα από ένα πολύπλοκο και φαινομενικά άσχετο σύνολο πληροφοριών [35], [36].

2.11.2 Πολυεπίπεδα Νευρωνικά Δίκτυα (Multilayer Perceptrons) και Οπισθοδιάδοση (Backpropagation)

Το Πολυεπίπεδο Νευρωνικό Δίκτυο (Multilayer Perceptron - MLP) είναι ένας τύπος πλήρως συνδεδεμένου ανατροφοδοτικού νευρωνικού δικτύου, όπου κάθε νευρώνας ενός επιπέδου συνδέεται με όλους τους νευρώνες του επόμενου επιπέδου με μη γραμμική συνάρτηση ενεργοποίησης, οργανωμένους σε τουλάχιστον τρία επίπεδα, και διακρίνεται για την ικανότητά του να ξεχωρίζει δεδομένα που δεν είναι γραμμικά διαχωρίσιμα.
Η εκπαίδευση του MLP πραγματοποιείται μέσω αλγορίθμων βελτιστοποίησης, όπως ο αλγόριθμος της οπισθοδιάδοσης (backpropagation), που επιτρέπει στο δίκτυο να διορθώνει τα σφάλματα κατά τη διάρκεια της διαδικασίας μάθησης. Η

οπισθοδιάδοση είναι μια αποδοτική εφαρμογή του κανόνα της αλυσίδας στα Νευρωνικά Δίκτυα. Υπολογίζει τη μερική παράγωγο μιας συνάρτησης απωλειών (loss function) ως προς τα βάρη του δικτύου για ένα μεμονωμένο παράδειγμα εισόδου-εξόδου, και το κάνει με αποδοτικό τρόπο, υπολογίζοντας την κλίση (gradient) ανά ένα στρώμα τη φορά, με επανάληψη προς τα πίσω από το τελευταίο στρώμα για να αποφευχθούν περιττοί υπολογισμοί ενδιάμεσων όρων στον κανόνα της αλυσίδας. Αυτό μπορεί να επιτευχθεί με χρήση δυναμικού προγραμματισμού.

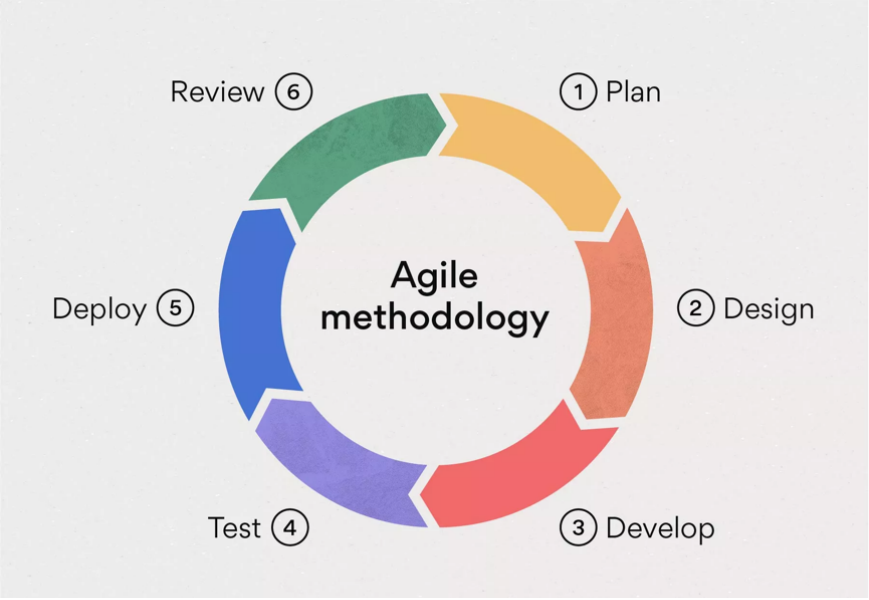
Κεφάλαιο 3: Δομή και μεθοδολογία υλοποίησης του συστήματος

3.1 Εισαγωγή

Σε αυτή την ενότητα παρουσιάζονται με λεπτομέρεια τα συστατικά μέρη του συστήματος, η αλληλεξάρτησή τους και ο τρόπος με τον οποίο συνδέονται και αλληλοεπιδρούν μεταξύ τους. Επιπλέον, περιγράφεται η μεθοδολογία που ακολουθήθηκε κατά τη διάρκεια της ανάπτυξης του λογισμικού. Η μέθοδος που χρησιμοποιήθηκε είναι η ευέλικτη μέθοδος ανάπτυξης (Agile), η οποία θα αναλυθεί διεξοδικά στην αμέσως επόμενη υποενότητα. Τέλος, αναλύεται και το κόστος που απαιτήθηκε για την υλοποίηση του έργου.

3.2 Ευέλικτη Μεθοδολογία (Agile)

Για την υλοποίηση του παρόντος συστήματος εφαρμόστηκε η ευέλικτη ανάπτυξη λογισμικού (agile). Η μεθοδολογία αυτή περιλαμβάνει διάφορες τεχνικές ανάπτυξης, όπου οι απαιτήσεις και οι λύσεις διαμορφώνονται μέσα από τη συνεργασία αυτοοργανωμένων και διατμηματικών ομάδων. Προωθεί την προσαρμογή του σχεδιασμού, την εξελικτική ανάπτυξη, την έγκαιρη παράδοση, τη συνεχή βελτίωση και ενθαρρύνει την ταχεία ανταπόκριση στις αλλαγές. Με απλά λόγια, επικεντρώνεται στον γρήγορο, σταδιακό σχεδιασμό και την παράδοση ενός αρχικού προϊόντος, το οποίο παρακολουθείται συνεχώς. Παραδίδεται το συντομότερο δυνατό στον πελάτη για να δοθεί ανατροφοδότηση, μετά την οποία το έργο επανασχεδιάζεται με βάση τις νέες ανάγκες, συνεχίζοντας τη διαδικασία μέχρι να ικανοποιηθούν πλήρως οι απαιτήσεις του πελάτη. Φυσικά στα πλαίσια του παρόντος δεν υφίσταται πελάτης, αντίθετα τον ρόλο της ανατροφοδότησης διαδραμάτισε ο επιβλέπων καθηγητής.



Εικόνα 7: Η Μεθοδολογία Agile

Όπως παρουσιάζεται και στην παραπάνω εικόνα, η μεθοδολογία διακρίνεται στα εξής στάδια, τα οποία ακολουθούνται επαναληπτικά μέχρι την ολοκλήρωσω του έργου:

- Πλάνο (Plan): Το πρώτο βήμα στην ευέλικτη μεθοδολογία είναι η δημιουργία ενός πλάνου, στο οποίο γίνεται ο καθορισμός των βασικών λειτουργιών και απαιτήσεων και ο εκτιμώμενος χρόνος για την σχεδίαση και υλοποίηση αυτών. Αυτές πρέπει να είναι υλοποιήσιμες, ρεαλιστικές, πρωτοποριακές και να εξυπηρετούν τις ανάγκες των χρηστών, αποφεύγοντας περιττά στοιχεία και πλεονασμούς.
- Σχεδιασμός (Design): Ακολουθεί ο σχεδιασμός του συστήματος, ο οποίος βασίζεται στις καθορισμένες απαιτήσεις του πρώτου σταδίου και στη διαθεσιμότητα πόρων, όπως χρήματα, εξοπλισμός, εργαλεία και χρόνος. Κάθε νέα απαίτηση επηρεάζει τον σχεδιασμό, οδηγώντας στην ανάγκη εύρεσης της κατάλληλης δομής για την υλοποίηση της.
- Ανάπτυξη (Development): Το τρίτο στάδιο αποτελεί η ανάπτυξη, η οποία έγινε σταδιακά, με την ολοκλήρωση κάθε φορά συγκεκριμένου τμήματος, ώστε αυτό να ενσωματώνεται ομαλά στο σύνολο, εξασφαλίζοντας ομοιογένεια. Παράλληλα γινόντουσαν και οι τυχόν απαραίτητες αλλαγές στα ήδη υπάρχοντα τμήματα. Το GitHub χρησιμοποιήθηκε για την ασφαλή αποθήκευση του κώδικα, την παρακολούθηση της εξέλιξης και τη διατήρηση ιστορικού και παλαιότερων εκδόσεων.
- Δοκιμή και Διάθεση (Testing and Deployment) Μετά την ανάπτυξη κάθε λειτουργίας, ακολουθεί δοκιμή αρχικά σε τοπικό περιβάλλον και στη συνέχεια υπό πραγματικές συνθήκες, για να διασφαλιστεί η ορθότητα και η αποδοτικότητα της εφαρμογής. Το deployment γινόταν μόνο μετά από επιτυχημένο testing.
- Αξιολόγηση (Review): Το τελικό στάδιο περιλαμβάνει την αξιολόγηση των αποτελεσμάτων και του τρόπου ανάπτυξης, με στόχο τον εντοπισμό προβλημάτων και την αποφυγή τους σε μελλοντικές λειτουργίες. Πολλοί επανέλεγχοι πραγματοποιήθηκαν για να διασφαλιστεί η ακρίβεια και η ορθότητα του τελικού συστήματος.

3.3 Μέθοδος συλλογής δεδομένων επιταχυνσιόμετρου για το Νευρωνικό Δίκτυο

Η συλλογή δεδομένων για την εκπαίδευση και την αξιολόγηση του νευρωνικού δικτύου βασίστηκε σε υπάρχοντα δημόσια σύνολα δεδομένων, καθώς στα πλαίσια μίας διπλωματικής εργασίας οι υφιστάμενοι χρονικοί και υλικοί περιορισμοί καθιστούν δύσκολη την εύρεση υποκειμένων (subjects) και τη δημιουργία των κατάλληλων πειραματικών και ρεαλιστικών συνθηκών. Η δομή, η μορφή και το μέγεθος των δεδομένων αυτών, τα οποία θα παρουσιαστούν αναλυτικά στο κεφάλαιο 4, είναι όμοια με τα πραγματικά δεδομένα λαμβάνονται από το Puck.js. Οι έτοιμες μετρήσεις εμπλουτίστηκαν και από μικρό αριθμό (~100) προσωπικών μετρήσεων. Ο στόχος ήταν να δημιουργηθεί ένα πλούσιο και διαφοροποιημένο σύνολο δεδομένων που θα επιτρέψει στο σύστημα να διακρίνει με ακρίβεια πτώσεις

από άλλες καθημερινές δραστηριότητες, προσομοιώνοντας όσο το δυνατόν καλύτερα τις πραγματικές μετρήσεις που θα λαμβάναμε από το Puck.js ώστε να συμπεράνουμε αν είναι εφικτή η δημιουργία ενός μικρού αλλά ικανοποιητικού μοντέλου Νευρωνικού Δικτύου που να μπορεί να χρησιμοποιηθεί με τους περιορισμένους πόρους του Puck.js ή παρόμοιων μικροσυσκευών.

3.3.1 Υπάρχοντα Σύνολα Δεδομένων

Χρησιμοποιήθηκαν τα ακόλουθα δύο δημόσια σύνολα δεδομένων:

Α) Το πρώτο και μεγαλύτερο είναι ένα σύνολο που παρέχεται από το Kaggle [37]. Το Kaggle είναι μια πλατφόρμα διαγωνισμών επιστήμης δεδομένων και μια διαδικτυακή κοινότητα για επιστήμονες δεδομένων και επαγγελματίες μηχανικής μάθησης, η οποία ανήκει στην Google LLC. Το σύνολο αυτό περιέχει δεδομένα από επιταχυνσιόμετρα για την ανίχνευση πτώσεων, χωρισμένα σε διάφορα σενάρια. Κάθε σενάριο περιλαμβάνει έναν διαφορετικό τύπο δραστηριότητας ή πτώσης, επιτρέποντας τη διαφοροποίηση των κινήσεων για τη βελτίωση της ακρίβειας του συστήματος. Τα σενάρια είναι τα εξής:

- Downsit: Αφορά την περίπτωση όπου το άτομο κάθεται από όρθια θέση. Αυτή η δραστηριότητα μπορεί να προκαλέσει ασαφείς μετρήσεις καθώς η επιβράδυνση κατά την καθιστή διαδικασία μπορεί να μοιάζει με πτώση.
- Freefall: Εδώ, το άτομο βρίσκεται σε ελεύθερη πτώση, η οποία καταγράφεται από το επιταχυνσιόμετρο. Αυτή η περίπτωση είναι κρίσιμη για την ανίχνευση πραγματικών πτώσεων καθώς συνδέεται με απότομες αλλαγές στην ταχύτητα και στην επιτάχυνση.
- Runfall: Προσομοιώνει την πτώση κατά το τρέξιμο. Σε αυτήν την περίπτωση, το άτομο τρέχει και πέφτει απότομα. Οι μετρήσεις αποτυπώνουν την έντονη αρχική κίνηση και την ξαφνική διακοπή λόγω της πτώσης.
- Runsit: Εδώ, το άτομο τρέχει και στη συνέχεια κάθεται. Αυτή η δραστηριότητα δημιουργεί έντονες μεταβολές στην επιτάχυνση κατά το τρέξιμο και επιβράδυνση κατά την καθιστή διαδικασία, μοιάζοντας εν μέρει με πτώση.
- Walkfall: Αφορά την περίπτωση όπου το άτομο περπατάει και πέφτει ξαφνικά. Αυτή η περίπτωση είναι πιο συνηθισμένη σε άτομα μεγαλύτερης ηλικίας ή με κινητικά προβλήματα, καθώς η απώλεια ισορροπίας κατά το περπάτημα μπορεί να προκαλέσει πτώση.
- Walksit: Σε αυτήν την περίπτωση, το άτομο περπατάει και έπειτα κάθεται. Όπως και στα προηγούμενα σενάρια, υπάρχει μια σταδιακή επιβράδυνση καθώς το άτομο κάθεται, η οποία θα μπορούσε να μοιάζει με πτώση.

Αυτά τα δεδομένα είναι ιδιαίτερα χρήσιμα για την εκπαίδευση του νευρωνικού δικτύου, καθώς καλύπτουν διαφορετικά σενάρια πτώσης και καθημερινών δραστηριοτήτων, επιτρέποντας την ακριβέστερη αναγνώριση πτώσεων.

Β) Το δεύτερο σύνολο είναι ένα ευρέως χρησιμοποιούμενο σύνολο δεδομένων για την ανίχνευση πτώσεων, το οποίο παρέχεται από το Διεπιστημονικό Κέντρο Υπολογιστικής Μοντελοποίησης Πανεπιστήμιο του Rzeszow στην Πολωνία [38]. Αυτό

το σύνολο δεδομένων περιέχει πληροφορίες από αισθητήρες που τοποθετούνται σε άτομα που προσομοιώνουν πτώσεις. Συγκεκριμένα, οι μετρήσεις προέρχονται από RGB κάμερες, κάμερες βάθους (depth) και επιταχυνσιόμετρα. Στα πλαίσια του παρόντος έργου χρησιμοποιήθηκαν μόνο τα δεδομένα από επιταχυνσιόμετρα. Το σύνολο περιλαμβάνει τόσο δεδομένα πτώσεων (fall sequences) όσο και δεδομένα μη πτώσεων που αντιστοιχούν σε καθημερινές δραστηριότητες (Activities of Daily Living).

3.3.2 Προσωπικές μετρήσεις με χρήση του Puck.js

Επιπρόσθετα των δημοσίων συνόλων δεδομένων, πραγματοποιήθηκε και μικρός αριθμός (~100) προσωπικών μετρήσεων χρησιμοποιώντας το Puck.js, το οποίο – όπως έχει ήδη παρουσιαστεί - ενσωματώνει επιταχυνσιόμετρο για την καταγραφή της κίνησης. Οι μετρήσεις αυτές πραγματοποιήθηκαν με προσεκτικά σχεδιασμένες δοκιμές, ώστε να καλύψουν σενάρια πραγματικών πτώσεων και καθημερινών δραστηριοτήτων που θα μπορούσαν να προκαλέσουν εσφαλμένα σήματα. Οι προσωπικές μετρήσεις διακρίνονται στα παρακάτω δύο σενάρια:

- Προσομοιώσεις Πτώσεων: Διάφορες πτώσεις προσομοιώθηκαν από διαφορετικές θέσεις και καταστάσεις, όπως πτώση από όρθια θέση, πτώση κατά το περπάτημα, και πτώση από καθιστή θέση. Οι μετρήσεις πτώσεων εκτελέστηκαν σε αίθουσα πολεμικών τεχνών η οποία διαθέτει δάπεδο tatami, προκειμένου να εξασφαλιστεί η εκτέλεση όσο το δυνατόν ρεαλιστικότερων πτώσεων, διασφαλίζοντας παράλληλα την αποφυγή τραυματισμών. Το ειδικό αυτό δάπεδο προσέφερε ένα ασφαλές περιβάλλον για τις δοκιμές, επιτρέποντας την καταγραφή δεδομένων από πραγματικές πτώσεις χωρίς τον κίνδυνο τραυματισμών.
- Καθημερινές Δραστηριότητες: Περιλαμβάνουν δραστηριότητες όπως το περπάτημα, το κάθισμα, η ανάβαση σκαλοπατιών και η εναλλαγή μεταξύ όρθιας και καθιστής θέσης. Τα δεδομένα αυτά ήταν σημαντικά για την αναγνώριση μη πτώσεων και τη διαχείριση ψευδών θετικών. Οι μετρήσεις μη πτώσεων – καθημερινών δραστηριοτήτων εκτελέστηκαν σε ποικίλα περιβάλλοντα.

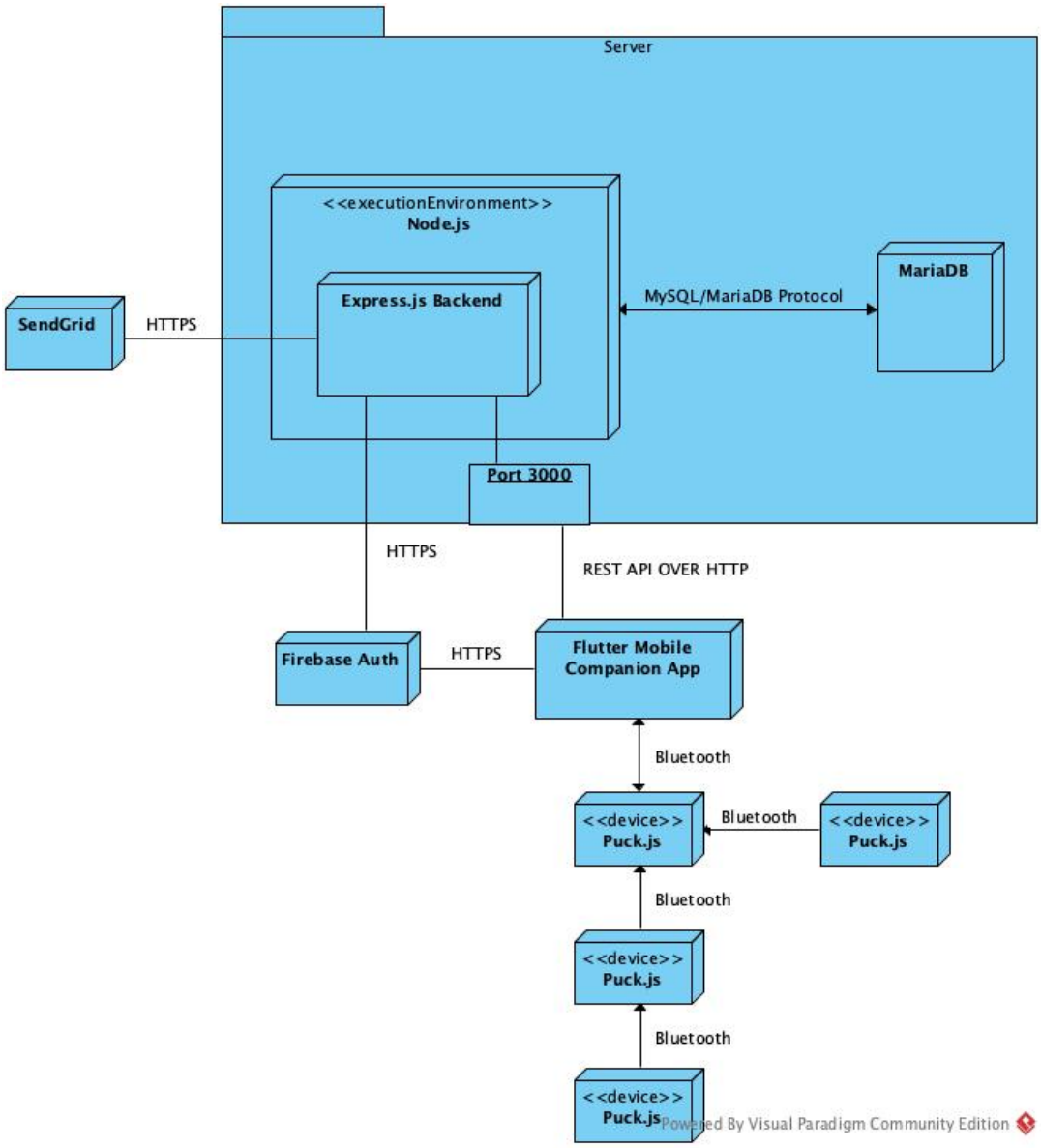
Για σκοπούς μεγαλύτερης επιστημονικής εγκυρότητας και ακρίβειας στα δεδομένα, αναφέρεται ότι το υποκείμενο είναι αρσενικού φύλου, ηλικίας 25 ετών, ύψους 169.8 εκατοστόμετρων και βάρους 69.6 κιλών. Τα φυσιολογικά χαρακτηριστικά του υποκειμένου ελήφθησαν κατά την έναρξη των δοκιμών.

Τα χαρακτηριστικά αυτά είναι σημαντικά καθώς το φύλο, η ηλικία, το ύψος και το βάρος μπορούν να επηρεάζουν την κίνηση του σώματος και την επιτάχυνσή του, επομένως επηρεάζουν και τις μετρήσεις του επιταχυνσιόμετρου.

3.4 Η Αρχιτεκτονική του συστήματος συνολικά

Το σύστημα που πραγματεύεται η παρούσα εργασία απαρτίζεται από 5 θεμελιώδη και μεταξύ τους ανεξάρτητα μέρη, τα οποία συνοπτικά περιγράφονται ως εξής: την συσκευή Puck.js η οποία λαμβάνει μηνύματα πανικού μέσω πατήματος του κουμπιού

από το χρήστη και παρακολουθεί τις κινήσεις του φέροντος αυτήν με σκοπό τον εντοπισμό πτώσεων, τη συνοδευτική εφαρμογή κινητού η οποία είναι υπεύθυνη για τη λήψη τέτοιων κρίσιμων μηνυμάτων από το Puck.js και τη μεταφορά αυτών στον



Εικόνα 8: UML Deployment Διάγραμμα ολόκληρου του Συστήματος

εξυπηρετητή, τον εξυπηρετητή ο οποίος παρέχει λειτουργίες διαχείρισης και ανάκτησης των δεδομένων, επικοινωνίας με τις λοιπές βοηθητικές υπηρεσίες και διαχωρίζεται περεταίρω στην backend εφαρμογή και στη βάση δεδομένων MariaDB, την υπηρεσία Firebase Authentication, που -σε συνεργασία με το backend- παρέχει τη δυνατότητα ασφαλούς αυθεντικοποίησης των χρηστών και την πλατφόρμα SendGrid μέσω της οποίας στέλνονται τα ειδοποιητήρια mails για γεγονότα πανικού ή πτώσης στις έμπιστες επαφές του χρήστη. Η Αρχιτεκτονική παρουσιάζεται , στο σύνολό της, στο παρακάτω Διάγραμμα Deployment:

Πρόκειται για μια μοντέρνα web αρχιτεκτονική, σχεδιασμένη από τον γράφοντα, βασισμένη σε RESTful API backend με διασύνδεση υπηρεσιών, όπως η

αυθεντικοποίηση μέσω Firebase, η επικοινωνία με τη βάση δεδομένων MariaDB, και η αποστολή emails μέσω του SendGrid. Παρόλο που το σύστημα περιλαμβάνει τη διασύνδεση πολλαπλών υπηρεσιών, δεν ακολουθείται κλασική Service-Oriented Architecture (Αρχιτεκτονική Προσανατολισμένη στις Υπηρεσίες - SOA). Παρομοιάζει, χωρίς να ταυτίζεται, με μία αρχιτεκτονική μικροϋπηρεσιών (microservices) λόγω της ανεξαρτησίας των στοιχείων, της χρήσης εξωτερικών υπηρεσιών νέφους για βασικές λειτουργίες και του αποκεντρωμένου τρόπου διαχείρισης των υπηρεσιών, ο οποίος προσφέρει ευελιξία και δυνατότητα κλιμάκωσης. Όλα αυτά τα μέρη συνεργάζονται μέσω σαφώς καθορισμένων API και το καθένα έχει τον δικό του ρόλο στη συνολική λειτουργικότητα του συστήματος. Στην επόμενη ενότητα, θα περιγραφεί διεξοδικά κάθε συστατικό μέρος της αρχιτεκτονικής αυτής καθώς και ο τρόπος που αυτά αλληλεπιδρούν και ανταλλάσσουν δεδομένα.

3.5 Ανάλυση των συστατικών μερών του συστήματος

3.5.1 Συσκευές Puck.js

Η συσκευή Puck.js διαδραματίζει σημαντικό ρόλο στο παρόν σύστημα, λειτουργώντας τόσο ως ανιχνευτής πανικού και πτώσεων όσο και ως κόμβος επικοινωνίας. Τα συμβάντα πανικού ανιχνεύονται μέσω του πατήματος του κουμπιού και τα συμβάντα πτώσεων ανιχνεύονται χρησιμοποιώντας τον ενσωματωμένο επιταχυνσιόμετρο. Τα δεδομένα του επιταχυνσιόμετρου διέρχονται από το Νευρωνικό Δίκτυο, το οποίο τα κατηγοριοποιεί ως πτώση ή μη πτώση. Τα γεγονότα αυτά αποθηκεύονται σε ένα πίνακα, ο οποίος επιπρόσθετα περιέχει πιθανώς δεδομένα που έχουν μεταδοθεί από άλλες συσκευές Puck.js. Η μορφή αυτού του πίνακα θα παρουσιαστεί στο κεφάλαιο 4.

Ως κόμβος επικοινωνίας το Puck.js διαδραματίζει διττό ρόλο: μπορεί να λαμβάνει μηνύματα από άλλα Puck.js και να μεταφέρει τα δικά του - και πιθανώς άλλων - μηνύματα στην συνοδευτική εφαρμογή η οποία εκτελείται σε μία συσκευή κινητού. Το Puck.js εκπέμπει (advertising) συνεχώς μία υπηρεσία με UUID 20020001-53e4-11ee-8c99-0242ac120002. Η υπηρεσία αυτή φέρει δύο χαρακτηριστικές, την 20020002-53e4-11ee-8c99-0242ac120002 η οποία προσφέρεται για ανάγνωση από τη συνοδευτική εφαρμογή και παρέχει ενημερώσεις (notifications) και την 20020003-53e4-11ee-8c99-0242ac120002 η οποία προσφέρεται για εγγραφή από άλλα Puck.js. Οι αλληλεπιδράσεις είναι συγκεκριμένα οι εξής:

- Όταν το Puck.js λάβει πάτημα κουμπιού ή εντοπίσει συμβάν πτώσης, εκπέμπει μία υπηρεσία, την 0xFFFF, μέσω της οποίας δηλώνει ότι έχει νέα δεδομένα. Στη συνέχεια αναμένει 10 δευτερόλεπτα τη συνοδευτική εφαρμογή να διαβάσει τα δεδομένα από την 0xFFFF και να συνδεθεί στο Puck.js, ώστε εκείνο να σταματήσει την εκπομπή της 0xFFFF και να ξεκινήσει τη μετάδοση των δεδομένων μέσω της 20020002-53e4-11ee-8c99-0242ac120002 στην συνοδευτική εφαρμογή. Λόγω των τεχνικών περιορισμών που επιβάλλονται από το BLE και το Puck.js, (μόνο 20 bytes ανά ανάγνωση-εγγραφή χαρακτηριστικής) η μετάδοση ολόκληρου τον πίνακα γίνεται στοιχείο-στοιχείο. Το Puck.js διατρέχει τον πίνακα, ενημερώνει την τιμή της

χαρακτηριστικής με το τρέχον στοιχείο και αναμένει διάστημα 500ms προς αποφυγή κατακλυσμού του παραλήπτη – κινητού ή απώλειας κάποιας εγγραφής. Στη συνέχεια ενημερώνει την τιμή βάζοντας το επόμενο στοιχείο κοκ. Το τέλος της μετάδοσης σηματοδοτείται από κατάλληλη εγγραφή τέλους, όπως θα παρουσιαστεί αναλυτικότερα στο Κεφάλαιο 4

- Αν το Puck.js δεν λάβει σύνδεση από τη συνοδευτική εφαρμογή εντός του χρονικού διαστήματος των 10 δευτερολέπτων, αναζητεί άλλες συσκευές Puck.js μέσω του ονόματός τους και των υπηρεσιών που εκπέμπουν. Αν δεν βρει καμία τέτοια συσκευή ή βρει αλλά η σύνδεση σε αυτήν αποτύχει, αναμένει για διάστημα 2 λεπτών και ξαναδοκιμάζει. Ο συνολικός αριθμός επαναπροσπαθειών ορίστηκε στις δέκα και για απλότητα στην υλοποίηση όταν φτάσει αυτό το όριο το Puck.js σταματάει να προσπαθεί. Αντίθετα, αν σε κάποια από αυτές τις προσπάθειες εντοπιστεί κατάλληλη συσκευή και το Puck.js συνδεθεί επιτυχώς σε αυτήν, τότε ο αποστολέας γράφει ολόκληρο τον πίνακά του, στοιχείο-στοιχείο, στη χαρακτηριστική 20020003-53e4-11ee-8c99-0242ac120002 του παραλήπτη. Μεταξύ δύο διαδοχικών εγγραφών ο αποστολέας αναμένει διάστημα 500ms προς αποφυγή κατακλυσμού του παραλήπτη, ενώ το τέλος της μετάδοσης σηματοδοτείται από κατάλληλη εγγραφή τέλους, όπως θα παρουσιαστεί αναλυτικότερα στο Κεφάλαιο 4. Για ευκολία στην υλοποίηση τυχόν σφάλμα σε οποιοδήποτε στάδιο της διαδικασίας αυτής οδηγεί σε ολική επαναπροσπάθεια, όπως ακριβώς και στην περίπτωση μη εύρεσης συσκευής, εφόσον το όριο δεν έχει ξεπεραστεί. Αντίστοιχα, στην πλευρά του παραλήπτη, η λήψη της εγγραφής τέλους πυροδοτεί ακριβώς την ίδια διαδικασία με το πάτημα του κουμπιού ή τον εντοπισμό πτώσης, η οποία αναλύθηκε παραπάνω.

3.5.2 Συνοδευτική εφαρμογή Flutter – Κινητή συσκευή

Η συνοδευτική εφαρμογή είναι ο συνδετικός κρίκος του συστήματος, αφού αποτελεί το σταυροδρόμι επικοινωνίας μεταξύ Puck.js, χρηστών και υπηρεσιών Backend. Εκτελείται σε κινητές συσκευές που τρέχουν λειτουργικό σύστημα Android (έκδοση 5.0 ή νεότερη) ή iOS (έκδοση 12.0 ή νεότερη) και έχει ως στόχο να συγκεντρώνει δεδομένα από τα Puck.js, να ειδοποιεί για συμβάντα πανικού και πτώσης, και να διαχειρίζεται τις ειδοποιήσεις προς τους έμπιστους χρήστες - επαφές. Η εφαρμογή απαιτεί ενεργοποιημένες συνεχώς τις συνδέσεις Bluetooth και διαδικτύου (είτε μέσω Wi-Fi είτε μέσω δεδομένων κινητής τηλεφωνίας). Ακολουθούν οι κύριες λειτουργίες και αλληλεπιδράσεις της εφαρμογής:

- Εγγραφή και Αυθεντικοποίηση Χρηστών μέσω της Firebase: Για τη λειτουργία αυτή απαιτείται επικοινωνία της εφαρμογής τόσο με την υπηρεσία της Firebase όσο και με το δικό μας Backend. Η επικοινωνία της εφαρμογής με τη Firebase γίνεται μέσω του πρωτοκόλλου HTTPS ενώ η επικοινωνία με το backend μας γίνεται με το πρωτόκολλο HTTP για λόγους επίσπευσης ολοκλήρωσης της εργασίας. Κατά την είσοδο (login) ή την εγγραφή (register) του χρήστη στην εφαρμογή, γίνεται αρχικά επικοινωνία με το backend για την ανάκτηση, την αντιστοίχιση και την ταυτοποίηση του χρήστη στην περίπτωση της εισόδου, και για την καταχώριση των δεδομένων του στη βάση δεδομένων

μας στην περίπτωση της εγγραφής. Το backend δημιουργεί ένα προσαρμοσμένο διακριτικό JWT (custom JWT token) το οποίο αποστέλλεται στην εφαρμογή ως απάντηση στο αίτημα HTTP που έκανε εκείνη για είσοδο ή εγγραφη. Στη συνέχεια η εφαρμογή αποστέλλει το προσαρμοσμένο αυτό διακριτικό στη Firebase και λαμβάνει ως απάντηση ένα διακριτικό ανανέωσης (refresh token) κι ένα JWT διακριτικό ταυτότητας (JWT ID token). Τα ID tokens στο εξής αποστέλλονται μέσα σε οποιοδήποτε επόμενο αίτημα - ενέργεια κάνει η εφαρμογή στο backend, ώστε να μπορεί να ταυτοποιείται ο χρήστης με ασφάλεια. Τα ID tokens έχουν μικρή διάρκεια και λήγουν μετά το πέρας της μίας ώρας. Το Firebase SDK φροντίζει η εφαρμογή να επικοινωνεί ανά διαστήματα με τη Firebase στο παρασκήνιο, στέλνοντάς της το refresh token και λαμβάνοντας ως απάντηση νέα ID tokens, ώστε να ανανεώσει τη συνεδρία (session) της παρέχοντας έτσι συνεδρίες μακράς διαρκείας χωρίς να προκύπτουν διακοπές στην εμπειρία χρήσης.

- Διαχείριση Δεδομένων: Για τις λειτουργίες αυτές πραγματοποιείται επικοινωνία της εφαρμογής με το δικό μας Backend μέσω HTTP αιτημάτων. Η εφαρμογή προσφέρει δυνατότητες προσθήκης και αφαίρεσης έμπιστων επαφών-χρηστών, καθώς και συσκευών Puck.js.
- Διαχείριση ειδοποιήσεων – κόμβος επικοινωνίας μεταξύ backend και Puck.js: Πέρα από τις λειτουργίες που αναφέρθηκαν και αφορούν την επικοινωνία του χρήστη με τη Firebase και το backend, ο σημαντικότερος ίσως ρόλος της εφαρμογής είναι ότι αποτελεί τον κόμβο επικοινωνίας μεταξύ των συσκευών Puck.js και του backend. Η εφαρμογή αναζητεί συνεχώς Puck.js συσκευές που εκπέμπουν την υπηρεσία 0xFFFF που όπως αναφέραμε αφορά την ύπαρξη νέων δεδομένων επείγοντος περιστατικού σε ένα Puck.js. Επειδή η διαδικασία αναζήτησης συσκευών είναι ακριβή σε πόρους, η εφαρμογή στην πραγματικότητα δεν σκανάρει αδιάλειπτα, αλλά αντίθετα εκτελεί μία αναζήτηση διάρκειας των 15 δευτερολέπτων κάθε 10 δευτερόλεπτα, προς αποφυγή μονοπώλησης της CPU. Εφόσον εντοπίσει κατάλληλες συσκευές συνδέεται μέσω Bluetooth με καθεμία από αυτές και διαβάζει τα δεδομένα τους. Στα πλαίσια της παρούσας εργασίας, για λόγους απλότητας η εφαρμογή διαβάζει από οποιοδήποτε Puck.js και δεν πραγματοποιείται κάποια διαδικασία ζευγαροποίησης - εμπιστοσύνης μεταξύ τους. Η ανάγνωση γίνεται μέσω εγγραφής (subscription) με ειδοποιήσεις στην χαρακτηριστική 20020002-53e4-11ee-8c99-0242ac120002 του αντίστοιχου Puck.js. Με αυτό τον τρόπο η εφαρμογή λαμβάνει ειδοποίηση όταν η τιμή της χαρακτηριστικής μεταβληθεί, διαβάζει την νέα τιμή και την αποθηκεύει σε έναν πίνακα. Όταν ληφθεί κατάλληλη εγγραφή τέλους, μέσω της οποίας επιβεβαιώνεται ότι σημαίνει ότι έχει πλέον ληφθεί ολόκληρος ο πίνακας από το εκάστοτε Puck.js, τα δεδομένα αυτά αποστέλλονται στο backend, το οποίο αναλαμβάνει να διαχειριστεί την αποστολή ειδοποιήσεων μέσω email προς τις έμπιστες επαφές.

3.5.3 Εξυπηρετητής – Backend

Το backend του συστήματος παίζει κρίσιμο ρόλο στη διαχείριση των δεδομένων, την αυθεντικοποίηση των χρηστών και την επικοινωνία με εξωτερικές υπηρεσίες, όπως η Firebase για την αυθεντικοποίηση και η SendGrid για την αποστολή ειδοποιήσεων μέσω μηνυμάτων ηλεκτρονικού ταχυδρομείου. Το backend, δηλαδή, είναι ένα Υποσύστημα Επεξεργασίας και Διαχείρισης Δεδομένων που αποτελείται από δύο συστατικά μέρη, τη βάση δεδομένων και το REST API του Express.js, το οποίο διαδραματίζει το ρόλο του κόμβου επικοινωνίας μεταξύ εφαρμογής - βάσης δεδομένων και εφαρμογής – εξωτερικών υπηρεσιών. Ακολουθούν οι κύριες λειτουργίες και αλληλεπιδράσεις των μερών του Backend:

- Εγγραφή και Αυθεντικοποίηση Χρηστών μέσω της Firebase: Για τη λειτουργία αυτή απαιτείται επικοινωνία του REST API τόσο με την υπηρεσία της Firebase όσο και με την εφαρμογή. Η επικοινωνία του REST API με τη Firebase γίνεται μέσω του πρωτοκόλλου HTTPS ενώ η επικοινωνία με την εφαρμογή γίνεται με το πρωτόκολλο HTTP για λόγους επίσπευσης ολοκλήρωσης της εργασίας. Το REST API παρέχει κατάλληλα endpoints (μία πιθανή μετάφραση είναι «τερματικά σημεία» αλλά ο όρος είναι αδόκιμος οπότε σε όλη την έκταση του παρόντος χρησιμοποιείται η αγγλική ορολογία) με POST μέθοδο για την είσοδο (login) ή την εγγραφή (register) του χρήστη στην εφαρμογή, στα οποία η εφαρμογή στέλνει αίτημα. Όπως ήδη αναλύθηκε στην προηγούμενη υποενότητα κατά την επιτυχία της ταυτοποίησης ή εγγραφής, το backend δημιουργεί ένα custom JWT token με το οποίο απαντά στο αίτημα HTTP που έκανε εκείνη. Στη συνέχεια, όπως αναφέρθηκε, η εφαρμογή εκδίδει με τη βοήθεια της Firebase ένα JWT ID token. Το ID token περιλαμβάνεται στην κεφαλίδα 'token' κάθε αιτήματος (το οποίο δεν είναι εγγραφή ή είσοδος) που στέλνει το frontend στο backend. Το backend προτού προχωρήσει στην επεξεργασία του αιτήματος, στέλνει το ID token στην υπηρεσία της Firebase προκειμένου εκείνη να το επικυρώσει. Αν η επαλήθευση είναι επιτυχής, το backend προχωράει στην επεξεργασία, αλλιώς απαντά με τον κατάλληλο κωδικό κατάστασης σφάλματος, όπως αυτός υπαγορεύεται από το πρότυπο HTTP. Περισσότερα σχετικά με τους κωδικούς σφάλματος θα παρουσιαστούν στο επόμενο κεφάλαιο.
- Διαχείριση Δεδομένων – Επικοινωνία με τη βάση δεδομένων: Το REST API προσφέρει endpoints για βασικές λειτουργίες CRUD (Create, Read, Update, Delete – Δημιουργία, Ανάγνωση, Ενημέρωση, Διαγραφή) σχετικά με χρήστες, έμπιστες επαφές, αιτήματα προσθήκης επαφών, και Puck.js που ανήκουν στους χρήστες. Η εφαρμογή επικοινωνεί μέσω HTTP αιτημάτων με αυτά τα endpoints. Αφού το αίτημα αυθεντικοποιηθεί με τον τρόπο που παρουσιάστηκε παραπάνω, πραγματοποιείται επικύρωση (validation) των παραμέτρων εισόδου για λόγους ασφαλείας και ακεραιότητας των δεδομένων. Αν η επικύρωση αυτή αποτύχει, το backend απαντά με κατάλληλο κωδικό σφάλματος. Αντίθετα, αν η επικύρωση είναι επιτυχής, το REST API εν συνεχεία επικοινωνεί με τη βάση δεδομένων μέσω σύνδεσης TCP (Transmission Control Protocol) πραγματοποιώντας κατάλληλα SQL ερωτήματα για να εκτελεστούν οι ζητούμενες λειτουργίες ανάγνωσης, εγγραφής, ενημέρωσης ή διαγραφής, ενώ κατά περίπτωση λειτουργίας εκτελεί και δικούς του υπολογισμούς. Η επιτυχής ολοκλήρωση του αιτήματος

οδηγεί σε απάντηση με κατάλληλο HTTP κωδικό επιτυχίας και τα ζητούμενα δεδομένα στο σώμα της απάντησης (response body), ενώ η αποτυχία εξυπηρέτησης του αιτήματος για οποιοδήποτε λόγο πέρα των προαναφερθέντων παράγει απάντηση με τον κατάλληλο κωδικό κατάστασης σφάλματος 500, ο οποίος αντιστοιχεί σε απάντηση γενικού σφάλματος διακομιστή.

- Αποστολή ειδοποιητηρίων μηνυμάτων ηλεκτρονικού ταχυδρομείου μέσω της Sendgrid: Όπως παρουσιάστηκε στην προηγούμενη ενότητα, η εφαρμογή λαμβάνει δεδομένα για συμβάντα πτώσης ή πανικού από τις συσκευές Puck.js. Κάθε φορά που λαμβάνει τέτοια δεδομένα, επικοινωνεί με το backend μέσω του κατάλληλου POST endpoint που παρέχεται από το REST API. Αφού το HTTP αίτημα αυθεντικοποιηθεί και επικυρωθεί με τις διαδικασίες που εξηγήθηκαν παραπάνω, το backend αρχικά επικοινωνεί με τη βάση δεδομένων ώστε να συλλέξει τις λίστες αποστολέων-παραληπτών που αντιστοιχούν σε έμπιστες επαφές, συνθέτει τα δεδομένα που αφορούν τα κατάλληλα emails που πρέπει να αποσταλλούν και εν συνεχεία στέλνει τα δεδομένα αυτά στο API της υπηρεσίας SendGrid, μέσω HTTPS. Η υπηρεσία της SendGrid, με τη σειρά της αποστέλλει τα ζητούμενα emails στους αντίστοιχους παραλήπτες που αποτελούν έμπιστες επαφές του εκάστοτε ιδιοκτήτη του Puck.js.

3.6 Κόστος υλοποίησης της διπλωματικής εργασίας

Για την ανάπτυξη της παρούσας διπλωματικής εργασίας επιλέχθηκαν κυρίως δωρεάν εργαλεία και πλάνα των υπηρεσιών, τα οποία ήταν επαρκή για τις ανάγκες της. Στις επόμενες υποενότητες ακολουθεί μια ανάλυση των κόστους για το υλικό και το λογισμικό που χρησιμοποιήθηκε.

3.6.1 Συσκευές Puck.js

Λόγω της ανάγκης για δοκιμή της επικοινωνίας μεταξύ Puck.js συσκευών, αγοράστηκαν 2 τέτοιες συσκευές από έναν επίσημο διανομέα. Η τιμή κάθε συσκευής από τον προμηθευτή αυτόν ανέρχεται στα 44.29€ συμπεριλαμβανομένου ΦΠΑ και δεν υπάρχουν επιπλέον συνδρομητικά κόστη ή άδειες χρήσης για το λογισμικό που τη συνοδεύει.

3.6.2 Περιβάλλοντα ανάπτυξης

Τα εργαλεία ανάπτυξης λογισμικού που χρησιμοποιήθηκαν, όπως το Visual Studio Code (VSCode) και το Espruino Web IDE, είναι δωρεάν και ανοιχτού κώδικα. Το Espruino Web IDE επιτρέπει τη σύνδεση με το Puck.js και την ανάπτυξη κώδικα JavaScript μέσω του προγράμματος περιήγησης, χωρίς να απαιτείται η εγκατάσταση πρόσθετων εργαλείων. Η χρήση αυτών των εργαλείων δεν επιφέρει κόστος.

3.6.3 Υλοποίηση της συνοδευτικής εφαρμογής

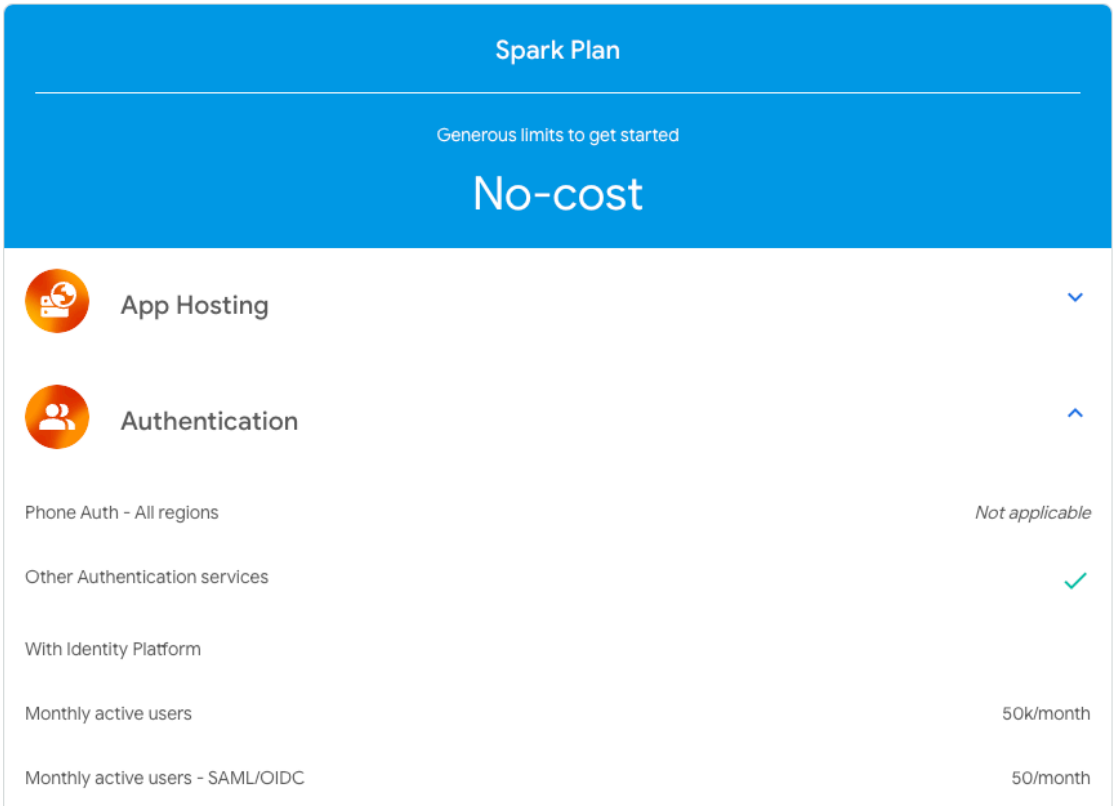
Η υλοποίηση της εφαρμογής βασίστηκε στη χρήση του Flutter, ενός πλαισίου ανάπτυξης ανοιχτού κώδικα, το οποίο είναι δωρεάν για χρήση. Η εφαρμογή δεν απαιτεί εξειδικευμένες λειτουργίες ή συγκεκριμένο λειτουργικό σύστημα και οικοσύστημα, οπότε οι δοκιμές πραγματοποιήθηκαν με τις προσωπικές συσκευές του γράφοντος, χωρίς να επέλθει κόστος.

3.6.4 Υλοποίηση του Backend

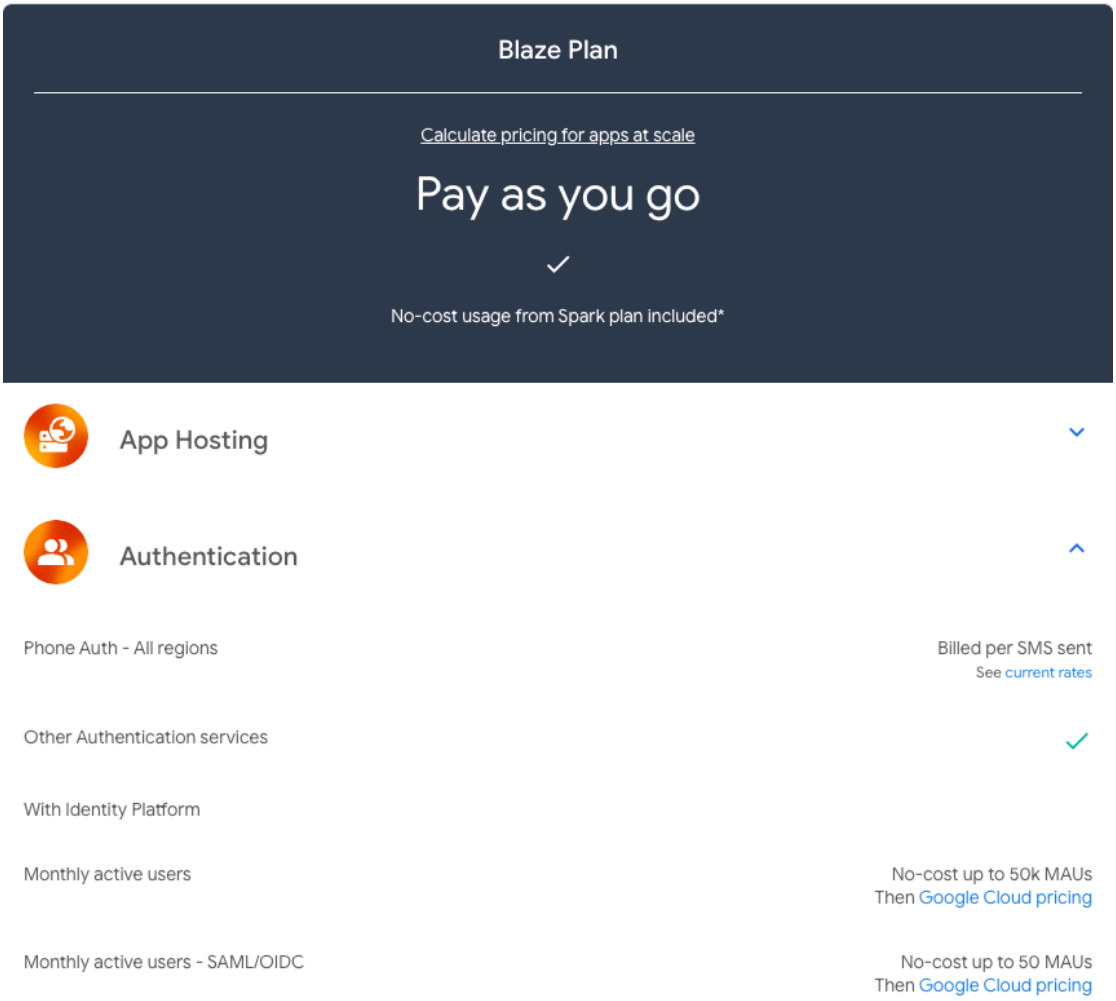
Το backend του συστήματος, αναπτύχθηκε χρησιμοποιώντας Node.js, Express.js και TypeScript, ενώ η βάση δεδομένων που επιλέχθηκε είναι η MariaDB. Όλα αυτά τα εργαλεία είναι δωρεάν και ανοιχτού κώδικα, χωρίς κόστη αδειών ή συνδρομών. Το backend και η βάση για της ανάγκες της εργασίας αρκούσε να φιλοξενούνται τοπικά στον υπολογιστή του γράφοντος, γεγονός που δεν επιφέρει επιπλέον κόστος φιλοξενίας σε εξωτερικούς servers.

3.6.5 Firebase (Spark Plan)

Η μόνη υπηρεσία της Firebase την οποία χρησιμοποιεί το σύστημα είναι η αυθεντικοποίηση. Η Firebase παρέχει εντελώς δωρεάν το πλάνο Spark, το οποίο περιλαμβάνει όλες τις βασικές υπηρεσίες που χρειάζονται για την αυθεντικοποίηση των χρηστών στα πλαίσια του παρόντος έργου. Παρακάτω φαίνονται συγκριτικά οι παροχές του δωρεάν πλάνου Spark και του “Pay as you go” (πληρωμή ανά χρήση) πλάνου Blaze σχετικά με την υπηρεσία αυθεντικοποίησης:



Εικόνα 9: Firebase Spark Plan



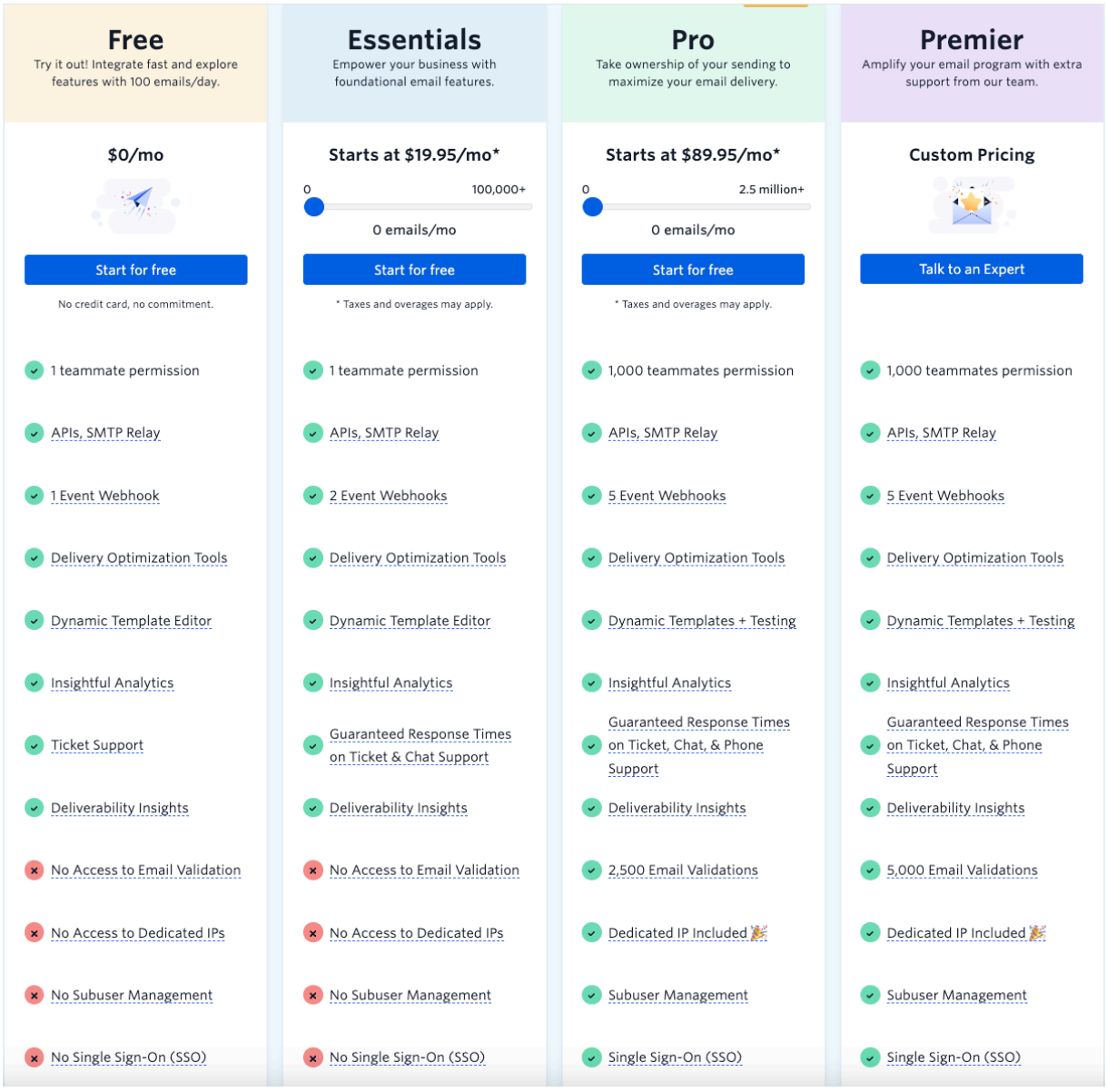
Εικόνα 10: Firebase Blaze Plan

Συγκεκριμένα διακρίνονται τα εξής:

- Phone Auth: Η υπηρεσία αυθεντικοποίησης μέσω τηλεφώνου δεν καλύπτεται στο δωρεάν πακέτο, αλλά δεν ήταν απαραίτητη για την παρούσα υλοποίηση και δεν χρησιμοποιήθηκε.
- Other Authentication services: Παρέχει δυνατότητες αυθεντικοποίησης χρηστών με email/κωδικό, Google, Facebook, και άλλους τρίτους παρόχους, ανώνυμη αυθεντικοποίηση και αυθεντικοποίηση μέσω παρόχου custom token χωρίς περιορισμούς στο Spark Plan, δηλαδή εντελώς δωρεάν. Αυτοί οι τρόποι αυθεντικοποίησης υπερκαλύπτουν τις ανάγκες της εργασίας αφού στα πλαίσια αυτής χρησιμοποιήθηκε μόνο ο τελευταίος.
- Monthly Active Users (MAUs): Υποστηρίζει έως 50.000 ενεργούς χρήστες μηνιαίως, που είναι υπεραρκετοί για τα πλαίσια της παρούσας εργασίας.
- Οι υπηρεσίες SAML (Security Assertion Markup Language), OIDC (OpenID Connect) και Identity Platform παρέχουν προηγμένες δυνατότητες αυθεντικοποίησης που ξεφεύγουν από τα πλαίσια του παρόντος.

3.6.6 SendGrid (Free Plan)

Η υπηρεσία SendGrid, που χρησιμοποιείται για την αποστολή email ειδοποιήσεων σε έμπιστες επαφές, προσφέρει δωρεάν πλάνο που καλύπτει πλήρως τις απαιτήσεις της παρούσας εργασίας. Παρακάτω παρουσιάζονται συγκριτικά οι παροχές όλων των πλάνων της SendGrid:



Εικόνα 11: Τα πλάνα της SendGrid

Συγκεκριμένα για το δωρεάν πλάνο διακρίνονται τα ακόλουθα:

- Δυνατότητα αποστολής έως 100 emails την ημέρα χωρίς κόστος.
- Πρόσβαση σε APIs και SMTP Relay για την αποστολή των emails.
- Εργαλεία παρακολούθησης και βελτιστοποίησης της παράδοσης των emails.

- Δυναμικό επεξεργαστή προτύπων για τη δημιουργία εξατομικευμένων μηνυμάτων.

Οι παραπάνω παροχές αρκούν για τις ανάγκες της εργασίας. Παρόλο που η δωρεάν έκδοση της SendGrid έχει περιορισμούς, όπως η έλλειψη πρόσβασης σε dedicated IPs και δυνατότητες email validation, αυτοί οι περιορισμοί δεν επηρεάζουν αρνητικά την υλοποίηση του συστήματος.

Κεφάλαιο 4: Παρουσίαση, ανάλυση και ερμηνεία του κώδικα υλοποίησης

4.1 Εισαγωγή

Σε αυτό το κεφάλαιο, παρουσιάζεται ο κώδικας υλοποίησης των διαφόρων μερών του συστήματος που αναπτύχθηκε στα πλαίσια αυτής της διπλωματικής εργασίας. Ο κώδικας αποτελεί το μεγαλύτερο μέρος της διαδικασίας υλοποίησης. Η ανάλυση και η ερμηνεία του έχουν ως στόχο να αναδείξουν τον τρόπο με τον οποίο υλοποιήθηκαν τα διάφορα υποσυστήματα, συνθέτοντας πλέον τις λεπτομέρειες υλοποίησης με τις τεχνολογίες και τις αρχιτεκτονικές αποφάσεις που ελήφθησαν κατά την ανάπτυξη όπως αυτές παρουσιάστηκαν στα Κεφάλαια 2 και 3 αντίστοιχα. Ο κώδικας θα αναλυθεί ανά υποσύστημα, παρουσιάζοντας λεπτομερώς τον τρόπο λειτουργίας κάθε μέρους και τις αλληλεπιδράσεις τους με το σύνολο του συστήματος.

Επιπλέον των παραπάνω, θα παρουσιαστεί η διαδικασία υλοποίησης του νευρωνικού δικτύου (MLP) για την ανίχνευση πτώσεων. Το νευρωνικό δίκτυο εκπαιδεύτηκε με δεδομένα επιταχυνσιόμετρου και ενσωματώθηκε στο σύστημα, προκειμένου να αναγνωρίζει μοτίβα που αντιστοιχούν σε πτώσεις. Θα αναλυθεί ο τρόπος εκπαίδευσης του μοντέλου, οι παράμετροι που χρησιμοποιήθηκαν και η αλληλεπίδρασή του με τα υπόλοιπα συστατικά του συστήματος.

4.2 Κώδικας στο Puck.js

4.2.1 Καθολικές Μεταβλητές

Στην παρακάτω εικόνα διακρίνονται οι καθολικές μεταβλητές του κώδικα, ο οποίες διατηρούν χρήσιμη πληροφορία για τη λειτουργία του:

```
const puckId = NRF.getAddress().replace(/:/g, ''); // Use device address as a unique puck ID
let array = []; // Array to store compact data: [puckId, fell, panicked]
let retryCount = 0; // Counter to track retry attempts
const maxRetries = 10; // Set maximum retry attempts to 10
let hasNewData = false;
let advertiseTimeout = null; // For tracking advertisement timeout
let lastSenderPuckId = null; // Track the last Puck that sent data
const f = require("Storage").open("general_log", "a");
```

Εικόνα 12: Καθολικές μεταβλητές κώδικα Puck.js

Αναλυτικά οι ρόλοι τους είναι οι εξής:

- puckId: Αποθηκεύει τη μοναδική διεύθυνση MAC (Media Access Control) της συσκευής την οποία λαμβάνει χρησιμοποιώντας τη συνάρτηση NRF.getAddress() που παρέχει ο Espruino, αφού αφαιρέσει τις άνω και κάτω τελείες από αυτήν (η μορφή μίας MAC διεύθυνσης είναι XX:XX:XX:XX:XX:XX). Η προκύπτουσα συμβολοσειρά χρησιμοποιείται για λόγους απλότητας ως το μοναδικό αναγνωριστικό για το puck σε όλα τα μέρη του συστήματος.
- array: Ένας πίνακας που αποθηκεύει δεδομένα έκτακτων συμβάντων στην περιεκτική μορφή [puckId, fell, panicked]. Το puckId αντιστοιχεί στο αναγνωριστικό του Puck.js που παρήγαγε την εγγραφή, η σημαία fell είναι 1 όταν πρόκειται για γεγονός πτώσης, αλλιώς είναι 0, και η σημαία panicked

είναι 1 αν πρόκειται για γεγονός πανικού-πατήματος κουμπιού, αλλιώς είναι 0.

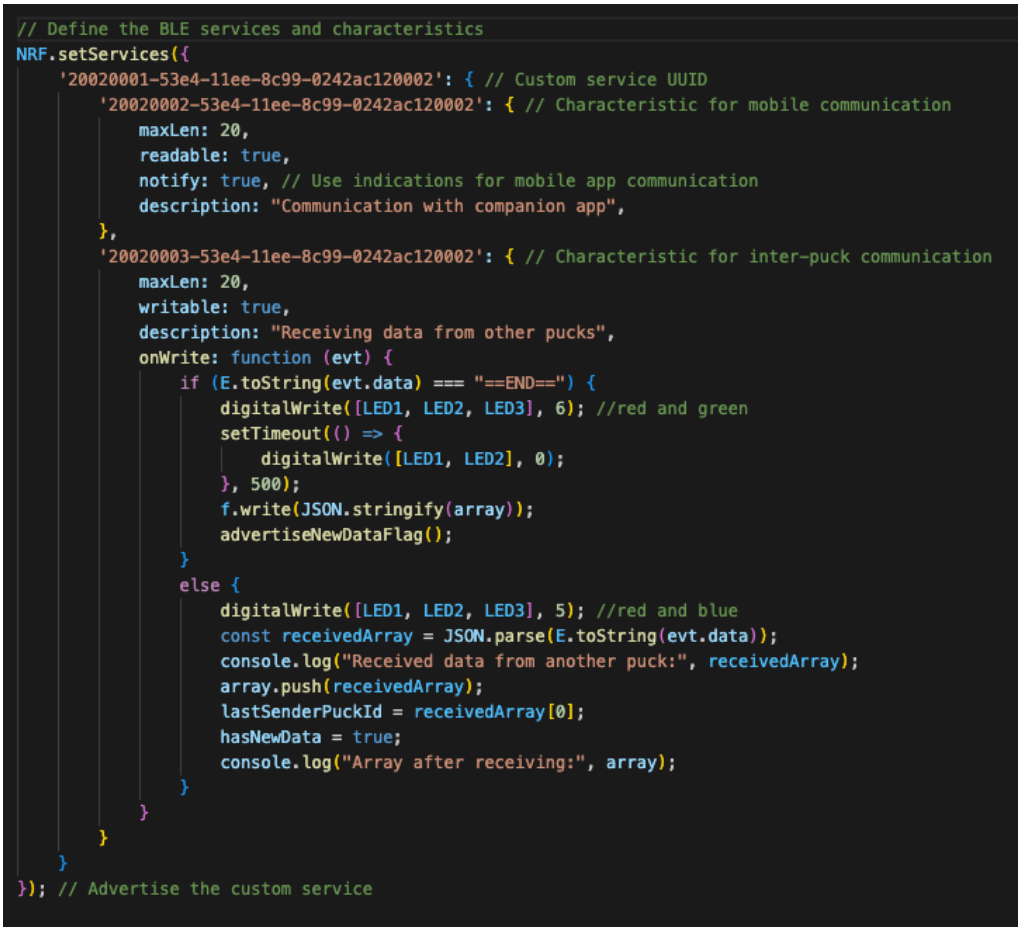
- retryCount: Μετρητής που παρακολουθεί τις προσπάθειες επανασύνδεσης ή αποστολής δεδομένων σε άλλο Puck.
- maxRetries: Η σταθερά που ορίζει τον μέγιστο αριθμό προσπαθειών εύρεσης ενός Puck-παραλήπτη σε 10.
- hasNewData: Σημαία που δείχνει αν υπάρχουν νέα δεδομένα για αποστολή.
- advertiseTimeout: Χρησιμοποιείται για την παρακολούθηση του χρονικού ορίου εκπομπής (advertising) των δεδομένων.
- lastSenderPuckId: Κρατά το αναγνωριστικό του τελευταίου Puck που έστειλε δεδομένα, για αποφυγή αποστολής δεδομένων πίσω σε αυτό. Η σημασία της μεταβλητής αυτής είναι τεράστια καθώς η απουσία της με μεγάλη πιθανότητα οδηγεί σε ατέρμονες βρόχους. Το απλούστερο παράδειγμα ενός τέτοιου βρόχου θα ήταν ένα σενάριο στο οποίο δύο Puck.js από τα οποία κανένα από τα δύο δεν έχει σύνδεση με συνοδευτική εφαρμογή, στέλνουν δεδομένα το ένα στο άλλο ατέρμονα γεμίζοντας τη RAM και οδηγώντας σε κατάρρευση (crash), αφού οι πίνακές τους μεγαλώνουν απεριόριστα.
- f: Ανοίγει ή δημιουργεί (αν αυτό δεν υφίσταται) ένα αρχείο καταγραφής στη μνήμη flash με όνομα "general_log" σε λειτουργία προσάρτησης (append mode) για σκοπούς αποσφαλμάτωσης σε περιπτώσης που η κονσόλα του Web IDE δεν είναι προσβάσιμη. Μία τέτοια περίπτωση είναι η αποσφαλμάτωση κατά τις δοκιμές της επικοινωνίας μεταξύ Puck.js και συνοδευτικής εφαρμογής, καθώς το Puck.js δεν είναι δυνατόν να συνδεθεί ως περιφερειακό ταυτόχρονα και με την εφαρμογή και με το Web IDE λόγω περιορισμών του BLE.

```
// Log data to flash storage
function logToStorage(message) {
    const timestamp = new Date().toISOString(); // Timestamp for the log
    const logEntry = `${timestamp}: ${message}\n`; // Log format

    // Append to storage (you can replace 'logfile' with your desired filename)
    f.write(logEntry);
}

function getLogs(callback) {
    const file = require("Storage").open("general_log", "r");
    let l = file.readLine();
    while (l !== undefined) {
        callback(l);
        l = file.readLine();
    }
}
```

Εικόνα 13: Συναρτήσεις εγγραφής και ανάγνωσης στο αρχείο καταγραφών της flash

4.2.2 Ρύθμιση BLE Υπηρεσιών

Η ρύθμιση των BLE υπηρεσιών, όπως αυτές περιγράφηκαν στο Κεφάλαιο 3, πραγματοποιείται με τον ακόλουθο κώδικα:



```
// Define the BLE services and characteristics
NRF.setServices({
    '20020001-53e4-11ee-8c99-0242ac120002': { // Custom service UUID
        '20020002-53e4-11ee-8c99-0242ac120002': { // Characteristic for mobile communication
            maxLen: 20,
            readable: true,
            notify: true, // Use indications for mobile app communication
            description: "Communication with companion app",
        },
        '20020003-53e4-11ee-8c99-0242ac120002': { // Characteristic for inter-puck communication
            maxLen: 20,
            writable: true,
            description: "Receiving data from other pucks",
            onWrite: function (evt) {
                if (E.toString(evt.data) === "==END==") {
                    digitalWrite([LED1, LED2, LED3], 6); //red and green
                    setTimeout(() => {
                        digitalWrite([LED1, LED2], 0);
                    }, 500);
                    f.write(JSON.stringify(array));
                    advertiseNewDataFlag();
                }
                else {
                    digitalWrite([LED1, LED2, LED3], 5); //red and blue
                    const receivedArray = JSON.parse(E.toString(evt.data));
                    console.log("Received data from another puck:", receivedArray);
                    array.push(receivedArray);
                    lastSenderPuckId = receivedArray[0];
                    hasNewData = true;
                    console.log("Array after receiving:", array);
                }
            }
        }
    }
}); // Advertise the custom service
```

Εικόνα 14: Ρύθμιση των υπηρεσιών BLE του Puck.js

Η συνάρτηση NRF.setServices παρέχεται από τον Espruino για ορισμό των υπηρεσιών, των χαρακτηριστικών και των περιγραφέων που εκπέμπει το Puck.js για επικοινωνία μέσω BLE. Μέσω του παραπάνω κώδικα ορίζεται η υπηρεσία με μοναδικό αναγνωριστικό 20020001-53e4-11ee-8c99-0242ac120002, η οποία περιλαμβάνει τις εξής δύο χαρακτηριστικές:

Χαρακτηριστική 20020002-53e4-11ee-8c99-0242ac120002: Όπως έχει ήδη αναφερθεί προσφέρεται για ανάγνωση από τη συνοδευτική εφαρμογή. Τίθεται ως αναγνώσιμη (readable), με ενεργοποιημένες τις ειδοποιήσεις (notify) και μέγιστο μήκος τιμής τα 20 bytes (maxLen).
Χαρακτηριστικό 20020003-53e4-11ee-8c99-0242ac120002: Όπως έχει ήδη αναφερθεί προσφέρεται για εγγραφή από άλλα Puck.js. Τίθεται ως εγγράψιμη (writable) με μέγιστο μήκος τιμής τα 20 bytes. Ορίζεται επίσης σώμα για τη συνάρτηση onWrite, η οποία καλείται κάθε φορά που πραγματοποιείται εγγραφή στη συγκεκριμένη χαρακτηριστική. Τα ληφθέντα-εγγραφέντα δεδομένα δίνονται ως το όρισμα evt στη συνάρτηση. Διακρίνονται δύο περιπτώσεις. Αν τα ληφθέντα δεδομένα ταυτίζονται με την συμβολοσειρά σηματοδότησης τέλους "==END==", καταγράφεται ότι ολοκληρώθηκε η αποστολή και γίνεται αποθήκευση του πίνακα συμβάντων στη flash και εκπομπή ύπαρξης νέων δεδομένων. Παράλληλα, με κλήση της συνάρτησης digitalWrite του Espruino ανάβει το πράσινο και κόκκινο LED για 500ms προς οπτική επιβεβαίωση της λήψης του τελικού πακέτου. Αντίθετα, αν τα ληφθέντα δεδομένα είναι κάτι διαφορετικό από την “==END==”, το ληφθέν JSON μετατρέπεται σε

αντικείμενο της JavaScript μέσω της JSON.parse και εγγράφεται στον πίνακα συμβάντων. Για λόγους απλότητας ο χειρισμός εξαιρέσεων στην περίπτωση που τα δεδομένα δεν είναι έγκυρο JSON έχει παραλειφθεί και θεωρείται πάντοτε έγκυρη μορφή δεδομένων. Παράλληλα καταγράφεται το αναγνωριστικό της εγγραφής ως ο τελευταίος αποστολέας δεδομένων, και ανάβει το κόκκινο και μπλε LED.

4.2.3 Εκπομπή ύπαρξης νέων δεδομένων

Η εκπομπή ύπαρξης νέων δεδομένων και η κατάργηση αυτής υλοποιούνται με τις παρακάτω συναρτήσεις:

```
function stopAdvertisingNewData() {
    hasNewData = false;
    NRF.setAdvertising({}, { name: 'Puck ' + puckId, showName: true, discoverable: true, connectable: true, whenConnected: true });
    console.log("Stopped advertising new data.");
    if (advertiseTimeout) clearTimeout(advertiseTimeout);
    digitalWrite(LED3, 0); // Turn off blue LED
}

function advertiseNewDataFlag() {
    hasNewData = true;
    digitalWrite(LED3, 1); // Turn on blue LED while advertising new data flag
    // Include the new data flag in the advertising packet
    NRF.setAdvertising({ 0xFFFF: hasNewData ? [0x01] : [0x00] }, { name: 'Puck ' + puckId, showName: true, discoverable: true, connectable: true, whenConnected: true, interval: 500 });
    console.log("Started advertising with new data flag.");
    // Set a timeout to stop advertising if the phone doesn't connect
    if (advertiseTimeout) clearTimeout(advertiseTimeout);
    advertiseTimeout = setTimeout(() => {
        console.log("Phone did not connect. Stopping advertisement and finding other pucks.");
        stopAdvertisingNewData();
        findOtherPucks(); // Try to find other pucks after timeout
    }, 10000);
}
```

Εικόνα 15: Εκπομπή ύπαρξης νέων δεδομένων και κατάργηση αυτής

Όταν υπάρχουν νέα δεδομένα προς αποστολή, καλείται η advertiseNewDataFlag η οποία λειτουργεί ως εξής:

- Ενεργοποιεί τη μπλε λυχνία (LED3) όσο διαφημίζονται τα νέα δεδομένα.
- Χρησιμοποιεί τη NRF.setAdvertising του Espruino για να δημοσιεύσει την υπηρεσία 0xFFFF με δεδομένο τη σημαία νέων δεδομένων. Περιλαμβάνονται και κάποιες άλλες παράμετροι που αφορούν το δημόσιο όνομα της συσκευής, την ανακαλυψιμότητάς και την συνδεσιμότητά της.
- Καθορίζει ένα χρονικό όριο 10 δευτερολέπτων θέτοντας ένα timeout και καθαρίζει τυχόν προϋπάρχοντα timeouts για τον ίδιο σκοπό. Αν δεν γίνει σύνδεση με τηλέφωνο, σταματά την εκπομπή της 0xFFFF και καλεί τη συνάρτηση εύρεσης άλλων Puck.js.

Η κατάργηση της εκπομπής της 0xFFFF γίνεται με κλήση της stopAdvertisingNewDataFlag η οποία καλεί ξανά την NRF.setAdvertising με άδειες υπηρεσίες και τις ίδιες παραμέτρους ονόματος, ανακαλυψιμότητας και συνδεσιμότητας. Η stopAdvertisingNewDataFlag καλείται επίσης μία φορά κατά την εκκίνηση του κώδικα ώστε να θέσει το σωστό όνομα για το Puck.js (το οποίο αποτελείται από τη λέξη Puck και το αναγνωριστικό του) και να καταστήσει τη συσκευή ορατή και ανοιχτή προς σύνδεση.

4.2.4 Αποστολή δεδομένων στη συνοδευτική εφαρμογή

Η αποστολή των δεδομένων στο κινητό επιτυγχάνεται με χρήση του παρακάτω κώδικα:

```
// Event handler when the phone connects
NRF.on('connect', function (address) {
    logToStorage("Phone connected:", address);
    stopAdvertisingNewData(); // Stop advertising the new data flag
    sendArrayToPhone(); // Start sending the array when the phone connects
});


// Function to send array elements to another puck using Promises
function sendArrayToPhone() {
    let promiseChain = Promise.resolve(); // Start with a resolved promise

    for (let i = 0; i < array.length; i++) {
        const element = JSON.stringify(array[i]);
        const callback = logToStorage;
        const message = `Sending element ${i + 1}/${array.length}: ${element}`;
        const delayFun = delay;
        const nrf = NRF;
        promiseChain = promiseChain
            .then(() => {
                callback(message);
                return nrf.updateServices({
                    '20020001-53e4-11ee-8c99-0242ac120002': {
                        '20020002-53e4-11ee-8c99-0242ac120002': {
                            value: element, // Send chunk as notification
                            readable: true,
                            notify: true // Ensure notifications are enabled
                        }
                    }
                });
            })
            .then(() => delayFun(500)); // Delay 500ms between sending each element
    }
    return promiseChain
        .then(() => {
            return NRF.updateServices({
                '20020001-53e4-11ee-8c99-0242ac120002': {
                    '20020002-53e4-11ee-8c99-0242ac120002': {
                        value: "==END==", // Send chunk as notification
                        readable: true,
                        notify: true // Ensure notifications are enabled
                    }
                }
            });
        })
        .then(() => {
            logToStorage("All elements sent to phone");
        })
        .then(() => {
            clearArray();
        })
        .catch(err => {
            console.log("Error during sending elements:", err);
        });
}
```

Εικόνα 16: Κώδικας αποστολής δεδομένων στην εφαρμογή

Αρχικά, με κλήση της συνάρτησης NRF.on('connect', function (address) {...}) ορίζεται ένας ακροατής συμβάντων (event listener) που παρακολουθεί το συμβάν της σύνδεσης μιας συσκευής (όπως ένα κινητό) με το Puck.js μέσω Bluetooth. Κάθε φορά που συνδέεται μία συσκευή με το Puck.js, εκτελείται η συνάρτηση που ορίζεται μέσα σε αυτόν τον event listener, η λογική της οποίας είναι η ακόλουθη:

- Καλείται η logToStorage για να καταγραφεί το γεγονός της σύνδεσης στη μνήμη flash του Puck.js.
- Αυτή η συνάρτηση καλείται stopAdvertisingNewData για να σταματήσει η εκπομπή ύπαρξης νέων δεδομένων, η οποία δεν είναι πλέον απαραίτητη εφόσον υπήρξε σύνδεση.
- sendArrayToPhone();: Καλείται η συνάρτηση sendArrayToPhone η οποία αποστέλλει ολόκληρο τον πίνακα συμβάντων, στοιχείο-στοιχείο, στη συνοδευτική εφαρμογή.

Η συνάρτηση sendArrayToPhone λειτουργεί ως ακολούθως:

- Η συνάρτηση χρησιμοποιεί μία αλυσίδα υποσχέσεων (Promise chain), της οποίας ο πρώτος κρίκος – υπόσχεση είναι ένα λυμένο (resolved) promise.
- Κάθε στοιχείο του πίνακα μετατρέπεται σε συμβολοσειρά JSON με χρήση της JSON.stringify. Στη συνέχεια για το εκάστοτε στοιχείο προστίθονται δύο κρίκοι – promises στην αλυσίδα. Ο πρώτος γράφει κατάλληλο μήνυμα στο αρχείο καταγραφών και ενημερώνει την τιμή της χαρακτηριστικής 20020002-53e4-11ee-8c99-0242ac120002 με το JSON string του εκάστοτε στοιχείου ο δεύτερος κρίκος προσθέτει καθυστέρηση 500ms ώστε κάθε ενημέρωση να απέχει 500ms από την προηγούμενη. Ο σκοπός της καθυστέρησης είναι η αποφυγή κατακλυσμού του κινητού παραλήπτη με μηνύματα, ώστε η εφαρμογή να προλαβαίνει να λαμβάνει και να επεξεργάζεται σωστά τα δεδομένα.
- Μετά τη διάσχιση ολόκληρου του πίνακα προστίθενται κάποιοι τελικοί κρίκοι οι οποίοι αφορούν: την αποστολή της τερματικής συμβολοσειράς “==END==” προς σηματοδότησης του τέλους της μετάδοσης, την καταγραφή στο αρχείο κατάλληλου μηνύματος που επιβεβαιώνει την επιτυχή αποστολή ολόκληρου του πίνακα, το άδειασμα του πίνακα και το χειρισμό εξαιρέσεων.

Οι κρίκοι της αλυσίδας εκτελούνται σειριακά προκειμένου να επιτευχθεί η σειριακή αποστολή των στοιχείων με παρεμβολή καθυστέρησης. Κάτι τέτοιο στη JavaScript πρέπει να γίνει με ασύγχρονες κλήσεις. Το συντακτικό async/await θα απλοποιούσε την παραπάνω διαδικασία αλλά απουσιάζει από τον Espruino λόγω μεγάλης τεχνικής πολυπλοκότητας.

4.2.5 Εύρεση και αποστολή σε άλλα Puck.js

Η εύρεση άλλων Puck.js και η αποστολή δεδομένων σε αυτά γίνεται με την κλήση των παρακάτω συναρτήσεων:

```
function findOtherPucks() {
    if (retryCount >= maxRetries) {
        console.log("Max retry attempts reached. Stopping search for other pucks.");
        digitalWrite(LED3, 0); // Turn off blue LED after max retries
        return;
    }

    NRF.requestDevice({ filters: [{ namePrefix: 'Puck' }, { services: ['20020001-53e4-11ee-8c99-0242ac120002'] }] }) // Filter by namePrefix 'Puck'
        .then(function (device) {
            // Skip the last Puck that sent data
            Test Regex...
            if (device.id.replace(/:/g, '').split(' ')[0] === lastSenderPuckId) {
                throw "Skipping last Puck that sent data:" + lastSenderPuckId;
            }
            console.log("Found a puck:", device.id, "Name:", device.name);
            digitalWrite(LED2, 1); // Turn on green LED for found puck
            retryCount = 0; // Reset retry counter on successful connection
            // Connect and send the array to the other puck
            NRF.connect(device.id).then(connection => {
                console.log("Connected to puck:", device.id);
                digitalWrite(LED2, 1); // Turn on green LED for connection success

                connection.getPrimaryService('20020001-53e4-11ee-8c99-0242ac120002').then(service => {
                    return service.getCharacteristic('20020003-53e4-11ee-8c99-0242ac120002');
                }).then(characteristic => {
                    return sendArrayElementsToPuck(characteristic);
                }).then(() => {
                    console.log("Array sent to puck:", device.id);
                    digitalWrite(LED2, 1); // Flash green LED for data sent success
                    setTimeout(() => digitalWrite(LED2, 0), 500); // Turn off green LED after 500ms
                    connection.disconnect();
                    clearArray(); // Clear the array after sending
                });
            }).catch(err => {
                console.log("Failed to connect to puck:", err);
                digitalWrite(LED3, 0); // Turn off blue LED after search is complete
                digitalWrite(LED1, 1); // Turn on red LED for connection failure
                setTimeout(() => digitalWrite(LED1, 0), 1000); // Turn off red LED after 1 second
                retryCount++;
                if (retryCount < maxRetries) {
                    console.log("Retrying... Attempt:", retryCount);
                    setTimeout(findOtherPucks, 120000); // Retry after 2 minutes
                }
            });
        })
        .catch(function (err) {
            console.log("No suitable pucks found or failed to connect:", err);
            digitalWrite(LED3, 0); // Turn off blue LED after search is complete
            digitalWrite(LED1, 1); // Turn on red LED for no puck found
            setTimeout(() => digitalWrite(LED1, 0), 1000); // Turn off red LED after 1 second
            retryCount++;
            if (retryCount < maxRetries) {
                console.log("Retrying... Attempt:", retryCount);
                setTimeout(findOtherPucks, 120000); // Retry after 2 minutes
            }
        });
}
```

Εικόνα 17: Συνάρτηση εντοπισμού άλλων Puck.js

```
// Function to send array elements to another puck using Promises
function sendArrayElementsToPuck(characteristic) {
    let promiseChain = Promise.resolve(); // Start with a resolved promise

    for (let i = 0; i < array.length; i++) {
        const element = JSON.stringify(array[i]);
        const callback = console.log;
        const message = `Sending element ${i + 1}/${array.length}: ${element}`;
        const characteristicBind = characteristic;
        const delayFun = delay;
        promiseChain = promiseChain
            .then(() => {
                callback(message);
                return characteristicBind.writeValue(element);
            })
            .then(() => delayFun(500)); // Delay 500ms between sending each element
    }

    return promiseChain
        .then(() => {
            return characteristic.writeValue("==END==");
        })
        .then(() => {
            console.log("All elements sent to puck.");
        })
        .catch(err => {
            console.log("Error during sending elements:", err);
        });
}
```

Εικόνα 18: Συνάρτηση αποστολής δεδομένων σε άλλα Puck.js

Η αποστολή των δεδομένων σε άλλα Puck.js γίνεται με τρόπο πανομοιότυπο με την αποστολών των δεδομένων στην συνοδευτική εφαρμογή, που αναλύθηκε στην προηγούμενη ενότητα. Για το λόγο αυτό, παραλείπεται.

Όσον αφορά την εύρεση άλλων κατάλληλων προς αποστολή Puck.js, καλείται η findOtherPucks η οποία λειτουργεί ως εξής:

- Αρχικά ελέγχεται ότι δεν έχει ξεπεραστεί το όριο επαναπροσπαθειών. Αν αυτό έχει ξεπεραστεί, τότε η συνάρτηση απλώς σβήνει τη μπλε λυχνία και επιστρέφει χωρίς επιπλέον παρενέργειες. Αντίθετα, αν οι επαναπροσπάθειες δεν έχουν ξεπεραστεί, τότε καλείται η συνάρτηση NRF.requestDevice φιλτράροντας τις συσκευές με βάση την ύπαρξη της υπηρεσίας 20020001-53e4-11ee-8c99-0242ac120002 και της λέξης «Puck» στο δημόσιο όνομά τους, ώστε να βρει μόνο τις κατάλληλες συσκευές.
- Αν δεν βρεθεί κατάλληλη συσκευή ή η σύνδεση στην ευρεθείσα συσκευή αποτύχει, τυπώνεται κατάλληλο μήνυμα σφάλματος, σβήνει η μπλε λυχνία και ανάβει η κόκκινη για διάστημα ενός δευτερολέπτου, και ο μετρητής επαναπροσπαθειών αυξάνεται κατά 1. Εν συνεχεία, αν οι επαναπροσπάθειες είναι λιγότερες από το όριο, τίθεται ένα χρονικό timeout το οποίο εκτελεί ξανά την findOtherPucks μετά το πέρας 2 λεπτών.
- Αντίθετα αν η ευρεθεί συσκευή και η σύνδεση σε αυτήν είναι επιτυχής, ζητείται η υπηρεσία 20020001-53e4-11ee-8c99-0242ac120002 και η

χαρακτηριστική της 20020002-53e4-11ee-8c99-0242ac120002, η οποία δίνετια ως όρισμα στην κλήση της sendArrayElemetsToPuck.

- Μετά την επτυχή ολοκλήρωση της sendArrayElemetsToPuck, δηλαδή της αποστολής ολόκληρου του πίνακα, ανάβει το πράσινο LED για 50ms, η σύνδεση τερματίζεται και ο πίνακας συμβάντων του Puck.js – αποστολέας αδειάζει.

4.2.6 Παρακολούθηση πατήματος κουμπιού

Η παρακολούθηση και διαχείριση του πατήματος κουμπιού γίνεται με την παρακάτω κλήση της συνάρτησης setWatch του Espruino:

```
// Button press event to simulate fall or panic event
setWatch(function () {
    console.log("Button pressed, adding panic event.");
    digitalWrite(LED1, 1); // Turn on red LED when button is pressed
    setTimeout(() => {
        digitalWrite(LED1, 0);
        addEvent(false, true); // Example: panic event
        advertiseNewDataFlag(); //
    }, 200); // Turn off blue LED after 500ms
}, BTN, { edge: 'rising', debounce: 50, repeat: true });
```

Εικόνα 19: Παρακολούθηση και διαχείριση πατήματος του κουμπιού στο Puck.js

Μέσω αυτής ορίζεται ένας event listener που ανιχνεύει συνεχώς (ορίζοντας ως true την σημαία repeat) το πάτημα του κουμπιού (BTN) του Puck.js. Κάθε φορά που πατιέται το κουμπί, κατά την άνοδο του σήματος (rising edge – ανερχόμενη ακμή, δηλαδή την ώρα του πατήματος) εκτελείται η συνάρτηση που ορίζεται μέσα σε αυτόν τον event listener, η λογική της οποίας είναι η ακόλουθη:

- Τυπώνεται κατάλληλο μήνυμα στην κονσόλα
- Ανάβει η κόκκινη ενδεικτική λυχνία προς οπτική επιβεβαίωση από τον χρήστη
- Μετά από 200ms σβήνει το κόκκινο led, γίνεται κλήση της addEvent(false, true) για προσθήκη ενός γεγονός "πανικού" στον πίνακα συμβάντων, όπου το πρώτο όρισμα false σημαίνει ότι δεν συνέβη πτώση (fell) και το δεύτερο όρισμα true σημαίνει ότι συνέβη πανικός (panicked), όπως επεξηγήθηκε στην υποενότητα 4.2.1. Τέλος καλείται η συνάρτηση που εκκινεί την εκπομπή ύπαρξης νέων δεδομένων, όπως αυτή αναλύθηκε στην υποενότητα 4.2.2.

Τέλος, σημειώνεται ότι ορίζεται 50ms debounce, το οποίο είναι μια τεχνική που χρησιμοποιείται για να αποτρέψει την ανίχνευση πολλαπλών πατημάτων σε πολύ μικρό χρονικό διάστημα. Με την τιμή 50ms, αποφεύγονται διπλά πατήματα που μπορεί να ανιχνευτούν εξαιτίας "θορύβου".

4.2.7 Παρακολούθηση συμβάντων πτώσης

Η παρακολούθηση και ανίχνευση των συμβάντων πτώσης επιτυγχάνεται με τον παρακάτω κώδικα:

```
require("puckjsv2-accel-movement").on({
    duration: 4,
    threshold: 35,
    lowPower: false,
});

let idleTimeout;
Puck.on('accel', function (a) {
    LED.set();
    if (idleTimeout) clearTimeout(idleTimeout);
    else {
        idleTimeout = setTimeout(function () {
            idleTimeout = undefined;
            LED.reset();
        }, 100);
        if (n.call([a.acc.x, a.acc.y, a.acc.z]) > threshold) {
            addEvent(true, false); // Example: panic event
            advertiseNewDataFlag(); //
        }
    }
});
```

Εικόνα 20: Ανίχνευση κίνησης και εντοπισμός πτώσης

Όπως φαίνεται στην παραπάνω εικόνα, γίνεται πρώτα η αρχικοποίηση της παρακολούθησης κινήσεων φορτώνοντας τη βιβλιοθήκη για το επιτυαχυνσιόμετρο του Puck.js. Η αρχικοποίηση στην προκειμένη περίπτωση λαμβάνει τρεις παραμέτρους, οι οποίες ρυθμίστηκαν ύστερα από δοκιμές ώστε να αποφευχθεί η υπερβολική ευαισθησία στις κινήσεις, αλλά να εντοπίζεται μία κρίσιμη κίνηση - υποψήφια πτώση:

- duration: Καθορίζει ότι η κίνηση πρέπει να διαρκέσει για 4 μονάδες χρόνου πριν ενεργοποιηθεί ένα γεγονός. Αυτό σημαίνει ότι δεν αρκεί μια στιγμιαία κίνηση· πρέπει να υπάρχει παρατεταμένη κίνηση. Οι μονάδες χρόνου δεν προσδιορίζονται στον ορισμό της βιβλιοθήκης.
- threshold: Ορίζει ένα όριο που σχετίζεται με το πόσο έντονη πρέπει να είναι η κίνηση (δηλαδή πόσο υψηλή η τιμή της επιτάχυνσης) για να θεωρηθεί σημαντική και άρα να μην αγνοηθεί. Αν η επιτάχυνση (τιμές x, y, z) υπερβεί αυτό το όριο, θα ενεργοποιηθεί το γεγονός.
- lowPower: Καθορίζει αν ο αισθητήρας θα λειτουργήσει σε λειτουργία χαμηλής κατανάλωσης ενέργειας. Η σημαία τέθηκε ως false, που σημαίνει ότι θα χρησιμοποιηθεί η κανονική δειγματοληψία στα 12.5Hz. Αν τεθεί ως true, η δειγματοληψία θα γινόταν στα 0.6Hz (για χαμηλότερη κατανάλωση ενέργειας), αλλά το threshold θα έπρεπε να είναι μεγαλύτερο για αξιόπιστη λειτουργία.

Μετά την αρχικοποίηση γίνεται κλήση της συνάρτησης Puck.on(accel, function (a) {...}), μέσω της οποίας ορίζεται ένας ακροατής συμβάντων (event listener) που

παρακολουθεί το συμβάν της αλλαγής στην κίνηση (στην επιτάχυνση). Κάθε φορά που ανιχνεύεται επιτάχυνση η οποία είναι εντός των ορίων που τέθηκαν στην αρχικοποίηση (οι συντομότερες ή λιγότερο έντος κινήσεις αγνοούνται), εκτελείται η συνάρτηση που ορίζεται μέσα σε αυτόν τον event listener με όρισμα το αντικείμενο a, που φέρει τις τιμές της επιτάχυνσης που μετρήθηκαν στους 3 άξοενς x, y, z. Η λογική της συνάρτησης είναι η ακόλουθη:

- Ενεργοποιείται το κόκκινο led για 100ms προς οπτική επιβεβαίωση ανίχνευσης μίας έντονης κίνησης.
- Γίνεται κλήση του Νευρωνικού Δικτύου n, με ορίσματα τις τιμές επιταχύνσεων στους τρεις άξονες. Αν η τιμή εξόδου του νευρωνικού ξεπερνά το κατώφλι που έχει καθοριστεί ύστερα από την εκπαίδευση καιτ ην αξιολόγηση, τότε η κίνηση θεωρείται πτώση, οπότε προστίθεται συμβάν πτώσης στον πίνακα συμβάντων και καλείται η συνάρτησης εκπομπής νέων δεδομένων. Αλλιώς η συνάρτηση επιστρέφει χωρίς επιπλέον παρενέργειες. Ο κώδικας του Νευρωνικού Δικτύου και ο καθορισμός του κατωφλίου θα παρουσιαστούν στην αντίστοιχη υποενότητα.

4.3 Διεπαφή χρήστη και κώδικας συνοδευτικής Εφαρμογής

4.3.1 Αρχική ρύθμιση, σημείο εκκίνησης και ορισμός αδειών της εφαρμογής

Αρχικά παρατίθεται το παρακάτω κομμάτι κώδικα, το οποίο αποτελεί το σημείο εκκίνησης και την αρχική ρύθμιση της εφαρμογής Flutter για τη διαχείριση συνδέσεων Bluetooth Low Energy (BLE) μέσω της βιβλιοθήκης flutter_blue_plus, την παρακολούθηση της κατάστασης του Bluetooth και την επικοινωνία με την Firebase για την αυθεντικοποίηση χρηστών.

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();

  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  [
    Permission.location,
    Permission.storage,
    Permission.bluetooth,
    Permission.bluetoothConnect,
    Permission.bluetoothScan
  ].request().then((status) {
    runApp(const ProviderScope(child: FlutterBlueApp()));
  });
}
```

Εικόνα 21: Αρχική ρύθμιση και σημείο εκκίνησης εφαρμογής Flutter

Συγκεκριμένα, διακρίνεται η συνάρτηση main, η οποία είναι το σημείο εκκίνησης της εφαρμογής. Χρησιμοποιείται για την αρχικοποίηση των απαραίτητων ρυθμίσεων πριν την εκκίνηση του Flutter. Με κλήση της Flutter.initializeApp αρχικοποιείται η Firebase χρησιμοποιώντας τις προκαθορισμένες επιλογές της πλατφόρμας (iOS ή Android) και στη συνέχεια η εφαρμογή εξασφαλίζει ότι έχει τις απαραίτητες άδειες για τη χρήση του Bluetooth και της τοποθεσίας, ζητώντας τες αν χρειάζεται, πριν ξεκινήσει. Οι άδειες αυτές απαιτούνται για την αναζήτηση και σύνδεση συσκευών μέσω Bluetooth, που αποτελεί την κύρια λειτουργία της εφαρμογής (η άδεια τοποθεσίας είναι απαραίτητη μόνο στο λειτουργικό Android καθώς το Android θεωρεί ότι η σάρωση για BLE μπορεί να οδηγήσει στην αναγνώριση της τοποθεσίας του χρήστη μέσω γειτονικών συσκευών). Οι άδειες στις εφαρμογές είναι σημαντικές διότι επιτρέπουν στον προγραμματιστή να αποκτήσει πρόσβαση σε ευαίσθητα δεδομένα ή λειτουργίες της συσκευής, όπως το Bluetooth, η τοποθεσία ή τα αρχεία. Για να μπορέσει ένας προγραμματιστής να έχει πρόσβαση σε δεδομένα πέρα από τη δική του εφαρμογή, είναι υποχρεωμένος να ζητά την άδεια του χρήστη και να εξηγεί με σαφήνεια πώς και γιατί θα χρησιμοποιήσει αυτά τα δεδομένα. Αυτό το σύστημα υπάρχει για να προστατεύσει την ιδιωτικότητα του χρήστη και να περιορίσει την ανεξέλεγκτη χρήση των προσωπικών του δεδομένων χωρίς τη ρητή συγκατάθεσή του. Οι άδειες διασφαλίζουν ότι ο χρήστης είναι πλήρως ενήμερος για τους κινδύνους που μπορεί να προκύψουν από την παροχή πρόσβασης σε δεδομένα ή λειτουργίες, και παρέχουν στον χρήστη τον έλεγχο πάνω στις πληροφορίες που κοινοποιεί σε μια εφαρμογή. Στο Android, οι άδειες ζητούνται στο AndroidManifest.xml, ενώ στο iOS στο Info.plist, όπως φαίνεται στις εικόνες που αποτελούν τμήμα των αρχείων αυτών.

```
<key>NSBluetoothAlwaysUsageDescription</key>
<string>Need BLE permission</string>
<key>NSBluetoothPeripheralUsageDescription</key>
<string>Need BLE permission</string>
<key>NSLocationAlwaysAndWhenInUseUsageDescription</key>
<string>Need Location permission</string>
<key>NSLocationAlwaysUsageDescription</key>
<string>Need Location permission</string>
<key>NSLocationWhenInUseUsageDescription</key>
<string>Need Location permission</string>
```

Εικόνα 22: Τμήμα αρχείου Info.plist για καθορισμό αδειών στο iOS

```
<uses-permission android:name="android.permission.BLUETOOTH_SCAN" android:usesPermissionFlags="neverForLocation" />
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />

<!-- legacy for Android 11 or lower -->
<uses-permission android:name="android.permission.BLUETOOTH" android:maxSdkVersion="30" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" android:maxSdkVersion="30" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" android:maxSdkVersion="30"/>

<!-- legacy for Android 9 or lower -->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" android:maxSdkVersion="28" />
```

Εικόνα 23: Τμήμα αρχείου AndroidManifest.xml για ακθορισμό αδειών στο Android

Στη συνέχεια εκκινεί η κύρια εφαρμογή με χρήση της runApp, η οποία παίρνει ως όρισμα τη ρίζα του δέντρου των Widgets, δηλαδή το Widget που είναι γονέας όλων των άλλων, το οποίο είναι το παρακάτω:

```
class FlutterBlueApp extends StatelessWidget {
  const FlutterBlueApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      color: Colors.lightBlue,
      home: const AuthGate(),
    ); // MaterialApp
  }
}
```

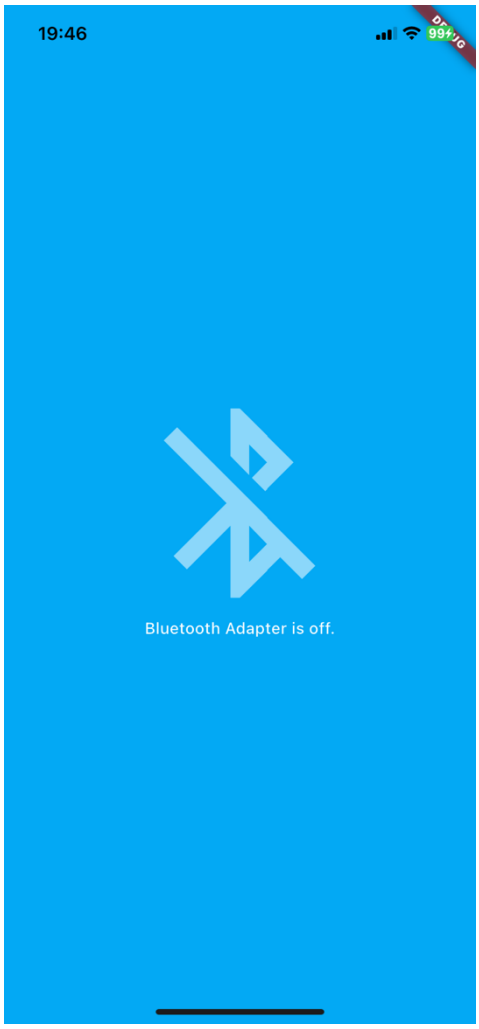
Εικόνα 24: Η ρίζα του Widget tree της εφαρμογής

Το FlutterBlueApp χρησιμοποιεί το MaterialApp. Το MaterialApp είναι ένα βασικό widget στο Flutter που παρέχει τη δομή για την εφαρμογή και εφαρμόζει τις αρχές σχεδιασμού Material Design, το οποίο είναι το επίσημο στυλ και το γραφικό περιβάλλον που χρησιμοποιεί η Google. Το Material Design παρέχει ένα σύνολο από οδηγίες για τη δημιουργία διεπαφών χρήστη (UI), διασφαλίζοντας συνέπεια και ευχρηστία στις εφαρμογές. Ως αρχική οθόνη ορίζεται να είναι το AuthGate, το οποίο παρακολουθεί την κατάσταση αυθεντικοποίησης του χρήστη και την κατάσταση του Bluetooth στη συσκευή, όπως φαίνεται παρακάτω:

```
class AuthGate extends StatelessWidget {
  const AuthGate({super.key});

  @override
  Widget build(BuildContext context) {
    return StreamBuilder<User?>(
      stream: FirebaseAuth.instance.authStateChanges(),
      builder: (context, snapshot) {
        if (!snapshot.hasData) {
          return const AuthRegScreen();
        }

        return StreamBuilder<BluetoothAdapterState>(
            stream: FlutterBluePlus.adapterState,
            initialData: BluetoothAdapterState.unknown,
            builder: (c, snapshot) {
              final adapterState = snapshot.data;
              if (adapterState == BluetoothAdapterState.on) {
                return HomeScreen();
              } else {
                FlutterBluePlus.stopScan();
                return BluetoothOffScreen(adapterState: adapterState);
              }
            }); // StreamBuilder
      },
    ); // StreamBuilder
  }
}
```

Εικόνα 25: Πύλη αυθεντικοποίησης της εφαρμογής

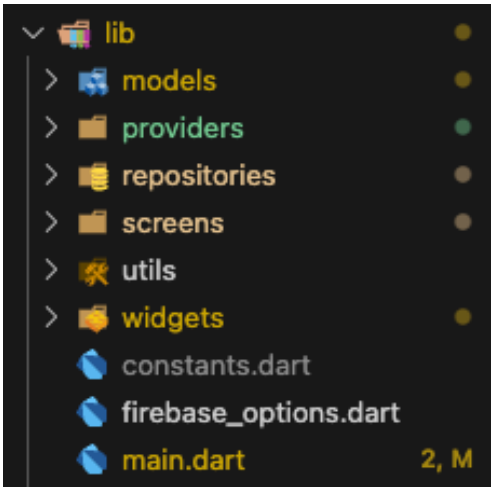
Αν ο χρήστης δεν είναι συνδεδεμένος, εμφανίζεται η οθόνη σύνεδης/εγγραφής, αλλιώς παρακολουθείται η κατάσταση του Bluetooth. Αν η κεραία είναι ενεργοποιημένη, εμφανίζεται η αρχική οθόνη Home, αλλιώς εμφανίζεται η οθόνη που φαίνεται στα δεξιά.



Εικόνα 26: Οθόνη απενργοποιημένου Bluetooth

4.3.2 Αρχιτεκτονική κώδικα και αρχείων

Η αρχιτεκτονική με την οποία είναι δομημένος ο κώδικας και τα αρχεία βασίζεται σε μία modular προσέγγιση που υλοποιεί το Μοτίβο Αποθήκης (Repository Pattern) με σκοπό τη διαχείριση δεδομένων και τη διασύνδεση της εφαρμογής με εξωτερικές πηγές δεδομένων. Κάθε φάκελος έχει συγκεκριμένο ρόλο στην εφαρμογή και συμβάλλει στην απομόνωση της διαχείρισης δεδομένων (data management), της επιχειρησιακής λογικής (business logic) και των οπτικών στοιχείων - προβολών (views), εισάγοντας κατάλληλα στρώματα αφαίρεσης (abstraction layer) μεταξύ τους. Αυτή η προσέγγιση εξασφαλίζει μια καθαρή, διαχωρισμένη και επεκτάσιμη αρχιτεκτονική. Η δομή των φακέλων παρουσιάζεται στην εικόνα παρακάτω:



Εικόνα 27: Δομή κώδικα εφαρμογής

Συγκεκριμένα διακρίνονται οι εξής φάκελοι:

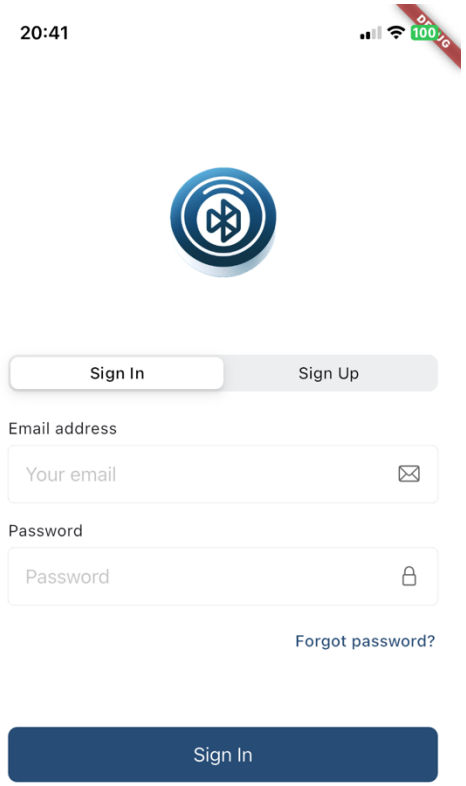
- lib: Ο κεντρικός φάκελος της εφαρμογής Flutter που περιέχει όλα τα αρχεία και τους πόρους της εφαρμογής.
- models: Περιέχει τα μοντέλα δεδομένων της εφαρμογής. Αυτά τα μοντέλα ορίζουν τη δομή των δεδομένων που χειρίζονται τα repositories. Αυτά τα δεδομένα χρησιμοποιούνται για να ανταλλάσσονται πληροφορίες μεταξύ του frontend και του backend.
- providers: Περιέχει τους providers, οι οποίοι είναι υπεύθυνοι για τη διαχείριση της κατάστασης (state) της εφαρμογής. Οι providers εδώ είναι γραμμένοι με χρήση της βιβλιοθήκης Riverpod, συνδέουν τα repositories με την business logic της εφαρμογής και παρέχουν δεδομένα στα widgets. Οι providers αλληλεπιδρούν με τα repositories για να φέρουν δεδομένα, όπως πληροφορίες χρηστών ή συσκευών Puck.js.
- repositories: Περιέχει τα repositories, τα οποία είναι υπεύθυνα για την πρόσβαση στα δεδομένα. Τα repositories απομονώνουν τις πηγές δεδομένων, που εδώ είναι η Firebase και το backend μας (REST API και βάση δεδομένων) από την υπόλοιπη εφαρμογή. Τα repositories εκτελούν λειτουργίες όπως λήψη δεδομένων από ένα REST API, αποστολή δεδομένων στο backend ή ανάγνωση δεδομένων από τοπική πηγή αποθήκευσης. Παρέχουν μια ενιαία διεπαφή για την πρόσβαση και τη διαχείριση των δεδομένων, ανεξάρτητα από την πηγή τους.
- screens: Περιέχει τις διαφορετικές οθόνες της εφαρμογής (views). Κάθε οθόνη αντιπροσωπεύει μια διαφορετική λειτουργικότητα της εφαρμογής, όπως η κεντρική σελίδα, η σελίδα συνδεδεμένων Puck.js ή η σελίδα σύνδεσης. Οι οθόνες αλληλεπιδρούν με τους providers για να λάβουν δεδομένα και να ενημερώσουν το UI με βάση την κατάσταση της εφαρμογής.
- utils: Περιέχει βοηθητικές λειτουργίες και εργαλεία (utilities) που χρησιμοποιούνται σε όλη την εφαρμογή. Αυτά μπορεί να περιλαμβάνουν κοινές συναρτήσεις για validation, formatters, ή σταθερές τιμές. Αυτές οι βοηθητικές λειτουργίες συμβάλλουν στη μείωση της επανάληψης κώδικα και κάνουν τη συντήρηση ευκολότερη.

- widgets: Περιέχει τα μεμονωμένα widgets που χρησιμοποιούνται επαναλαμβανόμενα σε διάφορες οθόνες της εφαρμογής. Αυτά τα widgets είναι επαναχρησιμοποιήσιμα UI components, όπως κουμπιά, κάρτες ή φόρμες. Τα widgets αυτά χρησιμοποιούν δεδομένα από τους providers για να εμφανίσουν δυναμικά στοιχεία στην οθόνη, ανάλογα με την κατάσταση της εφαρμογής.

4.3.3 Σύνδεση και εγγραφή χρήστη

4.3.3.1 Οθόνη σύνδεσης

Στην εικόνα δεξιά παρουσιάζεται η οθόνη σύνδεσης (sign in) που χρησιμοποιείται για αυθεντικοποίηση ενός υπάρχοντα χρήστη. Πρόκειται για μία φόρμα με δύο πεδία κειμένου, όπου το ένα προορίζεται για την δεύθυνση ηλεκτρονικού ταχυδρομείου του χρήστη και το άλλο για τον κωδικό του:



Εικόνα 28: Οθόνη σύνδεσης (sign in)

Παρακάτω παρουσιάζεται ο κώδικας που υλοποιεί την οθόνη αυτή. Όπως φαίνεται και στην εικόνα, αρχικά ορίζονται δύο TextEditingControllers, ένας για κάθε πεδίο κειμένου. Οι χειριστές του Flutter χρησιμοποιούνται για τη διαχείριση και την παρακολούθηση του κειμένου που εισάγεται σε πεδία κειμένου (TextFields), επιτρέποντας την πρόσβαση, τροποποίηση και έλεγχο του κειμένου που

πληκτρολογεί ο χρήστης σε πραγματικό χρόνο.. Τα πεδία κειμένου δημιουργούνται με χρήση δύο reusable Widgets που έχουμε ορίσει στον φάκελο widgets, το PuckTextfield και το PasswordInput, των οποίων ο κώδικας παραλείπεται. Το κουμπί ορίζεται ως το παιδί ενός FutureBuilder. Ο FutureBuilder είναι ένα widget στο Flutter που παρακολουθεί ένα Future, δηλαδή μία ασύγχρονη πράξη, και ενημερώνει το UI ανάλογα με την κατάσταση του Future (π.χ. αν είναι σε εξέλιξη, αν έχει ολοκληρωθεί ή αν έχει αποτύχει). Χρησιμοποιείται για την εμφάνιση δυναμικού περιεχομένου, όπως φόρτωμα δεδομένων από το δίκτυο ή κάποια μακροχρόνια διεργασία. Στην προκειμένη περίπτωση το Future είναι η λειτουργία αυθεντικοποίησης, δηλαδή η κλήση της συνάρτησης signIn του AuthRepository με ορίσματα τα στοιχεία που γράφει ο χρήστης στις φόρμες. Ο FutureBuilder χρησιμοποιείται για να εμφανίσει έναν κυκλικό δείκτη φόρτωσης κατά την αυθεντικοποίηση, το κπυμπί σε κάθε άλλη κατάσταση, ενώ εμφανίζει επίσης και τα πιθανά μηνύματα σφάλματος μέσω του CustomToast widget, του οποίου ο κώδικας παραλείπεται.

```dart
class SigninForm extends StatefulWidget {
  const SigninForm({super.key});

  @override
  State<SigninForm> createState() => _SigninFormState();
}

You, 3 weeks ago | 1 author (You)
class _SigninFormState extends State<SigninForm> {
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();
  final AuthRepository userRepo = AuthRepository();

  Future<void>? _signInFuture;
  late FToast fToast;

  @override
  void initState() {
    super.initState();
    fToast = FToast();
    // if you want to use context from globally instead of content we need to pass
    fToast.init(context);
  }

  void signIn() => setState(() {
        _signInFuture = userRepo.signIn(
            _emailController.text.trim(), _passwordController.text.trim());
      });

  @override
  Widget build(BuildContext context) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.center,
      crossAxisAlignment: CrossAxisAlignment.center,
      children: <Widget>[
        PuckTextfield(
            controller: _emailController,
            placeholder: "Your email",
            label: "Email address",
            suffix: const Icon(
              CupertinoIcons.envelope,
              size: 20,
              color: Color.fromRGBO(0, 0, 0, 0.6),
            )), // Icon // PuckTextfield
        PasswordInput(controller: _passwordController),
        Padding(
          padding: const EdgeInsets.only(right: 10, bottom: 41),
          child: Row(
            mainAxisAlignment: MainAxisAlignment.end,
            children: [
              TextButton(
                onPressed: () => Navigator.of(context).push(MaterialPageRoute(
                    builder: (context) => const ForgotPasswordScreen())), // Materi
                child: const Text(
                  "Forgot password?",
                  style: TextStyle(color: Color.fromRGBO(39, 76, 117, 1)),
                ), // Text
              ) // TextButton
            ],
          ), // Row
        ), // Padding
        FutureBuilder<void>(
            future: _signInFuture,
            builder: (BuildContext context, AsyncSnapshot<void> snapshot) {
              if (snapshot.connectionState == ConnectionState.waiting) {
        FutureBuilder<void>(
            future: _signInFuture,
            builder: (BuildContext context, AsyncSnapshot<void> snapshot) {
              if (snapshot.connectionState == ConnectionState.waiting) {
                return const CircularProgressIndicator(); // Loading indicator
              } else {
                if (snapshot.connectionState == ConnectionState.done) {
                  WidgetsBinding.instance.addPostFrameCallback((_) {
                    snapshot.hasError
                        ? CustomToast(fToast: fToast, context: context)
                            .showToast("error", snapshot.error.toString())
                        : null;
                    setState(() {
                      _signInFuture = null;
                    });
                  });
                }
                return PuckFilledButton(
                  onPressed: signIn,
                  text: 'Sign In',
                );
              }
            }),
        const SizedBox(
          height: 17,
        ),
      ],
    );
  }
}
```

Εικόνα 29: Κώδικας υλοποίησης της οθόνης Sign In

Η συνάρτηση signIn της κλάσης AuthRepository είναι η παρακάτω:

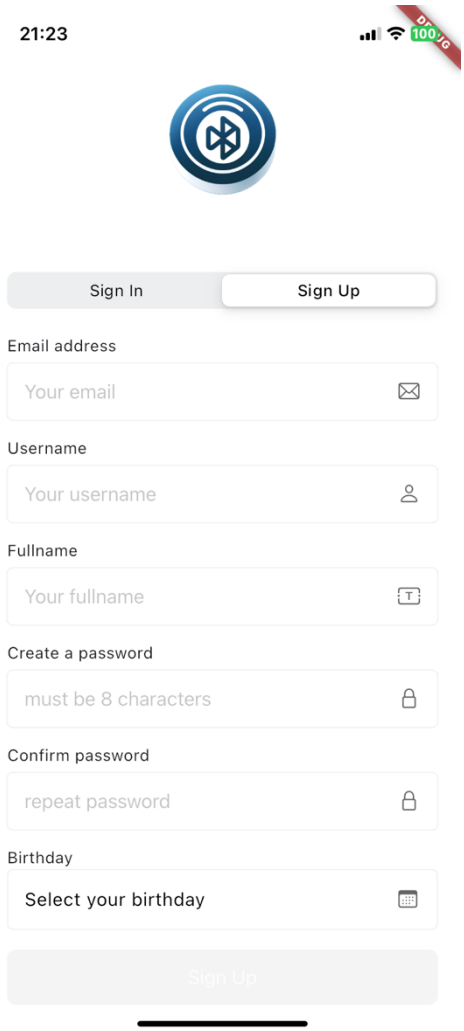
```dart
Future<void> signIn(String email, String password) async {
  // Call your backend's login endpoint
  final response = await http.post(
    Uri.parse('http://192.168.0.11:3000/auth/login'),
    body: {'email': email, 'password': password},
  );

  if (response.statusCode == 200) {
    final customToken = jsonDecode(response.body)['token'];

    // Sign in with the custom token using Firebase
    await FirebaseAuth.instance.signInWithCustomToken(customToken);

    print(await getIdToken());
  } else if (response.statusCode == 401) {
    throw "Incorrect password";
  } else if (response.statusCode == 404) {
    throw "User does not exist";
  } else {
    print('Authentication failed: ${response.statusCode}');

    throw "Server error";
  }
}
```

Εικόνα 30: Συνάρτηση signIn του AuthRepository

Η συνάρτηση signIn στέλνει ένα αίτημα μεθόδου POST στο backend για να κάνει είσοδο με email και κωδικό πρόσβασης. Αν η αυθεντικοποίηση είναι επιτυχής, (κωδικός κατάστασης 200), το σώμα της απάντησης (response body) περιέχει ένα custom token, το οποίο η συνάρτηση χρησιμοποιεί για να συνδεθεί στη Firebase με χρήση της μεθόδου signInWithCustomToken που παρέχει το SDK, και λαμβάνει ένα ID token το οποίο πλέον αθ χρησιμοποιεί για ταυτοποίηση. Αν υπάρξει σφάλμα, όπως λανθασμένος κωδικός ή ανύπαρκτος χρήστης (κωδικοί κατάστασης 401 και 404 αντίστοιχα), παράγει εξαιρέσεις με κατάλληλα μηνύματα σφάλματος. Περισσότερα για τους κωδικούς κατάστασης HTTP θα εξηγηθούν στην επόμενη ενότητα η οποία αφορά το backend.

4.3.3.2 Οθόνη εγγραφής

Στην εικόνα δεξιά παρουσιάζεται η οθόνη εγγραφής (sign up) που χρησιμοποιείται για εγγραφή ενός νέου χρήστη στο σύστημα. Πρόκειται για μία φόρμα με πέντε πεδία κειμένου και έναν επιλογέα ημερομηνίας. Τα πεδία αφορούν την δεύθυνση ηλεκτρονικού ταχυδρομείου του χρήστη, το username με το οποίο θα εμφανίζεται στην εφαρμογή, το πραγματικό πλήρες όνομά του και τον κωδικό του, ενώ το δεύτερο πεδίο κωδικού αφορά την επαλήθευση του κωδικού. Ο επιλογέας ημερομηνίας αφορά την ημερομηνία γενεθλίων του χρήστη προς προσδιορισμό της ηλικίας του:



Εικόνα 31: Οθόνη εγγραφής (sign up)

Ο κώδικας υλοποίησης της οθόνης παραλείπεται ως παρόμοιος με την οθόνη sign in. Η συνάρτηση signUp της κλάσης AuthRepository που πραγματοποιεί την επικοινωνία με το backend προς εγγραφή του χρήστη είναι η παρακάτω:

```dart
Future<void> signUp(RegisterFormState registerFormContents) async {
  // Call your backend's login endpoint
  final response = await http.post(
    Uri.parse('http://192.168.0.11:3000/auth/register'),
    body: registerFormContents.toJson(),
  );

  if (response.statusCode == 201) {
    final customToken = jsonDecode(response.body)['token'];

    // Sign in with the custom token using Firebase
    await FirebaseAuth.instance.signInWithCustomToken(customToken);

    print(await getIdToken());
  } else if (response.statusCode == 401) {
    throw "Incorrect password";
  } else if (response.statusCode == 404) {
    throw "User does not exist";
  } else if (response.statusCode == 409) {
    throw "Username or email is taken";
  } else {
    print('Authentication failed: ${response.statusCode}');

    throw "Server error";
  }
}
```

Εικόνα 32: Συνάρτηση signUp του AuthRepository

Ο κώδικάς της είναι πανομοιότυπος με το sign in με δύο διαφορές: η κατάσταση επιτυχίας τώρα χαρακτηρίζεται με τον κωδικό 201 αντί για 200, για λόγους που θα εξηγηθούν στην ενότητα που αφορά το backend, ενώ υπάρχει επίσης μία ακόμα περίπτωση σφάλματος, η προσπάθεια εγγραφής ενός χρήστη ο οποίος είναι ήδη εγεγραμμενος, και η οποία χαρακτηρίζεται από τον κωδικό κατάστασης 409.

4.3.4 Εύρεση και διαχείριση συσκευών Puck.js

Παρακάτω παρουσιάζονται οι δύο οθόνες που σχετίζονται με την εύρεση και διαχείριση συσκευών Puck.js. Η οθόνη εύρεσης συσκευών (Find Devices – αριστερή εικόνα), αφορά την εύρεση των κοντινών Puck.js και την αυτόματη προσθήκη τους στη βάση δεδομένων κατά την σύνδεση. Η οθόνη My Pucks (δεξιά εικόνα), αφορά την προβολή όσων Puck.js έχει αποθηκεύση ο χρήστης, και την ικανότητα αφαίρεσης αυτών:

Εικόνα 34: Οθόνη εύρεσης συσκευών

Εικόνα 33: Οθόνη προβολής και διαχείρισης των συσκευών μου

Ο κώδικας που υλοποιεί την οθόνη εύρεσης συσκευών παρουσιάζεται παρακάτω. Αυτός ο κώδικας είναι υπεύθυνος για την εύρεση και διαχείριση συσκευών Bluetooth μέσω του Bluetooth Low Energy (BLE). Ο χρήστης μπορεί να σαρώνει για συσκευές Puck.js, να συνδέεται ή να αποσυνδέεται από αυτές, και να προσθέτει συσκευές στη λίστα του αυτόματα όταν συνδεθεί. Συγκεκριμένα:

- Όταν ο χρήστης πατήσει το κουμπί "CONNECT" (Σύνδεση), η συσκευή συνδέεται μέσω της μεθόδου _handleDeviceConnection, και αν η σύνδεση είναι επιτυχής, η συσκευή προστίθεται αυτόματα - ως ανήκουσα στο χρήστη - στη βάση δεδομένων με τη χρήση του UserRepository. Για λόγους απλότητας, όπως ήδη έχει αναφερθεί, στα πλαίσια του παρόντος έργου δεν έχει υλοποιηθεί κάποια διαδικασία ζευγοποίησης μεταξύ των συσκευών και η εφαρμογή μπορεί να συνδεθεί με όσα και οποια Puck.js θέλει.

- Το κουμπί σύνδεσης αλλάζει δυναμικά σε "DISCONNECT" όταν η συσκευή είναι συνδεδεμένη, και επιτρέπει την αποσύνδεση από τη συσκευή αυτή.
- Η σάρωση για συσκευές ξεκινά και σταματά με τις μεθόδους _startScan και _stopScan, και το UI ανανεώνεται μέσω του StreamBuilder κάθε φορά που υπάρχουν νέες συσκευές ή αλλαγές στην κατάσταση σύνδεσης.

Ο StreamBuilder είναι ένα widget στο Flutter που χτίζει το UI ανάλογα με τις αλλαγές σε ένα stream. Το stream περιέχει δεδομένα που έρχονται σε πραγματικό χρόνο και το builder ανακατασκευάζει το UI κάθε φορά που τα δεδομένα αλλάζουν. Χρησιμοποιείται για να παρακολουθεί τη ροή δεδομένων, όπως οι καταστάσεις σύνδεσης συσκευών ή τα αποτελέσματα σάρωσης BLE.

Η συνάρτηση addPuck της κλάσης UserRepository που πραγματοποιεί την επικοινωνία με το backend προς προσθήκη του Puck.js στη συλλογή του χρήστη είναι η παρακάτω:

```dart
Future<void> addPuck(String puckId) async {
  String? idToken = await getIdToken();

  if (idToken == null) {
    throw Exception('User not logged in');
  }

  final response = await http.post(
    Uri.parse('http://192.168.0.11:3000/profile_info/add_puck'),
    headers: {
      'Content-Type': 'application/json',
      'token': idToken,
    },
    body: json.encode({"puckId": puckId}),
  );

  if (response.statusCode == 204) {
    return;
  } else {
    throw Exception('Failed to add puck');
  }
}
```

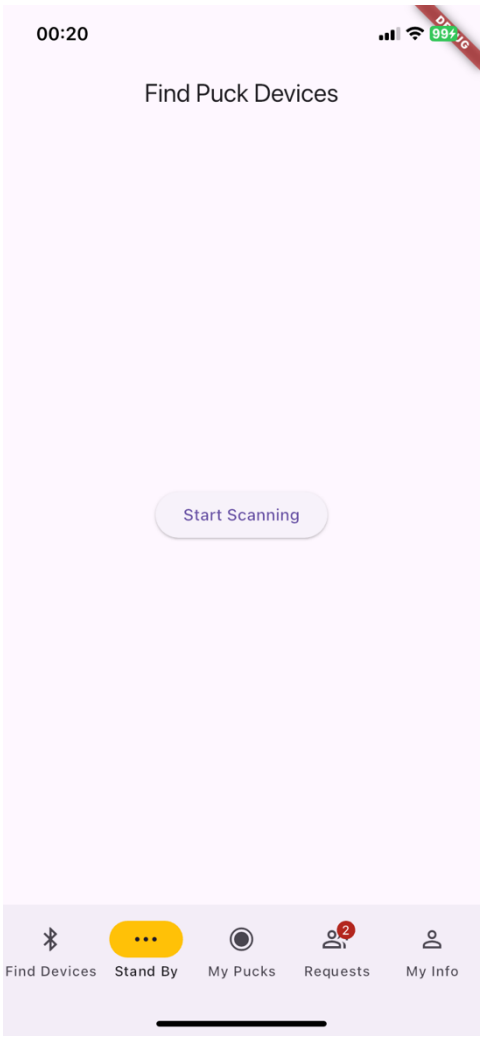
Εικόνα 35: Η συνάρτηση addPuck του UserRepository

Η λογική της είναι παρόμοια με όσα έχουν ήδη ειπωθεί για συναρτήσεις repository, με τη διαφορά ότι η κατάσταση επιτυχίας τώρα χαρακτηρίζεται με τον κωδικό 204 αντί για 200, για λόγους που θα εξηγηθούν στην ενότητα που αφορά το backend.

Ο κώδικας υλοποίησης της οθόνης My Pucks, καθώς και οι αντίστοιχες repository συναρτήσεις, παρότι έχει υλοποιηθεί εξ' ολοκλήρου παραλείπεται ως δευτερευούσης σημασίας για τα πλαίσια της εργασίας.

4.3.5 Λήψη δεδομένων για επείγοντα συμβάντα από συσκευές Puck.js

Προκειμένου να είναι δυνατή η σάρωση για συσκευές που εκπέμπουν την ύπαρξη νέων δεδομένων, ήταν απαραίτητο να δημιουργηθεί η οθόνη αναμονής (Stand by Screen) η οποία παρουσιάζεται στα δεξιά:



Εικόνα 36: Οθόνη αναμονής για λήψη επείγοντων μηνυμάτων

Για να αιτιολογηθεί η ανάγκη για αυτήν την οθόνη και να αναλυθεί ο κώδικας της, πρέπει πρώτα να γίνουν κατανοητά τα τεχνικά περιοριστικά μέτρα που ισχύουν για την εκτέλεση σαρώσεων Bluetooth σε συσκευές iOS. Η Apple επιτρέπει πολύ περιορισμένη λειτουργικότητα Bluetooth στο παρασκήνιο (background), γεγονός που κάνει αναποτελεσματική τη συνεχή παρακολούθηση για νέα δεδομένα όταν η εφαρμογή δεν είναι ενεργή στο προσκήνιο (foreground). Λόγω αυτού του περιορισμού, αποδείχτηκε απαραίτητη η δημιουργία μιας ειδικής οθόνης, της StandByScreen όπου ο χρήστης μπορεί να σκανάρει για συσκευές που μεταδίδουν τη σημαία νέων δεδομένων (υπηρεσία 0xFFFF), όπως έχει αναλυθεί ήδη στο προηγούμενο κεφάλαιο. Ο κώδικας της οθόνης παρατίθεται στην ακόλουθη εικόνα:

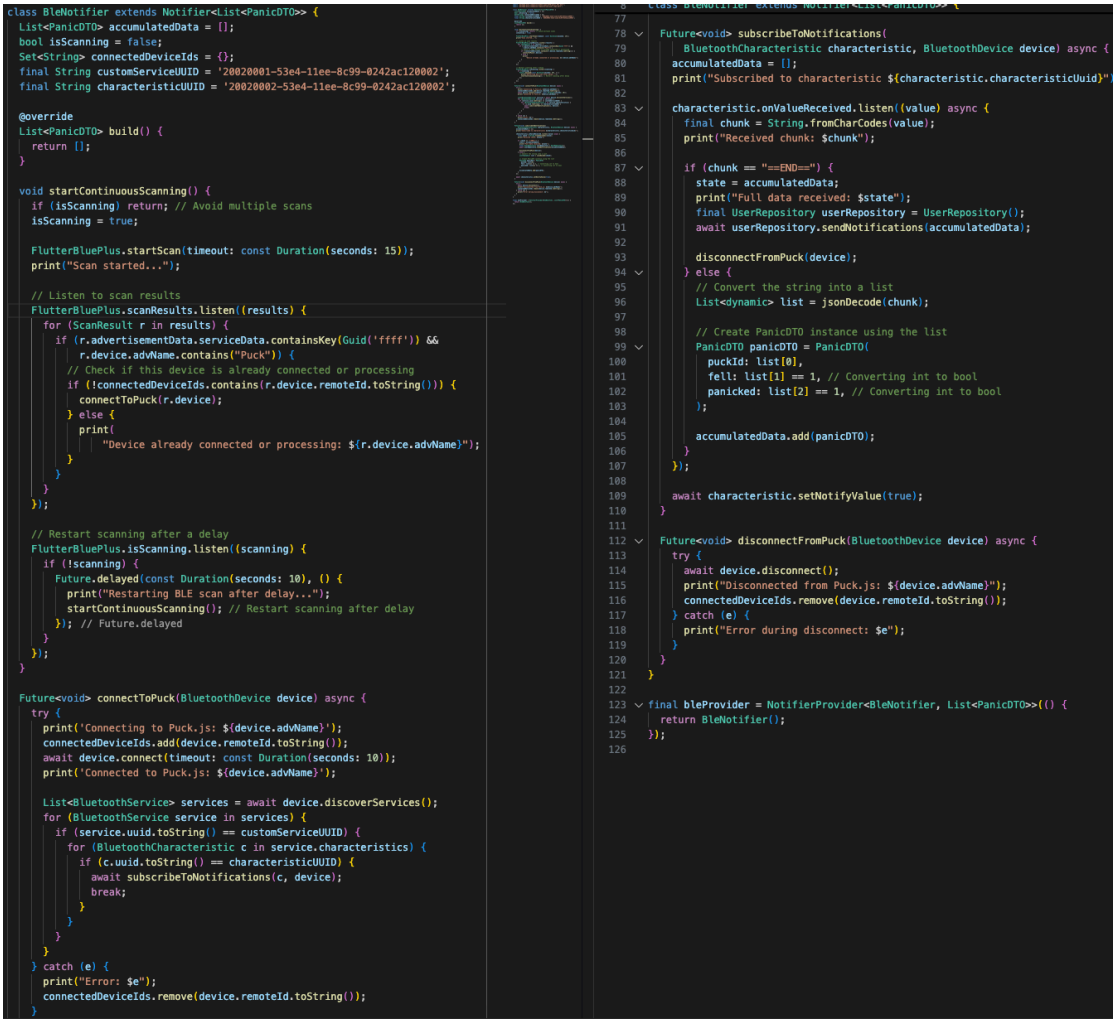
```dart
class StandByScreen extends ConsumerWidget {
  const StandByScreen({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final bleService = ref.read(bleProvider.notifier);
    final receivedData = ref.watch(bleProvider);

    return Scaffold(
      appBar: AppBar(
        title: const Text('Find Puck Devices'),
      ), // AppBar
      body: Center(
        child: receivedData.isEmpty
            ? ElevatedButton(
                onPressed: bleService.startContinuousScanning,
                child: const Text('Start Scanning'),
              ) // ElevatedButton
            : Text('Received Data: $receivedData'),
      ), // Center
    ); // Scaffold
  }
}
```

Εικόνα 37: Κώδικας οθόνης αναμονής

Η οθόνη StandByScreen χρησιμοποιεί τον Riverpod για τη διαχείριση της κατάστασης BLE, και την υπηρεσία bleProvider για τη διαχείριση της σάρωσης και της σύνδεσης με τις συσκευές Puck.js. Όταν ο χρήστης πατήσει το κουμπί "Start Scanning", ξεκινάει η διαδικασία συνεχούς σάρωσης μέσω του bleService.startContinuousScanning.

Αν βρεθούν συσκευές Puck.js που μεταδίδουν τη σημαία νέων δεδομένων, η εφαρμογή συνδέεται αυτόματα σε αυτές και αρχίζει να λαμβάνει δεδομένα, τα οποία αποστέλλονται στο backend μόλις ολοκληρωθεί η λήψη. Τα δεδομένα αυτά επιπλέον παρουσιάζονται στον χρήστη για σκοπούς αποσφαλμάτωσης κατά την υλοποίηση της εργασίας.

Ο κώδικας του provider που η StandByScreen αξιοποιεί για τις λειτουργίες και τη διαχείριση της κατάστασής της (state) παρατίθεται στις ακόλουθες εικόνες:



```dart
class BleNotifier extends Notifier<List<PanicDTO>> {
  List<PanicDTO> accumulatedData = [];
  bool isScanning = false;
  Set<String> connectedDeviceIds = {};
  final String customServiceUUID = '20020001-53e4-11ee-8c99-0242ac120002';
  final String characteristicUUID = '20020002-53e4-11ee-8c99-0242ac120002';

  @override
  List<PanicDTO> build() {
    return [];
  }

  void startContinuousScanning() {
    if (isScanning) return; // Avoid multiple scans
    isScanning = true;

    FlutterBluePlus.startScan(timeout: const Duration(seconds: 15));
    print("Scan started...");

    // Listen to scan results
    FlutterBluePlus.scanResults.listen((results) {
      for (ScanResult r in results) {
        if (r.advertisementData.serviceData.containsKey(Guid('ffff')) &&
            r.device.advName.contains("Puck")) {
          // Check if this device is already connected or processing
          if (!connectedDeviceIds.contains(r.device.remoteId.toString())) {
            connectToPuck(r.device);
          } else {
            print(
                "Device already connected or processing: ${r.device.advName}");
          }
        }
      }
    });

    // Restart scanning after a delay
    FlutterBluePlus.isScanning.listen((scanning) {
      if (!scanning) {
        Future.delayed(const Duration(seconds: 10), () {
          print("Restarting BLE scan after delay...");
          startContinuousScanning(); // Restart scanning after delay
        }); // Future.delayed
      }
    });
  }

  Future<void> connectToPuck(BluetoothDevice device) async {
    try {
      print('Connecting to Puck.js: ${device.advName}');
      connectedDeviceIds.add(device.remoteId.toString());
      await device.connect(timeout: const Duration(seconds: 10));
      print('Connected to Puck.js: ${device.advName}');

      List<BluetoothService> services = await device.discoverServices();
      for (BluetoothService service in services) {
        if (service.uuid.toString() == customServiceUUID) {
          for (BluetoothCharacteristic c in service.characteristics) {
            if (c.uuid.toString() == characteristicUUID) {
              await subscribeToNotifications(c, device);
              break;
            }
          }
        }
      }
    } catch (e) {
      print("Error: $e");
      connectedDeviceIds.remove(device.remoteId.toString());
    }

  Future<void> subscribeToNotifications(
      BluetoothCharacteristic characteristic, BluetoothDevice device) async {
    accumulatedData = [];
    print("Subscribed to characteristic ${characteristic.characteristicUuid}");

    characteristic.onValueReceived.listen((value) async {
      final chunk = String.fromCharCodes(value);
      print("Received chunk: $chunk");

      if (chunk == "==END==") {
        state = accumulatedData;
        print("Full data received: $state");
        final UserRepository userRepository = UserRepository();
        await userRepository.sendNotifications(accumulatedData);

        disconnectFromPuck(device);
      } else {
        // Convert the string into a list
        List<dynamic> list = jsonDecode(chunk);

        // Create PanicDTO instance using the list
        PanicDTO panicDTO = PanicDTO(
          puckId: list[0],
          fell: list[1] == 1, // Converting int to bool
          panicked: list[2] == 1, // Converting int to bool
        );

        accumulatedData.add(panicDTO);
      }
    });

    await characteristic.setNotifyValue(true);
  }

  Future<void> disconnectFromPuck(BluetoothDevice device) async {
    try {
      await device.disconnect();
      print("Disconnected from Puck.js: ${device.advName}");
      connectedDeviceIds.remove(device.remoteId.toString());
    } catch (e) {
      print("Error during disconnect: $e");
    }
  }
}

final bleProvider = NotifierProvider<BleNotifier, List<PanicDTO>>(() {
  return BleNotifier();
});
```

Εικόνα 38: Κώδικας του provider για λήψη μηνυμάτων από τα Puck.js

Ο provider BleNotifier είναι υπεύθυνος για την υλοποίηση όλης της λογικής που σχετίζεται με τη σάρωση, σύνδεση και λήψη δεδομένων από συσκευές Puck.js μέσω Bluetooth Low Energy (BLE). Η λειτουργία του χωρίζεται σε τρία κύρια βήματα: σάρωση για συσκευές, σύνδεση με συσκευές και λήψη δεδομένων.

- Σάρωση για Συσκευές: Η μέθοδος startContinuousScanning αρχίζει τη διαδικασία σάρωσης των συσκευών Puck.js. Αυτή η μέθοδος εκτελείται όταν ο χρήστης πατάει το κουμπί "Start Scanning" στην οθόνη. Χρησιμοποιεί την υπηρεσία FlutterBluePlus για να ξεκινήσει τη σάρωση και ορίζει ένα χρονικό όριο 15 δευτερολέπτων, για λόγους που εξηγήθηκαν στο κεφάλαιο 3. Η εφαρμογή ακούει τα αποτελέσματα της σάρωσης μέσω του scanResults.listen. Στα αποτελέσματα της σάρωσης, αν ανιχνευθεί μια συσκευή Puck.js που εκπέμπει την υπηρεσία 0xFFFF η εφαρμογή ελέγχει αν η συσκευή είναι ήδη συνδεδεμένη ή υπό επεξεργασία, χρησιμοποιώντας το σύνολο connectedDeviceIds. Αν η συσκευή δεν είναι συνδεδεμένη, τότε καλείται η μέθοδος connectToPuck για να γίνει σύνδεση.
- Σύνδεση με Συσκευές: Η μέθοδος connectToPuck αναλαμβάνει τη σύνδεση με τη συσκευή Puck.js. Αρχικά προστίθεται το αναγνωριστικό της συσκευής στο

connectedDeviceIds για να αποφευχθεί η επεξεργασία της ίδιας συσκευής δύο φορές. Η σύνδεση με τη συσκευή γίνεται με την κλήση της μεθόδου connect του BluetoothDevice, με ένα χρονικό όριο 10 δευτερολέπτων. Μόλις ολοκληρωθεί η σύνδεση, η εφαρμογή ανακαλύπτει τις διαθέσιμες υπηρεσίες της συσκευής μέσω της μεθόδου discoverServices. Η εφαρμογή αναζητά την υπηρεσία 20020001-53e4-11ee-8c99-0242ac120002, και όταν βρεθεί, προχωρά στη σύνδεση με την χαρακτηριστική μέσω της μεθόδου subscribeToNotifications.

- Λήψη Δεδομένων: Η μέθοδος subscribeToNotifications είναι υπεύθυνη για τη λήψη των δεδομένων από τη χαρακτηριστική της συσκευής Puck.js. Η σύνδεση με τη χαρακτηριστική ενεργοποιεί τις ειδοποιήσεις, επιτρέποντας στη συσκευή να στέλνει δεδομένα στην εφαρμογή σε πραγματικό χρόνο. Τα δεδομένα λαμβάνονται σε μορφή chunks μέσω του characteristic.onValueReceived.listen. Κάθε chunk αντιστοιχεί σε ένα ολόκληρο στροιχείο του πίνακα συμβάντων, όπως ήδη παρουσιάστηκε στην υποενότητα που αφορά στην πλευρά του Puck.js. Κάθε chunk μετατρέπεται από byte array σε string και αποθηκεύεται προσωρινά στο accumulatedData. Όταν η εφαρμογή λάβει το σήμα τερματισμού δεδομένων ("==END=="), συνδυάζει όλα τα chunks και τα προσθέτει στο state, που είναι η λίστα PanicDTO (Data Transfer Object). Μετά την ολοκλήρωση της λήψης δεδομένων, η εφαρμογή αποστέλλει τα δεδομένα στο backend μέσω του UserRepository.sendNotifications και αποσυνδέεται από τη συσκευή με τη μέθοδο disconnectFromPuck.
- Λογική για την Αποσύνδεση: Η αποσύνδεση από τη συσκευή πραγματοποιείται όταν έχουν ληφθεί όλα τα δεδομένα. Η μέθοδος disconnectFromPuck χειρίζεται την αποσύνδεση και αφαιρεί τη συσκευή από τη λίστα connectedDeviceIds, επιτρέποντας έτσι την επανασύνδεση της ίδιας συσκευής σε επόμενη σάρωση. Με αυτή τη λογική, ο provider επιτρέπει στη συσκευή να εντοπίζει και να συνδέεται αυτόματα με τις συσκευές Puck.js, λαμβάνοντας δεδομένα και στέλνοντάς τα στο backend, ακόμα κι αν η εφαρμογή δεν είναι συνεχώς συνδεδεμένη με αυτές.

Στη συνέχεια για λόγους πληρότητας παρατίθενται η συνάρτηση sendNotifications του UserRepository και ο ορισμός της κλάσης PanicDTO που είναι η μορφή αντικειμένων που ανταλλάζονται που στέλνει το frontend στο backend όσον αφορά συμβάντα πτώσης και πανικού. Η sendNotifications είναι σε πλήρη συμφωνία με όσα έχουν εξηγηθεί ήδη για τις repository functions και για το λόγο αυτό η ερμηνεία του κώδικα παραλείπεται. Το PanicDTO είναι ένας τυπικός ορισμός immutable κλάσης με χρήση της βιβλιοθήκης freezed, για το λόγο αυτό η ερμηνεία του κώδικα κρίνεται περιττή.

```dart
Future<void> sendNotifications(List<PanicDTO> panicEvents) async {
  String? idToken = await getIdToken();

  if (idToken == null) {
    throw Exception('User not logged in');
  }

  final List<Map<String, dynamic>> panicJsonList =
      panicEvents.map((event) => event.toJson()).toList();

  final response = await http.post(
    Uri.parse('http://192.168.0.11:3000/emails/send_all'),
    headers: {
      'Content-Type': 'application/json',
      'token': idToken,
    },
    body: json.encode(panicJsonList),
  );

  if (response.statusCode == 204) {
    return;
  } else {
    throw Exception('Failed to send notifications');
  }
}
```

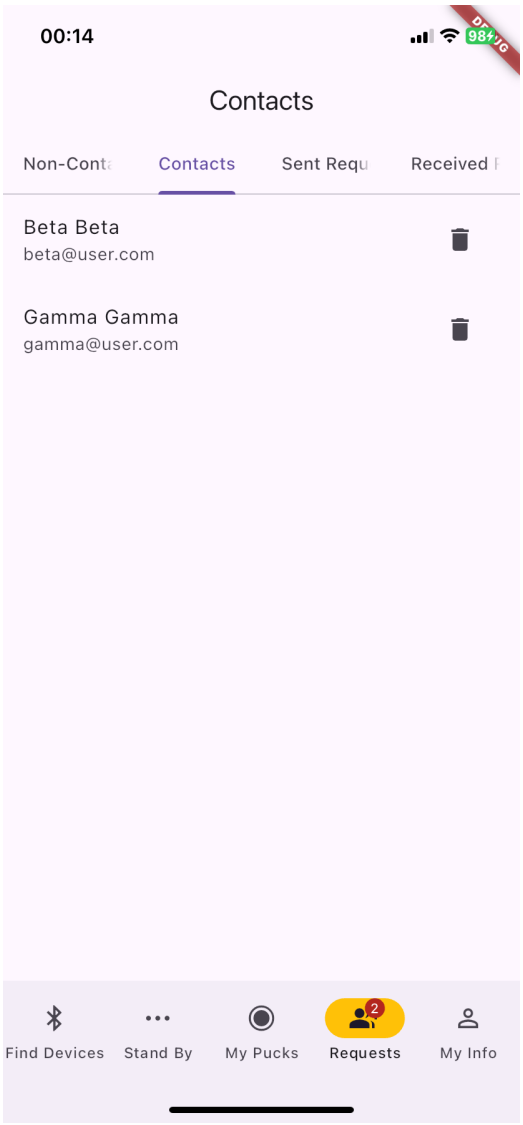
Εικόνα 40: Συνάρτηση sendNotifications του UserRepository

```dart
@freezed
class PanicDTO with _$PanicDTO {
  factory PanicDTO({
    required String puckId,
    required bool fell,
    required bool panicked,
  }) = _PanicDTO;

  factory PanicDTO.fromJson(Map<String, dynamic> json) =>
      _$PanicDTOFromJson(json);
}
```

Εικόνα 39: Ορισμός κλάσης PanicDTO

4.3.6 Λοιπές οθόνες

Ως επίλογος της παρούσας ενότητας παρουσιάζονται δύο οθόνες οι οποίες δεν διαδραματίζουν κρίσιμο ρόλο στα πλαίσια του συστήματος, αλλά προστέθηκαν προς εμπλουτισμό του συστήματος. Οι οθόνες είναι πλήρως λειτουργικές, ωστόσο ο κώδικας υλοποίησής τους και οι αντίστοιχες συναρτήσεις repository παραλείπονται. Η αριστερά οθόνη (requests) επιτρέπει στους χρήστες να βρίσκουν έμπιστες επαφές, να διαχειρίζονται τις επαφές τους, να στέλνουν αιτήματα για προσθήκη επαφής σε άλλους χρήστες, καθώς και να αποδέχονται ή να απορρίπτουν τα αιτήματα που λαμβάνουν οι ίδιοι. Η δεξιά οθόνη ()My info) παρέχει μία προβολή των στοιχείων του χρήστη, καθώς και το κουμπί αποσύνδεσης (logout):



Εικόνα 42: Οθόνη διαχείρισης έμπιστων επαφών και αιτημάτων

Εικόνα 41: Οθόνη προβολής προσωπικών πληροφοριών του χρήστη

4.4 REST API, κώδικας Backend και σχήμα βάσης δεδομένων

4.4.1 Σχήμα βάσης δεδομένων

Η παρούσα ενότητα ξεκινά με την παρουσίαση του σχήματος της βάσης δεδομένων στην παρακάτω εικόνα, ώστε να μπορεί να γίνει κατανοητό το backend σύστημα:

Εικόνα 43: Σχήμα βάσης δεδομένων

Όπως φαίνεται στο παραπάνω σχήμα, η βάση δεδομένων αποτελείται από τους ακόλουθους πίνακες:

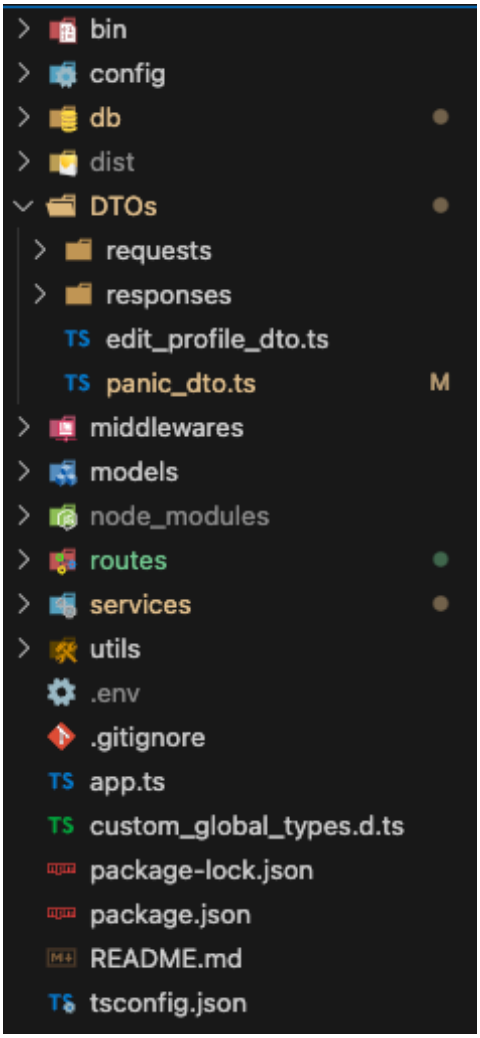
- users: Περιλαμβάνει τα βασικά στοιχεία κάθε χρήστη, όπως το όνομα χρήστη (username), το πλήρες όνομα (fullname), το email, την ημερομηνία γέννησης (birthday), τον κωδικό πρόσβασης και την ημερομηνία εγγραφής.
- trustedContacts: Συσχετίζει κάθε χρήστη με τις έμπιστες επαφές του, αποθηκεύοντας το ID του χρήστη (userId) και το ID της επαφής (contactId).
- contactRequests: Διαχειρίζεται τα αιτήματα προσθήκης επαφών, με το ID του αποστολέα (sender), του παραλήπτη (receiver), την ημερομηνία αποστολής (sentAt) και την ημερομηνία αποδοχής (acceptedAt).
- pucks: Αποθηκεύει τα στοιχεία κάθε συσκευής Puck.js, όπως το ID της συσκευής (id) και το ID του κατόχου (ownerId), συνδέοντας τη συσκευή με τον χρήστη.

Αυτό το σχήμα επιτρέπει τη διαχείριση των χρηστών, των συσκευών τους, και των επαφών τους για ειδοποιήσεις σε περίπτωση πτώσεων ή πανικού.

4.4.2 Αρχιτεκτονική κώδικα και αρχείων

Η αρχιτεκτονική με την οποία είναι δομημένος ο κώδικας και τα αρχεία βασίζεται στο modular pattern, ή αλλιώς κατάτμηση σε modules. Αυτή η δομή που ακολουθείται, συχνά αναφέρεται ως layered architecture ή modular layered architecture. Η λογική πίσω από αυτή τη δομή είναι ο διαχωρισμός της εφαρμογής σε διάφορα επίπεδα αφαίρεσης (abstraction layers), όπως το επίπεδο δρομολόγησης (routes), το επίπεδο υπηρεσιών (services), και το επίπεδο δεδομένων (models, db), ώστε να επιτευχθεί καθαρότητα στον κώδικα, οργάνωση και εύκολη συντήρηση.

Μια άλλη πιθανή ονομασία είναι το service layer pattern, καθώς η αρχιτεκτονική περιλαμβάνει την ύπαρξη ενός ξεχωριστού "service layer" που είναι υπεύθυνο για την επιχειρησιακή λογική (business logic) και την αλληλεπίδραση με τις πηγές δεδομένων (εδώ μόνο τη βάση δεδομένων), αλλά και με εξωτερικές υπηρεσίες όπως η Firebase (για αυθεντικοποίηση χρηστών) και η SendGrid (για την αποστολή emails). Αυτή η δομή επιτρέπει εύκολη επέκταση της λειτουργικότητας και ευελιξία στη διαχείριση των εξωτερικών εξαρτήσεων του συστήματος. Η δομή των φακέλων παρουσιάζεται στην δεξιά εικόνα:



Εικόνα 44: Δομή κώδικα Backend

Συγκεκριμένα, διακρίνονται οι εξής φάκελοι:

- bin: Περιέχει αρχεία για την εκκίνηση της εφαρμογής.
- config: Ρυθμίσεις για τη σύνδεση με τη βάσης δεδομένων.
- db: Περιέχει αρχεία για τα DDLs (Data Definition Language) και το σχήμα της βάσης δεδομένων.
- DTOs (Data Transfer Objects): Εδώ περιλαμβάνονται ορισμοί τύπων και διεπαφών αντικειμένων που χρησιμοποιούνται για τη μεταφορά δεδομένων μεταξύ frontend και backend. Ο φάκελος αυτός χωρίζεται περεταίρω στα requests, που αποτελούν αντικείμενα που η εφαρμογή στέλνει μέσα στο

σώμα των αιτημάτων που κάνει στο backend, και στο responses, τα οποία αποτελούν αντικείμενα που το backend στέλνει στα διάφορα αιτήματα του frontend μέσα στο σώμα της εκάστοτε επιτυχημένης απάντησης.

- middlewares: Περιέχει middleware που εκτελούνται πριν την επεξεργασία των αιτημάτων, όπως η διαχείριση σφαλμάτων ή ο έλεγχος αυθεντικοποίησης.
- models: Περιλαμβάνει μοντέλα δεδομένων που αντιστοιχούν στους πίνακες της βάσης.
- routes: Ορισμοί για τα routes της εφαρμογής και την επεξεργασία των αιτημάτων.
- services: Εδώ ζει όλη η λογική που αλληλεπιδρά με εξωτερικές υπηρεσίες και τη βάση δεδομένων και εκτελεί όλους τους απαραίτητους υπολογισμούς, μετασχηματισμούς και συνδυασμούς στα δεδομένα ώστε να παράξει την επιθυμητή απάντηση.
- utils: Βοηθητικές συναρτήσεις και εργαλεία για την εφαρμογή. Στην προκειμένη περίπτωση περιλαμβάνει ορισμένες απαραίτητες επεκτάσεις που πραγματοποιήθηκαν προκειμένου να επιτευχθεί πλήρης συμβατότητα της βιβλιοθήκης Kysely με τη MariaDB.

4.4.3 Σύνδεση του backend προγράμματος με τη βάση δεδομένων

4.4.3.1 Ρυθμίσεις και πραγματοποίηση σύνδεσης

Παρακάτω παρουσιάζεται το αρχείο που περιέχει τις ρυθμίσεις για τη βάση δεδομένων (αριστερή εικόνα), με βασικές παραμέτρους σύνδεσης, όπως τη διεύθυνση του οικοδεσπότη (host), το όνομα χρήστη, τον κωδικό πρόσβασης, και τη βάση δεδομένων στην οποία θα συνδεθεί η εφαρμογή. Οι τιμές σε ένα ρεαλιστικό πλαίσιο θα αποτελούσαν ευαίσθητες πληροφορίες, επομένως για λόγους ασφαλείας αλλά και εύκολης εναλλαγής μεταξύ μηχανημάτων, δεν γράφονται κατευθείαν στον κώδικα αλλά προέρχονται από το αρχείο περιβάλλοντος (το .env, ένα παράδειγμα για το αντίστοιχο τμήμα του οποίου παρουσιάζεται στη δεξιά εικόνα) και φορτώνονται δυναμικά μέσω της κλήσης process.env. Το connectionLimit ορίζει τον μέγιστο αριθμό των ταυτόχρονων συνδέσεων που μπορεί να δεχτεί η βάση (σε αυτή την περίπτωση 30). Πρόκειται για το μέγεθος της «δεξαμενής συνδέσεων» (connection pool). Η connection pool αποτελεί ένα σύνολο από προκατασκευασμένες συνδέσεις με τη

```
import { PoolConfig } from 'mariadb'

const dbConfig: PoolConfig = {
    host: process.env.db_host,
    user: process.env.db_user,
    password: process.env.db_pass,
    database: process.env.database,
    connectionLimit: 30,
}

export default dbConfig;
```

Εικόνα 46: Αρχείο ρυθμίσεων βάσης δεδομένων

```
db_host=localhost
db_user=puckAdmin
db_pass=mypassword
database=puckdatabase
```

Εικόνα 45: Τμήμα αρχείου περιβάλλοντος

βάση δεδομένων, οι οποίες επαναχρησιμοποιούνται για να βελτιώσουν την απόδοση της εφαρμογής, καθώς αποφεύγεται η συνεχής δημιουργία και καταστροφή συνδέσεων για κάθε αίτημα.

Τέλος παρατίθεται το αρχείο που δημιουργεί και διαχειρίζεται την πραγματική σύνδεση (για την ακρίβεια την connection pool) με τη βάση δεδομένων μέσω της βιβλιοθήκης Kysely, η οποία όπως αναλύθηκε στο Κεφάλαιο 2 ίναι υπεύθυνη για την ορισμό του τύπου της βάσης δεδομένων και την εκτέλεση των SQL ερωτημάτων με ασφάλεια και ακρίβεια. Για τη σύνδεση της Kysely με τη MariaDB απαιτήθηκε η δημιουργία της προσαρμοσμένης διαλέκτου MariaDBDialect, καθώς η MariaDB δεν υποστηρίζεται ακόμα από την Kysely (υποστηρίζεται όμως η MySQL, και οι διαφορές στη διάλεκτο είναι πολύ μικρές). Ο κώδικας της διαλέκτου παραλείπεται διότι ξεφεύγει του σκοπού της παρούσας εργασίας.

```
import dbConfig from '../config/db_config';
import { createPool } from 'mariadb'
import { Kysely } from 'kysely'
import { MariaDBDialect } from "../utils/MariaDBDialect/mariadb-dialect";
import Database from 'models/database';

const pool = createPool(dbConfig)

const db = new Kysely<Database>({
    dialect: new MariaDBDialect({
        pool: pool,
    }),
})

export { db, pool }
```

Εικόνα 47: Ορισθμός της σύνδεσης στη βα΄ση δεδομένων μέσω της Kysely

4.4.3.2 Ορισμός μοντέλων για την Kysely και αντιστοιχία με οντότητες της βάσης

Όπως αναφέρθηκε ήδη στο κεφάλαιο 2, η Kysely δεν είναι ORM, αλλά απλώς αποτελεί ένα ελαφρύ αφαιρετικό επίπεδο (abstraction layer) πάνω από την SQL. Για τη χρήση της βάσης δεδομένων μέσω αυτής δεν χρειάζεται να οριστούν κλάσεις, αλλά απαιτείται να οριστούν είτε τύποι (types) είτε διεπαφές (interfaces) της TypeScript, προκειμένου να επιτευχθεί ο συμπερασμός (type inference) άρα η ασφάλεια τύπων μέσω της βιβλιοθήκης. Οι τύποι ή οι διεπαφές αποτελούν λοιπόν τα μοντέλα στον κώδικα. Στη συνέχεια παρουσιάζονται ενδεικτικά η διεπαφή που μοντελοποιεί τη βάση δεδομένων και η διεπαφή που μοντελοποιεί τον πίνακα users της βάσης. Ο τύπος Generated χρησιμοποιείται για να δηλώσει πεδία που παράγονται αυτόματα στη βάση χωρίς να απαιτείται να της δωθεί ρητά κάποια τιμή (όπως για παράδειγμα είναι τα πεδία που ορίζονται ως auto increment). Η αντιστοιχία τύπων της MariaDB γίνεται εμ τύπους της TypeScript. Έτσι, τύποι όπως το char, το varchar ή το text, δηλώνονται όλοι ως string:

```
interface Database {
    users: UserTable
    trustedContacts: TrustedContact
    contactRequests: ContactRequest
    pucks: Puck
    sessions: { token: string }
}
```

Εικόνα 49: TypeScript μοντέλο της βάσης δεδομένων

```
interface UserTable {
    id: Generated<string>;
    username: string;
    fullname: string;
    email: string;
    birthday: string;
    password: string;
    registeredAt: Generated<Date>
}
```

Εικόνα 48: TypeScript μοντέλο του πίνακα users

4.4.4 Αυθεντικοποίηση, σύνδεση με τη Firebase, ρύθμιση Express.js και ορισμός διαδρομών

4.4.4.1 Ρύθμιση σύνδεσης με τη Firebase

Ο παρακάτω κώδικας αρχικοποιεί το Firebase Admin SDK στην εφαρμογή. Η συνάρτηση initializeApp ρυθμίζει την εφαρμογή χρησιμοποιώντας τα προεπιλεγμένα διαπιστευτήρια από το περιβάλλον, επιτρέποντας στην εφαρμογή να αλληλεπιδρά με τις υπηρεσίες Firebase:

```
import { applicationDefault, initializeApp } from 'firebase-admin/app';

initializeApp({
  credential: applicationDefault(),
});
```

Εικόνα 50: Αρχικοποίηση Firebase Admin SDK

Για να μπορεί η initializeApp να χρησιμοποιήσει τα προεπιλεγμένα διαπιστευτήρια, χρειάζεται να οριστεί στο αρχείο περιβάλλοντος .env ένα πεδίο σαν το παρακάτω. Το πεδίο αυτό δείχνει σε ένα αρχείο που περιέχει τα διαπιστευτήρια ενός service account της Firebase, το οποίο χρησιμοποιείται για ασφαλή αυθεντικοποίηση κατά την αλληλεπίδραση με τις υπηρεσίες Firebase Admin.

```
GOOGLE_APPLICATION_CREDENTIALS="/path/to/service-account-key.json"
```

Ένα service account είναι ένας ειδικός τύπος λογαριασμού που χρησιμοποιείται από εφαρμογές για αυθεντικοποίηση και αλληλεπίδραση με τη Firebase ή άλλες υπηρεσίες Google Cloud χωρίς να απαιτείται η συμμετοχή χρήστη. Συνήθως συνοδεύεται από ένα αρχείο κλειδιού που παρέχει δικαιώματα για την εκτέλεση διαχειριστικών εργασιών προγραμματιστικά.

4.4.4.2 Ορισμός διαδρομών (routes)

Αρχικά παρουσιάζεται το παρακάτω κομμάτι κώδικα, το οποίο αποτελεί το σημείο εκτελεί την δημιουργιά και την αρχική ρύθμιση της εφαρμογής Express.js, καθώς και τον ορισμό διαδρομών:

```
const app = express();

const unless = function (paths: string | string[], middleware: Function) {
  return function (req: Request, res: Response, next: NextFunction) {
    if (paths.includes(req.url) || paths.includes(req.path)) {
      return next();
    } else {
      return middleware(req, res, next);
    }
  };
};

//Attach Middlewares
//!!!! The order of middleware loading is important: !!!!
//middleware functions that are loaded first are also executed first.

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(unless(['/auth/login', '/auth/register', '/auth/otpCheck', '/auth/username_check'], tokenValidator))
app.use('/profile_info', usersRouter);
app.use('/auth', auth);
app.use('/settings', settings)
app.use('/emails', emails)
app.use('/contacts', contacts)

//Any route not found in attached routes, will throw a 404.
//Attach a middleware tha catches 404 and forwards to error handler
app.use(function (req: Request, res: Response, next: NextFunction) {
  next(createError(404));
});

//Attach error handler as the last middleware
app.use(errorHandler);
```

Εικόνα 51: Ρύθμιση Express.js και ορισμός διαδρομών

Ο παραπάνω κώδικας υλοποιεί μια εφαρμογή Express.js και ρυθμίζει διάφορα middleware για να διαχειριστεί τα αιτήματα HTTP. Αρχικά, φορτώνει εργαλεία για την καταγραφή των αιτημάτων (logger), parsers για τη διαχείριση JSON και cookies, και χρησιμοποιεί έναν έλεγχο αυθεντικοποίησης μέσω του middleware tokenValidator, ο οποίος εκτελείται για όλες τις διαδρομές εκτός από αυτές που αφορούν τη διαδικασία αυθεντικοποίησης, όπως το login και το register. Ο tokenValidator, ο οποίος θα παρουσιαστεί στην επόμενη υποενότητα, διασφαλίζει ότι μόνο εξουσιοδοτημένα αιτήματα προχωρούν. Στη συνέχεια ορίζονται οι διαδρομές /profile_info, /auth, /settings, /emails, και /contacts, οι οποίες κατευθύνουν τα αιτήματα στους αντίστοιχους χειριστές (route handlers). Ο κώδικας περιλαμβάνει επίσης διαχείριση σφαλμάτων για αιτήματα που δεν βρίσκουν αντιστοιχία στις παραπάνω διαδρομές, καθώς και ένα middleware για τον χειρισμό σφαλμάτων 404 και άλλων λαθών, το οποίο εκτελείται τελευταίο.

4.4.4.3 Επικυρωτής διακριτικού (token validator)

Ο token validator είναι ένα middleware που ελέγχει την αυθεντικοποίηση, επιτρέποντας ή απορρίπτοντας την πρόσβαση στις προστατευμένες διαδρομές της εφαρμογής. Υλοποιείται με τον ακόλουθο κώδικα:

```
const tokenValidator = async (req: Request, res: Response, next: NextFunction) => {
    const token = req.header('token')

    if (!token)
        next(createError.Unauthorized())
    else {
        try {
            const decodedToken = await auth().verifyIdToken(token);

            req.decodedToken = decodedToken
            next()
        }
        catch (err: any) {
            if (err.code === 'auth/argument-error' || err.code === 'auth/id-token-expired') {
                next(createError.Unauthorized('Invalid or expired token'));
            } else {
                next(err instanceof createError[401] ? err : createError(500, err, { message: "Server error" }))
            }
        }
    }
}
```

Εικόνα 52: Κώδικα επικύρωσης διακριτικού

Ο ρόλος του επικυρωτή είναι να ελέγχει την ύπαρξη ενός ID token στα αιτήματα που στέλνει η εφαρμογή και να επιβεβαιώνει την εγκυρότητα του διακριτικού αυτού. Αρχικά, ελέγχει αν υπάρχει διακριτικό στο αίτημα μέσω της κεφαλίδας req.header('token'). Αν δεν υπάρχει, επιστρέφει σφάλμα με κωδικό κατάστασης 401, ο οποίος αντιστοιχεί σε απάντηση «μη αυθεντικοποιημένος πελάτης» όπως αυτός υπαγορεύεται από το πρότυπο HTTP [39]. Αν υπάρχει, προχωράει στην επαλήθευσή του χρησιμοποιώντας τη μέθοδο auth().verifyIdToken του Firebase Admin SDK. Αν το διακριτικό είναι ένα έγκυρο Firebase ID token , αποθηκεύει τα δεδομένα του διακριτικού στο req.decodedToken για χρήση από τα επόμενα middleware συνεχίζει στο επόμενο middleware. Σε περίπτωση μη έγκυρου ή ληγμένου διακριτικού, επιστρέφει ξανά σφάλμα με κωδικό κατάστασης 401 και το μήνυμα "Invalid or expired token" στο σώμα της απάντησης. Οποιαδήποτε άλλα σφάλματα οδηγούν σε απάντηση με κωδικό σφάλματος 500, ο οποίος αντιστοιχεί σε γενικό σφάλμα του διακομιστή σύμφωνα με το πρότυπο HTTP.

4.4.4.4 Διαδρομές και υπηρεσίες αυθεντικοποίησης και εγγραφής χρηστών

Ο κώδικας των διαδρομών (routes) για την εγγραφή (register) και την είσοδο (login) χρηστών είναι ο εξής:

```
router.post('/register', async (req: Request, res: Response, next: NextFunction) => {
    try {
        typia.assertGuardEquals<CreateProfileDTO>(req.body)
        const createProfileDTO: CreateProfileDTO = req.body;

        const auth: AuthResponse = await profileInfoService.createUser(createProfileDTO)

        res.status(201).json(auth)
    }
    catch (err) {
        if (err instanceof SqlError && err.errno === 1062) next(createError(409, 'User already exists'))
        else if (err instanceof TypeGuardError) next(createError(400, 'Invalid URL fields'))
        else next(createError(500, err instanceof Error ? err : { err, message: 'Unknown Error occured' }))
    }
})

router.post('/login', async (req: Request, res: Response, next: NextFunction) => {
    try {
        typia.assertGuardEquals<LoginReqBody>(req.body)
        const password: string = req.body.password
        const email: string = req.body.email

        const auth: AuthResponse = await authService.getAuth(email, password)
        res.status(200).json(auth)
    }
    catch (err) {
        err instanceof TypeGuardError ?
            next(createError(400, 'Invalid URL fields')) :
            next(createError(err instanceof WrongPasswordException ? 401 :
                err instanceof UserNotFoundException ? 404 :
                    500, err instanceof Error ? err : { err, message: 'Unknown Error occured' }))
    }
})
```

Εικόνα 53: Κώδικας διαδρομών login και register

Χρησιμοποιείται η βιβλιοθήκη typia για την επικύρωση των δεδομένων εισόδου (request validation), διασφαλίζοντας ότι τα δεδομένα που αποστέλλονται ταιριάζουν με τα αναμενόμενα πρότυπα.

Στο route /register, ο κώδικας λαμβάνει τα δεδομένα από τον χρήστη, τα επικυρώνει με το CreateProfileDTO και καλεί τη συνάρτηση createUser από το service για τη δημιουργία του χρήστη στη βάση. Αν η εγγραφή του χρήστη είναι επιτυχής, δημιουργείται ένα custom token μέσω της Firebase και αποστέλλεται ως απάντηση στο αίτημα. Αντίθετα, αν η εγγραφή αποτύχει επειδή υπάρχει ήδη χρήστης με αυτά τα στοιχεία, πιάνεται η εξαίρεση διπλού μοναδικού κλειδιού της βάσης και επιστρέφεται κωδικός σφάλματος 409, ο οποίος σύμφωνα με το πρότυπο HTTP αντιστοιχεί σε σύγκρουση (conflict) με την παρούσα κατάσταση του διακομιστή.

Στο route /login, γίνεται παρόμοια επικύρωση των δεδομένων με το LoginReqBody, και στη συνέχεια η συνάρτηση getAuth αναζητά τον χρήστη στη βάση δεδομένων. Αν βρεθεί και ο κωδικός είναι σωστός, δημιουργείται και επιστρέφεται ως απάντηση ένα custom token. Αντίθετα, αν ο κωδικός είναι λανθασμένος επιστρέφεται σφάλμα με κωδικό κατάστασης 401, ενώ αν ο χρήστης δεν βρεθεί επιστρέφεται κωδικός σφάλματος 404, ο οποίος αντιστοιχεί σε αδυναμία του διακομιστή να βρει το ζητούμενο πόρο, σύμφωνα με το πρότυπο HTTP.

Και στις δύο παραπάνω περιπτώσεις, όπως και σε όλα τα υπόλοιπα endpoints του συστήματος, αν η επικύρωση του αιτήματος αποτύχει, το backend απαντά με τον

κωδικό κατάστασης σφάλματος 400, ο οποίος, σύμφωνα με το πρότυπο HTTP, αντιστοιχεί σε αδυναμία ή άρνηση του διακομιστή να επεξεργαστεί το αίτημα, λόγω κάποιου σφάλματος που θεωρείται ότι προέρχεται από τον πελάτη (π.χ., εσφαλμένη σύνταξη αιτήματος, μη έγκυρη δόμηση μηνύματος αιτήματος ή παραπλανητική δρομολόγηση αιτήματος) χωρίς να προχωρήσει σε επεξεργασία του αιτήματος. Ομοίως, σε κάθε περίπτωση, τα σφάλματα που δεν αφορούν κάποια γνωστή περίπτωση οδηγούν σε απάντηση με κωδικό σφάλματος 500.

Τα DTOs και ο κώδιακς των αντίστοιχων services παρουσιάζονται παρακάτω:

```
export type Username = string & tags.MinLength<4> & tags.MaxLength<30> & tags.Pattern<"^[a-zA-Z0-9._]+$">
export type Fullname = string & tags.MinLength<4> & tags.MaxLength<30> & tags.Pattern<"^[A-Z][a-z]+ [A-Z][a-z]+$">

type CreateProfileDTO = {
    username: Username;
    fullname: Fullname;
    email: Email;
    birthday: string & Format<'date'>;
    password: Password;
}
```

Εικόνα 55: Το DTO για την εγγραφή χρήστη

```
export type Password = string & tags.MinLength<8> & tags.MaxLength<128> & tags.Pattern<"^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d).+$">
export type Email = string & tags.MinLength<4> & tags.MaxLength<254> & tags.Format<"email">

You, 3 weeks ago | 1 author (You)
export interface LoginReqBody {
    email: Email
    password: Password
}
```

Εικόνα 56: Το DTO για τη σύνδεση χρήστη

```
const getAuth = async (mail: string, password: string) => {

    const user: User = await db
        .selectFrom('users')
        .selectAll()
        .where((eb) => eb.or([
            eb('email', '=', mail),
            eb('username', '=', mail)
        ]))
        .executeTakeFirstOrThrow(() => new UserNotFoundException())

    if (await bcrypt.compare(password, user.password)) {

        const token: string | undefined = await auth().createCustomToken(user.id)

        if (!token) throw new Error('Failed to issue custom token')

        return { token };
    } else {
        throw new WrongPasswordException();
    }
};
```

Εικόνα 54: Κώδικας υπηρεσίας σύνδεσης χρήστη

```
const createUser = async (createProfileDTO: CreateProfileDTO) => {
  const salt = await bcrypt.genSalt(8);
  const passwordHashed = await bcrypt.hash(createProfileDTO.password, salt);

  const userId = await db
    .insertInto('users')
    .values({
      ...createProfileDTO,
      password: passwordHashed,
    })
    .returning('id')
    .executeTakeFirstOrThrow()

  const token: string | undefined = await auth().createCustomToken(userId.id)

  if (!token) throw new Error('Failed to issue custom token. Please try to login manually')

  return { token }
};
```

Εικόνα 57: Κώδικας υπηρεσίας εγγραφής χρήστη

Στις υπηρεσίες (services), οι δύο βασικές συναρτήσεις createUser και getAuth αλληλεπιδρούν με τη βάση δεδομένων και το Firebase για να υλοποιήσουν τη λειτουργία της εγγραφής και της αυθεντικοποίησης χρηστών. Ακολουθεί μια πιο λεπτομερής εξήγηση για καθεμία.

1) createUser

- Παίρνει τα δεδομένα από το CreateProfileDTO, που περιέχουν πληροφορίες όπως το email και τον κωδικό πρόσβασης του χρήστη.
- Πριν αποθηκεύσει τον κωδικό πρόσβασης στη βάση δεδομένων, χρησιμοποιεί την bcrypt για τη δημιουργία ενός salted hash (με την εντολή bcrypt.hash()), το οποίο παρέχει μεγαλύτερη ασφάλεια στον κωδικό προς αποφυγή διαρροών κωδικών.
- Τα δεδομένα του χρήστη αποθηκεύονται στη βάση δεδομένων με την εντολή insertInto() από το Kysely, η οποία μετατρέπει σε ερώτημα insert της MariaDB. Αφού αποθηκευτεί, η συνάρτηση επιστρέφει το ID του νέου χρήστη.
- Στη συνέχεια, καλείται η auth().createCustomToken() της Firebase Admin SDK για τη δημιουργία ενός custom token που συνδέεται με το συγκεκριμένο χρήστη. Αυτό το token επιστρέφεται στον client για χρήση με την εφαρμογή.
- Αν υπάρξει πρόβλημα κατά τη δημιουργία του token, πετάγεται ένα γενικό σφάλμα.

2) getAuth:

- Ψάχνει στη βάση δεδομένων για τον χρήστη χρησιμοποιώντας είτε το email είτε το username (με την εντολή or στο Kysely query). Αν ο χρήστης δεν βρεθεί, πετάγεται η εξαίρεση UserNotFoundException.
- Αν ο χρήστης βρεθεί, χρησιμοποιείται η bcrypt.compare() για να συγκριθεί ο κωδικός πρόσβασης που έδωσε ο χρήστης με το hashed password που είναι

αποθηκευμένο στη βάση. Αν ταιριάζουν, δημιουργείται ένα custom token με την auth().createCustomToken().

- Αν το password δεν ταιριάζει, ρίχνεται η εξαίρεση WrongPasswordException.
- Τέλος, το custom token επιστρέφεται στον client για χρήση στην αυθεντικοποίηση.

4.4.5 Αποστολή ειδοποιητηρίων emails για επείγοντα συμβάντα

Στην παρούσα υποενότητα παρουσιάζεται ο κώδικας που αφορά την αποστολή ειδοποιήσεων μέσω μηνυμάτων ηλεκτρονικού ταχυδρομείου στις έμπιστες επαφές των χρηστών για τους οποίους εντοπίστηκε κρίσιμο συμβάν (πτώση ή πανικός). Το PanicDTO παραλείπεται καθώς παρουσιάστηκε ήδη στην ενότητα που αφορά τη συνοδευτική εφαρμογή.

```
router.post('/send_all', async (req: Request, res: Response, next: NextFunction) => {
    const userId: string = req.decodedToken!.user_id;
    const panicList: PanicDTO[] = req.body;
    try {

        await emailService.sendAllEmails(panicList)

        res.status(200).json({ message: 'Emails sent successfully' })

    }
    catch (err) {
        next(createError(500, err instanceof Error ? err : { err, message: 'Unknown Error occured' }))
    }
})
```

Εικόνα 58: Κώδικας διαδρομής για αποστολή ειδοποιήσεων

Η επεξήγηση του κώδικα του route παραλείπεται καθώς η λογική του είναι πανομοιότυπη με όσα routes έχουν ήδη παρουσιαστεί στα προηγούμενα. Όσον αφορά τον κώδικα της υπηρεσίας, υλοποιεί τη λογική για την αποστολή μαζικών email ειδοποιήσεων στις εμπιστευμένες επαφές κάθε χρήστη, σε περίπτωση που ανιχνευτεί πτώση ή ενεργοποιηθεί το κουμπί πανικού σε κάποια συσκευή Puck.js. Αρχικά, συλλέγονται οι αναγνωριστικοί αριθμοί (puckIds) των συσκευών από τη λίστα panicList. Στη συνέχεια, με χρήση SQL ερωτήματος, το σύστημα ανακτά από τη βάση δεδομένων τις εμπιστευμένες επαφές κάθε ιδιοκτήτη συσκευής και τις οργανώνει σε ομάδες. Το API κλειδί της SendGrid ρυθμίζεται και τα δεδομένα της κάθε συσκευής εμπλουτίζονται με τις πληροφορίες του αν το συμβάν είναι πτώση ή πανικός.

Μετά, για κάθε συσκευή, δημιουργείται το περιεχόμενο του μηνύματος email. Το email περιέχει τον αναγνωριστικό αριθμό της συσκευής (puckId), το όνομα χρήστη του ιδιοκτήτη, και την περιγραφή του συμβάντος (πτώση ή πανικός). Το email αποστέλλεται σε όλες τις εμπιστευμένες επαφές που αντιστοιχούν σε κάθε ιδιοκτήτη, με χρήση της μεθόδου sgMail.send. Τέλος, γίνεται logging της επιτυχούς αποστολής ή τυχόν σφαλμάτων.

```
const sendAllEmails = async (panicList: PanicDTO[]) => {
  const puckIds: string[] = panicList.map((entry) => entry.puckId);

  const recipientsData = await db
    .selectFrom("pucks")
    .innerJoin("users", "users.id", "pucks.ownerId")
    .innerJoin("trustedContacts", "trustedContacts.userId", "users.id")
    .innerJoin("users as contacts", "contacts.id", "trustedContacts.contactId")
    .select([
      "pucks.id as puckId",
      "users.username as ownerUsername",
      "users.id as ownerId",
    ])
    .where("pucks.id", "in", puckIds)
    .groupBy('ownerId')
    .select(sql`JSON_EXTRACT(
      COALESCE(
      JSON_ARRAYAGG(contacts.email
      ),
      '[]'
      ),
      '$'
  )`.$castTo<string[]>().as('recipients'))
    .execute();

  sgMail.setApiKey(process.env.SENDGRID_API_KEY || "")

  const updatedRecipientsData = recipientsData.map((recipient) => {
    const panicData = panicList.find(panic => panic.puckId === recipient.puckId);
    return {
      ...recipient,
      fell: panicData?.fell || false,
      panicked: panicData?.panicked || false,
    };
  })

  for (const recipient of updatedRecipientsData) {
    const { puckId, ownerUsername, recipients, fell, panicked } = recipient;

    const eventMessage = fell
      ? 'a fall was detected'
      : panicked
        ? 'a panic button was pressed'
        : 'no specific event occurred';

    const msg = {
      to: recipients, // The recipients array for bulk sending
      from: 'puckpanicbutton@gmail.com',
      subject: `Panic Event from ${ownerUsername}`,
      html: `<strong>Puck ID: ${puckId}</strong><br>
             Event: ${eventMessage}<br>
             Owner: ${ownerUsername}`,
    };

    // Send the email
    try {
      await sgMail.send(msg);
      console.log(`Email sent to: ${recipients.join(", ")}`);
    } catch (error) {
      console.error(`Failed to send email for puckId ${puckId}:`, error);
    }
  }
};
```

Εικόνα 59: Κώδικας υπηρεσίας για αποστολή ειδοποιήσεων

4.4.6 Λοιπές διαδρομές και υπηρεσίες

Ως επίλογος της παρούσας ενότητας παρουσιάζονται οι κώδικες για τις διαδρομές και τις υπηρεσίες που σχετίζονται με την διαχείριση των Puck.js συσκευών του χρήστη. Η εξήγηση του κώδικα παραλείπεται. Τέλος σημειώνεται ότι ο κώδικας που αφορά άλλες λειτουργίες, οι οποίες δεν διαδραματίζουν κρίσιμο ρόλο στα πλαίσια του συστήματος, αλλά προστέθηκαν προς εμπλουτισμό του, (όπως η λειτουργία προσθήκης ή διαχείρισης έμπιστων επαφών) αν και πλήρως λειτουργικός παραλείπεται ως δευτερευούσης σημασίας.

```
const getMyPucks = async (userId: string) => db
  .selectFrom("pucks")
  .select("id as puckId")
  .where("ownerId", "=", userId)
  .execute()

const addPuck = async (
  userId: string,
  puckId: string,
) =>
  db
    .insertInto("pucks")
    .values({ id: puckId, ownerId: userId })
    .executeTakeFirstOrThrow()


const removePuck = async (
  userId: string,
  puckId: string,
) =>
  db
    .deleteFrom("pucks")
```

Εικόνα 60: Κώδικας υπηρεσιών διαχείρισης των Puck.js

```
router.get('/my_pucks', async (req: Request, res: Response, next: NextFunction) => {
    const userId: string = req.decodedToken!.user_id
    try {
        const puckIds = await profileInfoService.getMyPucks(userId)
        res.status(200).json(puckIds)
    }
    catch (err) {
        next(createError(500, err instanceof Error ? err : { err, message: 'Unknown Error occured' }))
    }
})

router.post('/add_puck', async (req: Request, res: Response, next: NextFunction) => {
    try {
        typia.assertGuardEquals<{ puckId: string }>(req.body)
        const userId: string = req.decodedToken!.user_id
        const puckId: string = req.body.puckId

        await profileInfoService.addPuck(userId, puckId)
        res.sendStatus(204)
    }
    catch (err) {
        next(createError(500, err instanceof Error ? err : { err, message: 'Unknown Error occured' }))
    }
})

router.delete('/remove_puck', async (req: Request, res: Response, next: NextFunction) => {
    try {
        typia.assertGuardEquals<{ puckId: string }>(req.body)
        const userId: string = req.decodedToken!.user_id
        const puckId: string = req.body.puckId

        await profileInfoService.removePuck(userId, puckId)
        res.sendStatus(204)
    }
    catch (err) {
        next(createError(500, err instanceof Error ? err : { err, message: 'Unknown Error occured' }))
    }
})
```

Εικόνα 61: Κώδικας διαδρομών διαχείρισης των Puck.js

4.5 Κώδικας υλοποίησης, εκπαίδευσης και αξιολόγησης Νευρωνικού Δικτύου

4.5.1 Υλοποίηση Νευρωνικού Δικτύου

4.5.1.1 Εισαγωγή

Η υλοποίηση του Νευρωνικού Δικτύου που θα παρουσιαστεί στην παρούσα ενότητα είναι μινιμαλιστική, ειδικά προσαρμοσμένη για να λειτουργεί σε συσκευές με περιορισμένους πόρους, όπως το Puck.js. Ιδιαίτερο εμπόδιο αποτελεί η μνήμη. Ορίζονται κλάσεις για τις βασικές δομές όπως οι νευρώνες και τα επίπεδα (layers), καθώς και δημοφιλείς συναρτήσεις ενεργοποίησης όπως η σιγμοειδής (sigmoid), η relu, και η υπερβολική εφαπτομένη (tanh). Η βελτιστοποίηση και η επαναφορά gradient γίνεται με οπισθοδιάδοση (backpropagation), με τη δυνατότητα υπολογισμού παραγώγων (gradients) για τη διαδικασία εκπαίδευσης του μοντέλου. Αυτή η απλή δομή μπορεί να επεκταθεί με περισσότερες λειτουργίες, αλλά προσφέρει επαρκή λειτουργικότητα για μικρές συσκευές και προβλήματα με περιορισμένη χωρητικότητα μνήμης και επεξεργαστικής ισχύος.

4.5.1.2 Η κλάση Value

Η κλάση Value αντιπροσωπεύει αριθμητικές τιμές και ενσωματώνει τη λειτουργικότητα για την αποθήκευση και τον υπολογισμό των παραγώγων τους. Κάθε αντικείμενο αυτής της κλάσης περιέχει την αριθμητική τιμή (data), την παράγωγο της τιμής αυτής (grad), καθώς και μια συνάρτηση οπισθοδιάδοσης (backward) που υπολογίζει τις παραγώγους για τις πράξεις που εμπλέκουν την τιμή. Η κλάση υποστηρίζει βασικές αριθμητικές πράξεις όπως πρόσθεση, αφαίρεση, πολλαπλασιασμό, διαίρεση, καθώς και μη γραμμικές συναρτήσεις όπως η ReLU και η υπερβολική εφαπτομένη (tanh). Το πλεονέκτημα της κλάσης είναι ότι παρακολουθεί αυτόματα το διάγραμμα των υπολογισμών και υποστηρίζει την οπισθοδιάδοση για τον υπολογισμό των παραγώγων όλων των ενδιάμεσων τιμών. Ο κώδικας υλοποίησής της είναι ο ακόλουθος:

```
class Value {
    constructor({ data, children = [], op = '', label = '' }) {
        this.data = data;
        this.grad = 0.0;
        this.backward = () => { };
        this.children = children;
        this.op = op;
        this.label = label;
    }

    add(otherArg) {
        let other;

        if (otherArg instanceof Value) {
            other = otherArg;
        } else if (typeof otherArg === 'number') {
            other = new Value({ data: otherArg });
        }
        else {
            throw new Error("Invalid argument type");
        }

        const out = new Value({ data: this.data + other.data, children: [this, other], op: '+' });

        out.backward = () => {
            this.grad += out.grad;
            other.grad += out.grad;
        };

        return out;
    }

    sub(otherArg) {
        let other;

        if (otherArg instanceof Value) {
            other = otherArg;
        } else if (typeof otherArg === 'number') {
            other = new Value({ data: otherArg });
        }
        else {
            throw new Error("Invalid argument type");
        }

        const out = new Value({ data: this.data - other.data, children: [this, other], op: '-' });

        out.backward = () => {
            this.grad += out.grad;
            other.grad += out.grad;
        };

        return out;
    }

    mul(otherArg) {

        let other;

        if (otherArg instanceof Value) {
            other = otherArg;
        } else if (typeof otherArg === 'number') {
            other = new Value({ data: otherArg });
        }
        else {
            throw new Error("Invalid argument type");
        }
```

```
    mul(otherArg) {

        const out = new Value({ data: this.data * other.data, children: [this, other], op: '*' });

        out.backward = () => {
            this.grad += other.data * out.grad;
            other.grad += this.data * out.grad;
        };

        return out;
    }

    div(otherArg) {

        let other;

        if (otherArg instanceof Value) {
            other = otherArg;
        } else if (typeof otherArg === 'number') {
            other = new Value({ data: otherArg });
        }
        else {
            throw new Error("Invalid argument type");
        }

        return this.mul(other.power(-1));
    }

    power(exponent) {
        if (typeof exponent === 'number') {
            const out = new Value({ data: this.data ** exponent, children: [this], op: `**${exponent}` });

            out.backward = () => {
                this.grad += exponent * (this.data ** (exponent - 1)) * out.grad;
            };

            return out;
        }
        else {
            throw new Error("Invalid argument type");
        }
    }

    tanh() {
        const out = new Value({ data: Math.tanh(this.data), children: [this], op: 'tanh' });

        out.backward = () => {
            this.grad += (1 - out.data ** 2) * out.grad;
        };

        return out;
    }

    exp() {
        const out = new Value({ data: Math.exp(this.data), children: [this], op: 'exp' });

        out.backward = () => {
            this.grad += out.data * out.grad;
        };

        return out;
    }

    log() {
        const out = new Value({ data: Math.log10(this.data + Number.EPSILON), children: [this], op: 'log' });
```

Εικόνα 62: Κλάση Value, μέρος α'

```
log() {
    const out = new Value({ data: Math.log10(this.data + Number.EPSILON), children: [this], op: 'log' });

    out.backward = () => {
        this.grad += (1 / (this.data + Number.EPSILON * Math.log(10))) * out.grad;
    };

    return out;
}

sigmoid() {
    const out = new Value({ data: sigmoid(this.data), children: [this], op: 'sigmoid' });

    out.backward = () => {
        this.grad += out.data * (1 - out.data) * out.grad;
    };

    return out;
}

relu() {
    const out = new Value({ data: relu(this.data), children: [this], op: 'relu' });

    out.backward = () => {
        this.grad += this.data > 0 ? out.grad : 0;
    };

    return out;
}


fullBackward() {
    const topo = [];
    const visited = new Set();

    const build_topo = (v) => {
        if (!visited.has(v)) {
            visited.add(v);
            for (const child of v.children)
                build_topo(child);
            topo.push(v);
        }
    };
    build_topo(this);

    this.grad = 1.0;

    for (const node of topo.reverse())
        node.backward();
}

toString() {
    return `Value(data=${this.data})`;
}

toJSON() {
    return this.data;
}
```

Εικόνα 63: Κλάση Value, μέρος β'

4.5.1.3 Η κλάση Neuron

Η κλάση Neuron αντιπροσωπεύει έναν νευρώνα μέσα σε ένα νευρωνικό δίκτυο. Κάθε νευρώνας περιλαμβάνει έναν πίνακα βαρών (weights) και μια προκατάληψη (bias), οι οποίες αρχικοποιούνται τυχαία. Η κλάση έχει τη μέθοδο call που υπολογίζει την έξοδο του νευρώνα, εφαρμόζοντας τη συνάρτηση ενεργοποίησης στα αποτελέσματα των βαρών και της προκατάληψης. Οι συναρτήσεις ενεργοποίησης που υποστηρίζονται περιλαμβάνουν ReLU, tanh, και sigmoid. Παρέχει επίσης τη μέθοδο parameters για την εξαγωγή όλων των παραμέτρων του νευρώνα, δηλαδή των βαρών και της προκατάληψης του. Ο κώδικας υλοποίησής της είναι ο ακόλουθος:

```
import Value from "./Value";

import seedrandom from 'seedrandom';
let rng = seedrandom('puckpanicbutton'); // Replace 'your-seed-here' with your seed

function randomUniform(min, max, isSeeded) {
  min = min == null ? 0 : +min;
  max = max == null ? 1 : +max;
  if (arguments.length === 1) max = min, min = 0;
  else max -= min;
  return (isSeeded ? rng() : Math.random()) * max + min; // Use rng instead of Math.random
}

You, 5 months ago | 2 authors (ICantChooseAName and one other)
class Neuron {
  constructor(numOfInputs, activation) {
    this.weights = [...Array(numOfInputs)].map(e => new Value({ data: randomUniform(-1, 1, false) }));
    this.bias = new Value({ data: randomUniform(-1, 1, false) });
    this.activation = activation;
  }

  call(x) {
    let act = this.bias;

    for (let i = 0; i < this.weights.length; i++)
      act = act.add(this.weights[i].mul(x[i]));

    switch (this.activation) {
      case "tanh": return act.tanh();
      case "relu": return act.relu();
      case "sigmoid": return act.sigmoid();
      default: return act.sigmoid();
    }
  }

  parameters() {
    return [...this.weights, this.bias];
  }

  static fromJson(json, minified) {
    let neuron = new Neuron(0);
    if (minified) {
      neuron.weights = json.weights;
      neuron.bias = json.bias;
      neuron.activation = json.activation;
    }
    else {
      neuron.weights = json.weights.map(data => new Value({ data }));
      neuron.bias = new Value({ data: json.bias });
      neuron.activation = json.activation;
    }
    return neuron;
  }
}
```

Εικόνα 64: Κλάση Neuron

4.5.1.4 Η κλάση Layer

Η κλάση Layer αντιπροσωπεύει ένα επίπεδο (layer) νευρώνων σε ένα νευρωνικό δίκτυο. Αποτελείται από έναν πίνακα νευρώνων (neurons), όπου κάθε νευρώνας έχει τη δική του αρχικοποιημένη σειρά βαρών και προκατάληψη. Η μέθοδος call επιστρέφει την έξοδο του επιπέδου, καλώντας τη μέθοδο call του κάθε νευρώνα και υπολογίζοντας τις εξόδους για όλα τα εισερχόμενα δεδομένα. Η κλάση παρέχει επίσης τη μέθοδο parameters, που επιστρέφει τα βάρη και τις προκαταλήψεις όλων των νευρώνων του επιπέδου. Ο κώδικας υλοποίησής της είναι ο ακόλουθος:

```
class Layer {
  constructor(numOfInputs, numOfOutputs, activation) {
    this.neurons = [...Array(numOfOutputs)].map(e => new Neuron(numOfInputs, activation));
  }

  call(x) {
    const outputs = [];

    for (const neuron of this.neurons)
      outputs.push(neuron.call(x))
    return outputs.length === 1 ? outputs[0] : outputs;
    //return outputs;
  }

  parameters() {
    const params = [];
    for (const neuron of this.neurons)
      params.push(...neuron.parameters());
    return params;
  }

  static fromJson(json, minified) {
    const layer = new Layer(0, 0);
    layer.neurons = json.neurons.map(neuronJson => Neuron.fromJson(neuronJson, minified));
    return layer;
  }
}
```

Εικόνα 65: Κλάση Layer

4.5.1.5 Η κλάση Network

Η κλάση Network αντιπροσωπεύει το νευρωνικό δίκτυο ως σύνολο επιπέδων (layers). Δημιουργείται παίρνοντας ως είσοδο τον αριθμό των εισόδων (numOfInputs) και μια λίστα με πληροφορίες για τα κρυφά επίπεδα (hiddenLayers), που περιλαμβάνει τον αριθμό των νευρώνων και τη συνάρτηση ενεργοποίησης για κάθε επίπεδο. Κατά την αρχικοποίηση, η κλάση δημιουργεί μια σειρά από αντικείμενα τύπου Layer, κάθε ένα από τα οποία αποτελείται από πολλούς νευρώνες που αντιστοιχούν στο εκάστοτε επίπεδο.

Η μέθοδος call λαμβάνει ένα σύνολο εισόδων και τις προωθεί μέσα από τα διαδοχικά επίπεδα του δικτύου, επιστρέφοντας το τελικό αποτέλεσμα. Αυτή η διαδικασία είναι η λεγόμενη "προώθηση" (forward pass), όπου οι είσοδοι μετασχηματίζονται σταδιακά μέσα από το δίκτυο.

Η μέθοδος parameters επιστρέφει τα βάρη και τις προκαταλήψεις όλων των επιπέδων, που είναι τα παραμετροποιήσιμα στοιχεία του δικτύου τα οποία εκπαιδεύονται. Παρέχεται επίσης η στατική μέθοδος fromJson, η οποία μπορεί να ανακατασκευάσει ένα αποθηκευμένο νευρωνικό δίκτυο από δομή JSON, χρήσιμη για τη φόρτωση προεκπαιδευμένων μοντέλων. Ο κώδικας υλοποίησής της είναι ο ακόλουθος:

```
class Network {
  constructor(numOfInputs, hiddenLayers) {
    const result = hiddenLayers.reduce((acc, layer) => {
      acc.layersOutputs.push(layer.numOfNeurons);
      acc.layersActivations.push(layer.activation);
      return acc;
    }, { layersOutputs: [], layersActivations: [] });

    const inputsOutputs = [numOfInputs, ...result.layersOutputs];
    this.numOfInputs = numOfInputs
    this.layers = [...Array(hiddenLayers.length)].map((_, index) => new Layer(inputsOutputs[index], inputsOutputs[index + 1], result.layersActivations[index]));
  }

  call(x) {
    for (const layer of this.layers)
      x = layer.call(x);
    return x;
  }

  parameters() {
    const params = [];
    for (const layer of this.layers)
      params.push(...layer.parameters());
    return params;
  }

  static fromJson(json, minified) {
    if (minified === void 0) { minified = false; }
    const network = new Network(json.numOfInputs, []);
    network.layers = json.layers.map(layerJson => Layer.fromJson(layerJson, minified));
    return network;
  }
}
```

Εικόνα 66: Κλάση Network

4.5.2 Εκπαίδευση Νευρωνικού Δικτύου

4.5.2.1 Προεπεξεργασία συνόλων δεδομένων – δημιουργία ενιαίου συνόλου

Προκειμένου να εκπαιδευτεί το Νευρωνικό Δίκτυο, όπως αναλύθηκε στο Κεφάλαιο 3, χρησιμοποιήθηκαν κυρίως 2 δημόσια σύνολα δεδομένων (datasets) καθώς και μικρός αριθμός προσωπικών μετρήσεων. Προκειμένου να δημιουργηθεί ένα dataset κατάλληλο για την εκπαίδευση ενός νευρωνικού δικτύου, που θα ανιχνεύει πτώσεις βάσει δεδομένων επιταχυνσιομέτρων, χρειάζεται τα δεδομένα να μετασχηματιστούν και να συνδυαστούν σε μία ενιαία επιθυμητή μορφή. Απαιτείται, δηλαδή, να εκτελεστεί κατάλληλη προεπεξεργασία η προεπεξεργασία των δεδομένων που προέρχονται από τα δύο datasets, Kaggle και Fenix. Η μορφή που επιλέχθηκε είναι η δημιουργία ν-άδων (n-tuples) τιμών επιτάχυνσης για διαδοχικές χρονικές στιγμές, όπως η παρακάτω:

```
label,AccelerationX_1,AccelerationY_1,AccelerationZ_1,sv_1,AccelerationX_2,AccelerationY_2,AccelerationZ_2,sv_2
1,0.49432373046875,-0.59625244140625,-1.48504638671875,1.674883785548678,0.61328125,-0.4935150146484375,-1.3969573974609375,1.6034839979263422
1,-0.536712646484375,0.844329833984375,-0.2422027587890625,1.029376272223805,-0.67474365234375,0.8476715087890625,-0.0696563720703125,1.0856693757129317
1,-0.3080291748046875,-0.0099945068359375,0.9351959228515624,0.9846691204744544,-0.3075714111328125,-0.00994873046875,0.9336395263671876,0.9830472599927818
1,-0.4248809814453125,-0.9480133056640624,0.0766143798828125,1.041692295889176,0.0746917724609375,0.4705047607421875,0.009674072265625,0.4764946782794327
```

Εικόνα 67: Τελική μορφή δεδομένων εκπαίδευσης και αξιολόγησης

Σε αυτή τη μορφή περιέχονται οι τιμές επιτάχυνσης στους τρεις άξονες x, y, z για ν διαδοχικές χρονικές στιγμές (στο παρόν παράδειγμα έχουν δημιουργηθεί δυάδες), μαζί με τον συνολικό διανυσματικό συντελεστή (sv), ο οποίος υπολογίζεται με τον τύπο:

$$SV_{total} = \sqrt{A_x^2 + A_y^2 + A_z^2}$$

Τέλος, περιέχεται μία ετικέτα χαρακτηρισμού, όπου η τιμή 1 υποδεικνύει πτώση, ενώ η τιμή 0 υποδεικνύει μη πτώση. Η προσέγγιση αυτή επιτρέπει στο μοντέλο να αναγνωρίζει τις πτώσεις με βάση διαδοχικές αλλαγές στις τιμές επιτάχυνσης.

Οι αρχικές μορφές των δεδομένων διαφέρουν ανάλογα με την πηγή, όπως θα αναλυθεί ευθύς αμέσως.

1) Σύνολο Δεδομένων Kaggle

Το Kaggle dataset περιλαμβάνει δεδομένα στην παρακάτω μορφή:

```
Date;Timestamp;DeviceOrientation;AccelerationX;AccelerationY;AccelerationZ;Label
2020-07-14 14:55:48;1594763748.773793;portrait;-0.425506591796875;0.62469482421875;-0.6300506591796875;
2020-07-14 14:55:48;1594763748.778411;portrait;-0.4248809814453125;0.626556396484375;-0.5794830322265625;
```

Εικόνα 68: Μορφή δεδομένων του Kaggle dataset

Όπως φαίνεται στην παραπάνω εικόνα περιλαμβάνονται πληροφορίες για τον προσανατολισμό της συσκευής, τις τρεις συνιστώσες επιτάχυνσης (X, Y, Z), την ημερομηνία και τη χρονοσφραγίδα (timestamp) λήψης της εκάστοτε τιμής.

2) Σύνολο Δεδομένων Fenix και προσωπικές μετρήσεις

Το Fenix dataset και οι προσωπικές μετρήσεις έχουν την ίδια μορφή. Πρόκειται για τη μορφή η οποία φαίνεται παρακάτω:

```
-16,1.082942679,-0.014738726,1.08108558,-0.061657005
15,1.063182692,-0.020142926,1.060451363,-0.073447986
31,1.025346899,-0.024073253,1.02237632,-0.074184923
62,0.994284199,-0.023827608,0.991424994,-0.071482823
```

Εικόνα 69: Μορφή δεδομένων του Fenix dataset και των προσωπικών μετρήσεων

Εδώ έχουμε τις τιμές επιτάχυνσης για κάθε άξονα, το συνολικό διανυσματικό συντελεστή και το χρόνο σε milliseconds από την αρχή λήψης μετρήσεων.

Σε όλες τις παραπάνω μορφές οι επιταχύνσεις είναι εκφρασμένες σε μονάδες βαρύτητας (g).

Ο κώδικας που εκτελεί την κατάλληλη προεπεξεργασία για τα σύνολα δεδομένων και τα αποθηκεύει σε ένα τελικό αρχείο με την επιθυμητή μορφή ν-άδων είναι ο ακόλουθος:

```python
import pandas as pd    Import "pandas" could not be resolved from source
import numpy as np    Import "numpy" could not be resolved
import os
import inquirer    Import "inquirer" could not be resolved

def find_closest_indices(n, df, start_index, interval, timestamp_col='Timestamp'):
    indices = [start_index]
    current_timestamp = df.at[start_index, timestamp_col]
    for _ in range(1, n):
        next_timestamps = df.loc[start_index+1:, timestamp_col]
        try:
            closest_index = next_timestamps.sub(current_timestamp + interval).abs().idxmin()
        except Exception as e:
            return indices

        indices.append(closest_index)
        current_timestamp = df.at[closest_index, timestamp_col]
        start_index = closest_index
    return indices

def process_file(n, file_path, names, separator, header, dataset, interval):
    global total_skipped
    # Read the CSV file into a DataFrame
    df = pd.read_csv(file_path, sep=separator, header=header, names = names if dataset == 'fenix' else None)

    if dataset == 'kaggle':
        df['sv'] = np.sqrt(df['AccelerationX']**2 + df['AccelerationY']**2 + df['AccelerationZ']**2)

    timestamp_col = names[0]
    # df[timestamp_col] = pd.to_numeric(df[timestamp_col], errors='coerce')

    # List to hold all sequences
    all_sequences = []

    # Loop through the DataFrame and generate sequences
    for i in range(len(df)):
        if i > len(df) - n:
            break
        closest_indices = find_closest_indices(n, df, i, interval=interval, timestamp_col=timestamp_col)
        # print(closest_indices)
        if len(closest_indices) < n:
            total_skipped += 1
            print(f'Skipped {i} because no match was found between the given bounds')
            continue
        sequence = df.iloc[closest_indices].loc[:, names[1:]].values.flatten()
        # print(sequence)
        all_sequences.append(sequence)

    # print(all_sequences[0])
    # Convert the list of sequences into a DataFrame
    columns = [f'{dim}_{j}' for j in range(1, n+1) for dim in names[1:]]
    sequences_df = pd.DataFrame(all_sequences, columns=columns)

    # Determine the label based on the file name
    label = 1 if 'fall' in os.path.basename(file_path).lower() else 0
    sequences_df.insert(0, 'label', label)

    return sequences_df

total_skipped = 0
total_matched = 0
```

Εικόνα 70: Προεπεξεργασία δεδομένων, μέρος α'

```
total_skipped = 0
total_matched = 0

n = int(input("Enter the number of Timestamps per Input: "))

questions = [
  inquirer.List('path',
                message="Which datasets do you want to process?",
                choices=['kaggle', 'fenix'],
            ),
]
dataset = inquirer.prompt(questions)['path']

# Your directory path where CSV files are located
directory_path = f"fall_detection_data/{dataset}_datasets"

separator, header, interval, column_names = (";", 0, 0.1, ["Timestamp", "AccelerationX", "AccelerationY", "AccelerationZ", "sv"]) if dataset == 'kaggle' else (',', None, 100, ['Timestamp', 'sv', 'AccelerationX', 'AccelerationY', 'AccelerationZ'])

# Initialize an empty DataFrame to concatenate all processed files
final_df = pd.DataFrame()

all_files = os.listdir(directory_path)

# Filter out non-CSV files
csv_files = [f for f in all_files if f.endswith('.csv')]

# Create a list to hold the dataframes
df_list = []

# Iterate over files in the given directory
for csv in csv_files:  # Check if the file is a CSV
    file_path = os.path.join(directory_path, csv)
    print(csv)
    processed_df = process_file(n, file_path, names=column_names, separator=separator, header=header, dataset=dataset, interval = interval)
    total_matched += len(processed_df)
    df_list.append(processed_df)

print(f"Skipped {total_skipped}, Matched {total_matched} rows in total! Will create the final file...\n")
# Concatenate all data into one DataFrame
big_df = pd.concat(df_list, ignore_index=True)
big_df.to_csv(f"fall_detection_data/outputs/{dataset}-output.csv", index = False)
```

Εικόνα 71: Προεπεξεργασία δεδομένων, μέρος β'

Η ερμηνεία του είναι η εξής:

- find_closest_indices: Αυτή η συνάρτηση υπολογίζει τα πλησιέστερα χρονικά σημεία (indices) που πληρούν την προϋπόθεση της απόστασης μεταξύ χρονικών στιγμών και χρησιμοποιούνται για τη δημιουργία ν-άδων επιταχύνσεων.
- process_file: Αυτή είναι η βασική συνάρτηση που εκτελεί την επεξεργασία κάθε αρχείου του dataset. Ανάλογα με το επιλεγμένο dataset, το αρχείο διαβάζεται σε μορφή CSV και προετοιμάζονται οι επιθυμητές ν-άδες δεδομένων για κάθε επιταχυνόμενο χρονικό διάστημα. Για το Kaggle, υπολογίζεται ο sv για κάθε εγγραφή, ενώ για το Fenix δεν απαιτείται αυτός ο υπολογισμός, καθώς περιέχεται ήδη.
- Labeling: Το label κάθε εγγραφής (0 ή 1) καθορίζεται από το αν το όνομα του αρχείου περιέχει τη λέξη fall. Εάν το αρχείο αφορά πτώση, το label είναι 1, διαφορετικά είναι 0.
- Μεταβλητές: Η μεταβλητή n καθορίζει πόσα timestamps θα περιλαμβάνει κάθε ακολουθία, δηλαδή καθορίζει το μέγεθος των ν-άδων. Οι μεταβλητές separator, header, interval, column_names: Καθορίζουν τις ιδιότητες ανάγνωσης των αρχείων (π.χ., διαχωριστής και τίτλοι στηλών) και το χρονικό διάστημα μεταξύ των μετρήσεων.
- Επεξεργασία όλων των αρχείων: Το πρόγραμμα κάνει επεξεργασία όλων των αρχείων CSV στον κατάλληλο φάκελο (ανάλογα με το αν επεξεργάζεσαι τα δεδομένα του Kaggle ή του Fenix). Για κάθε αρχείο, δημιουργεί sequences και τα προσθέτει σε ένα ενιαίο DataFrame.
- Αποθήκευση δεδομένων: Στο τέλος, όλα τα sequences συνδυάζονται σε ένα ενιαίο DataFrame, το οποίο αποθηκεύεται σε ένα νέο αρχείο CSV στον φάκελο /fall_detection_data/outputs.

Τέλος παρατίθεται ο κώδικας ο οποίος συγκεντρώνει όλα τα αρχεία CSV από τον φάκελο "fall_detection_data/outputs" και τα συγχωνεύει σε ένα ενιαίο αρχείο δεδομένων:

```python
# replace with your folder's path
folder_path = "fall_detection_data/outputs"

all_files = os.listdir(folder_path)

# Filter out non-CSV files
csv_files = [f for f in all_files if f.endswith('output.csv')]

# Create a list to hold the dataframes
df_list = []

for csv in csv_files:
    file_path = os.path.join(folder_path, csv)
    try:
        # Try reading the file using default UTF-8 encoding
        df = pd.read_csv(file_path)
        df_list.append(df)
    except UnicodeDecodeError:
        try:
            # If UTF-8 fails, try reading the file using UTF-16 encoding with tab separator
            df = pd.read_csv(file_path, sep='\t', encoding='utf-16')
            df_list.append(df)
        except Exception as e:
            print(f"Could not read file {csv} because of error: {e}")
    except Exception as e:
        print(f"Could not read file {csv} because of error: {e}")

# Concatenate all data into one DataFrame
big_df = pd.concat(df_list, ignore_index=True)

shuffled_df = big_df.sample(frac=1)

# Save the final result to a new CSV file
big_df.to_csv(os.path.join(folder_path, 'all_data.csv'), index=False)

shuffled_df.to_csv(os.path.join(folder_path, 'all_data_shuffled.csv'), index=False)
```

Εικόνα 72: Κώδικας δημιουργίας ενιαίου συνόλου δεδομένων

Αρχικά, φιλτράρει τα αρχεία CSV και προσπαθεί να τα διαβάσει με κωδικοποίηση UTF-8 ή UTF-16 αν προκύψει σφάλμα. Στη συνέχεια, τα δεδομένα από όλα τα αρχεία συγχωνεύονται σε ένα DataFrame. Αφού τα δεδομένα συγχωνευτούν, ανακατεύονται τυχαία (random shuffling) με τη μέθοδο sample(). Τέλος, τα δεδομένα αποθηκεύονται σε δύο αρχεία: ένα ανακατεμένο και ένα χωρίς ανακάτεμμα. Η ανάμειξη των δεδομένων (shuffling) είναι σημαντική για να διασφαλιστεί ότι τα δεδομένα που θα χρησιμοποιηθούν κατά την εκπαίδευση του μοντέλου είναι τυχαία κατανεμημένα. Αυτό αποτρέπει το μοντέλο από το να μάθει τυχόν ανεπιθύμητα μοτίβα που θα προέκυπταν αν τα δεδομένα είχαν κάποια συγκεκριμένη ακολουθία (π.χ. συνεχόμενες πτώσεις). Το ανακάτεμα βοηθά στη βελτίωση της γενίκευσης του μοντέλου, δηλαδή της ικανότητάς του να αποδίδει καλά σε νέα, άγνωστα δεδομένα.

4.5.2.2 Προετοιμασία δεδομένων

Αρχικά, τα δεδομένα χωρίζονται σε τρία σύνολα: εκπαίδευσης (training set), επικύρωσης (validation set), και δοκιμής (test set). Η διαδικασία αυτή διασφαλίζει ότι τα δεδομένα που χρησιμοποιούνται κατά την εκπαίδευση είναι διαφορετικά από αυτά που θα χρησιμοποιηθούν για την αξιολόγηση του μοντέλου, προκειμένου να μετρηθεί η ικανότητα γενίκευσης του. Ο κώδικας που εκτελεί το διαχωρισμό είναι ο εξής:

```python
# Step 1: Load your data
data = pd.read_csv('fall_detection_data/outputs/all_data_shuffled.csv')

# Step 2: Split data into training and a temporary set (e.g., 70% training, 30% temp)
train_set, temp_set = train_test_split(data, test_size=0.3, random_state=42)

# Step 3: Split the temporary set into validation and test sets (e.g., 15% validation, 15% test)
validation_set, test_set = train_test_split(temp_set, test_size=0.5, random_state=42)

# Save the training set to a CSV file
train_set.to_csv('fall_detection_data/outputs/train_set.csv', index=False)

# Save the validation set to a CSV file
validation_set.to_csv('fall_detection_data/outputs/validation_set.csv', index=False)

# Save the test set to a CSV file
test_set.to_csv('fall_detection_data/outputs/test_set.csv', index=False)
```

Εικόνα 73: Διαχωρισμός δεδομένων σε train, validation και test σύνολα

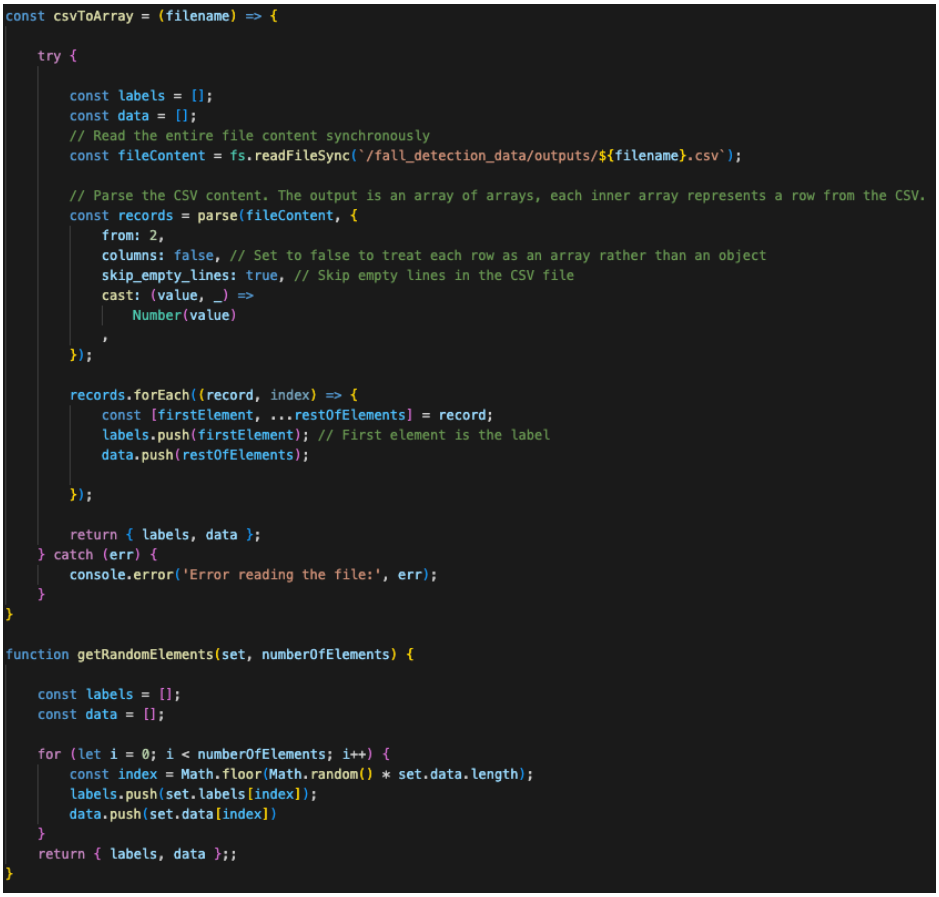
Το αρχικό dataset φορτώνεται από ένα CSV αρχείο που περιέχει τα δεδομένα των επιταχύνσεων και των γεγονότων πτώσης ή πανικού, και στη συνέχεια διαχωρίζεται σε training και temporary set (30% του αρχικού dataset). Το temporary set διαχωρίζεται περαιτέρω σε validation και test set με ίσο ποσοστό (15% έκαστο). Τα τρία αυτά σύνολα αποθηκεύονται σε ξεχωριστά CSV αρχεία και χρησιμοποιούνται για την εκπαίδευση και την αξιολόγηση του μοντέλου, όπως θα γίνει εμφανές στα επόμενα.

4.5.2.3 Διαδικασία εκπαίδευσης

Ο κώδικας εκπαίδευσης του μοντέλου είναι ο παρακάτω:

```javascript
export const binaryCrossEntropy = (yTrue, yPred) =>
    yPred
        .map((prediction, index) => {
            if (prediction.data == yTrue[index]) return new Value({ data: 0 })
            else {
                const caseA = (prediction.log()).mul(yTrue[index])
                const caseB = ((prediction.mul(-1)).add(1).log()).mul(1 - yTrue[index])

                //console.log(`Prediction ${prediction}, truth ${yTrue[index]}, case A ${caseA.data} + case B ${caseB.data}\n`)
                return (caseA.add(caseB)).mul(-1)
            }
        })
        .reduce((partialSum, currentValue) => currentValue.add(partialSum), 0).div(yTrue.length)

export const meanSquare = (yTrue, yPred) => yPred
    .map((prediction, index) => (prediction.sub(yTrue[index])).power(2))
    .reduce((partialSum, currentValue) => currentValue.add(partialSum), 0)
```

Εικόνα 74: Συναρτήσεις απωλειών



```
const csvToArray = (filename) => {

    try {

        const labels = [];
        const data = [];
        // Read the entire file content synchronously
        const fileContent = fs.readFileSync(`/fall_detection_data/outputs/${filename}.csv`);

        // Parse the CSV content. The output is an array of arrays, each inner array represents a row from the CSV.
        const records = parse(fileContent, {
            from: 2,
            columns: false, // Set to false to treat each row as an array rather than an object
            skip_empty_lines: true, // Skip empty lines in the CSV file
            cast: (value, _) =>
                Number(value)
            ,
        });

        records.forEach((record, index) => {
            const [firstElement, ...restOfElements] = record;
            labels.push(firstElement); // First element is the label
            data.push(restOfElements);

        });

        return { labels, data };
    } catch (err) {
        console.error('Error reading the file:', err);
    }
}

function getRandomElements(set, numberOfElements) {

    const labels = [];
    const data = [];

    for (let i = 0; i < numberOfElements; i++) {
        const index = Math.floor(Math.random() * set.data.length);
        labels.push(set.labels[index]);
        data.push(set.data[index])
    }
    return { labels, data };;
}
```

Εικόνα 75: Βοηθητικές συναρτήσεις εκπαίδευσης

```
function trainNetwork(validation) {

    console.log("Reading training set...")

    const train = csvToArray('train_set')

    const n = new Network(numOfInputs, hiddenLayers)

    fs.writeFileSync('initial_network.json', JSON.stringify(n, null, 2), 'utf8');

    let ypred

    let a = alpha;

    //gradient descent
    for (let k = 0; k < epochs; k++) {

        const { data: xs, labels: ys } = getRandomElements(train, batchSize);

        // forward pass
        ypred = xs.map(input => n.call(input))

        const meanSquareLoss = meanSquare(ys, ypred)

        const logLoss = binaryCrossEntropy(ys, ypred);

        for (const p of n.parameters())
            p.grad = 0.0
        // backward pass
        logLoss.fullBackward()

        // update
        for (const p of n.parameters())
            p.data += -a * p.grad

        if (!(k % 100) || k == epochs - 1) {

            xs = validation.data.slice(0, 12000);
            ys = validation.labels.slice(0, 12000);

            ypred = xs.map(input => n.call(input))

            const validationMSE = meanSquare(ys, ypred)

            const validationLogLoss = binaryCrossEntropy(ys, ypred);

            if (validationLogLoss.data < 0.233 && a > 0.5) {
                a = 0.5
                console.log(`decreased a to ${a} because validationLogloss = ${validationLogLoss.data}`)
            }
            if (validationLogLoss.data < 0.23 && a > 0.1) {
                a = 0.1
                console.log(`decreased a to ${a} because validationLogloss = ${validationLogLoss.data}`)
            }

            console.log(`Log Loss ${k} ${logLoss.data}`)

            console.log(`Mean Square Loss ${k} ${meanSquareLoss.data}\n`)

            console.log(`Validation set ${validationMSE.data} ${validationLogLoss.data}\n`)
        }
    }

    console.log(`Number of Parameters: ${n.parameters().length}`)

    fs.writeFileSync('trained_network.json', JSON.stringify(n, null, 2), 'utf8');

    console.log(ypred)
```

Εικόνα 76: Κώδικας εκπαίδευσης νευρωνικού δικτύου

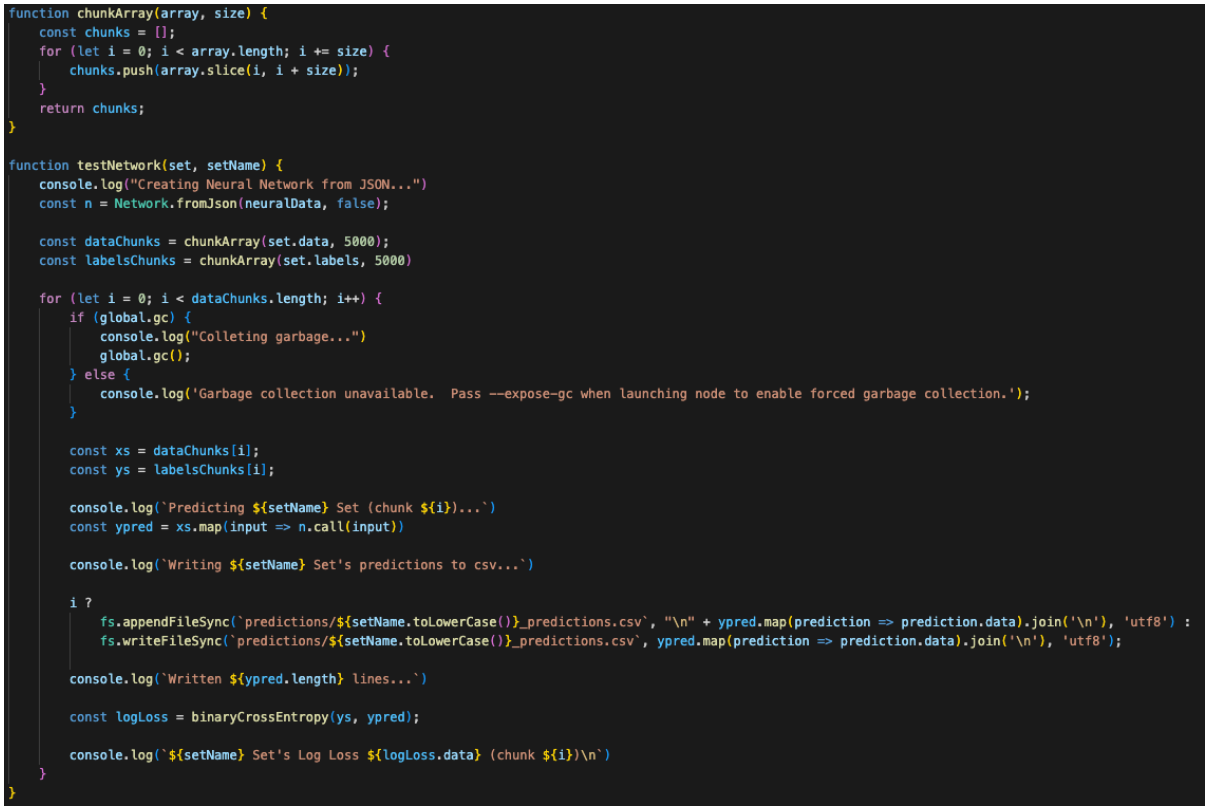
Η διαδικασία εκπαίδευσης ακολουθεί τα παρακάτω βήματα:

- Φόρτωση και αρχικοποίηση του δικτύου: Η κλάση Network αρχικοποιείται με τον ορισμό των επιπέδων (layers) και των νευρώνων (neurons). Αρχικοποιούνται τυχαία τα βάρη των νευρώνων και τα bias. Το αρχικό μοντέλο αποθηκεύεται σε JSON αρχείο.
- Gradient Descent: Χρησιμοποιείται η τεχνική του gradient descent για τη βελτιστοποίηση των παραμέτρων του δικτύου. Κάθε παρτίδα (batch) δεδομένων (μια μικρή ομάδα από το σύνολο εκπαίδευσης) προωθείται στο δίκτυο και υπολογίζεται η έξοδος του δικτύου. Στη συνέχεια, χρησιμοποιούνται οι συναρτήσεις απώλειας (binary cross entropy και mean square error) για να μετρηθεί η απόκλιση της πρόβλεψης του δικτύου από τα πραγματικά δεδομένα (labels).
- Backward Pass & Ενημέρωση Παραμέτρων: Το δίκτυο υπολογίζει τις παραγώγους (gradients) για κάθε μία από τις παραμέτρους του (βάρη και bias) με τη διαδικασία του backward propagation. Αυτές οι παράγωγοι χρησιμοποιούνται για την ενημέρωση των παραμέτρων με βάση το ρυθμό εκμάθησης (alpha). Με αυτό τον τρόπο, το δίκτυο "μαθαίνει" να βελτιώνει τις προβλέψεις του, μειώνοντας την απώλεια.
- Επικύρωση και Ρύθμιση Ρυθμού Εκμάθησης: Σε τακτά διαστήματα, το μοντέλο αξιολογείται με το σύνολο επικύρωσης (validation set), για να διαπιστωθεί αν υπερεκπαιδεύεται (overfitting) ή αν χρειάζεται προσαρμογές. Ο ρυθμός εκμάθησης μειώνεται ανάλογα με την απόδοση του δικτύου. Αν η απώλεια στο validation set πέσει κάτω από ορισμένα όρια, ο ρυθμός εκμάθησης μειώνεται για να αποφευχθεί η υπερεκπαίδευση.
- Αποθήκευση του Τελικού Μοντέλου: Αφού ολοκληρωθεί η εκπαίδευση, το τελικό μοντέλο (με τα βελτιστοποιημένα βάρη) αποθηκεύεται σε ένα αρχείο JSON, το οποίο μπορεί να χρησιμοποιηθεί για να κάνει προβλέψεις σε νέα δεδομένα.

4.5.3 Αξιολόγηση Νευρωνικού Δικτύου

4.5.3.1 Εξαγωγή προβλέψεων για το test set

Αυτή η φάση επικεντρώνεται στην πρόβλεψη των δεδομένων του συνόλου δοκιμής (test set) και την αποθήκευση των προβλέψεων, με χρήση του παρακάτω κώδικα:



```
function chunkArray(array, size) {
    const chunks = [];
    for (let i = 0; i < array.length; i += size) {
        chunks.push(array.slice(i, i + size));
    }
    return chunks;
}

function testNetwork(set, setName) {
    console.log("Creating Neural Network from JSON...")
    const n = Network.fromJson(neuralData, false);

    const dataChunks = chunkArray(set.data, 5000);
    const labelsChunks = chunkArray(set.labels, 5000)

    for (let i = 0; i < dataChunks.length; i++) {
        if (global.gc) {
            console.log("Colleting garbage...")
            global.gc();
        } else {
            console.log('Garbage collection unavailable.  Pass --expose-gc when launching node to enable forced garbage collection.');
        }

        const xs = dataChunks[i];
        const ys = labelsChunks[i];

        console.log(`Predicting ${setName} Set (chunk ${i})...`)
        const ypred = xs.map(input => n.call(input))

        console.log(`Writing ${setName} Set's predictions to csv...`)

        i ?
            fs.appendFileSync(`predictions/${setName.toLowerCase()}_predictions.csv`, "\n" + ypred.map(prediction => prediction.data).join('\n'), 'utf8') :
            fs.writeFileSync(`predictions/${setName.toLowerCase()}_predictions.csv`, ypred.map(prediction => prediction.data).join('\n'), 'utf8');

        console.log(`Written ${ypred.length} lines...`)

        const logLoss = binaryCrossEntropy(ys, ypred);

        console.log(`${setName} Set's Log Loss ${logLoss.data} (chunk ${i})\n`)
    }
}
```

Εικόνα 77: Κώδικας εξαγωγής προβλέψεων του test set

Αρχικά, δημιουργείται ένα αντικείμενο Network από ένα αποθηκευμένο JSON που περιέχει τις πληροφορίες του εκπαιδευμένου μοντέλου. Στη συνέχεια, τα δεδομένα του test set χωρίζονται σε μικρότερα τμήματα (chunks) των 5000 παρατηρήσεων με τη συνάρτηση chunkArray, και για κάθε κομμάτι εκτελείται πρόβλεψη μέσω της μεθόδου call του νευρωνικού δικτύου. Αφού υπολογιστούν οι προβλέψεις, αυτές αποθηκεύονται σε ένα αρχείο CSV για περαιτέρω ανάλυση. Σε κάθε επανάληψη, γίνεται επίσης εκτίμηση του κόστους (log loss) για να παρακολουθείται η ακρίβεια του μοντέλου στις προβλέψεις του.

Το test set αποθηκεύει τις προβλέψεις σε chunks, έτσι ώστε να διαχειρίζεται μεγάλα datasets και να αποφύγει προβλήματα με τη μνήμη, εκτελώντας ρητά συλλογή σκουπιδιών (garbage collection) μεταξύ δύο διαδοχικών chunks.

4.5.3.2 Αξιολόγηση των προβλέψεων

Αυτή η φάση ασχολείται με την ανάλυση των προβλέψεων για την εκτίμηση της απόδοσης του μοντέλου και την εύρεση του βέλτιστου κατωφλίου (threshold) για τον διαχωρισμό των περιπτώσεων πτώσης από τις μη πτώσεις.

Η εύρεση του βέλτιστου κατωφλίου είναι ουσιαστικής σημασίας, καθώς το νευρωνικό δίκτυο δίνει ως έξοδο πιθανότητες, δηλαδή τιμές μεταξύ 0 και 1. Είναι αναγκαίο να οριστεί μια τιμή πάνω από την οποία θεωρείται ότι πρόκειται για πτώση. Η διαδικασία αυτή περιλαμβάνει την ανάλυση των false positives (ψευδώς θετικών) και

false negatives (ψευδώς αρνητικών) ώστε να βρεθεί το κατώφλι που μεγιστοποιεί το F1 score, δηλαδή τη βέλτιστη ισορροπία μεταξύ της ακρίβειας (precision) και της ανάκλησης (recall).

Με την εύρεση του βέλτιστου κατωφλιού, μπορούμε να επιλέξουμε την τιμή που ελαχιστοποιεί τις λανθασμένες προβλέψεις και παρέχει την καλύτερη δυνατή απόδοση για την ανίχνευση πτώσεων. Αυτές οι διαδικασίες υλοποιούνται με τον παρακάτω κώδικα:

```python
def categorize(row):
    if row['label'] == 1 and row['quantized_pred'] == 1:
        return 'truePositive'
    elif row['label'] == 1 and row['quantized_pred'] == 0:
        return 'falseNegative'
    elif row['label'] == 0 and row['quantized_pred'] == 0:
        return 'trueNegative'
    elif row['label'] == 0 and row['quantized_pred'] == 1:
        return 'falsePositive'

def calculate_false_negatives(df, threshold):
    df['quantized_pred'] = (df['pred'] >= threshold).astype(int)
    false_negatives = len(df[(df['label'] == 1) & (df['quantized_pred'] == 0)])
    return false_negatives

def calculate_f1_score(df, threshold):
    df['quantized_pred'] = (df['pred'] >= threshold).astype(int)
    TP = len(df[(df['label'] == 1) & (df['quantized_pred'] == 1)])
    FP = len(df[(df['label'] == 0) & (df['quantized_pred'] == 1)])
    FN = len(df[(df['label'] == 1) & (df['quantized_pred'] == 0)])

    if TP + FP == 0 or TP + FN == 0:  # Prevent division by zero
        return 0

    precision = TP / (TP + FP)
    recall = TP / (TP + FN)
    if precision + recall == 0:  # Prevent division by zero
        return 0

    f1_score = 2 * (precision * recall) / (precision + recall)
    return f1_score

def find_optimal_threshold(df):
    # Test various thresholds from 0.0 to 1.0 at intervals of 0.01
    thresholds = np.arange(0.2, 0.9, 0.01)
    false_negatives_list = [calculate_false_negatives(df.copy(), t) for t in thresholds]

    # Find the threshold with the minimum false negatives
    min_false_negatives = min(false_negatives_list)
    optimal_threshold = thresholds[false_negatives_list.index(min_false_negatives)]

    print(f"Optimal threshold (minimizing false negatives): {optimal_threshold}, with {min_false_negatives} false negatives\n")

    f1_scores = [calculate_f1_score(df.copy(), t) for t in thresholds]
    max_f1_score = max(f1_scores)
    optimal_threshold = thresholds[f1_scores.index(max_f1_score)]

    print(f"Optimal threshold (maximizing f1 score): {optimal_threshold} achieves {max_f1_score} F1-score\n")

def set_score(set_name):

    set = pd.read_csv(f'fall_detection_data/outputs/{set_name}_set.csv', usecols=['label'])

    predictions = pd.read_csv(f'neural_network/predictions/{set_name}_predictions.csv', header=None, names=['pred'])

    full_set = pd.concat([set, predictions], axis=1)

    find_optimal_threshold(full_set)

    full_set['quantized_pred'] = (full_set['pred'] >= 0.37).astype(int)

    # Apply the function to create a new column 'result'
    full_set['result'] = full_set.apply(categorize, axis=1)

    # Count occurrences of each result
    result_counts = full_set['result'].value_counts()

    print(result_counts)
```

Εικόνα 78: Αξιολόγηση μοντέλου και εύρεση βέλτιστου κατωφλίου

Ο κώδικας αυτός εκτελεί τις εξής διαδικασίες:

- Κατηγοριοποίηση: Οι προβλέψεις ταξινομούνται σε τέσσερις κατηγορίες: αληθώς θετικές (truePositive), ψευδώς θετικές (falsePositive), αληθώς αρνητικές (trueNegative), και ψευδώς αρνητικές (falseNegative), ανάλογα με το πόσο καλά το μοντέλο πρόβλεψε τα δεδομένα του test set.
- Υπολογισμός F1-Score: Το F1-Score είναι ένα μέτρο που χρησιμοποιείται για να αξιολογηθεί η ισορροπία μεταξύ της ακρίβειας (precision) και της ανάκλησης (recall) των προβλέψεων του μοντέλου. Για διαφορετικά κατώφλια, υπολογίζεται το F1-Score ώστε να βρεθεί το βέλτιστο κατώφλι που μεγιστοποιεί την απόδοση του μοντέλου.
- Εύρεση βέλτιστου κατωφλίου: Ο αλγόριθμος δοκιμάζει διάφορα κατώφλια (thresholds) για να εντοπίσει το κατώφλι που ελαχιστοποιεί τα ψευδώς αρνητικά ή μεγιστοποιεί το F1-Score, δίνοντας προτεραιότητα στην ακρίβεια των προβλέψεων. Το τελικό κατώφλι εφαρμόζεται στα δεδομένα και οι προβλέψεις κατηγοριοποιούνται.

4.5.3.3 Αποτελέσματα αξιολόγησης μοντέλου

Από τα μοντέλα που δοκιμάστηκαν επιλέχθηκαν τα παρακάτω που εμφάνισαν τα βέλτιστα αποτελέσματα:

1) 20-20_neurons_1_a_20000_epochs_relu

Το μοντέλο αυτό έχει την εξής δομή:

- Κρυφά Επίπεδα: 2 επίπεδα
- Νευρώνες στο 1ο επίπεδο: 20 (ReLU ενεργοποίηση)
- Νευρώνες στο 2ο επίπεδο: 20 (ReLU ενεργοποίηση)
- Νευρώνες στην έξοδο: 1 (Sigmoid ενεργοποίηση)
- Εποχές: 20.000
- Λάμδα (λ): 1
- Ρυθμός εκμάθησης (alpha): 1

Τα αποτελέσματά του είναι τα παρακάτω (το πρώτο τμήμα των αποτελεσμάτων αφορά τα σκορ για το validation set και το δεύτερο τμήμα αφορά τα σκορ για το test set):

```
Optimal threshold (minimizing false negatives): 0.2, with 618 false negatives

Optimal threshold (maximizing f1 score): 0.37000000000000016 achieves 0.7405452946350044 F1-score

result
truePositive     13893
trueNegative     10095
falsePositive     6270
falseNegative     3465
Name: count, dtype: int64


Optimal threshold (minimizing false negatives): 0.2, with 622 false negatives

Optimal threshold (maximizing f1 score): 0.3900000000000002 achieves 0.7375364151047106 F1-score

result
truePositive     13737
trueNegative     10155
falsePositive     6366
falseNegative     3466
Name: count, dtype: int64
```

Εικόνα 79: Σκορ του μοντέλου 20-20_neurons_1_a_20000_epochs_relu

2) 20-20_neurons_1_a_20000_epochs_tanh

Το μοντέλο αυτό έχει την εξής δομή:

- Κρυφά Επίπεδα: 2 επίπεδα
- Νευρώνες στο 1ο επίπεδο: 20 (Tanh ενεργοποίηση)
- Νευρώνες στο 2ο επίπεδο: 20 (Tanh ενεργοποίηση)
- Νευρώνες στην έξοδο: 1 (Sigmoid ενεργοποίηση)
- Εποχές: 20.000
- Λάμδα (λ): 1
- Ρυθμός εκμάθησης (alpha): 1

Τα αποτελέσματά του είναι τα παρακάτω:

```
Optimal threshold (minimizing false negatives): 0.2, with 842 false negatives

Optimal threshold (maximizing f1 score): 0.36000000000000015 achieves 0.7480310683830318 F1-score

result
truePositive     13946
trueNegative     10374
falsePositive     5991
falseNegative     3412
Name: count, dtype: int64


Optimal threshold (minimizing false negatives): 0.2, with 847 false negatives

Optimal threshold (maximizing f1 score): 0.35000000000000014 achieves 0.745601982812037 F1-score

result
truePositive     13838
trueNegative     10443
falsePositive     6078
falseNegative     3365
Name: count, dtype: int64
```

Εικόνα 80: Σκορ του μοντέλου 20-20_neurons_1_a_20000_epochs_tanh

3) 40_neurons_1_a_20000_epochs_relu

Το μοντέλο αυτό έχει την εξής δομή:

- Κρυφά Επίπεδα: 1 επίπεδο
- Νευρώνες στο 1ο επίπεδο: 40 (ReLU ενεργοποίηση)
- Νευρώνες στην έξοδο: 1 (Sigmoid ενεργοποίηση)
- Εποχές: 20.000
- Λάμδα (λ): 1
- Ρυθμός εκμάθησης (alpha): 1

Τα αποτελέσματά του είναι τα παρακάτω:

```
Optimal threshold (minimizing false negatives): 0.2, with 414 false negatives

Optimal threshold (maximizing f1 score): 0.4100000000000002 achieves 0.738923641627462 F1-score

result
truePositive     14293
trueNegative      9330
falsePositive     7035
falseNegative     3065
Name: count, dtype: int64


Optimal threshold (minimizing false negatives): 0.2, with 424 false negatives

Optimal threshold (maximizing f1 score): 0.4200000000000002 achieves 0.735737946264262 F1-score

result
truePositive     14154
trueNegative      9378
falsePositive     7143
falseNegative     3049
Name: count, dtype: int64
```

Εικόνα 81: Σκορ μοντέλου 40_neurons_1_a_20000_epochs_relu

4) 40_neurons_01_a_20000_epochs_relu

Το μοντέλο αυτό έχει την εξής δομή:

- Κρυφά Επίπεδα: 1 επίπεδο
- Νευρώνες στο 1ο επίπεδο: 40 (ReLU ενεργοποίηση)
- Νευρώνες στην έξοδο: 1 (Sigmoid ενεργοποίηση)
- Εποχές: 20.000
- Λάμδα (λ): 1
- Ρυθμός εκμάθησης (alpha): 0.1

Τα αποτελέσματά του είναι τα παρακάτω:

```
Optimal threshold (minimizing false negatives): 0.2, with 292 false negatives

Optimal threshold (maximizing f1 score): 0.36000000000000015 achieves 0.7289989366550206 F1-score

result
truePositive     14202
trueNegative      8927
falsePositive     7438
falseNegative     3156
Name: count, dtype: int64


Optimal threshold (minimizing false negatives): 0.2, with 297 false negatives

Optimal threshold (maximizing f1 score): 0.38000000000000017 achieves 0.7270397489539748 F1-score

result
truePositive     14072
trueNegative      9027
falsePositive     7494
falseNegative     3131
Name: count, dtype: int64
```

Εικόνα 82: Σκορ μοντέλου 40_neurons_01_a_20000_epochs_relu

5) 40_neurons_1_a_20000_epochs_tanh

Το μοντέλο αυτό έχει την εξής δομή:

- Κρυφά Επίπεδα: 1 επίπεδο
- Νευρώνες στο 1ο επίπεδο: 40 (Tanh ενεργοποίηση)
- Νευρώνες στην έξοδο: 1 (Sigmoid ενεργοποίηση)
- Εποχές: 20.000
- Λάμδα (λ): 1
- Ρυθμός εκμάθησης (alpha): 1

Τα αποτελέσματά του είναι τα παρακάτω:

```
Optimal threshold (minimizing false negatives): 0.2, with 1075 false negatives

Optimal threshold (maximizing f1 score): 0.3000000000000001 achieves 0.7430738438849678 F1-score

result
truePositive     14108
trueNegative      9859
falsePositive     6506
falseNegative     3250
Name: count, dtype: int64


Optimal threshold (minimizing false negatives): 0.2, with 1042 false negatives

Optimal threshold (maximizing f1 score): 0.3000000000000001 achieves 0.7406172251336616 F1-score

result
truePositive     13991
trueNegative      9933
falsePositive     6588
falseNegative     3212
Name: count, dtype: int64
```

Εικόνα 83: Σκορ μοντέλου 40_neurons_1_a_20000_epochs_tanh

6) 40_neurons_01_a_20000_epochs_tanh

Το μοντέλο αυτό έχει την εξής δομή:

- Κρυφά Επίπεδα: 1 επίπεδο
- Νευρώνες στο 1ο επίπεδο: 40 (Tanh ενεργοποίηση)
- Νευρώνες στην έξοδο: 1 (Sigmoid ενεργοποίηση)
- Εποχές: 20.000
- Λάμδα (λ): 1
- Ρυθμός εκμάθησης (alpha): 0.1

Τα αποτελέσματά του είναι τα παρακάτω:

```
Optimal threshold (minimizing false negatives): 0.2, with 329 false negatives

Optimal threshold (maximizing f1 score): 0.37000000000000016 achieves 0.7241057228687552 F1-score

result
truePositive     14342
trueNegative      8452
falsePositive     7913
falseNegative     3016
Name: count, dtype: int64


Optimal threshold (minimizing false negatives): 0.2, with 330 false negatives

Optimal threshold (maximizing f1 score): 0.45000000000000023 achieves 0.7237330209466433 F1-score

result
truePositive     14220
trueNegative      8545
falsePositive     7976
falseNegative     2983
Name: count, dtype: int64
```

Εικόνα 84: Σκορ μοντέλου 40_neurons_01_a_20000_epochs_tanh

7) 40_neurons_relu_optimized

Το μοντέλο αυτό έχει την εξής δομή:

- Κρυφά Επίπεδα: 1 επίπεδο
- Νευρώνες στο 1ο επίπεδο: 40 (ReLU ενεργοποίηση)
- Νευρώνες στην έξοδο: 1 (Sigmoid ενεργοποίηση)
- Εποχές: 20.000
- Λάμδα (λ): 1
- Ρυθμός εκμάθησης (alpha): Ξεκινά από 1 και μειώνεται στα 0.5 και ύστερα στα 0.1 με βάση το validation loss.

Τα αποτελέσματά του είναι τα παρακάτω:

```
Optimal threshold (minimizing false negatives): 0.2, with 486 false negatives

Optimal threshold (maximizing f1 score): 0.36000000000000015 achieves 0.7394673371991584 F1-score

result
truePositive     14410
trueNegative      9159
falsePositive     7206
falseNegative     2948
Name: count, dtype: int64


Optimal threshold (minimizing false negatives): 0.2, with 494 false negatives

Optimal threshold (maximizing f1 score): 0.38000000000000017 achieves 0.7381103360811668 F1-score

result
truePositive     14322
trueNegative      9207
falsePositive     7314
falseNegative     2881
Name: count, dtype: int64
```

Εικόνα 85: Σκορ μοντέλου 40_neurons_relu_optimized

8) 80_neurons_relu_optimized

Το μοντέλο αυτό έχει την εξής δομή:

- Κρυφά Επίπεδα: 1 επίπεδο
- Νευρώνες στο 1ο επίπεδο: 80 (ReLU ενεργοποίηση)
- Νευρώνες στην έξοδο: 1 (Sigmoid ενεργοποίηση)
- Εποχές: 20.000
- Λάμδα (λ): 1
- Ρυθμός εκμάθησης (alpha): Ξεκινά από 1 και μειώνεται στα 0.5 και ύστερα στα 0.1 με βάση το validation loss.

Τα αποτελέσματά του είναι τα παρακάτω:

```
Optimal threshold (minimizing false negatives): 0.2, with 454 false negatives

Optimal threshold (maximizing f1 score): 0.3900000000000002 achieves 0.7435185919289622 F1-score

result
truePositive     14438
trueNegative      9265
falsePositive     7100
falseNegative     2920
Name: count, dtype: int64


Optimal threshold (minimizing false negatives): 0.2, with 477 false negatives

Optimal threshold (maximizing f1 score): 0.37000000000000016 achieves 0.7394931935008912 F1-score

result
truePositive     14314
trueNegative      9325
falsePositive     7196
falseNegative     2889
Name: count, dtype: int64
```

Εικόνα 86: Σκορ μοντέλου 80_neurons_relu_optimized

Τα μοντέλα με 80 νευρώνες ήταν πιο απαιτητικά σε πόρους και χρησιμοποιήθηκαν για να αξιολογηθεί αν η αύξηση των νευρώνων βελτιώνει σημαντικά την ακρίβεια των προβλέψεων. Ωστόσο, το σύστημα έδειξε ότι τα 40 νευρώνες ήταν η ασφαλέστερη επιλογή για την τρέχουσα υλοποίηση στο Puck.js, ενώ τα 80 νευρώνες είχαν μικρή βελτίωση στην απόδοση, αλλά απαιτούσαν περισσότερη μνήμη RAM.

4.6 Συμπεράσματα και ιδέες για πιθανές βελτιώσεις και επεκτάσεις

4.6.1 Εισαγωγή

Μια πιθανή κατεύθυνση για τη βελτίωση του συστήματος ανίχνευσης πτώσεων και πανικού αφορά την ενσωμάτωση νέων τεχνολογιών και την επέκταση των δυνατοτήτων του συστήματος, με στόχο την αύξηση της ασφάλειας, της ακρίβειας και της ευχρηστίας. Η εφαρμογή μπορεί να επεκταθεί με νέες λειτουργίες και να ενσωματώσει επιπλέον αισθητήρες, να βελτιώσει την ασφάλεια των επικοινωνιών και να προσθέσει δυνατότητες σύνδεσης με περισσότερες συσκευές και υπηρεσίες. Στις επόμενες υποενότητες αναλύονται ορισμένες προτάσεις βελτίωσης που θα μπορούσαν να εφαρμοστούν στο μέλλον.

4.6.2 Ασφάλεια και διαδικασία ζευγοποίησης

Η προσθήκη ενός μηχανισμού ζευγοποίησης μεταξύ του κινητού και του Puck.js είναι σημαντική βελτίωση για την ασφάλεια, ώστε να αποτραπεί η ανεπιθύμητη πρόσβαση από μη εξουσιοδοτημένες συσκευές. Αυτό θα διασφαλίσει ότι μόνο εξουσιοδοτημένες συσκευές θα μπορούν να συνδέονται και να επικοινωνούν με το Puck. Επιπλέον, η επικοινωνία από Puck σε Puck θα μπορούσε να ενισχυθεί με μηχανισμούς ασφαλείας, όπως κρυπτογράφηση, ώστε να αποτρέπεται η παραβίαση των δεδομένων κατά τη μεταφορά.

4.6.3 Ενσωμάτωση περισσότερων αισθητήρων

Εκτός από το επιταχυνσιόμετρο, θα μπορούσαν να αξιοποιηθούν και άλλοι αισθητήρες όπως ο αισθητήρας θερμοκρασίας, το μαγνητόμετρο ή ο αισθητήρας φωτός. Αυτοί οι αισθητήρες θα μπορούσαν να ανιχνεύουν αλλαγές στις συνθήκες περιβάλλοντος και να ενεργοποιούν ενέργειες όπως η αποστολή ειδοποιήσεων σε εμπιστευμένες επαφές. Για παράδειγμα, αν αυξηθεί η θερμοκρασία σε επικίνδυνα επίπεδα, το Puck θα μπορούσε να ενεργοποιήσει τον κλιματισμό (air-condition) μέσω του IR πομπού του. Αντίστοιχα, το μαγνητόμετρο θα μπορούσε να χρησιμοποιηθεί για την παρακολούθηση μαγνητικών πεδίων και την αποστολή προειδοποιήσεων σε ασθενείς με βηματοδότη.

4.6.4 Ενσωμάτωση γυροσκοπίου για βελτιωμένες προβλέψεις

Το γυροσκόπιο, που μετράει την γωνιακή επιτάχυνση, θα μπορούσε να χρησιμοποιηθεί για να βελτιώσει την ακρίβεια των προβλέψεων του συστήματος σε συνδυασμό με τα δεδομένα του επιταχυνσιόμετρου, παρέχοντας πιο ακριβείς ενδείξεις για τις κινήσεις του σώματος και τις πτώσεις.

4.6.5 Hub για αποστολή SMS

Μια ενδιαφέρουσα πιθανότητα θα ήταν η χρήση ενός hub (κεντρικού σημείου) που θα μπορούσε να αποστέλλει SMS σε εμπιστευμένες επαφές χωρίς να απαιτείται σύνδεση στο διαδίκτυο. Αν και αυτό δεν είναι δυνατό από την ίδια την εφαρμογή στο iOS (καθώς απαιτείται πάντα ενέργεια από τον χρήστη για την αποστολή SMS), το hub θα μπορούσε να λειτουργεί αυτόνομα, προσφέροντας αυτή τη δυνατότητα ακόμα και χωρίς πρόσβαση στο ίντερνετ.

4.6.6 Σύνδεση με γιατρούς και υπηρεσίες έκτακτης ανάγκης

Μια άλλη πιθανή βελτίωση είναι η προσθήκη γιατρών ή υπηρεσιών έκτακτης ανάγκης που θα μπορούσαν να παρακολουθούν την κατάσταση του χρήστη και να ειδοποιούνται σε περίπτωση επείγοντος περιστατικού, όπως η πτώση, ώστε να παρέχουν άμεση βοήθεια.

4.6.7 Δοκιμή διαφορετικών συσκευών

Το σύστημα θα μπορούσε να επεκταθεί και σε άλλες συσκευές εκτός του Puck.js, όπως το Nordic Thingy:52 ή το ESP32 με Bluetooth, οι οποίες παρέχουν αντίστοιχες δυνατότητες ανίχνευσης πτώσεων και παρακολούθησης μέσω BLE, προσφέροντας παράλληλα πρόσθετες λειτουργίες και δυνατότητες σε διαφορετικά περιβάλλοντα χρήσης.

4.6.8 Περεταίρω συμπίεση του Νευρωνικού Δικτύου

Τέλος, ουσιαστική βελτίωση αποτελεί η περεταίρω συμπίεση της υλοποίησης του νευρωνικού δικτύου στην πλευρά του Puck.js. Επί του παρόντος, η υλοποίηση στο Puck παραλείπει εντελώς την κλάση Value για να εξοικονομήσει RAM, καθώς το Puck χρειάζεται μόνο την αριθμητική τιμή και όχι τα επιπλέον δεδομένα που παρέχει η κλάση (όπως τα διανύσματα gradient και ιδιαίτερα οι κόμβοι - παιδιά). Αυτή η βελτίωση επιτρέπει τη λειτουργία του συστήματος σε περιορισμένους πόρους, αλλά θα μπορούσε να βελτιστοποιηθεί περαιτέρω για να υποστηρίξει μεγαλύτερα και πιο ακριβή νευρωνικά δίκτυα στο Puck.js. Με τη μείωση της μνήμης που καταλαμβάνουν οι επιπλέον λειτουργίες, θα ήταν δυνατή η χρήση πιο πολύπλοκων μοντέλων, βελτιώνοντας έτσι την ακρίβεια των προβλέψεων του συστήματος.

Βιβλιογραφία

[1] World Health Organization (WHO), “Falls.” Accessed: Aug. 27, 2024. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/falls

[2] P. Kannus, H. Sievänen, M. Palvanen, T. Järvinen, and J. Parkkari, “falls_lancet,” *Lancet*, vol. 366, pp. 1885–1893, 2005.

[3] M. K. Karlsson, T. Vonschewelov, C. Karlsson, M. CÃster, and B. E. Rosengen, “Prevention of falls in the elderly: A review,” *Scand J Public Health*, vol. 41, no. 5, pp. 442–454, 2013, doi: 10.1177/1403494813483215.

[4] E. J. Bisson, E. W. Peterson, and M. Finlayson, “Delayed Initial Recovery and Long Lie After a Fall Among Middle-Aged and Older People With Multiple Sclerosis,” *Arch Phys Med Rehabil*, vol. 96, no. 8, pp. 1499–1505, Aug. 2015, doi: 10.1016/J.APMR.2015.04.012.

[5] J. Pope, M. Truesdale, and M. Brown, “Risk factors for falls among adults with intellectual disabilities: A narrative review,” *Journal of Applied Research in Intellectual Disabilities*, vol. 34, no. 1, pp. 274–285, Jan. 2021, doi: 10.1111/JAR.12805.

[6] J. Finlayson, J. Morrison, A. Jackson, D. Mantry, and S. A. Cooper, “Injuries, falls and accidents among adults with intellectual disabilities. Prospective cohort study,” *Journal of Intellectual Disability Research*, vol. 54, no. 11, pp. 966–980, Nov. 2010, doi: 10.1111/J.1365-2788.2010.01319.X.

[7] J. Kubitza, I. T. Schneider, and B. Reuschenbach, “Concept of the term long lie: a scoping review,” *European Review of Aging and Physical Activity*, vol. 20, no. 1, pp. 1–11, Dec. 2023, doi: 10.1186/S11556-023-00326-3/TABLES/3.

[8] J. Fleming and C. Brayne, “Inability to get up after falling, subsequent time on floor, and summoning help: prospective cohort study in people over 90,” *BMJ*, vol. 337, no. 7681, pp. 1279–1282, Nov. 2008, doi: 10.1136/BMJ.A2227.

[9] M. E. Tinetti, W. L. Liu, and E. B. Claus, “Predictors and Prognosis of Inability to Get Up After Falls Among Elderly Persons,” *JAMA*, vol. 269, no. 1, pp. 65–70, Jan. 1993, doi: 10.1001/JAMA.1993.03500010075035.

[10] J. Blackburn, K. Ousey, J. Stephenson, and S. Lui, “Exploring the impact of experiencing a long lie fall on physical and clinical outcomes in older people requiring an ambulance: A systematic review,” *Int Emerg Nurs*, vol. 62, p. 101148, May 2022, doi: 10.1016/J.IENJ.2022.101148.

[11] S. Usmani, A. Saboor, M. Haris, M. A. Khan, and H. Park, “Latest Research Trends in Fall Detection and Prevention Using Machine Learning: A Systematic Review,” *Sensors (Basel)*, vol. 21, no. 15, Aug. 2021, doi: 10.3390/S21155134.

[12] “nRF52832 - Versatile Bluetooth 5.2 SoC - nordicsemi.com.” Accessed: Oct. 14, 2024. [Online]. Available: https://www.nordicsemi.com/Products/nRF52832

[13] “Puck.js - Espruino.” Accessed: Oct. 14, 2024. [Online]. Available: https://www.espruino.com/Puck.js#tutorials

[14] “Espruino - JavaScript for Microcontrollers.” Accessed: Oct. 14, 2024. [Online]. Available: https://www.espruino.com/

[15] “Top Programming Languages 2024 - IEEE Spectrum.” Accessed: Aug. 29, 2024. [Online]. Available: https://spectrum.ieee.org/top-programming-languages-2024
[16] “Most used languages among software developers globally 2024 | Statista.” Accessed: Aug. 29, 2024. [Online]. Available: https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/
[17] “JavaScript | MDN.” Accessed: Oct. 14, 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript
[18] “Programming language community sizes worldwide 2023 | Statista.” Accessed: Aug. 31, 2024. [Online]. Available: https://www.statista.com/statistics/1241923/worldwide-software-developer-programming-language-communities/
[19] “Developer Nation Community.” Accessed: Aug. 31, 2024. [Online]. Available: https://www.developernation.net/blog/language-communities-who-leads-the-way/
[20] “TypeScript: JavaScript With Syntax For Types.” Accessed: Oct. 14, 2024. [Online]. Available: https://www.typescriptlang.org/
[21] “Node.js — Run JavaScript Everywhere.” Accessed: Oct. 14, 2024. [Online]. Available: https://nodejs.org/en
[22] “Express - Node.js web application framework.” Accessed: Oct. 14, 2024. [Online]. Available: https://expressjs.com/
[23] “Kysely | Kysely.” Accessed: Oct. 14, 2024. [Online]. Available: https://kysely.dev/
[24] “MariaDB Foundation - MariaDB.org.” Accessed: Oct. 14, 2024. [Online]. Available: https://mariadb.org/
[25] “Blockly | Google for Developers.” Accessed: Oct. 14, 2024. [Online]. Available: https://developers.google.com/blockly
[26] “Flutter - Build apps for any screen.” Accessed: Oct. 14, 2024. [Online]. Available: https://flutter.dev/
[27] “Dart programming language | Dart.” Accessed: Oct. 14, 2024. [Online]. Available: https://dart.dev/
[28] “Firebase | Google’s Mobile and Web App Development Platform.” Accessed: Oct. 14, 2024. [Online]. Available: https://firebase.google.com/
[29] “Firebase Authentication.” Accessed: Oct. 14, 2024. [Online]. Available: https://firebase.google.com/docs/auth
[30] “SendGrid Email API and Email Marketing Campaigns | SendGrid.” Accessed: Oct. 14, 2024. [Online]. Available: https://sendgrid.com/en-us
[31] “A Developer’s Guide to Bluetooth Technology | Bluetooth® Technology Website.” Accessed: Sep. 24, 2024. [Online]. Available: https://www.bluetooth.com/blog/a-developers-guide-to-bluetooth/
[32] “Welcome to Python.org.” Accessed: Oct. 14, 2024. [Online]. Available: https://www.python.org/
[33] “NumPy -.” Accessed: Oct. 15, 2024. [Online]. Available: https://numpy.org/
[34] “pandas - Python Data Analysis Library.” Accessed: Oct. 15, 2024. [Online]. Available: https://pandas.pydata.org/

[35] “Neural network (machine learning) - Wikipedia.” Accessed: Sep. 26, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Neural_network_(machine_learning)
[36] “What is a Neural Network? | IBM.” Accessed: Sep. 26, 2024. [Online]. Available: https://www.ibm.com/topics/neural-networks
[37] “fall detection accelerometer data.” Accessed: Oct. 12, 2024. [Online]. Available: https://www.kaggle.com/datasets/harnoor343/fall-detection-accelerometer-data
[38] “UR Fall Detection Dataset.” Accessed: Oct. 12, 2024. [Online]. Available: http://fenix.ur.edu.pl/~mkepski/ds/uf.html
[39] “HTTP response status codes - HTTP | MDN.” Accessed: Oct. 18, 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status