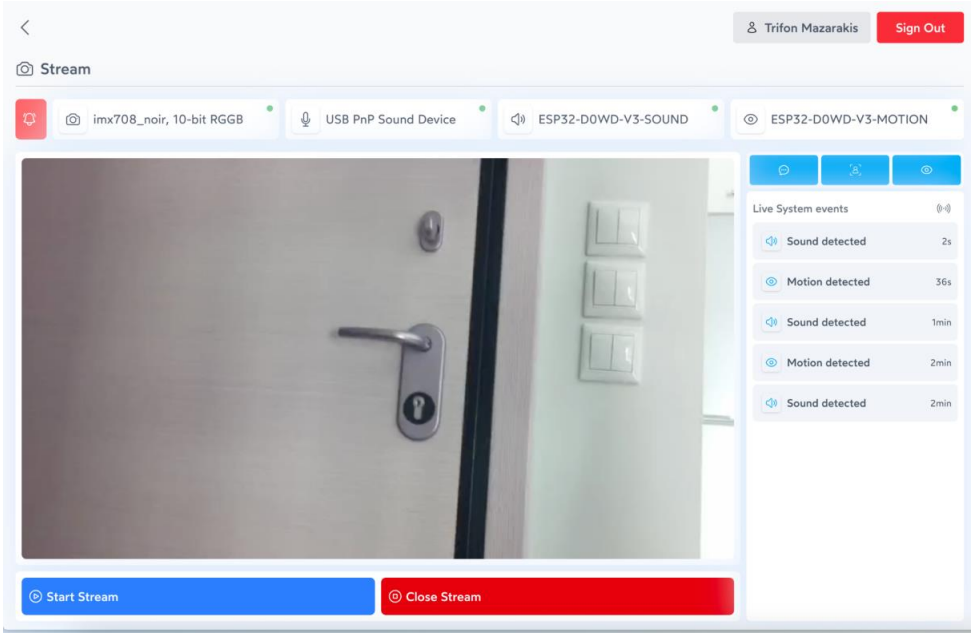
ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Έξυπνο σύστημα παρακολούθησης χώρου»



Του φοιτητή
Τρύφωνας Μαζαράκης
Αρ. Μητρώου: 2020085

Επιβλέπων
Κοκκώνης Γεώργιος
Επίκουρος Καθηγητής

28 Μαΐου 2025

Τίτλος Δ.Ε: Υλοποίηση έξυπνου συστήματος παρακολούθησης χώρου με αναγνώριση ανθρώπων/κατοικιδίων/αντικειμένων και αποστολή ειδοποιήσεων.
Κωδικός Δ.Ε: 25128
Ονοματεπώνυμο φοιτητή: Τρύφωνας Μαζαράκης
Ονοματεπώνυμο εισηγητή: Γεώργιος Κοκκώνης
Ημερομηνία ανάληψης: 21-02-2025
Ημερομηνία περάτωσης: 28-05-2025

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

«Στην επιμονή που δεν γνωρίζει τερματισμό»

Πρόλογος

Η επιλογή του θέματος συνδέεται με την φιλοδοξία μου να δημιουργώ πρακτικές υπηρεσίες. Η ενασχόληση μου με το frontend και η δίψα μου για τον συνδυασμό και την βελτίωση των τεχνικών μου γνώσεων, με εφεραν σε αυτό το μονοπάτι εκμάθησης νέων τεχνολογιών.

Ο συνδυασμός σύγχρονων πρακτικών λογισμικού, η ενσωμάτωση τεχνητής νοημοσύνης, η διαχείριση συσκευών εισόδου/εξόδου και η υλοποίηση τεχνικών Internet of Things, με καθοδήγησαν στην λήψη της απόφασης για το θέμα της διατριβής.

Περίληψη

Η εν λόγω διπλωματική εργασία, επικεντρώνεται στον διερεύνηση τεχνικών για την ανάπτυξη έξυπνων συστημάτων παρακολούθησης χώρου. Μέσω της μελέτης παρόμοιων συστημάτων και την έρευνα σε τεχνολογίες αρκετών παρόμοιων πεδίων, σχεδιάζεται και υλοποιείται ένα έξυπνο σύστημα παρακολούθησης χώρου. Η τελική λύση, είναι ικανή να αναγνωρίσει και να ειδοποιήσει τον ιδιοκτήτη σε περίπτωση παραβίασης του χώρου εστίασης. Η ειδοποίηση γίνεται μέσω διαφόρων μεθόδων για την προσαρμογή στην χρήστη. Προσφέρει σύγχρονες πρακτικές, ρύθμισης και ζωντανής επικοινωνίας, για την άμεση ενημέρωση των γεγονότων. Αποτελεί προσβάσιμο και ευέλικτο στις ανάγκες του χρήστη, παρέχοντας γραφική διεπαφή σε Web και Mobile εκδόσεις. Ενσωματώνει τεχνικές, διαχείρισης της ενέργειας και των υπολογιστικών πόρων, για την βελτιστοποίηση της απόδοσης. Επιπλέον, η χρήση μοντέλων τεχνητής νοημοσύνης, επιτρέπει την αυτόματη και την αξιόπιστη λειτουργία του συστήματος. Περεταίρω, το σύστημα υλοποιεί διάφορες μεθόδους ασφάλειας, για την προστασία των δεδομένων του διαχειριστή και του συστήματος κατά την μεταφορά. Η κυριότερη δυσκολία της υλοποίησης αποτελεί ο συνδυασμός πολλών διαφορετικών τεχνολογιών για την δημιουργία ενός μοναδικού συστήματος. Τέλος, η εφαρμογή αγγίζει διάφορα σημεία και τεχνολογίες, με θέληση, την συνεισφορά και την διέγερση του αναγνώστη, πάνω στον τομέα αυτόν.

«Smart Video Surveillance System»

«Tryfon Mazarakis»

Abstract

The diploma thesis, in question, focuses on the exploration of techniques for the development of smart video surveillance systems. Through the study of similar systems and the research of many similar fields, comes the designing and the implementation of a smart video surveillance system. The final solution is capable of detecting and notifying the owner in case of a breach in the monitored area. Notifications are delivered through various methods to adapt the user's needs. It offers modern practices for configuration and real-time communication, ensuring immediate updates on events. The system is accessible and flexible to the user's requirements, providing a graphical interface in both Web and Mobile versions. It incorporates techniques for energy and computational resource management to optimize performance. Additionally, the use of artificial intelligence models enables the automatic and reliable operation of the system. Furthermore, the system implements various security methods to protect the owner's and system's data when being transmitted. The main challenge of the implementation lies in the integration of multiple different technologies to create a single system. Finally, the application touches several points and technologies, with the will of contributing to and aspiring the reader in this field.

Ευχαριστίες

Αρχικά, θέλω να ευχαριστήσω τον εαυτό μου για την αφοσίωση και τους φίλους και συμφοιτητές, για την υποστήριξη που μου έδειξαν.

Ιδιαίτερα, ευχαριστώ τον Κ. Κοκκώνη, που μου έδωσε το πάτημα και την πολύτιμη καθοδήγηση του, για την εκπόνηση της διατριβής.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Συντομογραφίες

WSL - Windows Subsystem for Linux

PaaS - Platform as a Service

CLI - Command Line Interface

DOM - Document Object Model

FPS - Frames Per Second

RTMP - Real Time Media Protocol

DVCS - Distributed Version Control System

BaaS - Backend as a Service

FCM - Firebase Cloud Messaging

APNs - App Push Notification Service

NPM - Node Package Manager

JSON - JavaScript Object Notation

SQL - Structured Query Language

BSON - Binary Javascript Object Notation

XML - Extensible Markup Language

CRUD - Create Read Update Delete

CPU - Core Processing Unit

GPU - Graphics Processing Unit

TPU - Tensor Processing Unit

NPL - Natural Language Processing

CNN - Convolutional Neural Network

UI - User interface

HTML - Hyper Text Markup Language

JSX - JavaScript XML

HMR - Hot Module Replacement

CSS - Cascading Style Sheet

API - Application Programming Interface

OS - Operating System

JSI - JavaScript Interface

MQTT - Message Queuing Telemetry Transport

HTTP - HyperText Transfer Protocol

TCP - Transport Control Protocol

IP - Internet Protocol

HLS - HTTP Live Streaming

IOT - Internet Of Things

RFCOMM - Radio Frequency Communication Protocol

SMS - Short Message Service

GSM - Global System for Mobile Communications

SMSC - Short Message Service Center

SSL - Secure Sockets Layer

TLS - Transport layer Security

SD - Secure Digital

SSD - Solid State Drive

RAM - Random Access Memory

FPS - Frames Per Second

BLE - Bluetooth Low Energy

CSI - Camera Serial Interface

MIPI - Mobile Industry Processor interface

GPIO - General Purpose Input Output

IOPS - Input Output Operation per Second

USB - Universal Serial Bus

GPRS - General Packet Radio Service

UART - Universal Asynchronous Receiver/Transmitter

RTC - Real Time Clock

ULP - Ultra Low Power

AP - Access Point

SSD - Single Shot MultiBox Detection

SSH - Secure Shell

APT - Advanced Package Tool

CLI - Command Line Interface

MAC - Media Access Control

UUID - Universally Unique Identifiers

SSID - Service Set Identifier

BLOB - Binary Large Object

IV - Initialization Vector

Κεφάλαιο 1ο: Εισαγωγή

1.1 Εισαγωγή

Τα συστήματα παρακολούθησης, συνιστούν πλέον ανάγκη και υπάρχουν σε όλους τους χώρους που απαιτούν επίβλεψη. Οι επιχειρήσεις, τα γραφεία, οι οικείες, τα επιστημονικά εργαστήρια, εγκαθιστούν συστήματα παρακολούθησης, τα οποία πολλές φορές περιέχουν και επιλογές απομακρυσμένης πρόσβασης μέσω Mobile application. Τα κοινά μειονεκτήματα που έχουν οι συχνές αυτές υλοποιήσεις, αφορούν την συνεχή καταγραφή του χώρου καθώς, και σε κάποιες περιπτώσεις, την μη ενημέρωση του ιδιοκτήτη σε περίπτωση εισβολής στον χώρο. Αυτό έχει ως αποτέλεσμα την δημιουργία μεγάλου όγκου, μη σημαντικών, αρχείων βίντεο, την σπατάλη ενέργειας και τον εξαναγκασμό του χρήστη να παρακολουθεί συνέχεια ο ίδιος. Το σύστημα, που θα παρουσιαστεί στην εργασία, λύνει τα προβλήματα αυτά με την χρήση τεχνικών, που αρμόζουν στην σύγχρονη εποχή [1].

1.2 Στόχος διπλωματικής

Οι στόχοι της διατριβής, αποτελούν την λεπτομερής διερεύνηση των σύγχρονων συστημάτων παρακολούθησης χώρου και την κριτική ανάλυση και σύγκριση των τεχνικών, των υλικών (hardware) και των τεχνολογιών (software) που τα απαρτίζουν. Κύριο σημείο συνιστά η ανάπτυξη ενός έξυπνου συστήματος παρακολούθησης χώρου, μέσω μιας συστηματικής διαδικασίας, με βάση αυτά που ακολούθησαν προηγουμένως. Πιο συγκεκριμένα:

1) Ανάπτυξη ενός συστήματος παρακολούθησης χώρου με μικροϋπολογιστή Raspberry Pi και περιφερειακές συσκευές.
2) Ενσωμάτωση, μοντέλων μηχανικής μάθησης για την έξυπνη αναγνώριση ανθρώπων, προσώπων, κατοικίδιων, αντικειμένων και θορύβων.
3) Υλοποίηση αισθητήρων με ESP32 για την βέλτιστη αποδοτικότητα της ενέργειας.
4) Ειδοποίηση του ιδιοκτήτη μέσω SMS και Push Notifications, μέσω Mobile App.
5) Ζωντανή ενημέρωση του ιδιοκτήτη για την κατάσταση του συστήματος, μέσω διαδραστικής ιστοσελίδας.

Οι στόχοι αυτοί, αποσκοπούν, στην δημιουργία ενός έξυπνου συστήματος παρακολούθησης, που συνδυάζει διάφορες τεχνικές γύρω από IOT, Machine Learning, Live Streaming και Fullstack Development για την δημιουργία ενός έξυπνου συστήματος παρακολούθησης χώρου.

1.3 Δομή διπλωματικής

Η δομή της διπλωματικής, έχει σχεδιαστεί με τέτοιο τρόπο ώστε να κάνει εισαγωγή στο θέμα και στην συνεχεια να παρουσιάσει ομαλά όλα τα βήματα της ανάπτυξης. Το πρώτο κεφάλαιο, συμβάλλει στην εισαγωγή της εργασίας ενώ τα επόμενα αποτελούν:

1) Το 2ο κεφάλαιο, παρουσιάζει το θεωρητικό υπόβαθρο για την χρήση και υλοποίηση παρόμοιων συστημάτων, είτε εμπορικά είτε ερευνητικά, καθώς και αναφέρει μερικές από τις σημαντικότερες τεχνικές στον κλάδο αυτόν.
2) Το 3ο κεφάλαιο, αποσκοπεί στην ανάλυση των υλικών που χρησιμοποιήθηκαν για την δημιουργία του συστήματος.

3) Το 4ο κεφάλαιο, επικεντρώνεται στις τεχνολογίες λογισμικού και στα πρωτόκολλα που βασίζουν την ορθή λειτουργία του συστήματος.
4) Το 5ο κεφάλαιο, δίνει την εικόνα του συστήματος, μιλώντας για την αρχιτεκτονική συνολικά και ατομική.
5) Το 6ο κεφάλαιο, αποτελεί την υλοποίηση, όπου περιγράφεται αναλυτικά ο τρόπος ανάπτυξης του κάθε κομματιού του συστήματος.
6) Το 7ο κεφάλαιο, συνοψίζει την διατριβή, κάνοντας λόγο για διάφορα συμπεράσματα και προτάσεις βελτιώσεις από ολόκληρη την μελέτη και υλοποίηση.

Μετά την ολοκλήρωση των κεφαλαίων, ακολουθεί η βιβλιογραφία και τα παραρτήματα. Τα παραρτήματα αποσκοπούν στην παρουσίαση ολόκληρη της υλοποίησης των σημαντικότερων κομματιών του συστήματος.

Κεφάλαιο 2ο: Βιβλιογραφική Ανασκόπηση

2.1 Ιστορική εξέλιξη

Η συνεχής τεχνολογική εξέλιξη και η ειδικότερα η ενσωμάτωση της τεχνητής νοημοσύνης, τα τελευταία χρόνια, έχει παίξει σημαντικό ρόλο για την πρόοδο των συστημάτων παρακολούθησης χώρου. Αρχικά, η αρχικές υλοποιήσεις, παρουσιάζουν την ανάγκη παρακολούθησης της κάμερας από άνθρωπο, ολόκληρο το 24ωρο. Συνεπώς, η ανίχνευση περιστατικών, ήταν στην ανθρώπινη κρίση. Με αποτέλεσμα, την αυξημένη ανάγκη ανθρώπινου δυναμικού [2].

Καθώς, οι κάμερες ψηφιοποιήθηκαν, αναπτύχθηκαν καλύτεροι τρόποι παρακολούθησης του χώρου. Δυνατότητες όπως περιστροφή της κάμερας, ανίχνευση κίνησης και μεγέθυνση της εικόνας αποτέλεσαν σημαντική καινοτομία για την καλύτερη παρακολούθηση από τους παρατηρητές. Το πρόβλημα, όμως της ανάγκης προσωπικού, δεν μειώθηκε μέσω της ψηφιοποίησης αλλά αυξήθηκε, καθώς η υλοποίηση συστημάτων με πολλές κάμερες, χρειαζόταν πολλούς διαχειριστές. Την λύση στο πρόβλημα αυτό, λύνει τα τελευταία χρόνια η έξυπνη παρακολούθηση [2].

Η έξυπνη παρακολούθηση, συνιστά την χρήση μοντέλων μηχανικής μάθησης και τεχνητή νοημοσύνη, για την παρακολούθηση της ζωντανής ροής. Συνεπώς, η ανάγκη για προσωπικό, καλυβεται απο την νέα τεχνολογία, η οποία με ταχείς ρυθμούς εξελίσσεται. Η αναγνώριση ανθρώπων, προσώπων, αντικειμένων και ήχων και η αυτόματα ειδοποίηση αποτελούν πλέον, αυτόματες διαδικασίες που μπορούν να υλοποιηθούν ακόμα και σε μία συσκευή Raspberry Pi των 100 ευρώ.

Συνοψίζοντας, η τελευταίες τεχνολογικές εξελίξεις, έχουν και αλλάζουν συνεχώς τον τομέα της παρακολούθησης χώρου προς την αυτοματοποίηση. Δυνατότητες, όπως η αυτόματα εύρεση εγκληματιών σε μία πόλη, από τις αρχές, και η αυτόματη φύλαξη χώρων από μία συσκευή, αποτελούν πλέον υλοποιήσιμα μέσω της τεχνητής νοημοσύνης.

2.2 Υπάρχων συστήματα

Τα συστήματα παρακολούθησης χώρου, χωρίζονται σε αυτά που αποτελούνται από λογισμικό και μπορούν να εγκατασταθούν σε μηχανήματα όπως Raspberry Pi, και σε ολοκληρωμένες λύσεις. Οι ολοκληρωμένες λύσεις, αποτελούν συνήθως και εμπορικές, κλειστού κώδικα, με ενσωματωμένο hardware και software σε ένα πλήρες προϊόν. Το παρόν κεφάλαιο, θα ερευνήσει και θα συγκρίνει, τα δημοφιλέστερα συστήματα, της πρώτης κατηγορίας, τα οποία είναι open source.

2.2.1 Σύστημα Motioneye

Το σύστημα Motioneye, αποτελεί μία υλοποίηση ιστοσελίδας, frontend, για την διαχείριση της τεχνολογίας motion daemon. Μέσω της τεχνολογίας, αυτής, είναι δυνατή η ανίχνευση κίνησης από την ροή βίντεο και στην συνέχεια η εκτελεση προκαθορισμένων ενεργειών. Το Motioneye, εκμεταλλεύεται την τεχνολογία αυτή, την γλώσσα Python και τις γλώσσες Web, HTML, CSS και JavaScript, για την δημιουργία ενός Web Interface.

Μέσω της διεπαφής, είναι δυνατή η ρύθμιση αρκετών παραμέτρων της ζωντανής ροής και των διάφορων καμερών. Συνεπώς, μέσω της ιστοσελίδας, ο χρήστης μπορεί να ρυθμίσει και να διαχωριστεί πλήρως τις συνδεδεμένες κάμερες. Η εγκατάσταση του λογισμικού, είναι συμβατή και δημοφιλές σε συσκευές Raspberry Pi, και αποτελεί μία απλή διαδικασία μερικών εντολών. Επιπλέον, καθώς δεν χρησιμοποιούνται μοντέλα μηχανικής μάθησης, η απαιτήσεις της υλοποίησης είναι χαμηλές.

Συνοψίζοντας το Motioneye, αποτελεί ένα πλήρες ρυθμιζόμενο σύστημα παρακολούθησης χώρου, με παραδοσιακή αναγνώριση κίνησης, χωρίς την χρήση τεχνητής νοημοσύνης. Αποτελεί μία γρήγορη λύση, για την εγκατάσταση συστήματος παρακολούθησης χώρου, σε συσκευές με περιορισμένους υπολογιστικούς πόρους.

2.2.2 Σύστημα Kerberos.io

Το σύστημα Kerberos.io, αποτελεί ένα λογισμικό στο Cloud, που παρέχει τα εργαλεία για το στήσιμο μεγάλης κλίμακας συστημάτων παρακολούθησης χώρου. Επιπλέον, επιτρέπει την εγκατάσταση τοπικά, για την δημιουργία ενός offline συστήματος, αλλά και την χρήση της πλατφόρμας. Με την online, επιλογή, όλα τα δεδομένα, της ζωντανής ροής, αποθηκεύονται στο Cloud και είναι προσβάσιμα οποιαδήποτε στιγμή από τον χρήστη.

Παρέχει μία πληθώρα εργαλείων, τα οποία ενεργοποιούνται ανάλογα τις ανάγκες της κάθε υλοποίησης. Εστιάζεται στα video analytics, computer vision και μηχανική μάθηση. Την κάθε λειτουργία, δίνεται η δυνατότητα επιλογής και διαχείρισης από τον χρήστη, στοχεύοντας έτσι στην πλήρη προσαρμοστικότητα του συστήματος.

Συνοψίζοντας, το Kerberos.io, προορίζεται για εγκαταστάσεις συστημάτων παρακολούθησης με έμπειρο προσωπικό, με στόχο την κλιμάκωση και την διαχείριση πολλών δεδομένων.

2.2.3 Σύστημα Frigate NVR

Το σύστημα Frigate NVR (Network Video Recorder), είναι ένα λογισμικό αναγνώρισης αντικειμένων real-time. Δεν αποτελεί μία Cloud λύση, συνεπώς, χρειάζεται η εγκατάσταση τοπικά σε συσκευή και ενσωματώνεται με συστήματα IOT όπως Google Home.

Τα χαρακτηριστικά του συστήματος αποτελούν η αναγνώριση ανθρώπων και αντικειμένων και η δήλωση περιοχών ειδοποίησης (alert zones). Προορίζεται κυρίως για εξωτερική χρήση, καθώς ο χρήστης μπορεί να ρυθμίσει περιοχές, τις οποίες, ορισμένες αγνοεί και ορισμένες πραγματοποιεί αναγνώριση ζωντανού χρόνου για την αποστολή ειδοποιήσεων.

Χρησιμοποιεί Python μαζί με OpenCV και Tensorflow, για την υλοποίηση των μοντέλων τεχνητής νοημοσύνης. Επιπλέον, υποστηρίζει επεξεργαστές AI όπως Google Coral, με στόχο την real-time αναγνώριση αντικειμένων σε πολλά Camera feeds και με αρκετούς μικρούς χρόνους καθυστέρησης.

Συνοψίζοντας, το σύστημα Frigate NVR, δίνει έμφαση στην ευκολία ενσωμάτωσης από τον απλό χρήση μέσω Smart Home εφαρμογών. Προορίζεται κυρίως για εξωτερικούς χώρους, όπως την εξώπορτα και τον κήπο. Επιπλέον, κάνει χρήση τεχνικών για την βελτίωση της αξιοπιστίας και της ταχύτητας της αναγνώρισης εισβολέων.

2.2.4 Σύστημα Shinobi CCTV

Το σύστημα Shinobi CCTV, συνιστά ένα open source λογισμικό για την δημιουργία κλειστού συστήματος παρακολούθησης χώρου. Αποτελεί ένα από τα ελάχιστα συστήματα υλοποιημένο με περιβάλλον Nodejs και Tensorflowjs. Προσφέρει πλήρες δυνατότητες καταγραφής και Video Streaming καθώς και αναγνώριση αντικειμένων, προσώπων και καταγραφή κινήσεων.

Είναι συμβατό με συσκευές Raspberry Pi, Jetson Nano αλλά και μέσω της χρήσης Docker μπορεί να εγκατασταθεί σε οτιδήποτε σύστημα Linux, ακόμα και σε Windows.

Η ενσωμάτωση του, προϋποθέτει τεχνικές γνώσεις, καθώς προσφέρει εργαλεία στον προγραμματιστή να αναπτύξει ένα πλήρες έξυπνο σύστημα παρακολούθησης χώρου, μέσω ενός λεπτομερούς documentation.

Συνοψίζοντας, η επιλογή του κάθε συστήματος εξαρτάται άμεσα από τις απαιτήσεις της υλοποίησης και τις γνώσεις του χρήστη. Το Motioneye, αποτελεί ένα απλό σύστημα στην εγκατάσταση με περιορισμένες δυνατότητες, χωρίς την χρήση τεχνητής νοημοσύνης. Από την άλλη, τα υπόλοιπα συστήματα χρησιμοποιούνται και βασίζονται σε τεχνικές έξυπνης αναγνώρισης. Το λογισμικό Frigate NVR, αποτελεί μία αρκετά αξιόπιστη λύση σε καθημερινές οικιακές ανάγκες, όπου ο χρήστης δεν έχει ιδιαίτερες γνώσεις πάνω στον τομέα. Το Kerberios.io και Shinobi CCTV, είναι δύο παρόμοιες υλοποιήσεις, οι οποίες δίνουν τα εργαλεία και το πάνω χέρι στον χρήστη, να στήσει το σύστημα όπως αυτός επιθυμεί. Τέλος, κοινό χαρακτηριστικό των συστημάτων, αποτελεί η χρήση Docker, για τον διαχωρισμό των διαφόρων λειτουργιών σε μικρουπηρεσίες.

2.3 Τεχνικές στην έξυπνη παρακολούθηση

2.3.1 Video Streaming

Η έξυπνη παρακολούθηση (Smart Video surveillance), βασίζεται σε σύγχρονες τεχνολογίες για την παρακολούθηση και ανάλυση της ροής βίντεο σε πραγματικό χρόνο. Ειδικά, παρουσιάζονται υψηλές απαιτήσεις, σε υλοποιήσεις ασφαλείας ιδιωτικών χώρων, όπου η χρειάζεται η γρήγορη ανάλυση πολλών ροών βίντεο παράλληλα.

Η πρώτη και βασικότερο ζήτημα, σε ένα σύστημα έξυπνης παρακολούθησης, αποτελεί η υλοποίηση του Video Streaming. Η ροή των δεδομένων βίντεο, είναι ο πυλώνας για την σωστή λειτουργία των υπόλοιπων δυνατοτήτων ενός συστήματος.

Η καλύτερη ανάλυση της εικόνας βίντεο, συνεπάγεται με την καλύτερη αναγνώριση του χώρου. Συνεπώς, συνεχώς βελτιώνονται και ανακαλύπτονται πρωτόκολλα και τεχνικές Video Streaming, με κύριο στόχο, την ελαχιστοποίηση του μεγέθους μεταφερόμενων δεδομένων και την μεγιστοποίηση της τελικής ποιότητας, κρατόντας την καθυστέρηση όσο χαμηλότερα γίνεται.

Η χρήση κωδικοποιητών, συνεισφέρει στην ικανοποίηση του στόχου, με τον πιο δημοφιλές να αποτελεί ο HEVC (High Efficiency Video Coding) ή αλλιώς H.265. Μέσω του οποίου, η ζωντανή ροή, συμπιέζεται, με τελικό μέγεθος έως και 50% λιγότερο. Μετά την συμπίεση της ροής, απομένει η μεταφορά στο σημείο παρακολούθησης. Υπάρχουν διαθέσιμα, αρκετά πρωτόκολλα επικοινωνίας για Video Streaming, με τα πιο δημοφιλές να αποτελούν RTSP, RTMP, HLS και WebRTC. Η χρήση του κατάλληλου πρωτοκόλλου συνδέεται με τις ανάγκες υλοποίησης της έξυπνης παρακολούθησης.

2.3.2 Έξυπνη αναγνώριση

Όλες οι δυνατότητες των σύγχρονων συστημάτων παρακολούθησης χώρου, βασίζονται στην έξυπνη αναγνώριση. Συγκεκριμένα, η αναγνώριση προσώπων, αντικειμένων και ήχου σε πραγματικό χρόνο, έχουν φέρει δραστικές αλλαγές στον τρόπο και στις απαιτήσεις υλοποίησης τέτοιων συστημάτων. Η ανάγκη για την ανάπτυξη πιο ισχυρών συστημάτων, συγκεκριμένα επεξεργαστές, προέρχεται από την χρήση μοντέλων τεχνητής νοημοσύνης σε πραγματικό χρόνο. Συνεπώς, τα σύγχρονα συστήματα παρακολούθησης χώρου, αποτελούνται από ισχυρές συσκευές και δεν χρήζουν ανάγκη συνεχής ανθρώπινης επιτήρησης [3].

Η κύριες δοκιμασίες στην αναγνώριση με υψηλή εμπιστοσύνη, αποτελούν οι τεχνικές εκπαίδευσης των μοντέλων και ειδικότερα το set δεδομένων. Υπάρχουν προ εκπαιδευμένα μοντέλα, που μπορούν να ικανοποιήσουν τις βασικές ανάγκες της έξυπνης αναγνώρισης. Βιβλιοθήκες και εργαλεία όπως η Tensorflow, απλοποιούν και βελτιώνουν την χρήση και την εκπαίδευση μοντέλων.

Η πιο διαδεδομένη τεχνική έξυπνης αναγνώρισης, αποτελεί η αναγνώριση προσώπων (Face Recognition). Κύρια υλοποίηση των συστημάτων έξυπνης παρακολούθησης, είναι η αναγνώριση προσώπων σε πραγματικό χρόνο. Η τεχνική της αναγνώρισης προσώπων εισάγει τα περισσότερα προβλήματα, καθώς τα πρόσωπα είναι σπάνια πλήρως ορατά από γωνίας της κάμερας, συνεπώς αρκέζονται στην σύγκριση ελάχιστον χαρακτηριστικών του προσώπου. Πιο συγκεκριμένα η ροή της διαδικασίας είναι η εξής [3]:

1) Διαχωρισμός της ζωντανής ροής σε καρέ, με έναν ικανοποιητικό ρυθμό καρέ ανα δευτερόλεπτο (FPS).
2) Σε κάθε καρέ, πραγματοποιείται ανίχνευση προσώπων, μέσω ειδικού μοντέλου (Face Detection).
3) Από το κάθε εντοπισμένο πρόσωπο, γίνεται εξαγωγή χαρακτηριστικών (Feature Extraction). Με τελική έξοδο, ένα διάνυσμα embedding.
4) Σύγκριση των embeddings με τα αποθηκευμένα embeddings στην βάση, για τον καθορισμό του ποσοστού ομοιότητας.

Το ποσοστό ομοιότητας, είναι σχετικό και ανάλογα με την ποιότητα του τραβηγμένου προσώπου. Είναι σημαντική, κατά την αποθήκευση των γνωστών προσώπων, η παρατήρηση του προσώπου από πολλές διαφορετικές γωνίες και φωτισμούς, έτσι ώστε να μπορεί να αναγνωριστεί στις περισσότερες δυνατές περιπτώσεις [3].

Η τεχνική αναγνώριση εικόνων, είναι σημαντική στην αναγνώριση αντικειμένων από ένα σύστημα. Για παράδειγμα, η αυτόματη αναγνώριση και εύρεση πινακίδων κυκλοφορίας και παράνομων υλικών, παίζει μεγάλο ρόλο για την βελτίωση της ασφάλειας σε όλους τους χώρους. Τα μοντέλα αυτά, αποτελούνται από εκατοντάδες χιλιάδες ποιοτικές εικόνες για την εκπαίδευση τους, έτσι ώστε να μπορούν να αναγνωρίζουν ένα αντικείμενο σε οποιαδήποτε μορφή, γωνία και φωτισμό. Επιπλέον ικανότητα, συνιστά, η παροχή του σημείου αναγνώρισης σε μία εικόνα. Είναι αρκετά χρήσιμο, σε εφαρμογές δηλωσης επικίνδυνων και ασφαλών περιοχών στο οπτικό πεδίο μιας κάμερας [3].

Η αναγνώριση ήχου, είναι χρήσιμη στην αναγνώριση ήχων από συμβάντα που δεν καταγράφονται στην κάμερα, όπως πυροβολισμοί. Χρησιμοποιείται πιο σπάνια σε σχέση με τις άλλες τεχνικές, καθώς ένας ήχος μπορεί να διαστρεβλωθεί με πολλούς τρόπους και επιπλέον επικρατεί το πρόβλημα του θορύβου. Γι αυτό, συνήθως χρησιμοποιείται σε συνδυασμό με αναγνώριση εικόνας [3].

Συνοψίζοντας, η αναγνώριση προσώπων και εικόνας αποτελούν τα βασικότερα στοιχεία των συστημάτων έξυπνης παρακολούθησης. Υπάρχουν έτοιμες λύσεις, προεκπαιδευμένων μοντέλων, όμως μέσω της εκπαίδευσης σε ειδικά data set μπορούν να δημιουργηθούν καλύτερες λύσεις για συγκεκριμένες περιπτώσεις χρήσης. Καθώς η χρήση έξυπνης αναγνώρισης, απαιτεί μεγάλη υπολογιστική ισχύ, γίνεται αποστολή των δεδομένων για επεξεργασία από εξωτερικούς server, γεγονός που πάσχει από προβλήματα ιδιωτικότητας.

2.3.3 Χρήση αισθητήρων

Η χρήση αισθητήρων, παίζει σημαντικό ρόλο στην βελτίωση της έξυπνης παρακολούθησης. Αρχικά, η συνεχή λειτουργία των συσκευών κάμερας και των μοντέλων τεχνητής νοημοσύνης για αναγνώριση, προσδίδει στην αυξημένη κατανάλωση ενέργειας. Με αποτέλεσμα, συστήματα που τροφοδοτούνται από μπαταρία, να μην είναι υλοποιήσιμα. Επιπλέον, ο αποθηκευτικός χώρος του συστήματος, γεμίζει από ανούσια αρχεία βίντεο, με αποτέλεσμα να διαγράφονται ύστερα από ένα μικρό διάστημα, όπως 10 μέρες. Επιπλέον, πέρα από την εικόνα και τον ήχο, δεν είναι δυνατή η καταγραφή περισσότερων μετρήσεων από έναν χώρο.

Η χρήση αισθητήρων, σε έναν χώρο, αποτελεί πλήρες λύση στα προβλήματα που αναφέρθηκαν. Οι αισθητήρες κίνησης και ήχου μπορούν να παραμένουν σε λειτουργία συνέχεια, καθώς καταναλώνουν μικρή ενέργεια, και να ενεργοποιούν το σύστημα έξυπνης παρακολούθησης, μόνο όταν πυροδοτηθούν. Με αποτέλεσμα, την σημαντική μείωση στην κατανάλωση της ενέργειας και των αποθηκευμένων βίντεο.

Επιπλέον, η χρήση αισθητήρων μετρήσεων περιβαλλοντολογικών συνθηκών, μπορούν να δώσουν μία πλήρη εικόνα του χώρου, καθόλη την διάρκεια της ημέρας. Όχι μόνο, είναι δυνατή η παρακολούθηση θερμοκρασίας και υγρασίας αλλά η μέτρηση της ποιότητας του αέρα και των σωματιδίων στον αέρα, μπορεί να λειτουργήσει ως δείκτης για την πρόληψη πυρκαγιάς.

Συνοψίζοντας, ένα έξυπνο παρακολούθησης χώρου, μπορεί μόνο να εισπράξει θετικά από την χρήση αισθητήρων. Αποτελούν μία φθηνή λύση και αποτελεί στην σημαντική μείωση των εξόδων ενός συστήματος.

2.4 Προκλήσεις υλοποίησης

Η υλοποίηση έξυπνων συστημάτων παρακολούθησης χώρου, συνοδεύεται με αρκετές προκλήσεις και σοβαρά θέματα που πρέπει να αντιμετωπιστούν. Η συλλογή δεδομένων, από συσκευές εισόδου, όπως κάμερα και μικρόφωνο, συνοδεύεται με νομικά θέματα περί ιδιωτικότητας. Επιπλέον, η χρήση τεχνητής νοημοσύνης σε συνδυασμό με την επεξεργασία βίντεο, αναγκάζουν την περισσότερη υπολογιστική ισχύ [5].

Τα συστήματα που υλοποιούνται τοπικά, στο δίκτυο, δεν πάσχουν από προβλήματα περί ιδιωτικότητας. Όμως η λύση αυτή, συνοδεύεται με την δημιουργία περιορισμών στην υπολογιστική ισχύ και στον χώρο αποθήκευσης, τα οποία συνεπάγονται με την αύξηση του κόστους υλοποίησης. Από την άλλη, Cloud υλοποιήσεις, συνήθως προσφέρουν πιο φθηνές και αφηρημένες λύσεις, αλλά είναι αναγκαία η συλλογή προσωπικών δεδομένων [4][5].

Ειδικά σε ιδιωτικούς χώρους, η συλλογή δεδομένων, είναι αναγκαία για την λειτουργία των συστημάτων παρακολούθησης χώρου μέσω Cloud. Στην Ευρωπαϊκή Ένωση, μέσω του κανονισμού GDPR (General Data Protection Regulation), ο χρήστης πρέπει πρώτα να συναινέσει στην συλλογή των προσωπικών του δεδομένων. Η πλατφόρμα, πρέπει να ενημερώσει για την χρησιμότητα της καταγραφής των δεδομένων και με την αποδοχή του χρήστη συναινεί, από την μεριά του. Η διαφάνεια από την μεριά του συστήματος, δεν εγγυάται και την εμπιστευτικότητα των δεδομένων [4].

Κεφάλαιο 3ο: Επιλογή υλικών

3.1 Εισαγωγή

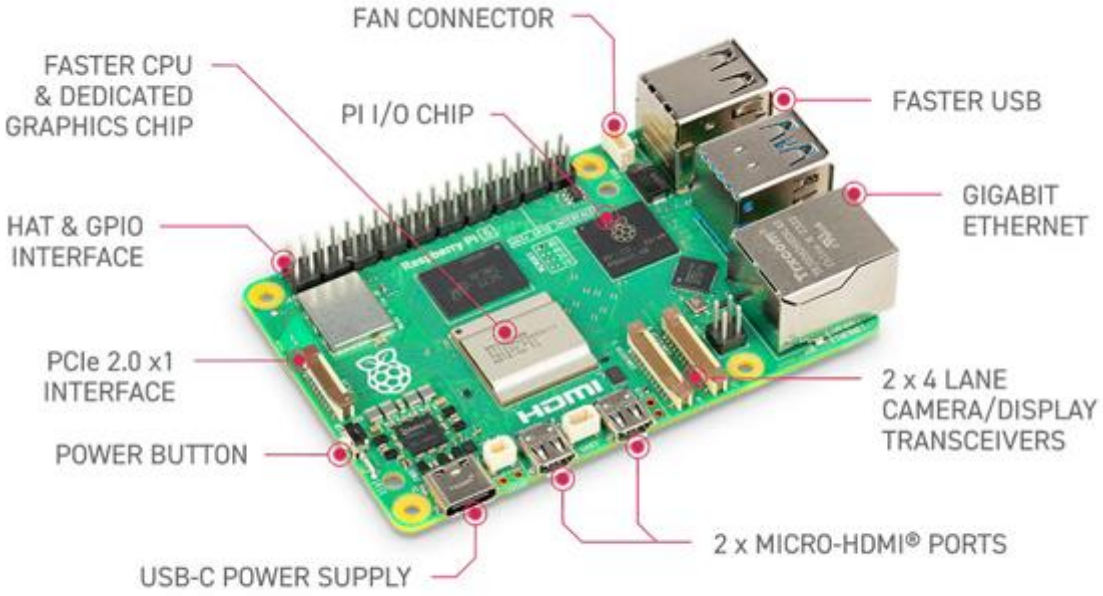
Σε αυτό το κεφάλαιο θα αναλυθούν τα εξαρτήματα που επιλέχθηκαν και αποτελούν την υλική υλοποίηση του συστήματος. Κύριο κομμάτι συζήτησης, αποτελεί το Raspberry Pi που λειτουργεί ως η υπολογιστική ισχύ του συστήματος, επιπλέον χρησιμοποιήθηκαν μία Raspberry Pi Camera και ένα χαμηλού κόστους μικρόφωνο, τα οποία αποτελούν τις εισόδους όλου του εγχειρήματος. Για την αποδοτικότητα της ενέργειας γίνεται χρήση αισθητήρων κίνησης και ήχου, οι οποίοι συνδέονται και επικοινωνούν με το σύστημα μέσω του μικροελεγκτή ESP32. Για την αποστολή ειδοποιήσεων στον χρήστη, υπάρχει το GSM Hat, το οποίο κουμπώνει στο Raspberry Pi και αποτελεί GSM Modem για την αποστολή SMS προγραμματιστικά μέσω SIM Card. Τέλος, θα αναφερθεί το συνολικό κόστος υλικών καθώς και διάφορες εναλλακτικές επιλογές.

3.2 Κριτήρια επιλογής υλικού

Κριτήρια για την επιλογή υλικού αποτέλεσαν διάφορα παράγοντες οι οποίοι ανάλογα με το σύστημα και τις συνθήκες μπορεί να αλλάξουν. Αρχικά, ως γνώμονα πρώτα την επιστημονική ανάλυση, δεν επιλέχθηκαν ολοκληρωμένες και έτοιμες λύσεις. Επιπλέον ρόλο έπαιξε το κόστος των υλικών να είναι το μικρότερο δυνατό χωρίς να επηρεάζεται η συμβατότητα του συστήματος. Περαιτέρω, η διαθέσιμη πληροφορία και η υποστήριξη της κοινότητας, για κάθε hardware κομμάτι αποτελεί σημαντικό παράγοντα για την ευκολία ενσωμάτωσης. Τέλος, κριτήρια όπως η αποδοτικότητα και η ευκολία κλιμάκωσης πάρθηκαν υπ όψιν.

3.3 Raspberry Pi

Το Raspberry Pi, αποτελεί την καρδιά του συστήματος και συνεπώς το πιο σημαντικό εξάρτημα. Πιο συγκεκριμένα, τα Raspberry Pi μοντέλα αποτελούν μια σειρά από ισχυρούς μικροϋπολογιστές single-board. Αναπτύχθηκε το 2006 με στόχο την διάθεση υπολογιστών με καλή σχέση κόστους-απόδοσης με προορισμό την εμπορική και εκπαιδευτική χρήση. Πλέον βρίσκεται σε παραγωγή το 5ο μοντέλο της σειράς αποτελώντας μία καινοτομία στο χώρο των μικροϋπολογιστών. Επιπλέον παρέχεται διαθεσιμότητα από διάφορα περιφερειακά εξαρτήματα ειδικά για το συγκεκριμένο board, όπως κάμερες, ψύκτρες, κουτιά (cases), κάρτες micro SD (Secure Digital) και σκληρούς δίσκους SSD (Solid State Drive), GSM modules, επεξεργαστές μηχανικής μάθησης και άλλα. Τέλος, χρησιμοποιείτε σε εφαρμογές, όπως παρακολούθησης χώρου και υγείας, βιομηχανική αυτοματοποίηση και σε άλλες περιπτώσεις με παρόμοιες ανάγκες [6].



Σχήμα 3.1: Εικόνα Raspberry Pi 5, με αναφορά στα νέα χαρακτηριστικά του [7].

Για την υλοποίηση του συστήματος επιλέχθηκε το 5ο μοντέλο Raspberry Pi. Αποτελεί την πιο καινούρια προσθήκη της σειράς και διατίθεται σε εκδόσεις των 2, 4, 8 και πλέον 16 gigabyte μνήμης RAM. Συγκεκριμένα, κρίθηκε απαραίτητη η χρήση της διασκευής των 8GB μνήμης. Καθώς στο σύστημα θα γίνεται η εκτέλεση πολλών διακομιστών, Live Streaming και έξυπνης αναγνώρισης εικόνας και ήχου. Οι ενέργειες αυτές χρειάζονται μεγάλη διάθεση μνήμης με αποτέλεσμα, τα μοντέλα με μικρότερο χώρο μνήμης να λειτουργούν σε πιο αργές ρυθμίσεις που κρίνονται μη ιδανικές σε περιβάλλοντα έξυπνης παρακολούθησης. Μερικά από τα χαρακτηριστικά του Raspberry Pi, που χρησιμοποιούνται αποτελούν [8]:

1) Ειδική θήκη για κάρτες Micro SD. Μπορούν να τοποθετηθούν οποιαδήποτε εταιρίας αρκεί να τηρούν τις προδιαγραφές αλλά υπάρχουν και υλοποιήσεις συγκεκριμένες για το μηχάνημα. Υποστηρίζονται 32, 64 και 128 gigabyte αποθηκευτικού χώρου.
2) Θύρα σύνδεσης ψύκτρας και heatsink, για την αποφυγή της υπερθέρμανσης των εξαρτημάτων.
3) Επιλογή σύνδεσης στο διαδίκτυο μέσω θύρας Ethernet και αντάπτορα WiFi. Υποστηρίζεται WiFi 5ης γενιάς (πρότυπο 802.11ac) διπλής ζώνης 2.4GHz και 5GHz.
4) Αντάπτορας Bluetooth, μέσω Bluetooth 5.0 και BLE, χαμηλής ενέργειας.
5) 2 MIPI (Mobile Industry Processor Interface) θήκες για ενσωμάτωσης κάμερας μέσω CSI (Camera Serial Interface). Προσφέρουν μεγαλύτερη ταχύτητα από θύρες USB.
6) 2 θύρες USB 2.0 και 2 θύρες 3.0, υποστηρίζοντας 5Gbps ταχύτητες.
7) 40-pin GPIO. Για την αποστολή και λήψη σημάτων από hardware, όπως αισθητήρες και GSM Module HAT.
8) Κάρτα γραφικών (GPU) 800MHz, με υποστήριξη 3D γραφικών OpenGL και 4K Video Streaming. Ιδανικό για εφαρμογές με χρήση κάμερας και Streaming.

9) Επεξεργαστής (CPU), αρχιτεκτονικής ARM 64-bit, 2.4GHz με 4 πυρήνες. Η αρχιτεκτονική του βοηθάει στην υποστήριξη μοντέρνων λογισμικών και αποτελεί ισχυρή υλοποίηση για διαχείριση πολύπλοκων διαδικασιών.

Σχήμα 3.2: Τοποθέτηση ψύκτρας και heatsink στο board [9].

Απαραίτητη χρίζεται η τοποθέτηση ψύκτρας και heatsink. Η ψύκτρα χρησιμοποιείται για την μείωση της θερμοκρασίας στους επεξεργαστές, με σκοπό την αποφυγή υπερθέρμανσης στις περιπτώσεις που χρησιμοποιείται το μεγαλύτερο ποσοστό ισχύς. Το heatsink, είναι υπεύθυνο για την απορρόφηση της θέρμανσης.

Επιπλέον, έγινε χρήση Raspberry Pi micro SD Card του μεγέθους των 64GB. Διαθέσιμες επιλογές υπάρχουν στα 32 και 128 gigabyte, όμως λόγου της υλοποίησης του συστήματος να καταγράφει βίντεο, τα 32 χρίζονται λίγα. Τα 128GB, είναι μη αναγκαία καθώς πραγματοποιείτε καταγραφή μόνο σε περιπτώσεις εισβολών, μειώνοντας την ανάγκη για μεγάλο χώρο αποθήκευσης. Έχει υποστήριξη SDR104 με ταχύτητα διαύλου 104 MB/s και ταχύτητα read/write των 5000 και 2000 IOPS των 4KB. Συνοψίζοντας, η σειρά micro SD καρτών αυτή αποτελεί την καλύτερη και προτεινόμενη επιλογή, micro SD card, για συστήματα Raspberry Pi [7][8].

3.4 Κάμερα και μικρόφωνο

Η κάμερα και το μικρόφωνο ευθύνονται για την καταγραφή βίντεο και ήχου αντίστοιχα. Με έμφαση κυρίως στην κάμερα, είναι σημαντική η σωστή επιλογή εξαρτήματος για την αποδοτικότερη λειτουργία του Live Streaming. Υπάρχουν 2 επιλογές τύπων καμερών για το Raspberry Pi, η πρώτη είναι μέσω USB 3.0, μία αρκετά γρήγορη διεπαφή, και η δεύτερη είναι μέσω της CSI (Camera Serial Interface) θύρας, ειδικά κατασκευασμένη με μεγάλες ταχύτητες. Η επιλογή είναι εμφανής, καθώς είναι διαθέσιμες στην παραγωγή ειδικές Raspberry Pi κάμερες που εκμεταλλεύονται αυτή την θύρα. Έτσι επιλέχθηκε το 3ο και πιο καινούριο μοντέλο αυτής της σειράς το Raspberry Pi Camera Module V3 (NoIR).

Σχήμα 3.3: Raspberry Pi Camera Module V3 - NoIR 11.9MP 75° [10].

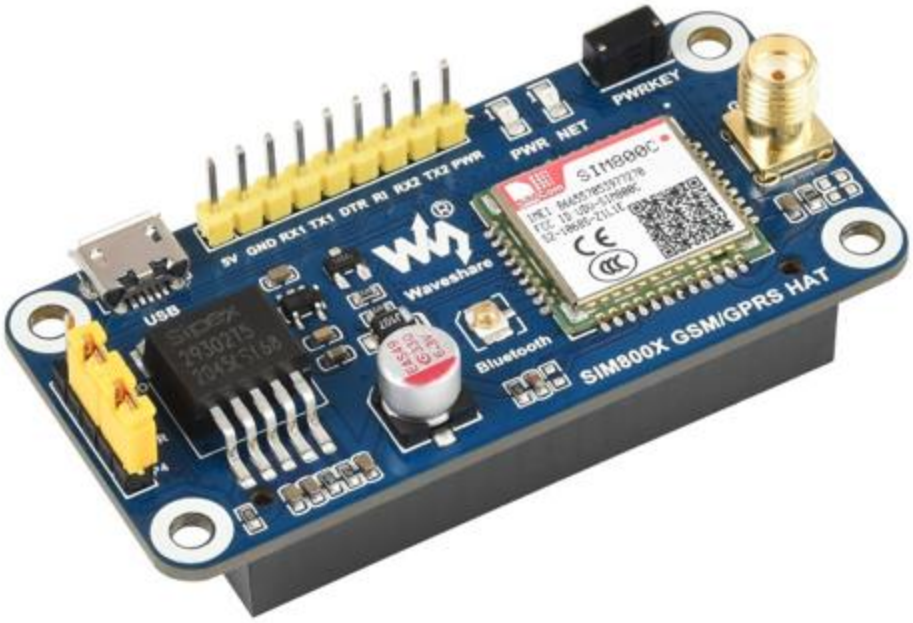
Στο σχήμα 3.3, απεικονίζεται το μοντέλο της κάμερας και αναφέρετε το όνομα του μοντέλου. Πιο συγκεκριμένα, η κάμερα που επιλέχθηκε είναι NoIR, δηλαδή χωρίς φίλτρο υπέρυθρης ακτινοβολίας. Η επιλογή αυτή, έχει ως αποτέλεσμα η κάμερα μέσω ειδικού φωτισμού IR (Infrared Radiation), να είναι εμφανές οι σκοτεινοί χώροι από ανθρώπινο μάτι. Κάνοντας την ειδική σε περιβάλλοντα όπου είναι αναγκαία η νυχτερινή όραση. Πέρα από το κομμάτι αυτό, παρέχει ισχυρές δυνατότητες Video με υποστήριξη πολλών FPS σε υψηλές αναλύσεις. Πιο συγκεκριμένα [10]:

1) Υποστήριξη 4K ποιότητας Video σε 60FPS και 1080p HD στα 90FPS.
2) Ενώνεται μέσω της διεπαφής CSI στο Raspberry Pi, για την αποδοτικότερη αποστολή δεδομένων βίντεο.
3) Έχει 75 μοιρών οπτικό πεδίο για την καταγραφή του χώρου.

Το σύστημα, εστιάζει στο Live Streaming και στην αναγνώριση αντικειμένων, κατοικιδίων και προσώπων μέσω εικόνων. Ως, επιπρόσθετο (addon), παρέχεται υποστήριξη καταγραφής φωνής και συνεπώς αναγνώρισης ήχων μέσω μικροφώνου. Η σειρά των Raspberry Pi δεν παρέχει κάποια επίσημη λύση για τον ήχο, έτσι μπορεί να χρησιμοποιηθεί οποιοδήποτε μικρόφωνο έχει θύρα USB 2.0 ή ακόμη καλύτερα USB 3.0 και υποστηρίζεται από το λειτουργικό σύστημα του μικροϋπολογιστή [8]. Έτσι επιλέχθηκε ένα χαμηλού κόστους μικρόφωνο που υποστηρίζει σύνδεση USB 3.0, και δεν αποτελεί ενδιαφέρουσα η περισσότερη ανάλυση του.

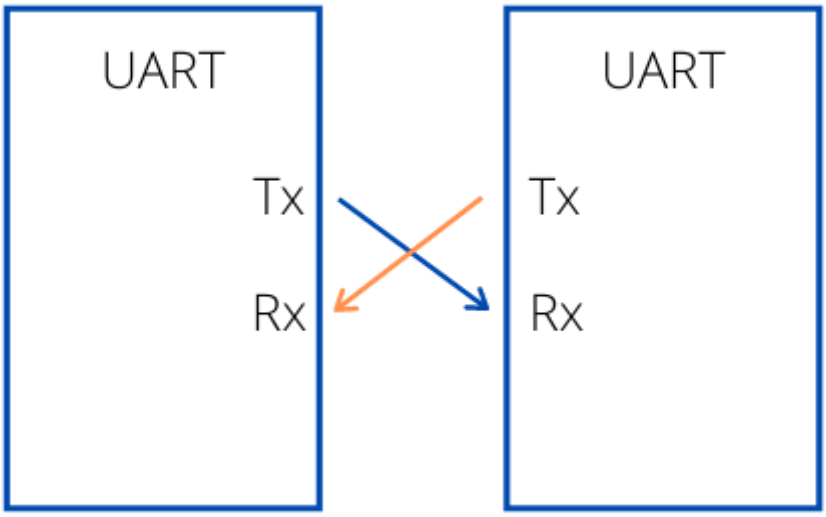
Συνοψίζοντας, τα εξαρτήματα εισόδων, ήχου και εικόνας, επιλέχθηκαν με γνώμονα την υποστήριξη, την απόδοση, το σενάριο χρήσης καθώς και το κόστος. Σημειώνοντας, πως, είναι δυνατή οποιαδήποτε χρήση κάμερας, συμβατής με Raspberry Pi και CSI διεπαφής, όπως και μικροφώνου συμβατό με σύνδεση USB [8].

3.5 GSM HAT



Σχήμα 3.4: SIM800C GSM/GPRS HAT για Raspberry Pi [11].

Το GSM/GPRS HAT Module, όπως φαίνεται στο σχήμα 3.4, κουμπώνει στα GPIO pins του Raspberry Pi λειτουργόντας ως HAT (Hardware Attached on Top). Δηλαδή λειτουργεί ως addon, και παρέχει όλες του τις λειτουργίες μέσω της UART διεπαφής στο Raspberry Pi. Προτού συζητηθεί η συσκευή αυτή περαιτέρω, θα αναφερθούν μερικά λόγια για την διεπαφή UART [12].



Σχήμα 3.5: Ροή πληροφορίας μέσω UART διεπαφών [13].

Η UART (Universal Asynchronous Receiver/Transmitter) διεπαφή χρησιμοποιείται για την σειριακή επικοινωνία μεταξύ συσκευών. Όπως παρατηρείται και στο σχήμα 3.5, χρησιμοποιούνται 2 καλώδια για την αποστολή (Transmit) και λήψη (Receive) σειριακών δεδομένων. Η κάθε συσκευή, μέσω της διεπαφής, μπορεί να λάβει πληροφορία μέσω του καλωδίου RX και να στείλει μέσω του καλωδίου TX, ακόμη και παράλληλα. Χρησιμοποιείται κυρίως την επικοινωνία του Raspberry Pi με Modules όπως GSM/GPRS, GPS και Bluetooth [12].

Η μορφή δεδομένων που στέλνει το Raspberry Pi, στο Module αυτό, αποτελεί την μορφή AT (Attention) εντολών. Χρησιμοποιούνται για να στείλουν εντολές εκτέλεσης στα Modem όπως το GSM/GPRS. Οι εντολές συνιστούν, την αποστολή και ανάγνωση SMS, την παροχή κατάστασης, τον έλεγχο και εισαγωγής PIN της κάρτας SIM καθώς και την διαχείριση φωνητικών κλήσεων όπου υποστηρίζεται [12].

Συνεχίζοντας, το SIM800C GSM/GPRS Module εκμεταλλεύεται αυτή την τεχνολογία για την παροχή υπηρεσιών αποστολής και λήψης μηνυμάτων SMS και κλήσεων τηλεφωνικής επικοινωνίας αλλά και Bluetooth μέσω πρόσθετης κεραίας. Τα GSM/GPRS αποτελούν πρωτόκολλα κινητής επικοινωνίας της γενιάς 2G, μέσω της οποίας και με την χρήση SIM κάρτας γίνεται δυνατή η αποστολή και λήψη SMS μηνυμάτων στο σύστημα, οι δυνατότητες τηλεφωνικών κλήσεων και Bluetooth δεν χρησιμοποιούνται καθώς δεν κρίνονται απαραίτητες, αφού το Bluetooth παρέχεται από το ίδιο το Raspberry Pi [12].

Η αποστολή και λήψη SMS αποτελεί μία ελαφριά διαδικασία, καθώς το μέγεθος πακέτων μεταφοράς είναι μικρό και η συχνότητα αυτής της διαδικασίας θα είναι αρκετά αραιά. Το σύστημα θα στέλνει SMS ως μία μορφή ειδοποιήσεων σε περίπτωση εισβολής, οπότε δεν είναι απαραίτητη η χρήση ταχύτερων τεχνολογιών όπως 3G και 4G. Καθώς η μέγιστη ροή του GPRS, των 85.6kbps κρίνεται αρκετή για αυτού του είδους επικοινωνίας και τα Modem που υποστηρίζουν αυτή την τεχνολογία είναι σαφώς φθηνότερα [12].

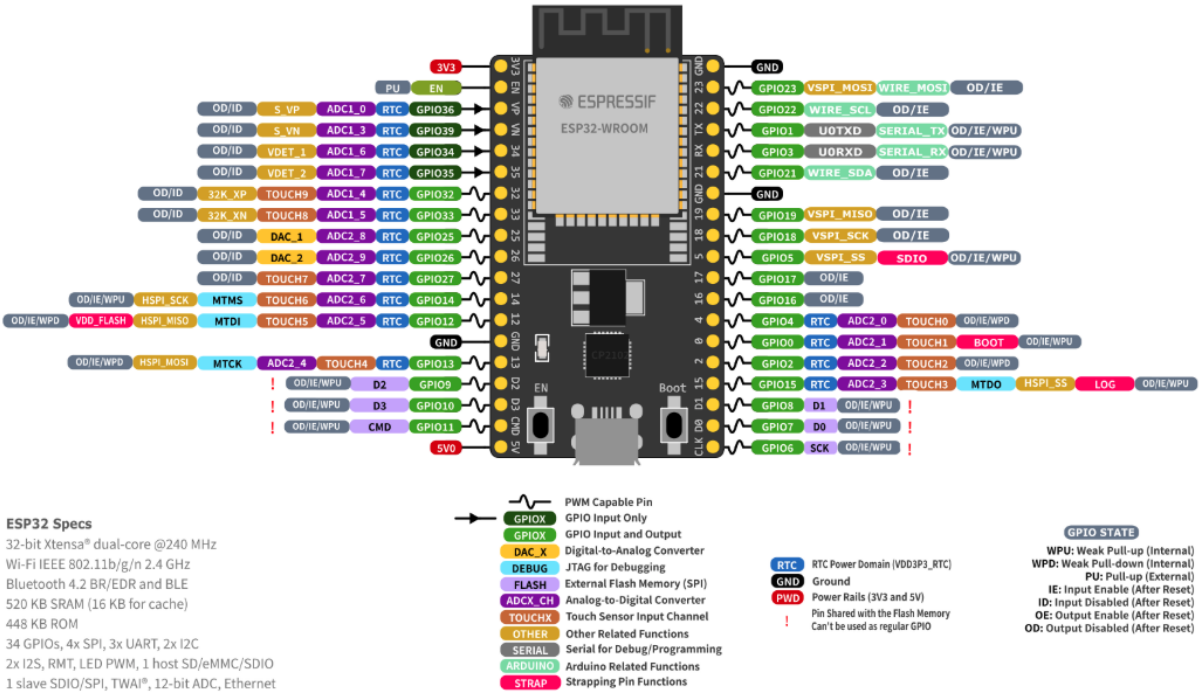
Συνοψίζοντας, με γνώμονα την αποστολή SMS, κρίθηκε ως ελάχιστη απαίτηση, η χρήση του SIM800C GSM/GPRS HAT Module, το οποίο αποτελεί μία φθηνή και επαρκής λύση για την υλοποίηση του συστήματος.

3.6 ESP32

Για να καταγράψει ένα σύστημα παρακολούθησης μια τυχόν επιβολή ή ένα συμβάν, χρειάζεται η ενεργοποίηση του συστήματος κάθε στιγμή της ημέρας. Αυτό έχει ως γεγονός την χρήση περισσότερης ενέργειας και την αποθήκευση μεγάλων αρχείων στον αποθηκευτικό του χώρο. Ως λύση, στο πρόβλημα αυτό, επιλέχθηκε η έξυπνη καταγραφή και το έξυπνο ξύπνημα των περιφερειακών, όπως την κάμερα, το μικρόφωνο και των λειτουργιών έξυπνης αναγνώρισης. Για να ξυπνάει το σύστημα αυτόματα όταν, υπάρχει πιθανότητα εισβολής, επιλέχθηκε η χρήση αισθητήρων όπως κίνησης και ήχου. Το Raspberry Pi υποστηρίζει σύνδεση αισθητήρων μέσω των GPIO pins, όμως αυτό θα ανάγκαζε την μικρή απόσταση μεταξύ του συστήματος κάμερας και αισθητήρων, καθώς η κάμερα συνδέεται με τον μικροϋπολογιστή. Για τον λόγο αυτό, επιλέχθηκε η χρήση ενός μικροελεγκτή, ο οποίος θα καταναλώνει χαμηλή ενέργεια, θα παρέχει δυνατότητα WiFi για την αποστολή δεδομένων, καθώς και GPIO pins για την σύνδεση των αισθητήρων. Παρακάτω θα συζητηθεί και θα αναλυθεί ο μικροελεγκτής που αναφέρθηκε [14][15].

Ο ESP32, είναι ένας, χαμηλού κόστος, ισχυρός μικροελεγκτής SoC (System on Chip), σύστημα σε ένα Chip. Έχει ενσωματωμένες λειτουργίες WiFi και Bluetooth. Προορίζεται για αντικαταστάτης του ESP8266 λόγω της περισσότερης υπολογιστικής του ισχύς. Χρησιμοποιείται κυρίως σε εφαρμογές IOT, για τους προαναφέρον λόγος και τον αριθμό των GPIO pin να φτάνει τα 36, υποστηρίζοντας την σύνδεση πολλών συσκευών όπως αισθητήρες [14].

Για την χρήση του μικροελεγκτή, απαιτείται ο προγραμματισμός του μέσω του Arduino IDE, σε γλώσσα παρόμοια της C. Επιπλέον η εταιρεία ανάπτυξης του υλικού αυτού, Espressif, προσφέρει μία πληθώρα βιβλιοθηκών για το περιβάλλον αυτό, για την καλύτερη διαχείριση των λειτουργιών του ESP32, όπως τα GPIO pins, το WiFi και το Bluetooth.



Σχήμα 3.6: ESP WROOM-32, μαζί με τους τύπους των διαθέσιμων GPIO pins [16].

Συγκεκριμένα, για το σύστημα, έγινε χρήση του ESP WROOM-32, αποτελεί μία έκδοση του ESP32 ειδικό για την ανάπτυξη καθώς περιέχει και ενσωματωμένα περιφερειακά, εκτός το ESP32 Chip, όπως USB Type-C και GPIO pins. Αποτελεί την πιο συνηθισμένη επιλογή για IOT υλοποιήσεις, καθώς εξισορροπεί καλά την σχέση κόστος-απόδοσης που προσφέρει. Προσφέρει μνήμη flash των 4MB και επεξεργαστή των 240MHz, που είναι αρκετά για τις απαιτήσεις του συστήματος, οι οποίες προϋποθέτουν την χρήση ενός αισθητήρα κίνησης και ενός προαιρετικού αισθητήρα έντασης ήχου [17].

Επιπλέον, πληροί τις προϋποθέσεις για την αποδοτικότητα της ενέργειας. Παρέχει λειτουργίες ύπνου, όπου ανάλογα με την διαβάθμιση, απενεργοποιεί διάφορες λειτουργίες του με αποτέλεσμα την ελαχιστοποίηση της χρήσης ρεύματος, φτάνοντας στα μερικά μA. Συγκεκριμένα με την λειτουργία Deep Sleep (βαθύς ύπνος), σβήνουν όλες οι λειτουργίες εκτός από τα RTC (Real Time Clock) και ULP Co-Processor. Το πρώτο αποτελεί ένα αργό ρολόι με μία μικρού μεγέθους μνήμη, και το δεύτερο αποτελεί έναν επεξεργαστή χαμηλής ισχύος ειδικός για την λειτουργία αυτή. Μέσω των λειτουργιών που παραμένουν ενεργοποιημένες, είναι δυνατό το ξύπνημα του ESP32 μέσω παροχής σήματος σε συγκεκριμένα pins (RTC Pins) αλλά και μέσω σταθερού ρυθμιζόμενου χρονόμετρου. Συνεπώς, ο μικροελεγκτής μπορεί να παραμείνει σε κατάσταση ύπνου που θα ξυπνήσει μόνο όταν λάβει σήμα από τους συνδεδεμένους αισθητήρες [15][17].

Συνοψίζοντας, αποτελεί μία από τις φθηνότερες επιλογές, παρέχοντας καλή απόδοση ενέργειας, μεγάλη ισχύ και μεγάλη συνδεσιμότητα μέσω των 34 GPIO pins, ξεπερνώντας παρόμοιες εναλλακτικές όπως το Arduino Uno.

3.7 Αισθητήρες

Όπως προαναφέρθηκε στην ενότητα του μικροελεγκτή ESP32, οι αισθητήρες χρησιμοποιούνται ως μέσω αποδοτικότερης χρήσης της ενέργειας του συστήματος. Το σύστημα βασίζεται στην χρήση τουλάχιστον ενός αισθητήρα κίνησης και στην προαιρετική χρήση αισθητήρων ήχου. Στις επόμενες υπο ενότητες θα αναφερθεί η επιλογή αυτών των δύο εξαρτημάτων.

3.7.1 Αισθητήρας κίνησης

Σχήμα 3.7: Αισθητήρας κίνησης υπέρυθρης θερμότητας [18].

Για τον αισθητήρα κίνησης, μπορεί να επιλεχθεί ένας απλός, που έχει μικρή κατανάλωση ρεύματος και χαμηλό κόστος αρκεί να είναι συμβατός για την σύνδεση του σε GPIO pins. Συγκεκριμένα, επιλέχθηκε ο Gravity Digital Infrared Motion Sensor For Arduino, της DFRobot, καθώς προορίζεται για Arduino, δουλεύει τέλεια και με ESP32 αλλά ακόμα και με το Raspberry Pi. Ο αισθητήρας αυτός, μπορεί να καταγράφει κίνηση μέσω της υπέρυθρης ζέστης που εκπέμπεται από διάφορα σώματα, όπως το ανθρώπινο και των ζώων. Καθιστώντας το ιδανικό για τις ανάγκες του συστήματος. Παρακάτω αναφέρεται η ροή λειτουργίας του αισθητήρα:

1) Με την ενεργοποίηση, τα πρώτα 1-2 δευτερόλεπτα, αποκτά ένα στιγμιότυπο του χώρου.
2) Μόλις αισθανθεί υπέρυθρη θερμότητα, συγκρίνει με το αρχικό στιγμιότυπο.
3) Μετά την σύγκριση, στην περίπτωση διαφοράς, εξάγει 4V ως σήμα εξόδου, αλλιώς συνεχίζει την παροχή 0V.

Μερικά από τα κύρια χαρακτηριστικά του αποτελούν:

1) Η γωνία αίσθησης είναι 110 μοιρών και η απόσταση 7 μέτρα, αρκετά για την κάλυψη εξόδων.
2) Η χρήση ρεύματος ανέρχεται στα 50μA, αποτελώντας σημαντική διαφορά κατανάλωσης της ενέργειας για λειτουργίες έξυπνου ξυπνήματος όπως προαναφέρθηκε.

Συνοψίζοντας, η επιλογή αυτή αποτελεί μία πολύ αποδοτική, γρήγορη, φθηνή και αξιόπιστη λύση για την ενημέρωση του μικροελεγκτή στην οποιαδήποτε κίνηση. Τέλος, λόγω του χαμηλού κόστος και της απλής χρήσης, είναι εύκολη η ενσωμάτωση πολλών ίδιων αισθητήρων, για την μεγαλύτερη κάλυψη του χώρο.

3.7.2 Αισθητήρας ήχου

Σχήμα 3.8: Αισθητήρας ήχου LM386 της WaveShare [19].

Αντίστοιχη λογική επιλογής αισθητήρα κίνησης, αποτελεί και η επιλογή αισθητήρα ήχου. Πιο ειδικά, επιλέχθηκε το μοντέλο LM386 Sound Sensor της WaveShare, το οποίο είναι συμβατό με GPIO pins. Αποτελεί μία χαμηλού κόστος λύση για την αίσθηση περιβαντολλογικού θορύβου. Η ροή λειτουργίας του αισθητήρα είναι:

1) Το ενσωματωμένο μικρόφωνο καταγράφει θόρυβο ως σήμα.
2) Το σήμα περνάει από τον ενσωματωμένο LM386 ενισχυτή ήχου, με δυνατότητα ενίσχυσης κέρδους έως 200.
3) Εξάγει το σήμα σε ενισχυμένη μορφή μέσω αναλογικής μορφής, είτε μέσω ψηφιακής μορφής ως τάση με γνώμονα την ένταση του ήχου.

Επιπλέον, χαρακτηριστικά του αισθητήρα συνιστούν τα ακόλουθα:

1) 2 τύποι εξόδων, αναλογική και ψηφιακή. Η ψηφιακή εξάγει 0 ή 1, η χαμηλή έξοδος προσδιορίζει πως ο θόρυβος είναι μεγαλύτερος του ορισμένου κατωφλίου και η υψηλή το ακριβώς αντίθετο. Η αναλογική έξοδος εξάγει ένα εύρος τιμών από 0 έως 4095, λόγω του 12-bit μεγέθους που παρέχει ο ESP32.
2) Η ευαισθησία του ήχου κυμαίνεται στα 52db, με αποτέλεσμα να είναι δυνατή η καταγραφή πολύ χαμηλών ήχων, όπως τον ψίθυρο.
3) Καταγραφή θορύβου από τα 50Hz έως τα 20KHz. Συμπεριλαμβάνει ένα μεγάλο εύρος των σημαντικότερων ήχων όπως την ομιλία ανθρώπων, ήχους ζώων και διάφορους συναγερμούς.

Τέλος, ο αισθητήρας της WaveShare, είναι μία συνετή επιλογή, πληρώντας πλήρως τις ανάγκες και απαιτήσεις του συστήματος να καταγράφει την ένταση θορύβων προσφέροντας καλή σχέση μεταξύ κόστος-ποιότητας.

3.8 Συνολικό κόστος υλοποίησης

Η επιλογή υλικών, όπως συζητήθηκε και στα προηγούμενα κεφάλαια, έγινε με γνώμονα την τήρηση των ελάχιστων απαραίτητων απαιτήσεων του συστήματος, για την ελαχιστοποίηση του κόστους υλοποίησης. Στον πίνακα 3.1, γίνεται αναφορά στο κόστος του κάθε εξαρτήματος που χρησιμοποιήθηκε, οι τιμές αποτελούν ενδεικτικές τιμές λιανικής πώλησης.

Πίνακας 3.1: Κόστος υλικών, Μάιος 2025.

Εξάρτημα	Μοντέλο	Προσεγγιστικό κόστος (€)
Raspberry Pi 5	8GB RAM	96
Raspberry Pi 5 Power Supply	USB-C 27W	14
Active Cooler	Raspberry Pi 5	6
Raspberry Pi MicroSD Card	64GB	11
Raspberry Pi Camera	V3 - NoIR 11.9MP 75°	34
USB Microphone	-	6
GSM/GPRS HAT	Waveshare SIM800C	29
ESP32	NodeMCU-32S	8
DFRobot Motion Sensor	Infrared Motion Sensor	5
Waveshare Sound Sensor	LM386 Sound Sensor	4

Κεφάλαιο 4ο: Τεχνολογίες και πρωτόκολλα

4.1 Εισαγωγή

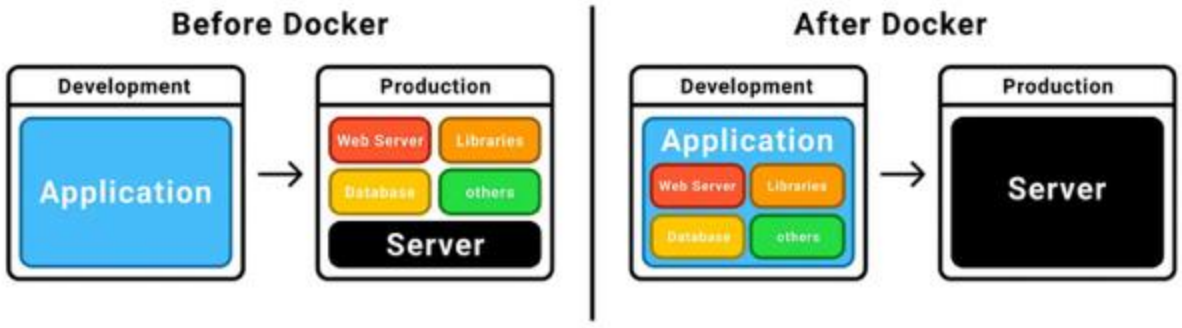
Η υλοποίηση ενός συστήματος έξυπνης παρακολούθησης, που συνδυάζει αρκετές πτυχές της τεχνολογίας, απαιτεί την προσεκτική επιλογή των εργαλείων και τεχνολογιών που θα το απαρτίζουν. Όπως προαναφέρθηκε, γίνεται συνδυασμός πολλών υπηρεσιών, όπου η καθεμία βασίζεται στις υπόλοιπες, καθιστώντας την ορθή επιλογή, κρίσιμη για την σωστή λειτουργία του συστήματος. Στο παρόν κεφάλαιο, θα αναφερθούν αρχικά, τα εργαλεία που χρησιμοποιήθηκαν ευρέως. Στην πορεία, θα αναλυθούν, οι τεχνολογίες που εφαρμόστηκαν για την κατασκευή των υπηρεσιών Backend, της Web ιστοσελίδας και της Mobile εφαρμογής. Τέλος, θα γίνει επέκταση στα πρωτόκολλα επικοινωνίας που λειτουργούν σαν συνδετικός κρίκος ολόκληρου του συστήματος.

4.2 Εργαλεία και κοινές τεχνολογίες ανάπτυξης

4.2.1 Docker

Το Docker είναι μία πλατφόρμα που αυτοματοποιει τις διαδικασίες ανάπτυξης, εκτέλεσης και διάθεσης εφαρμογών. Το Docker πρώτο εμφανίστηκε το 2013, ενώ στα επόμενα χρόνια ξεκίνησε να συγκεντρώνει την εμπιστοσύνη της κοινότητας, και πλέον αποτελεί την πιο δημοφιλής επιλογή για την παραγωγή συστημάτων [18].

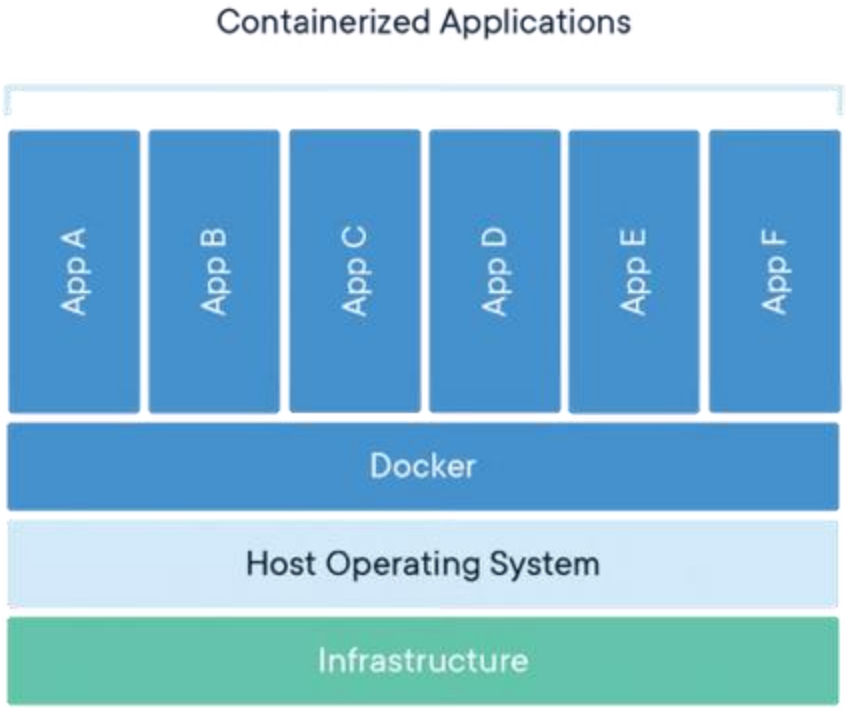
Προτού εμφανιστεί το Docker, η διαχείριση και η ανάπτυξη εφαρμογών ήταν μια εξαιρετικά πολύπλοκη διαδικασία. Όχι μόνο, ο προγραμματιστής έπρεπε να χρησιμοποιήσει πολλά εργαλεία για να πετύχει τον ίδιο σκοπό αλλά και πολλές φορές δεν είχε την ικανότητα να το πετύχει. Μερικές πρακτικές αποτελούσαν τις εικονικές μηχανές, εργαλεία αυτοματοποίησης των παραμέτρων του συστήματος, διάφορες βιβλιοθήκες και πλατφόρμες ως Υπηρεσία (PaaS). Τα προβλήματα αυτών των λύσεων, αποτελούν, οι ανάγκες πόρων που απαιτεί η εκτέλεση εικονικών μηχανών, καθώς αποτελούν ολόκληρο λειτουργικό σύστημα μαζί με εικονικό περιβάλλον μέσα σε ένα ήδη υπάρχων σύστημα, με αποτέλεσμα η ανάπτυξη μιας περίπλοκης υπηρεσίας να απαιτεί αρκετές εικονικές μηχανές. Επιπλέον ο προγραμματιστής έπρεπε να αναπτύξει κώδικα ή να πληρώσει υπηρεσίες για την αυτόματη ρύθμιση των υπηρεσιών, γεγονός που ανεβάζει το κόστος και τον χρόνο ανάπτυξης [18][19].



Σχήμα 4.1: Ανάπτυξη χωρίς Docker vs ανάπτυξη με την χρήση Docker [20].

Το Docker, ήρθε την κατάλληλη στιγμή με την κατάλληλη λύση, καθώς λύνει όλα αυτά τα προβλήματα και ενσωματώνει όλες τις λύσεις σε ένα εργαλείο. Αυτό που το καθιστά ξεχωριστό είναι, η ευκολία χρήσης, η αποδοτικότητα και η γρηγοράδα. Αρχικά χωρίζει την κάθε υπηρεσία ενός συστήματος στο δικό της Container, δηλαδή στο δικό της εικονικό περιβάλλον. Το κάθε container περιέχει τα απολύτως

απαραίτητα για να τρέξει η κάθε αυτή μικρο-υπηρεσία, καθώς και καθιστά συγκεκριμένες εκδόσεις του κάθε dependency, με αποτέλεσμα η κάθε υπηρεσία να γίνεται συμβατή με οποιοδήποτε μηχάνημα που υποστηρίζει τον πυρήνα του Linux Kernel, ακόμα και στα Windows μέσω του WSL [19].



Σχήμα 4.2: Αρχιτεκτονική Docker Containers [21].

Την διαδικασία αυτή την επιτυγχάνει μέσω του Docker Engine, το οποίο λειτουργεί ακριβώς πάνω από το λειτουργικό σύστημα. Διαχειρίζεται όλες τις λειτουργίες των Containers. όπως την δημιουργία και την εκτέλεση τους, και δίνονται ως εντολές είτε μέσω του Dockerfile είτε μέσω του Docker CLI. Για κάθε υπηρεσία δημιουργείται μία Εικόνα (Image) η οποία συνιστά το σχέδιο της υπηρεσίας, π.χ. όπως η κλάση σε μία αντικειμενοστραφή γλώσσα προγραμματισμού. Στο Image δηλώνονται όλες οι ενέργειες, οι εκδόσεις και τα απαραίτητα που χρειάζεται μία υπηρεσία για να λειτουργήσει. Μέσω της Εικόνας, το Docker Engine δημιουργεί το Container, που αποτελεί την εκτελέσιμη μορφή του Image, όπως το αντικείμενο μιας κλάσης σε μία αντικειμενοστραφή γλώσσα. Τέλος η Μηχανή Docker, παρέχει δυνατότητες αποθήκευσης των δεδομένων του κάθε Container, καθώς και δυνατότητες δικτύωσης, καθώς σε ένα σύστημα οι υπηρεσίες συχνά, χρειάζονται να επικοινωνήσουν μεταξύ τους [19].

Για την ευκολία στην διαχείριση συστημάτων που αποτελούνται από πολλές υπηρεσίες, μπορεί να χρησιμοποιηθεί το Docker Compose, το οποίο μέσω του αρχείο docker-compose.yml δηλώνονται όλες οι μικρο-υπηρεσίες αναλυτικά, το πως αποθηκεύονται, την δικτύωση και τις ανάγκες που απαιτούν.

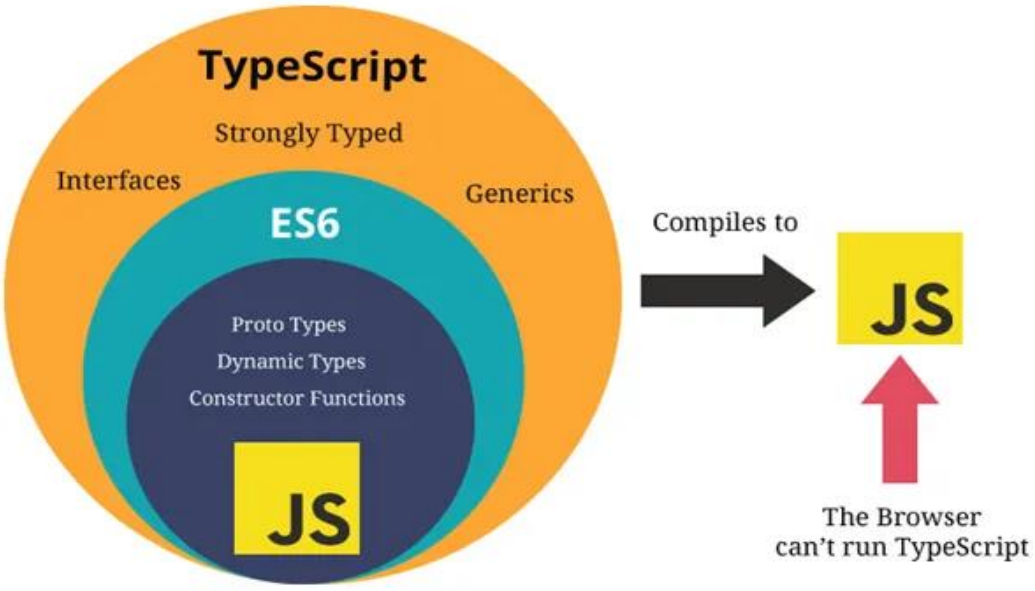
4.2.2 TypeScript

Η TypeScript είναι μία προέκταση της γλώσσας προγραμματισμού JavaScript. Με σκοπό την διευκόλυνση του προγραμματιστή στην ανάπτυξη large-scale εφαρμογών [22].

Αρχικά η JavaScript, είναι μία δυναμικού τύπου interpreted γλώσσα προγραμματισμού και ευρέως διαδεδομένη με το μεγαλύτερο ποσοστό χρήσης της να προορίζεται από τον Browser. Σε κάθε ιστοσελίδα εκτελείται κώδικας JavaScript για να κάνει το περιεχόμενο που εμφανίζεται στον χρήστη διαδραστικό. Περαιτέρω, μπορεί να χρησιμοποιηθεί για την ανάπτυξη Backend συστημάτων, Web παιχνιδιών και την εκτέλεση και την εκπαίδευση μοντέλων μηχανικής μάθησης.

Η JavaScript στον Browser εκτελείται μέσω των Browser Engines, χρησιμοποιώντας το μοντέλο DOM για την παραγωγή ιστοσελίδας, με τις πιο καθιερωμένες, την V8 της Google για το Google Chrome και την JavaScriptCore της Apple για το Safari. Καθώς υπάρχουν πολλές μηχανές για την μετάφραση της JavaScript σε γλώσσα μηχανής, και την αποφυγή απαίτησης διαφορετικού κώδικα από τον προγραμματιστή για κάθε διαφορετική μηχανή, υπάρχει το πρότυπο ECMAScript. Το το πρότυπο αυτό έχει σκοπό την διαλειτουργικότητα της JavaScript μεταξύ διαφορετικών Web Browsers, αλλά χρησιμοποιείται και ως πρότυπο για server-side εφαρμογές [23][24].

Επιπλέον η JavaScript, μπορεί να εκτελεστεί εκτός περιβάλλοντος Browser, για την δημιουργία server-side εφαρμογών, μέσω περιβαλλόντων όπως το Nodejs. Το Nodejs είναι γραμμένο στην γλώσσα C/C++ και ακολουθεί το μοντέλο της μηχανής V8 της Google [23].



Σχήμα 4.3: Lifecycle bubble προγράμματος γραμμένο σε TypeScript [25].

Η ελαστικότητα και δυναμικότητα που προσφέρει η JavaScript είναι αυτό που την κάνει να ξεχωρίζει, όμως υπάρχουν πολλά προβλήματα όσον αφορά την ανάπτυξη και διατήρηση μεγάλων βελησνεκούς εφαρμογών. Μεγαλύτερο και βασικότερο πρόβλημα αποτελεί η έλλειψη στατικών τύπων, έτσι ο προγραμματιστής δεν γνωρίζει ολοκληρωτικά την μορφή μιας μεταβλητής με αποτέλεσμα την εισαγωγή bugs. Το πρόβλημα αυτό αντιμετωπίζει με επιτυχία η TypeScript [23].

Κάθε πρόγραμμα TypeScript αποτελεί ένα πρόγραμμα JavaScript, μέσω του TypeScript compiler ο κώδικας μετατρέπεται στην μορφή της JavaScript έτσι ώστε αργότερα να τρέξει στην προοριζόμενη μηχανή. Συνεπώς, η επέκταση αυτή χρησιμοποιείται μόνο από τον προγραμματιστή κατά την ανάπτυξη, και δεν φαίνεται καθόλου στον τελικό χρήστη, με αποτέλεσμα το τελικό πρόγραμμα να πετύχει τον ίδιο χρόνο εκτέλεσης και με σημαντική μείωση των πιθανών σφαλμάτων [23].

4.2.3 FFmpeg

Το FFmpeg, είναι ένα ανοιχτού κώδικα, multimedia framework, με την υποστήριξη των περισσότερων κωδικοποιήσεων Video και ήχου, πρωτοκόλλων επικοινωνίας και των Media formats. Για παράδειγμα, το FFmpeg μπορεί να πάρει ένα αρχείο βίντεο, που περιέχει ήχο, σε MP3 μορφή, να εισάγει διάφορες κωδικοποιήσεις και φίλτρα και να το εξάγει σε μία ζωντανή ροή σε έναν server μέσω RTMP. Η ικανότητα αυτή το καθιστά ένα από τα καθιερωμένα εργαλεία για Video και Audio streaming [26].

Κάνει χρήση των Codecs, τεχνολογία υπεύθυνη για την αποκωδικοποίηση και κωδικοποίηση των πολυμέσων (Media) για εφαρμογή διαφόρων λειτουργιών, με την πιο συνηθισμένη να αποτελείται η συμπίεση. Υπάρχει η δυνατότητα μετατροπής ενός Codec σε ένα άλλο, μέσω του Transcoding. Το πιο συνηθισμένο codec απαρτίζει το h.256, το συμπιέζει το βίντεο και καταφέρνει την διατήρηση της ποιότητας [27].

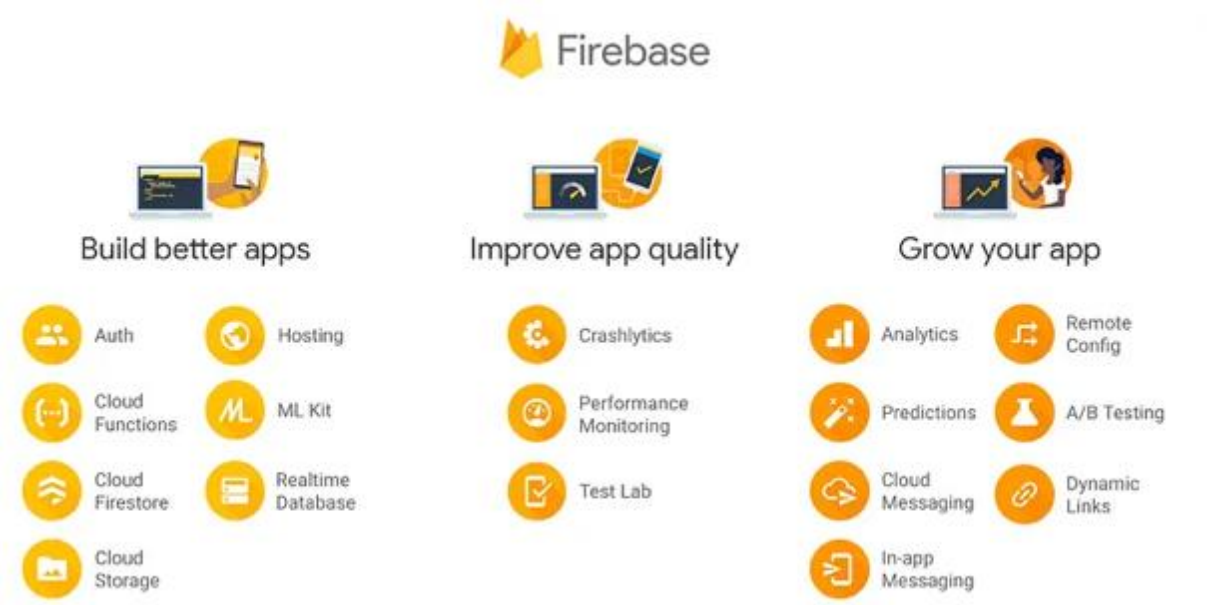
Επιπλέον έχει την δυνατότητα μετατροπής του Container ενός Media αρχείου σε ένα άλλο. Το Container, παρέχει πληροφορίες για την δομή ενός αρχείου, μερικά παραδείγματα Container για Video αποτελούν τα MP4, MOV και WAV για τον ήχο. Η δυνατότητα αυτή αποκαλείται Transmuxing [27].

Όπως προαναφέρθηκε εκτός τις ικανότητες μετατροπής Codec και Container, το εργαλείο FFmpeg μπορεί να εισάγει διάφορα φίλτρα. Μερικά από τα φίλτρα αποτελούν, μετατροπή του Bitrate, μετατροπή των FPS, εισαγωγή και εξαγωγή υπότιτλων και timestamp και μετατροπή ποιότητας και ανάλυσης του ήχου και της εικόνας [26][27].

4.2.4 Firebase

Το Firebase, είναι μία ολοκληρωμένη πλατφόρμα ανάπτυξης υπηρεσιών, όπως full-stack Web και Mobile εφαρμογών. Επιπλέον παρέχει εργαλεία για την βελτίωση της ποιότητας των εφαρμογών. Με άλλα λόγια, λειτουργεί ως Backend-as-a-Service (Baas). Αναπτύχθηκε και συνεχίζει να αναπτύσσεται από την Google, με στόχο να επιτρέπει στους προγραμματιστές να εστιάσουν στα κύρια χαρακτηριστικά της υπο ανάπτυξης εφαρμογής, αναλαμβάνοντας πολλές συχνά αναπτυσσόμενες λειτουργίες [28].

Εκτός από την διευκόλυνση του προγραμματιστή στην γρηγορότερη δημιουργία εφαρμογών, αποτελεί ένα γρήγορο και ασφαλές σύστημα, κάνοντας την αξιόπιστη και διάσημη τεχνολογία στην κοινότητα. Παρόμοιες υπηρεσίες είναι η AWS της Amazon και η ανερχόμενη Supabase. Τα χαρακτηριστικό που το διαφοροποιεί με τις υπόλοιπες επιλογές προϋποθέτουν την ταχύτητα, την ασφάλεια, την ευκολία και φυσικά την χαμηλή έως καθόλου κόστους χρέωσης υπηρεσιών. Με τις πιο σημαντικές υπηρεσίες να αποτελούν, το Firebase Authentication, Firestore, Firebase Cloud Storage και Firebase Cloud Messaging και άλλα. Καθώς και προσφέρει ποικίλα εργαλεία για την βελτίωση της εμπειρίας του χρήστη, όπως τα Google Analytics, Crashlytics και Firebase Hosting. Συνεπώς, παρέχεται στον προγραμματιστή ένα μία πλήρες υπηρεσία στην οποία χρειάζεται η ενσωμάτωση της με το front end [28].



Σχήμα 4.4: Λίστα υπηρεσιών της πλατφόρμας Firebase [29].

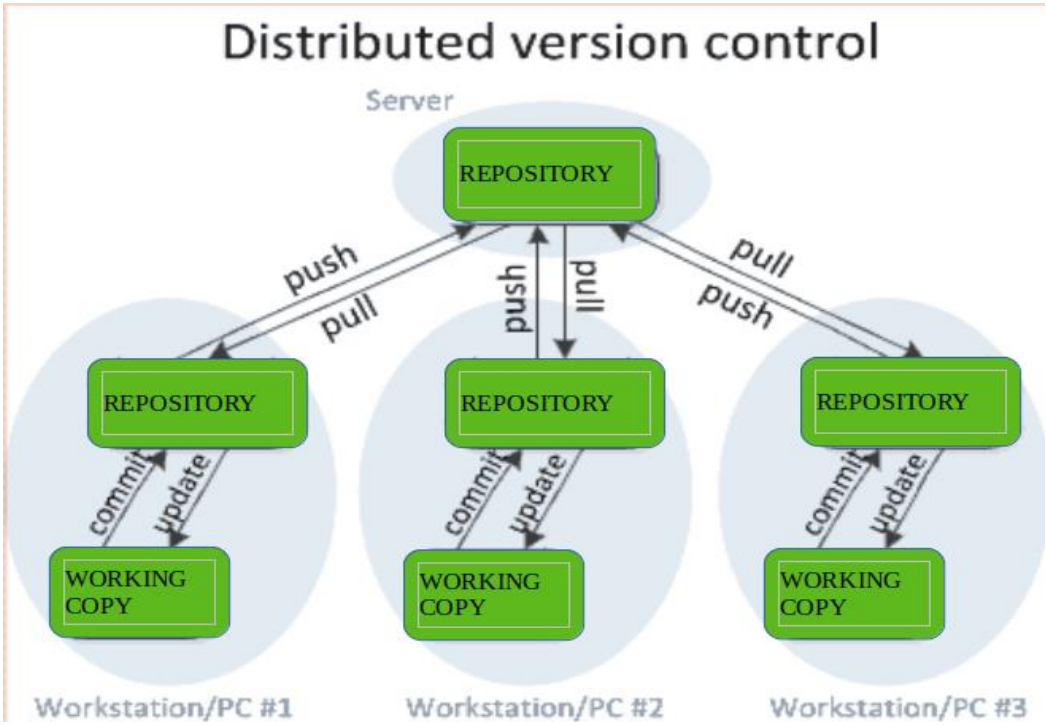
Το Firebase Cloud Messaging (FCM), συνιστάται ως η κύρια επιλογή αποστολής Push Notifications σε ένα Android κινητό. Push Notifications ονομάζονται οι ειδοποιήσεις που εμφανίζονται λόγω αιτήματος από απομακρυσμένο server. Καθώς η εμφάνιση ειδοποιήσεων αποτελεί Native χαρακτηριστικό του Android, κάνοντας το διαθέσιμο σε όλους τους προγραμματιστές εφαρμογών, υπάρχει ένας περιορισμός στην περίπτωση που η εφαρμογή είναι τελείως κλειστή. Στο σενάριο αυτό, καθώς η εφαρμογή δεν μπορεί να λάβει σήμα από έναν απομακρυσμένο server για την εμφανιση ειδοποιήσεων, χρησιμοποιείται για την αποστολή αυτή το Cloud Messaging του Firebase. Η υπηρεσία αυτή εκμεταλλεύεται τα Google Play Services, τα οποία υπάρχουν σε όλα τα Android ως εφαρμογή συστήματος, δηλαδή έχουν πλήρη εξουσιοδότηση στο λειτουργικό σύστημα με αποτέλεσμα την εμφάνιση ειδοποιήσεων και στην περίπτωση που η τυχόν εφαρμογή είναι κλειστή. Παρόμοια λειτουργία, επικρατεί στο iOS μέσω της συνεργασίας με τα APNs (Apple Push Notification Service). Συνοψίζοντας, για την αποστολή ειδοποιήσεων σε μία συσκευή είναι απαραίτητο το FCM token, μοναδικό για κάθε συσκευή που δημιουργείται κατά την φόρτωση της εφαρμογής [28][30].

Η υπηρεσία Firebase Authentication, προσφέρει μία ασφαλή και γρήγορη υλοποίηση αυθεντικοποίησης χρηστών σε μία εφαρμογή. Ειδικά, υποστηρίζει τις περισσότερες μορφές Login, όπως την κλασική σύνδεση μέσω email και password και την σύνδεση μέσω τηλεφώνου. Επιπροσθέτως, υποστηρίζει one-click συνδέσεις μέσω τρίτων λογαριασμών από παρόχους σαν τις Google, Apple, Facebook, Github, Twitter, Microsft και άλλα. Η υλοποίηση μιας μεθόδου σύνδεσης αποτελεί μερικά click από τον προγραμματιστή και την εγκατάστασης της κύριας βιβλιοθήκη firebase που υποστηρίζεται από τις περισσότερες γλώσσες προγραμματισμού. Αξίζει να σημειωθεί η ευκολία ταυτοποίησης χρηστών από ένα σύστημα Backend, όπου με την αποστολή κάθε αιτήματος γίνεται η αποστολή και μίας επικεφαλίδας Header, του HTTP, και στην συνέχεια με την κλίση μιας συνάρτησης παρέχονται οι πληροφορίες του χρήστη [28][31].

Τέλος, το Firebase περιέχει ακόμα πολλές χρήσιμες υπηρεσίες, η κάθε μία σημαντική και με δικό της ενδιαφέρον. Καθώς για την ανάλυση όλων των υπηρεσιών Firebase χρειάζεται εκτενής αναφορά, κρίνεται εκτός του σκοπού αυτής της ενότητας και συνεπώς δεν περιγράφονται περαιτέρω [28].

4.2.5 Git

Η τεχνολογία Git, είναι ένα κατανεμημένο σύστημα version control (DVCS). Βοηθάει συνεχή συνεργασία προγραμματιστών και κρατάει ιστορικό της κάθε αλλαγής σε ένα source code καθώς και τις διαφορετικές παραλλαγές του. Δημιουργήθηκε το 2005, σαν ένα μέσο διαχείρισης της ανάπτυξης του Linux Kernel και από τότε έχει καθιερωθεί ως την κύρια επιλογή για version control [32].



Σχήμα 4.5: Διάγραμμα ροής Distributed Version Control [33].

Το Git λειτουργεί σαν ένα κατανεμημένο σύστημα, δηλαδή κάθε προγραμματιστής έχει μία αντιγραφή του κώδικα, από το repository στο οποίο είναι δημοσιευμένο, στο τοπικό του μηχάνημα. Μπορεί να επιβάλλει προσθήκες και τροποποιήσεις και στην συνέχεια να δημιουργήσει μία καινούρια έκδοση του κώδικα που λέγεται commit και να το δημοσιεύσει. Το λογισμικό υπό ανάπτυξη μπορεί να χωριστεί σε διάφορα branches στο οποίο μπορούν να δουλεύουν ένα ή περισσότερα άτομα, όπου ανάλογα με την κάθε πρακτική, το κάθε branch αποτελεί ένα χαρακτηριστικό του λογισμικού. Κατά το τέλος της ανάπτυξης όλα τα branch συγχωνεύονται (merge) για να παραχθεί το τελικό προϊόν. Περαιτέρω, ο προγραμματιστής μπορεί να πειραματιστεί στο τοπικό repository, όπου κάθε commit και merge γίνονται τοπικά και στην πορεία να δημοσιεύει αυτές τις αλλαγές στο απομακρυσμένο (remote) repository, όπου συνήθως χρησιμοποιείται για την παραγωγή του λογισμικού [32].

Η παροχή του ιστορικού ενός source code, παρουσιάζει σπουδαια πλεονεκτήματα. Κατά την πρόκυψη σφαλμάτων διευκολύνεται σημαντικά η επαναφορά σε προηγούμενη λειτουργική κατάσταση, με την χρήση μερικών commands, μειόνοντας αρκετά την απώλεια χρόνου και την εύρεση του λάθους. Επιπροσθέτως, γίνεται εύκολη η εγκατάλειψη των αλλαγών στην περίπτωση ακύρωσης ενός χαρακτηριστικού της εφαρμογής αλλά και η αποθήκευση κομματιών κώδικα για την αργότερη χρήση τους [32].

4.3 Τεχνολογίες Backend

4.3.1 Go

Η Go, είναι μία γλώσσα ανοιχτού λογισμικού, open source, που δημιουργήθηκε και δημοσιεύτηκε από την Google το 2009. Αναπτύχθηκε να είναι μία απλή, γρήγορη, αποδοτική, αξιόπιστη και εύκολη στην συντήρηση γλώσσα προγραμματισμού. Χρησιμοποιείται κυρίως σε εφαρμογές Cloud, όπως backend servers και για δημιουργία εργαλείων συστήματος, με το πιο πρόσφατο παράδειγμα, η χρήση της για την ανάπτυξη της TypeScript [34].

Η γλώσσα αυτή, έχει θέση, ανάμεσα στις γρηγορότερες γλώσσες, χωρίς να θυσιάζει την ταχύτητα και ευκολία ανάπτυξης που συναντάται σε γλώσσες όπως οι C/C++ και Rust αντιστοίχως. Μερικά από τα πλεονεκτήματα της Go αποτελούν [34]:

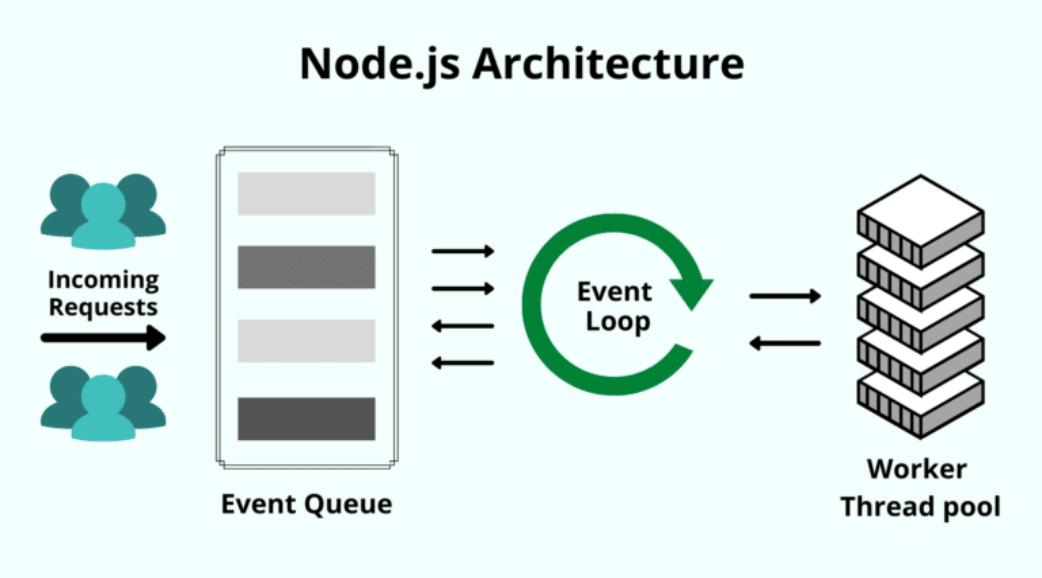
1. Μοντέρνα γλώσσα, η οποία χρησιμοποιεί μοντέρνο συντακτικό και είναι σύμφωνη με σύγχρονες πρακτικές. Καθιστώντας την, εύκολη στην ανάγνωση και στην κατανόηση.
2. Υποστηρίζει Garbage collection, διαχειρίζεται την μνήμη αυτόματα και δίνει την δυνατότητα χρήσης pointers από τον προγραμματιστή.
3. Η Go, έχει πλούσια standard βιβλιοθήκη, είναι αρκετά μειωμένη η ανάγκη για εγκατάσταση εξωτερικών βιβλιοθηκών, καθώς υπάρχουν τα περισσότερα στο βασικό πακέτο βιβλιοθηκών της γλώσσας.
4. Έχει παράλληλα υποστήριξη για, procedural, concurrent και distributed προγραμματισμό.
5. Ο παραλληλισμός εργασιών μέσω Go Routines και η διαχείριση της μνήμης τους μέσω των Channels, είναι αρκετά απλή και σύντομη διαδικασία που γίνεται μόνο με την χρήση ενός keyword.
6. Είναι στατικού τύπου, προσφέροντας ασφάλεια τύπων κατά την μεταγλώττιση.
7. Παρέχει built in μηχανισμούς για την ανάπτυξη κώδικα καθαρού και συντηρίσιμου κώδικα.

Όπως όλες οι γλώσσες, έτσι και η Go, πάσχει από αδυναμίες. Με τις πιο σημαντικές να αποτελούν η έλλειψη κλάσεων, η περιορισμένη υποστήριξη για δημιουργία γραφικών διεπαφών GUI καθώς και ο μικρός αριθμός διαθέσιμων βιβλιοθηκών [34].

4.3.2 Nodejs

Το Nodejs, είναι ένα εκτελέσιμο περιβάλλον JavaScript. Επιτρέπει την εκτέλεση κώδικα JavaScript χωρίς την χρήση Browser. Κυρίως χρησιμοποιείται για την δημιουργία full stack εφαρμογών χωρίς την χρήση επιπλέον γλωσσών, για την ανάπτυξη του backend [35].

Αποτελεί τα θεμέλια, για όλες τις σύγχρονες τεχνικές Web development, καθώς τα Web frameworks και οι βιβλιοθήκες, όπως η React, Angular, Nextjs βασίζονται πάνω σε αυτό. Αντικαταστώντας όλο και περισσότερο την χρήση PHP, αφού είναι συγκριτικά γρηγορότερο τόσο στην απόδοση όσο και στην ανάπτυξη. Χρησιμοποιείται για Web development από τις περισσότερες πλατφόρμες όπως την Netflix και Paypal.



Σχήμα 4.6: Σχήμα αρχιτεκτονικής διαχείρισης αιτημάτων δικτύου του Nodejs [36].

Είναι βασισμένο, όπως έχει προαναφερθεί, στην μηχανή V8 της Google. Η μηχανή αυτή, μετατρέπει τον JavaScript κώδικα σε εντολές μηχανής (machine code), με μεγάλη ταχύτητα. Αποτελεί ιδανικό για την δημιουργία αποδιτικών και επεκτάσιμων εφαρμογών, καθώς χρησιμοποιεί ένα ασύγχρονο event-driven πρότυπο. Μπορεί να διαχειριστεί μεγάλους αριθμούς αιτημάτων δικτύου στο ίδιο μοναδικό thread μέσω αυτού του μηχανισμού, γίνοντας ιδανική για real-time εφαρμογές [35].

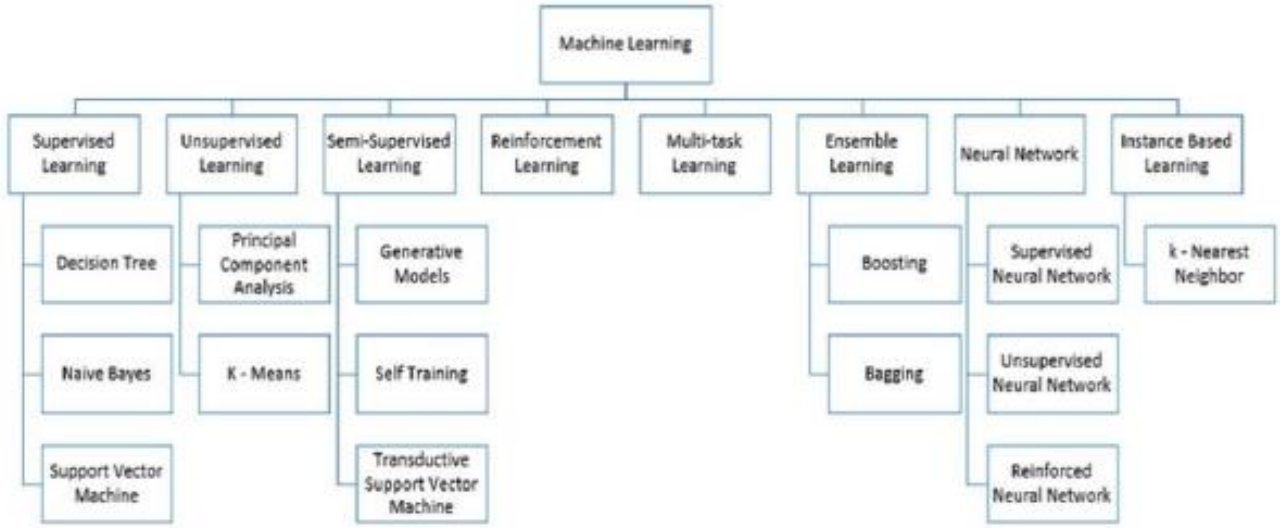
Περαιτέρω, διατίθεται το npm (Node Package Manager), που αποτελεί το μεγαλύτερο οικοσύστημα open source βιβλιοθηκών παγκοσμίως. Με αποτέλεσμα να είναι δυνατή με την γνώση αυτής της τεχνολογίας η γρήγορη ανάπτυξη οποιουδήποτε συστήματος ακόμη και 3D Web παιχνιδιών.

4.3.3 Tensorflow

Το Tensorflow, είναι μία βιβλιοθήκη ανοιχτού λογισμικού και αναπτύχθηκε από την Google ως ένα εργαλείο διεπαφών για την ευκολότερη χρήση και εκπαίδευση μοντέλων μηχανικής μάθησης και βαθιάς μάθησης (Deep Learning). Παρέχει υποστήριξη εκτέλεσης σε επεξεργαστές CPU, κάρτες γραφικών GPU και ειδικών chip μηχανικής μάθησης TPU (Tensor Processing Unit) [37].

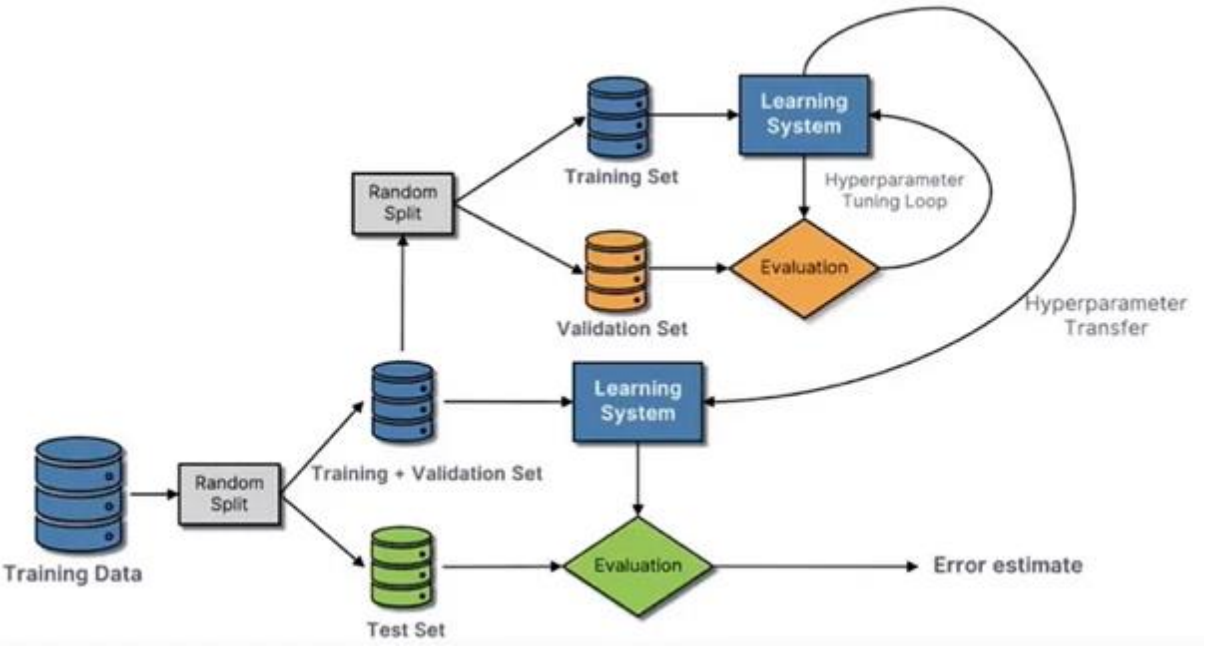
Έχει αναπτυχθεί κυρίως για την γλώσσα προγραμματισμού Python αλλά μπορεί να χρησιμοποιηθεί και μέσω JavaScript, Java και C++. Κυρίως εκτελείται σε ισχυρούς διακομιστές αλλά μέσω των Tensorflow.js και Tensorflow Lite υπάρχει η δυνατότητα εκτέλεσης σε ιστοσελίδας και mobile εφαρμογές αντίστοιχα. Λόγος της ανοιχτής φύσης του και της ενεργής του κοινότητας, υπάρχει μία μεγάλη γκάμα προ-εκπαιδευμένων μοντέλων στην αναγνώριση εικόνων και ήχου και στην επεξεργασία φυσικής γλώσσας (NLP - Natural Language Processing). Προτού αναλυθεί η τεχνολογία αυτή περαιτέρω, θα επεγηθεί ο τομέας της μηχανικής μάθησης [37].

Η μηχανική μάθηση, αποτελεί την μέθοδο διδασκαλίας των μηχανών με σκοπό την αποδοτικότερη διαχείριση των δεδομένων. Κύρια οφέλη παρουσιάζει στις περιπτώσεις όπου οι πληροφορίες και τα μοτίβα δεν είναι εύκολα ορατά από ένα σύνολο δεδομένων. Απώτερος στόχος ενός μοντέλου μηχανικής μάθησης είναι η εκμάθηση από τα διαθέσιμα δεδομένα. Οι τεχνικές που μπορούν να εφαρμοστούν είναι δεκάδες, με την καθεμία να προορίζεται για διαφορετικού τύπου προβλήματος δεδομένων [38].



Σχήμα 4.7: Διαφορετικοί μέθοδοι εκπαίδευσης μοντέλων μηχανικής μάθησης [38].

Η πιο συνηθισμένη τεχνική Machine Learning, συνιστά το Supervised Learning. Με την μέθοδο αυτή, η μηχανή μαθαίνει μία λειτουργία μέσω της αντιστοίχισης μιας εισόδου με μία έξοδο με βάση παραδειγμάτων από ζευγάρια εισόδων και εξόδων. Ο αλγόριθμος εκπαιδεύεται σε ένα σύνολο δεδομένων από ετικέτες (labels). Τα δεδομένα χωρίζονται σε σύνολα δεδομένων train, validation και test αυστηρώς με τυχαίο τρόπο. Σκοπός του μοντέλου, είναι η αντιστοίχιση των δεδομένων στις αντίστοιχες ετικέτες. Το μοντέλο χρησιμοποιεί αρχικά το train dataset για την εκπαίδευση του, και μέσω των μοτίβων που έχει ανακαλύψει, στο τέλος, τα εφαρμόζει στο test dataset για να εκτιμηθεί η ικανότητα του μοντέλου να προβλέπει άγνωστα δεδομένα, και συνεπώς η γενική αξιολόγηση του. Αξίζει να σημειωθεί πως, το μοντέλο κατά την εκπαίδευση έχει γνώση των labels, γεγονός που δεν ισχύει κατά την αξιολόγηση, για την αντικειμενική εκτίμηση του. Συνοψίζοντας, τα validation δεδομένα χρησιμοποιούνται κατά την εκπαίδευση για τον έλεγχο και την αποφυγή υπερπροσάρμοσης (overfitting) και υποπροσάρμοσης (underfitting) του μοντέλου [38].



Σχήμα 4.8: Διάγραμμα ροής εκπαίδευσης μέσω χρήσης των train, set και validation datasets [39].

Ένας αλγόριθμος μηχανικής μάθησης, κάνει overfitting, όταν μαθαίνει υπερβολικά καλά τα δεδομένα εκπαίδευσης και χάνει την απόδοση σε νέα τυχαία δεδομένα. Στην περίπτωση του underfitting, το μοντέλο δεν μαθαίνει με επιτυχία καλά τα δεδομένα εκπαίδευσης, με αποτέλεσμα να μην μπορεί να ανταπεξέλθει ούτε στα τυχαία δεδομένα. Η αποφυγή των προβλημάτων αυτών, προέρχεται μέσω της ρύθμισης των υπερ-παραμέτρων (hyperparameters). Οι επιλογές αυτές, ορίζονται από τον προγραμματιστή και τις πιο βασικές αποτελούν [38]:

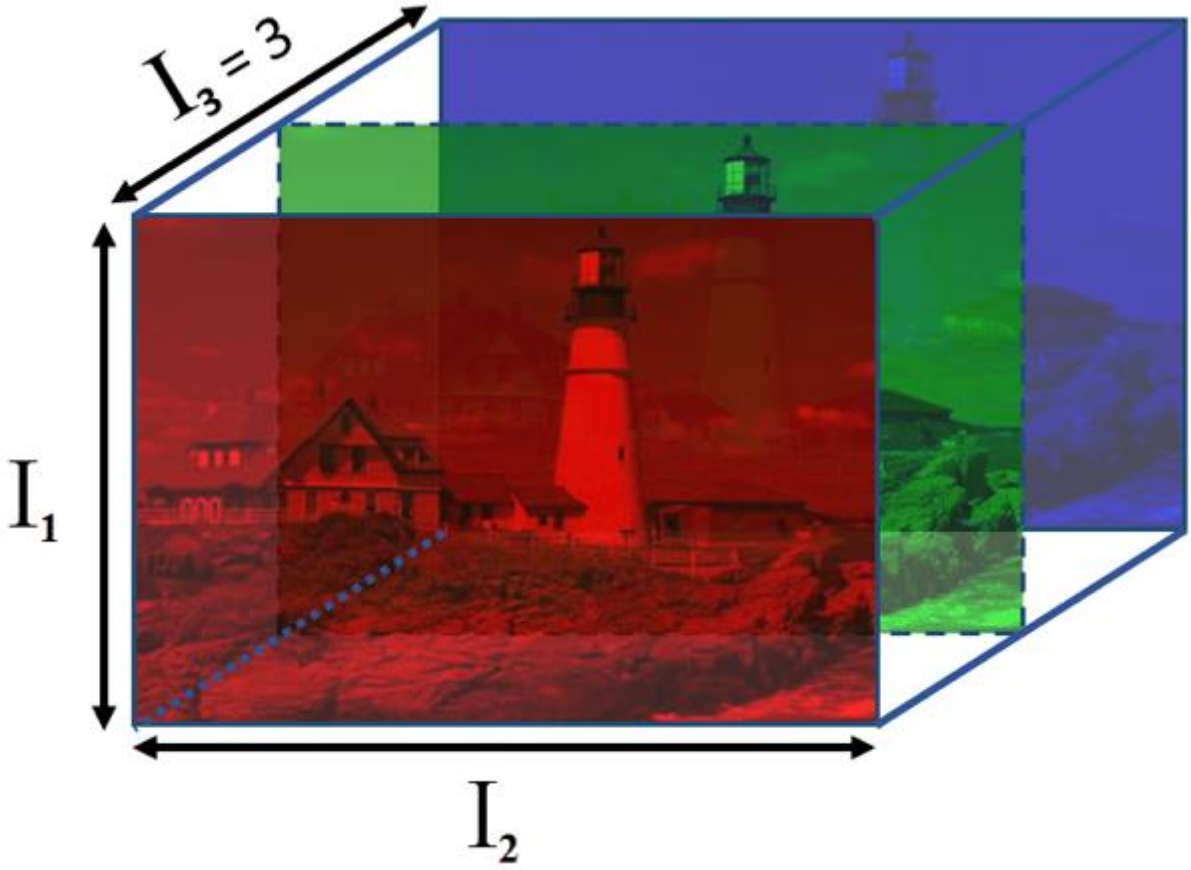
1. Ο αριθμός εποχών (epochs), που καθορίζει τον αριθμό επαναλήψεων της εκπαίδευσης σε όλα τα δεδομένα.
2. Ο ρυθμός μάθησης (learning rate), ορίζει το μέγεθος των βημάτων κατά την ενημέρωση το παραμέτρων του μοντέλου.
3. Το μέγεθος παρτίδας (batch size), προβλέπει τον αριθμό δειγμάτων που επεξεργάζεται προτού ενημερώσει τις παραμέτρους του. Μεγαλύτερο batch size σημαίνει ταχύτερη εκπαίδευση αλλά είναι περιοριστική ρύθμιση στα μηχανήματα με χαμηλούς πόρους ισχύος.
4. Αριθμός κρυφών στρωμάτων και νευρώνων (Hidden layers and Neurons). Με την συγκεκριμένη ρύθμιση να επηρεάζει άμερα την πολυπλοκότητα του μοντέλου.

Η βαθιά μάθηση (Deep Learning), αποτελεί υποκατηγορία της μηχανικής μάθησης και μπορεί να εφαρμοστεί με διάφορους τρόπους εκπαίδευσης, με τον κυριότερο να αποτελεί το Supervised Learning. Βασίζεται σε τεχνητά νευρωνικά δίκτυα, τα οποία υφίστανται από πολλαπλά στρώματα νευρώνων με το κάθε στρώματα να επεξεργάζεται δεδομένα εισόδου. Στόχος είναι η εξαγωγή αφηρημένων χαρακτηριστικών για την εκμάθηση σύνθετων μοτίβων στα δεδομένα. Η εξαγωγή αυτή γίνεται αυτόματα σε αντίθεση με άλλες τεχνικές κάνοντας το ειδικό σε προβλήματα αναγνώρισης κειμένου, εικόνας και ήχου [38].

Στην δομή του Tensorflow, ο κάθε αλγόριθμος μηχανικής μάθησης αναπαριστάνεται από υπολογιστικούς γράφους (computational graphs). Ο υπολογιστικός γράφος, είναι μία μορφή γράφου από κόμβους (nodes), με τον κάθε κόμβο να συνιστά μία μαθηματική πράξη, και από ακμές (edges) οι οποίες αποτελούν την σύνδεση για την ροή πληροφορίας μεταξύ των κόμβων. Η συγκεκριμένη αρχιτεκτονική βοηθάει στην καλύτερη απόδοση των προαναφερόμενων τύπων επεξεργαστών, (CPU, GPU και TPU), στην εκτέλεση πολύπλοκων υπολογισμών [37].

Το Tensorflow, κάνει χρήση των τανυστών (Tensors), και αποτελεί την βασική δομή του εργαλείου αυτού. Τα Tensors στο περιβάλλον αυτό αποτελούνται από πολυδιάστατους πίνακες, με κάθε διάσταση να παίρνει μεταβλητό και ανεξάρτητο από τις υπόλοιπες μέγεθος.

1. 0D Tensor ή αλλιώς αριθμός Scalar. Αποτελείται από έναν μονάδα αριθμό.
2. 1D Tensor, ή αλλιώς διάνυσμα Vector. Αποτελεί μία σειρά τιμών όπως μία λίστα.
3. 2D Tensor, ή αλλιώς πίνακας Matrix. Αποτελεί έναν πίνακα τιμών με γραμμές και στήλες.
4. nD Tensor. Συνιστά πίνακες μεταβλητών διαστάσεων.



Σχήμα 4.9: Τρισδιάστατη απεικόνιση έγχρωμης εικόνας σε Tensor [40].

Οι τανυστές χρησιμοποιούνται για την αναπαράσταση δεδομένων εισόδου. Για παράδειγμα η εικόνα αναπαριστάται από 3 κανάλια RGB, για το χρώμα, και από τον αριθμό των pixels στο ύψος και στο πλάτος, συνεπώς το αντίστοιχο Tensor θα ακολουθήσει την μορφή:

(4.1):

$$(height, width, 3)$$

Επιπλέον σε Tensors, αποθηκεύονται οι παράμετροι του μοντέλου κατά την εκπαίδευση. Οι παράμετροι αυτοί αποτελούν τις τιμές που μαθαίνει ο αλγόριθμος κατά την εκπαίδευση και χρησιμοποιούνται για τις προβλέψεις. Τέλος, την ίδια δομή συνιστούν και οι έξοδοι του μοντέλου. Για παράδειγμα η έξοδος πρόβλεψης εικόνων προϋποθέτει τις πιθανότητες για κάθε κατηγορία και για κάθε εικόνα. Οπότε μία έξοδος Tensor θα έχει την ακόλουθη μορφή για 2 εικόνες με 4 κλάσεις:

(4.2):

$$(Images, Classes) = (2, 4) = ([0.5, 0.1, 0.2, 0.2], [0.6, 0.1, 0.1, 0.2])$$

Συνοψίζοντας, το Tensorflow, αποτελεί ένα ισχυρό εργαλείο για την αποδοτική, εύχρηστη και γρήγορη ανάπτυξη και υλοποίηση τεχνικών έξυπνης αναγνώρισης.

4.3.4 InsightFace

Το InsightFace είναι μια open source βιβλιοθήκη, που προσφέρει προ εκπαιδευμένα μοντέλα αναγνώρισης προσώπων, τελευταίας τεχνολογίας. Είναι βασισμένο στην γλώσσα προγραμματισμού Python, και στα frameworks βαθιάς μηχανικής μάθησης όπως το PyTorch και MXNet. Αποτελεί

εξειδικευμένη λύση στην αναγνώριση και ανάλυση προσώπων, καθιστώντας το ευρέως αναγνωρισμένο και την κορυφαία επιλογή στο συγκεκριμένο ζήτημα [41][42].

Χρησιμοποιεί βαθιά συνελικτικά νευρωνικά δίκτυα (CNNs), όπως τα ResNet, MobileNet και DenseNet. Μέσω της τεχνικής αυτής, πραγματοποιείται εξαγωγή μοτίβων και χαρακτηριστικών προσώπων από εικόνες, με σκοπό την την αναγνώριση ατόμων [41][42].

Το πιο γνωστό μοντέλο του InsightFace αποτελεί το ArcFace, και χρησιμοποιείται για την αναγνώριση προσώπων. Αυτό που το κάνει να ξεχωρίζει, είναι η υλοποίηση της συνάρτησης απώλειας (Loss Function). Η συνάρτηση αυτή αποτελεί σημαντικό χαρακτηριστικό ενός μοντέλου, καθώς μέσω αυτής υπολογίζεται το μέτρο λάθους πρόβλεψης στα δεδομένα εκπαίδευσης, το οποίο μέτρο προσπαθεί να ελαχιστοποιήσει. Συνεχίζοντας στην υλοποίηση του InsightFace, εισάγει καινοτομία στην συνάρτηση κόστους μέσω του Additive Angular Margin Loss. Αυτή η καινοτομία, εκμεταλλεύεται την γωνιακή απόσταση (Angular Margin) με στόχο της ενίσχυση της διαχωριστικότητας των χαρακτηριστικών προσώπου με αποτέλεσμα την καλύτερη πρόβλεψη [41].

Η αναγνώριση προσώπων μπορεί να επιτευχθεί και με διάφορες εναλλακτικές όπως η FaceNet, η DeepFace και άλλες. Όμως η ευκολία ενσωμάτωσης, η αποδοτικότητα και η ευελιξία που προσφέρει το InsightFace το καθιστά μία από τις καλύτερες επιλογές [42].

4.3.5 MongoDB

Το MongoDB, είναι μία open source, γραμμένη στην C++, βάση δεδομένων. Βασισμένη στη μη σχεσιακή (NoSQL) τύπου Εγγράφου (Document) λογική. Τα δεδομένα αποθηκεύονται σε εγγραφές BSON (Binary JavaScript Object Notation) επιτρέποντας μία ευέλικτη και δυναμική δομή

Οι βασικοί τύποι βάσεων δεδομένων, αποτελούν οι SQL και NOSQL. Με την πρώτη, την SQL, να αποτελεί σχεσιακού τύπου ανάμεσα στα δεδομένα. Χωρίζονται σε πίνακες με γραμμές και στήλες με αυστηρό σχήμα και σχέσεις μεταξύ τους. Για την ανάκτηση και διαχείριση τους χρησιμοποιείται η γλώσσα SQL (Structured Query Language) [43].

Από την άλλη πλευρά, η NOSQL, δεν χρησιμοποιεί πίνακες αλλά collections και documents, αντί για στήλες, για την αποθήκευση δεδομένων. Τα δεδομένα αποθηκεύονται σε ζευγάρια κλειδιού-τιμής (key-value). Το κλειδί αποτελεί το χαρακτηριστικό της τιμής για αργότερη εύρεση ενώ η τιμή μπορεί να είναι οποιαδήποτε μορφής, ακόμα και ζευγάρι key-value. Η πιο συνηθισμένη μορφή ενός μη σχεσιακού αρχείου είναι το JSON και BSON, η δυαδική κωδικοποίηση του JSON, αλλά ακόμα και την μορφή XML [43].

Το JSON (Javascript Object Notation), αποτελεί μια μορφή ανταλλαγης δεδομένων και χρησιμοποιείτε ευρέως στη αιτήματα προς και από απομακρυσμένους servers. Βασίζεται στην σύνταξη της Javascript, όμως υπάρχει υποστήριξει του μοντέλου αυτού στις περισσότερες γλώσσες προγραμματισμού. Η σύνταξή του περιέχει, αντικείμενα μορφής key-value, με την τιμή να μπορεί να πάρει οποιαδήποτε μορφή, ακόμα και ξανά τιμή κλειδιού-τιμής αλλά και λίστας. Το BSON, είναι ακριβώς το ίδιο, με ένα επιπλέον στρώμα κωδικοποίησης για την αποδοτικότερη αποθήκευση και διάσχιση των δεδομένων.

Σχήμα 4.10: Παράδειγμα σχήματος δεδομένων ενός Document στο MongoDB [44].

Η τεχνολογία MongoDB, χρησιμοποιεί όπως προαναφέρθηκε την βασισμένη key-value Document τύπου λογική. Διαθέτει μεγάλο βαθμό ευελιξίας, καθώς τα αντικείμενα και τα αρχεία δεν χρειάζεται να ακολουθούν κανόνες που επιβάλλονται στις αντίστοιχες σχεσιακές λύσεις. Συνεπώς, 2 αρχεία Document στην ίδια συλλογή μπορούν να έχουν διαφορετικού τύπου δεδομένα. Γεγονός που μπορεί να προκαλέσει προβλήματα στη λάθος πρακτική και χρήση του εργαλείου αυτού. Η αρχιτεκτονική αυτή αποκαλείτε Schema-less Design. Περαιτέρω, κάνει την δυνατότητα κλιμάκωσης ενός Database αρκετά ευκολότερο λόγω της δυναμικής του φύσης. Επιπλέον, για τις 4 βασικές ενέργειες μιας βάσης δεδομένων, τις ενέργειες CRUD (Create Read Update Delete), προσφέρει μικρότερους χρόνους εκτέλεσης απο σχεσιακές βάσεις όπως την MySQL. Τέλος η ενέργεια του Write, γίνεται με ασύγχρονο τρόπο, γεγονός που μειώνει τον χρόνο εγγραφής αλλά σε περίπτωση σφάλματος δεν βεβαιώνεται ολοκληρωτικά η αποτυχία της εγγραφής.

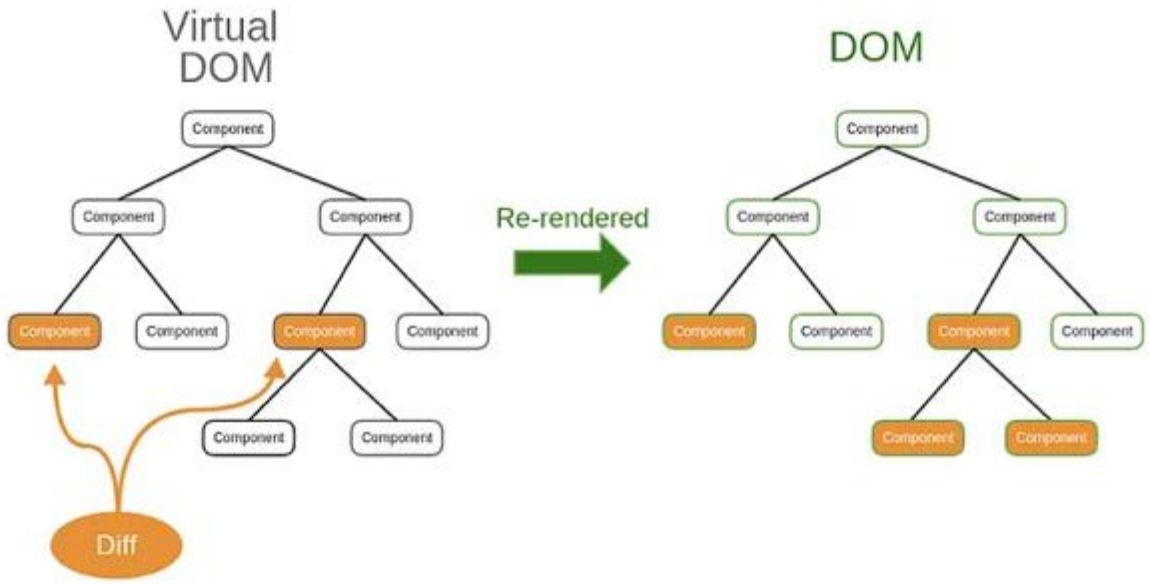
Η επιλογή χρήσης MongoDB αντί για σχεσιακές SQL βάσεις, αποτελείται από διάφορα κριτήρια. Σε περιβάλλοντα όπου είναι αναγκαία η ταχύτητα εκτέλεσης λόγω μεγάλου όγκου δεδομένων, υπάρχει συχνή αλλαγή της μορφής του σχήματος της βάσης και χρειάζεται η δυναμικότητα στον τύπο δεδομένων, τότε η λύση αυτή συνιστά μία κατάλληλη επιλογή. Άλλες παρόμοιες τεχνολογίες με το MongoDB, αποτελούν τα RavenDB, CouchDB και Cassandra, όμως αποτελούν λιγότερο δημοφιλές προσεγγίσεις [43].

4.4 Τεχνολογίες Frontend

4.4.1 React

Η React είναι μία βιβλιοθήκη JavaScript με σκοπό την διευκόλυνσης της δημιουργίας αντιδραστικών γραφικών διεπαφών σε περιβάλλον ιστού. Δημιουργήθηκε από την ομάδα της Facebook, για την ίδια την πλατφόρμα, ως λύση στις μεγάλες απαιτήσεις μιας αρκετά δυναμικής διεπαφής χρήστη (User Interface). Από το 2013 διατίθεται ως open source λογισμικό για την κοινότητα, καθώς και η κύρια frontend τεχνολογία της Meta [45].

Χρησιμοποιεί δηλωτικό τρόπο, μέσω JSX συντακτικού, για την δημιουργία των UI, όπου ο προγραμματιστής μπορεί να χωρίσει την ιστοσελίδα σε τμήματα, με το κάθε τμήματα να απαρτίζει μία ξεχωριστή ροή της κατάστασης (State) των δεδομένων, τα τμήματα αυτά ονομάζονται Components. Η React διαχειρίζεται την ενημέρωση της γραφικής διεπαφής κατά την αλλαγή του State προκαλώντας ενημέρωση των απαιτούμενων component όπου χρειάζεται (re-render). Ο τρόπος με τον οποίο πετυχαίνει αυτή την τεχνική είναι η δημιουργία και η ενημέρωση του εικονικού DOM (Virtual DOM) [45].



Σχήμα 4.11: Σχήμα δέντρων των DOM κατά την αλλαγή κατάστασης [46].

Το DOM (Document Object Model), αποτελεί τον συνδετικό κρίκο μεταξύ μιας ιστοσελίδας (HTML) και του κώδικα, συνήθως της JavaScript. Στο μοντέλο αυτό, η κάθε σελίδα αποτελεί ένα Document, με ένα λογικό δέντρο. Το κάθε κλαδί του δέντρο τελειώνει με έναν κόμβο (Node), δηλαδή ένα HTML Element ή κάποιο κείμενο. Το οποίο περιέχει αντικείμενα και μέσω του DOM είναι δυνατή η πρόσβαση και η τροποποίηση τους. Η HTML, είναι ο τρόπος με τον οποίο δομείται κάθε ιστοσελίδα. Μέσω των HTML elements, ο browser εμφανίζει περιεχόμενο όπως κείμενο, εικόνες, βίντεο, συνδέσμους και άλλα [45].

Το Virtual DOM, είναι μία προγραμματιστική έννοια όπου, μία ιδανικά εικονική μορφή της γραφικής διεπαφής υπάρχει στην μνήμη και συγχρονίζεται με το πραγματικό DOM. Κατά την αλλαγή των δεδομένων (State), η React δημιουργεί το νέο εικονικό DOM και το αντιστοιχεί με το ήδη αποθηκευμένο στη μνήμη, και έτσι γνωρίζει τα σημεία που έχουν αλλάξει, με στόχο την αποδοτικότερη αντικατάσταση της σελίδας μόνο στα απαραίτητα σημίας [45].

Σχήμα 4.12: Παράδειγμα χρήσης JSX σε React Component [47].

Καθώς η χρήση React, προϋποθέτει την χρήση JavaScript χωρίς HTML, για την ενσωμάτωση λογικής και UI στο ίδιο αρχείο, δημιουργήθηκε το συντακτικό της JSX (JavaScript XML). Μοιάζει αρκετά με το συντακτικό HTML καθώς ακολουθεί το tag-based πρότυπο. Αποτελεί τον τρόπο δημιουργίας των React Elements, δηλαδή την δημιουργία των Component, που συχνά αποτελούν κάποιο κείμενο ή HTML Element σε μορφή αντικειμένου [45].

Η React, εντάσσει καλές πρακτικές κώδικα. Μερικές από αυτές αποτελούν, η επαναχρησιμοποίηση του κώδικα. Λόγω της αρκετά δυναμικής του φύση, κάθε Component μπορεί να χρησιμοποιηθεί σε πολλά σημεία της ιστοσελίδας, αυξάνοντας τον ρυθμό ανάπτυξης, βελτιώνοντας την συνοχή, την συντήρηση και την κλιμάκωση του κώδικα. Εξαλείφεται η ανάγκη διαχωρισμού της διεπαφής με την λογική και ο κώδικας γίνεται πιο ευανάγνωστος και προβλέψιμος μέσω της δηλωτικής πρακτικής. Επιπλέον υποστηρίζεται από τις μεγαλύτερες κοινότητες, με μία πληθώρα συμβατών βιβλιοθηκών όπως την Redux και Zustand για την διαχείριση του State. Επιπλέον, παρέχει δυνατότητα δημιουργίας ιστοσελίδων από τον server, μέσω του Nodejs. Με αποτέλεσμα να μειώνεται ο χρόνος αναμονής σε συσκευές με χαμηλή υπολογιστική ισχύ, αλλά και να κρύβεται η υλοποίηση με σκοπό την βελτίωση της ασφάλειας. Τέλος, μέσω της React Native, είναι δυνατή η ανάπτυξη υβριδικών mobile εφαρμογών για Android και iOS, παίρνοντας τον τίτλο learn once, write anywhere.

Ανταγωνιστές της React αποτελούν, η Angular, η Vue και η Svelte. Αποτελούν καλές λύσεις και μπορούν να χρησιμοποιηθούν, αναλόγως το use case, και είναι καλύτερες από την React σε συγκεκριμένους τομείς. Όμως αυτό που την κάνει να ξεχωρίζει, από τις υπόλοιπες, είναι ο συνδυασμός της ταχύτητας, της ευελιξίας και του οικοσυστήματος που προσφέρει συνολικά.

4.4.2 Vite

Το Vite, είναι ένα build tool, με στόχο την βελτίωση της ανάπτυξης μοντέρνων εφαρμογών ιστού. Αποτελείται από δύο κύρια μέρη, αρχικά από έναν server, για την διαδικασία ανάπτυξης όπου ο προγραμματιστής βλέπει ζωντανά τις αλλαγές στον κώδικα μέσω της τεχνικής HMR (Hot Module Replacement). Επιπλέον, αποτελείται από μια εντολή για build, δηλαδή για την δημιουργία των τελικών

αρχείων που θα χρησιμοποιηθούν στην παραγωγή της ιστοσελίδας. Μπορεί να χρησιμοποιηθεί από τεχνολογίες ανάπτυξης frontend εφαρμογών, όπως React, Vue και Svelte [48].

Μέσω του HMR, κατά την ανάπτυξη προσφέρει γρήγορη ενημέρωση των στοιχείων που αλλάζουν στον κώδικα, χωρίς την ανάγκα ανανέωσης της σελίδας, βελτιώνοντας την ταχύτητα της διαδικασίας ανάπτυξης ιστοσελίδων [48].

Για το χτίσιμο της τελικής ιστοσελίδας, χρησιμοποιεί τον Rollup bundler, όπου μαζεύει όλα τα αρχεία του κώδικα σε ένα και τα ετοιμάσει για παραγωγή. Με αποτέλεσμα το τελικό προϊόν να αποτελείται από 3 αρχεία, τα HTML, CSS και JavaScript αρχεία [48].

Συνοψίζοντας, αποτελεί ένα από τα θεμελιώδη εργαλεία ανάπτυξης React εφαρμογών λόγω της άνεσης που προσφέρει κατά την ανάπτυξη και την παραγωγή.

4.4.3 Tailwind CSS

Η CSS (Cascading Style Sheet), η τεχνολογία υπεύθυνη για τον τρόπο εμφάνισης (styling) της HTML στον Browser, όπως τρόπος διάταξης των στοιχείων, μέγεθος και χρώμα γραμματοσειράς και άλλα. Καθώς η CSS αποτελεί ένα ισχυρό εργαλείο για την τροποποίηση της εμφάνισης των ιστοσελίδων, υποφέρει από μερικά αρνητικά στοιχεία κυρίως στις large scale εφαρμογές [49].

Με το κυριότερο να αποτελεί η δημιουργία ξεχωριστών αρχείων css αποτέλεσμα της δημιουργίας πολλών αρχείων και ο χωρισμός του styling με την διάταξη των στοιχείων. Επιπλέον για την επιβολή styling σε στοιχεία HTML απαιτείται ο ορισμός ονόματος κλάσης, αυτή η διαδικασία γίνεται όλο και δυσκολότερη στον προγραμματιστή καθώς με την κλιμάκωση της ιστοσελίδας, μειώνονται οι διαθέσιμη συνδυασμοί χαρακτήρων. Επιπλέον, με την κλιμάκωση της ιστοσελίδας, εμφανίζονται css κλάσεις που δεν χρησιμοποιούνται, με αποτέλεσμα το τελικό css αρχείο να είναι μεγαλύτερο και να αυξάνεται ο χρόνος φόρτωσης της σελίδας από την μεριά του τελικού χρήστη [49].

Αυτά τα προβλήματα, έρχεται να λύσει η Tailwind CSS. Αποτελεί ένα σύγχρονο CSS πλαίσιο, όπου παρέχεται η δυνατότητα ενσωμάτωσης CSS styling απευθείας στο HTML Element μέσω του γνωρίσματος class. Ειδικά σε συνδυασμό με βιβλιοθήκες όπως η React, είναι δυνατή η ανάπτυξη ολόκληρων frontend εφαρμογών μόνο με JavaScript κώδικα, ενσωματώνοντας έτσι την λογική, την διάταξη και την εμφάνιση μαζί [49][50].

```
export const AnimatedArrow = ({className, iconClassName}: Props) => {
  return (
    <div
      className={twMerge(
        'absolute right-10 top-[50%] -translate-y-[50%] opacity-0 group-hover:opacity-100 transition-all group-hover:translate-x-4 z-[150]',
        className,
      )}>
      <Icon
        icon="arrow-right-long"
        className={twMerge(
          'size-5 group-hover:size-6 text-white',
          iconClassName,
        )}
      />
    </div>
  );
};
```

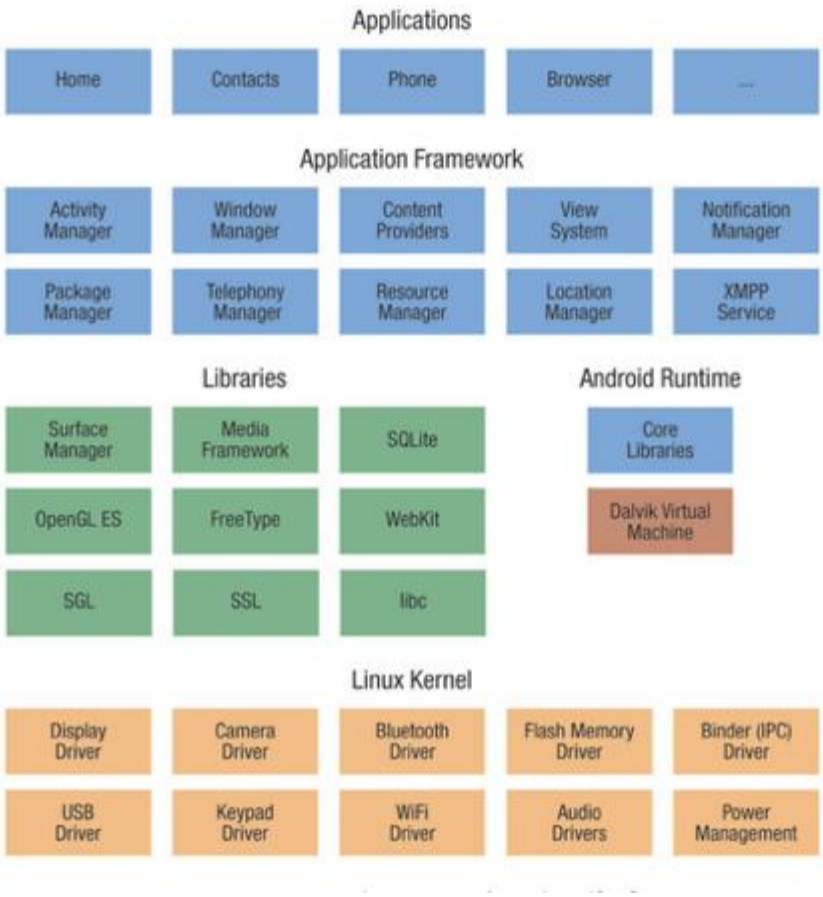
Σχήμα 4.13: Παράδειγμα χρήσης Tailwind CSS σε React.

Η Tailwind CSS, σκανάρει όλα τα αρχεία και δημιουργεί η ίδια την απαιτούμενη CSS από τα ονόματα κλάσεων που φαίνονται στο σχήμα 4.13. Με αποτέλεσμα το τελικό προϊόν να είναι ένα CSS αρχείο που περιέχει μόνο τα απαραίτητα από μία φορά, γίνοντας έτσι αποδοτικότερη από την χρήση κλασικής CSS.

4.4.4 React Native

Το Hybrid Development (υβριδική ανάπτυξη), προϋποθέτει την δημιουργία εφαρμογών για διάφορα λειτουργικά συστήματα μέσω ενός source code. Αυτή η τεχνική γίνεται ολοένα και πιο δημοφιλής, ειδικά σε περιπτώσεις όπου η ταχύτητα ανάπτυξης ξεπερνά τις ανάγκες για την ταχύτητα των εφαρμογών. Καθώς ο τρόπος λειτουργίας των υβριδικών εφαρμογών, συνδυάζει κομμάτια του Web με κομμάτια του κάθε λειτουργικού. Δηλαδή, ο κώδικας, δημιουργείται με τεχνικές ανάπτυξης σύγχρονων ιστοσελίδων, όμως ενσωματώνεται και εμφανίζεται σε mobile εφαρμογές, μέσω ενός WebView. WebView αποτελεί Component και παρέχεται από τα λειτουργικά συστήματα, όπως iOS και Android, με δυνατότητες εμφάνισης περιεχομένου ιστοσελίδας (δηλαδή HTML, CSS και JavaScript) μέσα στην mobile εφαρμογή. Αυτή τον τρόπο ανάπτυξης ακολουθούν τεχνολογίες όπως η React Native μέσω JavaScript και η Flutter της Google μέσω Dart [51].

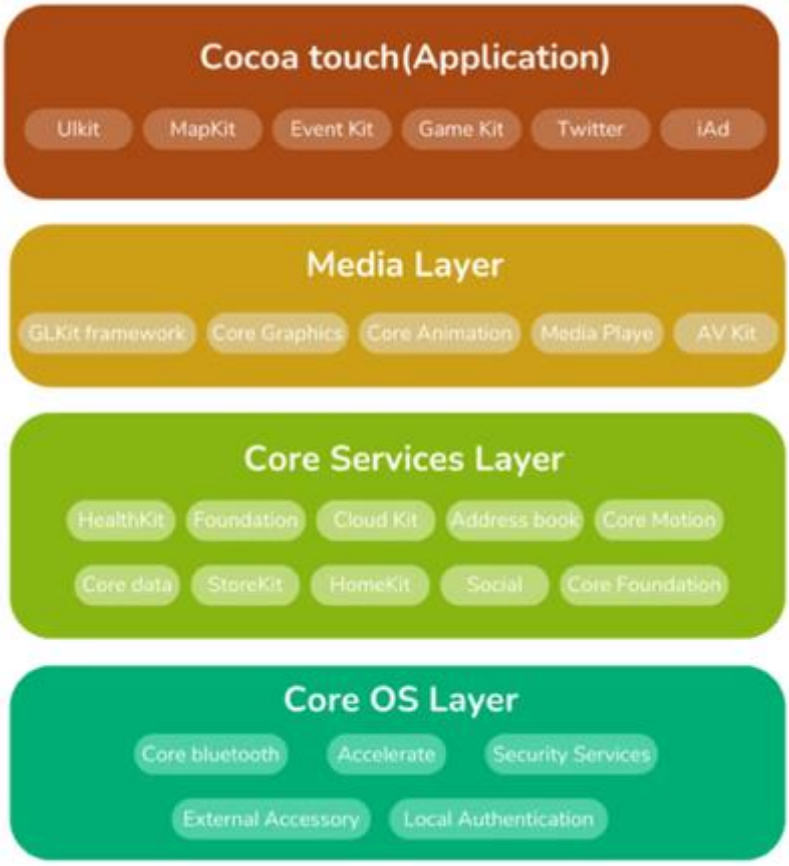
Η React Native, χρησιμοποιείται για την ανάπτυξη υβριδικών mobile εφαρμογών, iOS και Android, με την χρήση ενός source code, μέσω της JavaScript και με την χρήση της React βιβλιοθήκης. Σε αντίθεση με την React στο Web, που εμφανίζει HTML Elements, η React Native εκμεταλλεύεται τα Native Components της κάθε πλατφόρμας με αποτέλεσμα, στο οπτικό, να μην υπάρχει διαφορά από μία Native εφαρμογή όπως Kotlin και Swift. Δημοσιεύθηκε το 2015, από την ομάδα του Facebook, αρχικά με υποστήριξη για iOS και στην συνέχεια επεκτάθηκε και για Android κινητά. Προτού όμως αναλυθεί η React Native, περαιτέρω, θα συζητηθούν οι αρχιτεκτονικές των λειτουργικών iOS και Android [51].



Σχήμα 4.14: Στρώματα αρχιτεκτονικής Android OS με τα χαρακτηριστικά τους [51].

Το Android, είναι είναι ένα OS (Operating System) βασισμένο στο Linux Kernel, με τροποποιήσεις για να τρέχει σε κινητές συσκευές, όπως Smartphones, Tablets και smart TVs. Αναπτύχθηκε από την Google, το 2008, και αποτελεί το πιο δημοφιλές λειτουργικό σύστημα στον τομέα του mobile [51][52].

Η αρχιτεκτονική του Android, όπως φαίνεται στο σχήμα 4.13, αρχικά περιέχει τους Drivers, τρόποι με τους οποίους επικοινωνεί με το υλικό (hardware) και είναι γραμμένοι κυρίως στην C. Ακριβώς από πάνω, υπάρχουν οι Native βιβλιοθήκες, οι οποίες συνδέουν τις mobile εφαρμογές με τα χαρακτηριστικά που προσφέρει το λειτουργικό. Οι βιβλιοθήκες αυτές είναι γραμμένες στην C και C++, επιτρέποντας μεγάλη αποδοτικότητα, και είναι διαθέσιμες μέσω διεπαφών της Java. Στην συνέχεια υπάρχει το Application Framework, το οποίο παρέχεται σε μορφή Java κλάσεων, και χρησιμοποιούνται για τον προγραμματισμό και την διαχείριση του τρόπου λειτουργίας των εφαρμογών. Τέλος, στο τελευταίο επίπεδο, υπάρχουν οι εφαρμογές. Υπάρχουν δύο είδη εφαρμογών, οι προ εγκατεστημένες ή αλλιώς εφαρμογές συστήματος, οι οποίες παρέχουν τις κύριες δυνατότητες του Android στον χρήστη και οι εφαρμογές της κοινότητας για την παροχή έξτρα υπηρεσιών, όπως chatting [51][52].

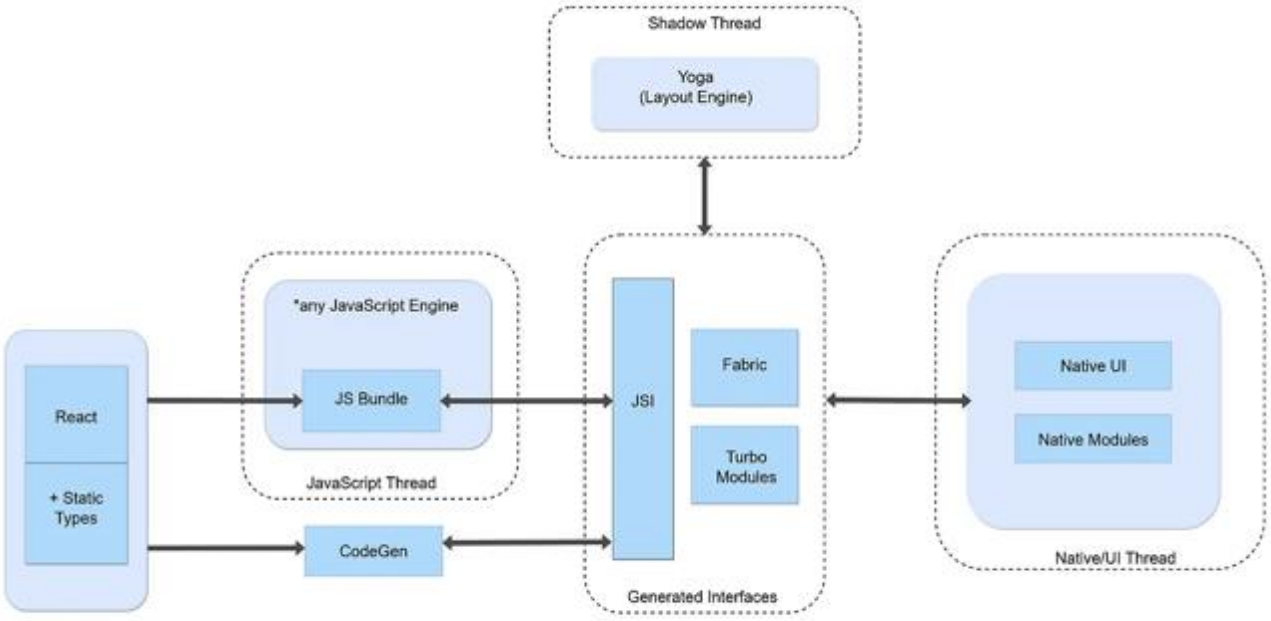


Σχήμα 4.15: Τα στρώματα αρχιτεκτονικής iOS με τις λειτουργίες τους [53].

Το iOS, είναι το λειτουργικό σύστημα που τρέχει στα γνωστά iPhone κινητά. Αναπτύχθηκε από την Apple το 2007, συνεχίζοντας την φιλοσοφία της στην παροχή κλειστού λογισμικού, μόνο σε Apple σχετικές συσκευές. Επιπροσθέτως, οι Apple εφαρμογές μπορούν να αναπτυχθούν μόνο από laptop που τρέχουν το αντίστοιχο λειτουργικό της, το macOS στο οποίο είναι και βασιμένο το iOS. Το λειτουργικό σύστημα της Apple βασίζεται στον Mach Kernel και στο FreeBSD, που παρομοιάζει το Unix [52].

Η αρχιτεκτονική του iOS, όπως παρουσιάζεται στο σχήμα 4.13, αποτελείται από 4 κύρια στρώματα. Το πρώτο layer, το Core OS, χρησιμοποιείται για την παροχή διεπαφών που επικοινωνούν απευθείας με το hardware του κινητού, όπως Bluetooth, Networking, Face και Touch ID. Ακολουθεί το στρώμα

των υπηρεσιών, Core Services, περιέχει την θεμελιώδη βάση που χρησιμοποιούν έμμεσα όλες οι εφαρμογές. Για παράδειγμα, διαχειρίζεται την αποθήκευση των δεδομένων τοπικά και τον συγχρονισμό μέσω iCloud. Στην συνέχεια, υπάρχει το στρώμα των πολυμέσων (Media Layer), υπεύθυνο για την παροχή τεχνολογιών που αφορούν τα γραφικά, το βίντεο και τον ήχο, προσφέροντας εξειδικευμένα εργαλεία στις εφαρμογές για την παροχή πλούσιων Media. Τέλος στην κορυφή της στοίβας στρωμάτων, υπάρχει το Cocoa Touch, μέσω του οποίου παρέχονται οι κύριες διεπαφές για την ανάπτυξη iOS εφαρμογών. Μερικά από τα χαρακτηριστικά που φέρει, συνιστούν την διαχείριση της αφής, τα push Notifications και τον κύκλο ζωής των εφαρμογών [52].



Σχήμα 4.16: Αρχιτεκτονική React Native [54].

Τώρα που υπάρχει μια γενική εικόνα των iOS και Android λειτουργικών, θα συνεχιστεί η ανάλυση της React Native. Στο σχήμα 4.16, παρουσιάζεται η νέα αρχιτεκτονική της React Native, που ισχύει από το 2021. Ο source code της React περνάει από πολλά στάδια για να φτάσει στην τελική του μορφή για mobile [55]:

1) Αρχικά ο κώδικας περνάει την μορφή του bundling μέσω μιας JavaScript μηχανής όπως το Hermes ή το JavaScriptCore. Στόχος, η δημιουργία ενός αποδοτικού εκτελέσιμου αρχείου, όσον αφορά την ταχύτητα, την χρήση μνήμης και το μέγεθος της εφαρμογής.
2) Στην συνέχεια, μέσω του της διεπαφής JSI (JavaScript Interface) δημιουργούνται οι αναφορές των αντικειμένων μεταξύ κώδικα JavaScript και C++, με αποτέλεσμα την άμεση σύνδεση του Native κώδικα των mobile λειτουργικών.
3) Επιπλέον μέσω της νέας διεπαφής Fabric, πραγματοποιείται το render, δηλαδή η εμφάνιση, των στοιχειών στην συσκευή. Υλοποιεί δυναμικά τεχνικές της React στην C++, αυξάνοντας την ταχύτητα και την υποστήριξη των Native στοιχείων.

4) Τα TurboModules, ή αλλιώς Turbo Native Modules, αποτελούν υλοποίηση των native API στην React Native. Για παράδειγμα, η ανάγκη χρήσης του Bluetooth της συσκευής μέσω κώδικα React. Τα πακέτα αυτα, δημιουργούνται δυναμικά όπου χρειάζονται για την εξασφάλιση της απόδοσης στο τελικό προορισμό.

5) Η χρήση της μηχανής διάταξης (Layout Engine) Yoga, εξυπηρετεί την διαρρύθμιση των γραφικών στοιχείων πάνω στην οθόνη της συσκευής και χρησιμοποιείτε από την διεπαφή Fabric.

6) Τέλος, μέσω του δυναμικά παραγόμενου C++ κώδικα, δημιουργείται ο αντίστοιχος native κώδικας για Android και iOS σε Kotlin και Swift. Συνεπώς, η χρήση της C++ αποτελεί γέφυρα μεταξύ των τεχνολογιών αυτών λόγο της ταχύτητας που προσφέρει, τον πλήρη έλεγχο της μνήμης και της παροχής βελτιστοποιημένου κώδικα μηχανής.

Συνοψίζοντας, η React Native συνεχώς εξελίσσεται με την νέα αρχιτεκτονική να φέρνει την απόδοση ακόμα πιο κοντά στις Native επιλογές ανάπτυξης, όπως Kotlin και Swift. Όμως, στο μέτρο της απόδοσης υφίσταται σύγκριση, καθώς την καθιστά πιο αργή λόγω των προαναφερόμενων διαδικασιών. Προφανώς, η επιλογή χρήσης κρίνεται ανάμεσα στην σοβαρότητα της απόδοσης εναντίων της ταχύτητας της παραγωγής, για κάθε εφαρμογή.

4.4.5 Zustand

Η React, αποτελεί μία ισχυρή βιβλιοθήκη με μία μεγάλη ποικιλία λύσεων από την κοινότητα. Έχουν δημιουργηθεί λύσεις για την επίλυση αρκετών αδυναμιών της. Μία από αυτές τις ανεπάρκειες, καθιστά η διαχείριση και η ροή του state μεταξύ Components στις μεγάλες εφαρμογές. Την αρχή στην λύση των αδυναμιών την έκανε η βιβλιοθήκη Redux, η οποία λύνει επαρκώς το συγκεκριμένο πρόβλημα, όμως εντάσσει πολυπλοκότητα και boilerplate code (επαναλαμβανόμενος κώδικας). Συνεπώς να υπάρχει η ανάγκη για την ανάπτυξη μιας απλούστερες και παράλληλα ισάξιας βιβλιοθήκης, την Zustand [56].

Η Zustand, αποτελεί βιβλιοθήκη ανοικτού κώδικα, για την διαχείριση του state σε εφαρμογές React. Είναι από τις λίγες βιβλιοθήκες που μπορεί να ενημερώσει και να διαβάσει React state εκτός του κύκλου ζωής των React Components χωρίς την δημιουργία σφαλμάτων ή bugs. Για να το πετύχει, εκμεταλλεύεται το UseExternalStore API της React, για να συγχρονίσει τα δεδομένα (state) με τον κύκλο ζωής των React Components. Αυτή η τεχνική, βοηθάει στην ομαλότερη ανάπτυξη αλλά και στην βελτίωση της αποδοτικότητας στην ιστοσελίδα. Επιπλέον, μέσω της χρήσης των Selectors, είναι δυνατή πιο συγκεκριμένης επιλογής στην αλλαγή του state, ώστε να επιτευχθεί μείωση των re-renders, που αποτελεί το σημαντικότερο πρόβλημα εφαρμογών React [56].

Τέλος, έχει εύκολη και άμεση υποστήριξη ασύγχρονων ενεργειών και παροχή middlewares, όπως η αυτόματη αποθήκευση του state στον browser, μέσω LocalStorage, για την επιμονή των δεδομένων μετά από επαναφόρτωσης της σελίδας [56].

4.5 Διακομιστές

Ο διακομιστής (Server) είναι ένα ειδικό λογισμικό, τοποθετημένο σε ένα ή περισσότερα μηχανήματα, με σκοπό την παροχή υπηρεσιών σε άλλους διακομιστές ή χρήστες. Δέχεται ένα αίτημα δικτύου, το επεξεργάζεται και αποκρίνεται κατάλληλα. Οι Servers, αποτελούν θεμελιώδη στοιχείο του διαδικτύου, καθώς όλες οι μεγάλες πλατφόρμες υλοποιούν τις δικες τους βελτιστοποιημένες εκδοχές για την παροχή υπηρεσιών. Στις επόμενες ενότητες θα αναλυθούν δημοφιλές υλοποιήσεις διακομιστών.

4.5.1 Nginx

Ο Nginx, αποτελεί ένας από τους πιο αποδοτικούς διακομιστές. Ειδικεύεται στο κομμάτι του Web και παρέχει λειτουργίες όπως Reverse Proxying και Load Balancing. Δημιουργήθηκε το 2004, αποτελεί την πιο δημοφιλή open source επιλογή της κοινότητας [57].

Προσφέρει μία μεγάλη επιλογή ρυθμίσεων μέσω τις οποίας ο προγραμματιστής μπορεί να επιλέξει την συμπεριφορά του server για κάθε αίτημα δικτύου (Network Request). Επιπλέον, μπορεί να γίνει επέκταση του μέσω των modules, για την προσθήκη περισσότερων χαρακτηριστικών όπως την διαχείριση live streaming (rtmp module), την ενσωμάτωσης SSL/TLS ασφαλείας (https module) και άλλα. Ο διακομιστής αυτός μπορεί να εγκατασταθεί μέσω της επίσημης ιστοσελίδας, όμως χρειάζεται να χτιστεί με απαραίτητες ρυθμίσεις και προεπιλεγμένα modules. Γεγονός που αυξάνει την απόδοση, καθώς περιέχει μόνο ότι χρειάζεται, αλλά δεν είναι πιθανή η τροποποίηση του, όπου στην περίπτωση αυτή είναι αναγκαία η επανάληψη της διαδικασίας εγκατάστασης από την αρχη [57].

```
http {

    server {
        listen 80;
        servername mydomain.com;

        location / {
            root /var/www/html;
            index index.html;
            try_files $uri $uri/ /index.html;
        }

        location /api/ {
            proxy_pass http://mysecretbackend:9001/;

            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
    }
```

Σχήμα 4.17: Παράδειγμα υλοποίησης Reverse Proxying σε Nginx HTTP Web Server.

Στο σχήμα 4.17, παρουσιάζεται απόσπασμα από το αρχείο ρυθμίσεων του Nginx, πιο συγκεκριμένα μέσω των γνωρισμάτων proxy, ορίζεται η συμπεριφορά του διακομιστή για Reverse Proxy. Αυτή την τεχνική επιτρέπει τον διακομιστή στο στην διεύθυνση, http://mydomain:80 να κρύψει την ύπαρξη του backend API που βρίσκεται στο http://mysecretbackend:9001, μέσω του http://mydomain/api/, αυξάνοντας έτσι την ασφάλεια και την ιδιωτικότητα, καθώς ο τελικός χρήστης δεν μπορεί να μάθει την πραγματική διεύθυνση του backend. Επιπλέον προσθέτει συνοχή κατά την ανάπτυξη, αφού σε περιβάλλον με πολλούς server δεν χρειάζεται η γνώση της διεύθυνσης παρά μόνο του έξτρα path όπως το /api σε αυτή την περίπτωση. Αξίζει να σημειωθεί η ευκολία στην χρήση SSL/TLS πιστοποιήσεων,

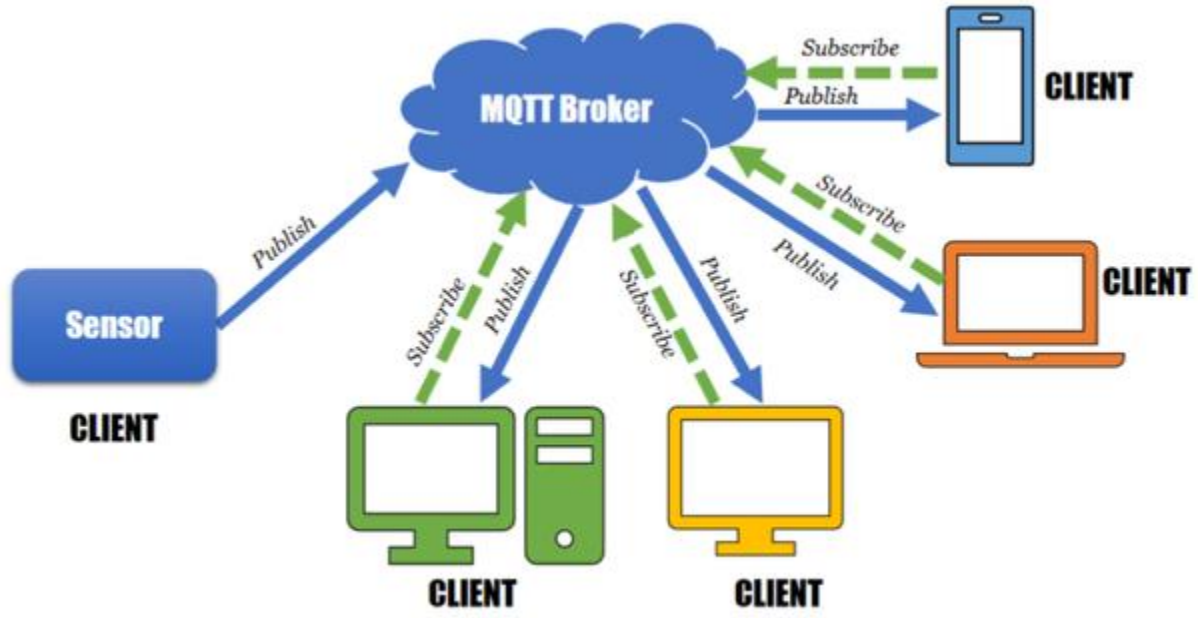
που είναι αναγκαία στην απόκτηση ασφαλές σύνδεσης HTTPs. Χωρίς την χρήση της τεχνικής αυτής, πρέπει όλοι οι server της πλατφόρμας να αποκτήσουν και να υλοποιήσουν την πιστοποίηση αυτή, σε αυτή την περίπτωση όμως, είναι απαραίτητο μόνο στον κεντρικό server που λειτουργεί ως Reverse Proxy.

Η τεχνική Load Balancing, αποτελεί χαρακτηριστικό όλων των μεγάλων υπηρεσιών και χρησιμοποιείται για την εξισορρόπηση του φορτίου σε διάφορους server/μηχανήματα. Ο διαχωρισμός των συνδέσεων γίνεται με διάφορους μεθόδους, όπως την ρύθμιση least_conn που στέλνει τις νέες συνδέσεις στον server με τον λιγότερο αριθμό συνδέσεων [57].

Συνοψίζοντας, ο διακομιστής Nginx, αποτελεί μία αποτελεσματική επιλογή για υλοποίηση συστημάτων που αποτελούνται με παραπάνω από 1 υπηρεσία, καθώς είναι εξαιρετικά ελαφρύς και αποδοτικός στην διαχείριση, ακόμα και χιλιάδων ταυτόχρονων συνδέσεων.

4.5.2 Mosquitto

Ο διακομιστής (broker) Mosquitto, γραμμένος στην C και C++, παρέχει υλοποίηση του MQTT πρωτοκόλλου σε κατάσταση πελάτη-χρήστη ή server. Το MQTT πρωτόκολλο, βασίζεται στο μοντέλο TCP/IP του δικτύου και χρησιμοποιείται κυρίως στο IOT από μικροελεγκτές λόγω της ελαφρότητας του. Στην ενότητα των πρωτοκόλλων TCP/IP θα παρεξηγηθεί περαιτέρω.



Σχήμα 4.18: Παράδειγμα χρήσης και λειτουργίας Mosquitto ως MQTT Broker [58].

Όπως προαναφέρθηκε, γίνεται χρήση του Mosquitto, ως server σε συστήματα IOT. Ο πελάτης (client), η συσκευή που επικοινωνεί με τον server, εγγράφεται σε συγκεκριμένα θέματα (topics) όπου μπορεί να λάβει μηνύματα, τα οποία δημοσιεύονται από άλλους πελάτες. Η δουλεια του broker είναι να λειτουργήσει ως δρομολογητής αυτής της ροής [59]. Πιο συγκεκριμένα [59]:

1) Έχει λειτουργίες ασφάλειας όπως η προϋπόθεση παροχής κωδικού και όνομα χρήστη πριν την αποστολή κάποιου μηνύματος ή την εγγραφή σε κάποιο θέμα.
2) Υποστηρίζει την υλοποίηση ασφαλής σύνδεσης MQTTS, μέσω SSL/TLS πιστοποιητικού.

3) Λύνει το πρόβλημα χρήσης MQTT από τους browsers, με την τεχνική MQTT over Websockets. Δηλαδή, μέσω μηνυμάτων WebSockets, μπορεί ο κωδικας JavaScript που τρέχει στον φυλλομετρητή να επικοινωνήσει σαν ενα MQTT μήνυμα. Το πρόβλημα υφίστανται στην λειτουργία του MQTT πάνω από το TCP, ενώ τα WebSockets υπάρχουν πάνω από το HTTP με το οποίο έχουν πλήρη υποστήριξη οι browsers.

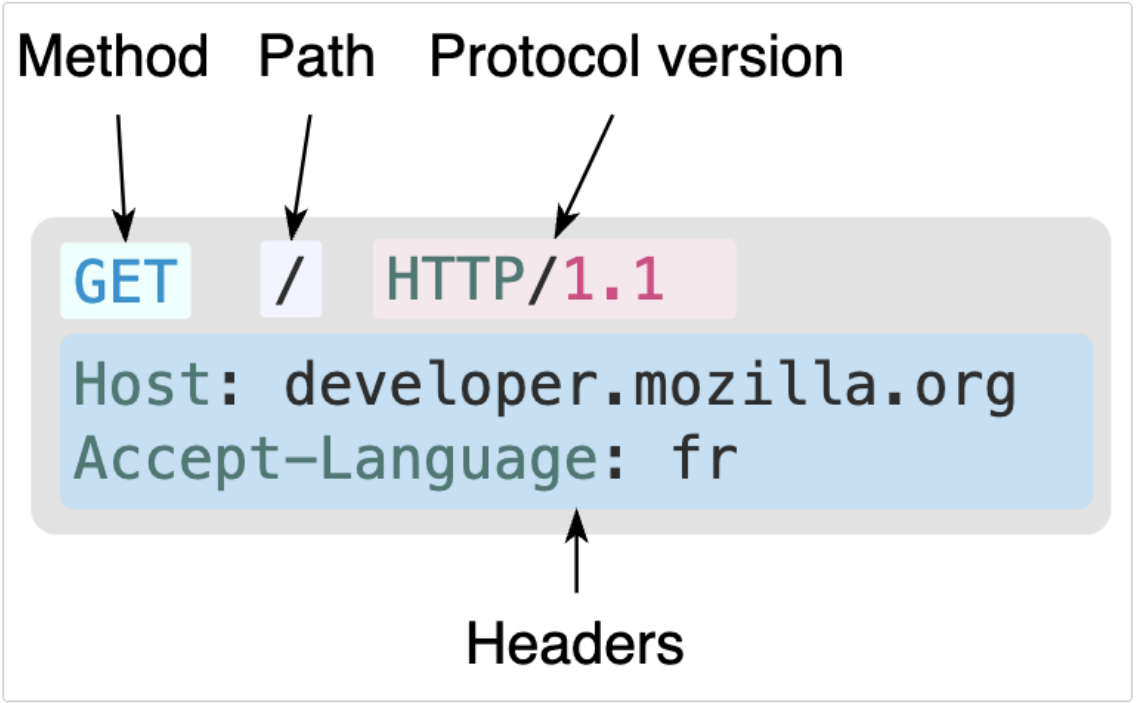
Τέλος, ο Mosquitto, αποτελεί μία συμβατή και αξιόπιστη λύσει στις εφαρμογές πραγματικού χρόνου και χρειάζονται απόδοση της ενέργειας, δηλαδή λειτουργούν με μπαταρία. Συνεπώς, αποτελεί την καθιερωμένη λύση διαχείρισης MQTT μηνυμάτων στα IOT συστήματα.

4.6 Πρωτόκολλα TCP/IP

Το TCP/IP μοντέλο, αποτελεί την βάση για την επικοινωνία του διαδικτύου. Αποτελεί ένα από τα στρώματα του OSI (Open Systems Interconnection) μοντέλου, το οποίο περιγράφει τον τρόπο μεταφοράς πληροφορίας από το επίπεδο bit μέχρι αυτό της εφαρμογής. Αποτελεί την βάση για πολλά πρωτόκολλα ιστοσελίδων, email και domain. Είναι υπεύθυνο για την αξιόπιστη μεταφορά δεδομένων μεταξύ του διαδικτύου [60].

4.6.1 HTTP

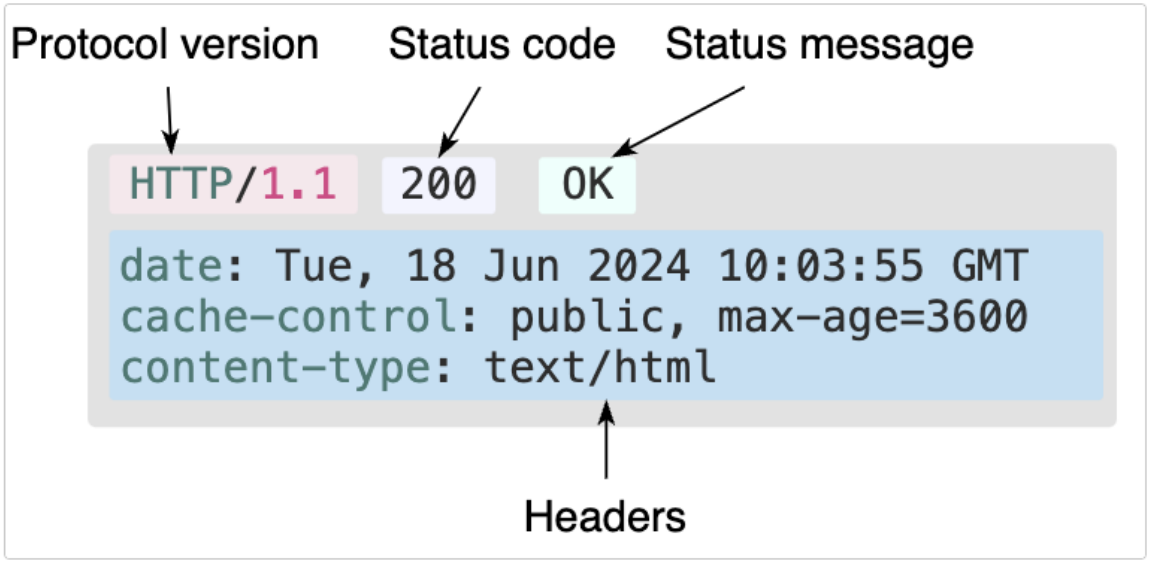
Το HTTP (HyperText Transfer Protocol), αποτελεί το βασικό πρωτόκολλο που χρησιμοποιείται για την επικοινωνία μεταξύ προγραμμάτων στον παγκόσμιο ιστό (World Wide Web). Είναι βασισμένο στο μοντέλο TCP/IP, καθιστώντας την μεταφορά δεδομένων αξιόπιστη και βασίζεται στο μοντέλο αιτήματος απάντησης (request-response). Χρησιμοποιείται κυρίως για την εμφάνιση περιεχομένου κατα την είσοδο σε μία ιστοσελίδα, για την επικοινωνία δεδομένων με Restful API με σκοπό την εμφάνιση και διαχείριση δυναμικού περιεχομένου σε εφαρμογές [61].



Σχήμα 4.19: Παράδειγμα HTTP αιτήματος [62].

Το μοντέλο που ακολουθεί, προϋποθέτει πως κάθε αίτημα προς τον server από τον client θα έχει μία απάντηση (request-response), ακόμα και στην περίπτωση σφάλματος. Ένα αίτημα HTTP αποτελείται από 3 σημεία [61]:

1) Η γραμμή αίτησης (Request Line). Στην γραμμή αυτή συμπεριλαμβάνεται η μέθοδος του αιτήματος, το path του επιθυμητού πόρου στην διεύθυνση προορισμού και η έκδοση του πρωτοκόλλου. Οι μέθοδοι αποτελούνται από τις GET, POST, PUT, DELETE, OPTION κ.α., και χρησιμοποιούνται για να ενημερώσουν τον server τον σκοπό του αιτήματος.
2) Οι κεφαλίδες (Headers). Χρησιμεύουν στην αποστολή key-value meta δεδομένων. Δηλαδή, πληροφορίες αιτήματος και απάντησης, όπως την διεύθυνση προέλευσης, τον τύπο σύνδεσης, τον τρόπο σύνδεσης κ.α.
3) Το σώμα (Body), αποτελεί τα δεδομένα του αιτήματος και είναι προαιρετικό. Τα δεδομένα μπορούν να πάρουν οποιαδήποτε μορφή όπως κείμενο, JSON, Form URL Encoded για key-value pairs και Bytes για την αποστολή αρχείων.

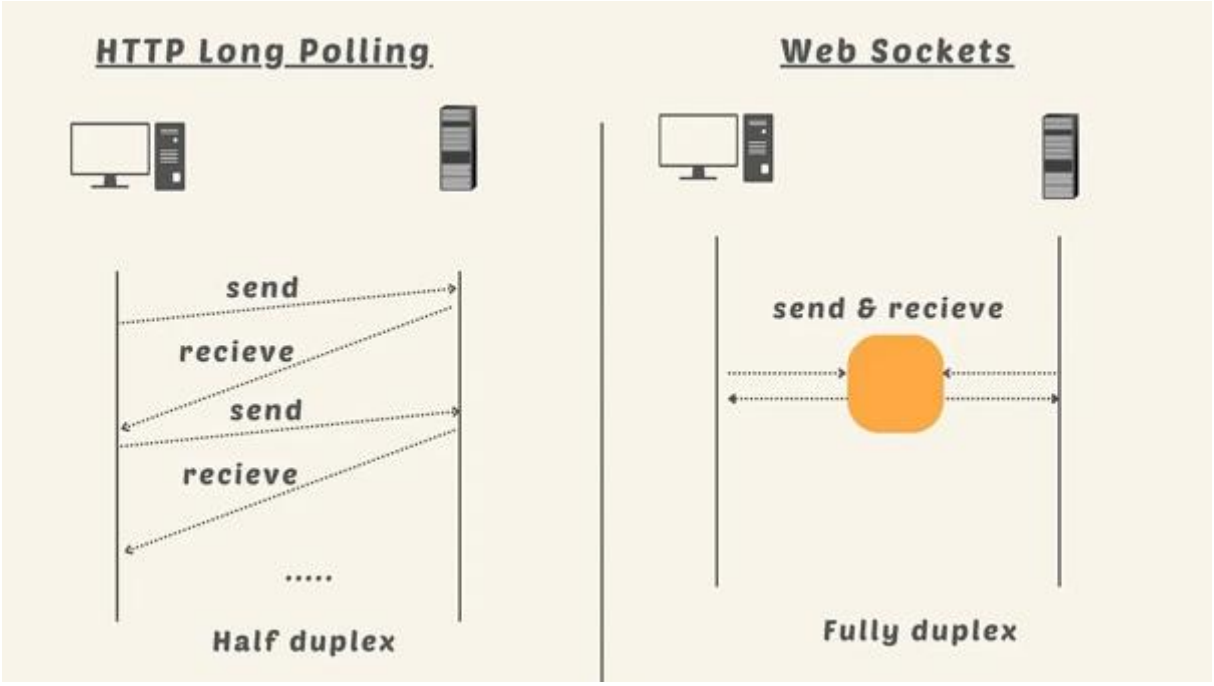


Σχήμα 4.20: Παράδειγμα HTTP απάντησης [62].

Ο διακομιστής, στην συνέχεια απαντάει με ένα Response HTTP μήνυμα παρόμοιας δομής, με την μόνη διαφορά, η πρώτη γραμμή περιέχει την έκδοση HTTP, τον αριθμό και το μήνυμα κατάστασης (status code). Το status code υποδεικνύει την κατάσταση του αιτήματος, με το εύρος των 200 να αναφέρεται στην επιτυχία του request, το εύρος των 400 στο σφάλμα λάθους δεδομένων από μεριά χρήστη και των 500 να αναδεικνύει σφάλμα από μεριά server [61].

4.6.2 WebSockets

Τα WebSockets, είναι ένα πρωτόκολλο στηριζόμενο στο HTTP, το οποίο επιτρέπει αμφίδρομη επικοινωνία μεταξύ client και server. Στο παραδοσιακό HTTP, ο server για να αποστέλλει δεδομένα, χρειάζεται πρώτα κάποιο αίτημα από τον πελάτη, αλλιώς δεν μπορεί καθώς η σύνδεση TCP κλείνει αυτόματα μετά την αποστολή απάντησης. Συνεπώς, εφαρμογές όπου χρειάζεται άμεση ενημέρωση όλων των χρηστών, όπως Live chatting και παιχνίδια, κρίνονται απίθανες μέσω HTTP [63].



Σχήμα 4.21: Διάγραμμα διαφοράς WebSockets από HTTP Polling [64].

Υπάρχουν τεχνικές για την δημιουργία live εφαρμογών μέσω HTTP, το λεγόμενο polling. Με αυτή την μέθοδο, αποστέλλεται κάθε κάποιο μικρό χρονικό διάστημα ένα αίτημα στον server με σκοπό την ενημέρωση του χρήστη. Όμως παρουσιάζει ένα μεγάλο πρόβλημα, στην περίπτωση που το χρονικό διάστημα ενημέρωσης παραμένει άγνωστο. Στο σενάριο αυτό, μία αρκετά μεγάλη τιμή μπορεί να προσθέσει μεγάλες καθυστερήσεις στην ενημέρωση της εφαρμογής, δημιουργώντας μια κακή εμπειρία, και στην περίπτωση που είναι αρκετά μικρό, υπερφορτωνει τον διακομιστή [63].

Αυτό το πρόβλημα έρχεται να λύσουν τα WebSockets, όπου αρχικά δημιουργείται μία σταθερή σύνδεση, εγγραφή, μεταξύ πελάτη-διακομιστή. Αυτή η σύνδεση γίνεται μέσω αιτήματος HTTP με ειδικές επικεφαλίδες υποδεικνύοντας την ανάγκη για αναβάθμισε σε WebSockets και την απάντηση του server με κωδικό κατάστασης 101 αλλαγή πρωτοκόλλων. Μετά την επιτυχή σύνδεση, ο client και ο server μπορούν να ανταλλάσσουν μηνύματα χωρίς περιορισμούς έως την αποσύνδεση του ενός. Τα μηνύματα στέλνονται ως Bytes ή ως κείμενο [63].

Συνοψίζοντας, τα WebSockets αποτελούν την καλύτερη επιλογή για εφαρμογές με ανάγκες ζωντανών ενημερώσεων, λόγω της αμφίδρομης και ελαφριάς φύσης τους.

4.6.3 MQTT

Το πρωτόκολλο MQTT (Message Queuing Telemetry Transport) είναι ένα ελαφρύ πρωτόκολλο για την δημοσίευση και την εγγραφή σε μηνύματα δικτύου. Βασίζεται σε brokers, όπως τον Mosquitto, για να στείλει και να λάβει τα μηνύματα αυτά. Αναπτύχθηκε το 1999 από την IBM και καθιερώθηκε το 2014 και είναι σχεδιασμένο ως μία απλή, γρήγορο και ελαφριά λύση στην επικοινωνία συστημάτων IOT (Internet Of Things), ειδικότερα στους αισθητήρες [58].

Όπως προαναφέρθηκε και στην υποενότητα του Mosquitto, συστήνει τις έννοιες της δημοσίευσης και της εγγραφής σε topics από πελάτες (clients). Δηλαδή, κατά την δημοσίευση μηνύματος ενός πελάτη σε ένα θέμα, οι πελάτες που είναι εγγεγραμμένοι στο ίδιο ακριβώς topic θα λάβουν το μήνυμα. [58]

Τα θέματα αποτελούν μέρη με σκοπό να διαχωριστούν τα διαφορετικού τύπου μηνύματα και αποτελούνται από χαρακτήρες και τα σύμβολα wildcard (#,+). Παράδειγμα topics αποτελούν το /system/sensors/motion, το /system/+/sound και το /system/# [58].

Προσφέρει διαφορετικά επίπεδα QoS (Quality of Service) για την παράδοση μηνυμάτων. Παράδοση το πολύ μία φορά, τουλάχιστον μία φορά και μόνο μία φορά. Η χρήση του κάθε επιπέδου ρυθμίζεται ανάλογα την σοβαρότητα του μηνύματος και τον συνθηκών δικτύου [58].

Μερικά από τα αρνητικά σημεία του πρωτοκόλλου, με το κυριότερο να αποτελεί την μονόδρομη επικοινωνία. Δεν υπάρχει τρόπος, αυτός που έκανε την δημοσίευση να γνωρίζει αν το μήνυμα στάλθηκε και σε ποιον στάλθηκε.

Συνοψίζοντας, η ελαφριά του φύση, το μοντέλο publish/subscribe και το προσαρμοστικό QoS αποτελούν σημαντικά θετικά γνωρίσματα και παρά τα μειονεκτήματα συνιστά μία αξιοπρεπή λύση στην επικοινωνία του IOT.

4.6.4 RTMP

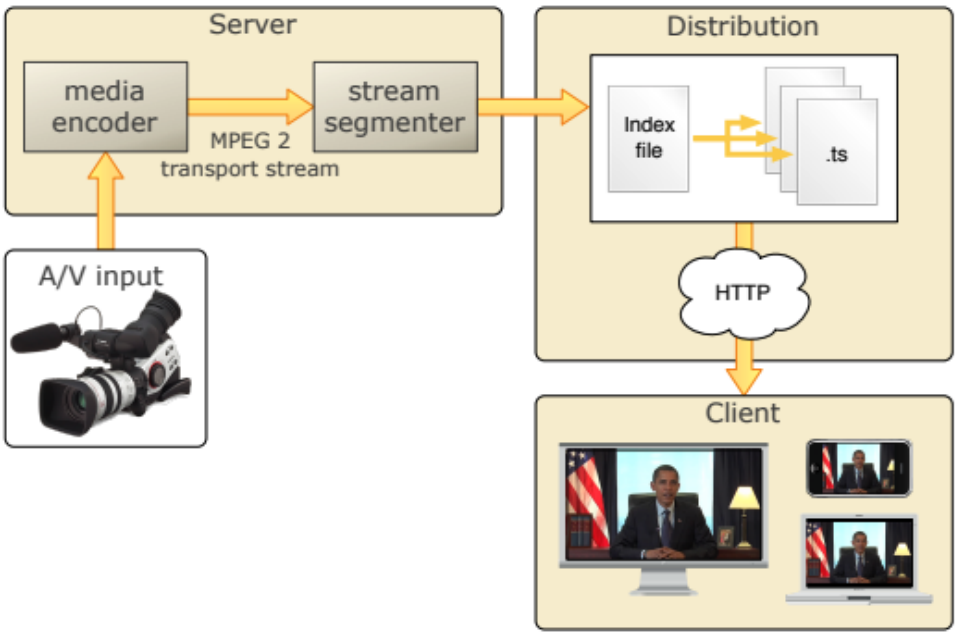
Το πρωτόκολλο Real Time Media Protocol (RTMP), χρησιμοποιείται σε εφαρμογές Live Streaming όπως το Youtube Live και Twitch. Προσφέρει υψηλής αξιοπιστίας μεταφοράς δεδομένων θυσιάζοντας μερική ταχύτητα, καθιστώντας το πιο αργό από αντίστοιχες επιλογές Video Streaming όπως το WebRTC.

Τα δεδομένα του ζωντανής ροής, μεταφέρονται μέσω πακέτων που ονομάζονται Messages (Μηνύματα). Το κάθε Μήνυμα αποτελείται από 5 παιδία, τον τύπο, το μέγεθος, την χρονοσήμανση, το αναγνωριστικό της ροής και το payload, δηλαδή τα πραγματική δεδομένα της ροής. Στην συνέχεια το κάθε Message χωρίζεται σε σταθερού μήκους chunks των μερικών kilobytes. Μέσω αυτού του διαχωρισμού γίνεται δυνατή η σύνθεση ροών video, audio και metadata για την παροχή υψηλού επιπέδου, streaming πρωτοκολλων πολυμέσων όπως το HLS [65].

Η μεταφορά των δεδομένων πραγματοποιείται από τον χρήστη που εκτελεί την ζωντανή ροή, προς έναν ειδικό server στην διαχείριση live media. Μερικά παραδείγματα διακομιστών αποτελούν ο Flash Media Server, Red5 Pro και ο Nginx μέσω του RTMP module. Ο διακομιστής με την σειρά του μετατρέπει τα Μηνύματα που λαμβάνει σε μορφές HLS ή Dash. Αυτό γίνεται για να επιστρέψει στον φυλλομετρητή να δείξει την ζωντανή ροή στον τελικό χρήστη [65].

4.6.5 HLS

Το HLS (HTTP Live Streaming) αποτελεί streaming πρωτόκολλο με προορισμό την συμβατότητα στον browser. Αναπτύχθηκε από την Apple για τον φυλλομετρητή Safari, όμως μπορεί να χρησιμοποιηθεί και από άλλους μέσω ειδικών βιβλιοθηκών. Υποστηρίζει μορφές streaming, on demand, όπως το Netflix, και live όπως το Youtube Live [66].



Σχήμα 4.22: Διάγραμμα ζωντανής ροής από την πηγή στον τελικό χρήστη [66].

Το HLS, χωρίζει την ζωντανή ροή, που δέχεται μέσω RTMP, σε κομμάτια (chunks) TS (Transport Stream), δηλαδή σε chunks μερικών δευτερολέπτων ανάλογα την ρύθμιση που έχει υποστεί. Στην συνέχεια δημιουργείται ένα αρχείο m3u8 το οποίο λειτουργεί ως playlist παρέχοντας την λίστα από τα διαθέσιμα chunks. Τέλος παρέχεται στον player για Safari, και στην ειδική βιβλιοθήκη HLS για τους υπόλοιπους browser, το path του m3u8 και ξεκινάει η απεικόνιση της ροής στον χρήστη [66].

Συνοψίζοντας, το HLS αποτελεί μία καλή επιλογή για Live streaming σε περιβάλλον browser, ειδικότερα στις εφαρμογές της Apple και παρέχει απλούστερη υλοποίηση με παρόμοιες αποδόσεις σε αντίστοιχες τεχνολογίες όπως το Dash [66].

4.7 RFCOMM

Το RFCOMM (Radio Frequency Communication Protocol) αποτελεί πρωτόκολλο της τεχνολογίας Bluetooth. Επιτρέπει την σειριακή επικοινωνία ασύρματα μέσω μιας απλής και αξιόπιστης ροής δεδομένων παρόμοια με το TCP και στηρίζεται πάνω στο BlueZ stack. Παρακάτω θα ακολουθήσει μια εισαγωγή στο Bluetooth και BlueZ και ύστερα θα συνεχιστεί η ανάλυση του RFCOMM.

Το Bluetooth, είναι μιας χαμηλού κόστους, χαμηλής ενέργειας διεπαφή ράδιο και λειτουργεί στην μπάντα των 2.45 GHz. Επιτρέπει στις συσκευές να συνδεθούν και να επικοινωνήσουν ασύρματα σε μικρές αποστάσεις. Επιπλέον χρησιμοποιεί Frequency-Hop, δηλαδή χωρίζει το διαθέσιμο εύρος συχνότητας σε πολλά κανάλια τα οποία αλλάζει για αποφυγή παρεμβολών. Τέλος, αποτελεί διαδεδομένη τεχνολογία για επικοινωνία και σύνδεση πολυμέσων, όπως ασύρματα ακουστικά [67].

Το BlueZ stack, παρέχει υποστήριξη για τα στρώματα και πρωτόκολλα του Bluetooth και ενσωμάτωση με τον Linux Kernel. Αποτελεί το απόλυτο εργαλείο διαχείρισης της διεπαφής Bluetooth σε Linux συσκευές που διαθέτουν τον αντίστοιχο Bluetooth αντάπτορα [67].

Το RFCOMM, λειτουργεί ως ένα εικονικό serial port, για παράδειγμα είναι προσβάσιμο σε Linux συσκευες στο path /dev/rfcomm0. Έτσι μετά το άνοιγμα σύνδεσης, που διαχειρίζεται το εργαλείο

BlueZ, είναι εφικτό προγραμματιστικά το άνοιγμα αυτού της σειριακής πύλης, καθώς και η λήψη και αποστολή δεδομένων, μέσω αυτής, στην συνδεδεμένη μέσω Bluetooth συσκευή [67].

Συνοψίζοντας, το RFCOMM, αποτελεί ένα πρωτόκολλο του κλασικού Bluetooth, με στόχο την αξιοπιστία και την απλότητα, κάνοντας το ιδανικό σε συστήματα με μικρή υλοποίηση χωρίς την ανάγκη για αποδοτικότερη χρήση της ενέργειας. Σε αντίθεση, άλλες επιλογές περιλαμβάνουν το BLE (Bluetooth Low Energy) το οποίο αποσκοπεί ιδανικό σε καταστάσεις IOT συσκευών που τρέφονται από μπαταρία [67].

4.8 SMS

Το πρότυπο (Global System for Mobile Communications), ή αλλιώς τεχνολογία 2G, έφερε την επανάσταση στην ασύρματη επικοινωνία με την παρουσίαση του το 1990. Είναι η πρώτη τεχνολογίας της γενιάς αυτής με κύριο χαρακτηριστικό την αποστολή και λήψη SMS. Ύστερα ακολούθησαν οι γενιές 3G, 4G με την πιο καινούρια να αποτελεί το 5G [68].

Τα SMS (Short Message Service), αποτελούν τα γνωστά σε όλους μηνύματα μέσω κινητών συσκευών. Διαχειρίζονται από τους παρόχους κινητής τηλεφωνίας, μέσω του SMS center (SMSC). Ο μηχανισμός αυτός στέλνει SMS μηνύματα στην τελική συσκευή με μέγιστο όριο 160 χαρακτήρων με 7-bit κωδικοποίηση, με υποστήριξη 128 διαφορετικών χαρακτήρων. Σε περίπτωση ξεπερασης των ορίων, το μήνυμα αποστέλλεται σε κομμάτια. Επιπλέον παρέχει μηχανισμούς διαχείρισης επαναποστολής των μηνυμάτων σε περίπτωση που ο παραλήπτης είναι εκτός δικτύου, δεν έχει παροχή σήματος [68].

Συνοψίζοντας, τα SMS αποτέλεσαν την πρώτη μορφή γραπτής επικοινωνίας μέσω κινητών, πλέον όμως η χρήση τους γίνεται ξεπερασμένη, λόγο της ευκολοτερης, ταχύτερης, πλουσιοτερης και φθηνότερης ανταλλαγής μηνυμάτων που προσφέρων διαδικτυακές υπηρεσίες όπως τα Viber, Messages και WhatsApp. Παρ'όλα αυτά, χρησιμοποιούνται ως διπλή ταυτοποίηση της ταυτότητας των ανθρώπων από εφαρμογές [68].

4.9 SSL/TLS

Η επικοινωνία στο διαδίκτυο γίνεται μέσω μεταφοράς bits, είτε μέσω καλωδίου είτε ασύρματα. Οποιοδήποτε σύστημα μπορεί να τα ανιχνεύσει και να ερμηνεύσει την σημασία τους, αποκτώντας πρόσβαση σε ευαίσθητες πληροφορίες όπως κωδικοί τράπεζας, περιεχόμενα αρχείων, έτσι, για τον λόγο αυτό δημιουργήθηκαν τα πρωτόκολλα ασφάλειας SSL (Secure Sockets Layer) και στην συνέχεια εξελίχθηκε στο TLS (Transport Layer Security) με την νέα ονομασία SSL/TLS. Με σκοπό την προστασία των δεδομένων μεταξύ πελάτη και διακομιστή [69].

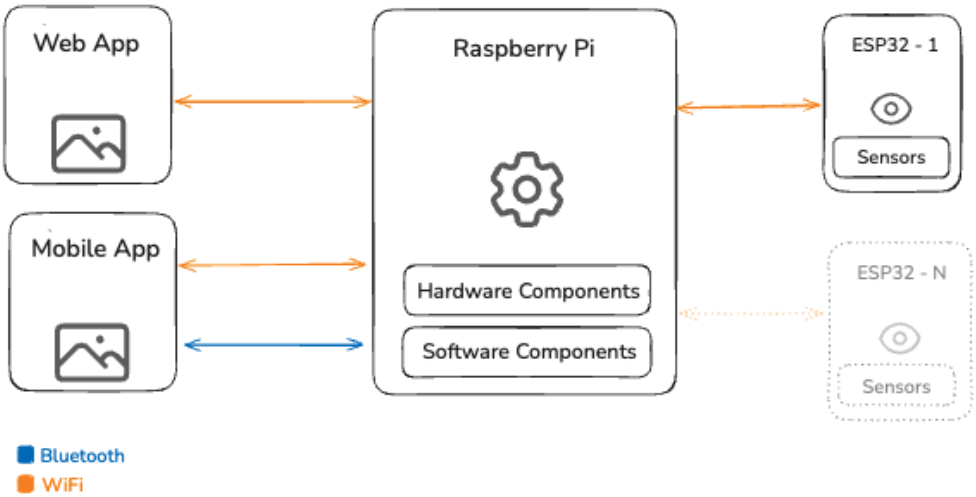
Τα πρωτόκολλα αυτά χρησιμοποιούνται από όλα τα πρωτόκολλα επικοινωνίας TCP. Στόχος όπως προαναφέρθηκε αποτελεί η προστασία των δεδομένων, μέσω κρυπτογραφίας, η ακεραιότητα τους βεβαιώνοντας ότι δεν έχουν αλλοιωθεί κατά την μεταφορά και η πιστοποίηση της ταυτότητας του server μέσω ψηφιακών πιστοποιητικών [69].

Για να λειτουργήσει ένας διακομιστής μέσω HTTPS, είναι αναγκαία η λήψη ψηφιακού πιστοποιητικού από μία Αρχή Πιστοποίησης (Certificate Authority). Χωρίς την εκτέλεση αυτής της ενέργειας, ο server ερμηνεύεται ως μη ασφαλής από τους φυλλομετρητές με αποτέλεσμα να μην προτείνεται η επίσκεψη του μέσω σχετικής προειδοποίησης [69].

Τέλος η χρήση των πιστοποιητικών αυτών, αποτελεί αναγκαία πράξη κάθε οργανισμού στον ιστότοπο του για την εξασφάλιση της ασφάλειας των πελατών και την ομαλότερη εμπειρία χρήσης.

Κεφάλαιο 5ο: Αρχιτεκτονική

5.1 Επισκόπηση συστήματος



Σχήμα 5.1: Γενικό διάγραμμα δομικών στοιχείων του συστήματος.

Το κεφάλαιο αυτό, θα επικεντρωθεί στην επισκόπηση του συστήματος, όπως φαίνεται και στην εικόνα 5.1. Το Raspberry Pi, αποτελεί την καρδιά του συστήματος. Συνδέεται με τα υπόλοιπα στοιχεία κυρίως μέσω WiFi και Bluetooth. Αρχικά, θα γίνει αναφορά στην ροή λειτουργίας των τρόπων επικοινωνίας, στην συνέχεια θα εξεταστεί η συνεισφορά του ESP32 και τέλος η χρήση του frontend.

5.1.1 Επικοινωνία στοιχείων

Το σύστημα επικοινωνεί μέσω WiFi και Ethernet, δηλαδή είναι διαθέσιμο μέσω του τοπικού δικτύου. Οι εφαρμογές Web και Mobile και ESP32, γνωρίζουν την διεύθυνση του συστήματος και συνάπτουν επικοινωνία. Η σύνδεση μέσω Bluetooth ενεργοποιείται στην περίπτωση που η σύνδεση Internet δεν είναι δυνατή για κάποιο χρονικό διάστημα, με σκοπό την παροχή διαπιστευτηρίων WiFi από τον χρήστη. Πιο συγκεκριμένα, αυτή η διαδικασία γίνεται με την χρήση της Mobile εφαρμογής, η οποία ελέγχει της πρόσβαση της με τον μικροϋπολογιστή μέσω WiFi, και στην περίπτωση αποτυχίας, ενεργοποιείται το Bluetooth, του κινητού, και πραγματοποιείται pairing, όπου ζητείται από τον χρήστη η εισαγωγή των στοιχείων για την σύνδεση τοπικού δικτύου. Τέλος, κατά την επιτυχής ολοκλήρωσης της διαδικασίας, δηλαδή την σύνδεση στο δίκτυο, απενεργοποιείται ο αντάπτορας Bluetooth, του Raspberry Pi, για την εξοικονόμηση ενέργειας.

Η ύπαρξη του ESP32, εξυπηρετεί στις ανάγκες αύξησης της αποδοτικότητας ενέργειας του συστήματος, καθώς και στην καλύτερη αποθήκευση βίντεο στον αποθηκευτικό χώρο του συστήματος. Η επικοινωνία μεταξύ των δύο, πραγματοποιείται μέσω WiFi. Αυτή η λειτουργία, παρουσιάζει ένα σημαντικό πρόβλημα, αφού ο μικροελεγκτής δεν γνωρίζει τα στοιχεία του του τοπικού δικτύου στο οποίο υπάρχει το σύστημα. Στην περίπτωση αυτή, δημιουργεί τον δικό του WiFi, μέσω του αντάπτορα που έχει, λειτουργώντας ως WiFi AP (Access Point). Τότε, το Raspberry Pi, σκανάρει τα δίκτυα και συνδέεται

όπου αποστέλλει τα δεδομένα WiFi που έχει προσκομίσει προηγουμένως. Ο ESP32 κατά την λήψη των διαπιστευτηρίων αλλάζει το mode ως Station και συνδέεται στο τοπικό δίκτυο. Αυτή η διαδικασία επαναλαμβάνεται στις περιπτώσεις όπου για ένα χρονικό διάστημα είναι μη δυνατή η εύρεση του συστήματος. Τέλος, η διαδικασία μπορεί να εκτελεστεί για περισσότερους από έναν ESP32.

5.1.2 Δομή Frontend

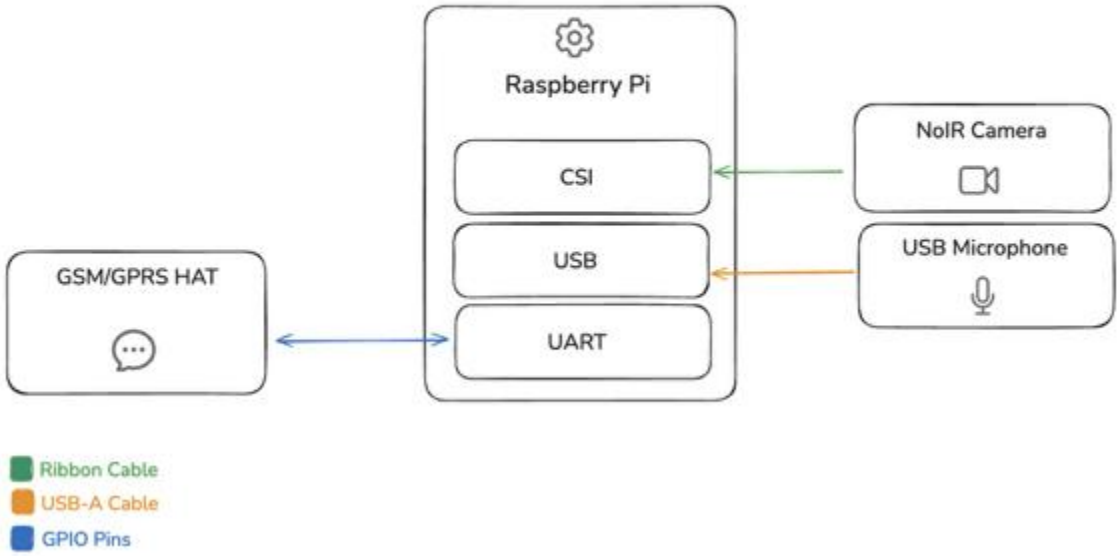
Η επιλογή της ανάπτυξης ιστοσελίδας, έγινε με γνώμονα την ευκολία πρόσβασης από οποιαδήποτε συσκευή η οποία είναι συνδεδεμένη στο τοπικό δίκτυο. Μέσω αυτής, παρέχονται πολλά εργαλεία στον χρήστη για να μπορεί να διαχειριστή το σύστημα πλήρως αλλά και να λαμβάνει άμεσες πληροφορίες σχετικά με το οτιδήποτε. Παρακάτω ακολουθούν οι διαθέσιμες λειτουργίες του Web App:

1) Σύνδεση στο σύστημα, μέσω Google.
2) Τροποποίηση των ρυθμίσεων του συστήματος στα δικά του θέλω.
3) Παρακολούθηση κατάστασης του συστήματος γενικά αλλά και κάθε εξαρτήματος.
4) Παρακολούθηση ζωντανής ροής βίντεο.
5) Διαχείριση ενεργοποίησης κάμερας.
6) Πρόσβαση σε αποθηκευμένα βίντεο καταγραφών εισβολών.
7) Ζωντανή ενημέρωση συμβάντων, π.χ. καταγραφή κίνησης, αναγνώριση εισβολέα.
8) Αποστολή βίντεο, που ενδείκνυται χαρακτηριστικά προσώπου, στο σύστημα για την εκπαίδευση του Face Recognition μοντέλου.
9) Πρόσβαση στα δεδομένα του συστήματος.

Η ανάπτυξη Mobile εφαρμογής, έχει στόχο την εκμετάλλευση του αντάπτορα Bluetooth που διαθέτουν τα smartphones. Όπως αναφέρθηκε προηγουμένως, η σύνδεση μέσω Bluetooth αποτελεί τρόπο αποστολής διαπιστευτηρίων WiFi στο σύστημα. Επιπλέον, μέσω τεχνικών, που θα συζητηθούν στο κεφάλαιο 6, υπάρχει η δυνατότητα εμφάνισης της ιστοσελίδας μέσω της Mobile συσκευής, για την ευκολία του χρήστη. Τέλος, σημαντικό γεγονός αποτελεί η αποστολή Push Notifications στο κινητό μέσω του συστήματος, ακόμα και στην περίπτωση που η εφαρμογή είναι κλειστή.

5.2 Διασύνδεση υλικού

5.2.1 Raspberry Pi

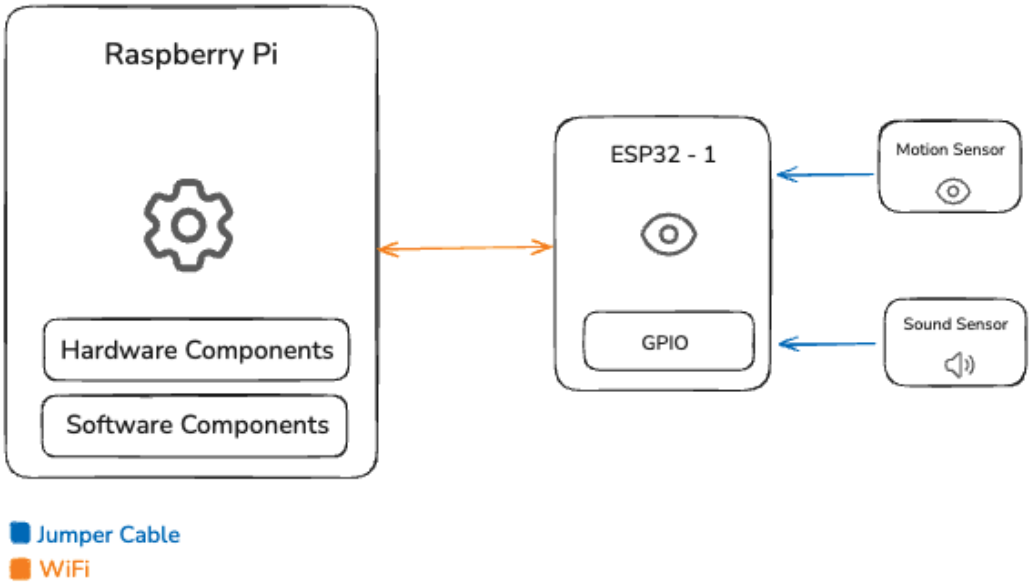


Σχήμα 5.2: Διάγραμμα σύνδεσης Raspberry Pi διεπαφών με περιφερειακά.

Το σύστημα αποτελείται από 3 εξαρτήματα, περιφερειακά, τα οποία συνδέονται στο Raspberry Pi μέσω διάφορον διεπαφών, των CSI, USB 3.0 και UART.

1) Η διεπαφή CSI (Camera Serial Interface), έχει ως κύριο σκοπό την σύνδεση κάμερας συμβατής με Raspberry Pi, όπως οι κάμερες της σειράς Raspberry Pi. Μέσω αυτής και του Ribbon Cable, είναι δυνατή η μεταφοράς δεδομένων εικόνων προς τον μικροϋπολογιστή.
2) Η θύρα USB 3.0. Αποτελεί 1 απο τις 2 USB 3.0 και τις 2 USB 2.0, οι οποίες παρέχονται από το μηχάνημα. Χρησιμοποιείται για την σύνδεση περιφερειακών, όπως πληκτρολόγιο, ποντίκι και μικρόφωνο. Στην συγκεκριμένη περίπτωση, πραγματοποιείται, μέσω της θύρας, η λήψη δεδομένων ήχου.
3) Το UART, όπως προαναφέρθηκε, στο κεφάλαιο 3.5, επιτρέπει την ασύγχρονη επικοινωνία μεταξύ των δύο συσκευών. Πιο συγκεκριμένα το σύστημα μπορεί να στέλνει και να λαμβάνει δεδομένα SMS.

5.2.2 ESP32

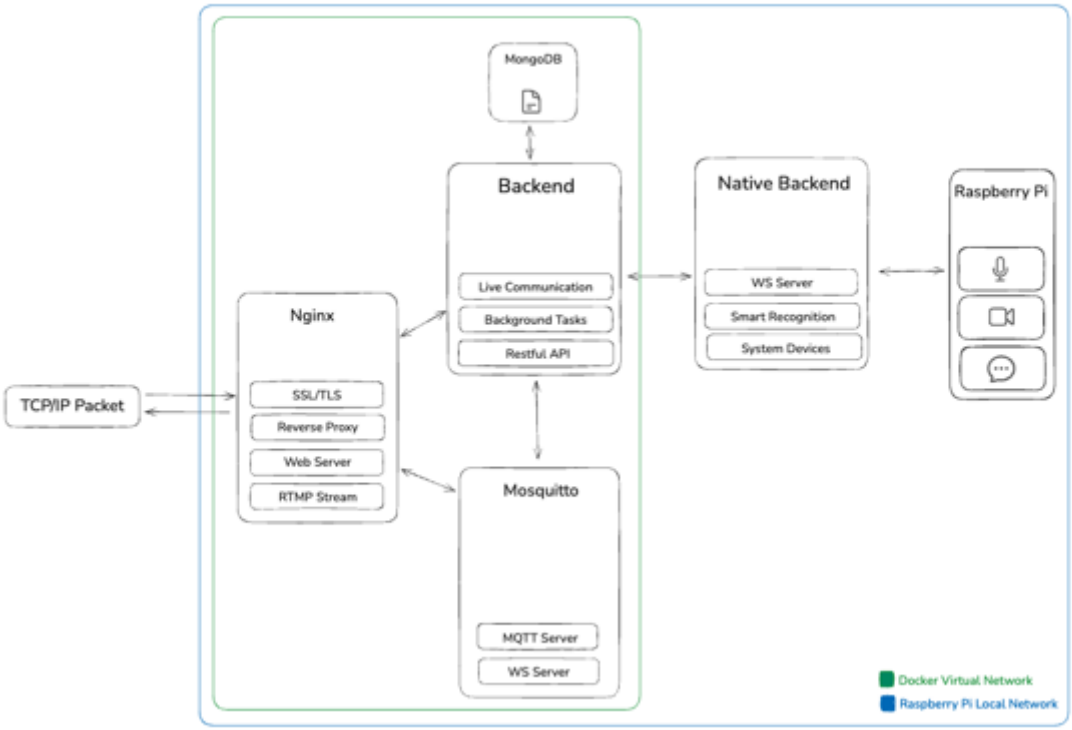


Σχήμα 5.3: Διάγραμμα σύνδεσης ESP32 με αισθητήρες και Raspberry Pi.

Η παρόν ενότητα, εστιάζει τον τρόπο χρήσης των αισθητήρων μέσω του μικροελεγκτή ESP32. Όπως φαίνεται στο σχήμα, η σύνδεση, των αισθητήρων, πραγματοποιείται μέσω Jumper Cables στα GPIO pins του. Η ροή λειτουργίας του μικροελεγκτή έχει ως εξής:

1) Μπαίνει σε λειτουργία ύπνου για την καλύτερη απόδοση της ενέργειας.
2) Δέχεται σήμα από κάποιον αισθητήρα, είτε κίνησης είτε ήχου.
3) Ξυπνάει, συνδέεται στο WiFi και στέλνει αντίστοιχο μήνυμα στο σύστημα.
4) Σε περίπτωση αδυναμίας σύνδεσης στο WiFi, αλλάζει σε λειτουργία AP αναμένοντας τα στοιχεία WiFi.
5) Τέλος, ξανα μπαίνει σε λειτουργία ύπνου, απολαμβάνοντας την διαδικασία επ αόριστον.

5.3 Αρχιτεκτονική υπηρεσιών



Σχήμα 5.4: Διάγραμμα ροής της επικοινωνίας των υπηρεσιών του συστήματος.

Οι υπηρεσίες του συστήματος, όπως φαίνεται στο σχήμα 5.4, λειτουργούν μέσω του Raspberry Pi. Οι λόγοι ύπαρξης τους, αποτελούν την αποστολή πληροφοριών στον χρήστη και την διαχείριση των συνδεδεμένων συσκευών. Στις επόμενες υποενότητες του κεφαλαίου αυτού, θα γίνει ανάλυση του διαγράμματος αυτό. Αρχικά, θα παρουσιαστεί η γενική ροή λειτουργίας των υπηρεσιών, στην συνέχεια η δομή Docker, η ροή του Live Streaming και τέλος η αποστολή ειδοποιήσεων στον χρήστη.

5.3.1 Λειτουργία υπηρεσιών

Οι υπηρεσίες, αποτελούνται από 2 Backend, 2 διακομιστές, τους Nginx και Mosquitto και μία βάση δεδομένων. Όπως προαναφέρθηκε, σκοπός των υπηρεσιών είναι:

1) Live ενημέρωση του χρήστη για τα χαρακτηριστικά του συστήματος.
2) Πρόσβαση δεδομένων από τον χρήστη, για ανάγνωση και τροποποίηση.
3) Διαχείριση περιφερειακών συσκευών Raspberry Pi.
4) Διαχείριση μικροελεγκτών ESP32 μαζί με τους αισθητήρες.

Αρχικά, μέσω των διεπαφών CSI, UART και USB παρέχεται η δυνατότητα εισαγωγής και εξαγωγής δεδομένων από τις συσκευές κάμερας, μικροφώνου και GSM/GPRS Modem. Η λειτουργία αυτή, ανήκει στην πρώτη υπηρεσία του συστήματος, την Native Backend. Η οποία βρίσκεται εκτός Docker Network για την καλύτερη διαχείριση της Raspberry Pi κάμερας μέσω ειδικών driver-βιβλιοθηκών. Συνεπώς, οι συσκευές της κάμερας και του μικροφώνου ανήκουν σε αυτή, εκτός της συσκευής για αποστολή και λήψη SMS, η οποία θα αναφερθεί αργότερα. Όπως φαίνεται στο σχήμα 5.4, υπάρχει η δυνατότητα Smart Recognition, δηλαδή, χρησιμοποιεί την κάμερα και το μικρόφωνο και την κάθε ροή της στέλνει

για αναγνώριση εικόνας και ήχου. Περισσότερα για την έξυπνη αναγνώριση θα αναφερθούν στην ενότητα 5.4. Τέλος, είναι υπεύθυνο για την παροχή διαχείρισης των περιφερειακών μέσω του κύριου Backend, του συστήματος, καθώς και η ζωντανή ενημέρωση του για την κατάσταση των υλικών συσκευών.

Η υπηρεσία Backend, αποτελεί το κύριο κομμάτι του συστήματος. Ενσωματώνει όλες τις υπηρεσίες του συστήματος, πιο συγκεκριμένα:

1) Διαχειρίζεται τις συνδέσεις του Mosquitto Server. Υλοποιεί λογική για την απόκριση αιτημάτων Websockets, από τον χρήστη, και MQTT από τους αισθητήρες μέσω του ESP32.
2) Λειτουργεί ως μέσο προσπέλασης της βάσης δεδομένων, μέσω του χρήστη ως Restful API.
3) Διαχειρίζεται την λήψη και αποστολή μηνυμάτων SMS μέσω του GSM/GPRS Module.
4) Επικοινωνεί ζωντανά με την υπηρεσία του Backend Native, για την επίβλεψη των υλικών συσκευών και την λήψη των προβλέψεων μέσω της έξυπνης αναγνώρισης.
5) Υπολογίζει την πιθανότητα εισβολής για την ενεργοποίηση του συναγερμού .
6) Εκτελεί λειτουργίες ειδοποίησης και καταγραφής κατά έναρξη την εισβολής.

Ο διακομιστής Nginx, λειτουργεί ως διαμεσολαβητής των εξωτερικών συνδέσεων, για την πρόσβαση του συστήματος. Πιο ειδικά παρέχει τις ακόλουθες λειτουργίες:

1) Σερβίρει τα αρχεία της ιστοσελίδας στον χρήστη.
2) Διαχειρίζεται την ζωντανή ροή RTMP για την δημιουργία ζωντανού βίντεο.
3) Λειτουργεί ως Reverse Proxy, όλα τα αιτήματα δικτύου περνάνε πρώτα από αυτόν και μετά προωθούνται, αν χρειάζεται, στις υπόλοιπες υπηρεσίες.
4) Ενσωματώνει ασφάλεια επικοινωνίας μέσω SSL/TLS πιστοποιητικού για όλο το δίκτυο του συστήματος.

Τέλος, ο Mosquitto broker, παρέχει συνδέσεις MQTT και WebSockets, μαζί με κωδικούς διαπιστευτηρίων για ιδιωτικότητα. Τις συνδέσεις, τις εκμεταλλεύεται το Backend, για την λήψη μηνυμάτων αισθητήρων μέσω του ESP32 και την Live ενημέρωση του χρήστη.

5.3.2 Δομή Docker

Το Docker δημιουργεί εικονικά περιβάλλοντα, διαφορετικό για κάθε υπηρεσία. Δηλαδή, το κάθε σύστημα που υπάρχει μέσα στο δίκτυο του, αποτελεί την δική του εικονική μηχανή. Πιο συγκεκριμένα η δομή περιέχει:

1) Την βάση δεδομένων MongodB.
2) Την υπηρεσία Backend.
3) Τον Mosquitto Broker.
4) Τον διακομιστή Nginx.

Η χρήση, Docker συμβάλλει σε πολλά θετικά στοιχεία, ειδικά σε συστήματα που συνδυάζουν πολλές υπηρεσίες. .Μερικά από τα πλεονεκτήματα χρήσης Docker στο παρόν σενάριο αποτελούν:

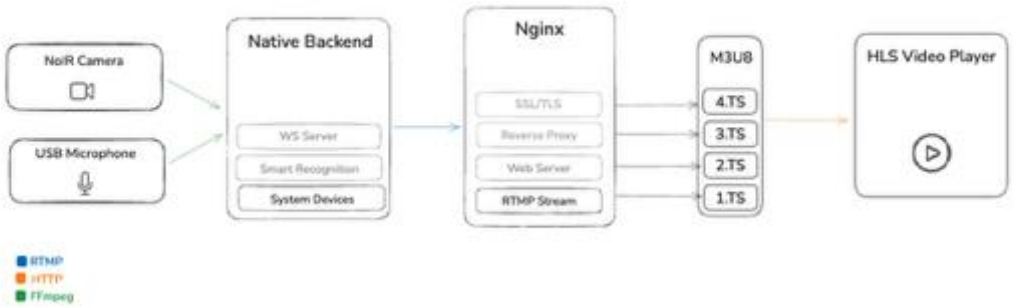
1) Η κάθε υπηρεσία, έχει μόνο τις δικές τις απαραίτητες βιβλιοθήκες. Βοηθάει στο να μείνει το Raspberry Pi καθαρό από τα πολλά dependencies, την αποφυγή conflicts μεταξύ βιβλιοθηκών.

2) Η αναπαραγωγή του συστήματος σε διαφορετικό Raspberry Pi είναι αρκετά γρηγορότερη και ευκολότερη. Χρειάζεται μόνο το source code και το Docker.

3) Μέσω του Docker Compose, γίνεται ομαδοποίηση όλων των υπηρεσιών, συμβάλλοντας στην ομαλότερη διαχείριση τους.

Συνοψίζοντας, η χρήση Docker συνιστά μία εξαιρετική επιλογή, καθώς αυξάνει την αποδοτικότητα της ανάπτυξης και βελτιώνει την αξιοπιστία στην παραγωγή. Όμως, η δημιουργία εικονικού περιβάλλοντος, φέρνει και αρνητικά γνωρίσματα. Το πιο σημαντικό, αποτελεί η δυσκολία χρήσης Native drivers. Για παράδειγμα, η αποδοτική χρήση της Raspberry Pi Camera, επιβάλει ειδικούς drivers στο λειτουργικό σύστημα Raspberry Pi OS, η χρήση Docker για την αξιοποίηση των βιβλιοθηκών αυτών απαιτεί την χειροκίνητη εγκατάσταση και το χτίσιμο τους από την αρχή με το συγκεκριμένο λειτουργικό σύστημα, γεγονός που συστήνει αρκετές δυσκολίες στην υλοποίηση.

5.3.3 Ροή Live Streaming



Σχήμα 5.5: Διάγραμμα ροής Video και Audio Streaming στον τελικό χρήστη.

Στο σχήμα 5.5, παρατηρείται η ροή της του Video και Audio Streaming, από τις συσκευές, συνδεδεμένες στο Raspberry Pi, έως τον Player του πελάτη.

Η ροή, ξεκινάει από την διαχείριση μέσω του Native Backend και στην συνέχεια, ο έλεγχος περνάει στον διακομιστή Nginx. Πιο συγκεκριμένα οι ενέργειας αποτελούνται από:

1) Εισαγωγή και ένωση των δύο Video και Audio Streams μέσω του εργαλείου FFmpeg, για τον ήχο, και Libcamera, για την κάμερα.

2) Χρήση του FFmpeg για την ρύθμιση των FPS, του bitrate, της ποιότητας, της ανάλυσης, της κωδικοποίησης και άλλων παραμέτρων.

3) Εξαγωγή μέσω FFmpeg ως RTMP Stream στον Nginx Server.

4) Διαχωρισμός της ζωντανής ροής σε M3U8 playlist που περιέχει τα αρχεία MPEG-2 TS. Αρχεία με τα δεδομένα της ροής, ρυθμιζόμενη από τον διακομιστή, συμβατά με τον HLS Player.

5) Αποστολή στον HLS Video Player, υπό αίτημα, για την αναπαραγωγή του ζωντανού βίντεο.

Συνοψίζοντας, το βίντεο στην οθόνη του πελάτη, περνάει από 5 βήματα, χωρίς όμως την εισαγωγή μεγάλης καθυστέρησης. Αυτό οφείλεται στην συμβατή με Raspberry Pi Camera βιβλιοθήκη, την Libcamera, η οποία, πολύ αποδοτικά προσφέρει βίντεο υψηλής ποιότητας, αλλά και στις ρυθμίσεις και κωδικοποιήσεις που εφαρμόζονται μέσω του εργαλείου FFmpeg.

5.3.4 Αποστολή ειδοποιήσεων

Το σύστημα έχει ως κύριο στόχο, την άμεση ενημέρωση του χρήστη, την στιγμη που το σύστημα κρίνει, πως υπάρχει εισβολή στον χώρο όπου φυλάει. Η ενημέρωση αυτή πραγματοποιείται με την αποστολή ειδοποιήσεων και χωρίζεται σε δύο κατηγορίες τα Push Notifications και την αποστολή SMS.

Τα Push Notifications, είναι ένας τύπος ειδοποιήσεων που γίνεται από το λειτουργικό σύστημα στα smartphones. Βασική προϋπόθεση, για την αποστολή, ο χρήστης να έχει εγκαταστήσει και να έχει ανοίξει την Mobile εφαρμογή έστω μια φορά. Η εφαρμογη δεν χρειάζεται να είναι ούτε στο προσκήνιο ούτε στο παρασκήνιο. Κατά την λήψη ειδοποίησης, το κινητό δονείται και εμφανίζεται, ένα ορθογώνιο κουτί, στο πάνω μέρος της οθόνης. Ο χρήστης μπορεί να το πατήσει και να μεταφερθεί στην ιστοσελίδα, μέσω της εφαρμογής, όπου μπορεί να δει και να διαχειριστεί το Live Stream.

Από την άλλη, η αποστολή SMS, αποτελεί μία πιο άμεση διαδικασία. Ο χρήστης δεν χρειάζεται να έχει εγκαταστήσει κάποια εφαρμογή, η μόνη προϋπόθεση, να έχει γνώση τον αριθμό τηλεφώνου του το σύστημα. Την γνώση αυτή, την λαμβάνει το σύστημα από τους εγγεγραμμένους χρήστες. Τέλος, μέσω κινητής τηλεφωνίας, συγκεκριμένα μέσω GSM/GPRS τεχνολογιών, πραγματοποιείται η αποστολή SMS με διαμεσολαβητή τον αντίστοιχο πάροχο κινητής τηλεφωνίας.

5.4 Έξυπνη Αναγνώριση

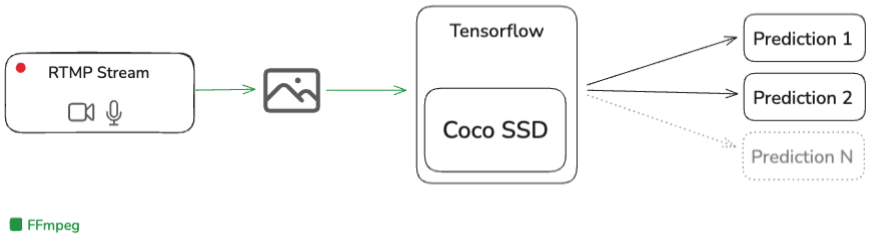
Το σύστημα, έχει λειτουργίες έξυπνης αναγνώρισης, αναλύει τον ήχο και την εικόνα, του μικροφώνου και της κάμερας αντίστοιχα. Με στόχο να καταλάβει, αν ο χώρος που φιλάει, βρίσκεται υπό περίπτωση εισβολής. Τον στόχο, αυτό τον πετυχαίνει μέσω της χρήσης μοντέλων μηχανικής μάθησης όπως το Coco SSD, το Yamnet και το ArcFace. Επίσης για την πιο αποδοτική χρήση των μοντέλων, χρησιμοποιούνται τα εργαλεία Tensorflow και InsightFace.

Το Coco SSD (Single Shot MultiBox Detection), είναι ένα προεκπαιδευμένο μοντέλο αναγνώρισης αντικειμένων σε μια εικόνα. Βασίζεται στην αρχιτεκτονική SSD, που είναι ειδική για την αναγνώριση αντικειμένων ζωντανά (real-time). Έχει εκπαιδευτεί στο Coco dataset, μία μεγάλης κλίμακας βιβλιοθήκη με δυνατότητες αναγνώρισης 80 διαφορετικών αντικειμένων. Το σύστημα χρησιμοποιεί το μοντέλο αυτό και τις δυνατότητες του για την αναγνώριση των αντικειμένων που το ενδιαφέρον, όπως ανθρώπους, κατοικίδια, σακίδια και αιχμηρά αντικείμενα.

Το Yamnet, αποτελεί προεκπαιδευμένο μοντέλο αναγνώρισης ήχου. Συγκεκριμένα, έχει βασιστεί στο AudioSet dataset, το οποίο περιέχει 2,1 εκατομμύρια αρχεία ήχου από το Youtube για 632 διαφορετικούς ήχους, από τους οποίους 521 μπορεί να αναγνωρίσει το μοντέλο Yamnet. Το σύστημα χρησιμοποιεί όλους τους ήχους, φιλτράροντας διάφορους, μη σημαντικούς, θορύβους.

Το ArcFace, είναι ένα προεκπαιδευμένο μοντέλο αναγνώρισης προσώπων του InsightFace. Ενσωματώνει το επαναστατικό ArcFace Loss, ειδικά σχεδιασμένο για την βελτίωση της αξιοπιστίας στην αναγνώριση προσώπων. Το σύστημα, μέσω αυτού, μπορεί να αναγνωρίσει τον ιδιοκτήτη. Στις επόμενες υποενότητες θα αναλυθούν περαιτέρω οι λειτουργίες του κάθε τύπου αναγνώρισης.

5.4.1 Αναγνώριση εικόνας



Σχήμα 5.6: Διάγραμμα ροής Image Recognition.

Στο σχήμα 5.6, φαίνεται η ροή που πραγματοποιείται για την έξυπνη αναγνώριση εικόνων. Αρχικά ζητείται από το Backend σύστημα στο Native Backend η ενεργοποίηση της αναγνώρισης, λόγω πυροδότησης του αισθητήρα κίνησης. Στην περίπτωση αυτή, η Native Backend υπηρεσία ανοίγει το RTMP Stream, δηλαδή την κάμερα και το μικρόφωνο, ανάλογα τις ρυθμίσεις του χρήστη και στην περίπτωση που δεν είναι ήδη ανοιχτό, στην συνέχεια τροφοδοτεί καρέ από την ζωντανή ροή στο μοντέλο Coco SSD μέσω του Tensorflow. Για κάθε καρέ, εξάγονται προβλέψεις, ίσες με τα αντικείμενα-σώματα που πιστεύει ότι βλέπει το μοντέλο. Τα στοιχεία της κάθε προβλέψης αποτελούν:

1) Το όνομα της κλάσης, π.χ. Person, που βλέπει το μοντέλο.
2) Το ποσοστό σιγουριάς (Confidence Score) του μοντέλου, πραγματικός αριθμός από 0 έως 1.
3) Το bbox (bounding box). Περιέχει συντεταγμένες γωνιών (σχετικές με την ανάλυση της εικόνας), σχηματίζοντας ένα ορθογώνιο, το οποίο αποτελεί το σημείο παρουσίας της πρόβλεψης στην εικόνα.

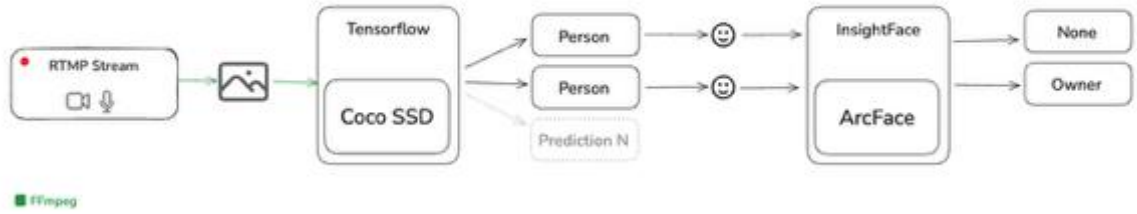
Τα πρώτα δύο πεδία, είναι αρκετά για την αξιολόγηση της πιθανότητας εισβολής στον χώρο του συστήματος. Συνεπώς, αυτά τα δεδομένα στέλνονται στο Backend, το οποίο με την σειρά του, μαζεύει πολλές προβλέψεις και τις αξιολογεί μέσω ορισμών κατωφλιού αριθμού προβλέψεων για κάθε κλάση και κατωφλιού μέσου όρου ποσοστού εμπιστοσύνης. Η διαδικασία αυτή, είναι απαραίτητη για την αποφυγή έναρξη συναγερμού, στην περίπτωση που το μοντέλο κάνει μία λάθος πρόβλεψη. Έτσι ανάλογα με τους αριθμούς που θα οριστούν, γίνεται είτε πιο εύκολη η έναρξη συναγερμού, με μεγαλύτερη πιθανότητα σφάλματος, είτε δυσκολότερη η έναρξη συναγερμού, με μικρότερη πιθανότητα σφάλματος. Για την ισορροπία, αυτών των δύο περιπτώσεων, χρίζεται αναγκαίος ο πειραματισμός των τιμών.

Το 3ο πεδίο, bbox, χρησιμοποιείται για λόγους εμφάνισης γραφικών στην οθόνη, με σκοπό να δείξουν στον χρήστη αυτό που βλέπει το μοντέλο. Στην συγκεκριμένη περίπτωση, χρησιμοποιούνται μόνο στην περίπτωση πρόβλεψης ανθρώπου από το CocoSSD, για την χρήση του στην αναγνώριση προσώπου, πιο αναλυτικά θα αναφερθεί στην επόμενη ενότητα.

Επιπλέον, επιρροή στην ικανότητα πρόβλεψης, αποτελεί η επιλογή των FPS, δηλαδή τον αριθμό των καρέ που θα τροφοδοτούνται στο μοντέλο το δευτερόλεπτο. Προφανώς, όσο μεγαλύτερος ο αριθμός, τόσο καλύτερα το μοντέλο μπορεί να εξετάσει μία ζωντανή ροή. Η επιλογή της αύξησης των FPS συνδέεται άμεσα με την αύξηση των απαιτούμενων υπολογιστικών πόρων και συνεπώς την μείωση της απόδοσης. Γι αυτό κρίνεται απαραίτητη η σωστή επιλογή του ρυθμού αυτού, που να εξισορροπεί τις δύο ανάγκες.

Συνοψίζοντας, η αναγνώριση εικόνας, αποτελεί ένα από τα βασικά χαρακτηριστικά του συστήματος. Συνεπώς, είναι κρίσιμη η σωστή επιλογή των παραμέτρων που αφορούν την αξιολόγηση εισβολής και τον ρυθμό καρέ τροφοδότησης.

5.4.2 Αναγνώριση προσώπου



Σχήμα 5.7: Διάγραμμα ροής Face Recognition.

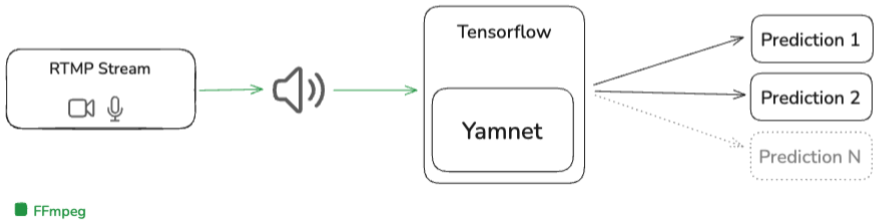
Με βάση την ροή που αναλύθηκε στην προηγούμενη υποενότητα, στην περίπτωση που το μοντέλο προβλέψει την παρουσία ανθρώπου, ακολουθεί η ροή του Face Recognition. Πιο συγκεκριμένα, στην περίπτωση αυτή, το σύστημα περικόπτει την εικόνα μέσω των συντεταγμένων που λαμβάνει από το πεδίο bbox, το οποίο αποδεικνύει την θέση του ανθρώπου στην εικόνα. Στην συνέχεια, παρέχεται στο μοντέλο ArcFace το οποίο ταυτίζει την κάθε εικόνα με τα αποθηκευμένα πρόσωπα.

Τα αποθηκευμένα πρόσωπα, είναι σε μορφή Face Embeddings, δηλαδή σε αρχεία, όπου το κάθε αρχείο περιέχει τα χαρακτηριστικά του προσώπου ενός ανθρώπου. Η διαδικασία αυτή έχει ως εξής:

1) Ο χρήστης, μέσω της ιστοσελίδας του συστήματος, καταγράφει ένα σύντομο βίντεο του προσώπου του.
2) Το ανεβάζει στο σύστημα δίνοντας του και ένα αναγνωριστικό όνομα.
3) Το σύστημα λαμβάνει το video, το χωρίζει σε εικόνες και μέσω του InsightFace δημιουργεί για τον άνθρωπο αυτό το αντίστοιχο αρχείο. Όπου περιέχονται τα μοναδικά και αναγνωριστικά χαρακτηριστικά του προσώπου.

Όπως προαναφέρθηκε, γίνεται ταύτιση της εικόνας προσώπου με το κάθε αρχείο. Με την κάθε ταύτιση, εξάγεται το ποσοστό ταύτισης. Το μεγαλύτερο ποσοστό ταύτισης αν ξεπερνά το όριο που ορίζεται από το κατώφλι, σημαίνει πως το σύστημα έχει αναγνωρισει το συγκεκριμένο πρόσωπο, που είναι συνήθως ο ιδιοκτήτης.

5.4.3 Αναγνώριση ήχου



Σχήμα 5.8: Διάγραμμα ροής Audio Recognition.

Στο σχήμα 5.8, φαίνεται η ροή της αναγνώρισης ήχου. Η λογική είναι παρόμοια με αυτή της αναγνώρισης εικόνας. Πιο αναλυτικά, η αναγνώριση ήχου ξεκινάει μαζί με την αναγνώριση εικόνας, εφόσον έχει ρυθμιστεί έτσι από τον διαχειριστή-χρήστη του συστήματος. Μέσω του εργαλείου FFmpeg, τροφοδοτείται στο μοντέλο Yamnet ο ήχος με βάση τις απαραίτητες ρυθμίσεις ήχου που απαιτεί το μοντέλο αυτο για την ορθή πρόβλεψη, όπως το sample rate και σε μορφή wav format. Το αποτέλεσμα της πρόβλεψεις περιέχει:

1) Το ποσοστό εμπιστοσύνης για κάθε κλάση, από 0 έως 1. Οι κλασεις που περνούν το κατώφλι, είναι οι ήχοι που ενδεχομένως ακούγονται.
2) Το spectrogram (φασματογράφημα) το οποίο μπορεί να χρησιμοποιηθεί για την οπτικοποίηση του φάσματος των συχνοτήτων του ήχου.

Την πρώτη έξοδο χρησιμοποιεί το σύστημα για την αξιολόγηση της πιθανότητας εισβολής στον χώρο. Με αντίστοιχη λογική όπως στην περίπτωση στην αναγνώρισης της εικόνας, συγκεντρώνει τον μέσο όρο πιθανότητας και αριθμό κάθε κλάσης, αν ξεπερνουν το ορισμένο κατώφλι, τότε ειδοποιεί τον χρήστη για τον αντίστοιχο ήχο.

Συνοψίζοντας, η αναγνώριση ήχου, αποτελεί προσθετική υπηρεσία του συστήματος, έτσι είναι δυνατή η απενεργοποίηση της από τον χρήστη αλλά και μένει ανενεργή στην περίπτωση όπου δεν υπάρχει σύνδεση μικροφώνου, συμβάλλοντας στην εξοικονόμηση ενέργειας και υπολογιστικών πόρων.

5.5 Αρχιτεκτονική ασφάλειας

Η ασφάλεια στο σύστημα, αποτελείται αποτελείται από την γενική μέσω του Nginx και από την ειδική που υπάρχει στο κάθε σύστημα. Αρχικά ο Nginx χρησιμοποιεί πρωτόκολλο ασφαλείας SSL/TLS και λειτουργεί ως Reverse Proxy, πιο αναλυτικά:

Για την χρήση SSL/TLS ασφάλειας, είναι αναγκαία η χρήση ψηφιακού πιστοποιητικού, το οποίο παρέχεται από υπηρεσίες CA (Central Authority), πιστοποιημένες υπηρεσίες που επαληθεύουν την ταυτότητα των διακομιστών. Στην συγκεκριμένη περίπτωση, δημιουργείται τοπικό CA, καθώς πρόκειται για περιβάλλον εντός δικτύου, οπότε υποστηρίζεται το ψηφιακό πιστοποιητικό του Nginx απο τις συσκευές που είναι συνδεδεμένες στο ίδιο δίκτυο. Τέλος το πρωτόκολλο SSL/TLS, συνεπάγεται με την χρήση κρυπτογραφίας των πακέτων που μετακινούνται, τον περιορισμό κακόβουλων επιθέσεων και την ομαλότερη πρόσβαση των διακομιστών.

Η χρήση SSL/TLS αποσκοπεί επίσης στην κρυπτογράφηση των των TS segments, που δημιουργεί ο Nginx μέσω της ζωντανής ροής RTMP, έτσι το HLS stream είναι κρυπτογραφημένο μέσω αλγορίθμου

AES-128, όπου μόνο ο πελάτης γνωρίζει το κλειδί αποκρυπτογράφησης, επιπλέον υπάρχει αλλαγή των κλειδιών ανά τακτό χρονικό διάστημα για την αποφυγή εύρεσης των κλειδιών από κακόβουλη πράξη.

Ο Nginx, λειτουργεί ως Reverse Proxy, για το Backend. Έτσι η ιστοσελίδα που σερβίρεται στον χρήστη δεν χρειάζεται να γνωρίζει τις διευθύνσεις των υπηρεσιών, παρά μόνο το path. Ο Nginx, διαχειρίζεται την λήψη και αποστολή αιτημάτων στις εσωτερικές υπηρεσίες του Backend. Αυτό έχει ως αποτέλεσμα την αύξηση της ιδιωτικότητας και την αποφυγή εγκατάστασης SSL/TLS πιστοποιητικού στα υπόλοιπα συστήματα, καθώς όλη η κίνηση περνάει μέσω του διακομιστή.

Επιπλέον, η υπηρεσία Native Backend, καθώς προορίζεται μόνο για την ανταλλαγή μηνυμάτων με το κύριο Backend, οπότε υπάρχει ανταλλαγή ενός token μεταξύ των δύο για επαλήθευση προέλευσης. Το token, το γνωρίζουν μόνο αυτές οι υπηρεσίες, και αποτελεί μία μεγάλη μοναδική σειρά από χαρακτήρες και σύμβολα. Έτσι, σε κάθε αίτημα, ελέγχεται αν υπάρχει αυτό το token και πραγματοποιείται η επαλήθευση του, στην περίπτωση λάθους, το μήνυμα απορρίπτεται. Η ίδια λογική, υλοποιείται στην αποστολή μηνυμάτων αισθητήρα του μικροελεγκτή ESP32 με το σύστημα.

Ο τελευταίος τύπος ασφάλειας, προϋποθέτει την σύνδεση του χρήστη. Στο σύστημα υλοποιείται η σύνδεση μέσω Firebase και συγκεκριμένα μέσω λογαριασμού Google. Με την επιτυχή σύνδεση του χρήστη, μέσω της ιστοσελίδας, δημιουργείται ένα Id Token, το οποίο στέλνεται στις επικεφαλίδες κάθε HTTP αιτήματος μεταξύ του πελάτη-χρήστη και του διακομιστή. Το backend, λαμβάνει το token, το επαληθεύει μέσω Firebase και συνεχίζει με την διεκπεραίωση του αιτήματος.

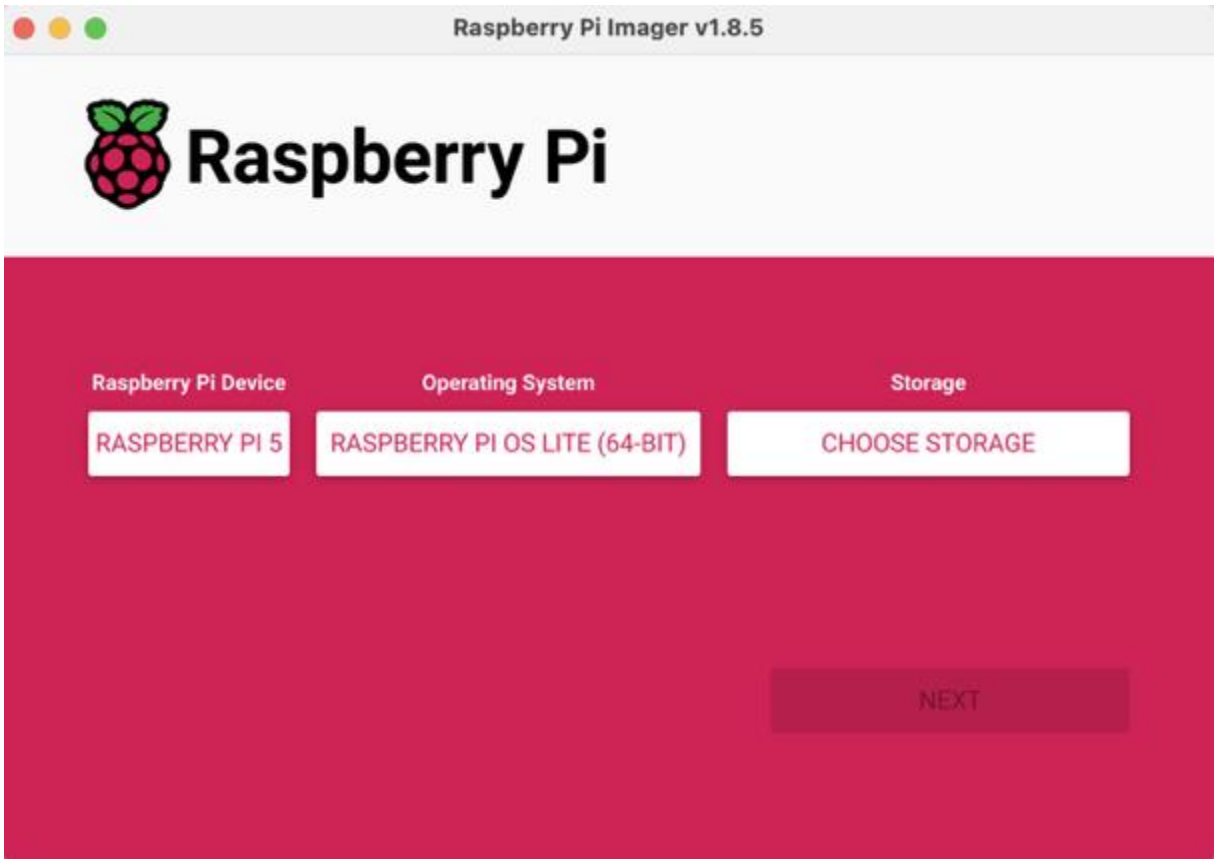
Συνοψίζοντας, το σύστημα εκτελεί αρκετές ενέργειες για την εξασφάλιση της ασφάλειας, της ιδιωτικότητας, της ομαλότητας και της εμπιστοσύνης, χωρίς την επιβολή εμφανών καθυστερήσεων στην απόκριση των αιτημάτων.

Κεφάλαιο 6ο: Υλοποίηση Συστήματος

6.1 Εισαγωγή

Το συγκεκριμένο κεφάλαιο, αποσκοπεί στην υλοποίηση ολόκληρου του συστήματος. Αρχικά, θα ξεκινήσει η εγκατάσταση και το στήσιμο όλων των απαραίτητων στο Raspberry Pi και στην συνέχεια η ανάπτυξη και συνδεσμολογία του ESP32 κλείνοντας έτσι την πλευρά του hardware. Στην συνέχεια θα εξηγηθεί ο τρόπος υλοποίησης των Backend υπηρεσιών όπως αποστολή SMS και έξυπνη αναγνώριση. Επιπλέον, η ανάπτυξη και δομή του Web application μαζί με όλες τι οθόνες και λειτουργίες που περιέχει. Η τρόπος εφαρμογής της Mobile εφαρμογής μαζί με την διαχείριση του Bluetooth και τέλος θα αναφερθεί η υλοποίηση των μέτρων ασφαλείας.

6.2 Εφαρμογές Raspberry Pi



Σχήμα 6.1: Desktop application Raspberry Pi Imager.

Για την υλοποίηση του Raspberry Pi, χρησιμοποιήθηκε το 5ο μοντέλο της σειράς με μέγεθος μνήμης RAM τα 8GB, μαζί με SD Card των 64GB. Επιπλέον, μέσω της εφαρμογής Raspberry Pi Imager, όπως φαίνεται στην εικόνα 6.1, έγινε η εγκατάσταση του ειδικά σχεδιασμένου λογισμικού Raspberry Pi OS Lite, συμβατό πλήρως με τον μικροϋπολογιστή. Το λογισμικό αυτό δεν περιέχει γραφικό περιβάλλον, καθώς δεν χρειάζεται για το συγκεκριμένο σύστημα, γεγονός που βελτιώνει την αποδοτικότητα του μηχανήματος. Η εγκατάσταση λειτουργικού συστήματος μέσω αυτού του προγράμματος γίνεται μέσω τον ακόλουθων βημάτων:

1) Επιλογή μοντέλο συσκευής Raspberry Pi.
2) Επιλογή λειτουργικού συστήματος. Περιέχει μια πληθώρα λειτουργικών συστημάτων βασισμένα στο Linux, με γραφικά ή χωρίς.
3) Επιλογή συσκευής δίσκου που είναι φορτωμένη στον υπολογιστή. Στο συγκεκριμένο βήμα επιλέγεται η SD Card, στην οποία θα γραφτεί επιλεγμένο το λειτουργικό, διαγράφοντας τα ήδη υπάρχων δεδομένα.
4) Στην συνέχεια, υπάρχει επιλογή ρυθμίσεων, όπως στοιχεία σύνδεσης WiFi, όνομα συσκευής και χρήστη για απομακρυσμένη σύνδεση μέσω SSH.
5) Τέλος, εισάγεται η κάρτα SD, μέσα στο Raspberry Pi, πραγματοποιεί εγκατάσταση και μετά την ολοκλήρωση της διαδικασίας γίνεται το boot του συστήματος.

Μετά την ολοκλήρωση της εγκατάστασης λειτουργικού, απομένουν η σύνδεση των περιφερειακών εξαρτημάτων, η ρύθμιση του συστήματος και η υλοποίηση της αυτόματης διαχείρισης του συστήματος.

6.2.1 Βασικές ρυθμίσεις

Μετά την εγκατάσταση του λειτουργικού, είναι αναγκαία η σύνδεση στο Raspberry Pi για την εγκατάσταση όλων των απαραίτητων βιβλιοθηκών.

Η σύνδεση γίνεται μέσω SSH όπου απαραίτητες προϋποθέσεις είναι:

1) Η γνώση του ονόματος και κωδικού του χρήστη. Ρυθμίστηκαν κατά την εγκατάσταση του λειτουργικού, στην αρχή του κεφαλαίου.
2) Η γνώση της local IP, φαίνεται στις ρυθμίσεις του Modem δικτύου.
3) Η συσκευή που πραγματοποιείται η σύνδεση και το Raspberry Pi να βρίσκονται στο ίδιο δίκτυο.

Στην συνέχεια, ακολουθεί η εντολή σύνδεσης όπου θα γίνει και η εισαγωγή του κωδικού:

```
ssh tmazarakis@192.168.1.20
```

Μετά την επιτυχή σύνδεση, ακολουθεί η ενημέρωση του εργαλείου διαχείρισης βιβλιοθηκών APT (Advanced Package Tool):

```
sudo apt update && sudo apt upgrade -y
```

Για την υλοποίηση του συστήματος, απαιτούνται αρκετές βιβλιοθήκες, που θα αναφέρονται κατά την πορεία του κεφαλαίου και η εγκατάσταση θα γίνεται μέσω του APT.

Για την ολοκλήρωση, της βασικής ρύθμισης του μικροϋπολογιστή, αναμένει η ρύθμιση του hostname, για την χρήση μιας πιο εύκολης διεύθυνσης μέσω mDNS (Multicast DNS).

Τροποποίηση του αρχείου /etc/hosts

```
127.0.0.1 trpia
```

Μετά από reboot, το σύστημα είναι προσβάσιμο από το τοπικό δίκτυο μέσω του ονόματος trpia.local αντί της διεύθυνσης IPv4.

6.2.2 Συνδεσμολογία εξαρτημάτων

Σχήμα 6.2: Τελική εικόνα Raspberry Pi με όλα τα εξαρτήματα συνδεδεμένα.

Όπως έχει αναφερθεί στο κεφάλαιο της αρχιτεκτονικής, τα περιφερειακά του Raspberry Pi αποτελούν από την κάμερα, το μικρόφωνο, το GSM/GPRS HAT, την ψύκτρα, το heatsink και τέλος τοποθετείται σε ένα Raspberry Pi case, για την καλύτερη προστασία των εξαρτημάτων. Στην πορεία της ενότητας αυτής, θα αναλυθεί η συνδεσμολογία του κάθε εξαρτήματος.

Σχήμα 6.3: Σύνδεση Raspberry Pi κάμερας στο CSI Connector.

Η σύνδεση της κάμερας μπορεί να γίνει σε μία από τις δύο συνδέσεις CSI, που είναι ειδικά σχεδιασμένες για την γρήγορη μεταφορά δεδομένων μεταξύ των Raspberry Pi καμερών. Στην συγκεκριμένη υλοποίηση γίνεται σύνδεση στο CSI 0. Παρόλα αυτά η σύνδεση μπορεί να γίνει και μέσω USB θύρας, αλλά θα επηρεαστεί σημαντικά η ταχύτητα του Video Streaming, γι αυτό και αποφεύχθηκε.

Η χρήση της κάμερας γίνεται μέσω των βιβλιοθηκών που προσφέρει η libcamera και προϋποθέτει εγκατάσταση μέσω:

```
sudo apt install -y libcamera-dev libcamera-apps
```

Η επαλήθευση της ύπαρξης κάμερας μπορεί να γίνει μέσω της εντολής:

```
libcamera-hello
```

Στην επιτυχή σύνδεση της κάμερας, εμφανίζονται ως έξοδο τα στοιχεία της κάμερας όπως:

```
tmazarakis@tRPIa:~ $ libcamera-hello
[2:45:40.234454775] [2809]  INFO Camera camera_manager.cpp:326 libcamera v0.5.0+59-d83ff0a4
[2:45:40.241739259] [2813]  INFO RPI pisp.cpp:720 libpisp version v1.2.1 981977ff21f3 29-04-2025 (14:13:50)
[2:45:40.242670847] [2813]  WARN CameraSensorProperties camera_sensor_properties.cpp:473 No static properties available for '
imx708_noir'
[2:45:40.242680310] [2813]  WARN CameraSensorProperties camera_sensor_properties.cpp:475 Please consider updating the camera
sensor properties database
[2:45:40.338031928] [2813]  WARN CameraSensor camera_sensor_legacy.cpp:501 'imx708_noir': No sensor delays found in static pr
operties. Assuming unverified defaults.
[2:45:40.338476704] [2813]  INFO RPI pisp.cpp:1179 Registered camera /base/axi/pcie@1000120000/rp1/i2c@88000/imx708@1a to CFE
 device /dev/media2 and ISP device /dev/media1 using PiSP variant BCM2712_C0
Preview window unavailable
Mode selection for 2304:1296:12:P
    SRGGB10_CSI2P,1536x864/0 - Score: 3400
    SRGGB10_CSI2P,2304x1296/0 - Score: 1000
    SRGGB10_CSI2P,4608x2592/0 - Score: 1900
Stream configuration adjusted
[2:45:40.343109126] [2809]  INFO Camera camera.cpp:1205 configuring streams: (0) 2304x1296-YUV420 (1) 2304x1296-BGGR_PISP_COM
P1
[2:45:40.343202144] [2813]  INFO RPI pisp.cpp:1483 Sensor: /base/axi/pcie@1000120000/rp1/i2c@88000/imx708@1a - Selected senso
r format: 2304x1296-SBGGR10_1X10 - Selected CFE format: 2304x1296-PC1B
```

Σχήμα 6.4: Στοιχεία Raspberry Pi κάμερας, μέσω libcamera.

Στην συνέχεια για το Live Streaming χρησιμοποιείται η εντολή libcamera-vid, για παράδειγμα η αποστολή των δεδομένων κάμερας σε ένα RTMP stream:

```
libcamera-vid -t 0 --inline --width 854 --height 480 --framerate 30 --
bitrate 2000000 --codec h264 --profile high --libav-format flv --awb auto
-o rtmp://0.0.0.0/live/stream
```

Περισσότερα για το Live Streaming αναφέρονται στο κεφάλαιο της αντίστοιχης υλοποίησης.

Σχήμα 6.5: Σύνδεση μικροφώνου μέσω USB 3.0.

Το Raspberry Pi διαθέτει 4 θύρες USB, 2 USB 2.0 και 2 USB 3.0. Το σύστημα έχει υλοποιηθει να υποστηρίζει οποιοδήποτε μικρόφωνο μπορεί να αναγνωριστεί από την κάρτα ήχου. Ο ιδιοκτήτης έχει την επιλογή χρήση οποιουδήποτε μικροφώνου ή ακόμα και την παράλειψη του. Για την ρύθμιση του USB μικροφώνου απαιτείται η εγκατάσταση των ακόλουθων βιβλιοθηκών:

```
sudo apt install -y alsa-utils uhubctl
```

Στην συνέχεια, ο έλεγχος του μικροφώνου γίνεται μέσω της εντολής:

```
arecord -l
```

```
tmazarakis@tRPIa:~ $ arecord -l
**** List of CAPTURE Hardware Devices ****
card 2: Device [USB PnP Sound Device], device 0: USB Audio [USB Audio]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

Σχήμα 6.6: Έξοδος συνδεδεμένων συσκευών με την κάρτα ήχου.

Ως έξοδος της εντολής, στην σύνδεση μικροφώνου, εμφανίζεται το όνομα της συσκευής καθώς και αριθμό κάρτας ήχου και αριθμό συσκευής, οι οποίοι αριθμοί χρησιμοποιούνται από το FFmpeg για την ενσωμάτωση της ροής ήχου στην ροή βίντεο.

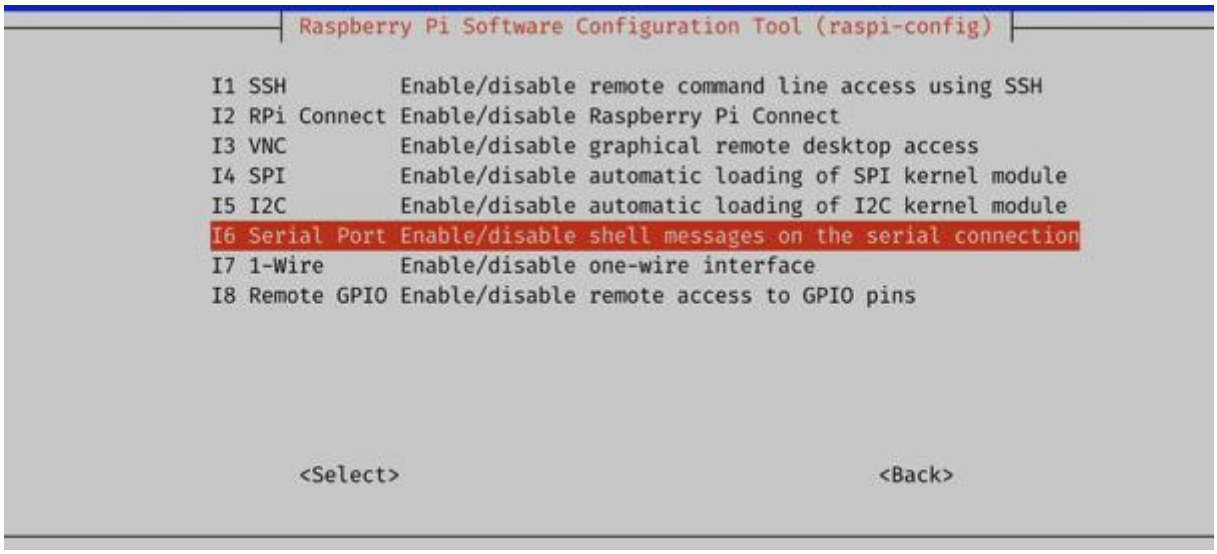
Σχήμα 6.7: Τοποθέτηση GSM/GPRS HAT στα GPIO pins.

Τελευταίο εξάρτημα, όπως φαίνεται στην εικόνα 6.7, αποτελεί το GSM/GPRS Modem. Η σύνδεση γίνεται μέσω των 40 GPIO pins του Raspberry Pi και λειτουργεί ως HAT (Hardware Attached on Top).

Το λειτουργικό σύστημα, μετά την τοποθέτηση του Modem, δημιουργεί ένα Serial Port μέσω ενός αρχείου συσκευής στο path /dev/ttyAMA0. Έτσι είναι δυνατή η επικοινωνία Raspberry Pi με GMS/GPRS HAT γράφοντας και διαβάζοντας από το συγκεκριμένο path, πρώτα όμως χρειάζονται ρυθμίσεις για την ορθή λειτουργία του Serial Port.

Αρχικά, αναγκαία η ενεργοποίηση του Serial Port, μέσω των ρυθμίσεων του λειτουργικού:

```
sudo raspi-config
```



Σχήμα 6.8: Ενεργοποίηση διεπαφής Serial Port.

Μετά την ενεργοποίηση της διεπαφής, όπως φαίνεται στο σχήμα 6.8, ακολουθεί η απενεργοποίηση του πρωτοκόλλου Bluetooth για την αποφυγή παρεμβολής, καθώς και τα δύο χρησιμοποιούν UART:

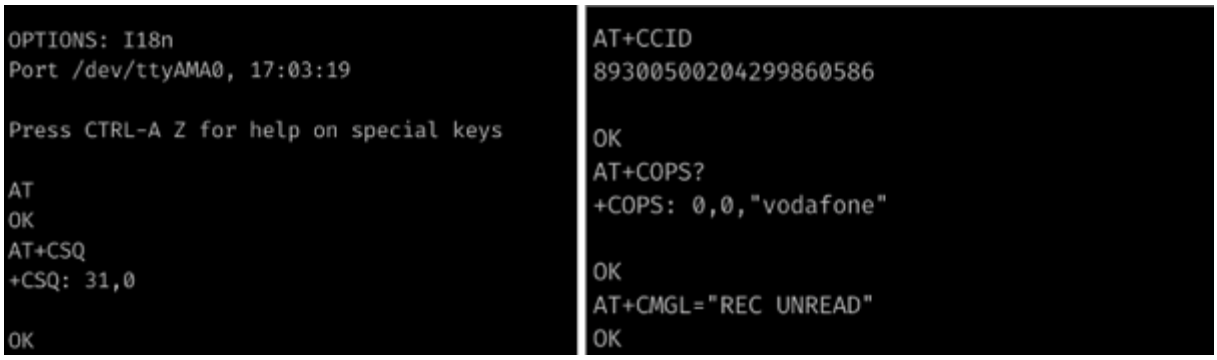
```
sudo systemctl stop hciuart
sudo systemctl disable hciuart
sudo systemctl stop bluetooth
sudo systemctl disable bluetooth
```

Ο έλεγχος για την σωστή λειτουργία του HAT γίνεται μέσω της εγκατάστασης της βιβλιοθήκης minicom για την αποστολή και λήψη απόκρισης μηνυμάτων AT μέσω terminal, τα οποία χρησιμοποιούν τα Modem για την εκτέλεση ενεργειών:

```
sudo apt install minicom
```

Είσοδος στο περιβάλλον minicom με την συσκευή του GSM/GPRS Modem /dev/ttyAMA0 και ρύθμιση του baud rate στα 115200, την μεγαλύτερη ταχύτητα που υποστηρίζει η συγκεκριμένη συσκευή.

```
minicom -D /dev/ttyAMA0 -b 115200
```



Σχήμα 6.9: Επικοινωνία με GSM/GPRS Module μέσω minicom.

Στο σχήμα 6.9, παρατηρείται επικοινωνία με το Modem μέσω AT εντολών. Στην αριστερή εικόνα, ελέγχεται η κατάσταση σύνδεσης και στην συνέχεια η ποιότητα σήματος, μέσω των εντολών AT και

AT+CSQ. Η απάντηση 31,0 σηματοδοτεί την άριστη ποιότητα σήματος. Στην δεξιά εικόνα, μέσω της εντολής AT+CCID, ζητείται το ID της SIM card και στην συνέχεια μέσω του AT+COPS? επιστρέφονται πληροφορίες σχετικά με τον πάροχο. Συνοψίζοντας, με την χρήση των εντολών αυτή είναι δυνατή η αποστολή και λήψη SMS, η συγκεκριμένη υλοποίηση θα αναλυθεί στο αντίστοιχο κεφάλαιο στην συνέχεια.

6.2.3 Αυτοματοποίηση εργασιών

Το σύστημα αποτελείται από πολλούς server που πρέπει να συγχρονιστούν για την ορθή λειτουργία τους, καθώς και από διάφορες υπηρεσίες που αφορούν την σωστή ρύθμιση του WiFi του ίδιου του υπολογιστή αλλά και των ESP32. Επιπλέον, δεν υφίσταται καλή πρακτική, στην περίπτωση επανεκκίνηση του συστήματος, η ανάγκη χειροκίνητης έναρξης όλων των λειτουργικών. Έτσι, έχουν υλοποιηθεί αυτόματες εργασίες, μικρά προγράμματα, τα οποία εκτελούνται από το ίδιο το λειτουργικό κατά την έναρξη του συστήματος.

Μέσω του Systemd (System Daemon), μπορεί να γίνει ρύθμιση, διαχείριση και δημιουργία αυτόματων διεργασιών. Η δημιουργία και ενεργοποίηση της κάθε διεργασίας αποτελείται από τα ακόλουθα βήματα:

Δημιουργία αναγνωριστικού αρχείου του diploma-setup.service.

```
sudo touch /etc/systemd/system/diploma-setup.service
```

Η διεργασία diploma-setup.service, τρέχει μετά το boot του συστήματος, με σκοπό τον έλεγχο σύνδεσης σε δίκτυο, διαχείριση του Bluetooth και παροχή WiFi credentials στους ESP32 που λειτουργούν ως AP. Μέσω του Bluetooth πραγματοποιείται αποστολή των στοιχείων του δικτύου από τον χρήστη μέσω της Mobile εφαρμογής. Τέλος, μετά την σωστή ρύθμιση του δικτύου ξεκινάει τις υπηρεσίες του συστήματος παρακολούθησης χώρου.

Δήλωση της υπηρεσίας diploma-setup.service, να αρχίσει μετά την αρχικοποίηση του αντάπτορα δικτυου και Bluetooth. Επιπλέον ρύθμιση, να ξεκινήσει η διεργασία diploma.service, που περιλαμβάνει το σύστημα παρακολούθησης, μετά το επιτυχή κλείσιμο της, με κατάσταση 0.

```
[Unit]
After=network-online.target NetworkManager.service bluetooth.service
Wants=network-online.target
OnSuccess=diploma.service
```

Δήλωση του προγράμματος (Script) που θα εκτελεστεί από την αυτόματη υπηρεσία, το path που βρίσκεται καθώς και την επανεκτέλεση στην περίπτωση σφάλματος.

```
[Service]
Type=simple
ExecStart=/home/tmazarakis/diploma/bluetooth_service/run.sh
WorkingDirectory=/home/tmazarakis/diploma/bluetooth_service
Restart=on-failure
```

```
User=tmazarakis
```

Τώρα που ρυθμίστηκε το αρχείο, απομένει η ενεργοποίηση της υπηρεσίας μέσω του systemctl CLI εργαλείου.

```
sudo systemctl daemon-reload
sudo systemctl enable diploma-setup.service
sudo systemctl start diploma-setup.service
```

Για την σωστή λειτουργία της διεργασίας, χρειάζονται βιβλιοθήκες για την διαχείριση του Bluetooth αντάπτορα:

```
sudo apt install bluetooth bluez libbluetooth-dev
```

Το πρόγραμμα (Script), της diploma-setup.service, θα εκτελέσει τις ακόλουθες ενέργειες:

Ενεργοποίηση Bluetooth αντάπτορα, σε επίπεδο λογισμικού και υλικού.

```
sudo systemctl enable bluetooth
sudo systemctl start bluetooth
sudo rfkill unblock bluetooth
```

Ο παρακάτω κώδικας, αποτελεί απόσπασμα για τον έλεγχο την συνδεσιμότητας στο WiFi, στην περίπτωση που δεν είναι δυνατή η σύνδεση, εκτελούνται τα προγράμματα διαχείρισης του Bluetooth. Τα προγράμματα αυτά, RFCOMMWatcher και RFCOMMHandler, συζητούνται στο κεφάλαιο της υλοποίησης Bluetooth, και αποσκοπούν στην σύνδεση με την Mobile συσκευή καθώς και την λήψη και αποστολή μηνυμάτων.

```
while true; do
   if hasInternetWifi; then
       echo "Internet connection detected. Exiting Bluetooth setup mode."
       killProcess "$rfcommHandlerPid"
       killProcess "$rfcommWatchPid"
       break
   else
       sleep 1
       if [[ -z "$rfcommWatchPid" ]] || ! kill -0 "$rfcommWatchPid"
2>/dev/null; then
           echo "No internet connection detected. Running RFCOMM
watcher..."
           startRFCOMMWatcher
       else
           echo "RFCOMM watcher already running (PID $rfcommWatchPid)."
       fi
```

```
        sleep 1
        if [[ -z "$rfcommHandlerPid" ]] || ! kill -0 "$rfcommHandlerPid"
2>/dev/null; then
            echo "No internet connection detected. Running RFCOMM
handler..."
            runRFCOMMHandler
        else
            echo "RFCOMM handler already running (PID $rfcommHandlerPid)"
        fi
        sleep 5
    fi
done
```

Η διαδικασία ελέγχου του WiFi επαναλαμβάνεται μέχρι την αποστολή στοιχείων από τον χρήστη και την επιτυχή σύνδεση στο δίκτυο. Στην συνέχεια, εκτελείται η αποστολή των WiFi στοιχείων στους μικροελεγκτές ESP32 που λειτουργούν ως AP και βρίσκονται σε απόσταση εντοπισμού. Περισσότερα, για την λειτουργία αυτή, αναφέρονται στο κεφάλαιο της υλοποίησης ESP32.

Επιπλέον, μετά την ολοκλήρωση της διαδικασίας, απενεργοποιείται ο αντάπτορας Bluetooth σε επίπεδο λογισμικού και υλικού για την καλύτερη αποδοτικότητα της ενέργειας:

```
bluetoothctl power off
sudo rfkill block bluetooth
```

Τέλος, την ίδια λογική αποτελεί η υπηρεσία diploma.service η οποία εκτελεί και διαχειρίζεται τα απαραίτητα όλου του συστήματος παρακολούθησης. Η υλοποίηση της θα αναλυθεί στο κεφάλαιο του Backend.

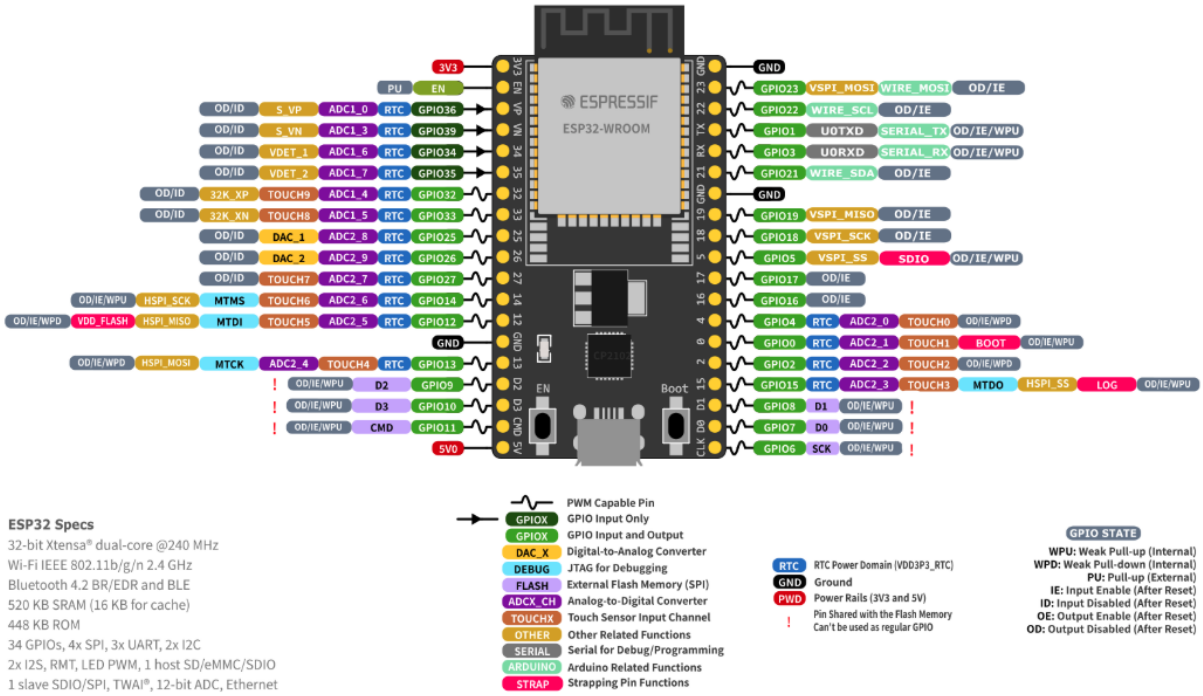
```
[Unit]
Description=Diploma services
After=diploma-setup.service
[Service]
Type=simple
ExecStart=/home/tmazarakis/diploma/run.sh
WorkingDirectory=/home/tmazarakis/diploma
Restart=on-failure
User=tmazarakis
```

Συνοψίζοντας, οι υπηρεσίες που αναφέρθηκαν, έχουν υλοποιηθεί με σκοπό την αυτοματοποίηση σύνδεσης και ρύθμισης του WiFi στο Raspberry Pi και στους ESP32, αλλά και την έναρξη των κύριων υπηρεσιών του συστήματος παρακολούθησης. Λόγω της λειτουργίας να εκτελούνται κάθε φορά μετά το boot, στην περίπτωση σφάλματος, ο χρήστης μπορεί να επανεκκινήσει τον μικροϋπολογιστή και να ξανα εκτελέσει την διαδικασία.

6.3 Υλοποίηση ESP32

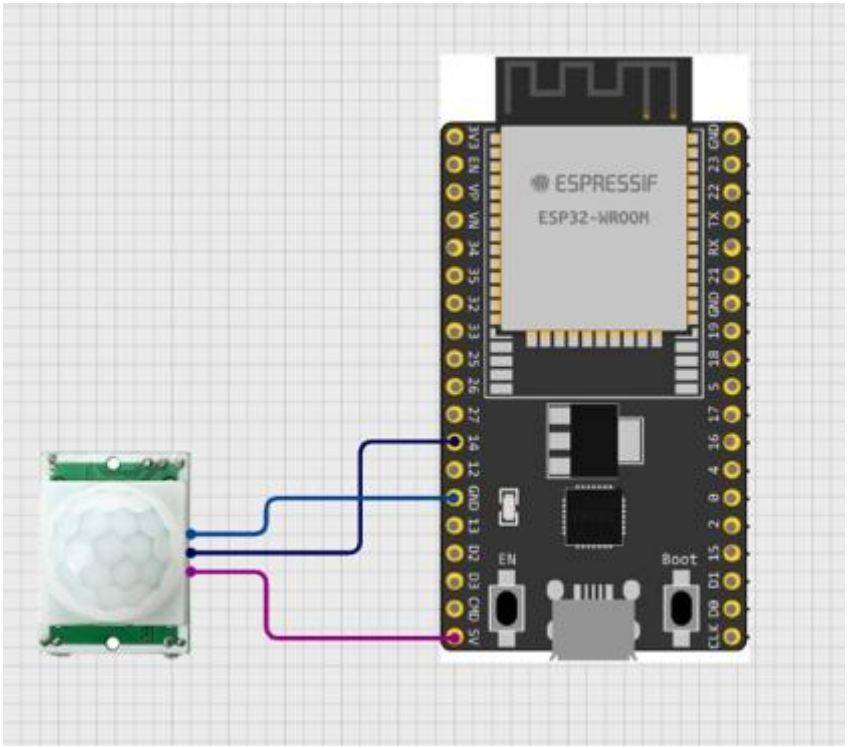
Το σύστημα υλοποιεί έναν μικροελεγκτή ESP32, όμως υποστηρίζονται περισσότεροι από ένας. Ο ESP32 λειτουργεί ως αποστολέας των ενδείξεων των αισθητήρων κίνησης και ήχου. Στις επόμενες υποενότητες θα αναφερθούν όλες οι λειτουργίες αναλυτικά.

6.3.1 Συνδεσμολογία αισθητήρων



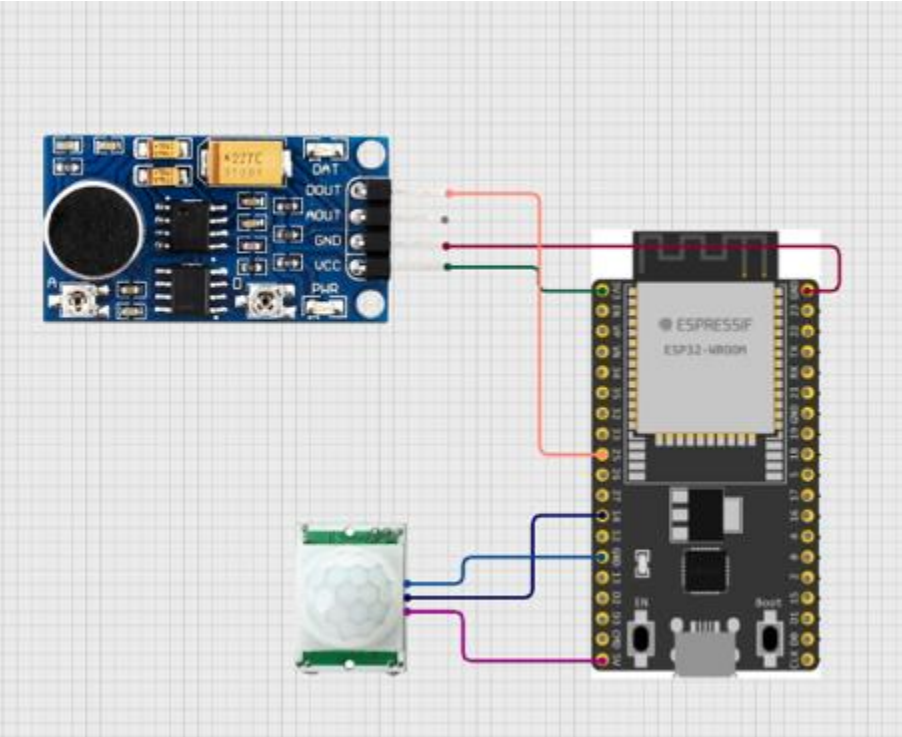
Σχήμα 6.10: GPIO pins του μικροελεγκτή ESP WROOM-32 [16].

Όπως παρατηρείται στο σχήμα 6.10, ο μικροελεγκτής αποτελείται από 34 GPIO pins, μόλις τα 2 θα χρησιμοποιηθούν για την υλοποίηση των δύο αισθητήρων. Συγκεκριμένα, πρέπει να χρησιμοποιηθούν 2 pins που φέρουν την ένδειξη RTC, ο λόγος αφορά την υλοποίηση της αποδοτικότητας της ενέργειας και θα αναφερθεί στην αντίστοιχη υποενότητα.



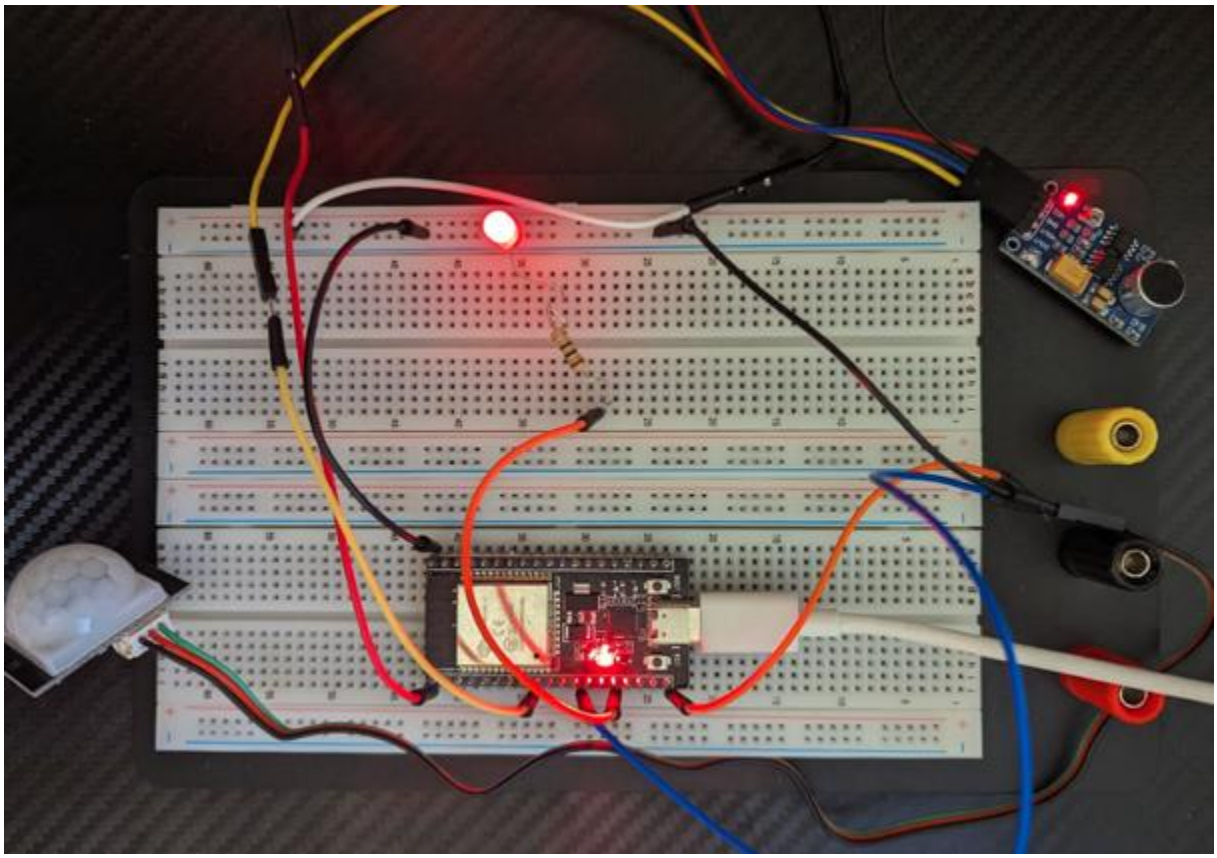
Σχήμα 6.11: Σύνδεση Motion Sensor με ESP32.

Αρχικά, για τον αισθητήρα κίνησης, θα χρησιμοποιηθεί το pin 14, το οποίο όπως φαίνεται στο σχήμα 6.10, υποστηρίζει RTC και ψηφιακή είσοδο την οποία θα λαμβάνει από τον αισθητήρα. Επιπλέον ως τάση εισόδου, υπάρχουν 2 επιλογές στα 3.3V και 5V, ο κατασκευαστής υποστηρίζει και τα δύο και επιλέγονται τα 5V για το μεγαλύτερο εύρος του αισθητήρα. Επιπλέον ο αισθητήρας, στο πίσω μέρος του, διαθέτει ποτενσιόμετρο για την ρύθμιση της καθυστέρησης. Έτσι λοιπόν, για την αμεσότερη απόκριση, σε διάστημα ms, έχει στραφεί τέρμα αριστερά.



Σχήμα 6.12: Σύνδεση Sound Sensor με ESP32.

Ο αισθητήρας ήχου, όπως φαίνεται στο σχήμα 6.12, έχει 4 pins, καθώς υποστηρίζει 2 εξόδους, αναλογική και ψηφιακή. Στην συγκεκριμένη υλοποίηση χρειάζεται μόνο η ψηφιακή έξοδος, η οποία στην αίσθηση ήχου βγάζει έξοδο LOW (0) αλλιώς έξοδο HIGH (1). Αν και θα ήταν προτιμότερη η χρήση της αναλογικής εξόδου, καθώς διαθέτει τιμές από 0 έως 4095, γεγονός που συνδέεται με την ένταση του ήχου, λόγω της υλοποίησης τεχνικών διαχείρισης της ενέργειας, δεν είναι εφικτό. Συνεπώς, η ψηφιακή έξοδος συνδέεται με το pin 25 του μικροελεγκτή, διότι υποστηρίζει RTC και ψηφιακή είσοδο.



Σχήμα 6.13: Τελική συνδεσμολογία ESP32 με τους αισθητήρες.

Στο παραπάνω σχήμα φαίνονται οι συνδέσεις στην πράξη που αναφέρθηκαν προηγουμένως, μαζί με την χρήση LED για το debugging κατά την ανάπτυξη, που σηματοδοτεί την ενεργοποίηση αισθητήρα του μικροελεγκτή. Στην συνέχεια θα αναφερθούν κομμάτια κώδικα της ανάπτυξης του λογισμικού σε περιβάλλον Arduino IDE.

Αρχική δήλωση των pin του που θα χρησιμοποιηθούν:

```
#define motionSensorPin GPIO_NUM_14
#define soundSensorDigitalPin GPIO_NUM_25
#define ledPin GPIO_NUM_13
```

Ένα πρόγραμμα Arduino-ESP32, αποτελείται από 2 κύριες μεθόδους, την setup και την loop. Η πρώτη εκτελείται στην αρχή της λειτουργίας του chip ενώ η δεύτερη επαναλαμβάνεται συνεχώς εφόσον είναι σε λειτουργία η συσκευή. Στην συγκεκριμένη υλοποίηση, χρησιμοποιείτε μόνο η setup, καθώς ο μικροελεγκτής μπαίνει σε κατάσταση ύπνου. Η loop θα χρησιμοποιηθεί μόνο στην περίπτωση που ο μικροελεγκτής είναι σε κατάσταση AP, δηλαδή αναμένει ρύθμιση WiFi.

Αρχικοποίηση pins και απενεργοποίηση Bluetooth στην έναρξη λειτουργίας:

```
void setup(){
 …
 btStop(); // Disable bluetooth adapter
 pinMode(ledPin, OUTPUT);
 pinMode(soundSensorDigitalPin, INPUT);
 pinMode(motionSensorPin, INPUT)
 digitalWrite(ledPin, LOW);
 …
}
```

Ο ESP32, κατά την λήψη σήματος από τους αισθητήρες, χρειάζεται να στείλει μήνυμα στο σύστημα μέσω MQTT με JSON μορφή. Αυτό προϋποθέτει την λήψη και εισαγωγή αντίστοιχων βιβλιοθηκών:

```
#include <PubSubClient.h>
#include <ArduinoJSON.h>
```

Στην συνέχεια ακολουθεί η ρύθμιση των στοιχείων για την σύνδεση στον MQTT broker του συστήματος, τον Mosquitto, και η δημιουργία MQTT client:

```
#define Token "NDMwYmFkZmItM2VlYy00NGQxLWFlMWUtNGUzYTFkOWJmNjIw"
const char* mqtt_server = "trpia.local";
const int mqtt_port = 1883;
const char* mosquittoUsername = "admin";
const char* mosquittoPassword = "diploma123";
const char* mqtt_topic_motion = "sensors/motion";
const char* mqtt_topic_sound = "sensors/sound";
const char* mqtt_topic_ping = "sensors/ping";
PubSubClient client(espClient);
```

Το token, επαληθεύεται από την Backend υπηρεσία ως μέτρο προστασίας ως μέσο αυθεντικοποίησης της προέλευσης του μηνύματος. Επιπλέον μέτρα προστασίας αποτελούν τα mosquitto username και password, των οποίων η ορθότητα επαληθεύεται από τον Mosquitto broker. Τέλος η αποστολή μηνυμάτων MQTT γίνεται προς 3 topics, sensors/motion για αισθητήρα κίνησης, sensors/sound για τον αισθητήρα ήχου και το sensors/ping για την αποστολή κατάστασης προς το σύστημα, ότι ο ESP32 είναι συνδεδεμένος.

Όπως αναφέρθηκε, στην περίπτωση σήματος από τους αισθητήρες, ο μικροελεγκτής στέλνει MQTT μήνυμα στον broker σε μορφή JSON η οποία αποτελεί:

1) sensorName, τύπος αισθητήρα μαζί με το μοναδικό όνομα του μοντέλου του ESP32.
2) status, όπως motion, sound και ping, για τον διαχωρισμό του μηνύματος.
3) token, το token για επαλήθευση του μηνύματος.

Η αποστολή γίνεται στο αντίστοιχο topic μαζί με το μήνυμα ως σειρά από χαρακτήρες, δηλαδή γίνεται μετατροπή απο JSON σε char:

```
…
char jsonBuffer[200];
serializeJSON(jsonData, jsonBuffer);
client.publish(topic, jsonBuffer);
…
```

Συνοψίζοντας, ο τρόπος λειτουργίας του μικροελεγκτή, όπως συνδέσεις WiFi, MQTT και λειτουργία ύπνου αποτελούν τα κύρια κομμάτια υλοποίησης για τον μικροελεγκτή και θα αναλυθούν στις 2 επόμενες υποενότητες.

6.3.2 Διαχείριση ενέργειας

Η διαχείριση ενέργειας, αποτελεί σημαντική λειτουργία, ειδικά σε περιπτώσεις όπου χρησιμοποιείται μπαταρία με περιορισμένη διάρκεια ζωής. Ο μικροελεγκτής ESP32 αποτελείται από πολλά κομμάτια, τα οποία μπορούν να ενεργοποιηθούν ανάλογα την περίσταση για την μείωση στην κατανάλωση ενέργειας. Επιπλέον παρέχονται διάφορες λειτουργίες ύπνου, που απλοποιούν την διαδικασία, διαχειρίζοντας την ενεργοποίηση και απενεργοποίηση αυτόματα. Συγκεκριμένα, η αποδοτικότερη λειτουργία υπνου είναι το deep sleep. Στην λειτουργία αυτή, γίνεται απενεργοποίηση του WiFi και Bluetooth. Επιπλέον η CPU και η RAM και τα σχεδόν όλα τα περιφερειακά απενεργοποιούνται. Ενεργά, απομένουν ο RTC Controller, RTC μνήμη και ο ULP επεξεργαστής, ειδικός για χαμηλή ισχύ. Σκοπός των ενεργών κομματιών, είναι το ξύπνημα των υπόλοιπων εξαρτημάτων. Η υλοποίηση της διαχείρισης ενέργειας βασίζεται στα:

1) Timer, ρυθμίζεται ο μικροελεγκτής να μπαίνει σε deep sleep για κάποιο χρονικό διάστημα. Χρήσιμο για την ενημέρωση του Backend, στο ότι ο ESP32 είναι ακόμα συνδεδεμένος.
2) External Wakeup (EXT0), ρυθμίζεται ο μικροελεγκτής να ξυπνήσει μετά την λήψη σήματος σε ένα RTC pin. Χρήσιμο για το ξύπνημα μετά από την έξοδο LOW του αισθητήρα ήχου.
3) External Wakeup (EXT1). Αποτελεί παρόμοια λογική με το ξύπνημα EXT0, με την διαφορά ότι υποστηρίζονται πάνω από ένα pin και χρειάζεται μόνο ένα να πάρει την τιμή HIGH ή LOW. Χρησιμοποιείτε για το ξύπνημα μετά από έξοδο HIGH του αισθητήρα κίνησης.

Όπως αναφέρθηκε προηγουμένως, στην υλοποίηση deep sleep η μέθοδος loop δεν χρησιμοποιείται, έτσι το πρόγραμμα πρέπει να αρκεστεί μόνο στην setup να ικανοποιήσει τις ανάγκες.

Αρχικά για την δήλωση του deep sleep χρειάζεται να οριστούν οι τρόποι αφύπνισης, δηλαδή μέσω Timer, EXT0 και EXT1.

```
#define WakePinMask (1ULL << motionSensorPin)
void setupPowerEfficiency(){
 esp_sleep_enable_ext1_wakeup(WakePinMask, ESP_EXT1_WAKEUP_ANY_HIGH);
 esp_sleep_enable_ext0_wakeup(soundSensorDigitalPin, LOW);
 esp_sleep_enable_timer_wakeup(60 * 1000000ULL);
 …
```

Μέσω των παραπάνω εντολών, δηλώνεται το ξύπνημα στην κατάσταση HIGH του αισθητήρα κίνησης, στην κατάσταση LOW της ψηφιακής εξόδου του αισθητήρα ήχου και το ξύπνημα μετά από 60 δευτερόλεπτα ύπνου.

Απομένει η κλήση της λειτουργίας deep sleep:

```
 …
 esp_deep_sleep_start();
}
```

Ο μικροελεγκτής, μετά την κλήση της συνάρτησης, μπαίνει σε λειτουργία deep sleep και αναμένει σήμα για να συνεχίσει την κανονική του λειτουργία μέσω του RTC. Κατά την αφύπνιση, μπορεί να παρθεί ο λόγος wakeup έτσι ώστε να γίνει σωστή αποστολή στο Backend. Επειδή, στην αφύπνιση από EXT1 υποστηρίζεται η χρήση πολλών pins, επιπλέον λάβετε το pin που προκάλεσε το ξύπνημα:

```
void setup(){
 …
 uint64_t wakeup_pin_mask = esp_sleep_get_ext1_wakeup_status();
 esp_sleep_wakeup_cause_t cause = esp_sleep_get_wakeup_cause();
 …
```

Στην συνέχεια ελέγχεται ο λόγος (cause), ακολουθεί συνάρτηση onSensorSensed, η οποία αποσκοπεί στην σύνδεση στο WiFi, MQTT broker και αποστολή μηνύματος JSON ανάλογα τον λόγο αφύπνισης.

```
 …
 switch(cause){
  case ESP_SLEEP_WAKEUP_EXT0:
    Serial.println("Woke up from SOUND sensor!");
    onSensorSensed("sound", mqtt_topic_sound, modelName);
    break;
  case ESP_SLEEP_WAKEUP_EXT1:
    while(digitalRead(motionSensorPin) == HIGH){
      delay(25);
    }
    if (wakeup_pin_mask & (1ULL << motionSensorPin)) {
      Serial.println("Woke up from MOTION sensor!");
    }else{
      Serial.println("unknown wakeup in EXT1");
      break;
    }
    onSensorSensed("motion", mqtt_topic_motion, modelName);
    break;
  case ESP_SLEEP_WAKEUP_TIMER:
    Serial.println("Wakeup caused by timer");
    onSensorSensed("motion,sensor", mqtt_topic_ping, modelName);
    break;
   default:
```

```
      Serial.print("Wakeup caused by unknown reason: ");
      break;
    }
 setupPowerEfficiency();
}
```

Ο τρόπος σύνδεσης στον MQTT broker, αποτελείται από την προσπάθεια σύνδεσης έως 10 φορές. Αυτό γίνεται γιατί αποτελεί συχνό φαινόμενο η αδυναμία σύνδεσης, οπότε μετά την 10η προσπάθεια μη επιτυχημένης σύνδεσης, σηματοδοτεί την ύπαρξη προβλήματος του Mosquitto broker, οπότε ο ESP32 εγκαταλείπει το μήνυμα και μπαίνει ξανά σε λειτουργία ύπνου.

```
void connectToMQTT() {
 int totalAttempts = 0;
 while (!client.connected() && totalAttempts < 10) {
   totalAttempts++;
   Serial.print("Connecting to MQTT broker...");
   if (client.connect("ESP32Client", mosquittoUsername,
mosquittoPassword)) {
     Serial.println("connected!");
   } else {
     Serial.print(client.state());
     delay(2000);
   }
 }
}
```

Πριν την σύνδεση στον MQTT broker, είναι αναγκαία η σύνδεση στο ίδιο τοπικό δίκτυο, με το Raspberry Pi, μέσω WiFi. Καθώς, ο MQTT δεν γνωρίζει τα στοιχεία σύνδεσης και σε ποιο δίκτυο πρέπει να συνδεθεί, έχει υλοποιηθεί τρόπος παροχής των στοιχείων αυτών και αναλύεται στην επόμενη υποενότητα.

Τέλος αξίζει να σημειωθεί πως, με την χρήση deep sleep, η κατανάλωση ρεύματος πέφτει στην τάξη των μερικών μA από τα 160-260mA στην κανονική του λειτουργία, δηλαδή η κατανάλωση μειώνεται έως και 20.000 φορές. Η τεχνική αυτή όμως, παρουσιάζει και ένα σημαντικό αρνητικό ειδικά στις υλοποιήσεις που απαιτούν αρκετά χαμηλό χρόνο απόκρισης. Δηλαδή, ο μικροελεγκτής πρέπει να επαναφέρει τα εξαρτήματα τα του στην κανονική τους λειτουργία και επιπλέον να συνδεθεί στο WiFi και στον MQTT broker, γεγονός που προσθέτει καθυστέρηση έως και μερικών δευτερολέπτων στην χειρότερη περίπτωση. Στην συγκεκριμένη υλοποίηση, αν και η καθυστέρηση των μερικών ms θα αποτελούσε ιδανική, όμως δεν αποτελεί σημαντικότερος παράγοντας από την μείωση της ενέργειας.

6.3.3 Λειτουργίες του WiFi

Για την αποστολή μηνυμάτων, κατά την αίσθηση κίνησης και ήχου, προς το σύστημα είναι αναγκαία ύπαρξη των ESP32 και του Raspberry Pi στο ίδιο τοπικό δίκτυο. Για την δυναμικότητα αυτής της λειτουργίας, έχει αναπτυχθεί η αποστολή στοιχείων WiFi από το Raspberry Pi προς τους διαθέσιμους μικροελεγκτές. Δηλαδή, υλοποιούνται 2 λειτουργίες του WiFi, ο μικροελεγκτής ως STA (Station) και ως AP (Access Point).

Η λειτουργία STA (Station), αποτελεί την κανονική λειτουργία μιας συσκευής με αντάπτορα WiFi. Δηλαδή, συνδέεται σε ένα δίκτυο WiFi που βρίσκεται σε εμβέλεια μέσω του SSID, μοναδικό χαρακτηριστικό κάθε δικτύου WiFi, και μέσω του κωδικού πρόσβασης. Στην συνέχεια, η συσκευή μπορεί να επικοινωνήσει με τις υπόλοιπες συσκευές στο δίκτυο αλλά και να πάρει πρόσβαση στο διαδίκτυο αν αυτό υποστηρίζεται. Αυτή η λειτουργία, είναι ο βασικός τρόπος σύνδεσης του μικροελεγκτή αφότου έχει λάβει τα στοιχεία WiFi.

Στην δεύτερη λειτουργία, Access Point, ο μικροελεγκτής δημιουργεί το δικό του δίκτυο WiFi. Αναθέτει ένα SSID και κωδικό πρόσβασης και αναμένει την σύνδεση άλλων συσκευών. Αυτή η κατάσταση, ενεργοποιείται όταν ο ESP32 δεν γνωρίζει ή δεν μπορεί να συνδεθεί στο WiFi. Με σκοπό της αποστολή WiFi στοιχείων μέσω HTTP από το Raspberry Pi.

Στην συνέχεια θα αναφερθεί ο τρόπος υλοποίησης WiFi AP από τον μικροελεγκτή και αργότερα η τρόπος αποστολής των στοιχείων από το Raspberry Pi.

Αρχικά για την λειτουργία ως WiFi AP, υπάρχει η βιβλιοθήκη WiFi.h, που απλοποιεί αρκετά την διαδικασία και η βιβλιοθήκη WebServer.h για την αφηρημένη διαχείριση HTTP server. Επιπλέον, χρειάζεται η χρήση ενός μοναδικού SSID και password:

```
#include <WiFi.h>
#include <WebServer.h>
const String SoftApPassword =
"ZGIwNTJiMDgtYTE0Ny00ZTA2LWI5YjYtYzc2OTY2N2NlMWM0";
const String SoftApName = "ESP32-AP-" +
String((uint32_t)ESP.getEfuseMac(), HEX);
```

Συγκεκριμένα για την χρήση κωδικού έχει επιλεχθεί ένας τυχαίος UUID-V4 μορφής, κωδικοποιημένος μέσω base64. Το SSID δημιουργείται από το κάθε μικροελεγκτή με πρόθεμα ESP-32AP- και τη μοναδική διεύθυνση MAC της συσκευής με τα τελευταία 6 ψηφία σε δεκαεξαδική μορφή.

Απομένει η δημιουργία του δικτύου WiFi:

```
void openAsAccessPoint(){
 WiFi.mode(WIFI_AP);
 WiFi.softAP(SoftApName, SoftApPassword);
 //* Address
 Serial.println("AP Ip Address: ");
 Serial.println(WiFi.softAPIP());
 …
```

Η δημιουργία ενός HTTP server, ο οποίος θα λάβει τα WiFi credentials ως μήνυμα JSON μέσω POST HTTP αιτήματος στο path /wifi.

```
 …
 server.on("/wifi", HTTP_POST, onWiFiCredentialsReceived);
 server.begin();
 Serial.println("Server started!");
}
```

```
void loop() {
 server.handleClient();
 delay(10);
}
```

Επομένως, ο server ακούει για αιτήματα σε συχνότητα 10ms. Κατά την λήψη HTTP αιτήματος στο path /wifi, εκτελεί την συνάρτηση onWiFiCredentialsReceived, η οποία το διαχειρίζεται:

```
#include <Preferences.h>
void onWiFiCredentialsReceived() {
 String body = server.arg("plain");
 StaticJsonDocument<200> jsonDoc;
 DeserializationError error = deserializeJson(jsonDoc, body);
 if (error) {
   server.send(400, "text/html", "Invalid JSON");
   delay(100);
   return;
 }
 const char* ssid = jsonDoc["ssid"];
 const char* password = jsonDoc["password"];
 //* Save to preferences
 preferences.begin("wifi", false);
 preferences.putString("ssid", ssid);
```

```
 preferences.putString("password", password);
 preferences.end();
 …
```

Η συνάρτηση, αρχικά παίρνει το μήνυμα του αιτηματος που βρίσκεται στο body, στην συνέχεια εκτελεί μετατροπή από JSON αντικείμενο, με σκοπό να παρθούν ο κωδικός και το SSID. Τέλος τα αποθηκεύει με την χρήση των Preferences.

Τα Preferences, είναι μία βιβλιοθήκη που προσφέρει προσπέλαση ενός μέρους της μη πτητικής μνήμης (NVS) που υπάρχει στον μικροελεγκτή, ο αποθηκευτικός χώρος αυτός επιμένει κατά την διάρκεια Restart και λειτουργιών ύπνου, κάνοντας τον ιδανικό για την συγκεκριμένη ανάγκη αποθήκευσης των στοιχείων για αργότερα.

Μετά την αποθήκευση των στοιχείων στην μνήμη, απομένει η αποστολή κωδικού κατάστασης και η επανεκκίνηση του μικροελεγκτή.

```
 …
 server.send(200, "text/html", "Credentials saved. Rebooting...");
 server.client().stop();
 delay(1000);
 ESP.restart();
}
```

Επομένως, μόλις ενεργοποιηθεί ο ESP32, ελέγχει αν υπάρχουν στα στοιχεία WiFi, στην περίπτωση που υπάρχουν, ρυθμίζεται η λειτουργία ύπνου. Κατά την αφύπνιση, διαβάζει τα στοιχεία και συνδέεται στο WiFi ως Station πλέον.

```
void onSensorSensed(){
 …
 preferences.begin("wifi", false);
 String ssid = preferences.getString("ssid", "");
 String password = preferences.getString("password", "");
 preferences.end();
 WiFi.mode(WIFI_STA);
 WiFi.begin(ssid, password);
 …
```

Συνοψίζοντας, αυτές αποτελούν οι διαδικασίες που ακολουθεί ο μικροελεγκτής για την αποστολή μηνυμάτων αισθητήρα στο Backend. Μέσω της λειτουργίας δύο mode WiFi, είναι δυνατή η αυτόματη και δυναμική ρύθμιση του WiFi από το σύστημα του Raspberry Pi, καθιστώντας την υλοποίηση αρκετά ευέλικτη.

6.4 Υλοποίηση λογισμικού Backend

Η υλοποίηση των υπηρεσιών backend, έχει βασιστεί κυρίως στις τεχνολογίες Go και Nodejs. Στο κεφάλαιο αυτό, θα αναφερθούν οι κυριότερες λειτουργίες του συστήματος. Αρχικά θα γίνει εισαγωγή στις κεντρικές υπηρεσίες, στην συνέχεια θα περιγραφεί η υλοποίηση του Live Streaming και η ενσωμάτωση των μοντέλων έξυπνης αναγνώρισης. Ύστερα, θα αναλυθούν οι υλοποιήσεις επικοινωνίας SMS και Bluetooth, η διαχείριση και ανίχνευση εισβολής. Τέλος, θα γίνει λόγος για την βάση δεδομένων και για την χρήση Docker compose που λειτουργεί ως συνδετικός κρίκος του συστήματος.

6.4.1 Ανάπτυξη κεντρικών υπηρεσιών

Όπως έχει αναφερθεί στο κεφάλαιο της αρχιτεκτονικής, το σύστημα υλοποιεί δύο υπηρεσίες Backend. Η πρώτη, η Backend Native, είναι υπεύθυνη για την διαχείριση των λειτουργιών που περιέχουν την κάμερα και το μικρόφωνο. Δηλαδή το Live Streaming, πληροφορία συσκευών και διαχείριση των μοντέλων της έξυπνης αναγνώρισης. Είναι υλοποιημένη με χρήση TypeScript στο περιβάλλον Nodejs, που επιτρέπει την εκτέλεση JavaScript αρχείων εκτός Browser.

Αρχικά, αναμένει μήνυμα WebSocket από την κύρια υπηρεσία Backend του συστήματος. Το μήνυμα αυτό φέρει πληροφορία σχετικά με τον τύπο υπηρεσίας και την δράση που πρέπει να παρθεί, πιο συγκεκριμένα:

1) Service, η υπηρεσία για εκτέλεση. Η υπηρεσία αποτελεί ένα ή περισσότερα command που θα εκτελεστεί στο λειτουργικό σύστημα του Raspberry Pi.
2) Action, έναρξη ή παύση της υπηρεσίας.
3) Token, αναγνωριστικό token μορφής UUID-V4 με base64 κωδικοποίηση. Αυτό το token, το γνωρίζει μόνο η Backend υπηρεσία και χρησιμοποιείται για την επαλήθευση προέλευσης του μηνύματος.

```
export const handleServiceMessage = (msg: ServiceMessage) => {
  if (msg.action === 'stop') {
    Process.killProcess(msg.service);
    return;
  }
  if (msg.action === 'start') {
    switch (msg.service) {
      case 'rtmp_stream':
        Process.startProcess({
          service: msg.service,
          withAudio: msg.state === 'audio-on',
          stdout: d => {},
          stderr: d => {},
        });
        break;
      case 'image_recognition':
        Process.startProcess({
          service: msg.service,
          stdout: jpegImageChunkHandler,
        });
        break;
      case 'face_recognition':
        Process.startProcess({
          service: msg.service,
          stdout: faceRecognitionImageChunkHandler,
        });
        break;
      case 'audio_recognition':
        Process.startProcess({
          service: msg.service,
          stdout: wavAudioChunkHandler,
        });
        break;
```

Σχήμα 6.14: Απόσπασμα υλοποίησης process handler με Nodejs.

Μετά την λήψη μηνύματος για την έναρξη ή κλείσιμο κάποιας υπηρεσίας, ο process handler ξεκινάει την αντίστοιχη υπηρεσία, μια σειρά από εντολές, και διαχειρίζεται την έξοδο της όπου χρειάζεται. Οι υπηρεσίες που υπάρχουν είναι:

1) rtmp_stream. Διαχείριση της ζωντανή ροής προς τον Nginx server.
2) face_embeddings. Δημιουργία face embeddings του Face recognition για την αναγνώριση κάποιου επιπλέον προσώπου.
3) camera_status: Λήψη πληροφοριών της κάμερας, όπως κατάσταση σύνδεσης και μοντέλο.
4) audio_info: Λήψη πληροφοριών μικροφώνου, αριθμός συσκευής, αριθμός κάρτας ήχου και όνομα συσκευής.
5) image_recognition: Διαχείριση μοντέλου αναγνώρισης εικόνων.
6) face_recognition: Διαχείριση μοντέλου αναγνώρισης προσώπων.
7) audio_recognition: Διαχείριση μοντέλου αναγνώρισης ήχου.

Οι υπηρεσίας σχετικά με την λήψη πληροφοριών των συσκευών εισόδου, καλούνται αυτόματα από το Backend περιοδικά για την άμεση εμφάνιση της κατάστασης στον χρήστη. Απο την άλλη, η έναρξη και το κλείσιμο της ζωντανής ροής βίντεο, πραγματοποιείται είτε κατόπιν αιτήματος του διαχειριστή από την ιστοσελίδα, είτε από την ενεργοποίηση του αισθητήρα κίνησης είτε από αλλαγή της κατάστασης του συστήματος μέσω προγράμματος που ορίζεται αυτόματα από τον διαχειριστή μέσω της ιστοσελίδας. Αντίστοιχη λογική συνδέει την διαχείριση των μοντέλων αναγνωρισης, που ακολουθούν την κατάσταση της ζωντανής ροής, καθώς αντλούν τα δεδομένα από αυτήν. Τέλος, η υπηρεσία για την δημιουργία face embeddings γίνεται μέσω αιτήματος του χρήστη, μόλις δηλαδή ανεβάσει τις εικόνες

που απαιτούνται για την διαδικασία αυτή. Περισσότερη επεξήγηση της κάθε υπηρεσίας, θα αναφερθεί στα αντίστοιχα κεφάλαια που ακολουθούν.

Στην συνέχεια, ακολουθεί η υλοποίηση του κύριου Backend, βασισμένο στην γλώσσα Go με κύριες λειτουργίες να αποτελούν:

1) Restful API, διαχείριση HTTP αιτημάτων από frontend, για εκτέλεση ενεργειών και παροχή δεδομένων.
2) Background processes. Διαχείριση του προγράμματος και αυτόματη αλλαγή κατάστασης του συστήματος. Αποστολή και λήψη SMS μηνυμάτων και Push Notifications.
3) Διαχείριση των προβλέψεων για την έναρξη συναγερμού.
4) Live ενημέρωση της ιστοσελίδας σχετικά με τα γεγονότα του συστήματος.
5) Διαχείριση μηνυμάτων από τους αισθητήρες (ESP32).

Το σύστημα υλοποιεί αυτόματο πρόγραμμα, το οποίο ρυθμίζεται από τον διαχειριστή του συστήματος. Η κάθε κατάσταση μπορεί να ρυθμιστεί επαναλαμβανόμενα για συγκεκριμένες μέρες και ώρες. Συγκεκριμένα υπάρχουν 4 πιθανές καταστάσεις και ορίζονται ως εξής:

1) Offline. Οι λειτουργίες του συστήματος είναι απενεργοποιημένες.
2) Watch. Το σύστημα εκπέμπει ζωντανή ροή Video, χωρίς έξυπνη αναγνώριση και αποστολή ειδοποιήσεων.
3) Smart. Παρόμοιο με την κατάσταση Watch αλλά με ενεργοποιημένες τις υπόλοιπες λειτουργίες.
4) Alert. Περιλαμβάνει την κανονική ροή του συστήματος. Παρόμοιο με την κατάσταση Offline, όμως είναι σε εγρήγορση να ενεργοποιηθεί σε περίπτωση πυροδότησης αισθητήρα.

Το Backend, αποτελεί υλοποίηση από μερικές χιλιάδες γραμμές κώδικα και δεν είναι δυνατή η εξήγηση του ολοκληρωτικά, γι αυτό στις επόμενες υποενότητες θα αναφερθούν οι κυριότερες του λειτουργίες.

6.4.2 Ανάπτυξη Live Streaming

Το βίντεο streaming αποτελεί υλοποίηση τριών μερών. Αρχικά, το κύριο Backend, που διαχειρίζεται την έναρξη και λήψη της ζωντανής ροής. Το Native Backend, που ενώνει τις ροές των συσκευών εισόδου σε μία και την στέλνει στο 3ο μέρος ως RTMP μορφή. Το 3ο μέρος, αποτελεί τον Nginx server, ο οποίος, δημιουργεί τα αρχεία TS που χρειάζονται για το streaming μέσω HLS.

Το σύστημα έχει υλοποιηθεί έτσι ώστε, η ζωντανή ροή να παραμένει κλειστεί και να ανοίγει μόνο όταν χρειάζεται με σκοπό την διαχείριση της κατανάλωσης ενέργειας. Έτσι, υπάρχουν 3 λόγοι για να ανοίξει το Live Stream:

1) Χειροκίνητα από τον χρήστη. Ο διαχειριστής μέσω της ιστοσελίδας έχει την δυνατότητα ενεργοποίησης και απενεργοποίησης της ροής.
2) Μήνυμα αισθητήρα από ESP32. Κατά την λήψη αιτήματος MQTT, που σηματοδοτεί την πυροδοτήσει αισθητήρα, ανοίγει η ζωντανή ροή, μόνο για ένα ορισμένο χρονικό διάστημα, που παρέχεται από τις ρυθμίσεις του διαχειριστή.
3) Αλλαγή κατάστασης του συστήματος. Με την αλλαγή κατάστασης σε Watch ή Smart, ενεργοποιείται η ζωντανή ροή. Ενώ με την αλλαγή σε Offline ή Alert απενεργοποιείται.

Η έναρξη του Live Stream γίνεται μέσω του Native Backend, συγκεκριμένα υπάρχουν 2 πιθανές εντολές, η πρώτη για ροή χωρίς ήχο και η δεύτερη με ήχο. Η ενσωμάτωση ήχου εξαρτάται από την ύπαρξη μικροφώνου και τις ρυθμίσεις του διαχειριστή.

Εντολή χωρίς ήχο:

```
libcamera-vid -t 0 --inline --width 960 --height 540 --framerate 25 --
bitrate 750000 --codec libav --libav-format flv -n -o
rtmp://0.0.0.0/live/stream
```

Η εντολή αυτή χρησιμοποιεί την βιβλιοθήκη libcamera-vid, για να ενεργοποιήσει την κάμερα και να λάβει τα απαραίτητα δεδομένα. Στην συνέχεια ρυθμίζει την ροή και την στέλνει στο Nginx. Στον πίνακα 6.1, παρατηρείται η σημασία της κάθε παραμέτρου.

Πίνακας 6.1: Libcamera arguments για Live Streaming χωρίς ήχο.

Όνομα παραμέτρου	Τιμή παραμέτρου	Σημασία
-t	0	Χρόνος καθυστέρησης έναρξης.
–inline	-	Συγχρονισμός ζωντανής ροής
–width	960	Πλάτος βίντεο
–height	540	Ύψος βίντεο
–framerate	25	Ρυθμός καρέ το δευτερόλεπτο (FPS)
–bitrate	750 kbps	Ρυθμός κωδικοποίησης βίντεο.
–codec	libav	Ορισμός κωδικοποίησης βίντεο
–libav-format	flv	Μορφή κωδικοποιημένου βίντεο
-n	-	Αποφυγή εμφάνισης της προεπισκόπηση ροής
-o	rtmp://0.0.0.0/live/stream	Ορισμός προορισμού της ζωντανής ροής.

Η δεύτερη εντολή, εκμεταλλεύεται ξανά την libcamera-vid και προσθέτει το εργαλείο FFmpeg για την λήψη ήχου και την ενσωμάτωση των δύο ροών σε μία:

```
libcamera-vid -t 0 --inline --width 960 --height 540 --framerate 20 --
bitrate 300000 --codec libav --libav-format flv -n -o - | ffmpeg -re -
thread_queue_size 512 -i - -f alsa -channels 1 -i
hw:${cardNumber},${deviceNumber} -c:v copy -fps_mode vfr -bufsize 300k -
maxrate 300k -minrate 200k -c:a aac -b:a 96k -ar 44100 -ac 1 -af
highpass=f=75 -flush_packets 1 -rtmp_buffer 1000 -rtmp_live live -tune
zerolatency -preset ultrafast -f flv rtmp://0.0.0.0/live/stream
```

Όπως παρατηρείται με μία πρώτη ματιά, το κομμάτι της Libcamera εντολής παραμένει σχεδόν ίδιο, υπάρχει όμως μείωση σε μερικές τιμές. Αυτό γίνεται λόγω της ύπαρξης του ήχου, διότι χρειάζεται περισσότερη υπολογιστική ισχύ η εκτέλεση της εντολής. Έτσι, μειώνεται το framerate και το bitrate για την αποφυγή καθυστερήσεων (lag) στο Live Stream. Ακολουθεί ο πίνακας για την επεξήγηση των νέων παραμέτρων που χρησιμοποιεί το FFmpeg:

Πίνακας 6.2: FFmpeg arguments για Live Streaming μαζί με ήχο.

Όνομα παραμέτρου	Τιμή παραμέτρου	Σημασία
-re	-	Υπολογισμός ροής ζωντανά.
–i	Έξοδος Libcamera	Ορισμός εισόδου βίντεο.
–f	alsa	Ορισμός μορφής ήχου εισόδου.
–channels	1	Ορισμός καναλιών ήχου εισόδου.
–i	hw:2,0	Ορισμός εισόδου συσκευής ήχου.
–c:v	copy	Ορισμός κωδικοποίησης βίντεο στην αντιγραφή της ήδη υπάρχων από Libcamera.
-fps_mode	vfr	Ορισμός τύπου FPS σε μεταβλητό ρυθμό.
-bufsize	300 kbps	Ορισμός buffer για αποφυγή καθυστερήσεων.
-maxrate	300 kbps	Ορισμός μέγιστου bit rate.
-minrate	200 kbps	Ορισμός ελάχιστου bit rate.
-c:a	aac	Ορισμός κωδικοποίησης ήχου.
-b:a	96 kbps	Ορισμός ρυθμού ροής ήχου.
-ar	44100	Ορισμός ρυθμού δειγμάτων ήχου
-ac	1	Ορισμός καναλιών εξόδου ήχου.
-af	highpass=f=75	Ορισμός φίλτρου για την μείωση του θορύβου.
-flush_packets	1	Ορισμός γραψίματος πακέτων για την αμεσότερη αποστολή πακέτων.
rtmp_buffer	1000	Ορισμός buffer απο την μεριά του video player για την αποφυγή καθυστερήσεων.
-rtmp_live	live	Ορισμός του Stream για την καλύτερη διαχείριση από FFmpeg.
-tune	zerolatency	Ορισμός καθυστέρησης για την βελτιστοποίηση της.
-preset	ultrafast	Ορισμός του κωδικοποιητή για την προτίμηση της ταχύτητας.
-f	flv	Ορισμός μορφής εξόδου βίντεο.

Στην συνέχεια, καλείται ο διακομιστής Nginx να διαχειριστή το RTMP Stream. Για να το πετύχει, χρειάζεται η εγκατάσταση του με πρόσθετο module. Μέσω της χρήσης docker, η εγκατάσταση γίνεται αυτόματα και στο δικό της εικονικό λειτουργικό σύστημα. Περισσότερα αναφέρονται στην ενότητα του Docker compose. Ο Nginx μπορεί να ρυθμιστεί μέσω ειδικού αρχείου, nginx.conf, με ειδικό συντακτικό που ορίζεται από το ίδιο. Ο παρακάτω κώδικας αποτελεί την μοναδική ρύθμιση του διακομιστή για την διαχείριση του RTMP Stream:

```
rtmp {
   server {
       listen 1935;

       timeout 60s;
       ping 30s;
       ping_timeout 10s;
```

```
        buflen 5s;

        application live {
            live on;
            hls on;
            hls_path /dev/shm/hls;
            hls_fragment 1;
            hls_playlist_length 3s;
            hls_cleanup on;
            hls_fragment_naming timestamp;
            hls_fragment_slicing aligned;
            wait_key on;
            wait_video on;
            drop_idle_publisher 20s;
            allow play all;
            interleave on;

            #* ENCRYPTION: Enable HLS keys
            hls_keys on;
            hls_key_path /dev/shm/keys;
            hls_key_url https://trpia.local/keys/;
            hls_fragments_per_key 10;
        }
    }
}
```

Ο Nginx ρυθμίζεται να παρέχει HLS Streaming μέσω του default port 1935. Πιο συγκεκριμένα:

1) Ενεργοποιείται η λειτουργία Live Streaming και HLS.
2) Επιλέγεται το path /dev/shm/hls ως ο χώρος αποθήκευσης των TS αρχείων. Ο χώρος αυτός βρίσκεται στην μνήμη RAM και χρησιμοποιείται για την βελτίωση της ταχύτητας προσπέλασης των αρχείων.
3) Ορίζεται το μέγεθος του κάθε fragment, δηλαδή TS αρχείου στα 1 δευτερόλεπτα. Για την λιγότερη δυνατή καθυστέρηση της ζωντανής ροής.
4) Ρυθμίζεται η διαγραφή των παλιών fragment, για την εξοικονόμηση χώρου αποθήκευσης.
5) Ορίζεται η αναμονή του διακομιστή, για την σύνδεση του χρήστη, προτού την έναρξη παραγωγής segments.
6) Ενεργοποιούνται λειτουργίες κρυπτογράφησης των TS αρχείων, για την εφαρμογή ιδιωτικότητας.

Τέλος η πρόσβαση του Stream γίνεται μέσω του path /hls/stream.m3u8:

```
    http{
        server{
            …
            location /hls{
                types {
                    application/dash+xml mpd;
                    application/vnd.apple.mpegurl m3u8;
```

```
                video/mp2t ts;
            }

            root /dev/shm;
            add_header Cache-Control no-cache;
            add_header Access-Control-Allow-Origin "*" always;
            add_header Access-Control-Allow-Methods "GET, OPTIONS" always;
            add_header Access-Control-Allow-Headers "*" always;
            etag on;
        }
        location /keys{
            root /dev/shm;
            add_header Cache-Control no-cache;
            add_header Access-Control-Allow-Origin "*" always;
            add_header Access-Control-Allow-Methods "GET, OPTIONS" always;
            add_header Access-Control-Allow-Headers "*" always;
            etag on;
        }
        …
```

Συνοψίζοντας, είναι δυνατή η πρόσβαση της ζωντανής ροής μέσω της ιστοσελίδας. Για τον browser Safari, η χρήση HLS υποστηρίζεται πλήρως ενώ για τους υπόλοιπους υλοποιείται HLS Video Player, που θα αναλυθεί στο αντίστοιχο κεφάλαιο στην πορεία της υλοποίησης. Επιπλέον, το Live Streaming περιέχει πολλές ρυθμίσεις και είναι σημαντική η σωστή ρύθμιση με βάση τις ανάγκες της υλοποίησης και των δυνατοτήτων του μηχανήματος.

6.4.3 Ενσωμάτωση μοντέλων έξυπνης αναγνώρισης

Η έξυπνη αναγνώριση προϋποθέτει την αναγνώριση εικόνας, προσώπων και ήχου. Η αναγνώριση εικόνας και ήχου γίνεται μέσω προεκπαιδευμένων μοντέλων μαζί με το εργαλείο Tensorflow, ενώ η αναγνώριση προσώπων υλοποιείται μέσω της Insightface βιβλιοθήκης.

Αρχικά, όσον αφορά την αναγνώριση εικόνας, χρησιμοποιείται το εργαλείο FFmpeg για την εξαγωγή εικόνων από την ζωντανή ροή.

```
ffmpeg -i rtmp://0.0.0.0/live/stream -an -vf fps=2,scale=300:300 -f
image2pipe -vcodec mjpeg -q:v 3 -
```

Μέσω του FFmpeg:

1) Ορίζεται ως είσοδο την ζωντανή ροή του RTMP Stream.
2) Απενεργοποιείται την επεξεργασία του ήχου, καθώς δεν χρειάζεται.
3) Ορίζονται φίλτρα για την δειγματοληψία των καρέ στα 2 το δευτερόλεπτο και γίνεται αλλαγή μεγέθους την εικόνας στα 300x300, που απαιτείται το μοντέλο αναγνώρισης.
4) Ορίζεται η έξοδος της ροής με κάθε καρέ, μια εικόνα JPEG μορφής.
5) Δηλώνεται υψηλή ποιότητα της εικόνας, διαθέσιμες τιμές είναι από 1 έως 31.

```
export const getObjectRecognitionPrediction = async (
  frame: Buffer,
): Promise<ObjectRecognitionPrediction[]> => {
  const model = Models.getModel('coco_ssd');
  if (!model) {
    return [];
  }

  try {
    //* Convert to Uint8Array as coco expects to decode an image from.
    const bytes = new Uint8Array(frame);
    let inputTensor = tf.node.decodeImage(bytes) as tf.Tensor3D;

    //* If the 3rd dimension of the tensor has 4 channels, remove the alpha channel
    if (inputTensor.shape[2] === 4) {
      //* [0,0,0] for each channel
      //* -1 = keep the whole tensor
      //* 3 = keep the 3 first channels
      inputTensor = inputTensor.slice([0, 0, 0], [-1, -1, 3]);
    }
    let output = await model.detect(inputTensor);

    //* Filter out unwanted classes
    output = output.filter(p => ALLOWED_CLASSES.includes(p.class));
    return output.map(p => ({...p, type: 'image'}));
  } catch (err) {
    console.log('Error processing image', err);
    return [];
  }
};
```

Σχήμα 6.15: Συνάρτηση πρόβλεψης αντικειμένων σε καρέ εικόνας.

Η έξοδος της εντολής, περνάει στην είσοδο διεργασίας, όπου με την χρήση Tensorflow μετατρέπεται σε Tensor και στην μορφή που απαιτεί το μοντέλο Coco SSD. Τέλος γίνεται η πρόβλεψη των αντικειμένων στην εικόνα και στέλνονται στο Backend για να μετρήσει την πιθανότητα εισβολής και να διαχειριστεί την έναρξη της. Στην περίπτωση πρόβλεψης ανθρώπου, πρώτα ακολουθεί η διαδικασία αναγνώρισης προσώπων.

Η διαδικασία αναγνώρισης προσώπων, συνεχίζει την αναγνώριση εικόνας στην περίπτωση που υπάρχει άνθρωπος, σύμφωνα με το μοντέλο CocoSSD. Έτσι, μέσω της εξόδου bbox του μοντέλου CocoSSD, πραγματοποιείται περικοπή της εικόνας για να περιέχει μόνο τον άνθρωπο. Στην συνέχεια, ακολουθεί η έξοδος στην είσοδο της διεργασίας που διαχειρίζεται το μοντέλο για την αναγνώριση προσώπων, δηλαδή του ArcFace μέσω της InsightFace.

Η διεργασία για την αναγνώριση προσώπων ακολουθεί τα ακόλουθα βήματα:

1) Φορτώνει τα αρχεία προσώπων που έχουν δοθεί από τον διαχειριστή, το κάθε αρχείο περιέχει τις συντεταγμένες των μοναδικών χαρακτηριστικών του κάθε ανθρώπου.
2) Λαμβάνει το καρέ από την αναγνώριση εικόνας, περικομμένο να περιέχει μόνο τον άνθρωπο.
3) Μετατρέπει το καρέ σε μορφή που απαιτείται, πίνακας θετικών ακεραίων των 8bit.
4) Αναλύει την εικόνα για να βρει όλα τα πρόσωπα στην εικόνα.

5) Κάθε πρόσωπο που υπάρχει στο καρέ, έρχεται στην μορφή, embedding, δηλαδή την ίδια μορφή με τα αρχεία των αποθηκευμένων προσώπων.
6) Για κάθε embedding, υπολογίζεται το ποσοστό ομοιότητας με κάθε αποθηκευμένο embedding.
7) Η μέγιστη πιθανότητα ομοιότητας του κάθε προσώπου στην εικόνα, στην περίπτωση που ξεπερνάει το κατώφλι, περνάει στην έξοδο της διεργασίας και στην συνέχεια στέλνετε στο Backend.

```python
def find_best_match(self, embeddingToRecognize):
    if not self.knownEmbeddingsNormalized:
        return None, 0.0

    normEmbeddingToRecognize = np.linalg.norm(embeddingToRecognize)
    if normEmbeddingToRecognize == 0:
        return None, 0.0

    normalizedEmbeddingToRecognize = (embeddingToRecognize / normEmbeddingToRecognize).astype(np.float32)

    maxSimilarity = -1.0
    maxName = None

    for name, normalizedKnownEmbedding in self.knownEmbeddingsNormalized.items():
        #* Calculate cosine similarity using dot product
        similarity = np.dot(normalizedKnownEmbedding, normalizedEmbeddingToRecognize)

        if similarity > maxSimilarity:
            maxSimilarity = similarity
            maxName = name

    return maxName, float(maxSimilarity)
```

Σχήμα 6.16: Συνάρτηση σύγκρισης ομοιότητας προσώπων μέσω των embeddings.

Στην εικόνα 6.16, παρατηρείται η σύγκριση του προσώπου που εμφανίστηκε στην εικόνα εναντίον των αποθηκευμένων προσώπων. Αρχικά, γίνεται μετατροπή του embedding, σε κανονικοποιημένη μορφή, για την αποδοτικότερη πρόβλεψη του μοντέλου. Στην συνέχεια υπολογίζεται η πιθανότητα πρόβλεψης μέσω του συνημιτόνου της γωνίας μεταξύ των δύο embeddings, που αποτελούν το κάθε ένα, ένα κανονικοποιημένο διάνυσμα. Η έξοδος αποτελεί την τιμή γωνίας του συνημιτόνου, μεταξύ των δύο embeddings:

$$cos(\theta) = \vec{normalizedKnownEmbedding} * \vec{normalEmbeddingToRecognize}$$

Η έξοδος της γωνίας θ, λαμβάνει τιμές:

$$\theta \in [0°, 180°] \implies \cos(\theta) \in [1, -1]$$

Έτσι στην περίπτωση, που τα δύο διανύσματα, δείχνουν προς την ίδια κατεύθυνση, η έξοδος παίρνει τιμές κοντά στο 1, ενώ στην αντίθετη περίπτωση τιμές κοντά στο -1, και σηματοδοτούν την τιμή ομοιότητας των δύο προσώπων. Δηλαδή:

$$|cos(theta)| * 100 = SimilarityPercentage$$

Για τιμές από 0 έως 1, υπολογίζεται το ποσοστό ομοιότητας των δύο προσώπων. Στην συνέχεια, γίνεται ελέγχεται αν η τιμή αυτή ξεπερνάει το κατώφλι του 0.5, δηλαδή η διαφορά γωνίας των δύο διανυσμάτων είναι μικρότερη των 60 μοιρών.

Κλείνοντας απλο την αναγνώριση εικόνας και προσώπων, απομένει η αναγνώριση ήχου. Η οποία, μέσω του εργαλείου FFmpeg, γίνεται εξαγωγή του ήχου από το Live Stream, και δίνεται ως είσοδος στο μοντέλο Yamnet για την αναγνώριση ήχων:

Εξαγωγή ήχου:

```
ffmpeg -i rtmp://0.0.0.0/live/stream -sample_fmt s16 -vn -f wav -ac 1 -ar 16000 -
```

Επιπλέον, δίνονται παράμετροι για την εξαγωγή ήχου, με σκοπό η έξοδος να είναι συμβατή με τις ανάγκες του μοντέλου Yamnet:

1) Ορίζεται το κάθε δείγμα σε ακέραιο των 16-bit.
2) Αγνοείται η ροή βίντεο της Live Stream.
3) Δίνεται η μορφή του ήχου ως Wav format.
4) Ορίζεται 1 κανάλι ήχου.
5) Γίνεται επιλογή των ρυθμού δειγματοληψίας στα 16 kHz.

Στην συνέχεια, παρέχεται η έξοδος στην είσοδο της διεργασίας που διαχειρίζεται το μοντέλο αναγνώρισης ήχου μέσω Tensorflow.

```
export const getAudioRecognitionPrediction = async (
  audioBuffer: Buffer,
): Promise<RecognitionResult> => {
  //* Audio buffer data is in 16-bit signed integer format
  const audioData = new Int16Array(audioBuffer);
  //* Normalize to float32 [-1,1] as yamnet expects
  const audioFloat = new Float32Array(audioData.length);
  for (let i = 0; i < audioData.length; i++) {
    audioFloat[i] = audioData[i] / 32768.0; // 32768 = 2^15 -> [-1,1]
  }
  const model = Models.getModel('yamnet');
  if (!model) {
    return {predictions: [], audioFloat: audioFloat, spectogram: []};
  }
  try {
    const inputTensor = tf.tensor1d(audioFloat);

    const [scores, embeddings, spectrogram] = model.predict(
      inputTensor,
    ) as tf.Tensor[];
    const scoresArray = scores.mean(0).dataSync();

    //* Get top 5 predictions over > 0.4 score
    const topScores: Prediction[] = Array.from(scoresArray)
      .map((score, index) => ({score, index}))
      .filter(({index}) => !EXCLUDED_CLASSES.includes(index))
      .filter(({score}) => score > 0.4)
      .sort((a, b) => b.score - a.score)
      .slice(0, 5)
      .map(({score, index}) => ({
        score,
        class: audioClasses?.[index]?.display_name ?? 'not_found',
        type: 'audio',
      }));
```

Σχήμα 6.17: Συνάρτηση αναγνώρισης δειγμάτων ήχου.

Στην εικόνα φαίνεται απόσπασμα κώδικα για την αναγνώριση ήχου. Αρχικά, γίνεται μετατροπή του ήχου, που αποτελείται από έναν πίνακα ακεραίων 16-bit, σε κανονικοποιημένη μορφή. Στην συνέχεια, δημιουργείται το Tensor του μονοδιάστατου πίνακα και πραγματοποιείται η πρόβλεψη μέσω του μοντέλου Yamnet. Το αποτέλεσμα αποτελεί έναν πίνακα με το ποσοστό πρόβλεψης για κάθε κλάση στην οποία έχει εκπαιδευτεί το μοντέλο. Γι αυτό πραγματοποιείται φιλτράρισμα των δεδομένων, για την αφαίρεση της πρόβλεψης θορύβου. που μπορεί να προσθέτει το μικρόφωνο και αποστέλλονται στο Backend οι 5 μεγαλύτερες προβλέψεις, με κατώτατο όριο στο 40%.

Συνοψίζοντας, το Backend, μέσω WebSockets, λαμβάνει τις προβλέψεις και εκτελεί ενέργειες για την καλύτερη εξακρίβωση της πιθανότητας εισβολής. Περισσότερα για την διαχείριση των προβλέψεων θα αναφερθούν στην επόμενη υποενότητα.

6.4.4 Ανίχνευση και διαχείριση εισβολής

Η διαχείρισης εισβολής γίνεται μέσω του κύριου Backend, που είναι υλοποιημένο στην γλώσσα Go. Κατά την αποστολή αιτήματος για την έναρξη του της ζωντανής ροής, εγγράφεται μέσω WebSocket, στο Native Backend, για να λαμβάνει μηνύματα προβλέψεων. Στην συνέχεια αρμοδιότητες του αποτελούν:

1) Αποθήκευση των προβλέψεων στην μνήμη.
2) Συνάθροιση του αριθμού προβλέψεων για κάθε τύπο πρόβλεψης, όπως ήχου πρόσωπο και εικόνα.
3) Υπολογισμός μέσου όρου πιθανότητας για κάθε κλάση.
4) Στην περίπτωση που ο αριθμός και ο μέσος όρος πιθανότητας των προβλέψεων, ξεπεράσει τα κατώφλια που έχουν οριστεί, αρχίζει την κατάσταση συναγερμού.

```go
/**
 * * Start object recognition handler for alerting the user
 */
func StartImageRecognitionHandler(){
    fmt.Println("Starting object recognition handler")
    settings := state.GetCachedSettings()

    state.CloseImageRecognitionChannel = make(chan struct{})

    for {
        select {
            case <- state.CloseImageRecognitionChannel:
                fmt.Println("[CLOSE] Object recognition handler stopped")
                state.CloseImageRecognitionChannel = nil
                return
            case prediction := <- state.ImagePredictionChannel:
                go func(prediction models.Prediction){
                        if prediction.Class == ""{
                            return
                        }
                        //* Handle Intrusion Alert
                        intrusion.ImagePredictionAssessment(prediction, settings.TriggerHumanOnly)
                }(prediction)
        }
    }
}
```

Σχήμα 6.18: Συνάρτηση λήψης πρόβλεψης, για Image Recognition στην Go.

Η εικόνα 6.18, αποτελεί την ανάγνωση του channel που περιέχει την πρόβλεψη του Coco SSD. Η συνάρτηση αυτή εκτελείται σε διαφορετικό thread στο παρασκήνιο. Σκοπός της είναι η συνεχή ανάγνωση του καναλιού ImagePredictionChannel, μέχρι αυτό να κλείσει στην περίπτωση διακοπής της

ζωντανής ροής. Τα κανάλια, αποτελούν τρόπος μεταφοράς δεδομένων μεταξύ διαφορετικών threads στην Go. Μόλις ληφθή ένα μήνυμα, ο WebSocket client, ελέγχει αν πρόκειται για πρόβλεψη εικόνας και το εγγράφει στο κανάλι αυτό. Στην συνέχεια μέσω της συνάρτησης StartImageRecognitionHandler, καλείται η ImagePredictionAssessment η οποία συσσωρεύει τις προβλέψεις και υπολογίζει την αθροιστική πιθανότητα εισβολής.

```go
func ImagePredictionAssessment(prediction models.Prediction, triggerHumanOnly bool){

    if(atomic.LoadInt32(&state.IntrusionAlertActive) == 1){
        fmt.Println("Intrusion alert already started, ignoring prediction")
        return
    }

    predictionMutex.Lock()
    defer predictionMutex.Unlock()

    //* Ignore low confidence predictions
    if prediction.Score < imagePredictionIncludeThreshold {
        // fmt.Println("Prediction score too low, ignoring")
        return
    }
    fmt.Printf("[%s-%f] chance of %s\n", prediction.Type,prediction.Score, prediction.Class)

    var classData = detections[prediction.Class]
    classData.Add(prediction.Score)
    detections[prediction.Class] = classData

    //* Do not assess trigger intrusion if the class is not a person and triggerHumanOnly is true
    if triggerHumanOnly && strings.ToLower(prediction.Class) != "person" {
        return
    }

    if classData.AverageScore >= imagePredictionScoreTriggerThreshold && classData.TotalCount >= imagePredictionCountTriggerThreshold {
        if !atomic.CompareAndSwapInt32(&state.IntrusionAlertActive, 0, 1){
            fmt.Println("Intrusion alert already started, ignoring prediction")
            return
        }
        fmt.Printf("[INTRUSION DETECTED] - Prediction average: %f, Prediction total: %d\n", classData.AverageScore, classData.TotalCount)
        go StartIntrusionAlert(
            models.Prediction{Class: prediction.Class, Score: classData.AverageScore, Type: models.PREDICTION_IMAGE})
    }else{
        if totalPredictions > totalImagePredictionsResetThreshold {
            resetPredictionMetrics()
            return
        }
    }
}
```

Σχήμα 6.19: Συνάρτηση αξιολόγησης προβλέψεων, για Image Recognition στην Go.

Στην εικόνα 6.19, φαίνεται η υλοποίηση της αξιολόγησης των προβλέψεων από το Image Recognition. Αρχικά γίνονται μερικοί έλεγχοι:

1) Μέσω της atomic μεταβλητής, οι οποία επιμένει την τιμή της μεταξύ διαφορετικών threads, ελέγχεται αν το σύστημα βρίσκεται σε κατάσταση συναγερμού. Στην περίπτωση αυτή, απορρίπτεται η αξιολόγηση της πρόβλεψης

2) Καθώς η κάθε κλήση της συνάρτησης, γίνεται με την δημιουργία νέων thread, κλειδώνεται η πρόσβαση της μέχρι την ολοκλήρωση των εντολών της συνάρτησης. Μέσω του predictionMutex.lock().

3) Πραγματοποιείται έλεγχος για την πιθανότητα του μοντέλου με βάση το κατώφλι imagePredictionIncludeThreshold.

4) Ελέγχεται, η περίπτωση που η ρύθμιση του διαχειριστή περιλαμβάνει, την κατάσταση εισβολής, μόνο σε περίπτωση αναγνώρισης ανθρώπων και η πρόβλεψη έχει κλάση Person. Στην περίπτωση αυτή, απορρίπτεται η έναρξη της κατάστασης εισβολής.

5) Τέλος, ελέγχεται αν ο μέσος όρος πιθανότητας της κλάσης πρόβλεψης και ο συνολικός αριθμός προβλέψεων, είναι μεγαλύτερα των κατωφλίων. Στην περίπτωση αυτή, ξεκινάει η κατάσταση συναγερμού.

Στην συνέχεια, θα αναφερθεί ο τρόπος λειτουργίας της κατάστασης συναγερμού.

```
func StartIntrusionAlert(predictionTrigger models.Prediction){
    fmt.Println("---- STARTING INTRUSION ALERT ----")

    if state.Services.IntrusionAlert == models.ServiceOffline {
        state.Services.IntrusionAlert = models.ServiceOperating
        webBridge.PublishServicesUpdate(state.Services)
    }

    intrusionAlert = &models.IntrusionAlert{}
    intrusionAlert.StartDate = time.Now().Local().UnixMilli()

    //* Start recording video
    settings := state.GetCachedSettings()
    go startSingleVideoRecording(int32(settings.KeepCameraOnForAfterIntrusion))
    timeouts.ResetStreamCloseTimeout(int(settings.KeepCameraOnForAfterIntrusion))

    //* Get all classes that are most likely included in the intrusion.
    intruderClasses := make([]models.Prediction, 0)
    for class, data := range detections {
        if data.AverageScore >= imagePredictionIncludeThreshold && data.TotalCount >= imagePredictionCountIncludeThreshold {
            intruderClasses = append(intruderClasses,
                models.Prediction{Class: class, Score: data.AverageScore, Type: models.PREDICTION_IMAGE})
        }
    }
    fmt.Println("Intruder classes: ", intruderClasses)
    intrusionAlert.PredictionsTrigger = intruderClasses

    alertUser(intruderClasses, predictionTrigger, intrusionAlert.StartDate)
    webBridge.PublishIntrusionAlertMessage(*intrusionAlert, models.IntrusionAlertTypeStart)
}
```

Σχήμα 6.20: Συνάρτηση εκκίνησης συναγερμού.

Η κατάσταση συναγερμού, περιλαμβάνει την ειδοποίηση του χρήση μέσω SMS και Push Notifications, την ενημέρωση του Web app και την έναρξη καταγραφής βίντεο.

Όπως φαίνεται στο σχήμα 6.20, αρχικά ενημερώνονται οι συνδεδεμένοι Web Clients για την έναρξη συναγερμού. Στην συνέχεια αρχίζει η καταγραφή Video σε διαφορετικό thread, μέσω του keyword go, και υπολογίζονται οι κλασεις που συμμετέχουν στην εισβολή, σε περίπτωση πάνω από ενός διαφορετικού τύπου πρόβλεψης. Τέλος, ενημερώνονται οι Web Clients και στέλνονται τα μηνύματα SMS και Notifications μέσω της alertUser.

Η αποστολή ειδοποιήσεων και μηνυμάτων SMS, αναφέρονται στα αντίστοιχα κεφάλαια υλοποίησης. Επιπλέον, η καταγραφή βίντεο πραγματοποιείται μέσω του εργαλείου FFmpeg και αναλύεται στην συνέχεια.

```
func startSingleVideoRecording(video_duration int32){
    fmt.Println("Starting video recording")
    currentDate := time.Now().Format("2006-01-02_15-04-05")
    fileName := api.RecordingsDir+"/video_"+currentDate+".mp4"
    recordingCommand = exec.CommandContext(
        context.Background(),
        "ffmpeg",
        "-i", globals.FFMPEG_OUTPUT,        // Input file
        "-vf", "drawtext=text='%{pts\\:hms}':x=10:y=10:fontsize=24:fontcolor=white:box=1:boxcolor=black@0.5",
        "-c:v", "libx264",                  // Video codec
        "-b:v", "2M",                          // Video bitrate
        "-c:a", "copy",                        // copy Audio codec
        "-preset", "fast",                  // Encoding preset
        "-crf", "18",                       // Constant Rate Factor for video quality 1-31
        "-threads", "1",
        "-t", strconv.Itoa(int(video_duration)), // Set duration of output video
        fileName,                          // Output file (single file, no segments)
    )
    err := recordingCommand.Start()
    if err != nil {
        fmt.Println("Error starting ffmpeg command ", err.Error())
        return
    }
    werr := recordingCommand.Wait()
    if werr != nil {
        fmt.Println("Error waiting for ffmpeg command ", werr.Error())
        return
    }
```

Σχήμα 6.21: Καταγραφή του Live Stream στην περίπτωση συναγερμού.

Κατά την κατάσταση συναγερμού, το σύστημα καταγράφει την ζωντανή ροή και την αποθηκεύει τοπικά, έτσι ώστε ο διαχειριστής να μπορεί να παρακολουθήσει όποτε θέλει την διάρκεια της εισβολής. Όπως φαίνεται στο σχήμα 6.21, μέσω του FFmpeg, ξεκινάει η καταγραφή και αποθηκεύεται σε αρχείο τύπου MP4. Η διάρκεια επιλέγεται από τον διαχειριστή μέσω των ρυθμίσεων. Επιπλέον, στο βίντεο, ζωγραφίζεται αυτόματα στο πάνω αριστερά μέρος, η χρονοσήμανση για την καλύτερη εμπειρία παρακολούθησης.

Συνοψίζοντας, παρόμοια λογική αξιολόγησης των προβλέψεων υλοποιείται στις περιπτώσεις ήχου και προσώπων. Με την διαφορά, στην περίπτωση αναγνώρισης προσώπου, το σύστημα δεν μπαίνει σε κατάσταση συναγερμού, αλλά, ειδοποιείται ο ιδιοκτήτης για την αναγνώριση των συγκεκριμένων προσώπων.

6.4.5 Μηνύματα μέσω SMS

Στην υποενότητα αυτή, θα αναφερθεί ο τρόπος ειδοποίησης του διαχειριστή μέσω SMS, καθώς, και ο τρόπος έναρξης της ζωντανής ροής από τον χρήστη μέσω SMS. Η υλοποίηση γίνεται μέσω της Go, η οποία εκμεταλλεύεται το Serial Port της συσκευής /dev/ttyAMA0 και συνεπάγεται με την συσκευή GSM HAT, που είναι συνδεδεμένη με το Raspberry Pi. Έτσι, μέσω αυτής της σειριακής θύρας, είναι δυνατή η αποστολή AT commands προς το Modem.

Για την αποστολή AT εντολών προς την GSM συσκευή, αρχικά χρειάζεται η δημιουργία σύνδεσης προς την σειριακή θύρα /dev/ttyAMA0 μέσω της Go.

```go
func (s *SMSConnection) OpenConnection() *serial.Port{
    fmt.Println("Opening port connection")
    _port, _ := s.Port.Load().(*serial.Port)
    if _port != nil {
        fmt.Println("Port already open")
        return nil
    }
    port, err := serial.Open(globals.SMS_SERIAL_PORT, &serial.Mode{
        BaudRate: 115200,
    })
    if err != nil {
        fmt.Println("Error opening port:", err)
        return nil
    }
    fmt.Println("Serial port opened!")
    s.Port.Store(&port)
    return &port
}
```

Σχήμα 6.22: Δημιουργία σύνδεσης προς την σειριακή θύρα /dev/ttyAMA0 μέσω Go.

Όπως φαίνεται στην εικόνα 6.22, η Go προσφέρει εργαλεία για την ασφαλή δημιουργία της σύνδεσης και επιπλέον δεν χρειάζεται χρήση επιπλέον βιβλιοθηκών, καθώς είναι ενσωματωμένες όλες οι λειτουργίες στην ίδιο πακέτο της Go. Αρχικά, ελέγχεται αν υπάρχει ήδη φορτωμένο το Port στην μνήμη, στην περίπτωση που είναι κλειστή η σύνδεση, πραγματοποιείται το άνοιγμα της σε ρυθμό baud 115200, τον μέγιστο ρυθμό ταχύτητας επικοινωνίας που υποστηρίζει το GSM Module. Τέλος, αποθηκεύεται σε στην μνήμη, η διεύθυνση της, για να χρησιμοποιηθεί αργότερα στην αποστολή SMS.

Πολύπλοκη διαδικασία. αποτελεί η αποστολή AT εντολών, καθώς το πρόγραμμα, πρέπει να παραμείνει σε αναμονή μέχρι να λάβει απάντηση από την σειριακή θύρα, δηλαδή το GSM Modem. Η διαδικασία αυτή πρέπει να γίνει σειριακά και συγχρονισμένα για κάθε εντολή AT, όπου η αποστολή SMS χρειάζεται 3 εντολές για την πραγματοποίηση της.

```go
func (s * SMSConnection) SendAtCommand(command string, delay time.Duration) (string, error) {
    port, _ := s.Port.Load().(*serial.Port)
    if port == nil {
        fmt.Println("Port not open")
        return "", errors.New("Port not open")
    }
    n, err := (*port).Write([]byte(command + "\r\n"))
    if err != nil || n == 0{
        return "", err
    }
    buffer := make([]byte, 1024)
    response := strings.Builder{}
    resultChan := make(chan string, 1)
    errorChan := make(chan error, 1)
    go func(){
        for {
            //* n is the number of bytes read from the buffer.
            n, err := (*port).Read(buffer)
            if err != nil {
                if err == io.EOF {
                    break // End Of Input
                }
                errorChan <- fmt.Errorf("failed to read response: %v", err)
            }
            if n == 0 {
                return
            }
            response.Write(buffer[:n])
            respStr := response.String()
            if strings.Contains(respStr, "OK") || strings.Contains(respStr, "ERROR") {
                resultChan <- respStr
                return
            }
        }
    }()
    select {
        case resp := <-resultChan:
            if resp == "" {
                return "", errors.New("no response received")
            }
            return strings.TrimSpace(resp), nil
        case err := <-errorChan:
            return "", err
        case <-time.After(delay):
            return "", errors.New("timeout waiting for response from SMS module")
    }
```

Σχήμα 6.23: Συνάρτηση αποστολής AT εντολής μέσω σειριακής θύρας.

Στην εικόνα 6.23, φαίνεται η υλοποίηση της συνάρτησης για την αποστολή AT εντολών, ικανοποιώντας τις ανάγκες που αναφέρθηκαν προηγουμένως. Αν και αποτελεί περίπλοκη υλοποίηση μέσω Go, τα χαρακτηριστικά της γλώσσας προς τον συγχρονισμό αναγκάζουν μία προβλέψιμη συμπεριφορά, χωρίς σφάλματα. Στο απόσπασμα, αρχικά, πραγματοποιείται το γράψιμο του AT command στην θύρα. Στην συνέχεια, δημιουργείται ένα thread, υπεύθυνο για την ανάγνωση της απάντησης της εντολής. Η απάντηση αποτελείται από διάφορες εξόδους χαρακτήρων, ανάλογα με την εντολή εισόδου, και συνήθως αποτελούνται από μερικές σειρές. Η κάθε απάντηση τελειώνει με την λέξη OK, για την επιτυχία και την λέξη ERROR σηματοδοτώντας σφάλμα επεξεργασίας του αιτήματος. Στην περίπτωση τέλους, επιστρέφεται η απάντηση για να συνεχίζει η ροή της αποστολής SMS, η οποία θα αναφερθεί στην συνέχεια.

```
fmt.Println("Sending SMS message: ", message)
//* Set SMS mode to text
s.SendAtCommand("AT+CMGF=1", time.Second)
for _, phoneNumber := range phoneNumbers {
    fmt.Println("Sending SMS to: ", phoneNumber)
    //* set SMS to phoneNumber
    s.SendAtCommand(fmt.Sprintf("AT+CMGS=\"%s\"", phoneNumber), time.Second)
    //* send message text
    success, err := s.sendTextMessage(message, time.Second * 3)
    fmt.Println("Success: ", success)
    if err != nil {
        fmt.Println("Error sending SMS message: ", err)
        continue
    }
}
```

Σχήμα 6.24: Αποστολή ειδοποίηση SMS σε αριθμό τηλεφώνου.

Το απόσπασμα κώδικα, στην εικόνα 6.24, αποτελεί την αποστολή SMS σε αριθμό κινητής τηλεφωνίας. Παρατηρείται η χρήση 2 εντολών AT και στην συνέχεια η αποστολή του κειμένου:

Πίνακας 6.3: Πίνακας AT εντολών, για την αποστολή SMS.

AT Command	Σημασία
AT+CMGF=1	Αλλαγή κατάστασης SMS, σε λειτουργία TEXT. Χρησιμοποιείται για την αποστολή μηνυμάτων.
AT+CMGS=+306912345678	Ορισμός αριθμού προορισμού.
-	Το μήνυμα σε μορφή κειμένου από χαρακτήρες.

Οι αριθμοί για την αποστολή SMS, λαμβάνονται από τους χρήστες του συστήματος με τον ρόλο διαχειριστή (Admin). Στον καθένα, κατά την έναρξη συναγερμού, γίνεται αποστολή χαρακτηριστικού μηνύματος που περιέχει τα ονόματα των κλάσεων που συμμετέχουν στην εισβολή.

Επιπλέον AT εντολές, που χρησιμοποιούνται είναι:

Πίνακας 6.4: Πίνακας AT εντολών, για την πραγματοποίηση ελέγχων.

AT Command	Σημασία
AT	Έλεγχος κατάστασης επικοινωνίας με GSM Module.
AT+CPIN?	Έλεγχος κατάστασης PIN της SIM. Δηλαδή, αν χρειάζεται εισαγωγή PIN.
AT+CPIN=1234	Εισαγωγή PIN για να ξεκλειδώσει η SIM.
AT+CMGL=”REC UNREAD”	Ανάγνωση μη διαβασμενων μηνυματων.

Συνοψίζοντας, ο διαχειριστής μπορεί μέσω SMS μηνύματος, start και close, να ανοίξει και να κλείσει την ζωντανή ροή του συστήματος. Δίνοντας, στον διαχειριστή απομακρυσμένη πρόσβαση στο Live Stream.

6.4.6 Υλοποίηση επικοινωνίας Bluetooth

Το σύστημα κατά την έναρξη, εκτελεί αυτόματες εργασίες, όπως έχει αναφερθεί και και στο κεφάλαιο της εφαρμογής του Raspberry Pi. Η πρώτη εργασία, που εκτελείται, διαχειρίζεται την σύνδεση σε δίκτυο. Στην περίπτωση που το Raspberry Pi, δεν μπορεί να συνδεθεί ή δεν έχει λάβει στοιχεία WiFi, τότε ενεργοποιείται η διαδικασία του Bluetooth.

Για την υλοποίηση της σύνδεσης Bluetooth χρειάζεται:

1) Η ρύθμιση του αντάπτορα Bluetooth.
2) Η δημιουργία Bluetooth Agent.
3) Η ρύθμιση του RFCOMM.

Η ρύθμιση του αντάπτορα γίνεται μέσω του bluetoothctl, που αποτελεί CLI εργαλείο για την διαχείριση του Bluetooth αντάπτορα:

```
bluetoothProcess = spawn('sudo', ['bluetoothctl'], {
  stdio: ['pipe', 'pipe', 'pipe'],
});
if (!bluetoothProcess) {
  console.log('Error starting Bluetooth Agent');
  process.exit(1);
}
bluetoothProcess.stdout?.setEncoding('utf8');
bluetoothProcess.stderr?.setEncoding('utf8');

//* Start agent
bluetoothProcess.stdin?.write('power on\n');
bluetoothProcess.stdin?.write('discoverable-timeout 3600\n');
bluetoothProcess.stdin?.write('discoverable on\n');
bluetoothProcess.stdin?.write('pairable on\n');
```

Σχήμα 6.25: Απόσπασμα ρύθμισης Bluetooth αντάπτορα μέσω Nodejs.

Στην εικόνα 6.25, παρατηρείται η ρύθμιση του αντάπτορα Bluetooth που ισοδυναμεί με την εκτέλεση των ακόλουθων εντολών μέσω τερματικού:

```
sudo bluetoothctl
power on
discoverable-timeout 3600
discoverable on
pairable on
```

Οι εντολές αυτές, συνεπάγονται, με την ενεργοποίηση του αντάπτορα, και τον κάνει ορατό και συνδέσιμο από συσκευές που βρίσκονται σε εύρος σύνδεσης. Η υλοποίηση γίνεται μέσω Nodejs (JavaScript), και όχι μέσω shell script καθώς είναι ευκολότερη η διαδικασία διαχείρισης των μηνυμάτων Pairing, όπως φαίνεται από το παράδειγμα στην επόμενη εικόνα.

```
tmazarakis@tRPIa:~/diploma $ bluetoothctl
Agent registered
[CHG] Controller 2C:CF:67:88:2B:96 Pairable: yes
[bluetooth]# power on
Changing power on succeeded
[bluetooth]# discoverable on
Changing discoverable on succeeded
[bluetooth]# pairable on
Changing pairable on succeeded
[bluetooth]# agent on
Agent is already registered
[bluetooth]# default-agent
Default agent request successful
[NEW] Device AC:3E:B1:7D:B5:34 Pixel 7
Request confirmation
[agent] Confirm passkey 538187 (yes/no):
```

Σχήμα 6.26: Χειροκίνητο Bluetooth Pairing με εξωτερική συσκευή.

Στην εικόνα 6.26, μετά την ρύθμιση των Bluetooth, λαμβάνεται μία σύνδεση από την συσκευή με όνομα Pixel 7, στην συνέχεια για την δημιουργία σύνδεσης απαιτείται η ολοκλήρωση του Pairing μεταξύ των δύο συσκευών. Όπως φαίνεται, από το τερματικό του Raspberry Pi, χρειάζεται η χειροκίνητη απάντηση με yes η no, γεγονός που αφαιρεί πλήρως την αυτοματοποίηση της διαδικασίας. Γι αυτό μέσω Nodejs γίνεται η διαχείριση των εντολών και παρατηρείται στο επόμενο σχήμα.

```
bluetoothProcess.stdin?.write('agent on\n');
bluetoothProcess.stdin?.write('default-agent\n');

bluetoothProcess.stdout?.on('data', (data: Buffer) => {
  const lines = data.toString().split('\n');
  lines.forEach(_line => {
    const line = _line.trim();
    if (!line) {
      return;
    }
    console.log(`[Bluetooth Agent] ${line}`);

    if (
      line.toLowerCase().includes('(yes/no):') &&
      line.toLowerCase().includes('[agent]')
    ) {
      console.log('Confirmation received, sending yes...');
      try {
        bluetoothProcess?.stdin?.write('yes\n');
      } catch (err) {
        console.log('Error sending confirmation');
      }
```

Σχήμα 6.27: Προγραμματιστική διαχείριση Bluetooth Pairing.

Συνεπώς, μέσω του αποσπάσματος στο σχήμα 6.27, ορίζεται το 2ο βήμα της διαδικασίας, η δημιουργία του Agent. Στην συνέχεια το πρόγραμμα περιμένει δεδομένα από τον αντάπτορα Bluetooth. Μόλις ληφθούν νέα δεδομένα και περιέχουν μήνυμα για Pairing, δηλαδή για την εισαγωγή yes/no, τότε γίνεται αποδοχή της σύνδεσης. Επιπλέον, μπορεί έτσι να παρθεί το όνομα της συσκευής και η MAC διεύθυνση

του αντάπτορα της, με σκοπό το φιλτράρισμα των συνδέσεων αλλά και την αποφυγή της σύνδεσης πολλών συσκευών, γεγονός που δεν έχει υλοποιηθεί καθώς δεν χρειάζεται στην συγκεκριμένη εφαρμογή.

Καθώς, έχει πραγματοποιηθεί η σύνδεση με επιτυχία, απομένει η αποστολή WiFi στοιχείων από την συσκευή, μέσω Mobile Application, στο Raspberry Pi. Για την λήψη και αποστολή δεδομένων, χρειάζεται ένα πρωτόκολλο το οποίο θα διαβάζει μηνύματα και θα μπορεί να απαντήσει κατάλληλα. Την λύση στο πρόβλημα, παρέχει το πρωτόκολλο RFCOMM, το οποίο δημιουργεί ένα εικονικό Serial Port, παρόμοιο με αυτό που δημιουργείται για το GSM Module. Έτσι μέσω αυτής της θύρας, είναι δυνατή η αποστολή και λήψη δεδομένων προς την συνδεδεμένη συσκευή.

Στο τερματικό του Raspberry Pi, η ενεργοποίηση του RFCOMM αποτελεί μία πολύ απλή και εύκολη διαδικασία, 1 εντολής:

```
sudo rfcomm watch 0 1
```

Όπως, αναφέρθηκε, το RFCOMM δημιουργεί μία εικονική σειριακή θύρα, οι θύρες αυτές είναι διαθέσιμες ως αρχεία συσκευών στο λειτουργικό σύστημα, συνεπώς μέσω της εντολής αυτής, ανοίγει η θύρα στο path /dev/rfcomm0, το 0 αποτελεί την παράμετρο που δίνεται στην εντολή, καθώς υπάρχει η δυνατότητα ανοίγματος περισσότερων θυρών. η παράμετρος 1, αποτελεί το κανάλι του RFCOMM, από όπου θα γίνεται η ανταλλαγή μηνυμάτων.

Καθώς η διαχείριση των μηνυμάτων λήψης και αποστολής πρέπει να είναι μία αυτόματη διαδικασία, η εντολή αυτή έχει υλοποιηθεί σε Nodejs, όπου προγραμματιστικά ανοίγει σύνδεση προς το Serial Port και αναμένει το πρόγραμμα σε αναμονή λήψης μηνυμάτων.

```
const RFCOMM_DEVICE = '/dev/rfcomm0';

console.log(`Checking id device RFCOMM exists: ${RFCOMM_DEVICE}`);
let attempts = 0;
console.log(`Waiting for device ${RFCOMM_DEVICE} to appear...`);
while (!fs.existsSync(RFCOMM_DEVICE)) {
  if (attempts >= 3600) {
    console.log('1 hour exceeded, exiting');
    process.exit(1);
  }
  await delay(1000);
  attempts++;
}

console.log('Opening Serial Port');
const port = new SerialPort({
  path: RFCOMM_DEVICE,
  baudRate: 9600,
});

const parser = port.pipe(
  new ReadlineParser({delimiter: '\n', encoding: 'utf8'}),
);
```

Σχήμα 6.28: Άνοιγμα RFCOMM Serial Port μέσω Nodejs.

Στην εικόνα 6.28, αρχικά παρατηρείται η αναμονή για την εμφάνιση του αρχείου συσκευής /dev/rfcomm0, που διαρκεί έως 1 ώρα, όσο και η ενεργοποίηση του Bluetooth αντάπτορα. Στην περίπτωση εύρεσης, δημιουργείται η σύνδεση προς την σειριακή θύρα. Στην συνέχεια ακολουθεί η αναμονή για μηνύματα μέσω της συνδεδεμένης συσκευής Bluetooth.

```
parser.on('data', line => {
  console.log(`(DATA) Received: ${line}`);
  try {
    const payload = decryptMessage(JSON.parse(line));
    if (!payload) {
      onPortResponse({success: false, error: 'Invalid credentials format'});
      return;
    }
    const {ssid, password} = payload;
    Network.connectToWifi(ssid, password, (err, stdout, stderr) => {
      if (err) {
        onPortResponse({success: false, error: err.stderr || err.message});
        return;
      }
      const output = stdout.trim();
      const error = stderr.trim();
      if (
        output.toLowerCase().includes('error') ||
        error.toLowerCase().includes('error')
      ) {
        onPortResponse({success: false, error: output || error});
        return;
      }
      if (output.toLowerCase().includes('successfully')) {
        console.log(`(SUCCESS) Connected to WiFi: ${ssid}`);
        //* Save credentials
        fs.mkdirSync('/settings', {recursive: true});
        fs.writeFileSync(
          '/settings/wifi_credentials.json',
          JSON.stringify(payload),
        );
        onPortResponse({success: true});
```

Σχήμα 6.29: Διαχείριση RFCOMM μηνυμάτων από συνδεδεμένη συσκευή.

Κατά την λήψη μηνύματος JSON απο την συνδεδεμένη Mobile συσκευή, αποκρυπτογραφείται το μήνυμα και στην συνέχεια, όπως φαίνεται στο σχήμα 6.29, δημιουργείται προσπάθεια σύνδεσης WiFi μέσω των στοιχείων που πάρθηκαν. Η υλοποίηση της κρυπτογράφησης και αποκρυπτογράφησης του μηνύματος αναλύεται στο αντίστοιχο κεφάλαιο της ασφάλειας. Η σύνδεση του WiFi είναι αφηρημένη στο συγκεκριμένο παράδειγμα και γίνεται μέσω του πακέτου nmcli, το οποίο προσφέρει εργαλεία για την διαχείριση των συνδέσεων δικτύου:

```
sudo nmcli dev wifi connect "VideoSurveillance Network" password "1234"
```

Συνεχίζοντας, κατά την επιτυχή σύνδεση, αποθηκεύονται τα στοιχεία WiFi στον δίσκο και γίνεται αποστολή απάντησης στην συσκευή, ενημερώνοντας για την επιτυχή σύνδεση. Τέλος, ενεργοποιείται η αυτόματα διαδικασία της αποστολής των στοιχείων προς τους μικροελεγκτές ESP32 και έπειτα ξεκινάει η κανονική ροή του συστήματος παρακολούθησης χώρου.

Συνοψίζοντας, η υλοποίηση του Bluetooth, αποτελεί αυτόματος τρόπος επιλογής δικτύου για την λειτουργία του συστήματος. Η αποστολή πραγματοποιείται μέσω Mobile Application και αναλύεται στο αντίστοιχο κεφάλαιο υλοποίησης της Mobile εφαρμογής.

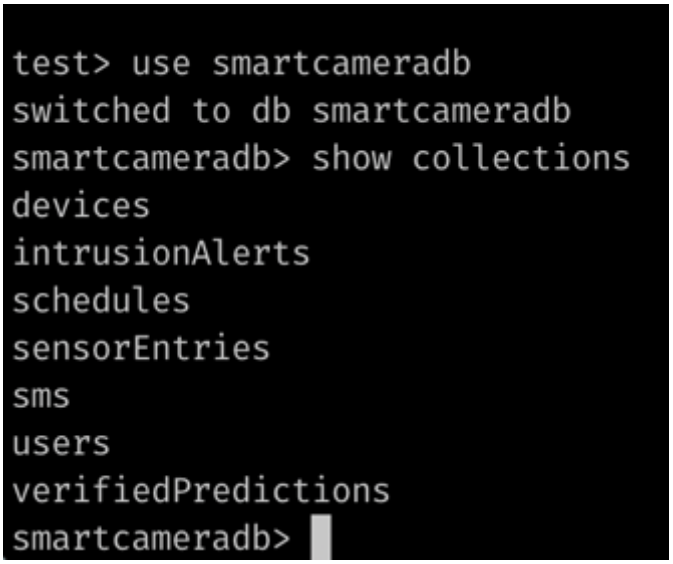
6.4.7 Σχεδιασμός βάσης δεδομένων

Σε αυτή την υποενότητα αναφέρεται η τεχνολογία πίσω από την βάση δεδομένων του συστήματος, καθώς και ο σχεδιασμός των οντοτήτων της. Αρχικά, έχει υλοποιηθεί μέσω της MongoDB, η οποία αποτελεί μία μη σχεσιακή βάση και βασίζεται στο μοντέλο Documents και Collections. Έτσι η κάθε οντότητα αποτελεί μία συλλογή (Collection) με μία λίστα από μοναδικά αρχεία (Documents). Η προσπέλαση της βάσης γίνεται μέσω του Backend και συγκεκριμένα μέσω βιβλιοθήκης στην Go. Όμως, κατά την ανάπτυξη ή για λόγους debugging, είναι δυνατή η πρόσβαση της βάσης μέσω του εργαλείου mongosh (MongoDB Shell), μέσω του τερματικού:

```
mongosh "mongodb://username:password@localhost:27017/?authSource=admin"
```

Με την χρήση της εντολής αυτής, πραγματοποιείται η σύνδεση στο Database μέσω terminal. Ο κωδικός και το όνομα χρήστη, ρυθμίζονται κατά την εγκατάσταση της βάσης. Επιπλέον, χρειάζεται η διεύθυνση του δικτύου της βάσης μαζί με την θύρα.

Μετά την επιτυχημένη σύνδεση, μπορούν να χρησιμοποιηθούν εντολές με το συντακτικό που ορίζεται από το mongosh.



Σχήμα 6.30: Αποθηκευμένες συλλογές δεδομένων στο MongoDB.

Στο σχήμα 6.30, πραγματοποιείτε αλλαγή database, σε αυτό του συστήματος, και στην συνέχεια μέσω της εντολής “show collections”, εμφανίζεται η λίστα των συλλογών που έχουν δημιουργηθεί. Συνεπώς η βάση δεδομένων του συστήματος αποτελείται από τις ακόλουθες συλλογές:

1) Users: Στοιχεία χρηστών που έχουν εγγραφεί στο σύστημα. Με κύρια προτεραιότητα τον αριθμό τηλέφωνο και τον ρόλο του χρήστη.
2) SensorEntries: Μηνύματα αισθητήρων που αποστέλλονται από τους μικροελεγκτές.

3) IntrusionAlerts: Δεδομένα σχετικά με την κάθε εισβολή. Όπως το path του βίντεο καταγραφής, την ημερομηνία συμβάντος και τις εντοπισμένες κλάσεις εισβολέων.
4) Schedules: Προγράμματα που έχουν ορισθεί από τον διαχειριστή για την συμπεριφορά του συστήματος με βάση την ώρα της ημέρας.
5) VerifiedPredictions: Επαληθευμένες προβλέψεις εισβολών στον χώρο.
6) SMS: Τα SMS που στέλνονται προς και από το σύστημα.
7) Devices: Περιέχει τα FCM Tokens, για την αποστολή Push-Notifications.

Η κάθε συλλογή αποθηκεύει αρχεία, όπου το κάθε αρχείο αποτελείται από αντικείμενα μορφής JSON. Για παράδειγμα, η προσπέλαση ενός αρχείου, γίνεται μέσω της ακόλουθης εντολής και έχει την ακόλουθη μορφή:

```
smartcameradb> db.sensorEntries.find().sort({_id:-1}).limit(2).pretty()
[
  {
    _id: ObjectId('682c3bf26df6e31cac51fee4'),
    d: Long('1747729394197'),
    s: 'ESP32-D0WD-V3-MOTION',
    t: 'sensors/motion'
  },
  {
    _id: ObjectId('682c3be36df6e31cac51fee3'),
    d: Long('1747729379960'),
    s: 'ESP32-D0WD-V3-SOUND',
    t: 'sensors/sound'
  }
]
```

Σχήμα 6.31: Προσπέλαση συλλογής MongoDB.

Παρατηρείται στο σχήμα 6.31, το διάβασμα των 2 πιο πρόσφατων αρχείων της συλλογής sensorEntries. Η έξοδος της εντολής, περιέχει τα 2 αυτά αρχεία που φαίνονται σε μορφή JSON και με 4 πεδία, ζευγαριών (key-value). Το κάθε αρχείο έχει το αναγκαστικό και μοναδικό πεδίο "_id", που χρησιμοποιείται για την αναγνώριση του.

Επιπλέον στο συγκεκριμένο παράδειγμα τα κλειδιά αποτελούνται από 1 μόνο χαρακτήρα. Το MongoDB, αποθηκεύει τα αρχεία σε μορφή BSON (Binary JSON), αν και αποτελεί αποδοτική μέθοδος αποθήκευσης, δεν παύει ο αριθμός χαρακτήρων να συνεπάγεται με τον μεγαλύτερο χώρο αποθήκευσης. Συνεπώς, στην περίπτωση που το σύστημα, σε μία μελλοντική πιθανή κατάσταση, έχει αποθηκευμένα πολλά μηνύματα αισθητήρα, τότε η αποθήκευση τους θα είναι η ελάχιστη δυνατή. Αυτό συμβαίνει διότι, αντί για τα ονόματα πεδίων "date, sensorName και topic" χρησιμοποιείται το πρώτο γράμμα "d, s και t". Αυτή η υλοποίηση βελτιώνει ελαφρώς τους χρόνους προσπέλασης και τον χώρο αποθήκευσης. Καθώς αποτελεί ελάχιστη βελτίωση, έχει υλοποιηθεί μόνο για την συγκεκριμένη συλλογή, που ενδέχεται να περιέχει τα περισσότερα δεδομένα με διαφορά, και όχι στις υπόλοιπες, καθώς η μείωση αυτή δεν αξίζει σε σχέση με αύξηση στην δυσκολία ανάγνωσης κατά την ανάπτυξη.

```go
func GetUser(uid string) (*models.UserCredentials, error) {
    res := state.UsersCollection.FindOne(context.TODO(), bson.M{"uid": uid})
    var user models.UserCredentials
    err := res.Decode(&user)
    if err != nil {
        return nil, err
    }
    return &user, nil
}
```

Σχήμα 6.32: Παράδειγμα, λήψης στοιχείων χρήση μέσω του αναγνωριστικού του.

Στο Backend, μέσω της βιβλιοθήκης "go.mongodb.org/mongo-driver", είναι δυνατή η προγραμματιστική προσπέλαση της βάσης MongoDB. Έτσι, κατά την έναρξη του συστήματος, δημιουργείται ένας Client ο οποιος συνδέεται στην βάση μέσω των στοιχείων σύνδεσης που έχουν ορισθεί. Στη συνέχεια, όπως φαίνεται στην εικόνα 6.32, μπορούν να εκτελεστούν ενέργειες, όπως η λήψη ενός Document της συλλογής Users μέσω του User ID.

Συνοψίζοντας, η λειτουργία του Backend ως Restful API, βασίζεται στην προσπέλαση του MongoDB. Είναι δυνατή μέσω του frontend, η εισαγωγή και τροποποίηση των Documents μέσω POST HTTP αιτημάτων, και μέσω GET HTTP αιτήματα, το Backend μαζί με το MongoDB παρέχει τα δεδομένα για την εμφάνιση στην γραφική διεπαφή.

6.4.8 Εφαρμογή του Docker Compose

Σε αυτή την υποενότητα, θα αναλυθεί ο ρόλος του Docker compose στο σύστημα, οι σημαντικές λειτουργίες που προσφέρει καθώς και ο τρόπος υλοποίησης. Αρχικά, όπως έχει αναφερθεί στο κεφάλαιο της αρχιτεκτονικής, το σύστημα, για τις περισσότερες υπηρεσίες, χρησιμοποιεί Docker για να τις χωρίσει σε εικονικά περιβάλλοντα. Οι υπηρεσίες MongoDB, Nginx, Mosquitto και Backend βρίσκονται στα δικά τους Docker Container όπου για την έναρξη και κλείσιμο του συστήματος πρέπει να συντονιστούν μαζί στο άνοιγμα και κλείσιμο. Επιπλέον, πρέπει κάθε φορά, στην ενεργοποίηση του κάθε Container, να παρέχονται οι σωστές παράμετροι που απαιτεί το κάθε περιβάλλον από την υλοποίηση του. Αυτή η διαδικασία, απαιτεί την ανάπτυξη ενός Script το οποίο πρέπει να διαχειρίζεται μία πολύπλοκη διαδικασία. Για την αποφυγή, αυτής της ταλαιπωρίας, υπάρχει το Docker Compose, με σκοπό την εύκολη διαχείριση πολλών Docker υπηρεσιών με ένα μόνο αρχείο - με λίγες μόνο εντολές.

Το αρχείο που διαχειρίζεται όλες τις υπηρεσιες, ονομάζεται docker-compose.yml, με ακολουθεί την μορφή:

```yaml
version: '3.9'

services:
 mongodb:
   image: mongo:8.0.6
   ports:
```

```
volumes:
environments:
devices:
depends_on:
```

Στο απόσπασμα του αρχείου, παρατηρείται η δήλωση της υπηρεσίας mongodb, με το image “mongo:8.0.6”. Η Εικόνα (Image), αποτελεί μία σειρά από εντολές για την εγκατάσταση των απαραίτητων για την δημιουργία και την ρύθμιση μιας υπηρεσίας. Στην προκειμένη περίπτωση, η εικόνα του mongo, εγκαταστεί συμβατό εικονικό λειτουργικό σύστημα Linux, arm64 στο Raspberry Pi μέσω του Docker. Επιπλέον, εγκαταστεί τα απαραίτητα για την χρήση του MongoDB και προσφέρει ευκολίες στην ρυθμιση του. Πέρα του image, που αποτελεί η βασικότερη ρύθμιση κάθε υπηρεσίας, υπάρχουν οι εξής ρυθμίσεις:

Πίνακας 6.5: Ρυθμίσεις υπηρεσίας μέσω docker-compose.yml.

Ρύθμιση	Σημασία
ports	Ορίζονται οι θύρες που θα είναι προσβάσιμες έξω από το εικονικό δίκτυο του Docker Compose.
volumes	Ορίζονται τα αρχεία που θα παραμένουν στον δίσκο.
environments	Ορίζονται οι μεταβλητές περιβάλλοντος.
devices	Δίνεται πρόσβαση στα αρχεία συσκευών του λειτουργικού.
depends_on	Ορίζονται οι υπηρεσίες του Docker Compose δικτύου στις οποίες βασίζεται η συγκεκριμένη υπηρεσία.

```
mongodb:
  image: mongo:8.0.6
  ports:
    - '27017:27017'
  environment:
    - MONGO_INITDB_ROOT_USERNAME=tmazarakis
    - MONGO_INITDB_ROOT_PASSWORD=IHCk1KZHfKhvSqo
  volumes:
    - mongo_data:/data/db
    - ./mongodb:/docker-entrypoint-initdb.d
```

Σχήμα 6.33: Ρύθμιση MongoDB μέσω Docker Compose.

Στην εικόνα 6.33, παρατηρείται η ρύθμιση του Docker Container mongodb. Αξίζει να σημειωθεί πως, το Docker Compose, δημιουργεί ένα εικονικό δίκτυο με τις υπηρεσίες που έχει δηλωμένες στο αρχείο ρυθμίσεων. Έτσι για την εξαγωγή θυρών προς το υπόλοιπο δίκτυο, ακόμα και στο ίδιο το μηχάνημα, είναι αναγκαία η δήλωση του port, όπως φαίνεται στην εικόνα. Επιπλέον, ορίζονται το όνομα και ο κωδικός της βάσης, ως μεταβλητές περιβάλλοντος, τα στοιχεία αυτά χρησιμοποιούνται από το Image mongo:8.0.6, κατά την δημιουργία της βάσης. Τέλος μέσω των volumes, ρυθμίζονται τα δεδομένα να παραμένουν αποθηκευμένα μετά από επανεκκίνηση της υπηρεσίας και γίνεται κοινοποίηση των

περιεχομένων του φακέλου mongodb στο ορισμένο entry point της βάσης. Ο φάκελος περιέχει ένα αρχείο, που με την σειρά του, δημιουργεί τα collections τα οποία αναφέρθηκαν στο κεφάλαιο του MongoDB.

Με παρόμοιο τρόπο, γίνεται η ρύθμιση των υπόλοιπων 3 υπηρεσιών. Οι υπηρεσίες αυτές, μπορούν να επικοινωνήσουν μεταξύ τους, δηλαδή μέσα στο εικονικό δίκτυο του Docker Compose, μέσω του ονόματος της κάθε υπηρεσίας ως διεύθυνση. Δηλαδή, ο Nginx έχει πρόσβαση στο Backend, συγκεκριμένα στην θύρα 3000, μέσω του “http://backend:3000”, αν πρόκειται για HTTP αίτημα.

Συνεπώς, μετά την δημιουργία και ρύθμιση του Docker Compose, μέσω του αρχείου, απομένει η έναρξη του συστήματος.

```
tmazarakis@tRPIa:~/diploma $ sudo docker-compose up
Starting diploma_mongodb_1    ... done
Starting diploma_mosquitto_1  ... done
Starting diploma_backend_1    ... done
Starting diploma_nginx_1      ... done
Attaching to diploma_mosquitto_1, diploma_mongodb_1, diploma_backend_1, diploma_nginx_1
backend_1     | Starting servers
backend_1     | Getting and caching settings
backend_1     | Settings loaded successfully
backend_1     | Settings loaded successfully
backend_1     | Settings are already set to the same value
backend_1     | MQTT-WS Server not initialized, ignoring publish
backend_1     | Starting database initialization
mongodb_1     | {"t":{"$date":"2025-05-23T16:24:39.810+03:00"},"s":"I",  "c":"CONTROL",  "id"
matically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}
mosquitto_1   | chown: /mosquitto/config/mosquitto.conf: Read-only file system
```

Σχήμα 6.34: Έναρξη Docker Compose.

Στην εικόνα 6.34, παρατηρείται, η εντολή για την έναρξη του Docker Compose. Στην συνέχεια, μέσω των logs εμφανίζεται η έναρξη της κάθε υπηρεσίας και τα logs της κάθε υπηρεσίας συσσωρευμένα.

Συνοψίζοντας, το Docker Compose, χρησιμοποιεί τα ήδη υπάρχων εργαλεία που προσφέρει το Docker, για την αυτοματοποίηση και την απλοποίηση της διαδικασίας διαχείρισης πολλών υπηρεσιών. Προσφέρει ένα απλό και πλήρες αρχείο ρυθμίσεων, δημιουργεί αυτόματα ένα εικονικό δίκτυο για τις υπηρεσίες και προσφέρει εντολές για την διαχείριση των υπηρεσιών μαζικά. Αποτελεί ένα θεμελιώδη εργαλείο σε συνδυασμό με το Docker, ειδικά στην περίπτωση, όπου χρειάζεται συνδυασμός πολλών υπηρεσιών.

6.5 Υλοποίηση Web ιστοσελίδας

Αυτό το κεφάλαιο, απευθύνεται στην υλοποίηση της ιστοσελίδας. Η ιστοσελίδα, αποτελεί τον τρόπο διαχείρισης του συστήματος από τον διαχειριστή. Οι κυριότερες δυνατότητες του διαχειριστή μέσω του Web Application είναι:

1) Παρακολούθηση και διαχείριση του Live Streaming.
2) Ζωντανή ενημέρωση συμβάντων που λαμβάνει το σύστημα.
3) Ζωντανή ενημέρωση της κατάστασης όλων των λειτουργιών του συστήματος.
4) Δυνατότητα ρύθμισης των λειτουργιών του συστήματος.
5) Εφαρμογή προγράμματος χρόνου, για την αυτόματη λειτουργία του συστήματος.

6) Παρακολούθηση καταγραφών βίντεο.

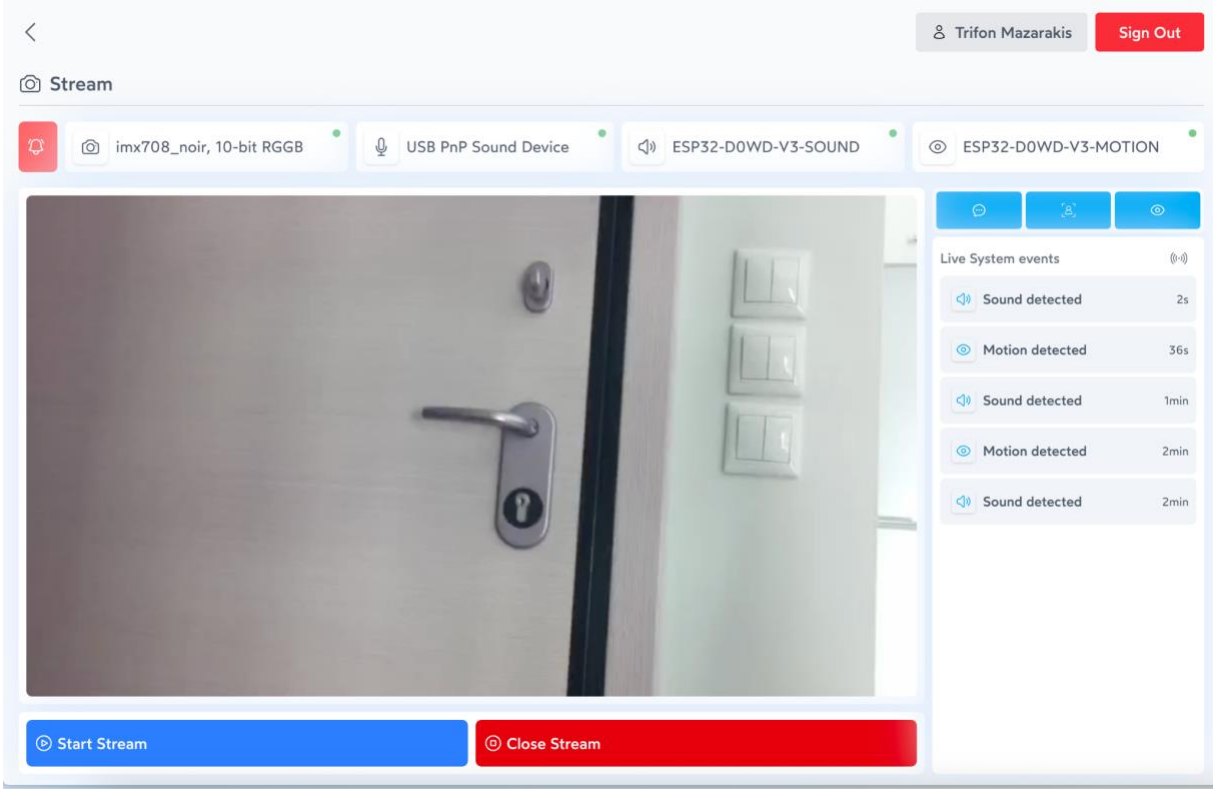
7) Διαχείριση προσώπων, για το μοντέλο αναγνώρισης προσώπου.

Η ιστοσελίδα, είναι υλοποιημένη με την χρήση της React. Επιπλέον, χρησιμοποιούνται για την βελτίωση της απόδοσης και της εμπειρίας ανάπτυξης. οι τεχνολογίες Tailwind CSS, Vite και TypeScript. Ως αποτέλεσμα, δημιουργούνται 3 αρχείων, HTML, CSS και JavaScript τα οποία αποτελούν τα ελάχιστα απαραίτητα για την λειτουργία της ιστοσελίδας και την μείωση των πιθανών σφαλμάτων.

Η υλοποίηση της ιστοσελίδας, αποτελείται, αρχικά από την δομή, που θα συζητηθούν οι διάφορες σελίδες και οι ενέργειες τους. Ο τρόπος upload προσώπων για εκπαίδευση από το μοντέλο αναγνώρισης προσώπων. Η ανάπτυξη Video Player για την παρακολούθηση του Live Streaming από τον διαχειριστή. Τέλος, θα αναφερθεί ο τρόπος επικοινωνίας από το Backend, για την δημιουργία μιας live επικοινωνίας. Όλα, αυτά είναι το σημείο συζήτησης των επόμενων ενοτήτων.

6.5.1 Δομή ιστοσελίδας

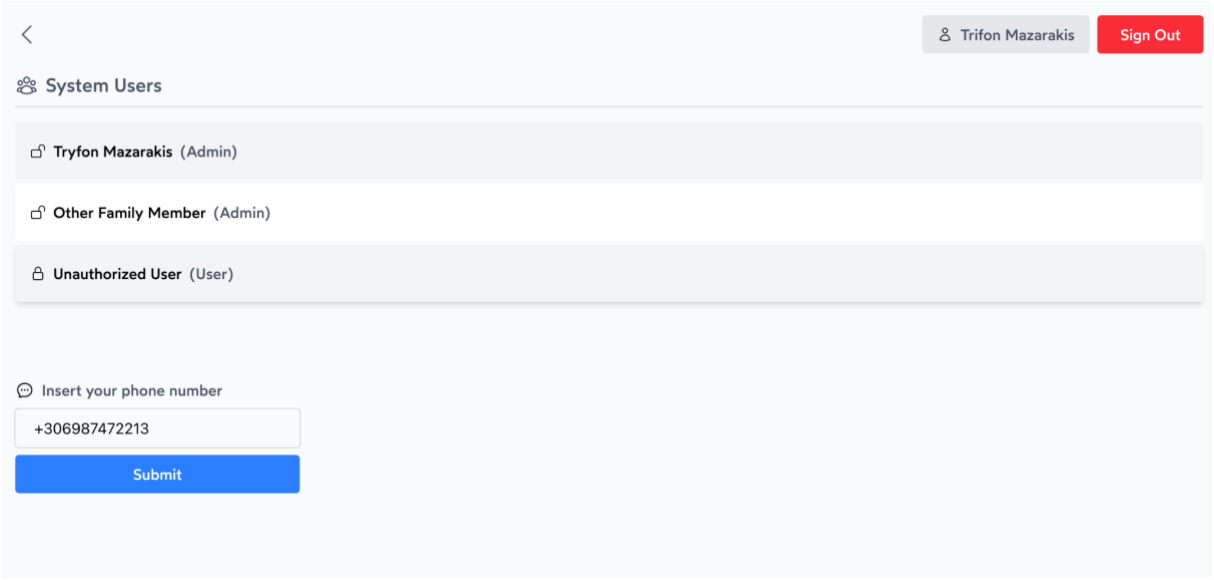
Η δομή της ιστοσελίδας, αποτελείται από 6 κύριες σελίδες. Η πρώτη και βασικότερη, το Stream Page, όπου ο διαχειριστής ενδεχομένως θα περάσει τον περισσότερο του χρόνο. Στην συνέχεια, παρατηρείται η γραφική διεπαφή της σελίδας και θα αναλυθούν οι λειτουργίες της.



Σχήμα 6.35: Σελίδα του Live Stream.

Στην εικόνα 6.35, εμφανίζεται η γραφική διεπαφή της ιστοσελίδας, στην οποία έχει πρόσβαση ο διαχειριστής του συστήματος. Στην πρώτη όψη, φαίνεται η ζωντανή ροή καθώς και 2 κουμπιά για την απενεργοποίηση και ενεργοποίηση του. Στο πάνω μέρος, ενημερώνεται ο διαχειριστής ζωντανά, για τις διαθέσιμες συσκευές, μαζί με τα ονόματα τους. Επιπλέον, στο δεξιό μέρος, υπάρχουν τα Live System Events, όπου ο χρήστης ενημερώνεται για τα συμβάντα του συστήματος μέσω σύνδεσης WebSockets

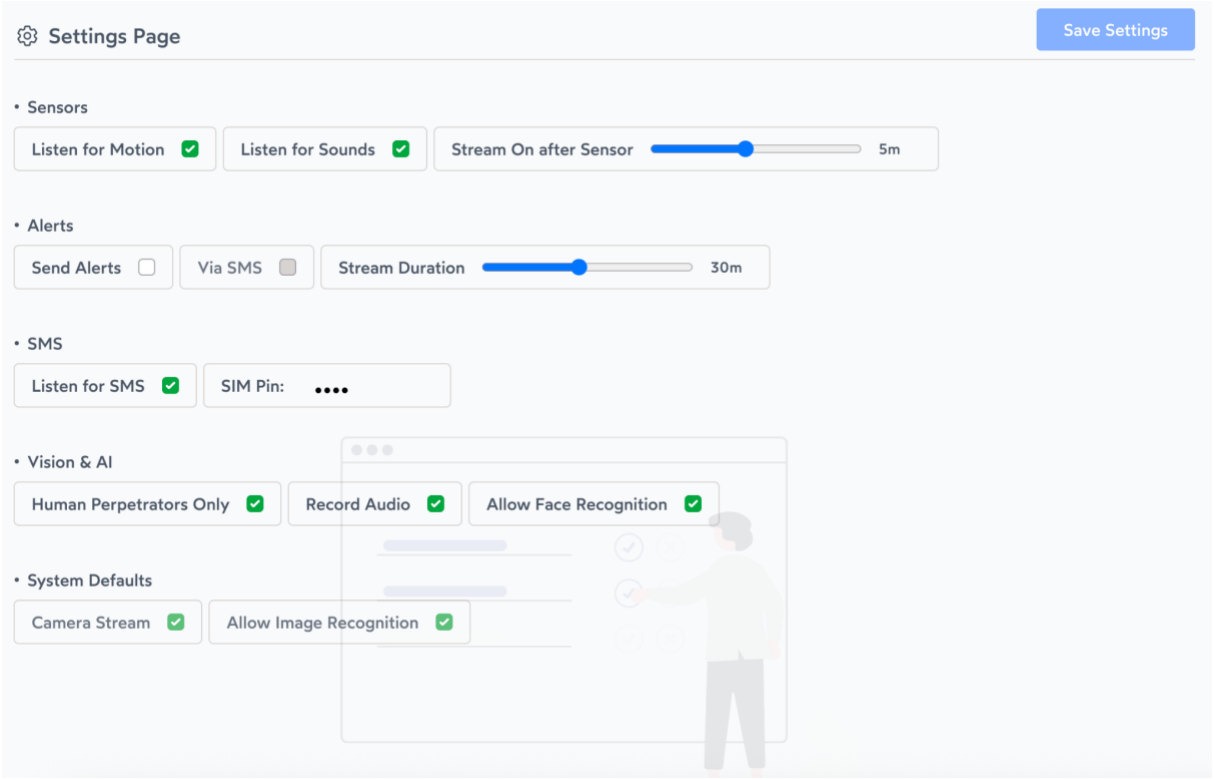
με τον διακομιστή. Πιο συγκεκριμένα, στην λίστα αυτή, γίνεται λήψη μηνυμάτων, σε περίπτωση συναγερμού, σε περίπτωση πυροδότησης των αισθητήρων, σε περίπτωση επαληθευμενης αναγνώρισης και σε περίπτωση λήψης SMS. Τέλος, στο τέρμα πάνω μέρος, κάθε σελίδας, εμφανίζεται το όνομα του χρήση καθώς και η επιλογή σύνδεσης και αποσύνδεσης. Ο χρήστης μπορεί να επιλέξει το όνομα του και να μεταφερθεί στην σελίδα διαχείρισης των χρηστών.



Σχήμα 6.36: Σελίδα διαχείρισης των System Users.

Στην σελίδα, που φαίνεται στην εικόνα 6.26, ο διαχειριστής μπορεί να δει τους εγγεγραμμένους χρήστες και να τροποποιήσει τον ρόλο τους. Καθώς, η εγγραφή και σύνδεση στο σύστημα είναι διαθέσιμη από όλους, έχει υλοποιηθεί η λειτουργία ρόλων. Οι ρόλου, αποτελούνται από τον κανονικό χρήση και τον διαχειριστή. Ο κανονικός χρήστης, δεν έχει πρόσβαση στο σύστημα, και περιμένει στον διαχειριστή να του δώσει, μέσω αυτής της σελίδας. Επιπλέον, ο κάθε διαχειριστής, μπορεί να προσθέσει τον αριθμό τηλεφώνου του, έτσι ώστε να λαμβάνει ειδοποιήσεις μέσω SMS.

Ο χρήστης που συνδέεται πρώτος στο σύστημα, γίνεται αυτόματα και ο πρώτος διαχειριστής. Μετά την ρύθμιση του αριθμού τηλεφώνου, απομένει η ρύθμιση του συστήματος. Ο τρόπος θα αναφερθεί στην συνέχεια.

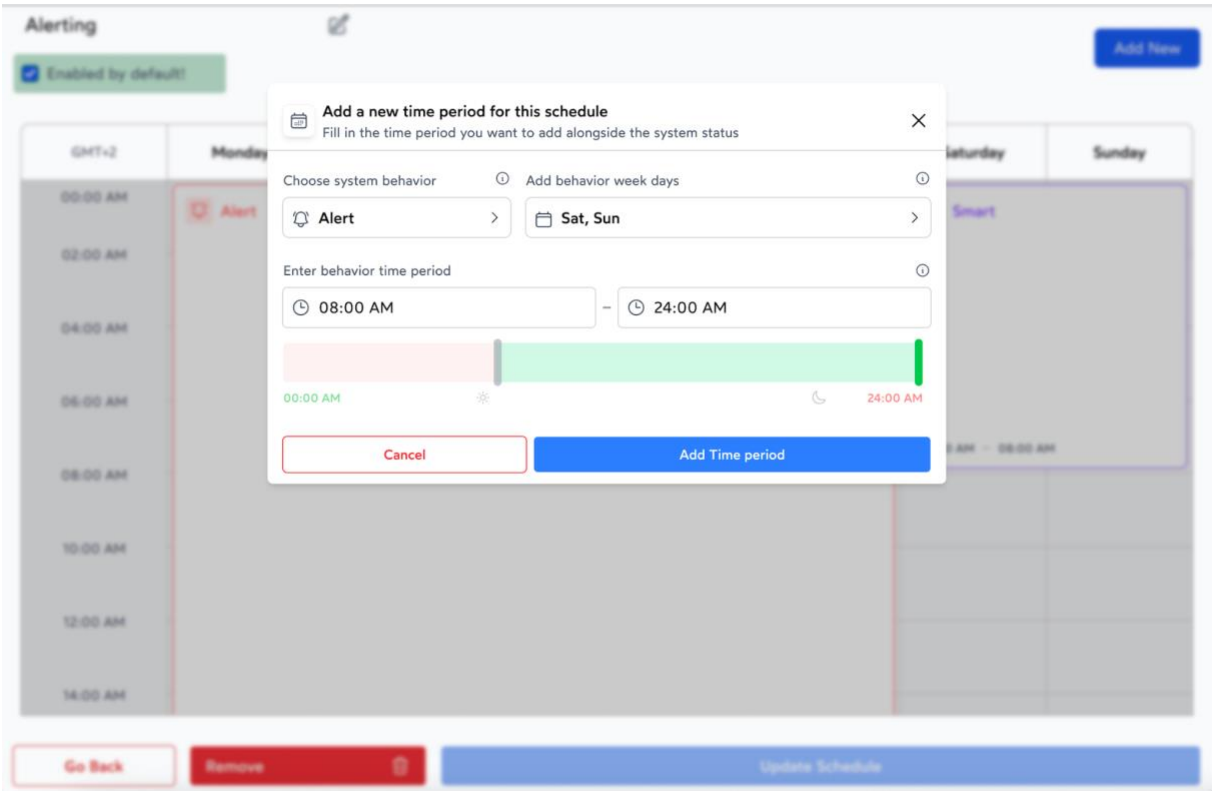


Σχήμα 6.37: Σελίδα ρύθμισης του συστήματος.

Η σελίδα των ρυθμίσεων, όπως παρατηρείται στο σχήμα 6.37, περιέχει ρυθμίσεις για τον τρόπο λειτουργίας του συστήματος. Ο διαχειριστής μπορεί να ρυθμίσει την συμπεριφορά του συστήματος όσον αφορά τους αισθητήρες, τις ειδοποιήσεις και την έξυπνη αναγνώριση. Συγκεκριμένα:

1) Listen for Motion: Ενεργοποίηση του συστήματος μέσω πυροδότησης αισθητήρα κίνησης.
2) Listen for Sounds: Ενεργοποίηση του συστήματος μέσω πυροδότησης αισθητήρα ήχου.
3) Stream on After Sensor: Ρυθμίζεται ο χρόνος που μένει ενεργό το σύστημα, μετά την έναρξη από μήνυμα αισθητήρα.
4) Send Alerts: Ενεργοποίηση Push-Notifications μέσω Mobile application.
5) Via SMS: Ενεργοποίηση λήψης SMS μηνυμάτων ως ειδοποίηση.
6) Stream Duration: Η διάρκεια καταγραφής και της ζωντανής ροής, στην περίπτωση συναγερμού.
7) Listen For SMS: Δυνατότητα έναρξης και κλείσιμο της ζωντανής ροής μέσω SMS από τον διαχειριστή.
8) SMS Pin: Το Pin της SMS, του GSM Module, στην περίπτωση που είναι ενεργοποιημένη η ρύθμιση PIN.
9) Human Perpetrators Only: Πυροδότηση του συναγερμού, μόνο στην περίπτωση αναγνώρισης ανθρώπου.
10) Record Audio: Χρήση του μικροφώνου για την ενσωμάτωση ήχου και αναγνώριση θορύβων.
11) Allow Face Recognition: Χρήση αναγνώρισης προσώπων. Στην περίπτωση που αναγνωριστεί ένας άνθρωπος και είναι αποθηκευμένο το πρόσωπο του, ειδοποιείται για την αναγνώριση του, χωρίς την έναρξη συναγερμού.

Επιπλέον, στο κάτω μέρος της σελίδας υπάρχουν οι ρυθμίσεις κάμερας, που είναι ενεργοποιημένες και χωρίς την δυνατότητα απενεργοποίησης, καθώς αποτελούν οι βασικές λειτουργίες του συστήματος.

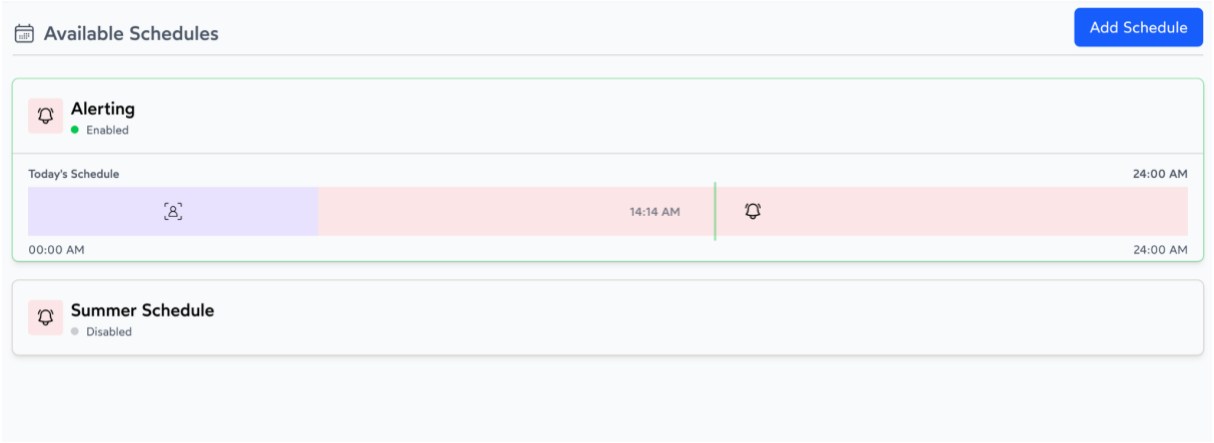


Σχήμα 6.38: Σελίδα προσθήκης προγράμματος χρόνου.

Τελευταία ρύθμιση, για την ορθή λειτουργία του συστήματος, είναι η προσθήκη προγράμματος, το οποίο καθορίζει την κατάσταση της λειτουργίας του συστήματος με βάση την χρονική στιγμή. Υπάρχουν 4 διαφορετικές καταστάσεις που μπορεί να τεθεί το σύστημα με βάση τον χρόνο και την ημέρα της εβδομάδας, όπως φαίνεται στην εικόνα 6.38. Οι 4 καταστάσεις αποτελούν:

1) Offline: Το σύστημα είναι απενεργοποιημένο.
2) Watch. Η ζωντανή ροή είναι ενεργή, οι υπόλοιπες λειτουργίες, όπως αισθητήρες και έξυπνη αναγνώριση, είναι ανενεργές.
3) Smart. Όλες οι λειτουργίες είναι ενεργοποιημένες. Η ζωντανή ροή είναι πάντα ανοιχτή.
4) Alert. Το σύστημα είναι έτοιμο, στην περίπτωση πυροδότησης αισθητήρα, ενεργοποιούνται όλες οι λειτουργίες.

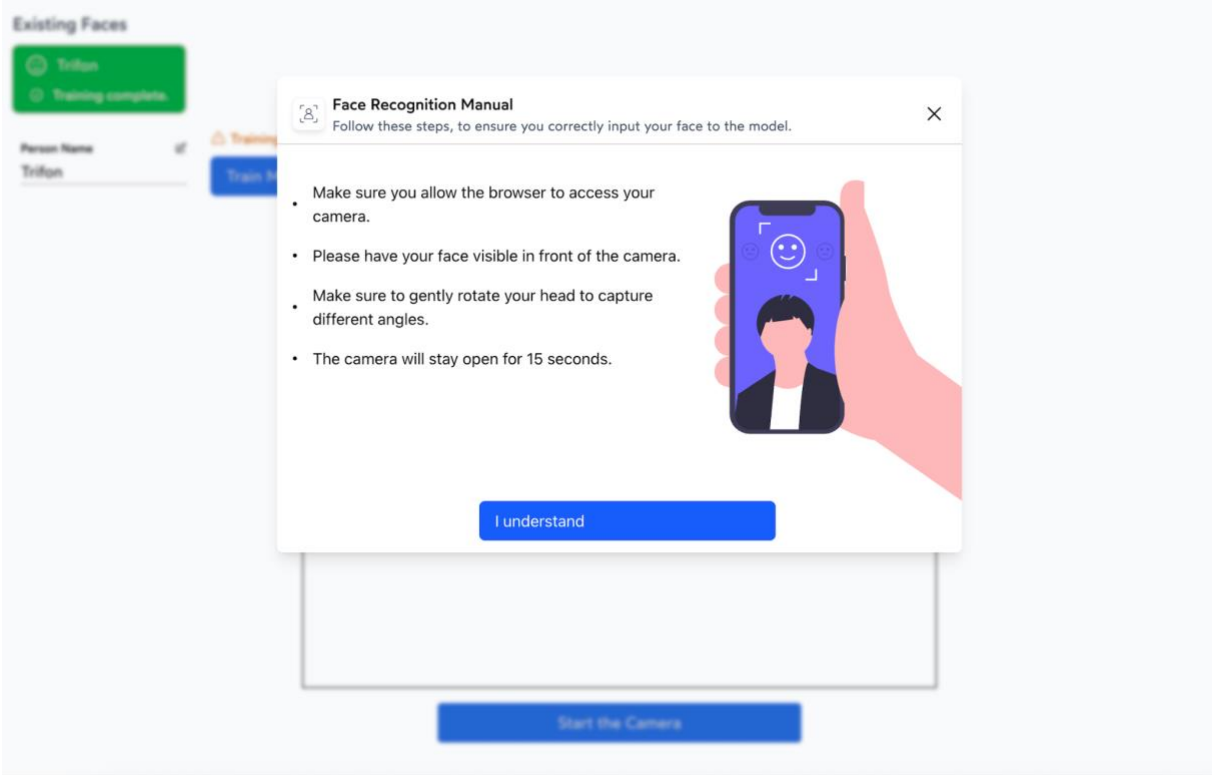
Επιπλέον, ο διαχειριστής, μπορεί να δημιουργήσει πολλά προγράμματα και να επιλέξει την ενεργοποίηση του ενός ανάλογα την εποχή και περίσταση.



Σχήμα 6.39: Διαθέσιμα προγράμματα του συστήματος.

Στην σελίδα, που φαίνεται στο σχήμα 6.39, ο διαχειριστής μπορεί να δει τα προγράμματα που έχει δημιουργήσει. Επιπλέον μπορεί να δει την κατάσταση του ενεργοποιημένου προγράμματος καθώς και τις προηγούμενες και επόμενες καταστάσεις. Τέλος, μπορεί να τροποποιήσει και να διαγράψει το κάθε πρόγραμματα.

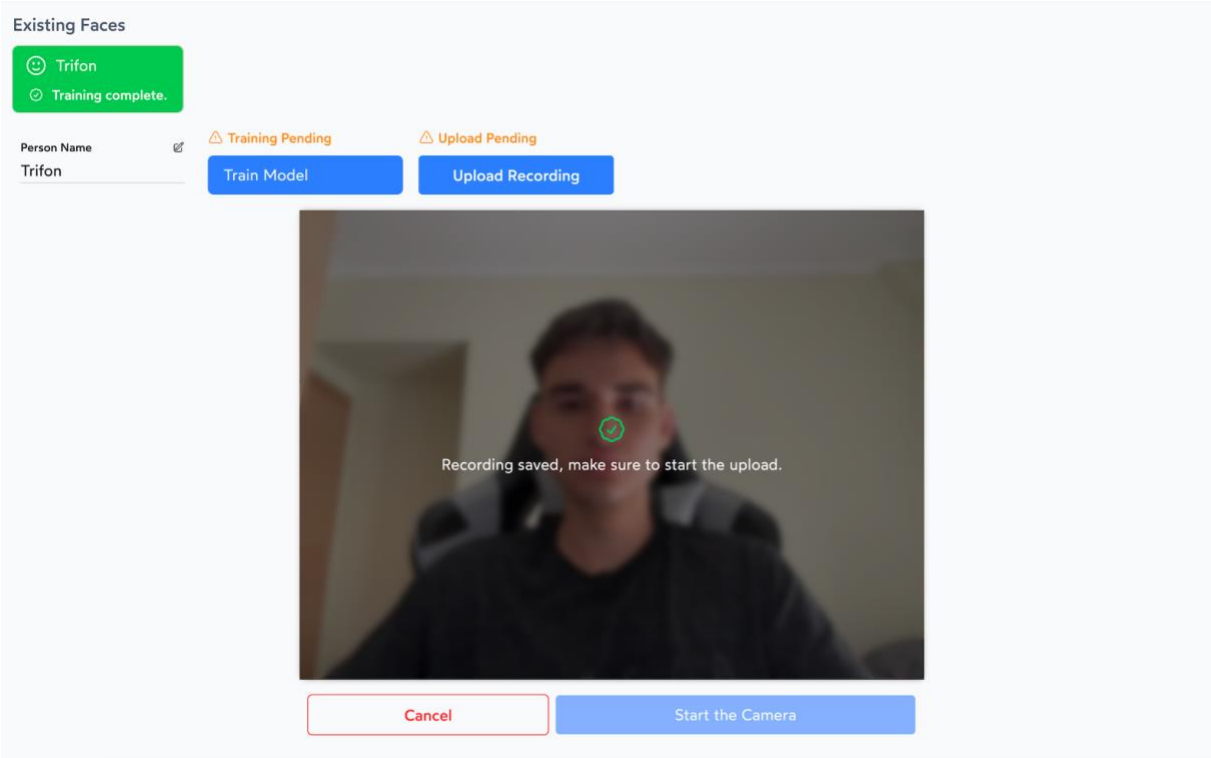
6.5.2 Upload προσώπων για εκπαίδευση



Σχήμα 6.40: Σελίδα διαχείρισης και upload προσώπων.

Ο διαχειριστής, έχει την δυνατότητα προσθήκης προσώπων. Τα πρόσωπα αυτά, χρησιμοποιούνται από το μοντέλο αναγνώρισης προσώπων, ArcFace, για να καταλάβει αν ο εισβολέας είναι γνωστό άτομο, π.χ. ο διαχειριστής. Έτσι, αυτή η σελίδα αποτελεί τρόπος διαχείρισης των προσώπων αυτών.

Πιο συγκεκριμένα, γίνεται εμφάνιση των εκπαιδευμένων προσώπων και είναι δυνατή η τροποποίηση, διαγραφή και η προσθήκη νέων προσώπων. Κατά την επιλογή προσθήκης νέου προσώπου, εμφανίζεται το Popup, σχήμα 6.40, το οποίο ενημερώνει τον χρήστη για την διαδικασία.



Σχήμα 6.41: Προσθήκη προσώπου για εκπαίδευση.

Κατά την διαδικασία εκπαίδευσης, με το πατημα στο κουμπί “Start the Camera”, ανοίγει η κάμερα του χρήση για 15 δευτερόλεπτα. Στο τέλος, ο διαχειριστής, επιλέγει να ανεβάσει την καταγραφή αυτή, ή να την ακυρώσει. Στην συνέχεια, το Backend, λαμβάνει το video, το χωρίζει σε εικόνες και μέσω του Insightface, δημιουργεί το νέο embedding προσώπου. Στην συνέχεια, θα αναλυθεί η υλοποίηση της διαδικασίας από μεριά Web και διακομιστή.

```
//* Initialize stream
recordedChunks.current = [];
recorder.current = null;
const stream = await navigator.mediaDevices.getUserMedia({
  video: {
    width: {ideal: sizes.width},
    height: {ideal: sizes.height},
    frameRate: {ideal: 20},
  },
  audio: false,
});

//* Play stream
videoRef.current.srcObject = stream;
await videoRef.current.play();

//* Record stream
recorder.current = new MediaRecorder(stream);
await startRecording();
```

Σχήμα 6.42: Άνοιγμα κάμερας από την ιστοσελίδα.

Στο σχήμα 6.42, παρατηρείται απόσπασμα, από την δημιουργία της ροής video στο Web. Η δυνατότητα αυτή, γίνεται μέσω του MediaStream API, το οποίο παρέχει μεθόδους ρύθμισης των συσκευών πολυμέσων, όπως κάμερα και μικρόφωνο, και δυνατότητες έναρξης και κλείσιμο των ροών. Επιπλέον, η καταγραφή της ροής γίνεται μέσω του MediaRecorder API, μέσω του οποίου γίνεται έναρξη της καταγραφής της ροής και προγραμματιστικά μπορούν να παρθούν τα δεδομένα. Ο τρόπος αυτός, παρατηρείται στην επόμενο εικόνα.

```
recorder.current.start();
recorder.current.onerror = e => {
  console.log('error recording stream', e);
  cleanup();
};
recorder.current.ondataavailable = e => {
  if (e.data.size > 0) {
    recordedChunks.current.push(e.data);
  }
};
onStopRef.current = async () => {
  //* Handle if stream was cancelled
  if (cancelRef.current) {
    setVideoBlob(null);
    setCameraOpen(false);
  } else {
    const videoBlob = new Blob(recordedChunks.current, {
      type: recorder.current?.mimeType ?? 'video/mp4',
    });
    const url = URL.createObjectURL(videoBlob);
    console.log(url);
    setVideoBlob(videoBlob);
    generateFrame(videoBlob);
    setCameraOpen(false);
  }
```

Σχήμα 6.43: Έναρξη και διαχείριση δεδομένων καταγραφής της κάμερας.

Στην συνέχεια, μετά την έναρξη της κάμερας, αρχίζει η καταγραφή και αποθηκεύονται τα δεδομένα της ροής στην μνήμη του προγράμματος. Στο τέλος, μετά από 15 δευτερόλεπτα, σταματάει η καταγραφή και η κλείνει η κάμερα και δημιουργείται το Blob (Binary Large Object) του βίντεο. Στην συνέχεια, κατόπιν εντολής του χρήστη, γίνεται η αποστολή του βίντεο στο Backend μέσω HTTP με body ως φόρμα.

Τέλος, αξίζει να σημειωθεί πως τα API, που χρησιμοποιήθηκαν, MediaRecorder και MediaStream, αποτελούν πρότυπο Web, και είναι διαθέσιμα με παρόμοια χρήση σε όλους τους κύριους Browsers.

6.5.3 Ανάπτυξη Video Player

Ο Video Player, είναι υπεύθυνος για την εμφάνιση της ζωντανής ροής στον χρήστη, καθώς και την παροχή εργαλείων για την διαχείριση της, από τον χρήστη. Η ανάπτυξη του, περιλαμβάνει την χρήση της βιβλιοθήκης HLS για το περιβάλλον JavaScript. Αυτό γίνεται, καθώς Browsers εκτός του Safari, δεν υποστηρίζουν HLS Streaming και χρειάζεται η χρήση βιβλιοθηκών Player.

Αρχικά, χρησιμοποιείται ένα από HTML Video Element, το οποίο θα εμφανίζει την ζωντανή ροή ως βίντεο στον Browser. Καθώς, το HLS, δεν υποστηρίζεται στους περισσότερους Browsers, η χρήση του συνδέσμου της ζωντανής ροής απευθείας στο Video Element δεν θα δουλέψει, γιατί δεν μπορεί να επεξεργαστεί τα δεδομένα της συγκεκριμένης ροής. Στο σημείο αυτό, χρησιμοποιείται η βιβλιοθήκη HLS.js, η οποία διαβάζει την ροή και την μεταφράζει, σε μορφή συμβατή με το στοιχείο Video που έχει δημιουργηθεί:

```
const hls = new HLS();
hls.loadSource("https://trpia.local/hls/stream.m3u8");
hls.attachMedia(videoElement);
```

Για την ορθή διαχείριση του Live Stream, μπορούν να ρυθμιστούν διάφορες παράμετροι του HLS Stream. Παρακάτω, στον πίνακα 6.6, φαίνονται αναλυτικά οι ρυθμίσεις HLS που έχουν εφαρμοστεί:

Πίνακας 6.6: Ρυθμίσεις HLS Player.

Ρύθμιση	Τιμή	Σημασία
maxBufferLength	20	Τα δευτερόλεπτα Stream που αποθηκεύονται, από την τωρινή στιγμή και έπειτα.
maxLiveSyncPlaybackRate	1.15	Η ταχύτητα του βίντεο, στην περίπτωση που μείνει πίσω από την ζωντανή ροή.
LiveDurationInfinity	true	Ορίζεται η διάρκεια του Live ως άπειρη.
LiveSyncDurationCount	4	Ο αριθμός των Segments, που είναι πίσω ο Player. Δηλαδή η καθυστέρηση του Live Stream. Το κάθε Segment είναι 1 δευτερόλεπτο ροής, άρα 4 δευτερόλεπτα καθυστέρηση. Χρήσιμο για την ομαλότερη παρακολούθηση της ροής.
backBufferLength	20	Τα δευτερόλεπτα Stream που αποθηκεύονται, για την περιήγηση προς τα πίσω.
enableWorker	true	Δημιουργείται ένας Worker, για την παράλληλη επεξεργασία της HLS ροής.
startFragPrefetch	true	Αποθηκεύει ένα τμήμα του Live προτού ξεκινήσει ο Player. Χρησιμεύει στην γρηγορότερη έναρξη του Player.

Η ρύθμιση του HLS, αποτελεί μία άκρως σημαντική λειτουργία, για την καλή εμπειρία του χρήστη. Οι ρυθμίσεις, πρέπει να αλλάζουν ανάλογα με την συσκευή του χρήστη και την ταχύτητα σύνδεσης του. Επιπλέον, στην περίπτωση σφάλματος, έχει υλοποιηθεί αυτόματα ανανέωση της ροής.

```
/**
 * * Handle errors
 */
hlsRef.current.on(Hls.Events.ERROR, function (_event, data) {
  if (
    data.type === Hls.ErrorTypes.NETWORK_ERROR &&
    data.response &&
    data.response.code === 404 &&
    useSystemStatus.getState().services?.cameraStream === 'operating'
  ) {
    console.log('Network error detected, trying to reload hls');
    loadRef.current();
  } else if (data.type === Hls.ErrorTypes.MEDIA_ERROR) {
    console.log('Media error detected, trying to recover');
    hlsRef.current?.recoverMediaError();
  }
});
```

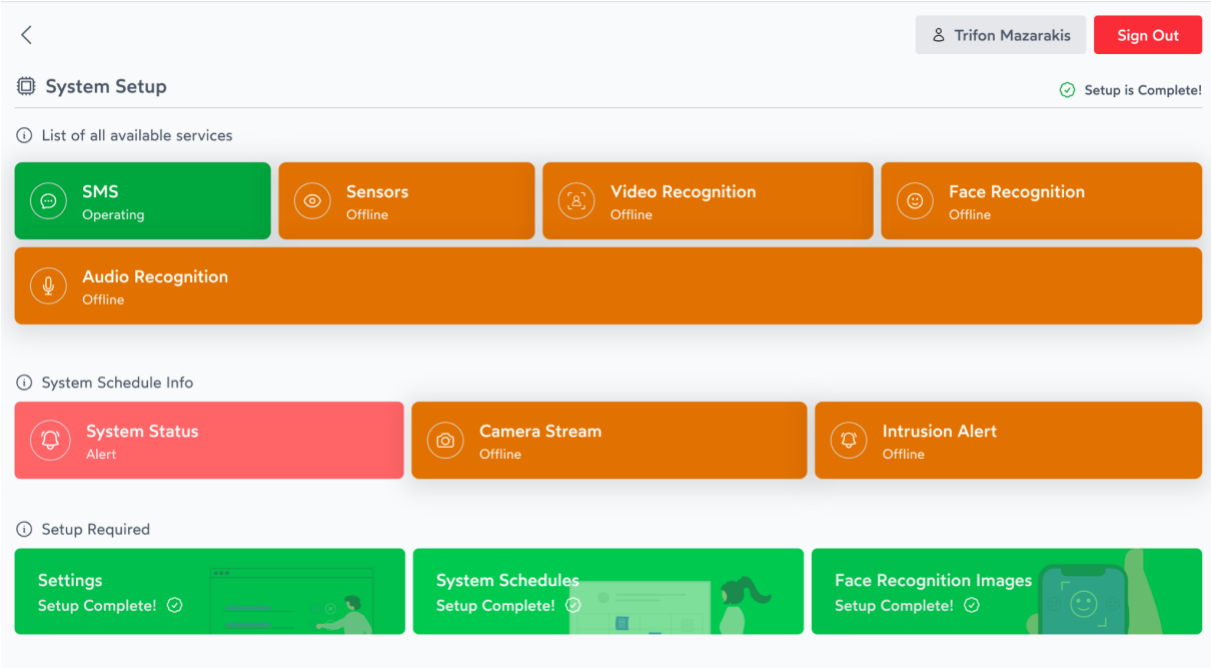
Σχήμα 6.44: Διαχείριση σφαλμάτων ζωντανής ροής.

Στο σχήμα 6.44, φαίνεται η διαχείριση σφαλμάτων του Live Stream. Συγκεκριμένα, κατά την λήψη σφάλματος, ανανεώνεται η ροή μέσω της μεθόδου recoverMediaError. Συνήθως, η πρακτική ανανέωσης της σελίδας στην περίπτωση που μία ζωντανή ροή έχει καθυστερήσεις, λύνει το πρόβλημα. Έτσι, αυτή η μέθοδος, αποτελεί μία αυτόματη και πιο αποδοτική λύση αυτής της διαδικασίας που γίνεται από τους χρήστες, καθώς ανανεώνει μόνο την ροή, μέχρι να βελτιωθεί.

6.5.4 Επικοινωνία με Backend

Η ιστοσελίδα, για την λήψη δεδομένων, στέλνει αιτήματα HTTP προς το Backend, που λειτουργεί ως Restful API. Κατά την φόρτωση της κάθε σελίδας, γίνεται αποστολή του αιτήματος, HTTP Request, για τα αντίστοιχα δεδομένα που ζητούνται και στέλνεται HTTP Response από το Backend που περιέχει σε μορφή JSON, τα δεδομένα.

Για την δημιουργία μιας άμεσης και ζωντανής ενημέρωσης της κατάστασης του συστήματος, έχει υλοποιηθεί WebSocket σύνδεση μεταξύ Web client και Backend server.



Σχήμα 6.45: Σελίδα ολικής κατάστασης των λειτουργιών του συστήματος.

Όπως φαίνεται στο σχήμα 6.45, ο διαχειριστής έχει στην διάθεση του την σελίδα System Setup. Η σελίδα αυτή, αποτελεί “lobby” για την δρομολόγηση σε όλες τις σελίδες του Web application και ενημερώνει τον χρήστη για την κατάσταση όλων των λειτουργιών. Η λήψη της κατάστασης γίνεται ζωντανά, μέσω της δημιουργίας WebSocket connection. Συνεπώς, ο διαχειριστής, με καθυστέρηση μερικών εκατοντάδων ms, ενημερώνεται για την κατάσταση της κάθε συσκευής και υπηρεσίας του συστήματος. Επιπλέον, ίδια λογική, λαμβάνει χώρα στην σελίδα του Live Stream, στην λίστα των Live System Events, το οποίο ανανεώνεται ζωντανά σε περίπτωση κάποιου συμβάντος.

```
const clientRef = useRef<MqttClient | null>(null);
useEffect(() => {
  if (clientRef.current) {
    return;
  }
  clientRef.current = mqtt.connect(
    'wss://trpia.local/ws/',
    {
      username: import.meta.env.VITE_MOSQUITTO_USERNAME,
      password: import.meta.env.VITE_MOSQUITTO_PASSWORD,
      clean: true,
    },
  );
  const handleMessage = (_topic: string, messageStr: Buffer) => {
    let message = {};
    console.log('message received: ', messageStr.toString());
    try {
      message = JSON.parse(messageStr.toString());
    } catch (err) {
      console.log('Invalid message: ', messageStr.toString(), err);
    }
    const topic: TopicsType[number] = _topic as TopicsType[number];
    if (isServerMessage(message)) {
      switch (topic) {
        case 'messages/sms':
          console.log('SMS: ', message.toString());
          onMesssageReceive(message);
          break;
        case 'messages/prediction':
          console.log('Recognition: ', message.toString());
          onMesssageReceive(message);
          break;
        case 'messages/sensors':
          console.log('Sensors: ', message.toString());
          onMesssageReceive(message);
          break;
```

Σχήμα 6.46: Απόσπασμα υλοποίησης WebSocket Web Client.

Στην εικόνα 6.46, παρουσιάζεται απόσπασμα της υλοποίησης WebSocket Client στην πλευρά της ιστοσελίδας. Αρχικά, η σύνδεση γίνεται μέσω του Mosquitto broker και χρησιμοποιείται τεχνική WebSockets Over MQTT. Αυτό συμβαίνει, καθώς οι Browser δεν υποστηρίζουν συνδέσεις MQTT αλλά είναι πλήρως συμβατοί με συνδέσεις WebSockets. Έτσι, μέσω του ασφαλούς πρωτοκόλλου WSS (WebSocketsSecure) πραγματοποιείται η επικοινωνία με τον Nginx server, ο οποίος λειτουργώντας ως Reverse Proxy, δρομολογεί το αίτημα και μεταφράζεται ως MQTT στον διακομιστή Mosquitto. Η υλοποίηση αυτή, έχει επιλεχθεί, για να εκμεταλλευτεί η λογική συνδρομής σε θέματα (Topic Subscription) που παρέχεται αυτομάτως από το MQTT.

Συνεπώς, όπως φαίνεται στο σχήμα, μετά την πραγματοποίηση σύνδεσης, και μετά την συνδρομή στα topics, το πρόγραμμα περιμένει την λήψη μηνυμάτων για το κάθε θέμα που έχει εγγραφεί. Στην συνέχεια, τα μηνύματα περνάνε ως παράμετροι στην μέθοδο onMessageReceive, η οποία τα αποθηκεύει στην μνήμη και ενημερώνεται μέσω της React, δυναμικά η ιστοσελίδα στα μέρη όπου χρειάζεται.

Συνοψίζοντας, τα θέματα που παρουσιάστηκαν στο κεφάλαιο αυτό, αποτελούν οι κύριες δυνατότητες της ιστοσελίδας. Σκοπός της, η δυνατότητα πλήρης τροποποίησης του συστήματος από τον διαχειριστή και η ζωντανή ενημέρωση του κάθε συμβάντος για την πλήρη κάλυψη. Τέλος, το ίδιο Web Application είναι προσβάσιμο από την Mobile εφαρμογή, μέσω τεχνικών που θα συζητηθούν στην επόμενη ενότητα. Για τον λόγο αυτό, υλοποιείται Responsive Design για την κάλυψη Mobile συσκευών.

6.6 Υλοποίηση Mobile εφαρμογής

Η υλοποίηση του Mobile Application, γίνεται μέσω της React Native σε συνδυασμό με TypeScript. Όπως έχει αναφερθεί στο κεφάλαιο των τεχνολογιών, η React Native είναι ένα framework, που δίνει την δυνατότητα ανάπτυξης Hybrid εφαρμογών, μέσω της React. Δηλαδή, με ένα source code, αναπτύσσεται η ίδια εφαρμογή για Android και για iOS smartphones.

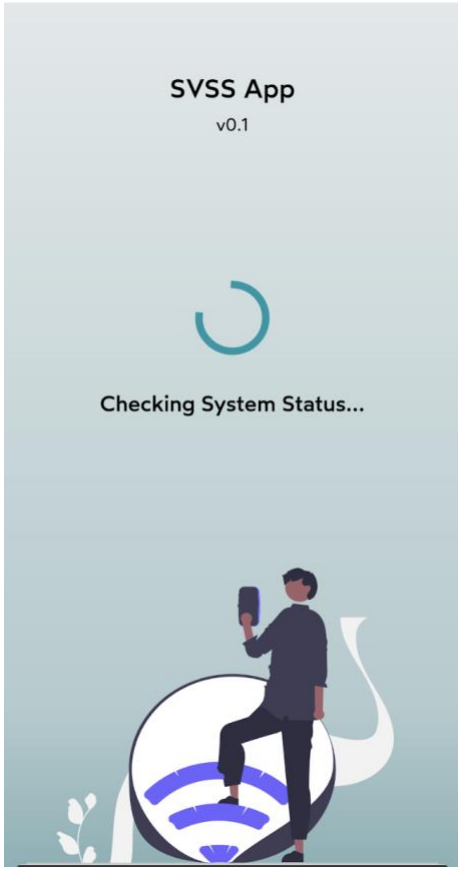
Η υλοποίηση της Mobile εφαρμογής μέσω της τεχνολογίας React Native, επιλέχθηκε για τους εξής λόγος:

1) Υποστήριξη Android και iOS μέσω ενός source code. Ταχύτερη ανάπτυξη και ευκολότερη τροποποίηση των περιεχομένων.
2) Μείωση διαφορετικών τεχνολογιών. Η ιστοσελίδα και η Mobile εφαρμογή, υλοποιούνται μέσω React.
3) Μεγαλύτερη εξοικείωση με περιβάλλον React.

Το βασικότερο και απο τα μοναδικά μειονεκτήματα, αυτής της υλοποίησης, αποτελεί η αυξημένη χρήση πόρων από την συσκευή. Γεγονός, που αποτελεί σημαντικό σε εφαρμογές που έχουν αυξημένη χρήση των πόρων της συσκευής και απαιτούν ομαλές εμπειρίες, όπως παιχνίδια. Η συγκεκριμένη εφαρμογή, δεν παρουσιάζει κάποια τέτοια ανάγκη. Σκοπός της λοιπόν, είναι η εκμετάλλευση του αντάπτορα Bluetooth των smartphones για την αποστολή WiFi στοιχείων προς το σύστημα και η ομαλότερη πρόσβαση του χρήστη στην ιστοσελίδα του συστήματος.

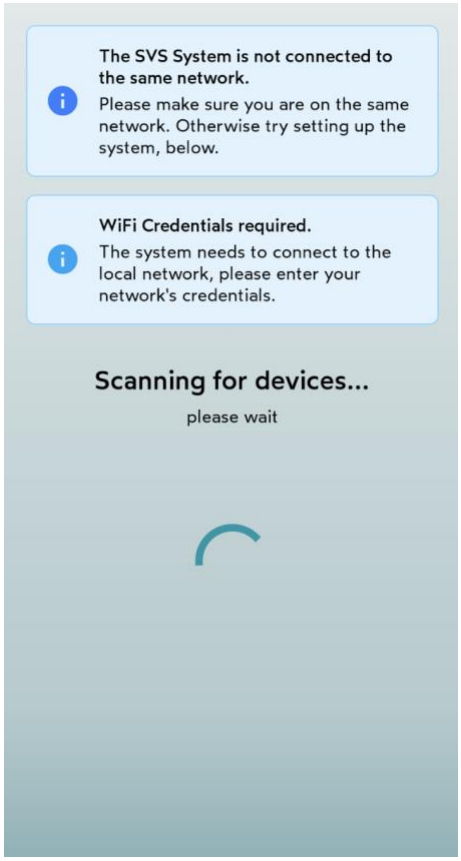
Στις επόμενες υποενότητες, θα γίνει λόγος, για την η δομή της ιστοσελίδας, την υλοποίηση επικοινωνίας Bluetooth απο την μεριά της συσκευής και τέλος η λήψη Push Notifications στην κατάσταση συναγερμού.

6.6.1 Δομή εφαρμογής



Σχήμα 6.47: Οθόνη εύρεσης του συστήματος μέσω δικτύου WiFi.

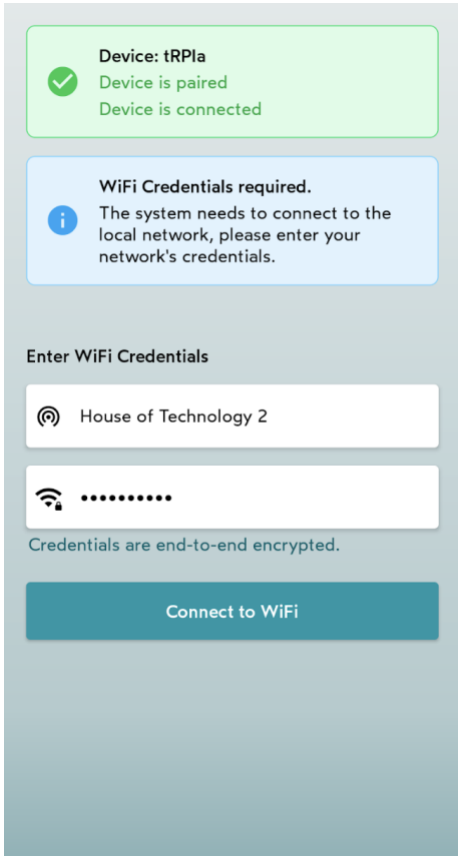
Κατά το άνοιγμα της εφαρμογής, πραγματοποιείται εύρεση του Raspberry Pi, στο δίκτυο. Στην περίπτωση επιτυχίας, ο χρήστης δρομολογείται στην ιστοσελίδα του συστήματος, μέσω WebView. Όμως, αν το σύστημα δεν είναι διαθέσιμο, τότε ξεκινάει η ροή αποστολής WiFi στοιχείων μέσω Bluetooth.



Σχήμα 6.48: Οθόνη σύνδεσης Bluetooth μεταξύ συσκευής και Raspberry Pi.

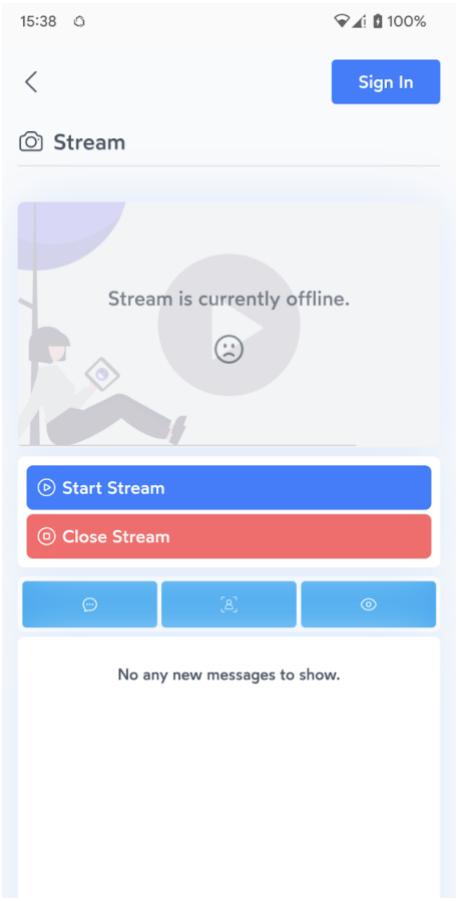
Μετά την δρομολόγηση του χρήστη στην ροή σύνδεσης Bluetooth. Η εφαρμογή, ζητάει από τον χρήστη την αποδοχή πρόσβασης στον αντάπτορα Bluetooth και την ενεργοποίηση του, όπως απαιτείται από το λειτουργικό σύστημα. Επιπλέον, τον ενημερώνει για την διαδικασία και για την ανάγκη αποστολής WiFi στοιχείων. Τέλος, πραγματοποιείται Scan μέσω Bluetooth, για την εύρεση του Raspberry Pi.

Στην περίπτωση, που δεν βρεθεί το αντάπτορας του συστήματος, τότε το Raspberry Pi είναι συνδεδεμένο σε διαφορετικό δίκτυο ή συνέβει κάποιο σφάλμα και ο χρήστης μπορεί να κάνει επανεκκίνηση του συστήματος και του Mobile application.



Σχήμα 6.49: Σύνδεση μέσω Bluetooth και αποστολή WiFi στοιχείων.

Μόλις γίνει εντοπισμός του συστήματος, πραγματοποιείται αυτόματη ζεύξη (Pairing) μεταξύ των δύο συσκευών. Στην περίπτωση αυτή, ο χρήστης καλείται να αποδεκτή το Pairing και πραγματοποιείται η σύνδεση. Επιπλέον, γίνεται εμφάνιση της φόρμας για την εισαγωγή του SSID και κωδικού WiFi. Γίνεται αποστολή των στοιχείων μέσω μηνύματος JSON, κρυπτογραφημένο με μέθοδο AES. Τέλος το Raspberry Pi, λαμβάνει το μήνυμα, το αποκρυπτογραφεί και πραγματοποιεί την σύνδεση στο WiFi, επιστρέφοντας μήνυμα επιτυχίας στην εφαρμογή.



Σχήμα 6.50: Πρόσβαση στην ιστοσελίδα του συστήματος.

Τελευταίο χαρακτηριστικό της εφαρμογής, αποτελεί η δυνατότητα πρόσβαση της ιστοσελίδας. Η υλοποίηση αυτή, εκμεταλλεύεται τα WebViews, που υπάρχουν σε κάθε Mobile λειτουργικό σύστημα, τα οποία δουλεύουν σαν ένας Browser. Συνεπώς, είναι δυνατή η επεξεργασία HTML, CSS και JavaScript για την εμφάνιση και διαχείριση περιεχομένου. Αυτό έχει ως αποτέλεσμα, την ενσωμάτωση ολόκληρης της ιστοσελίδας στην Mobile εφαρμογή χωρίς την ανάγκη ανάπτυξης ξανά όλων των χαρακτηριστικών στη πλατφόρμα του Mobile.

```
import WebView from 'react-native-webview';
import {View} from 'react-native';

export const WebViewScreen = () => {
  return (
    <View style={{flex: 1}}>
      <WebView
        javaScriptEnabled={true}
        domStorageEnabled={true}
        allowsInlineMediaPlayback={true}
        mixedContentMode="always"
        mediaPlaybackRequiresUserAction={false}
        allowsFullscreenVideo={true}
        source={{
          uri: 'https://trpia.local/stream',
        }}
        style={{flex: 1}}
      />
    </View>
```

Σχήμα 6.51: Υλοποίηση WebView.

Στην εικόνα 6.51, παρατηρείται η υλοποίηση του WebView μέσω της React Native. Συγκεκριμένα, χρησιμοποιείται η βιβλιοθήκη "react-native-webview", η οποία διαχειρίζεται την χρήση WebView της κάθε πλατφόρμας. Όπως φαίνεται, γίνεται ανάθεση της τοποθεσίας, στην διεύθυνση της ιστοσελίδας "https://trpia.local/stream". Επιπλέον, παρέχονται ρυθμίσεις για την τροποποίηση των δυνατοτήτων του WebView.

Πίνακας 6.7: Ρυθμίσεις WebView.

Ρύθμιση	Σημασία
javaScriptEnabled	Ενεργοποίηση της JavaScript.
domStorageEnabled	Ενεργοποίηση του Session και Local Storage.
allowsInlineMediaPlayback	Επιτρέπεται η έναρξη του βίντεο μέσα της σελίδας
mixedContentMode	Επιτρέπονται η φόρτωση αρχείων.
mediaPlaybackRequiresUserAction	Έναρξη Live Stream, χωρίς την ανάγκη αλληλεπίδρασης με την σελίδα από τον χρήση.
allowsFullScreenVideo	Ενεργοποίηση βίντεο πλήρης οθόνης.

Συνοψίζοντας, αυτές αποτελούν οι κύριες εικόνες και λειτουργίες του Mobile Application. Στην συνέχεια της ενότητας, θα αναφερθούν οι υλοποίησης της επικοινωνίας Bluetooth από την μεριά της συσκευής και η λήψη και εμφάνιση Push Notifications.

6.6.2 Επικοινωνία μέσω Bluetooth

Η επικοινωνία Bluetooth της Mobile εφαρμογής, υλοποιείται μέσω της βιβλιοθήκης "react-native-bluetooth-classic", η οποία προσφέρει μεθόδους για την διαχείριση του αντάπτορα Bluetooth της Mobile συσκευής για Android και για iOS, μαζί.

Αρχικά, για την διαχείριση του Bluetooth, χρειάζεται η αποδοχή των permission από τον χρήστη. Οι άδειες που χρειάζονται, περιλαμβάνουν την τοποθεσία της συσκευής και τις δυνατότητες αποδοχής και εύρεσης μέσω του Bluetooth αντάπτορα. Μόλις ο χρήστης, εξουσιοδοτήσει την εφαρμογή, στην συνέχεια ελέγχεται η κατάσταση του αντάπτορα και ζητείται η ενεργοποίηση του στην περίπτωση που είναι ανενεργός. Μετά την ενεργοποίηση, το σύστημα αυτόματα εκτελεί Scan για να βρει την συσκευή Raspberry Pi με το συγκεκριμένο Bluetooth MAC Address. Μόλις βρεθεί, πραγματοποιείται το pairing και στην συνέχεια εμφανίζεται η φόρμα για την αποστολή των WiFi στοιχείων. Παρακάτω, μέσω της μεθόδου writeToDevice, στέλνονται ως κρυπτογραφημένο JSON μήνυμα τα στοιχεία μέσω Bluetooth.

```
const writeToDevice = async ({password, ssid}: WifiCredentials) => {
  if (!connectedDevice) {
    console.log('No device connected');
    return false;
  }
  const payload = encrypt({
    ssid: ssid,
    password: password,
  });
  if (!payload) {
    console.log('Encryption failed');
    return false;
  }
```

```
    setWaitingResponse(true);
    const address = connectedDevice.address;
    let success = BluetoothClassic.writeToDevice(
      address,
      JSON.stringify(payload),
    );
    success = BluetoothClassic.writeToDevice(address, '\n');
    console.log('Write success: ', success);
    return success;
  };
```

Τέλος, το σύστημα, όπως έχει προαναφερθεί, λαμβάνει το μήνυμα, το αποκρυπτογραφεί, συνδέεται στο WiFi και αποκρίνεται με επιτυχία στην συσκευή. Η συσκευή, στην συνέχεια, εμφανίζει στον χρήστη την επιτυχία της σύνδεσης.

6.6.3 Push Notifications

Το σύστημα, στην έναρξη συναγερμού, στέλνει Push Notifications στις Mobile συσκευές που έχει αποθηκευμένα fcmTokens. Τα fcmTokens (Firebase Cloud Messaging), αποτελούν ένας τρόπος εμφάνισης ειδοποιήσεων από μία εφαρμογή, στην περίπτωση που δεν είναι ανοικτή αλλά ούτε στο παρασκήνιο.

Στο άνοιγμα της εφαρμογής, δημιουργείται ένα fcmToken και γίνεται αποστολή του στο Backend, για αποθήκευση στην βάση. Ο τρόπος αυτός, γίνεται πολύ εύκολα μέσω του Firebase Messaging:

```
import messaging from '@react-native-firebase/messaging';

const getDeviceFCMToken = async () => {
 await messaging().registerDeviceForRemoteMessages();
 const fcmToken = await messaging().getToken();
 return fcmToken;
};
```

Όπως φαίνεται, η μέθοδος getDeviceFCMToken, εκμεταλλεύεται τις μεθόδους που προσφέρει η Firebase για την αυτόματα δημιουργία του token.

Στην συνέχεια, για την λειτουργία των Push-Notifications, απομένει η αποστολή του σήματος από το Backend, και η διαχείριση του από την Mobile εφαρμογή.

```go
func SendIntrusionNotification(predictionTrigger models.Prediction) error{
    devices, err := db.GetDevices()
    if err != nil {
        return err
    }
    if len(devices) == 0 {
        return errors.New("No devices found to send alerts to")
    }
    app := firebaseState.GetApp()
    if app == nil {
        return errors.New("Firebase app not initialized")
    }
    client, err := app.Messaging(context.Background())
    if err != nil {
        return err
    }
    for _, device := range devices {
        fmt.Println("Sending alert to device: ", device.FcmToken[0:5])
        message := messaging.Message{
            Data: map[string]string{"intruder": string(predictionTrigger.Class), "action": "open-stream"},
            Token: device.FcmToken,
            Android: &messaging.AndroidConfig{
                Priority: "high",
            },
            APNS: &messaging.APNSConfig{
                Headers: map[string]string{
                    "apns-priority": "10",
                },
                Payload: &messaging.APNSPayload{
                    Aps: &messaging.Aps{
                        ContentAvailable: true,
```

Σχήμα 6.52: Αποστολή Push Notification από το Backend.

Στην εικόνα 6.52, φαίνεται η υλοποίηση της μεθόδου SendIntrusionNotification. Η συνάρτηση αυτή, αρχικά παίρνει τα αποθηκευμένα fcmTokens, το client του Firebase Messaging και στην συνέχεια για κάθε token στέλνει το μήνυμα, ενσωματώνοντας τα δεδομένα της κλάσης που προκάλεσε τον συναγερμό και ο τύπος “action” που θα διαχειριστεί από την εφαρμογή. Επιπλέον, ορίζονται παράμετροι για το κάθε λειτουργικό σύστημα, όπως η υψηλή προτεραιότητα της ειδοποίησης.

Επόμενα βήματα της υλοποίησης, αποτελούν η διαχείριση και η εμφάνιση της ειδοποίησης στον χρήστη. Η εμφάνιση της ειδοποίησης πραγματοποιείται από την βιβλιοθήκη “notifee”.

```
const alertUser = async (
  channelId: string,
  payload: NotificationPayload | null,
) => {
  if (!payload) {
    console.log('No payload, ignoring');
    return;
  }
  console.log(payload);
  await notifee.displayNotification({
    title: '🚨 INTRUSION ALERT! 🚨', // HTML supported for title
    body: `Potential intruder, ${payload.intruder}. Tap to review.`,
    android: {
      smallIcon: 'ic_launcher',
      channelId,
      importance: AndroidImportance.HIGH,
      sound: 'default',
      pressAction: {
        id: 'default', // This will open the app
      },
      actions: [
        {
          title: 'Open Stream',
          pressAction: {id: payload.action, launchActivity: 'default'},
        },
        {
          title: 'Dismiss',
          pressAction: {id: 'dismiss'},
        },
      ],
    },
```

Σχήμα 6.53: Συνάρτηση εμφάνισης ειδοποίησης.

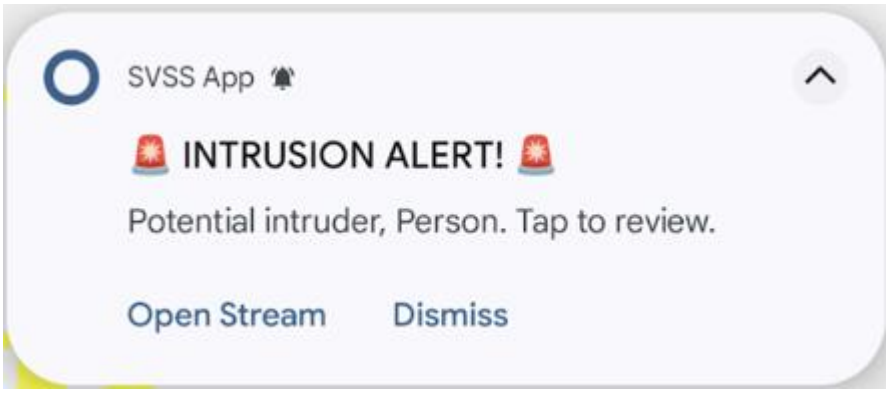
Μέσω της “alertUser”, πραγματοποιείται η εμφάνιση της ειδοποίησης στον χρήστη. Επιπλέον, μέσω της βιβλιοθήκης “notifee”, υπάρχει η δυνατότητα ρύθμισης των λειτουργικών της ειδοποίησης που θα εμφανιστεί. Στο συγκεκριμένο παράδειγμα, δηλώνονται 2 κουμπιά, ένα για την διαγραφή της ειδοποίησης, και το άλλο για το άνοιγμα της εφαρμογής.

Μετά την παραμετροποίηση της ειδοποίησης, απομένει η εμφάνιση της, όταν η εφαρμογή είναι κλειστή.

```
/**
 ** Handles Messages received from firebase messaging
 */
messaging().setBackgroundMessageHandler(async remoteMessage => {
  console.log('Handling a background message', remoteMessage);
  const payload = remoteMessage.data as NotificationPayload | undefined;
  if (!payload || payload.action !== 'open-stream') {
    console.log('Payload provided is not correct');
    return;
  }
  const channelId = await Notifications.createChannel();
  await Notifications.alertUser(channelId, payload);
});
```

Σχήμα 6.54: Διαχείριση σήματος Push Notification στο Background του λειτουργικού συστήματος.

Όπως φαίνεται στο παράδειγμα, μέσω της χρήσης του Firebase Messaging, δηλώνεται η διαχείριση της λήψης απομακρυσμένου μηνύματος στο παρασκήνιο. Αυτή η μέθοδος, εκτελείται και στην περίπτωση που η εφαρμογή είναι τελείως κλειστή, αρκεί μόνο ο συγχρονισμός του fcmToken που υπάρχει στο Backend.



Σχήμα 6.55: Εμφάνιση Push Notification στον χρήστη.

Τέλος, μέσω του Google Play Services, στα Android, αποτελεί εφαρμογή συστήματος και διαχειρίζεται την λήψη και διαχείριση του μηνύματος στην εφαρμογή. Συνεπώς, εμφανίζεται η ειδοποίηση της εισβολής, όπου ο χρήστης με το πάτημα της, δρομολογείται στην ιστοσελίδα της εφαρμογής.

6.7 Υλοποίηση μέτρων ασφάλειας

Για την καλύτερη ασφάλεια καθόλη την λειτουργία του συστήματος, έχουν υλοποιηθεί διάφορες τεχνικές. Αρχικά χρησιμοποιείται SSL/TLS πιστοποιητικό για την χρήση HTTPS και WSS μηνυμάτων, μεταξύ client και server. Επιπλέον, ο Nginx λειτουργεί ως Reverse Proxy, για να κρύψει την πραγματική διεύθυνση των υπηρεσίες που είναι ορατές στο frontend. Τέλος, για την αποστολή στοιχείων WiFi μέσω Bluetooth, υλοποιείται κρυπτογράφηση AES. Σε αυτό το κεφάλαιο, θα αναλυθεί η υλοποίηση του κάθε μέτρου ασφαλείας.

6.7.1 Εφαρμογή του SSL/TLS

Για την πρόσβαση της ιστοσελίδας μέσω HTTPS για την ομαλότερη πρόσβαση από τους Browsers και την κρυπτογράφηση των μηνυμάτων, υλοποιείται πρωτόκολλο SSL/TLS.

Η υλοποίηση αυτή, απαιτεί την χρήση ενός πιστοποιητικού, το οποίο μπορεί να δημιουργηθεί μέσω ενός CA (Central Authority). Για την αποφυγή χρήσης εξωτερικού CA, μέσω του εργαλείου mkcert, δημιουργείται ένα τοπικό CA το οποίο πιστοποιεί SSL/TLS πιστοποιητικό, κάνοντας το έγκυρο για το τοπικό δίκτυο και την σύνδεση του συστήματος. Μετά την εγκατάσταση του mkcert, η δημιουργία του τοπικού CA γίνεται μέσω της εντολής:

```
mkcert -install
```

Στην συνέχεια, η δημιουργία πιστοποιητικού SSL/TLS, για τις διευθύνσεις τοπικού δικτύου και συστήματος, γίνεται μέσω της εντολής:

```
mkcert trpia.local localhost 127.0.0.1
```

Τέλος, δημιουργούνται 2 αρχεία τα οποία περιέχουν 1 κρυπτογραφημένο κλειδί, το καθένα, το ένα δημόσιο και το άλλο ιδιωτικό. Το δημόσιο κλειδί, στέλνεται στον Browser για ταυτοποίηση του διακομιστή του συστήματος. Το ιδιωτικό κλειδί, χρησιμοποιείται μόνο από τον διακομιστή, Nginx, για την αποκρυπτογράφηση των εισερχόμενων μηνυμάτων. Συγκεκριμένα η ρύθμιση στον Nginx γίνεται:

```
ssl_protocols TLSv1.2 TLSv1.3;
ssl_prefer_server_ciphers on;
ssl_ciphers 'ECDHE+AESGCM:CHACHA20';
ssl_session_cache shared:SSL:10m;
ssl_session_timeout 1d;
ssl_session_tickets off;
server {
    listen 443 ssl;
    server_name trpia.local localhost 127.0.0.1;

    ssl_certificate /etc/nginx/ssl/trpia.local+2.pem;
    ssl_certificate_key /etc/nginx/ssl/trpia.local+2-key.pem;
```

Στο απόσπασμα του αρχείου ρυθμίσεων του Nginx, δηλώνεται η χρήση TLS πιστοποιητικού μαζί με την ρύθμιση στον τρόπο κρυπτογράφησης και της λήξης των πιστοποιημένων sessions. Επιπλέον δηλώνεται ο server, ο οποίος σερβίρει την ιστοσελίδα, να χρησιμοποιήσει τα SSL πιστοποιητικά και να ανοίγει στο port 443, το οποίο αποτελεί η προεπιλεγμένη HTTPS θύρα.

6.7.2 Nginx Reverse Proxy

Καθώς, έχει δηλωθεί η ιστοσελίδα με την χρήση HTTPS, η πρόσβαση στο Backend μέσω HTTP και WebSockets αποτελεί μη ασφαλής μέθοδος. Οπότε πρέπει μία μία υπηρεσία να υιοθετήσει SSL/TLS πιστοποιητικό. Για την αποφυγή της πρακτικής αυτής, αλλά και την αύξηση της ιδιωτικότητας του Backend, υλοποιείται ο Nginx ως Reverse Proxy.

Συνεπώς, ο διακομιστής, λειτουργεί ως διαμεσολαβητής αιτημάτων μεταξύ χρήση και Backend και Mosquitto broker. Οπότε, ο τελικός χρήστης γνωρίζει μόνο την διεύθυνση του Nginx και ότι χρησιμοποιεί SSL/TLS πρωτόκολλο και τίποτα και περισσότερο. Η ρύθμιση της λειτουργίας αυτής, αποτελείται από μερικές γραμμές:

```
    …
    location /api/ {
        proxy_pass http://backend:3000/;

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
    …
```

Η διεύθυνση του backend μέσω Docker, http://backend:3000, κρύβεται πίσω από τον Nginx στην διεύθυνση https://trpia.local/api/. Επιπλέον ρυθμίζεται και η σύνδεση WebSockets του Mosquitto broker.

```
…
location /ws/ {
     proxy_pass http://mosquitto:9001/;

     proxy_http_version 1.1;
     #* Upgrades connections from http to websocket.
     proxy_set_header Upgrade $http_upgrade;
     proxy_set_header Connection "upgrade";
     proxy_set_header Host $host;
 }
 …
```

Τέλος, η διεύθυνση του Mosquitto WebSocket broker, κρύβεται πίσω από την https://trpia.local/ws/. Επιπλέον, δηλώνονται proxy headers, για την διαχείριση και την αναβάθμιση του εισερχόμενου πρωτοκόλλου, HTTPS, σε WebSocket.

6.7.3 Κρυπτογραφία AES

Όπως έχει αναφερθεί, η αποστολή WiFi στοιχείων στο Raspberry Pi, μέσω της Mobile εφαρμογής, πραγματοποιείται με την μέθοδο κρυπτογράφησης AES.

Η κρυπτογράφηση και αποκρυπτογράφηση ενός μηνύματος που έχει υποστεί την μέθοδο AES, προϋποθέτει την δημιουργία ενός κρυφού κλειδιού, το οποίο είναι γνωστό και από τις 2 πλευρές. Επιπλέον, κατά την κρυπτογράφηση του μηνύματος, απαιτείται η δημιουργία ενός τυχαίου IV (Initialization Vector), για την εισαγωγή τυχαιότητας και μοναδικότητας του κρυπτογραφημένου μηνύματος. Τέλος, με την χρήση του κρυφού κλειδιού και του IV δημιουργείται το κρυπτογραφημένο κείμενο το οποίο στέλνεται στο Raspberry Pi μαζί με το IV. Ο μικρο υπολογιστής, χρησιμοποιεί το IV και το κρυφό κλειδί, που έχει αποθηκευμένο, για την αποκρυπτογράφηση του μηνύματος.

```
import crypto from 'crypto';
import {EncryptedPayload, WifiCredentials} from '../types';
export const decryptMessage = ({
  _ciphertext,
  _iv,
}: EncryptedPayload): WifiCredentials | null => {
  const decipher = crypto.createDecipheriv(
    'aes-256-cbc',
    Buffer.from(process.env.SECRET_KEY as string, 'base64'),
    Buffer.from(_iv, 'base64'),
  );
  let decrypted = decipher.update(_ciphertext, 'base64', 'utf8');
  decrypted += decipher.final('utf8');
  try {
    return JSON.parse(decrypted) as WifiCredentials;
  } catch (err) {
    console.log('Error parsing decrypted wifi credentials');
    return null;
  }
};
```

Σχήμα 6.56: Αποκρυπτογράφηση WiFi Credentials.

Στην εικόνα 6.56, παρουσιάζεται η αποκρυπτογράφηση των WiFi στοιχείων, που λαμβάνει το Raspberry Pi, μέσω Bluetooth. Όπως φαίνεται, στην διαδικασία συμμετέχουν:

1) crypto. Βιβλιοθήκη της Nodejs, που προσφέρει μεθόδους κρυπτογράφησης
2) το IV (_iv), το κρυπτογραφημένο κείμενο (_ciphertext) και το κρυφό κλειδί.
3) Αποκρυπτογράφηση μέσω AES-256-CBC. Δηλαδή, κρυπτογράφηση μέσω 256-bit κρυφού κλειδιού και CBC (Cipher block Chaining) λειτουργίας.

Συνοψίζοντας, αυτές οι 3 υλοποιήσεις ασφαλείας, αποτελούν οι βασικότεροι, σημαντικότεροι και πιο απλοί τρόποι προστασίας ενός συστήματος. Η χρήση HTTPS αποτελεί βασικό στοιχείο κάθε ιστοσελίδας. Η χρήση ενός σημείου για την πρόσβαση στο σύστημα, αποτελεί βασικό στοιχείο κάθε συστήματος με πολλές υπηρεσίες. Τέλος, η χρήση μεθόδων κρυπτογραφίας, προσθέτει παραπάνω μέτρα ασφάλειας στην ανταλλαγή μηνυμάτων, με ιδιαίτερη χρήση στην ανταλλαγή ευαίσθητων δεδομένων, όπως κωδικοί.

Κεφάλαιο 7ο: Συζήτηση και συμπεράσματα

7.1 Συμπεράσματα

Στην παρούσα διπλωματική εργασία έγινε λόγος για την ανάπτυξη ενός έξυπνου συστήματος παρακολούθησης χώρου. Με κύρια χαρακτηριστικά να αποτελούν η έξυπνη αναγνώριση, η αποστολή ειδοποιήσεων στον ιδιοκτήτη και το Live Streaming των συνδεδεμένων συσκευών.

Κοινό χαρακτηριστικό του συστήματος, αποτελεί η επιλογή σύγχρονων τεχνολογιών και τεχνικών για την υλοποίηση. Η δημιουργία μικρο υπηρεσιών μέσω Docker, η ανάπτυξη σε Raspberry Pi, η χρήση τεχνητής νοημοσύνης και η εφαρμογή τεχνικών διαχείρισης της ενέργειας, συνιστούν σημαντικές τεχνικές, στα συστήματα παρακολούθησης χώρου της αγοράς και του ερευνητικού κλάδου.

Η λειτουργία δυναμικής ρύθμισης του δικτύου, εκμεταλλεύοντας τις δυνατότητες Bluetooth του Raspberry Pi και του WiFi από τον μικροελεγκτή ESP32, κάνουν λόγο για σπανιότερη μέθοδο υλοποίησης που συναντάται κυρίως στον τομέα του IOT.

Η ανάπτυξη Web και Mobile application μέσω σύγχρονων εργαλείων, έχει ως αντίκτυπο την ευκολότερη και ομαλότερη κλιμάκωση του συστήματος σε μελλοντικά πλάνα. Η χρήση της Go ως κύρια Backend υπηρεσία, μειώνει αρκετά τους χρόνους επεξεργασίας, έτσι ώστε η συσκευή να μπορεί να ανταπεξέλθει σε μεγάλο φόρτο εργασίας.

Περαιτέρω, η χρήση Push Notifications και SMS μηνυμάτων, για την ειδοποίηση του χρήστη, αποτελεί μία ευέλικτη και αρκετά υποστηριζόμενη λύση. Καθώς, όλος ο πληθυσμός έχει πρόσβαση στην λήψη SMS και το μεγαλύτερο κομμάτι, κατέχει smartphones για εμφάνιση Push Notifications.

Επιπλέον, η ευελιξία της υλοποίησης, δίνει την δυνατότητα στον χρήστη, να χρησιμοποιήσει ένα μεγάλο εύρος παρόμοιων συσκευών εισόδου αλλά και Raspberry Pi. Γεγονός, που παρουσιάζει προσαρμοστικότητα στον προϋπολογισμό και στις ανάγκες της κάθε υλοποίησης.

Συνοψίζοντας, η διατριβή, εξετάζει αρκετά θέματα, από τον τομέα της παρακολούθησης χώρου και επιχειρεί να εντάξει και να αναλύσει την δική της υλοποίηση για την δημιουργία ενός έξυπνου, ασφαλούς και αποδοτικού συστήματος.

7.2 Προτάσεις βελτίωσης

Η υλοποίηση του συστήματος, αποτελεί μία πλήρης λύση, όμως καθώς το εύρος της έρευνας αποτελείται από πολλά διαφορετικούς τομείς, υπάρχουν αρκετά μέρη στα οποία αρμόζει βελτιώσεις. Οι προτάσεις βελτίωσης αποτελούν:

1) Εκπαίδευση μοντέλων για την έξυπνη αναγνώριση. Τα μοντέλα που χρησιμοποιούνται στην τωρινή υλοποίηση είναι προεκπαιδευμένα, αν και αποτελούν οι καλύτερες λύσεις της κοινότητας, παραμένουν γενικές λύσεις. Συστήματα σαν αυτό, για την βελτιστοποίηση της αναγνώρισης, χρειάζονται την εκπαίδευση των ήδη εκπαιδευμένων μοντέλων στις δικές τους απαιτήσεις. Η τεχνική αυτή ονομάζεται Fine tuning.

2) Υλοποίηση authentication χωρίς την ανάγκη εξωτερικών παρόχων. Η τωρινή υλοποίηση, βασίζεται στην αυθεντικοποίηση χρηστών μέσω Google. Αν και αποτελεί μία συνηθισμένη και ασφαλή μέθοδος, δεν μπορεί να αντικαταστήσει τα οφέλη της υλοποίησης του κλασικού authentication που διαχειρίζεται πλήρως από το σύστημα.

3) Ρύθμιση Mobile application για iOS. Η υλοποίηση του Mobile application, υποστηρίζει Android και iOS από το ίδιο source code με τις σωστές ρυθμίσεις. Η τωρινή λύση, επικεντρώνεται στα Android κινητά καθώς η διανομή της εφαρμογής είναι ευκολότερη σε σχέση με την iOS που χρειάζεται αρκετά έξτρα βήματα.
4) Υλοποίηση όλων των λειτουργιών της ιστοσελίδας και στην Mobile εφαρμογή. Με την τωρινή υλοποίηση, ο χρήστης μπορεί να αποκτήσει πρόσβαση στην ιστοσελίδα και μέσω του Mobile application, αλλά μέσω WebView. Η λύση αυτή, είναι αρκετά πρακτική για την αποφυγή δημιουργίας της ίδιας διεπαφής ξανά. Όμως, δεν είναι αρκετά διαχειρίσιμο από την μεριά της εφαρμογής.

Συνοψίζοντας, αυτά αποτελούν τα βασικότερα ζητήματα που χρειάζονται βελτίωση, χωρίς να σημαίνει πως είναι και μοναδικά. Ένα σύστημα, τέτοιου βεληνεκούς, που συνδυάζει τεχνητή νοημοσύνη, Live Streaming, κ.α., χρειάζεται και υπόκειται συνεχώς σε βελτιώσεις.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] A. Jain, S. Basantwani, O. Kazi, and Y. Bang, “Smart surveillance monitoring system,” IEEE Xplore, Feb. 01, 2017.

[2] Azeddine Beghdadi, M. Asim, Noor Almaadeed, and M. A. Qureshi, “Towards the design of smart video-surveillance system,” pp. 162–167, Aug. 2018, doi: https://doi.org/10.1109/ahs.2018.8541480.

[3] K. S, A. R, B. B. S. Krishna, and V. P, “Advancements in Real-Time Face Recognition Algorithms for Enhanced Smart Video Surveillance,” Procedia Computer Science, vol. 230, pp. 486–492, Jan. 2023, doi: https://doi.org/10.1016/j.procs.2023.12.104.

[4] A. B. Rahimi, P. A. Danesh, Noghre, Ghazal Alinezhad, C. Neff, A. Ravindran, and H. Tabkhi, “Understanding Ethics, Privacy, and Regulations in Smart Video Surveillance for Public Safety,” arXiv.org, 2022. https://arxiv.org/abs/2212.12936.

[5] A. Costin, “Security of CCTV and Video Surveillance Systems,” Proceedings of the 6th International Workshop on Trustworthy Embedded Devices - TrustED ’16, 2016, doi: https://doi.org/10.1145/2995289.2995290.

[6] H. Ghael, D. Solanki, and G. Sahu, “A Review Paper on Raspberry Pi and Its Applications,” International Journal of Advances in Engineering and Management (IJAEM), vol. 2, no. 12, p. 225, 2008, doi: https://doi.org/10.35629/5252-0212225227.

[7] “The Differences Between Raspberry Pi 4 Model B & Raspberry Pi 5,” Kitronik Ltd. https://kitronik.co.uk/blogs/resources/the-differences-between-raspberry-pi-4-model-b-raspberry-pi-5

[8] M. Fezari, B. Mokhtar, A. Al Dahoud, M. Fezari, and A. Al-Dahoud, “Raspberry Pi 5 : The new Raspberry Pi family with more computation power and AI integration Raspberry Pi 5 : The new Raspberry Pi family with more computation power and AI integration,” doi: https://doi.org/10.13140/RG.2.2.13547.52009.

[9] Raspberry Pi Ltd, “Buy a Raspberry Pi Active Cooler – Raspberry Pi,” Raspberry Pi, 2024. https://www.raspberrypi.com/products/active-cooler/

[10] “Raspberry Pi Camera Module 3 Standard NoIR Wide NoIR Wide,” 2023. Available: https://datasheets.raspberrypi.com/camera/camera-module-3-product-brief.pdf.

[11] “Waveshare GSM/GPRS/Bluetooth HAT - SIM800C,” Grobotronics.com, 2025. https://grobotronics.com/waveshare-ggsm-gprs-bluetooth-hat-sim800c.html?sl=en (accessed May 28, 2025).

[12] “GSM GPRS SIM800C Module ORDER CODE: RDL807 GSM GPRS SIM800C Module.” Accessed: May 27, 2025. [Online]. Available: https://www.researchdesignlab.com/projects/GSM-SIM800C-Manual.pdf

[13] “Understanding the Differences Between UART and I2C,” Total Phase Blog, Dec. 16, 2020. https://www.totalphase.com/blog/2020/12/differences-between-uart-i2c

[14] M. Babiuch, P. Foltynek, and P. Smutny, “Using the ESP32 Microcontroller for Data Processing,” 2019 20th International Carpathian Control Conference (ICCC), May 2019, doi: https://doi.org/10.1109/carpathiancc.2019.8765944.

[15] E. Gatial, Z. Balogh, and L. Hluchý, “Concept of Energy Efficient ESP32 Chip for Industrial Wireless Sensor Network,” IEEE Xplore, Jul. 01, 2020. https://ieeexplore.ieee.org/abstract/document/9147189.

[16] “ESP32-DevKitC V4 - ESP32 - — esp-dev-kits latest documentation,” Espressif.com, 2016. https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32/esp32-devkitc/user_guide.html?highlight=esp32%20wroom.

[17] espressif, “ESP32-WROOM-32 Datasheet,” 2023. Available: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf.

[18] “Gravity: Digital Infrared Motion Sensor For Arduino,” www.dfrobot.com. https://www.dfrobot.com/product-119.html.

[19] “Sound Sensor,” www.waveshare.com. https://www.waveshare.com/sound-sensor.htm.

[18] Docker, “Enterprise Application Container Platform | Docker,” Docker, 2024. https://www.docker.com/.

[19] “The Docker Book,” Google Books, 2016. https://books.google.gr/books?hl=el&lr=&id=4xQKBAAAQBAJ&oi=fnd&pg=PA1&dq=docker&ots=wy6K6v6cEY&sig=yPgEJaD-SGz9KIeKynqQWqnf8UQ&redir_esc=y#v=onepage&q=docker&f=false (accessed May 26, 2025).

[20] routerhan, “A Beginner’s Guide to Docker — Before&After - routerhan - Medium,” Medium, Sep. 07, 2023. https://routerhan.medium.com/a-beginners-guide-to-docker-before-after-b4ea0be70b64.

[21] Docker, “What is a Container?,” Docker, 2023. https://www.docker.com/resources/what-container/.

[22] Microsoft, “TypeScript - JavaScript that scales.” Typescriptlang.org https://www.typescriptlang.org.

[23] “V8 JavaScript engine,” v8.dev. https://v8.dev/.

[24] Lee, Hongki, et al. "SAFE: Formal specification and implementation of a scalable analysis framework for ECMAScript." FOOL 2012: 19th International Workshop on Foundations of Object-Oriented Languages. 2012.

[25] Vishnupriya, “What is Typescript ? - Vishnupriya - Medium,” Medium, Dec. 15, 2018. https://medium.com/@vishnupriya_web/what-is-typescript-faa0890b2baf.

[26] X. Lei, X. Jiang, and C. Wang, “Design and Implementation of a Real-Time Video Stream Analysis System Based on FFMPEG,” 2013 Fourth World Congress on Software Engineering, Dec. 2013, doi: https://doi.org/10.1109/wcse.2013.38.

[27] H. Zeng, Z. Zhang, and L. Shi, “Research and Implementation of Video Codec Based on FFmpeg,” Apr. 2016, doi: https://doi.org/10.1109/icnisc.2016.049.

[28] A. Trimbakrao et al., “FIREBASE -OVERVIEW AND USAGE FIREBASE -OVERVIEW AND USAGE,” Article in Journal of Engineering and Technology Management, vol. 11, p. 281, 2022, Available: https://www.researchgate.net/profile/Anil-Gaikwad-12/publication/362539877_FIREBASE_-OVERVIEW_AND_USAGE/links/62efc738505511283e9a5318/FIREBASE-OVERVIEW-AND-USAGE.pdf

[29] D. Stevenson, “What is Firebase? The complete story, abridged.,” Medium, Sep. 24, 2018. https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0

[30] “Managing Cloud Messaging Tokens,” The Firebase Blog, 2023. https://firebase.blog/posts/2023/04/managing-cloud-messaging-tokens/ (accessed May 27, 2025).

[31] “Authenticate Using Google Sign-In with JavaScript,” Firebase. https://firebase.google.com/docs/auth/web/google-signin

[32] Edi Muškardin, Tamim Burgstaller, M. Tappler, and B. K. Aichernig, “Active Model Learning of Git Version Control System,” pp. 78–82, May 2024, doi: https://doi.org/10.1109/icstw60967.2024.00024.

[33] Sahan Kumarasiri, “Deep dive into JS Promises & Callbacks| IIFE | ES6 Features | Version Controlling | NoSQL,” Medium, Mar. 12, 2022. https://medium.com/@SahanKumarasiri/deep-dive-into-js-promises-callbacks-iife-es6-features-version-controlling-nosql-972988c0b15a.

[34] “Mastering Go,” Google Books, 2019. https://books.google.gr/books?hl=el&lr=&id=ly6sDwAAQBAJ&oi=fnd&pg=PP1&dq=golang&ots=heUWMillN-&sig=k32akcnFF6cqWY2mYScmefPdDOc&redir_esc=y#v=onepage&q=golang&f=false

[35] N. Chhetri, “A Comparative Analysis of Node.js (Server-Side JavaScript),” Culminating Projects in Computer Science and Information Technology, Mar. 2016, Available: https://repository.stcloudstate.edu/csit_etds/5/.

[36] “What Is Node.js and Why You Should Use It,” Kinsta. https://kinsta.com/knowledgebase/what-is-node-js/.

[37] P. Goldsborough, “A Tour of TensorFlow,” arXiv:1610.01178 [cs], Oct. 2016, Available: https://arxiv.org/abs/1610.01178.

[38] B. Mahesh, “Machine learning algorithms - a review,” International Journal of Science and Research (IJSR) ResearchGate Impact Factor, vol. 9, no. 1, 2018, doi: https://doi.org/10.21275/ART20203995.

[39] P. N. H, “The Three Crucial Data Sets in Machine Learning: Training, Validation, and Testing Sets,” Medium, Nov. 30, 2023. https://medium.com/@prasanNH/the-three-crucial-data-sets-in-machine-learning-training-validation-and-testing-sets-99f882fad144.

[40] M. A. Qureshi, M. Deriche, and A. Beghdadi, “Quantifying blur in colour images using higher order singular values,” Electronics Letters, vol. 52, no. 21, pp. 1755–1757, Oct. 2016, doi: https://doi.org/10.1049/el.2016.1792.

[41] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, “ArcFace: Additive Angular Margin Loss for Deep Face Recognition,” 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Jun. 2019, doi: https://doi.org/10.1109/cvpr.2019.00482.

[42] Nafiz Sadman, K. A. Hasan, Elyas Rashno, Furkan Alaca, Y. Tian, and Farhana Zulkernine, “Vulnerability of Open-Source Face Recognition Systems to Blackbox Attacks: A Case Study with InsightFace,” 2021 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1164–1169, Dec. 2023, doi: https://doi.org/10.1109/ssci52147.2023.10371801.

[43] Y. Li and S. Manoharan, “A Performance Comparison of SQL and NoSQL Databases,” 2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), Aug. 2013, doi: https://doi.org/10.1109/pacrim.2013.6625441.

[44] J. Casal, “MongoDB Tutorial For .NET Developers,” Jun. 22, 2024. https://juliocasal.com/blog/MongoDB-Tutorial-For-Dotnet-Developers.

[45] “Learning React,” Google Books, 2020. https://books.google.gr/books?hl=el&lr=&id=tDjrDwAAQBAJ&oi=fnd&pg=PR2&dq=react&ots=DDJ4bhkMZV&sig=8I3LCXN4jnH5stRm07fJPEsVmwA&redir_esc=y#v=onepage&q=react&f=false.

[46] “What Is React? Everything a Tech Leader Needs to Know,” brainhub.eu. https://brainhub.eu/library/what-is-react.

[47] Shahid Yousafxai, “JSX Rules in React(A JavaScript Framework). - Nerd For Tech - Medium,” Medium, Apr. 23, 2021. https://medium.com/nerd-for-tech/jsx-rules-in-react-a-javascript-framework-4b0ab66fdbf9.

[48] Vite, “Getting Started,” vitejs, 2019. https://vite.dev/guide/.

[49] “Ultimate Tailwind CSS Handbook: Build sleek and modern websites with immersive UIs using Tailwind CSS,” Google Books, 2023. https://books.google.gr/books?hl=el&lr=&id=5yRIEQAAQBAJ&oi=fnd&pg=PT22&dq=tailwind+css&ots=IutZvvvq2W&sig=z_mSMtDwcEIoi7JqjJ56ossZrrk&redir_esc=y#v=onepage&q=tailwind%20css&f=false.

[50] Tailwind, “Installing with Vite - Installation,” Tailwindcss.com, 2025. https://tailwindcss.com/docs/installation/using-vite.

[51] W. Danielsson, A. Fröberg, and E. Berglund, “React Native application development -A comparison between native Android and React Native,” 2016. Available: https://www.diva-portal.org/smash/get/diva2:998793/FULLTEXT02.pdf.

[52] R. J. Samara, “Forensic investigations of popular applications on Android and iOS platforms,” May 10, 2022. https://www.researchgate.net/publication/360500246_Forensic_investigations_of_popular_applications_on_Android_and_iOS_platforms.

[53] “iOS Architecture – Redfox Security.” https://redfoxsec.com/blog/ios-architecture/.

[54] A. Patil, “Deep dive into React Native’s New Architecture,” COOX Tech, Jan. 22, 2022. https://medium.com/coox-tech/deep-dive-into-react-natives-new-architecture-fb67ae615ccd.

[55] “About the New Architecture · React Native,” Reactnative.dev, Oct. 31, 2024. https://reactnative.dev/architecture/landing-page.

[56] “Introduction - Zustand,” Pmnd.rs, 2024. https://zustand.docs.pmnd.rs/getting-started/introduction.

[57] C. Ma and Y. Chi, “Evaluation Test and Improvement of Load Balancing Algorithms of Nginx,” IEEE Access, vol. 10, pp. 14311–14324, 2022, doi: https://doi.org/10.1109/access.2022.3146422.

[58] “What is Mosquitto MQTT?,” www.eginnovations.com. https://www.eginnovations.com/documentation/Mosquitto-MQTT/What-is-Mosquitto-MQTT.htm.

[59] E. Vannebäck, “Using the Mosquitto implementation in an embedded environment.” Available: https://www.diva-portal.org/smash/get/diva2:1223840/FULLTEXT01.pdf.

[60] “TCP/IP Network Administration,” Google Books, 2025. https://books.google.gr/books?hl=el&lr=&id=wqabAgAAQBAJ&oi=fnd&pg=PR7&dq=tcp+ip&ots=zTTXxMAjQb&sig=jdRv7es15R11FbAWS_ScAC-zyKM&redir_esc=y#v=onepage&q=tcp%20ip&f=false.

[61] “HTTP: The Definitive Guide,” Google Books, 2025. https://books.google.gr/books?hl=el&lr=&id=3EybAgAAQBAJ&oi=fnd&pg=PR5&dq=HTTP&ots=X72XUbfQXp&sig=ynGmFfJwHj8O-oTZ7mK4IFzTQaY&redir_esc=y#v=onepage&q=HTTP&f=false.

[62] “An overview of HTTP - HTTP | MDN,” MDN Web Docs, Mar. 14, 2025. https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Overview.

[63] V. Pimentel and B. G. Nickerson, “Communicating and Displaying Real-Time Data with WebSocket,” IEEE Internet Computing, vol. 16, no. 4, pp. 45–53, Jul. 2012, doi: https://doi.org/10.1109/mic.2012.64.

[64] D. Mosyan, “HTTP Long Polling vs WebSockets - David Mosyan - Medium,” Medium, Aug. 17, 2024. https://medium.com/@dmosyan/http-long-polling-vs-websockets-dadab8f7f26f.

[65] X. Lei, X. Jiang, and C. Wang, “Design and implementation of streaming media processing software based on RTMP,” Oct. 2012, doi: https://doi.org/10.1109/cisp.2012.6469981.

[66] A. Fecheyr-Lippens, “A Review of HTTP Live Streaming.” Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=57d33cda30c2d497b694470aaa8b502613851fa5.

[67] G. S. Sundaram et al., “Bluetooth communication using a touchscreen interface with the Raspberry Pi,” 2013 Proceedings of IEEE Southeastcon, pp. 1–4, Apr. 2013, doi: https://doi.org/10.1109/secon.2013.6567448.

[68] J. Brown, B. Shipman, and R. Vetter, “SMS: The Short Message Service,” Computer, vol. 40, no. 12, pp. 106–110, Dec. 2007, doi: https://doi.org/10.1109/mc.2007.440.

[69] A. Satapathy and J. Livingston, “A Comprehensive Survey on SSL/ TLS and their Vulnerabilities,” International Journal of Computer Applications, vol. 153, no. 5, pp. 31–38, Nov. 2016, doi: https://doi.org/10.5120/ijca2016912063.

ΠΑΡΑΡΤΗΜΑ Α : Automation Scripts

run.sh – diploma.service

```
#!/bin/bash

# Exit immediately if a command exits with a non-zero status
set -e

RESET="\033[0m"         # Reset to default color
GREEN="\033[0;32m"      # Green for [WSS]
BLUE="\033[0;34m"       # Blue for [BACKEND]
MAGENTA="\033[0;35m"    # Magenta for [DOCKER-COMPOSE]
CYAN="\033[0;36m"       # Cyan for [WEB]
RED="\033[0;31m"        # Red for [CLEANUP, GIT]
YELLOW="\033[0;33m"     # Yellow for [NGINX]

# Function to handle Ctrl+C (SIGINT)
cleanup() {
   echo -e "\n${RED}Caught Ctrl+C! Cleaning up...\n${RESET}"
   sudo docker-compose down 2>&1 | while IFS= read -r line; do
      echo -e "${RED}[CLEANUP]:${RESET} $line"
   done
   echo -e "\n${RED}Cleanup complete. Exiting...\n${RESET}"
   exit 0
}
```

```
# Trap Ctrl+C (SIGINT) and call the cleanup function
trap cleanup SIGINT

run_backend_logs() {
    echo "Streaming backend logs..."
    cd /home/tmazarakis/diploma
    sudo docker container logs -f diploma_backend_1 | while IFS= read -r line; do
        echo -e "${BLUE}[BACKEND]:${RESET} $line"
    done
}

run_native_server_server(){
    echo "Running WebSocket server..."
    cd /home/tmazarakis/diploma/native_server
    npm run start 2>&1 | while IFS= read -r line; do
        echo -e "${GREEN}[WSS]:${RESET} $line"
    done
}

run_docker_compose(){
    echo "Starting Docker containers..."
    cd /home/tmazarakis/diploma
    sudo docker-compose up -d --build 2>&1 | while IFS= read -r line; do
        echo -e "${MAGENTA}[DOCKER-COMPOSE]:${RESET} $line"
    done
}

run_setup_nginx(){
    echo "Setting up nginx..."
    cd /home/tmazarakis/diploma
    sudo docker container exec -i diploma_nginx_1 sh -c "mkdir -p /dev/shm/hls && chown -R www-data:www-data /dev/shm/hls && ls -ld /dev/shm/hls" 2>&1 \
    | while IFS= read -r line; do
```

```
        echo -e "${YELLOW}[NGINX]:${RESET} $line"
    done
    sudo docker container exec -i diploma_nginx_1 sh -c "mkdir -p /dev/shm/keys && chown -R www-data:www-data /dev/shm/keys && ls -ld /dev/shm/keys" 2>&1 \
    | while IFS= read -r line; do
        echo -e "${YELLOW}[NGINX]:${RESET} $line"
    done
}

run_git_pull(){
    echo "Pulling latest changes..."
    cd /home/tmazarakis/diploma
    git pull 2>&1 | while IFS= read -r line; do
        echo -e "${RED}[GIT]:${RESET} $line"
    done
}
cd /home/tmazarakis/diploma
sudo docker-compose down

echo "Running Diploma Run Script..."
run_git_pull

# Step 1: Run WebSocket server
run_native_server_server &

# Step 2: Run Docker Compose
run_docker_compose

# Step 3: Run NGINX Setup
run_setup_nginx

# Step 4: Run backend logs in the background
run_backend_logs &
```

```
# Wait for background processes to finish
Wait
```

Bluetooth_service/run.sh – diploma-setup.service

```
#!/bin/bash

echo "Starting Bluetooth Setup Mode..."

runRFCOMMHandler(){
   # Run the rfcomm handler in the background.
   echo "Starting RFCOMM message handler..."
   exec sudo -E node bundled/index.js &
   rfcommHandlerPid=$!
   echo "Started RFCOMM handler (PID $rfcommHandlerPid)"
}

powerOnBluetooth(){
   echo "Enabling Bluetooth UART..."
   sudo systemctl enable hciuart
   sudo systemctl start hciuart
   echo "Enabling Bluetooth service..."
   sudo systemctl enable bluetooth
   sudo systemctl start bluetooth
   echo "Enabling Bluetooth adapter..."
   sudo rfkill unblock bluetooth
}

powerOffBluetooth(){
   echo "Disabling Bluetooth adapter..."
   bluetoothctl power off
   sudo systemctl stop bluetooth
   sudo systemctl disable bluetooth
```

```
    sudo systemctl stop hciuart
    sudo systemctl disable hciuart
}

cleanup(){
    echo "Cleaning up..."
    killProcess "$rfcommHandlerPid"
    killProcess "$rfcommWatchPid"
}

trapCallback(){
    echo "Trapped signal received. Cleaning up and exiting..."
    cleanup
    powerOffBluetooth
    echo "Exiting..."
    exit 1
}

#Used for debugging
hasInternetWifi(){
    echo "Checking WiFi connectivity..."

    maxAttempts=10
    attempts=0

    # Wait a few seconds for network interfaces to potentially come up
    while [[ $attempts -lt $maxAttempts ]]; do
        WIFI_CONNECTIVITY=$(sudo nmcli device status | awk '$2 == "wifi"' | awk '{print $3}')
        if [[ "$WIFI_CONNECTIVITY" == "connected" ]]; then
            echo "WiFi connectivity detected."
            return 0 # true
        fi
        attempts=$((attempts + 1))
```

```
        sleep 1
    done
    echo "WiFi connectivity not detected."
    return 1 #false
}

#Used for deployment
hasInternet(){
    echo "Checking network connectivity..."

    maxAttempts=10
    attempts=0

    # Wait a few seconds for network interfaces to potentially come up
    while [[ $attempts -lt $maxAttempts ]]; do
        CONNECTIVITY=$(sudo nmcli networking connectivity check)
        if [[ "$CONNECTIVITY" == "full" ]]; then
            echo "Network connectivity detected."
            return 0 # true
        fi
        attempts=$((attempts + 1))
        sleep 1
    done
    echo "Network connectivity not detected."
    return 1 #false
}

startRFCOMMWatcher(){
    echo "Starting RFCOMM watch process..."
    # sudo killall rfcomm
    sudo rfcomm release 0
    sudo rfcomm watch 0 1 &
    rfcommWatchPid=$!
```

```
    echo "Started RFCOMM watch (PID $rfcommWatchPid)"
}

killProcess(){
    local pid=$1
    if [[ -n "$pid" ]] && kill -0 "$pid" 2>/dev/null; then
        echo "Killing process $pid..."
        kill "$pid"
        wait "$pid" 2>/dev/null
        echo "Process $pid killed."
    else
        echo "Process $pid not running."
    fi
}

trap trapCallback SIGINT SIGTERM

echo "Bundling rfcomm handler..."
sudo npm run bundle

powerOnBluetooth

echo "Waiting for internet connection..."
rfcommHandlerPid=""
rfcommWatchPid=""
while true; do
    if hasInternetWifi; then
        echo "Internet connection detected. Exiting Bluetooth setup mode."
        killProcess "$rfcommHandlerPid"
        killProcess "$rfcommWatchPid"
        break
    else
        sleep 1
```

```
        if [[ -z "$rfcommWatchPid" ]] || ! kill -0 "$rfcommWatchPid" 2>/dev/null; then
            echo "No internet connection detected. Running RFCOMM watcher..."
            startRFCOMMWatcher
        else
            echo "RFCOMM watcher already running (PID $rfcommWatchPid)."
        fi
        sleep 1
        if [[ -z "$rfcommHandlerPid" ]] || ! kill -0 "$rfcommHandlerPid" 2>/dev/null; then
            echo "No internet connection detected. Running RFCOMM handler..."
            runRFCOMMHandler
        else
            echo "RFCOMM handler already running (PID $rfcommHandlerPid)"
        fi
        sleep 5
    fi
done

echo "Internet connection is detected!"

echo "Cleaning up..."
cleanup

powerOffBluetooth

echo "Starting ESP32 WiFi Sender Service..."
sudo /home/tmazarakis/diploma/esp.sh
echo "ESP32 WiFi Sender Service ended."

echo "Exiting..."

#Wait for the background processes to finish
wait
```

```
#If the shell script exits with code != 0, in will be rerun by systemd
exit 0
```

ΠΑΡΑΡΤΗΜΑ Β : WiFi credentials over Bluetooth

Rfcom_handler.ts

```
import {SerialPort} from 'serialport';

import {ReadlineParser} from '@serialport/parser-readline';

import fs from 'fs';

import {delay} from '../libs/process';

import {bluetoothAgentCleanup} from './bluetooth_agent';

import {Network} from '../network';

import {decryptMessage} from '../libs/decrypt';



export const startRFCommServiceHandler = async () => {

  console.log('Starting RFCOMM service Handler');

  const RFCOMM_DEVICE = '/dev/rfcomm0';



  console.log(`Checking id device RFCOMM exists: ${RFCOMM_DEVICE}`);

  let attempts = 0;

  console.log(`Waiting for device ${RFCOMM_DEVICE} to appear...`);

  while (!fs.existsSync(RFCOMM_DEVICE)) {

    if (attempts >= 3600) {

      console.log('1 hour exceeded, exiting');

      process.exit(1);
```

```
  }
  await delay(1000);
  attempts++;
}

console.log('Opening Serial Port');
const port = new SerialPort({
  path: RFCOMM_DEVICE,
  baudRate: 9600,
});

const parser = port.pipe(
  new ReadlineParser({delimiter: '\n', encoding: 'utf8'}),
);

port.on('open', () => {
  console.log('3) Opened Serial Port');
  console.log('4) Waiting for client connection via RFCOMM...');
});

parser.on('data', line => {
```

```
console.log(`(DATA) Received: ${line}`);
try {
 const payload = decryptMessage(JSON.parse(line));
 if (!payload) {
  onPortResponse({success: false, error: 'Invalid credentials format'});
  return;
 }
 const {ssid, password} = payload;
 Network.connectToWifi(ssid, password, (err, stdout, stderr) => {
  if (err) {
   onPortResponse({success: false, error: err.stderr || err.message});
   return;
  }
  const output = stdout.trim();
  const error = stderr.trim();
  if (
   output.toLowerCase().includes('error') ||
   error.toLowerCase().includes('error')
  ) {
   onPortResponse({success: false, error: output || error});
   return;
```

```
    }
    if (output.toLowerCase().includes('successfully')) {
      console.log(`(SUCCESS) Connected to WiFi: ${ssid}`);
      //* Save credentials
      fs.mkdirSync('/settings', {recursive: true});
      fs.writeFileSync(
        '/settings/wifi_credentials.json',
        JSON.stringify(payload),
      );
      onPortResponse({success: true});
      //* Wait 2 seconds before exiting
      new Promise(resolve => setTimeout(resolve, 2000)).then(() => {
        handlerCleanup();
      });
      return;
    }
    console.log(
      `(ERROR) Unknown error: ${err}, output: ${output}, error: ${error}`,
    );
  });
} catch (err) {
```

```
    onPortResponse({success: false, error: 'Invalid credentials format'});
    console.log(err);
  }
});

port.on('error', err => {
  console.log(`(ERROR) Serial Port Error: ${err}`);
});

port.on('close', () => {
  console.log('6) Serial Port Closed');
  handlerCleanup();
});

function onPortResponse({success, error}: {success: boolean; error?: {}}) {
  console.log(`(RESPONSE) Sending response: ${success} with error: ${error}`);
  try {
    if (success) {
      port.write(
        JSON.stringify({
          connected: success,
```

```
          localIp: Network.getLocalIP(),
          hostname: Network.getHostname(),
        }),
        'utf8',
      );
    } else {
      port.write(JSON.stringify({connected: success, error: error}), 'utf-8');
    }
    port.write('\n');
  } catch (err) {
    console.log(`(ERROR) Error sending response: ${err}`);
  }
}

function handlerCleanup() {
  bluetoothAgentCleanup();
  console.log('7) Cleaning up...');
  if (port && port.isOpen) {
    port.close(err => {
      if (err) {
        console.log(`(ERROR) Error closing port: ${err}`);
```

```
        } else {
          console.log('8) Port closed');
        }
        process.exit(0);
      });
    } else {
      process.exit(0);
    }
    //* To prevent port.close from hanging
    setTimeout(() => process.exit(1), 2000);
  }

  process.on('SIGINT', handlerCleanup);
  process.on('SIGTERM', handlerCleanup);
};
```

Bluetooth_agent.ts

```
import {ChildProcess, spawn} from 'child_process';
import {delay} from '../libs/process';

const CONFIRMATION_PATTERN_1 = '(yes/no):';
```

```
const CONFIRMATION_PATTERN_2 = '[agent]';

let bluetoothProcess: ChildProcess | null = null;

export function startBluetoothAgent() {

  console.log('Starting Bluetooth Agent...');



  bluetoothProcess = spawn('sudo', ['bluetoothctl'], {

    stdio: ['pipe', 'pipe', 'pipe'],

  });

  if (!bluetoothProcess) {

    console.log('Error starting Bluetooth Agent');

    process.exit(1);

  }

  bluetoothProcess.stdout?.setEncoding('utf8');

  bluetoothProcess.stderr?.setEncoding('utf8');



  //* Start agent

  bluetoothProcess.stdin?.write('power on\n');

  bluetoothProcess.stdin?.write('discoverable-timeout 3600\n');

  bluetoothProcess.stdin?.write('discoverable on\n');

  bluetoothProcess.stdin?.write('pairable on\n');

  bluetoothProcess.stdin?.write('agent on\n');
```

```
bluetoothProcess.stdin?.write('default-agent\n');


bluetoothProcess.stdout?.on('data', (data: Buffer) => {
  const lines = data.toString().split('\n');
  lines.forEach(_line => {
    const line = _line.trim();
    if (!line) {
      return;
    }
    console.log(`[Bluetooth Agent] ${line}`);


    if (
      line.toLowerCase().includes(CONFIRMATION_PATTERN_1) &&
      line.toLowerCase().includes(CONFIRMATION_PATTERN_2)
    ) {
      console.log('Confirmation received, sending yes...');
      try {
        bluetoothProcess?.stdin?.write('yes\n');
      } catch (err) {
        console.log('Error sending confirmation');
      }
```

```
    }
  });
});

bluetoothProcess.stderr?.on('data', data => {
  console.error(`[bluetoothctl ERR] ${data.toString().trim()}`);
});

// --- Handle process exit/close ---
bluetoothProcess.on('close', code => {
  console.log(`bluetoothctl process exited with code ${code}`);
  bluetoothProcess = null;
});

bluetoothProcess.on('error', err => {
  console.error(`Failed to start or communicate with bluetoothctl: ${err}`);
  bluetoothProcess = null;
});
}

export async function bluetoothAgentCleanup() {
```

```
if (!bluetoothProcess) {
  return;
}
try {
  console.log('Stopping Bluetooth Agent...');
  if (
    bluetoothProcess.stdin &&
    bluetoothProcess.stdin.writable &&
    !bluetoothProcess.stdin.destroyed
  ) {
    bluetoothProcess.stdin.write('exit\n');
    bluetoothProcess.stdin.end();
  }
  await delay(2000);
  console.log('Forcefully killing Bluetooth Agent...');
  bluetoothProcess?.kill('SIGTERM');
  bluetoothProcess = null;
  process.exit(0);
} catch (err) {
  console.log('Error stopping Bluetooth Agent');
}
```

```
}


process.on('SIGINT', bluetoothAgentCleanup);

process.on('SIGTERM', bluetoothAgentCleanup);
```

useBluetooth.tsx – Mobile app

```
import {useFocusEffect} from '@react-navigation/native';
import {useCallback, useEffect, useRef, useState} from 'react';
import BluetoothClassic, {
  BluetoothDevice,
  BluetoothEventSubscription,
  BluetoothNativeDevice,
} from 'react-native-bluetooth-classic';
import {
  BluetoothDeviceEvent,
  BluetoothDeviceReadEvent,
  BluetoothEventListener,
  StateChangeEvent,
} from 'react-native-bluetooth-classic/lib/BluetoothEvent';
import {encrypt} from '../libs/encrypt';
```

```
import {WifiCredentials, WifiCredentialsResponse} from '../types/wifi';


export const useBluetooth = () => {
  const [enabled, setEnabled] = useState(false);
  const onEnableRef = useRef<BluetoothEventListener<StateChangeEvent> | null>(
    null,
  );
  const [scanning, setScanning] = useState(false);
  const [connecting, setConnecting] = useState(false);
  const [connectedDevice, setConnectedDevice] =
    useState<BluetoothDevice | null>(null);
  const [waitingResponse, setWaitingResponse] = useState(false);
  const [systemDevice, setSystemDevice] = useState<BluetoothDevice | null>(
    null,
  );
  const [response, setResponse] = useState<WifiCredentialsResponse | null>(
    null,
  );
  const onDeviceConnectRef =
    useRef<BluetoothEventListener<BluetoothDeviceEvent> | null>(null);
  const onDeviceDisconnectRef =
```

```
  useRef<BluetoothEventListener<BluetoothDeviceEvent> | null>(null);
const onDeviceDiscoveredRef =
  useRef<BluetoothEventListener<BluetoothDeviceEvent> | null>(null);
const onDeviceReadRef =
  useRef<BluetoothEventListener<BluetoothDeviceReadEvent> | null>(null);
const deviceRedSubRef = useRef<BluetoothEventSubscription | null>(null);
const getBluetoothEnabled = async () => {
  const _enabled = await BluetoothClassic.isBluetoothEnabled();
  if (!enabled) {
    setEnabled(_enabled);
  }
};
const scanDevices = async () => {
  console.log('scanning for devices');
  setScanning(true);
  let maxTries = 5;
  while (maxTries > 0) {
    const connectedDevices = await BluetoothClassic.getConnectedDevices();
    console.log(connectedDevices);
    const foundDevice = connectedDevices.find(d => isSystemDevice(d));
    if (foundDevice) {
```

```
      setConnectedDevice(foundDevice);
      setSystemDevice(foundDevice);
      break;
    }
    console.log('scanning: ', maxTries);
    const devices = await BluetoothClassic.startDiscovery();
    if (devices.find(d => isSystemDevice(d))) {
      break;
    }
    maxTries--;
  }
  console.log('scanning complete');
  setScanning(false);
};
useFocusEffect(
  useCallback(() => {
    getBluetoothEnabled();
    scanDevices();
    //* Refs callbacls
    onEnableRef.current = (_event: StateChangeEvent) => {
      setEnabled(_event.enabled);
```

```
};
onDeviceDisconnectRef.current = (event: BluetoothDeviceEvent) => {
  console.log('ondevicedisconnection', event);
  const device = event as unknown as BluetoothDevice | null;
  if (device && isSystemDevice(device)) {
    console.log('disconnecting device');
    setConnectedDevice(null);
    setConnecting(false);
  }
};
onDeviceConnectRef.current = async (event: BluetoothDeviceEvent) => {
  console.log('ondeviceconnection', event);
  const device = event as unknown as BluetoothDevice | null;
  //* Handle system device read events
  if (device && isSystemDevice(device)) {
    const isConnected = await BluetoothClassic.isDeviceConnected(
      device.address,
    );
    if (!isConnected) {
      console.log('device is not connected.');
      return;
```

```
    }
    setConnectedDevice(device);
    setConnecting(false);
    //* Subscribe to device read events
  }
};
onDeviceDiscoveredRef.current = (event: BluetoothDeviceEvent) => {
  const device = event as unknown as BluetoothDevice | null;
  if (device && isSystemDevice(device)) {
    setSystemDevice(device);
  }
};
const sub4 = BluetoothClassic.onDeviceDiscovered(
  onDeviceDiscoveredRef.current,
);
//* Subscriptions
const sub1 = BluetoothClassic.onDeviceConnected(
  onDeviceConnectRef.current,
);
const sub2 = BluetoothClassic.onDeviceDisconnected(
  onDeviceDisconnectRef.current,
```

```
    );
    BluetoothClassic.onError(err => {
      console.log('Error: ', err);
    });
    const sub3 = BluetoothClassic.onStateChanged(onEnableRef.current);
    //* Cleanup
    return () => {
      sub1.remove();
      sub2.remove();
      sub3.remove();
      sub4.remove();
    };
  }, []),
);
useEffect(() => {
  if (connectedDevice) {
    console.log('Subscribing to device read events');
    onDeviceReadRef.current = (_event: BluetoothDeviceReadEvent) => {
      if (isSystemDevice(_event.device as BluetoothDevice)) {
        setWaitingResponse(false);
        console.log('ondeviceread', _event.data);
```

```
      try {
        const data = JSON.parse(_event.data) as WifiCredentialsResponse;
        let errStr: string | null = null;
        if (data.error) {
          errStr = data.error.substring(data.error.indexOf('Error:') + 7);
        } else if (data.localIp) {
        }
        setResponse({...data, error: errStr});
        console.log('data', data);
      } catch (err) {
        console.log('Error parsing data', err);
      }
    } else {
      console.log('unknown device read', _event.data);
    }
  };
  deviceRedSubRef.current = BluetoothClassic.onDeviceRead(
    connectedDevice.address,
    onDeviceReadRef.current,
  );
}
```

```
  return () => {
    deviceRedSubRef.current?.remove();
  };
}, [connectedDevice]);
useFocusEffect(
  useCallback(() => {
    if (systemDevice && !connectedDevice && !connecting) {
      if (systemDevice.bonded) {
        console.log('connecting');
        setConnecting(true);
        (async () => {
          while (true) {
            const d: BluetoothDevice | undefined = (await Promise.race([
              BluetoothClassic.connectToDevice(systemDevice.address),
              new Promise(resolve => setTimeout(resolve, 3000)),
            ])) as BluetoothDevice | undefined;
            console.log('connected: ', d);
            if (d) {
              const isConnected = await d?.isConnected();
              console.log('Connected: ', isConnected);
              if (isConnected) {
```

```
          setConnectedDevice(systemDevice);
        }
        setConnecting(false);
        break;
      }
    }
  })();
} else {
  console.log('bonding');
  setConnecting(true);
  (async () => {
    while (true) {
      console.log('Pairing with device', systemDevice.address);
      const device = await BluetoothClassic.pairDevice(
        systemDevice.address,
      );
      console.log(device.bonded);
      if (device.bonded) {
        setSystemDevice(device);
        const c = await device.connect();
        if (c) {
```

```
            setConnectedDevice(device);
          }
          break;
        }
      }
      setConnecting(false);
    })();
  }
 }
 }, [systemDevice, connectedDevice, connecting]),
);
const writeToDevice = async ({password, ssid}: WifiCredentials) => {
 if (!systemDevice) {
  console.log('No system device');
  return false;
 }
 if (!connectedDevice) {
  console.log('No device connected');
  return false;
 }
 const payload = encrypt({
```

```
  ssid: ssid,
  password: password,
});
if (!payload) {
  console.log('Encryption failed');
  return false;
}
if (!('write' in connectedDevice)) {
  const d = connectedDevice as BluetoothNativeDevice;
  console.log('Device does not support write');
  setWaitingResponse(true);
  const address = d.address;
  let success = BluetoothClassic.writeToDevice(
    address,
    JSON.stringify(payload),
  );
  success = BluetoothClassic.writeToDevice(address, '\n');
  console.log('Write success: ', success);
  return success;
} else {
  setWaitingResponse(true);
```

```
    const isConnected = await systemDevice.isConnected();
    if (!isConnected) {
      console.log('Device is not connected');
      return false;
    }
    let success = await connectedDevice.write(JSON.stringify(payload));
    success = await connectedDevice.write('\n');
    console.log('Write success: ', success);
    return success;
  }
};
return {
  enabled,
  connectedDevice,
  systemDevice,
  connecting,
  scanning,
  writeToDevice,
  waitingResponse,
  response,
};
```

```
};


const isSystemDevice = (device: BluetoothDevice) => {
  if (device.address === process.env.BLUETOOTH_ADDRESS) {
    return true;
  }
  return device.name.toLowerCase().includes('trpia');
};
```

ΠΑΡΑΡΤΗΜΑ C : Smart Recognition

Handler.ts

```
import Process from '../managers/process';

import {ServiceMessage} from '../types/ws';

import {audioDeviceInfoHandler} from './devices/audioDeviceInfoHandler';

import {

  cameraDeviceStatusCloseHandler,

  cameraDeviceStatusInfoHandler,

} from './devices/cameraDeviceStatusHandler';

import {faceEmbeddingsHandler} from './recognition/faceEmbeddingsHandler';

import {

  faceRecognitionImageChunkHandler,

  jpegImageChunkHandler,

} from './recognition/jpegImageHandler';

import {wavAudioChunkHandler} from './recognition/wavAudioHandler';



export const handleServiceMessage = (msg: ServiceMessage) => {

  if (msg.action === 'stop') {

    Process.killProcess(msg.service);

    return;

  }
```

```
if (msg.action === 'start') {
  switch (msg.service) {
    case 'rtmp_stream':
      Process.startProcess({
        service: msg.service,
        withAudio: msg.state === 'audio-on',
        stdout: d => {},
        stderr: d => {},
      });
      break;
    case 'image_recognition':
      Process.startProcess({
        service: msg.service,
        stdout: jpegImageChunkHandler,
      });
      break;
    case 'face_recognition':
      Process.startProcess({
        service: msg.service,
        stdout: faceRecognitionImageChunkHandler,
      });
```

```
  break;
case 'audio_recognition':
  Process.startProcess({
    service: msg.service,
    stdout: wavAudioChunkHandler,
  });
  break;
case 'face_embeddings':
  Process.startProcess({
    service: msg.service,
    person: msg.state,
    onClose: code => faceEmbeddingsHandler({code, person: msg.state}),
    stdout: data => {
      console.log(data.toString());
    },
    stderr: data => {
      console.log(data.toString());
    },
    onError: () => faceEmbeddingsHandler({code: null}),
  });
  break;
```

```
    case 'camera_status':
      Process.startProcess({
        service: msg.service,
        stdout: cameraDeviceStatusInfoHandler,
        onClose: code => cameraDeviceStatusCloseHandler(code),
        onError: () => cameraDeviceStatusCloseHandler(null),
      });
      break;
    case 'audio_info':
      Process.startProcess({
        service: msg.service,
        stdout: audioDeviceInfoHandler,
      });
      break;
    default:
      console.log('Unknown service: ', msg.service);
      break;
  }
 }
};
```

Process.ts

```
import {ServiceTypeType} from '../types/services';

import {ChildProcess, spawn} from 'child_process';

import kill from 'tree-kill';

import {getServiceCommand} from '../constants/process';

import {StartProcessArgs} from '../types/process';

import Subscription from './subscription';



const processStore = new Map<ServiceTypeType, ChildProcess>();



const audioDeviceStore: Pick<StartProcessArgs, 'cardNumber' | 'deviceNumber'> =

  {

    cardNumber: undefined,

    deviceNumber: undefined,

  };



const addDeviceToStore = ({

  cardNumber,

  deviceNumber,

}: Pick<StartProcessArgs, 'cardNumber' | 'deviceNumber'>) => {

  audioDeviceStore.cardNumber = cardNumber;
```

```
  audioDeviceStore.deviceNumber = deviceNumber;
};

const init = () => {
  processStore.clear();
  audioDeviceStore.cardNumber = undefined;
  audioDeviceStore.deviceNumber = undefined;
};

const startProcess = (args: StartProcessArgs): boolean => {
  try {
    if (processStore.has(args.service)) {
      console.log('Process already started');
      return false;
    }
    const cmd = getServiceCommand({
      ...args,
      cardNumber: args.cardNumber ?? audioDeviceStore.cardNumber,
      deviceNumber: args.deviceNumber ?? audioDeviceStore.deviceNumber,
    });
    if (!cmd) {
```

```
  console.log('Invalid service, command not found');
  return false;
}

//* Validation checks for services that rely on other, to prevent errors
if (!isRelyingProcessOpen(args.service)) {
  console.log(`Service ${args.service} is relying other process, ignoring`);
  return false;
}

console.log('Starting process: ', cmd);

const process = spawn('sh', ['-c', cmd]);
Subscription.notifySubscribers({
  topic: 'service_status',
  message: {service: args.service, action: 'start'},
});
processStore.set(args.service, process);

//* Handle process outputs
if (args.stdout && typeof args.stdout === 'function') {
```

```
      console.log('asigning stdout to ', args.service);
      process.stdout.on('data', args.stdout);
    }
    if (args.stderr && typeof args.stderr === 'function') {
      process.stderr.on('data', args.stderr);
    }
    process.on('close', code => {
      console.log(`[process] Stream closed with code ${code}`);
      if (args.onClose && typeof args.onClose === 'function') {
        args.onClose(code);
      }
      killProcess(args.service);
    });
    return true;
  } catch (err) {
    args.onError?.(err);
    console.log(args.service, err);
    return false;
  }
};
```

```
const killProcess = (service: ServiceTypeType) => {
  if (processStore.has(service)) {
    const process = processStore.get(service);
    //* In case of rtmp_stream, kill all the processes at are relying on it
    if (process && service === 'rtmp_stream') {
      killProcess('face_recognition');
      killProcess('image_recognition');
      killProcess('audio_recognition');
    }
    if (process && service === 'image_recognition') {
      killProcess('face_recognition');
    }
    if (process) {
      try {
        if (process.pid) {
          kill(process.pid);
          console.log('Process killed successfully');
          Subscription.notifySubscribers({
            topic: 'service_status',
            message: {service: service, action: 'stop'},
          });
```

```
        }
      } catch (err) {}
      processStore.delete(service);
    }
  }
};

const getProcess = (service: ServiceTypeType) => {
  if (processStore.has(service)) {
    return processStore.get(service);
  }
  return null;
};

/**
 *
 * @param param0 service
 * @param param1 buffer
 * @returns true if process was found and stdin is writable, false if not found or not writable
 * * Writes to the stdin of the process, only if it exists
 */
```

```
const writeToProcess = ({
  service,
  buffer,
}: {
  service: ServiceTypeType;
  buffer: Buffer;
}): boolean => {
  const process = getProcess(service);
  if (
    process &&
    process.pid &&
    process.stdin &&
    !process.stdin.destroyed &&
    process.stdin.writable
  ) {
    process.stdin.write(buffer);
    return true;
  }
  return false;
};
```

```
const Process = {
  init,
  startProcess,
  killProcess,
  writeToProcess,
  getProcess,
  addDeviceToStore,
};

export default Process;

const isRelyingProcessOpen = (service: ServiceTypeType) => {
  if (service === 'image_recognition' && !processStore.has('rtmp_stream')) {
    return false;
  }
  if (service === 'audio_recognition' && !processStore.has('rtmp_stream')) {
    return false;
  }
  if (service === 'face_recognition' && !processStore.has('rtmp_stream')) {
    return false;
  }
```

```
  return true;

};
```

wavAudioChunkHandler.ts

```
import Subscription from '../../managers/subscription';
import {getAudioRecognitionPrediction} from '../../predictions/audioRecognition';


//* bytes per sample = 2 (16-bit per sample ffmpeg output for WAV format - 16-bit PCM bit depth, float32 [-1,1])
//* -ar = 16000 => 16000 samples per second
//* -ac = 1 => 1 channel
//* bytes per second = samples per second * bytes per sample * channels = 16000 * 2 * 1 = 32000
const YAMNET_PROCESS_WINDOW = 0.96;
const CHUNK_SIZE = 16000 * 2 * 1 * YAMNET_PROCESS_WINDOW;

let audioBuffer = Buffer.alloc(0);
export const wavAudioChunkHandler = async (chunk: Buffer) => {
  audioBuffer = Buffer.concat([audioBuffer, chunk]);
  if (audioBuffer.length >= CHUNK_SIZE) {
    const audioData = audioBuffer.subarray(0, CHUNK_SIZE);
    audioBuffer = audioBuffer.subarray(CHUNK_SIZE);

    const {audioFloat, predictions, spectogram} =
      await getAudioRecognitionPrediction(audioData);
    if (predictions.length > 0) {
      Subscription.notifySubscribers({
        topic: 'prediction',
        message: predictions,
      });
    }
    if (audioFloat.length > 0) {
      Subscription.notifySubscribers({
        topic: 'audio_chunks',
```

```
      message: audioFloat,
    });
  }
 }
};
```

audioRecognition.ts

```
import {Models} from '../managers/models';
import {Prediction} from '../types/predictions';
import tf from '@tensorflow/tfjs-node';
import audioClasses from '../constants/audio_classes.json';

const EXCLUDED_CLASSES = [
 410, 411, 507, 508, 514, 515, 509, 510, 125, 490, 121, 376,
];

type RecognitionResult = {
 predictions: Prediction[];
 audioFloat: Float32Array;
 spectogram: number[][];
};

export const getAudioRecognitionPrediction = async (
 audioBuffer: Buffer,
): Promise<RecognitionResult> => {
 //* Audio buffer data is in 16-bit signed integer format
 const audioData = new Int16Array(audioBuffer);
 //* Normalize to float32 [-1,1] as yamnet expects
 const audioFloat = new Float32Array(audioData.length);
 for (let i = 0; i < audioData.length; i++) {
  audioFloat[i] = audioData[i] / 32768.0; // 32768 = 2^15 -> [-1,1]
 }
 const model = Models.getModel('yamnet');
```

```
if (!model) {
  return {predictions: [], audioFloat: audioFloat, spectogram: []};
}
try {
  const inputTensor = tf.tensor1d(audioFloat);

  const [scores, embeddings, spectrogram] = model.predict(
    inputTensor,
  ) as tf.Tensor[];
  const scoresArray = scores.mean(0).dataSync();

  //* Get top 5 predictions over > 0.4 score
  const topScores: Prediction[] = Array.from(scoresArray)
    .map((score, index) => ({score, index}))
    .filter(({index}) => !EXCLUDED_CLASSES.includes(index))
    .filter(({score}) => score > 0.4)
    .sort((a, b) => b.score - a.score)
    .slice(0, 5)
    .map(({score, index}) => ({
      score,
      class: audioClasses?.[index]?.display_name ?? 'not_found',
      type: 'audio',
    }));

  // console.log('topScores', topScores);

  const modelSpectogram = [...(spectrogram.arraySync() as number[][])];

  //* Clear tensors
  inputTensor.dispose();
  scores.dispose();
  embeddings.dispose();
  spectrogram.dispose();
```

```
    return {
      predictions: topScores,
      audioFloat: audioFloat,
      spectogram: modelSpectogram,
    };
  } catch (err) {
    console.log('Error processing audio', err);
    return {predictions: [], audioFloat: audioFloat, spectogram: []};
  }
};
```

objectRecognition.ts

```
import {ObjectRecognitionPrediction} from '../types/predictions';
import {Models} from '../managers/models';
import tf from '@tensorflow/tfjs-node';

const ALLOWED_CLASSES = [
  'person',
  'cat',
  'dog',
  'backpack',
  'handbag',
  'suitcase',
  'bottle',
  'knife',
];

export const getObjectRecognitionPrediction = async (
  frame: Buffer,
): Promise<ObjectRecognitionPrediction[]> => {
  const model = Models.getModel('coco_ssd');
  if (!model) {
```

```
    return [];
  }

  try {
    //* Convert to Uint8Array as coco expects to decode an image from.
    const bytes = new Uint8Array(frame);
    let inputTensor = tf.node.decodeImage(bytes) as tf.Tensor3D;

    //* If the 3rd dimension of the tensor has 4 channels, remove the alpha channel
    if (inputTensor.shape[2] === 4) {
      //* [0,0,0] for each channel
      //* -1 = keep the whole tensor
      //* 3 = keep the 3 first channels
      inputTensor = inputTensor.slice([0, 0, 0], [-1, -1, 3]);
    }
    let output = await model.detect(inputTensor);

    //* Filter out unwanted classes
    output = output.filter(p => ALLOWED_CLASSES.includes(p.class));
    return output.map(p => ({...p, type: 'image'}));
  } catch (err) {
    console.log('Error processing image', err);
    return [];
  }
};
```

ΠΑΡΑΡΤΗΜΑ D : ESP32 Code

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <Arduino.h>
#include <esp_bt.h>
#include <esp_system.h>
#include <string.h>
#include <Preferences.h>
#include <WebServer.h>

//* WiFi AP
WebServer server(80);
Preferences preferences;
//* WiFi AP Credentials
const String SoftApPassword = "ZGIwNTJiMDgtYTE0Ny00ZTA2LWI5YjYtYzc2OTY2N2NlMWM0";
const String SoftApName = "ESP32-AP-" + String((uint32_t)ESP.getEfuseMac(), HEX);
#define WIFIPreferencesKey "wifi-credentials"

#define motionSensorPin GPIO_NUM_14
#define soundSensorDigitalPin GPIO_NUM_25
#define ledPin GPIO_NUM_13

#define WakePinMask (1ULL << motionSensorPin)

// Cooldown times
const int sleepTime = 60;

// MQTT broker details
#define Token "NDMwYmFkZmItM2VlYy00NGQxLWFlMWUtNGUzYTFkOWJmNjIw"
const char* mqtt_server = "trpia.local";
const int mqtt_port = 1883;
```

```
const char* mosquittoUsername = "admin";
const char* mosquittoPassword = "diploma123";
const char* mqtt_topic_motion = "sensors/motion";
const char* mqtt_topic_sound = "sensors/sound";
const char* mqtt_topic_ping = "sensors/ping";
PubSubClient client(espClient);

// Create WiFi and MQTT clients
WiFiClient espClient;

// Function to connect to WiFi
bool connectToWifi() {
  Serial.println("Getting Saved Credentials");
  //* Open preferences in read & write mode.
  preferences.begin("wifi", false);
  String ssid = preferences.getString("ssid", "");
  String password = preferences.getString("password", "");
  Serial.println(ssid);
  Serial.println(password);
  preferences.end();
  //* If no credentials are stored, return to open AP.
  if (ssid.length() == 0 || password.length() == 0) {
    Serial.println("Saved Credentials not found.");
    return false;
  }
  //* If credentials are stored, Try and connect to WiFi.
  Serial.println("Connecting to WiFi via credentials...");
  Serial.println(ssid);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  int maxTries = 60;
  while (maxTries > 0) {
    delay(1000);
```

```
    if (WiFi.status() == WL_CONNECTED) {
      Serial.println("\nWiFi connected!");
      Serial.print("IP address: ");
      Serial.println(WiFi.localIP());
      return true;
    }
    Serial.print(WiFi.status());
    maxTries--;
  }
  Serial.println("Connection to WiFi timedout. after 30 seconds...");
  return false;
}

// Function to connect to MQTT broker
void connectToMQTT() {
  int totalAttempts = 0;
  while (!client.connected() && totalAttempts < 10) {
    totalAttempts++;
    Serial.print("Connecting to MQTT broker...");
    if (client.connect("ESP32Client", mosquittoUsername, mosquittoPassword)) {  // Client ID
      Serial.println("connected!");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" retrying in 2 seconds...");
      delay(2000);
    }
  }
}

void openClients() {
  Serial.println("Opening Clients");
  //* Check if the WiFi can be connected.
```

```
  //* If not, remove credentials are start to allow the system to go in to AP.
  //* This is to avoid blocking forever on a once connectful internet, now gone.
  int maxTries = 4;
  while (maxTries > 0) {
    bool connected = connectToWifi();
    if (connected) {
      client.setServer(mqtt_server, mqtt_port);
      if (!client.connected()) {
        connectToMQTT();
      }
      return;
    }
    Serial.println("Could not connect to wifi. retrying...");
    Serial.print("Tries left: ");
    Serial.println(maxTries);
    maxTries--;
  }
  // Serial.println("Wifi is no longer connectable. Removing credentials and restarting...");
  // removeCredentials();
  delay(1000);
  ESP.restart();
}

void closeClients() {
  Serial.println("Closing MQTT Client");
  client.disconnect();
  Serial.println("Closing WiFi");
  WiFi.disconnect(true);
  WiFi.mode(WIFI_OFF);
}

bool hasCredentials() {
  Serial.println("Checking if credentials are saved");
```

```
  preferences.begin("wifi", false);
  String ssid = preferences.getString("ssid", "");
  String password = preferences.getString("password", "");
  Serial.println(ssid);
  Serial.println(password);
  preferences.end();
  return ssid.length() > 0 && password.length() > 0;
}

void removeCredentials() {
  preferences.begin("wifi", false);
  preferences.clear();
  preferences.end();
}

void onWiFiCredentialsReceived() {
  String body = server.arg("plain");
  StaticJsonDocument<200> jsonDoc;
  DeserializationError error = deserializeJson(jsonDoc, body);
  if (error) {
    server.send(400, "text/html", "Invalid JSON");
    delay(100);
    return;
  }
  const char* ssid = jsonDoc["ssid"];
  const char* password = jsonDoc["password"];
  //* Save to preferences
  preferences.begin("wifi", false);
  preferences.putString("ssid", ssid);
  preferences.putString("password", password);
  preferences.end();
  //* Log Debug
  Serial.println("Saved Credentials to preferences:");
```

```
  Serial.println(ssid);
  Serial.println(password);
  server.send(200, "text/html", "Credentials saved. Rebooting...");
  server.client().stop();
  delay(100);
  //* Restart the ESP to take effect.
  bool canConnect = connectToWifi();
  if (!canConnect) {
    Serial.println("Removing saved wifi credentials");
    removeCredentials();
  }
  Serial.println("Restarting ESP for changes to take effect");
  delay(1000);
  ESP.restart();
}

const String SoftApPassword = "ZGIwNTJiMDgtYTE0Ny00ZTA2LWI5YjYtYzc2OTY2N2NlMWM0";
const String SoftApName = "ESP32-AP-" + String((uint32_t)ESP.getEfuseMac(), HEX);
void openAsAccessPoint() {
  Serial.println("Setting up esp32 as AP");
  //* WiFi config
  WiFi.mode(WIFI_AP);
  WiFi.softAP(SoftApName, SoftApPassword);
  //* Address
  Serial.println("AP Ip Address: ");
  Serial.println(WiFi.softAPIP());
  //* HTTP Server
  server.on("/wifi", HTTP_POST, onWiFiCredentialsReceived);
  server.begin();
  Serial.println("Server started!");
}
```

```
// Setup function
void setup() {
  Serial.begin(115200);
  delay(1000);

  //* Check if credentials are saved to non volatile flash.
  if (!hasCredentials()) {
    openAsAccessPoint();
    return;
  }

  //* Continue normal operation with WiFi set and WiFi connection available.
  btStop();
  pinMode(ledPin, OUTPUT);
  pinMode(soundSensorDigitalPin, INPUT);
  pinMode(motionSensorPin, INPUT)
    digitalWrite(ledPin, LOW);

  uint64_t wakeup_pin_mask = esp_sleep_get_ext1_wakeup_status();
  esp_sleep_wakeup_cause_t cause = esp_sleep_get_wakeup_cause();

  const char* modelName = ESP.getChipModel();

  char str[50];
  switch (cause) {
    case ESP_SLEEP_WAKEUP_EXT0:
      Serial.println("Woke up from SOUND sensor!");
      strcat(str, modelName);
      strcat(str, "-");
      strcat(str, "SOUND");
      onSensorSensed("sound", mqtt_topic_sound, str);
      break;
    case ESP_SLEEP_WAKEUP_EXT1:
```

```
  while (digitalRead(motionSensorPin) == HIGH) {
    delay(25);
  }
  if (wakeup_pin_mask & (1ULL << motionSensorPin)) {
    Serial.println("Woke up from MOTION sensor!");
  } else {
    Serial.println("unknown wakeup in EXT1");
    break;
  }
  strcat(str, modelName);
  strcat(str, "-");
  strcat(str, "MOTION");
  onSensorSensed("motion", mqtt_topic_motion, str);
  break;
case ESP_SLEEP_WAKEUP_TIMER:
  Serial.println("Wakeup caused by timer");
  //* Provide list of sensors type e.g. "motion,sound"
  strcat(str, modelName);
  strcat(str, "-");
  strcat(str, "MOTION");
  strcat(str, ",");
  strcat(str, modelName);
  strcat(str, "-");
  strcat(str, "SOUND");
  onSensorSensed("motion,sensor", mqtt_topic_ping, str);
  break;
default:
  Serial.print("Wakeup caused by unknown reason: ");
  Serial.println(cause);
  digitalWrite(ledPin, HIGH);
  delay(500);
  digitalWrite(ledPin, LOW);
  break;
```

```
  }
}

void onSensorSensed(const char* status, const char* topic, const char* name) {
  digitalWrite(ledPin, HIGH);

  openClients();
  sendMessage(status, topic, name);

  delay(2000);
  closeClients();
  digitalWrite(ledPin, LOW);
}

void sendMessage(const char* status, const char* topic, const char* name) {
  Serial.print("sending json message: ");
  Serial.println(status);
  // Create a JSON object
  StaticJsonDocument<200> jsonDoc;

  // Add key-value pairs to the JSON object
  jsonDoc["sensorName"] = name;
  jsonDoc["status"] = status;
  jsonDoc["token"] = Token;

  // Serialize the JSON object to a string
  char jsonBuffer[200];
  serializeJson(jsonDoc, jsonBuffer);

  // Publish the JSON message to the MQTT topic
  Serial.print("Publishing JSON message: ");
  Serial.println(jsonBuffer);
  client.publish(topic, jsonBuffer);
```

```
}

void setupPowerEfficiency() {
  //* RTC pin Wake up
  esp_sleep_enable_ext1_wakeup(WakePinMask, ESP_EXT1_WAKEUP_ANY_HIGH);
  esp_sleep_enable_ext0_wakeup(soundSensorDigitalPin, LOW);

  //* Timer Wake up
  esp_sleep_enable_timer_wakeup(sleepTime * 1000000ULL);

  Serial.println("Entering deep sleep...");
  Serial.flush();
  esp_deep_sleep_start();
}

void loop() {
  server.handleClient();
  delay(10);
}
```

ΠΑΡΑΡΤΗΜΑ Ε : Configuration Files

docker-compose.yml

```
version: '3.9'

services:
 nginx:
   image: tiangolo/nginx-rtmp:latest-2025-04-07
   ports:
     - '80:80'
     - '443:443'
     - '1935:1935'
   volumes:
```

```
    - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro
    - ./nginx/ssl:/etc/nginx/ssl:ro
    - ./nginx/web-app/dist:/var/www/html
    - recordings:/recordings
    - thumbnails:/thumbnails
    - /etc/localtime:/etc/localtime:ro
    - /etc/timezone:/etc/timezone:ro
  depends_on:
    - backend
mosquitto:
  image: eclipse-mosquitto:2.0.21
  ports:
    - '1883:1883'
    - '9001:9001'
  volumes:
    - ./mosquitto/mosquitto.conf:/mosquitto/config/mosquitto.conf:ro
    - ./mosquitto/passwd:/mosquitto/config/passwd:ro
    - /etc/localtime:/etc/localtime:ro
    - /etc/timezone:/etc/timezone:ro
mongodb:
  image: mongo:8.0.6
  ports:
    - '27017:27017'
  environment:
    - MONGO_INITDB_ROOT_USERNAME=tmazarakis
    - MONGO_INITDB_ROOT_PASSWORD=IHCk1KZHfKhvSqo
  volumes:
    - mongo_data:/data/db
    - ./mongodb:/docker-entrypoint-initdb.d
    - /etc/localtime:/etc/localtime:ro
    - /etc/timezone:/etc/timezone:ro
backend:
  build:
```

```
    context: ./backend
    dockerfile: Dockerfile
  ports:
    - '3000:3000'
  devices:
    - '/dev/ttyAMA0:/dev/ttyAMA0'
  stdin_open: true
  tty: true
  volumes:
    - settings_volume:/settings
    - ./serviceAccountKey.json:/serviceAccountKey.json
    - recordings:/recordings
    - thumbnails:/thumbnails
    - /face_images:/face_images
    - /etc/localtime:/etc/localtime:ro
    - /etc/timezone:/etc/timezone:ro
  environment:
    - MQTT_ADDRESS=mqtt://mosquitto:1883
    - WS_ADDRESS=ws://mosquitto:9001
    - MQTT_USERNAME=admin
    - MQTT_PASSWORD=diploma123
    - HTTP_PORT=3000
    -
MONGO_URI=mongodb://tmazarakis:IHCk1KZHfKhvSqo@mongodb:27017/?authSource=admin
    - SMS_SERIAL_PORT=/dev/ttyAMA0
  depends_on:
    - mosquitto
    - mongodb

volumes:
 mongo_data:
 settings_volume:
 recordings:
```

thumbnails:

nginx.conf

```
user www-data;
worker_processes auto;
worker_cpu_affinity auto;
pid /run/nginx.pid;
error_log /var/log/nginx/error.log;
include /etc/nginx/modules-enabled/*.conf;

rtmp_auto_push on;

events {
  worker_connections 768;
}

http {
  ##
  # Basic Settings
  ##

  sendfile on;
  tcp_nopush on;
  types_hash_max_size 2048;
  server_tokens build; # Recommended practice is to turn this off

  # server_names_hash_bucket_size 64;
  # server_name_in_redirect off;

  include /etc/nginx/mime.types;
  default_type application/octet-stream;

  ##
```

```
# SSL Settings
##

ssl_protocols TLSv1.2 TLSv1.3;
ssl_prefer_server_ciphers on;
ssl_ciphers 'ECDHE+AESGCM:CHACHA20';
ssl_session_cache shared:SSL:10m;
ssl_session_timeout 1d;
ssl_session_tickets off;

##
# Logging Settings
##

access_log /var/log/nginx/access.log;

##
# Gzip Settings
##

gzip on;

server {
    listen 443 ssl;
    server_name trpia.local localhost 127.0.0.1;

    ssl_certificate /etc/nginx/ssl/trpia.local+2.pem;
    ssl_certificate_key /etc/nginx/ssl/trpia.local+2-key.pem;

    # Protection headers
    add_header Strict-Transport-Security "max-age=63072000; includeSubDomains; preload" always;
    add_header X-Frame-Options DENY always;
    add_header X-Content-Type-Options nosniff always;
```

```
    add_header X-XSS-Protection "1; mode=block" always;
    add_header Cross-Origin-Opener-Policy "same-origin-allow-popups";
    add_header Cross-Origin-Embedder-Policy "unsafe-none";

    #* Reverse Proxy backend: HTTP Traffic
    location /api/ {
        # Allow CORS requests from localhost:5173 and 127.0.0.1:5173, for development purposes
        set $cors '';
        if ($http_origin ~* ^(http://localhost:5173|http://127.0.0.1:5173|https://trpia.local)$) {
            set $cors $http_origin;
        }
        #* Handles preflight requests
        if ($request_method = 'OPTIONS') {
            add_header 'Access-Control-Allow-Origin' '$cors' always;
            add_header 'Access-Control-Allow-Methods' 'GET, POST, PUT, DELETE, OPTIONS' always;
            add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Range,Authorization,Your-Custom-Header-If-Any' always;
            add_header 'Access-Control-Allow-Credentials' 'true' always;
            add_header 'Access-Control-Max-Age' 1728000 always; # 20 days
            add_header 'Content-Type' 'text/plain charset=utf-8' always;
            add_header 'Content-Length' 0 always;
            return 204; # HTTP 204 No Content - standard for successful preflight
        }
        add_header 'Access-Control-Allow-Origin' '$cors' always;
        add_header 'Access-Control-Allow-Credentials' 'true' always;

        #* Reverse Proxy
        proxy_pass http://backend:3000/;

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

```
        proxy_set_header X-Forwarded-Proto $scheme; # Tells backend it was an HTTPS request originally
    }

    #* Reverse Proxy backend websocket: HTTP Traffic
    location /ws/ {
        set $cors '';
        if ($http_origin ~* ^(http://localhost:5173|http://127.0.0.1:5173|https://trpia.local)$) {
            set $cors $http_origin;
        }
        #* Handles preflight requests
        if ($request_method = 'OPTIONS') {
            add_header 'Access-Control-Allow-Origin' '$cors' always;
            add_header 'Access-Control-Allow-Methods' 'GET, OPTIONS' always; # WebSockets use GET for handshake
            add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Range,Authorization,Your-Custom-Header-If-Any' always;
            add_header 'Access-Control-Allow-Credentials' 'true' always;
            add_header 'Access-Control-Max-Age' 1728000 always;
            add_header 'Content-Type' 'text/plain charset=UTF-8' always;
            add_header 'Content-Length' 0 always;
            return 204;
        }
        add_header 'Access-Control-Allow-Origin' '$cors' always;
        add_header 'Access-Control-Allow-Credentials' 'true' always;

        proxy_pass http://mosquitto:9001/;

        proxy_http_version 1.1;
        #* Upgrades connections from http to websocket.
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
```

```
}

# Static files that exist inside the nginx container
location /recordings {
    alias /recordings/;
    autoindex on;
    add_header Content-Type video/mp4;

    types {
        video/mp4 mp4;
    }

    add_header Cache-Control no-cache;
    add_header Access-Control-Allow-Origin "*" always;
    add_header Access-Control-Allow-Methods "GET, OPTIONS" always;
    add_header Access-Control-Allow-Headers "*" always;
    etag on;
}

location /thumbnails {
    alias /thumbnails/;
    autoindex on;
    add_header Content-Type image/jpeg;

    types {
        image/jpeg jpg jpeg;
    }

    add_header Cache-Control no-cache;
    add_header Access-Control-Allow-Origin "*" always;
    add_header Access-Control-Allow-Methods "GET, OPTIONS" always;
    add_header Access-Control-Allow-Headers "*" always;
    etag on;
```

```
}

# HLS stream files that exist inside the RAM disk
location /hls {
    types {
        application/dash+xml mpd;
        application/vnd.apple.mpegurl m3u8;
        video/mp2t ts;
    }
    root /dev/shm;
    add_header Cache-Control no-cache;
    add_header Access-Control-Allow-Origin "*" always;
    add_header Access-Control-Allow-Methods "GET, OPTIONS" always;
    add_header Access-Control-Allow-Headers "*" always;
    etag on;
}

#* ENCRYPTED: HLS stream files that exist inside the RAM disk
location /keys{
    root /dev/shm;
    add_header Cache-Control no-cache;
    add_header Access-Control-Allow-Origin "*" always;
    add_header Access-Control-Allow-Methods "GET, OPTIONS" always;
    add_header Access-Control-Allow-Headers "*" always;
    etag on;
}

#* Web Application files
location / {
    add_header Cross-Origin-Opener-Policy "same-origin-allow-popups";
    add_header Cross-Origin-Embedder-Policy "unsafe-none";

    root /var/www/html;
```

```
        index index.html;
        try_files $uri $uri/ /index.html;
    }
  }

  server {
    listen 80;
    server_name _;

    # Redirect all traffic to HTTPS
    location / {
      return 301 https://$host$request_uri;
    }
  }

  include /etc/nginx/conf.d/*.conf;
  include /etc/nginx/sites-enabled/*;
}

rtmp {
  server {
    listen 1935;

    timeout 60s;
    ping 30s;
    ping_timeout 10s;

    buflen 5s;

    application live {
      live on;
      hls on;
      hls_path /dev/shm/hls;
```

```
        hls_fragment 1;
        hls_playlist_length 3s;
        hls_cleanup on;
        hls_fragment_naming timestamp;
        hls_fragment_slicing aligned;
        wait_key on;
        wait_video on;
        drop_idle_publisher 20s;
        allow play all;
        interleave on;

        #* ENCRYPTION: Enable HLS keys
        hls_keys on;
        hls_key_path /dev/shm/keys;
        hls_key_url https://trpia.local/keys/;
        hls_fragments_per_key 10;
    }
  }
}
```