ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
Πρόγραμμα Μεταπτυχιακών Σπουδών
Επιστήμη και Τεχνολογία της Πληροφορικής και των Υπολογιστών
Ειδίκευση Υλικού και Υπολογιστικών Συστημάτων

Διπλωματική Εργασία

Σχεδίαση και Ανάπτυξη Συστήματος Αυτοματοποιημένων Υπηρεσιών Στάθμευσης Οχημάτων με Χρονοχρέωση

Αθανάσιος Νικολαΐδης
Α.Μ 19053

Σπύρος Σιώπης
Α.Μ 19052

Εισηγητής: Διονύσης Κανδρής, Καθηγητής

Αθήνα, Ιούνιος 2021

UNIVERSITY OF WEST ATTICA

FACULTY OF ENGINEERING

Department of informatics & computer engineering

Postgraduate course: Science & technology of informatics and computers

Diploma Thesis

Design and Development of an Automated System for Time Charged Vehicle Parking Services

Athanasios Nikolaidis
Registration Number: 19053

Spyros Siopis
Registration Number: 19052

Supervisor:

Dionisis Kandris

Professor

Athens, June 2021

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Σχεδίαση και Ανάπτυξη Συστήματος Αυτοματοποιημένων Υπηρεσιών Στάθμευσης Οχημάτων με Χρονοχρέωση

Αθανάσιος Νικολαΐδης
Α.Μ 19053
Σπύρος Σιώπης
Α.Μ 19052

Εισηγητής:

Διονύσης Κανδρής

Εξεταστική Επιτροπή:

Διονύσης Κανδρής, Καθηγητής

Αντώνης Μπόγρης, Καθηγητής

Σταύρος Φατούρος, Αναπληρωτής Καθηγητής

Ημερομηνία εξέτασης: 29 Ιουνίου 2021

Η Διπλωματική Εργασία έγινε αποδεκτή και βαθμολογήθηκε από την εξής τριμελή επιτροπή:

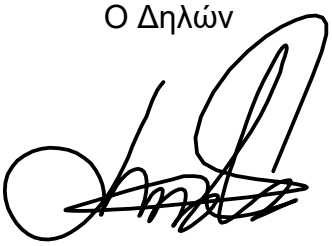
Διονύσης Κανδρής Καθηγητής	Αντώνης Μπόγρης Καθηγητής	Σταύρος Φατούρος Αναπληρωτής Καθηγητής
Dionisis Kandris Digitally signed by Dionisis Kandris Date: 2021.06.30 12:48:56 +03'00'	Antonios Bogris Digitally signed by Antonios Bogris Date: 2021.07.01 08:27:44 +03'00'	Stavros Fatouros Digitally signed by Stavros Fatouros Date: 2021.07.01 01:26:05 +03'00'

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΩΝ ΜΕΤΑΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Οι κάτωθι υπογεγραμμένοι Αθανάσιος Νικολαΐδης του Αντωνίου, με αριθμό μητρώου 19053 και Σιώπης Σπύρος του Σταύρου, με αριθμό μητρώου 19052 φοιτητές του Προγράμματος Μεταπτυχιακών Σπουδών Επιστήμη και Τεχνολογία της Πληροφορικής και των Υπολογιστών του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών της Σχολής Μηχανικών του Πανεπιστημίου Δυτικής Αττικής, δηλώνουμε ότι:

«Είμαστε συγγραφείς αυτής της μεταπτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχαμε για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες κάναμε χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνουμε ότι αυτή η εργασία έχει συγγραφεί από εμάς αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μας, όσο και του Ιδρύματος.

Ο Δηλών



Αθανάσιος Νικολαΐδης

Ο Δηλών

Σπύρος Σιώπης

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα διπλωματική εργασία ολοκληρώθηκε μετά από αρκετές προσπάθειες και προβλήματα λόγω απόστασης, δεδομένου των συνθηκών. Την προσπάθεια αυτή πίστεψε και υποστήριξε ο επιβλέπων καθηγητής Δ. Κανδρής και τον ευχαριστούμε για τον χρόνο του, τις ιδέες του και την άμεση ανταπόκρισή του.

ΠΕΡΙΛΗΨΗ

Η παρούσα μεταπτυχιακή διατριβή άπτεται της σχεδίασης και ανάπτυξης ενός συστήματος αυτοματοποιημένων υπηρεσιών στάθμευσης οχημάτων με χρονοχρέωση, με χρήση ενός συστήματος αυτόματης αναγνώρισης πινακίδων αριθμού (ANPR) με βάση έναν μικροελεγκτή Arduino και αντίστοιχες διαδικτυακές υπηρεσίες.

ABSTRACT

The present thesis deals with the design and development of a system of automated car parking services at time charges, via the creation of an Automatic Number Plate Recognition (ANPR) system based on an Arduino microcontroller and relevant internet services.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1: ΠΡΟΛΟΓΟΣ 14
1.1 Εισαγωγή 14
1.2 Συστήματα ANPR 14
1.3 Αντικείμενο Διατριβής 15
1.4 Διάρθρωση διατριβής 16
ΚΕΦΑΛΑΙΟ 2: ΜΙΚΡΟΕΛΕΓΚΤΗΣ ARDUINO 17
2.1 Εισαγωγή 17
2.2 Ιστορική αναδρομή 17
2.3 Εισαγωγή στους μικροελεγκτές 17
2.4 Εισαγωγή στο Arduino 18
2.4.1 Εκδόσεις Arduino 19
2.4.2 Arduino UNO 22
2.4.3 Εξαρτήματα που χρησιμοποιήθηκαν 28
2.5 Περιβάλλον ανάπτυξης (IDE) 37
2.6 Ανάλυση κώδικα 39
2.6.1 Κώδικας Arduino 40
2.6.2 Κώδικας ESP32-CAM 47
ΚΕΦΑΛΑΙΟ 3: ΔΙΑΔΙΚΤΥΑΚΕΣ ΕΦΑΡΜΟΓΕΣ ΚΑΙ ΥΠΗΡΕΣΙΕΣ 51
3.1 Εισαγωγή 51
3.2 Απαιτήσεις 52
3.3 Τεχνολογίες 53
3.3.1 .NET Core 54
3.3.2 C# 54
3.3.3 MS SQL Server 54

3.3.4 Node.js 55
3.3.5 Vue Js 55
3.4 Εργαλεία 56
3.4.1 Microsoft Visual Studio 56
3.4.2 IntelliJ IDEA 57
3.4.3 SQL Server Management Studio (SSMS) 58
3.4.4 Microsoft Azure 59
3.5 Αρχιτεκτονική 60
3.5.1 Microservices 60
3.5.2 Clean architecture 61
3.5.3 API Gateways 62
3.5.4 JWT Authentication 63
3.6 Αναγνώριση πινακίδας οχήματος 65
3.6.1 Οπτική Αναγνώριση Χαρακτήρων 66
3.6.2 Ιστορική αναδρομή 67
3.6.3 OCR.Space 67
3.7 Ανάλυση κώδικα 70
3.7.1 Διαχείριση κλήσεων (Request handling) 71
3.7.2 Αυθεντικοποίηση 74
3.7.3 Διαχείριση σφαλμάτων (Error handling) 77
3.7.4 Βάση δεδομένων (Database) 78
3.7.5 Χρονοχρέωση 79
3.7.6 Διεπαφές προγραμματισμού χρήστη (APIs) 80
3.7.7 Εφαρμογές χρηστών 81
ΚΕΦΑΛΑΙΟ 4: ΥΛΟΠΟΙΗΣΗ ΣΥΣΤΗΜΑΤΟΣ 84

4.1 Εισαγωγή 84
4.2 Σκοπός εργασίας 84
4.3 Περιγραφή συστήματος 84
4.4 Υλοποίηση συστήματος 86
ΚΕΦΑΛΑΙΟ 5: Επίλογος 91
5.1 Σύνοψη 91
5.2 Προβλήματα και αντιμετώπιση 92
5.3 Μελλοντικές επεκτάσεις συστήματος 93
5.4 Συμπεράσματα 94
ΒΙΒΛΙΟΓΡΑΦΙΑ 97

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

Εικόνα 2.1: Arduino pro mini 19
Εικόνα 2.2: Arduino mega 20
Εικόνα 2.3: Arduino LilyPad 21
Εικόνα 2.4: Arduino esplora 21
Εικόνα 2.5: Arduino UNO 22
Εικόνα 2.6: FTDI FT232 24
Εικόνα 2.7: Ηλεκτρονικά μέρη Arduino UNO 24
Εικόνα 2.8: Εξωτερική πηγή ενέργειας σε Arduino UNO 26
Εικόνα 2.9: Breadboard 28
Εικόνα 2.10: Εσωτερική δομή breadboard 29
Εικόνα 2.11: Servo motor 29
Εικόνα 2.12: TCRT5000 (αριστερά) και MH sensor serial (δεξιά) 30
Εικόνα 2.13: Τρόπος εκπομπής – λήψης αισθητήρα 31
Εικόνα 2.14: LCD screen και I2C adapter 32
Εικόνα 2.15: ESP32-CAM με κάμερα OV2640 33
Εικόνα 2.16: Ηλεκτρονικά μέρη ESP32 33
Εικόνα 2.17: Τα διαθέσιμα pins του ESP32 34
Εικόνα 2.18: FTDI FT232 35
Εικόνα 2.19: Σύνδεση ESP32 με FTDI FT232 για προγραμματισμό 35
Εικόνα 2.20: Arduino IDE terminal 36
Εικόνα 2.21: Διάγραμμα ολόκληρου κυκλώματος 37
Εικόνα 2.22: Arduino IDE 38
Εικόνα 2.23: Νέο project σε Arduino IDE 39
Εικόνα 3.1: Περιβάλλον εργασίας MS Visual Studio 57
Εικόνα 3.2: Περιβάλλον Εργασίας IntelliJ IDEA 58
Εικόνα 3.3: Περιβάλλον εργασίας SSMS 59
Εικόνα 3.4: Πάνελ διαχείρισης MS Azure 59
Εικόνα 3.5: Παράδειγμα microservices 60
Εικόνα 3.6: Clean architecture 61

Εικόνα 3.7: Επίπεδα Clean Architecture 62
Εικόνα 3.8: Gateways σε συνδιασμό με microservices 63
Εικόνα 3.9: Παράδειγμα JWT Token 64
Εικόνα 3.10: Request προς OCR.Space 68
Εικόνα 3.11: Response απο OCR.Space 70
Εικόνα 3.12: Configuration για δρομολόγηση μέσω Ocelot 72
Εικόνα 3.13: Ορισμός διεύθυνσης Controller 73
Εικόνα 3.14: JSON Response από επιτυχημένη ταυτοποίηση 76
Εικόνα 3.15: XSRF-TOKEN Cookie 76
Εικόνα 3.16: Σύλληψη και επεξεργασία σφαλμάτων 77
Εικόνα 3.17: Σχεδιάγραμμα βάσης δεδομένων 79
Εικόνα 3.18: Panel διαχειριστή 81
Εικόνα 3.19: Διαχείριση Tokens μέσω Vuex 83
Εικόνα 4.1: Ξύλινη βάση μακέτας 86
Εικόνα 4.2: Κανάλι καλωδίωσης 86
Εικόνα 4.3: Κεντρικό αυτοκόλλητο 87
Εικόνα 4.4: Αυτοκόλλητο σήμανσης 87
Εικόνα 4.5: Αυτοκόλλητο βάσης 87
Εικόνα 4.6: Διαχωριστικό χαρτόνι 88
Εικόνα 4.7: Μεταλλικά αυτοκινητάκια 88
Εικόνα 4.8: Γλωσσοπίεστα χειροτεχνιών 89
Εικόνα 4.9: Μακέτα κατασκευής 89
Εικόνα 4.10: Ηλεκτρονικός εξοπλισμός 90
Εικόνα 4.11: Εισαγωγή οχήματος 90

ΚΕΦΑΛΑΙΟ 1: ΠΡΟΛΟΓΟΣ

1.1 Εισαγωγή

Η συνεχόμενη αύξηση οχημάτων στις μέρες μας κάνει ολοένα και πιο επιτακτική την ανάγκη για έξυπνα συστήματα αναγνώρισης και διαχείρισης πληροφορίας τους. Ένα σημαντικό πρόβλημά τους είναι η διαχείριση χώρων στάθμευσης που θα πρέπει να ξεφύγει από τις παραδοσιακές μεθόδους και να μεταβεί στην εποχή της πληροφορίας και του αυτοματισμού. Για το βήμα αυτό χρειάζεται να αξιοποιηθούν προϋπάρχουσες και νέες τεχνολογίες με σκοπό την καλύτερη διαχείριση του χρόνου, του χώρου και της πληροφόρησης.

Στο κεφάλαιο αυτό γίνεται αναφορά αρχικά στα συστήματα που βοηθούν στη διαχείριση της αύξησης των οχημάτων. Έπειτα γίνεται μια περιγραφή του συστήματος που σχεδιάστηκε και αναπτύχθηκε στο πλαίσιο της παρούσας μεταπτυχιακής διατριβής και τέλος πραγματοποιείται μια περιγραφή του περιεχομένου των κεφαλαίων που ακολουθούν.

1.2 Συστήματα ANPR

Τα συστήματα Automatic Number Plate Recognition (ANPR) χρησιμοποιούνται για την αναγνώριση αριθμών κυκλοφορίας. Μπορούν να δεχτούν εικόνες από οποιαδήποτε κάμερα αλλά συνήθως υπάρχουν ειδικά διαμορφωμένες κάμερες που εξυπηρετούν τον σκοπό αυτό. Συνήθως συνοδεύονται με πρόσθετα χαρακτηριστικά ανίχνευσης για βελτίωση της εικόνας, όπως για παράδειγμα υπέρυθρο φωτισμό.

Η χρήση αυτών των συστημάτων συναντάται τόσο σε δρόμους υψηλής ταχύτητας όσο και σε χώρους στάθμευσης. Ανάλογα με τη χρήση που προορίζονται, επιλέγεται και ο τύπος του συστήματος που θα χρησιμοποιηθεί. Δυο από τους πιο βασικούς τύπους είναι οι αυτόνομοι και οι διαδικτυακοί. Στον πρώτο γίνεται επεξεργασία της εικόνας σε πραγματικό χρόνο, γιατί το λογισμικό αναγνώρισης βρίσκεται εγκατεστημένο μαζί με τα

υπόλοιπα εξαρτήματα του συστήματος. Ο δεύτερος απαιτεί τη χρήση δικτύου για την ανταλλαγή πληροφοριών με απομακρυσμένα συστήματα που επεξεργάζονται τις εικόνες.

Για την αναγνώριση των αριθμών κυκλοφορίας εφαρμόζονται τεχνικές οπτικής αναγνώρισης χαρακτήρων (Optical Character Recognition - OCR). Η συγκριμένη τεχνική αναλύεται στο τρίτο κεφάλαιο στην ενότητα «Αναγνώριση πινακίδας οχήματος».

1.3 Αντικείμενο Διατριβής

Η συγκεκριμένη μεταπτυχιακή διατριβή έχει ως στόχο τη σχεδίαση και ανάπτυξη ενός συστήματος παροχής αυτοματοποιημένων υπηρεσιών στάθμευσης οχημάτων με χρονοχρέωση. Η ιδέα βασίζεται στην ύπαρξη ενός ή πολλών αυτόνομων συστημάτων που διαχειρίζονται χώρους στάθμευσης και η επίβλεψή τους μπορεί να γίνει από ένα κεντρικό σημείο. Επίσης, οι οδηγοί μπορούν εύκολα με τη δημιουργία ενός λογαριασμού να κάνουν χρήση του συγκεκριμένου συστήματος.

Για την υλοποίηση της εργασίας, χρειάστηκε να δημιουργηθούν εφαρμογές σε διαφορετικούς τομείς όπου η κάθε μια είχε και τον δικό της διακριτό ρόλο. Το σύστημα βασίστηκε πάνω σε έναν μικροελεγκτή Arduino, δυο εφαρμογές διαδικτύου, μια για τον διαχειριστή και μια για τον χρήστη και σε υπηρεσίες διαδικτύου που μπορούν να εγκατασταθούν και να χρησιμοποιηθούν από περιβάλλοντα Linux, Windows και macOS.

Ο ελεγκτής είναι υπεύθυνος για όλες εκείνες τις λειτουργίες που χρειάζεται να γίνουν προκειμένου να ενημερώσει το κεντρικό σύστημα για την πρόθεση κάποιου οχήματος να εισέλθει ή να εξέλθει από το χώρο στάθμευσης. Επίσης, διαχειρίζεται την εμφάνιση μηνυμάτων προς τους χρήστες σε περίπτωση λάθους, αποτροπής ή αδυναμίας του συστήματος να εξυπηρετήσει το συγκεκριμένο όχημα.

Οι υπηρεσίες διαδικτύου από την άλλη είναι υπεύθυνες για τη συλλογή, ανάλυση και επιστροφή δεδομένων από και προς τον ελεγκτή Arduino. Όλη η λογική, δηλαδή, και η διαχειριστική ικανότητα βρίσκεται σε ένα σημείο για να μπορούν να υποστηριχθούν περισσότερες συσκευές σε ενδεχόμενη επέκταση.

Επίσης, οι υπηρεσίες αυτές διαχειρίζονται κι ένα σύνολο λειτουργιών που αφορά τους χρήστες του συστήματος. Από τη δημιουργία, επεξεργασία και αυθεντικοποίηση των χρηστών μέχρι την εξαγωγή πληροφοριών για τη χρήση τους σε γραφήματα.

Οι εφαρμογές διαδικτύου αναφέρονται σε διαφορετικές έννοιες η καθεμιά. Η μια συγκαταλέγεται στις εφαρμογές χρήστη και η άλλη σε αυτές της διαχείρισης του συστήματος. Η διαφοροποίηση αυτή προσφέρει καλύτερη απομόνωση και διαχείρισή τους καθώς το κοινό στο οποίο αναφέρονται έχει διαφορετικές απαιτήσεις και σε πληροφορίες αλλά και σε πόρους συστήματος.

Τέλος, ένα βασικό χαρακτηριστικό του συγκεκριμένου συστήματος είναι η ευελιξία του ως προς την συνέχεια, αναβάθμιση και αντικατάσταση τμημάτων του. Η δυνατότητα αυτή προκύπτει από τις open-source τεχνολογίες που χρησιμοποιήθηκαν και του cross-platform χαρακτηριστικού των υπηρεσιών διαδικτύου.

1.4 Διάρθρωση διατριβής

Το δεύτερο κεφάλαιο αναφέρεται στους μικροελεγκτές Arduino και με τον τρόπο με τον οποίο αυτοί μπορούν να χρησιμοποιηθούν για την κατασκευή ενός συστήματος αυτοματοποιημένων υπηρεσιών στάθμευσης οχημάτων. Ξεκινώντας με μια ιστορική αναδρομή στους μικροελεγκτές και τις εκδόσεις του Arduino και καταλήγοντας σε μια ανάλυση των εξαρτημάτων και των χαρακτηριστικών τους.

Ακολουθούν οι διαδικτυακές εφαρμογές και υπηρεσίες που συγκροτούν το κέντρο ελέγχου του συστήματος. Πιο συγκεκριμένα, το τρίτο κεφάλαιο αναφέρεται στις απαιτήσεις, τεχνολογίες, εργαλεία και αρχιτεκτονικές που χρησιμοποιήθηκαν προκειμένου να υποστηρίξουν ένα ολοκληρωμένο σύστημα ANPR μέσω διαδικτύου με παράλληλη χρήση εφαρμογών για διαχείριση των πληροφοριών χρήστη. Επίσης θα αναλυθεί ο τρόπος με τον οποίο χρησιμοποιεί εξωτερικές υπηρεσίες προκειμένου να αναγνωρίσει την πινακίδα ενός οχήματος.

Στο τέταρτο κεφάλαιο πραγματοποιείται η περιγραφή του τρόπου με τον οποίο υλοποιήθηκε η κατασκευή του συστήματος της διπλωματικής και ο κύκλος εργασιών που ακολουθείται προκειμένου να εισέλθει και να εξέλθει κάποιο όχημα.

Τέλος, γίνεται αναφορά στα προβλήματα που παρουσιάστηκαν κατά την υλοποίηση του συστήματος και στις πιθανές μελλοντικές επεκτάσεις του συστήματος που σχεδιάστηκε και αναπτύχθηκε.

ΚΕΦΑΛΑΙΟ 2: ΜΙΚΡΟΕΛΕΓΚΤΗΣ ARDUINO

2.1 Εισαγωγή

Στο συγκεκριμένο κεφάλαιο πραγματοποιείται εισαγωγή στους μικροελεγκτές, αφού προηγηθεί μία ιστορική αναδρομή όσον αφορά. Στη συνέχεια, αναλύεται ο μικροελεγκτής Arduino και ειδικότερα στις εκδόσεις του, στο Arduino Uno, και στα εξαρτήματα που χρησιμοποιήθηκαν για την υλοποίηση της συγκεκριμένης εργασίας. Ακολουθεί το περιβάλλον ανάπτυξης που αφορά στις εντολές που χρησιμοποιήθηκαν κι έπειτα πραγματοποιείται ανάλυση του κώδικα που δημιουργήθηκε για τον προγραμματισμό του συστήματος.

2.2 Ιστορική αναδρομή

Η ανάγκη για μια συσκευή που θα αλληλεπιδρά με το περιβάλλον, κυρίως για έλεγχο σχεδίων από μαθητές, έκανε τους μηχανικούς να κατασκευάσουν μια συσκευή που την ονόμασαν «Arduino». Σκοπός τους ήταν η συσκευή αυτή να κοστίζει λιγότερο από άλλες αντίστοιχες του εμπορίου. Το 2005, λοιπόν, στην περιοχή Ivrea της βόρειας Ιταλίας, λίγο πιο πάνω από το Τορίνο, οι Massimo Banzi, Tom Igoe, Gianluca Martino, David Cuartielles, David Mallis σχεδίασαν και υλοποίησαν την πλατφόρμα αυτή και της έδωσαν το όνομα από τον γνωστό βασιλιά της Ιταλίας, με όνομα Arduino, όπου κατοικούσε στην πόλη αυτή κατά τον 9^{ο} αιώνα.

Για την υλοποίησή του, βασίστηκαν στην πλατφόρμα της Wiring, η οποία αφορά μια εργασία του πανεπιστημίου Interaction Design Institute της πόλης Ivrea.Το σχέδιο είναι μια μίξη της πλατφόρμας Wiring που προγραμματίζεται με γλώσσα παρόμοια της C/C++ καθώς κι ενός περιβάλλοντος ανάπτυξης (IDE).

2.3 Εισαγωγή στους μικροελεγκτές

Με την έννοια «μικροελεγκτής» νοείται ένα ολοκληρωμένο ηλεκτρονικό κύκλωμα που έχουν τοποθετηθεί πάνω του ένας επεξεργαστής, μία μνήμη και έχει εισόδους και

εξόδους. Στις περισσότερες περιπτώσεις η μνήμη, και όχι μόνο, είναι ενσωματωμένη στο chip του επεξεργαστή. Έχει μικρό μέγεθος, χαμηλό κόστος και η παρουσία του είναι αισθητή σε διάφορα ηλεκτρονικά συστήματα που χρησιμοποιούμε σήμερα.

Οι μικροελεγκτές σχεδιάστηκαν για να εκτελούν μικρές λειτουργίες και για αυτόν τον λόγο έχουν περιορισμένους πόρους (χαμηλό σε ταχύτητα επεξεργαστή, μικρή σε χωρητικότητα μνήμη RAM, ROM και περιορισμένα κανάλια I/O).

Η διαφορά των μικροελεγκτών από τους μικροεπεξεργαστές, δηλαδή των επεξεργαστών των προσωπικών υπολογιστών, έγκειται στο γεγονός ότι ένας μικροεπεξεργαστής έχει πολύ μεγαλύτερη υπολογιστική ισχύ, καθώς η λειτουργικότητά του καθορίζεται από εξωτερικά περιφερειακά, όπως για παράδειγμα, μνήμες μη ενσωματωμένες μέσα στο chip, τα οποία διασυνδέονται με τον μικροεπεξεργαστή. Επίσης, υπάρχει πολύ μεγαλύτερη ευελιξία ανάπτυξης και αναπαραγωγής διαφορετικών εφαρμογών σε πολυπλοκότητα και σε μέγεθος. Αντίθετα, οι μικροελεγκτές, λόγω περιορισμένης υπολογιστικής ισχύς και μνήμης, δύναται να χρησιμοποιούνται σε μικρότερες εφαρμογές που δεν απαιτούν μεγάλη κατανάλωση σε τέτοιους πόρους.

Επομένως, κάποια από τα πλεονεκτήματα των μικροελεγκτών είναι τα παρακάτω:

- Μικρό μέγεθος, επειδή υπάρχει όλο το κύκλωμα εντός του κεντρικού chip.
- Χαμηλό κόστος, γιατί δεν υπάρχει επιπλέον ανάγκη για περιφερειακά κυκλώματα, όπως η μνήμη.
- Αυτονομία, διότι γίνεται χρήση ελάχιστων πόρων και καταναλώνεται μικρότερη ενέργεια.

2.4 Εισαγωγή στο Arduino

Το Arduino είναι μια πλατφόρμα ανοικτού κώδικα βασισμένη σε μια εύκολη διαχείριση υλικού και λογισμικού. Η πλακέτα μπορεί να διαβάζει εισόδους, όπως το φως σε έναν αισθητήρα, το μήνυμα σε μια πλατφόρμα κοινωνικής δικτύωσης, ένα SMS κ.ά. και να τα μετατρέπει σε εξόδους, όπως το να ενεργοποιήσει ένα servo motor, να ανάψει ένα led, να δημοσιοποιήσει κάτι online κ.ά. Δίνεται η δυνατότητα προγραμματισμού του μικροελεγκτή μέσω κώδικα, χρησιμοποιώντας τη γλώσσα προγραμματισμού Arduino που είναι βασισμένη στο Wiring, καθώς και το περιβάλλον ανάπτυξης κώδικα (IDE) που είναι βασισμένο στο Processing.

Όλα αυτά τα χρόνια, το Arduino χρησιμοποιείται σε διάφορα projects οποιασδήποτε πολυπλοκότητας από μαθητές, φοιτητές, προγραμματιστές κ.λπ.

Όλες οι πλατφόρμες του Arduino είναι ανοικτού κώδικα ενθαρρύνοντας τους χρήστες να το χρησιμοποιήσουν, αλλά και να κατασκευάσουν τα δικά τους ανεξάρτητα εξαρτήματα ανάλογα με τις ανάγκες τους. Το λογισμικό είναι επίσης ανοικτού κώδικα και μπορεί να συμβάλει οποιοσδήποτε στην ανάπτυξή του παγκοσμίως.

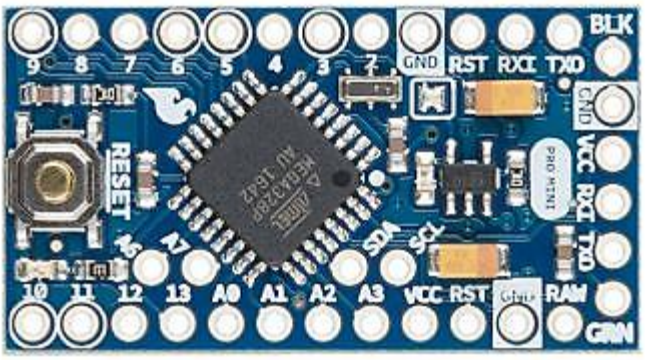
Υπάρχουν αρκετοί μικροελεγκτές και πλατφόρμες διαθέσιμοι στην αγορά. Ωστόσο, το Arduino είναι πολύ φιλικό και απλοποιημένο κατά τον προγραμματισμό του μικροελεγκτή. Κάποια από τα πλεονεκτήματα του Arduino έναντι των άλλων συστημάτων είναι τα εξής:

- Χαμηλό κόστος σχετικά με άλλες αντίστοιχες πλατφόρμες.
- Υποστήριξη πολλαπλών λειτουργικών συστημάτων (cross-platform).
- Απλό και εύχρηστο περιβάλλον προγραμματισμού.
- Ανοικτού κώδικα και επεκτάσιμο υλικό και λογισμικό.

Για τους παραπάνω λόγους, το Arduino έχει προτιμηθεί από την εκπαιδευτική κοινότητα παγκοσμίως, ώστε να μπορέσουν οι μαθητές/φοιτητές να μάθουν πώς θα προγραμματίσουν τον μικροελεγκτή για να χρησιμοποιηθεί στα δικά τους projects.

2.4.1 Εκδόσεις Arduino

- Arduino pro mini

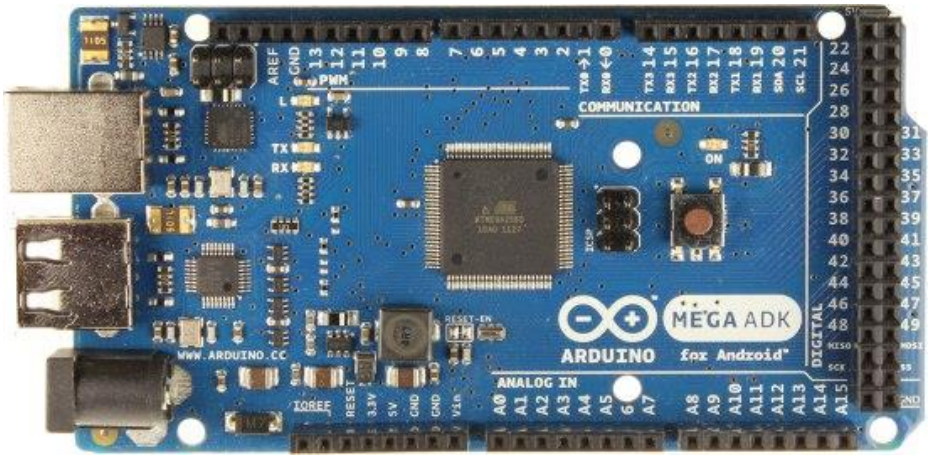


Εικόνα 2.1: Arduino pro mini

- Μικροελεγκτής: ATmega168

- Τάση: 3.3/5V
- Ψηφιακά pins: 14
- Αναλογικά pins: 6
- Flash μνήμη: 16KB
- SRAM: 1KB
- EEPROM: 512 bytes
- Clock speed: 8/16 MHz

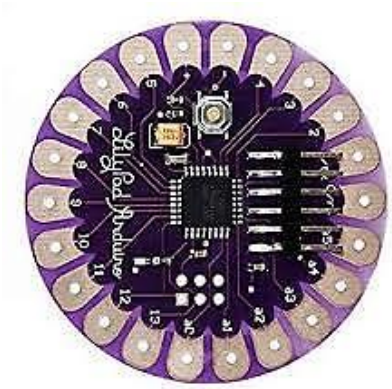
➢ Arduino Mega



Εικόνα 2.2: Arduino mega

- Μικροελεγκτής: ATmega1280
- Τάση: 5V
- Ψηφιακά pins: 54
- Αναλογικά pins: 16
- Flash μνήμη: 128KB
- SRAM: 8KB
- EEPROM: 4KB
- Clock speed: 16 MHz

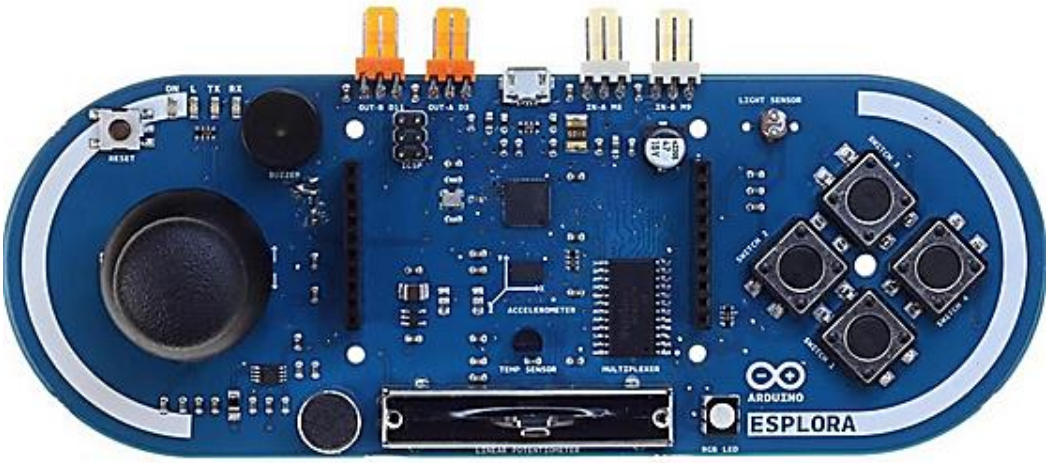
- Arduino LilyPad



Εικόνα 2.3: Arduino LilyPad

- Μικροελεγκτής: ATmega168V / ATmega328V
- Τάση: 2.7/5.5V
- Ψηφιακά pins: 14
- Αναλογικά pins: 6
- Flash μνήμη: 16KB
- SRAM: 1KB
- EEPROM: 512 bytes
- Clock speed: 8 MHz

- Arduino Esplora



Εικόνα 2.4: Arduino esplora

Analog joystick, 4 push-buttons, Linear potentiometer slider, microphone, light sensor, temperature sensor, 3-axis accelerometer, buzzer, RGB led.

2.4.2 Arduino UNO

Το Arduino UNO είναι η πιο διαδεδομένη πλακέτα ανάμεσα στα υπόλοιπα μοντέλα της σειράς Arduino, καθώς έχει αρκετά χαμηλό κόστος και χρησιμοποιείται για απλές εφαρμογές. Περιέχει όλα τα απαραίτητα στοιχεία ώστε να λειτουργεί σωστά. Συγκεκριμένα, αποτελείται από έναν μικροελεγκτή ATmelAVR (ATmega328, ATmega168 ή ATmega8), από εισόδους και εξόδους, ένα πλήκτρο reset, διάφορα ηλεκτρονικά μέρη, καθώς και την τροφοδοσία της.

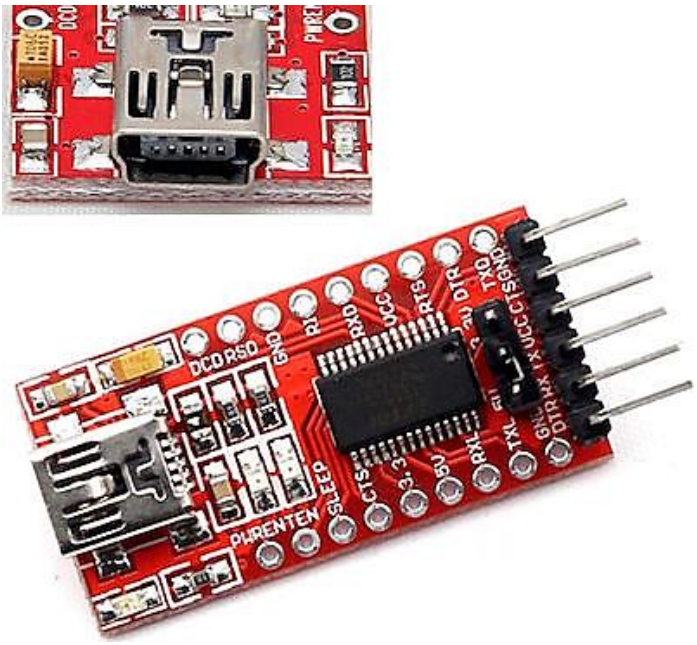
Εικόνα 2.5: Arduino UNO

- Μικροελεγκτής: ATmega328
- Τάση: 5V
- Ψηφιακά pins: 14
- Αναλογικά pins: 6
- Flash μνήμη: 32KB
- SRAM: 2KB
- EEPROM: 1KB
- Clock speed: 16 MHz

Στην FLASH memory εγκαθίσταται το πρόγραμμα που πρόκειται να εκτελεστεί, καθώς και ο φορτωτής εκκίνησης (bootloader). Η SRAM memory χρησιμοποιείται για την προσωρινή αποθήκευση των δεδομένων του προγράμματος που εκτελείται. Στην EEPROM memory, αποθηκεύονται οι τιμές των μεταβλητών (όταν η πλατφόρμα είναι κλειστή) και οι ρυθμίσεις παραμέτρων. Στην FLASH και στην EEPROM, οι πληροφορίες παραμένουν ακόμα και μετά την απενεργοποίηση της πλακέτας, σε αντίθεση με την SRAM, όπου οι πληροφορίες χάνονται. Είναι σημαντικό να χρησιμοποιούνται οι κατάλληλοι τύποι δεδομένων από τον προγραμματιστή, ώστε να μη γίνεται άσκοπη κατανάλωση της μνήμης και μεγαλώνει το πρόγραμμα, διότι ενδέχεται να αποτύχει λόγω μικρής μνήμης SRAM.

Οι πλακέτες αυτές εμπεριέχουν έναν γραμμικό ρυθμιστή τάσης 5V κι έναν κρυσταλλικό ταλαντωτή χρονισμένο στα 16MHz. Ο μικροελεγκτής είναι προγραμματισμένος με έναν bootloader, ώστε να μη χρειάζεται εξωτερικός προγραμματιστής. Το Arduino είναι σχεδιασμένο να δέχεται επιπλέον πλακέτες όπως, για παράδειγμα, Ethernet, Prototyping, GPS Shields κ.λπ. Επιπλέον, δέχονται modules όπου «κουμπώνουν» στην πλακέτα είτε απευθείας, είτε γίνεται διασύνδεση μέσω κατάλληλων καλωδίων, που μπορούν να δώσουν τις λειτουργίες που απαιτούνται για το εκάστοτε project. Τέτοιες λειτουργίες είναι το Ultrasonic, Bluetooth, LCD Display, Buzzer, SD Card reader, ESP32-CAM, Camera modules κ.λπ.

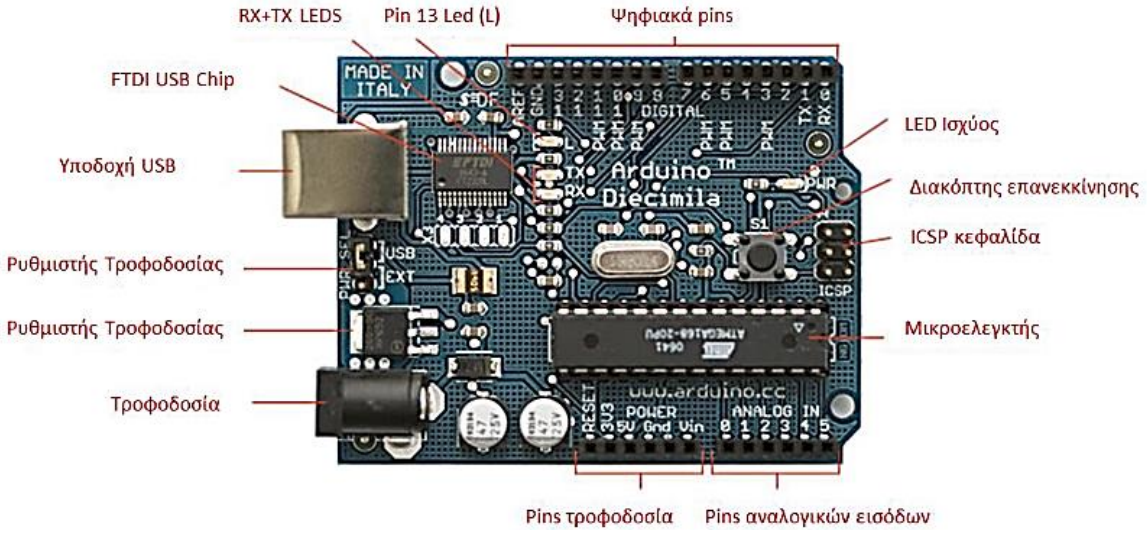
Οι πλακέτες που πωλούνται σήμερα, προγραμματίζονται μέσω USB ή εναλλακτικά εφαρμόζοντας έναν προσαρμογέα USB-to-Serial (π.χ. FTDI FT232).



Εικόνα 2.6: FTDI FT232

Κάποια από τα παραπάνω modules έχουν χρησιμοποιηθεί και για την υλοποίηση της παρούσας εργασίας.

Στην παρακάτω εικόνα, φαίνονται τα στοιχεία που απαρτίζουν την πλακέτα:



Εικόνα 2.7: Ηλεκτρονικά μέρη Arduino UNO

Τα ψηφιακά pins είναι αυτά που μπορούν να λειτουργήσουν ως είσοδοι και έξοδοι για το συγκεκριμένο πρόγραμμα. Χρειάζονται τάση 5V και ρεύμα μέχρι 40mA. Στον προγραμματισμό του pin υπάρχουν 2 καταστάσεις LOW και HIGH, οπότε ο μικροελεγκτής δίνει ή όχι ρεύμα ανάλογα την κατάσταση που έχει δηλώσει ο προγραμματιστής.

Κάποια από τα pins αυτά έχουν επιπλέον λειτουργίες όπως τα Rx (pin 0) και Tx (pin 1) που χρησιμοποιούνται για τη σειριακή θύρα (λειτουργούν παράλληλα με το καλώδιο USB που συνδέεται στον υπολογιστή). Αυτά τα 2 pins μας δίνουν τη δυνατότητα να συνδέσουμε το Arduino με άλλες συσκευές και να επιτευχθεί μια επικοινωνία για ανταλλαγή δεδομένων.

Τα pin 2 και 3 έχουν τη δυνατότητα να λειτουργήσουν και ως εξωτερικοί διακόπτες. Δηλαδή, μπορούν να προγραμματιστούν, ώστε να λειτουργούν όταν συμβαίνουν συγκεκριμένες αλλαγές. Είναι χρήσιμα σε εφαρμογές που απαιτούν μεγάλης ακρίβειας συγχρονισμό.

Τα pin3, 5, 6, 9, 10 και 11 έχουν τη δυνατότητα να λειτουργήσουν ως ψευδο-αναλογικές έξοδοι με το σύστημα διαμόρφωσης παλμών (PWM). Αυτό σημαίνει ότι όταν συνδεθεί κάτι πάνω σε ένα από αυτά τα pin, τότε υπάρχουν 256 καταστάσεις (0-255). Το σύστημα PWM δεν είναι πραγματικό αναλογικό σύστημα. Πιο συγκεκριμένα, δίνοντας μια τιμή 127, η έξοδος δε θα δώσει 2.5V, αλλά 5V με μεγάλη συχνότητα για ίσα διαστήματα μεταξύ των 0V και 5V, ώστε η μέση τιμή να ισούται στα 2.5V.

Επιπλέον, το Arduino παρέχει μια σειρά από αναλογικά pins (6 για την ακρίβεια) που μπορούν να λειτουργήσουν ως αναλογική είσοδος χρησιμοποιώντας τον ADC Converter (Analog-to-Digital Converter) που υπάρχει εντός του μικροελεγκτή. Αν τροφοδοτηθεί ένα από αυτά τα pins με μια τάση, τότε μέσα από το πρόγραμμα μπορεί να ερμηνευτεί η τάση αυτή σε έναν αριθμό χωρητικότητας 10 bits όπου θα δώσει 1024 επίπεδα (0-1023). Η τάση αυτή μπορεί να δηλωθεί μέσω προγραμματισμού. Ο προγραμματιστής έχει τη δυνατότητα να ορίζει την τάση αναφοράς στη θύρα AREF. Οπότε, δίνοντας μια τάση AREF = 3.3V, τότε όταν στο pin εισέρχεται τάση 1.65V, το Arduino θα επιστρέψει την τιμή 512 όπου αντιστοιχεί στα 1.65V από τα 3.3V που έχουμε. Αν δώσει AREF = 5V και στο pin εφαρμοστεί τάση 1.65V, τότε το Arduino θα επιστρέψει την τιμή 338. Τέλος, αν έχουμε AREF = 5V και στο pin τάση 2.5V, τότε θα επιστρέψει τιμή 512.

Το Arduino παρέχει μια επιπλέον ευελιξία όπου μπορούν αυτά τα 6 pins να μετατραπούν σε ψηφιακά μέσω των κατάλληλων εντολών από το πρόγραμμα. Σε τέτοια περίπτωση, τα pins δηλώνονται μέσα στο πρόγραμμα από 0 – 5 σε 14 – 19.

Η βασική σύνδεση του Arduino με τον ηλεκτρονικό υπολογιστή επιτυγχάνεται μέσω USB καλωδίου που περιέχεται στη συσκευασία. Η σύνδεση είναι σειριακή και είναι απαραίτητη για τον προγραμματισμό του μικροελεγκτή, αλλά και εναλλακτικά αν θέλουμε να παρέχουμε τροφοδοσία για τη λειτουργία του μέσω του USB καλωδίου. Η τροφοδοσία μπορεί να υλοποιηθεί και μέσω μιας υποδοχής 2.1mm που βρίσκεται στην κάτω αριστερή γωνία της πλακέτας.

Εικόνα 2.8: Εξωτερική πηγή ενέργειας σε Arduino UNO

Σε αυτήν την περίπτωση, η εξωτερική τάση θα πρέπει να κυμαίνεται μεταξύ 7 – 12V και μπορεί να χρησιμοποιηθεί πηγή DC τάσης που μπορεί να προέρχεται είτε από κάποια μπαταρία, είτε από μετασχηματιστή.

Στην πλακέτα υπάρχουν 6 επιπλέον pins, δίπλα από τα αναλογικά, όπου χρησιμοποιούνται για την τροφοδοσία. Συγκεκριμένα έχουμε:

- **RESET**: Όταν αυτό το pin συνδεθεί με κάποια γείωση GND, τότε πραγματοποιείται επανεκκίνηση της πλακέτας.
- **3.3V**: Από αυτό το pin μπορούν να τροφοδοτηθούν όλες οι συσκευές που λειτουργούν σε αυτήν την τάση. Η τάση αυτή (3.3V) δεν προέρχεται από την πηγή τροφοδοσίας που υπάρχει για να δίνει ρεύμα στο Arduino, αλλά από τον ελεγκτή Serial-over-USB, με αποτέλεσμα να έχει μέγιστη ένταση 50mA.

- **5V**: Στην περίπτωση αυτή, η τάση προέρχεται από την πηγή τροφοδοσίας και πριν καταλήξει στο pin, παρεμβάλλεται ένας ρυθμιστής τάσης όπου «ρίχνει» την τάση στα 5V.
- **GND**: Τα 2 αυτά pins υπάρχουν για τις κατάλληλες γειώσεις.
- **Vin**: Το pin αυτό μπορεί να χρησιμοποιηθεί με δύο τρόπους. Είτε ως εξωτερική τροφοδοσία σε συνδυασμό με το GND που βρίσκεται δίπλα του και στην περίπτωση που δε δίνεται η δυνατότητα χρήστης του φις 2.1mm, είτε αν χρησιμοποιείται το φις, τροφοδοτεί τα εξαρτήματα με την πλήρη τάση (πριν περάσει από τον ρυθμιστή τάσης όπως γίνεται στο pin με τα 5V).

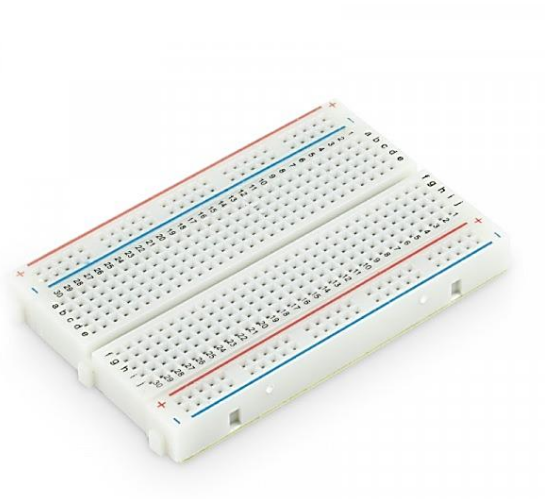
Επιπρόσθετα, η πλακέτα περιέχει ένα κουμπί RESET, καθώς και τέσσερα μικροσκοπικά LED. Το RESET έχει τη δυνατότητα να κάνει επανεκκίνηση το σύστημα σε περίπτωση που κάτι δε βαίνει ορθώς. Όπως αναφέρθηκε προηγουμένως, επανεκκίνηση μπορεί να γίνει και μέσω του pin RESET. Η λειτουργία των τεσσάρων LED αναφέρεται παρακάτω:

- **POWER (ή ON**): Όταν είναι αναμμένο δείχνει τη λειτουργία της πλατφόρμας και τη σωστή παροχή τροφοδοσίας.
- **Tx** και **Rx**: Τα δύο αυτά LEDs δείχνουν την επικοινωνία που επιτυγχάνεται μέσω της σειριακής επικοινωνίας είτε αυτή πραγματοποιείται μέσω USB, είτε μέσω των pins που αναφέρθηκαν παραπάνω.
- **L**: Το LED αυτό ενσωματώθηκε από τους κατασκευαστές για δοκιμαστική χρήση και συνδέεται με το pin 13 (digital pin). Οπότε, αν ο προγραμματιστής αναθέσει τιμή HIGH στο pin 13, τότε θα ανάψει το LED.

2.4.3 Εξαρτήματα που χρησιμοποιήθηκαν

- Breadboard

Πρόκειται για την πλακέτα που φαίνεται στην παρακάτω εικόνα:



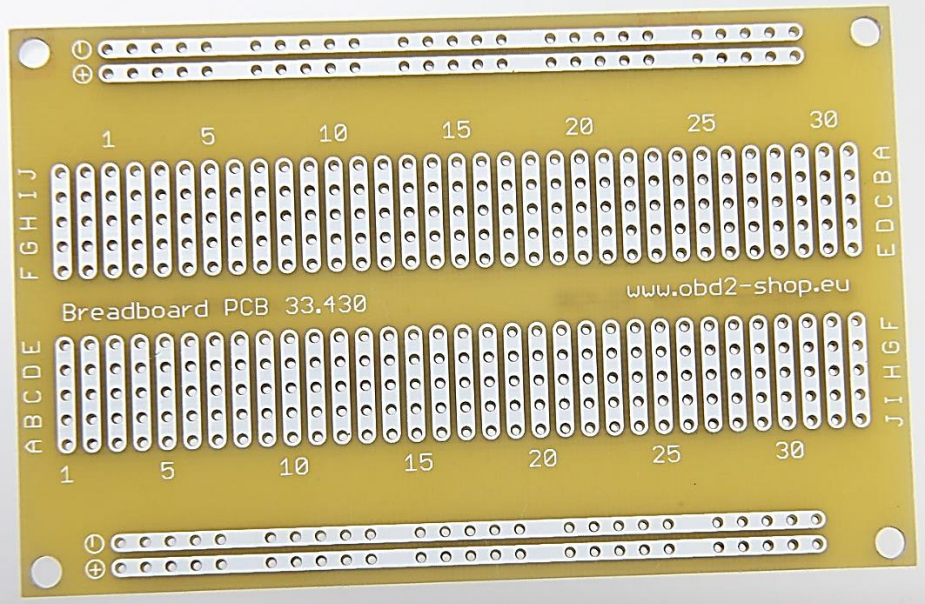
Εικόνα 2.9: Breadboard

Η χρήση της πλακέτας μας βοηθά να αποφύγουμε τις κολλήσεις ειδικά στο στάδιο της ανάπτυξης του κυκλώματος και των δοκιμών. Προσαρμόζονται προσωρινά όλα τα ηλεκτρονικά εξαρτήματα, τα οποία μετά τη χρήση μπορούν να αφαιρεθούν με ευκολία και να τοποθετηθούν σε άλλο κύκλωμα.

Πάνω στην πλακέτα υπάρχουν οι κατάλληλες εγκοπές για να τοποθετηθούν τα pins από τα εξαρτήματα ή καλώδια που θα δημιουργούν τις κατάλληλες συνδέσεις. Στα πλαϊνά μέρη, όπως φαίνεται και στην εικόνα, υπάρχουν έγχρωμες γραμμές κόκκινες και μπλε που δείχνουν ότι οι εγκοπές αυτές είναι συνδεδεμένες κάθετα, ενώ οι εσωτερικές εγκοπές είναι συνδεδεμένες οριζόντια. Στο μέσο της πλακέτας υπάρχει μια μεγάλη εγκοπή που χωρίζει σε δύο μέρη την πλακέτα και δείχνει ότι δεν υπάρχει κάποια σύνδεση μεταξύ των δύο μερών. Στο πάνω μέρος υπάρχουν γράμματα και στο πλάι υπάρχουν αριθμοί, ώστε να γίνεται «ονοματοδοσία» των οπών. Πολύ εύκολα μπορεί να χρησιμοποιηθούν πολλαπλές τέτοιες πλακέτες ανάλογα με τις ανάγκες του project.

Επίσης, υπάρχουν και διαφορετικά μεγέθη τέτοιων πλακετών.

Ένα υπόδειγμα για το πώς είναι το εσωτερικό μιας τέτοιας πλακέτας, μπορούμε να δούμε στην παρακάτω φωτογραφία:



Εικόνα 2.10: Εσωτερική δομή breadboard

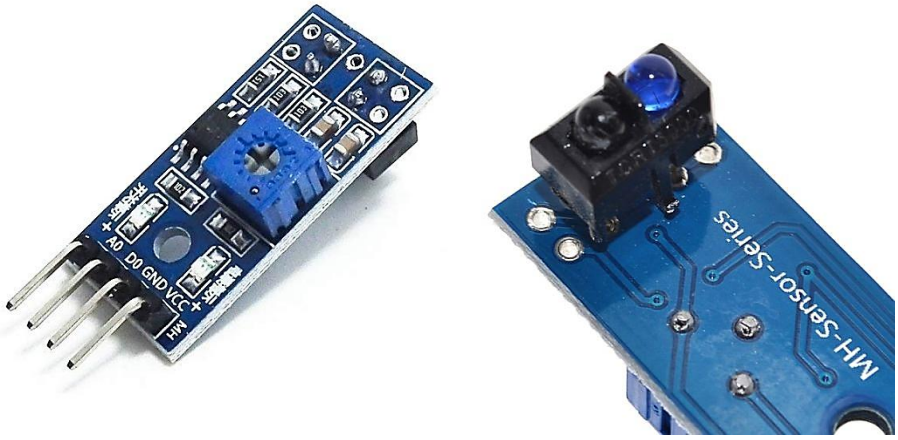
- Servo motor

Εικόνα 2.11: Servo motor

Οι κινητήρες τύπου servo είναι εξαρτήματα που έχουν έναν κεντρικό άξονα, ο οποίος μπορεί να περιστραφεί. Ανάλογα το σήμα που στέλνεται στο servo, πραγματοποιείται η αντίστοιχη περιστροφή. Χρησιμοποιούνται κυρίως στη ρομποτική, στην εκπαίδευση, σε τηλεχειριζόμενα συστήματα, για μετακίνηση δίσκου CD/DVD κ.λπ. Η ενέργεια που

καταναλώνει είναι ανάλογη του φορτίου που δέχεται. Η λειτουργία του είναι αναλογική και κάνει τόση περιστροφή όση αντιστοιχεί στο ηλεκτρικό σήμα που δέχεται. Ο κύριος λόγος χρήσης τέτοιων κινητήρων είναι ότι παρέχει γωνιακή ακρίβεια. Δηλαδή, θα πραγματοποιήσει την περιστροφή και στη συνέχεια θα σταματήσει μέχρι να λάβει το επόμενο σήμα για νέα περιστροφή. Σε αντίθεση με έναν τυποποιημένο ηλεκτροκινητήρα, ο οποίος αρχίζει να περιστρέφεται όσο τροφοδοτείται με ρεύμα και σταματάει όταν σταματήσει η τροφοδοσία του. Τέλος, έχει τη δυνατότητα ανίχνευσης σφαλμάτων και διόρθωσης της απόδοσής του.

- TCRT5000 + MH sensor serial



Εικόνα 2.12: TCRT5000 (αριστερά) και MH sensor serial (δεξιά)

Το συγκεκριμένο εξάρτημα μπορούμε να το συναντήσουμε με διαφορετικές ονομασίες π.χ. MH-sensor-seriesή KY-033 (συνήθως με 3 pins) ή TCRT5000 κ.λπ.

Ωστόσο, όταν αναφερόμαστε σε TCRT5000 εννοούμε το εξάρτημα που είναι πάνω στην πλακέτα, το οποίο εκπέμπει και απορροφά την υπέρυθρη ακτινοβολία. Στην ουσία πρόκειται έναν IR sensor (αισθητήρα υπερύθρων) ο οποίος μπορεί να ανιχνεύσει οποιοδήποτε εμπόδιο και να δώσει μια τιμή στην αναλογική του έξοδο (A0), ανάλογη της απόστασης του αντικειμένου που ανίχνευσε.

Όπως φαίνεται και στην εικόνα, αποτελείται από τέσσερα pins τα οποία αντιστοιχούν στην αναλογική έξοδο (A0), ψηφιακή έξοδο (D0), γείωση (GND), τροφοδοσία (Vcc)

αντίστοιχα. Η πλακέτα, ακόμα, αποτελείται από ένα chip LM393 καθώς και από ένα ποτενσιόμετρο (μπλε) για τη ρύθμιση της ευαισθητοποίησης του TCRT5000. Τέλος, περιέχει και ένα μικρό led που δίνει την τιμή του ψηφιακού pin. Όσο, δηλαδή, είναι κοντά κάποιο αντικείμενο και έχουμε για ψηφιακή έξοδο D0 = 0, τόσο το led ανάβει.

Χρησιμοποιείται για τον εντοπισμό αντικειμένων καθώς και για να ανιχνεύσει το χρώμα σε αποχρώσεις του γκρι, π.χ. εντοπισμός μιας γραμμής από ένα ρομποτικό αυτοκινούμενο όχημα.

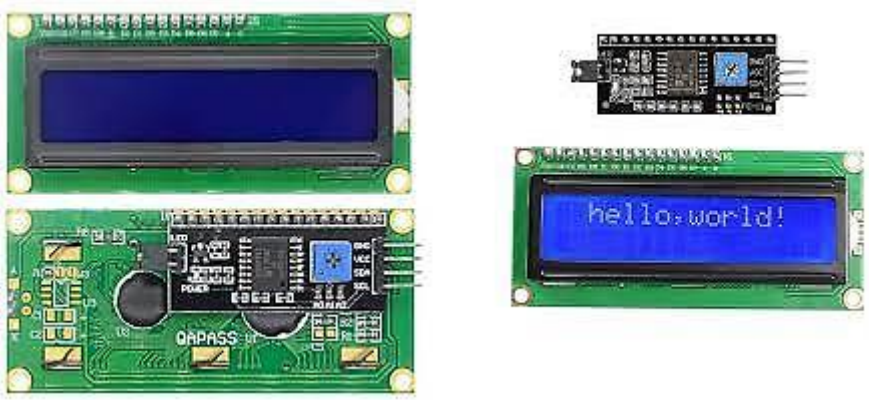
Στην πράξη, το TCRT5000 εκπέμπει από το γαλάζιο led και το μαύρο led λαμβάνει την αντανακλούμενη ακτινοβολία.

Εικόνα 2.13: Τρόπος εκπομπής – λήψης αισθητήρα

Όσο περισσότερο πλησιάζει ένα αντικείμενο, τόσο η αναλογική τιμή που δίνει στο pinA0 μειώνεται και αντίστοιχα η ψηφιακή τιμή που δίνει στο pin D0 μηδενίζει.

Το παρόν εξάρτημα, επηρεάζεται εύκολα από τις περιβαλλοντικές συνθήκες όπως, για παράδειγμα, αν έχει ήλιο ή κάποιο τεχνητό φως (λάμπα).

- LCD Screen + I2C Adapter



Εικόνα 2.14: LCD screen και I2C adapter

Η οθόνη είναι τύπου LCD, 2 γραμμών και 16 χαρακτήρων σε κάθε γραμμή. Εμφανίζει λευκού χρώματος χαρακτήρες πάνω σε μπλε φόντο. Για τη σύνδεσή της χρησιμοποιήθηκε ένας αντάπτορας.

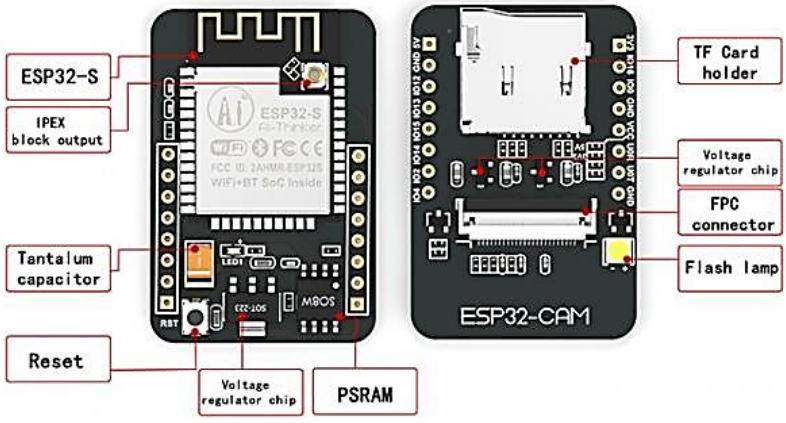
Η οθόνη έχει 16 pins και συνδέεται απευθείας με τον αντάπτορα (I2C), ο οποίος λαμβάνει το σήμα από το Arduino και το δίνει στην οθόνη για να εμφανιστεί. Ο αντάπτορας αποτελείται από 4 pins· τη γείωση (GND), την τροφοδοσία (Vcc) τη γραμμή δεδομένων (SDA), τη γραμμή ρολογιού (SCL), καθώς κι ένα ποτενσιόμετρο, από όπου μπορεί να ρυθμιστεί η ένταση της οθόνης (το πόσο έντονα θα φαίνονται τα γράμματα). Επιπλέον, έχει ένα led (POWER) που δείχνει τη σωστή τροφοδοσία του.

Ο I2C είναι ένας σειριακός δίαυλος επικοινωνίας για χρήση περιφερειακών χαμηλής ταχύτητας σε μητρικές πλακέτες. Για τη μεταφορά δεδομένων χρησιμοποιεί δύο καλώδια SDA και SCL τα οποία συνδέονται στα A4 και A5 pinstou Arduino, αντίστοιχα.

- ESP32-CAM + FDTI FT232

Εικόνα 2.15: ESP32-CAM με κάμερα OV2640

Το ESP32-CAM είναι ένα μικρό και χαμηλής ενέργειας εξάρτημα, με υποδοχέα κάμερας. Έχει τοποθετημένη πάνω του μια κάμερα OV2640 2MP με φλας καθώς και υποδοχή για κάρτες microSD. Μπορεί να χρησιμοποιηθεί ευρέως σε συστήματα IoT, όπως ασύρματη κάμερα ασφαλείας, αποστολή εικόνας σε server μέσω wifi, αναγνώριση QR, αναγνώριση προσώπων κ.ά.



Εικόνα 2.16: Ηλεκτρονικά μέρη ESP32

Χαρακτηριστικά ESP32-CAM

- Ενσωματωμένο wifi module (IEEE 802.11 b/g/n/e/i)
- Built-in flash memory 32Mbit
- Internal RAM 512KB
- External PSRAM 4MB
- On board PCB antenna
- Bluetooth 4.2 BLE
- Output image format JPEG, BMP, GRAYSCALE
- Support TF card up to 4GB
- 9 I/O ports
- 5V power supply

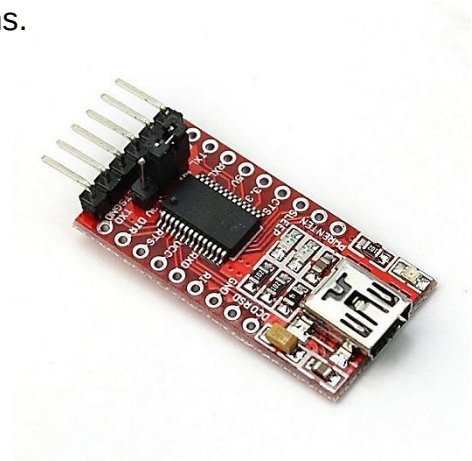
Παρακάτω αναλύονται τα pins όπως φαίνονται και στην εικόνα:

Εικόνα 2.17: Τα διαθέσιμα pins του ESP32

Υπάρχουν 3 GND pins και 2 pins για τροφοδοσία 3.3V και 5V. Επίσης, τα pin GPIO1 και GPIO3 είναι τα σειριακά pins. Χρησιμοποιούνται για τον προγραμματισμό της πλακέτας. Επιπλέον, το pin GPIO0 διαδραματίζει καθοριστικό ρόλο, διότι καθορίζει αν η πλακέτα βρίσκεται σε flashing mode. Όταν το GPIO0 είναι συνδεδεμένο με την γείωση GND, τότε βρίσκεται σε flashing mode. Τα pins GPIO 14: CLK, GPIO 15: CMD, GPIO 2:

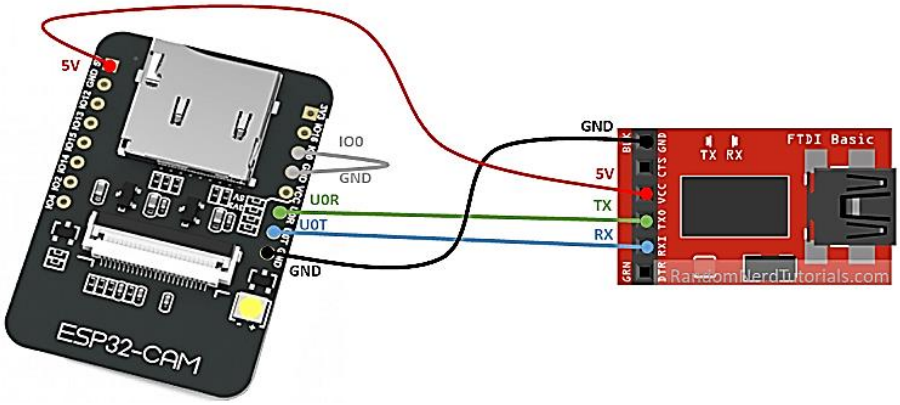
Data 0, GPIO 4: Data 1, GPIO 12: Data 2, GPIO 13: Data 3 είναι συνδεδεμένα με τον υποδοχέα microSD card reader.

Το εξάρτημα αυτό δεν έχει υποδοχή για USB, οπότε χρειάζεται ένας μετατροπέας για να προγραμματιστεί. Ο μετατροπέας αυτός είναι ο FT232RL FTDI και συνδέεται σειριακά μέσω των U0R και U0T pins.



Εικόνα 2.18: FTDI FT232

Η σύνδεση των δύο (2) εξαρτημάτων για τον προγραμματισμό του ESP32-CAM, φαίνεται στην παρακάτω εικόνα:



Εικόνα 2.19: Σύνδεση ESP32 με FTDI FT232 για προγραμματισμό

Για να προγραμματιστεί, θα πρέπει να υπάρχει η συνδεσμολογία που φαίνεται στην παραπάνω εικόνα και στη συνέχεια να δοθούν οι κατάλληλες παράμετροι από το πρόγραμμα IDE. Οπότε, ακολουθούνται τα παρακάτω βήματα από το μενού:

1. **Tools → Board → AI-Thinker ESP32-CAM.**
2. **Tools → Port** και επιλέγεται η πόρτα που είναι συνδεδεμένο το FTDI.

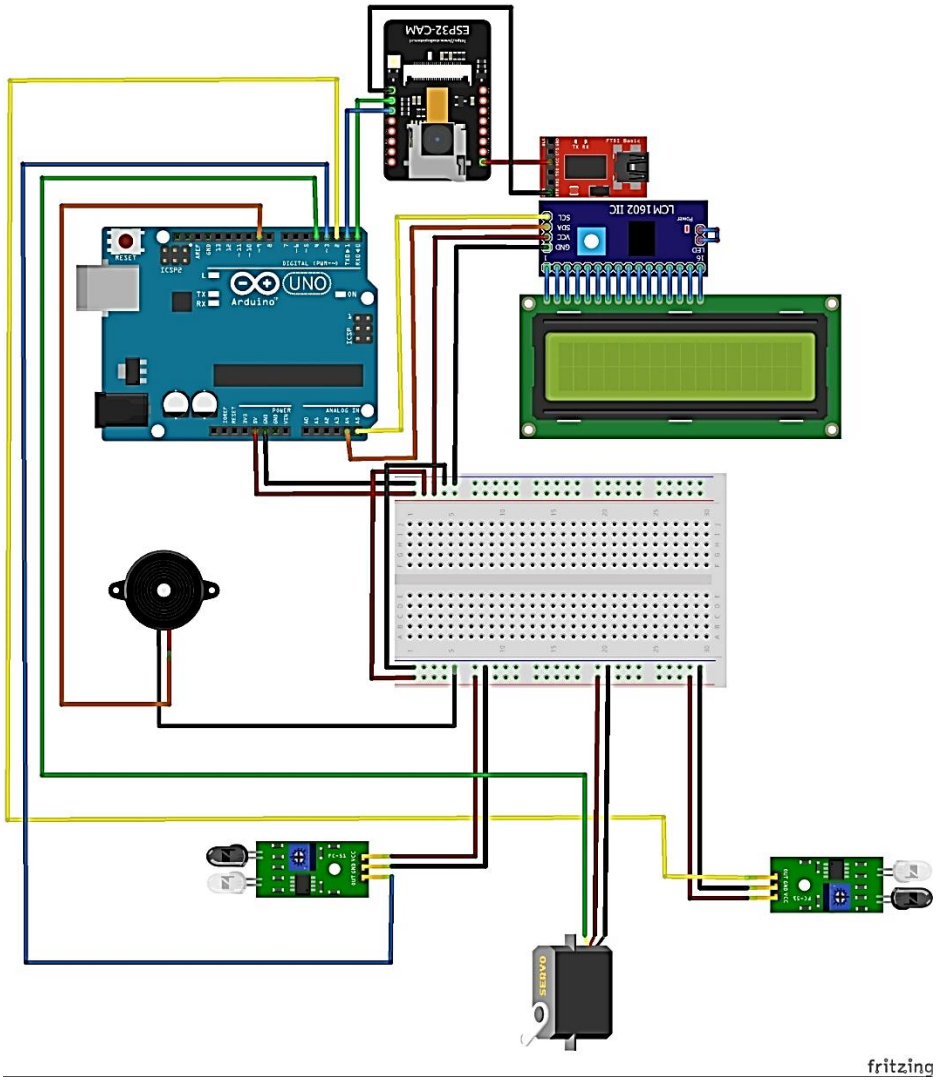
3.Στη συνέχεια ανεβαίνει ο κώδικας (Upload) από το IDE.

4.Όταν θα εμφανιστεί το παρακάτω μήνυμα, θα πρέπει να πατηθεί το RESET κουμπί της πλακέτας ESP32-CAM.

Εικόνα 2.20: Arduino IDE terminal

5.Με την ολοκλήρωση της φόρτωσης του προγράμματος, θα πρέπει να αποσυνδεθεί το GPIO0 από το GND για να βγει από το flashing mode.

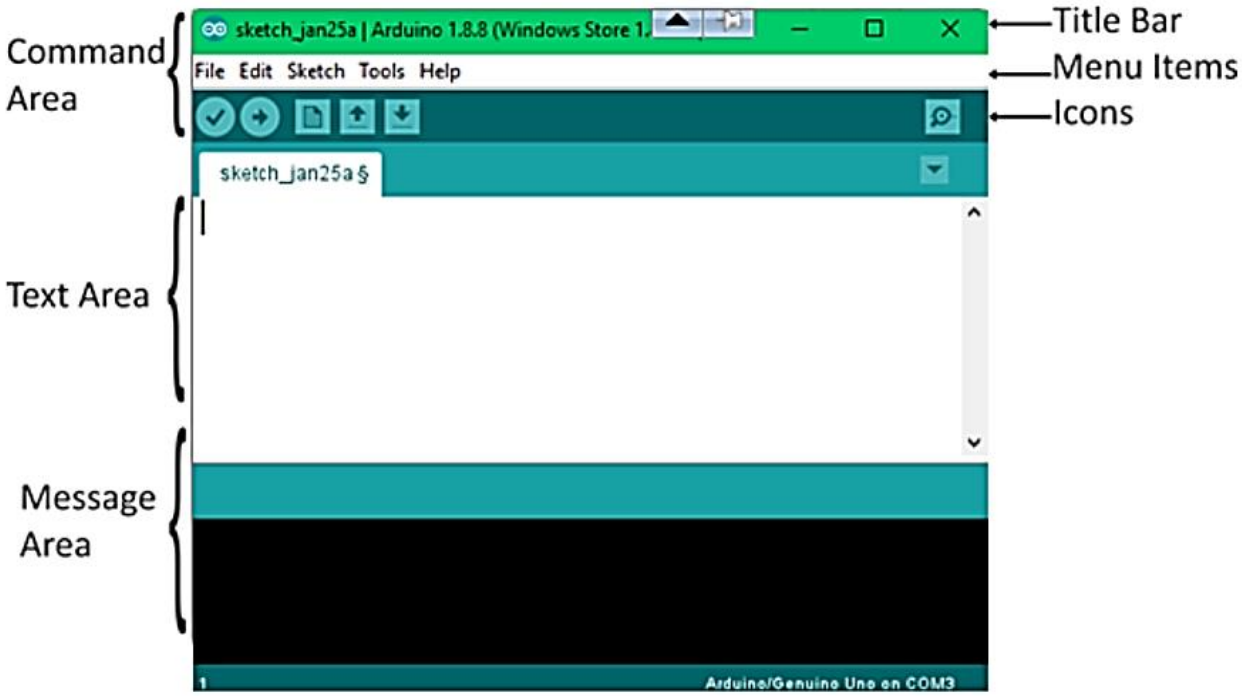
Στο σημείο αυτό είναι σημαντικό να παρουσιαστεί το διάγραμμα του κυκλώματος όπου συνδέονται όλα τα εξαρτήματα μεταξύ τους.



Εικόνα 2.21: Διάγραμμα ολόκληρου κυκλώματος

2.5 Περιβάλλον ανάπτυξης (IDE)

Το Arduino IDE είναι ένα περιβάλλον ανάπτυξης κώδικα, υλοποιημένο σε Java. Υποστηρίζει βιβλιοθήκες, περιέχει αρκετά χρήσιμα παραδείγματα και εύκολο toolbar με τις βασικές λειτουργίες που χρειάζεται ένας προγραμματιστής.



Εικόνα 2.22: Arduino IDE

Όπως φαίνεται και στην παραπάνω εικόνα, το πρόγραμμα διαχωρίζεται σε τρία (3) κύρια μέρη. Το μενού, η κυρίως περιοχή κειμένου και η περιοχή όπου εμφανίζονται μηνύματα.

Το IDE εμπεριέχει έναν κειμενογράφο, έναν μεταφραστή, ένα φορτωτή και τέλος ένα παράθυρο όπου παρακολουθείται η σειριακή λειτουργία. Δεν μπορεί το πρόγραμμα να γίνει debug και αυτό είναι ένα μειονέκτημα για τους προγραμματιστές.

Στην εργαλειοθήκη που προσφέρει το πρόγραμμα (κυκλικά κουμπιά), με τη σειρά που εμφανίζονται τα εργαλεία, κάνουν τις εξής λειτουργίες:

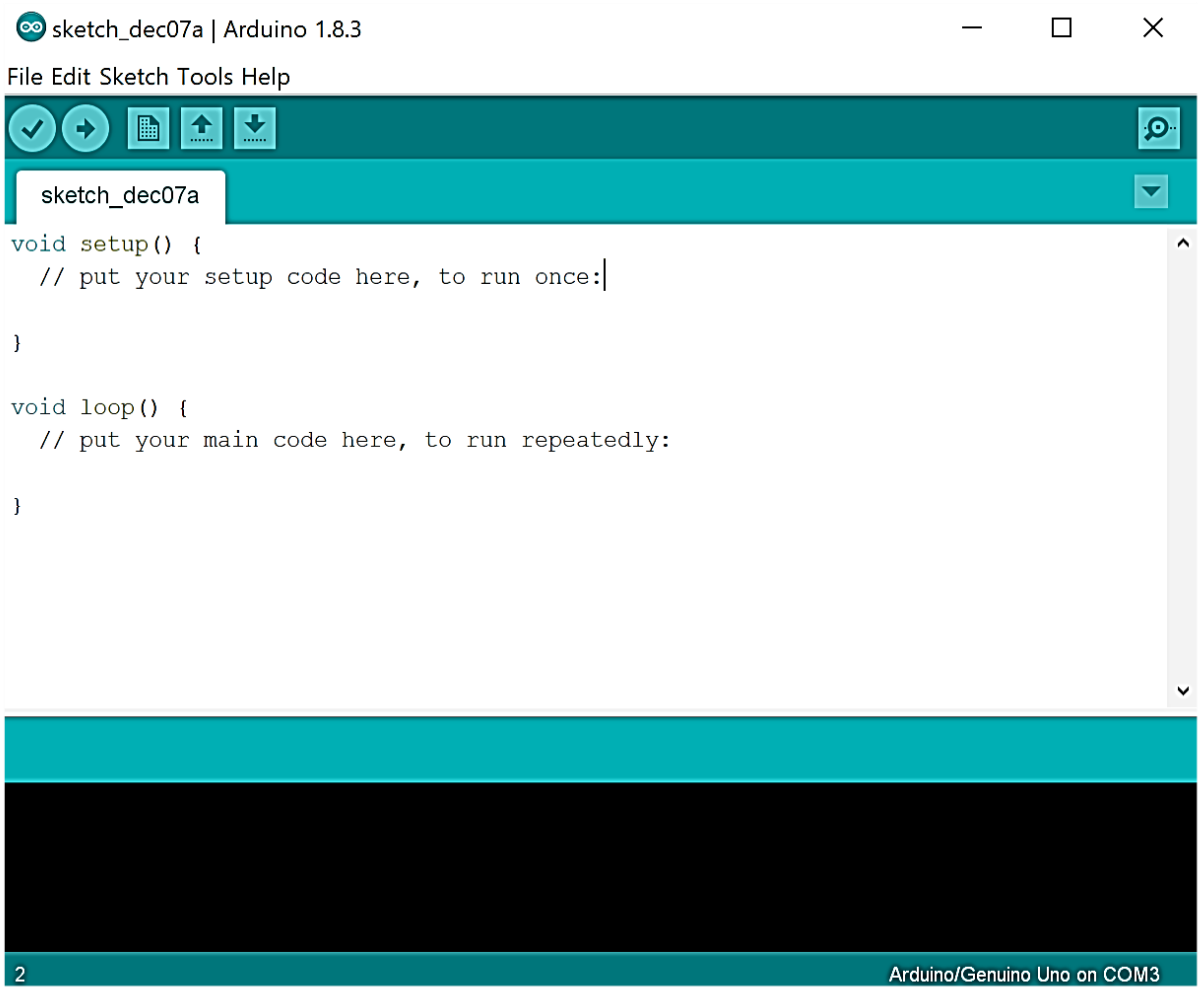
- Verify: Πραγματοποιείται η μεταγλώττιση του κώδικα και ο έλεγχος για συντακτικά λάθη.
- Upload: Κάνει verify και στη συνέχεια φορτώνει το πρόγραμμα στην πλακέτα του Arduino.
- New: Δημιουργείται ένα νέο πρόγραμμα, σβήνοντας το ήδη υπάρχον.
- Open: Ο προγραμματιστής μπορεί να φορτώσει ένα ήδη υπάρχον αποθηκευμένο πρόγραμμα.
- Save: Αποθηκεύει το υπάρχον πρόγραμμα στον υπολογιστή μας.

Το τελευταίο κυκλικό κουμπί που βρίσκεται στα δεξιά, ανοίγει ένα παράθυρο από όπου παρακολουθείται η σειριακή επικοινωνία. Η επικοινωνία αυτή πραγματοποιείται είτε μέσω USB, είτε μέσω των αντίστοιχων pins που έχουν αναφερθεί σε προγενέστερο κεφάλαιο.

2.6 Ανάλυση κώδικα

Στο παρόν κεφάλαιο πρόκειται να αναλυθούν κομμάτια κώδικα που χρησιμοποιήθηκαν για την ανάπτυξη αυτής της εργασίας.

Πιο συγκεκριμένα, με τη δημιουργία ενός νέου project στο Arduino, ο χρήστης θα παρατηρήσει την παρακάτω δομή κώδικα (εικόνα):



Εικόνα 2.23: Νέο project σε Arduino IDE

Στην ουσία, πρόκειται για δύο συναρτήσεις οι οποίες είναι ορισμένες από το σύστημα να εκτελούνται με την εκκίνηση λειτουργίας της πλακέτας. Η setup() εκτελείται μια φορά κατά την εκκίνηση, ενώ η loop() εκτελείται επαναλαμβανόμενα. Στην setup() ορίζουμε και αρχικοποιούμε τις μεταβλητές και ό,τι επιπλέον χρειάζεται για το project, ενώ στη loop() γράφεται ο κώδικας που χρειάζεται για παροδικούς ελέγχους.

2.6.1 Κώδικας Arduino

Στο πάνω μέρος του κώδικα γίνεται η εισαγωγή βιβλιοθηκών καθώς και εξωτερικών αρχείων:

```
#include <ArduinoJson.h>
#include <Servo.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
```

Στη συνέχεια αρχικοποιούνται οι μεταβλητές που χρησιμοποιούνται μέσα στο project:

```
// pins
const int pinIRout = 2;
const int pinIRin = 3;
const int motor = 4;
const int buzzer = 9;


// initial values
int IRin = 1;
int IRout = 1;
int previousIRin = 1;
int previousIRout = 1;
int servo_pos = 95;
int freeSlots = 6;
```

Έπειτα ακολουθεί ο κώδικας που βρίσκεται στην setup(). Ο κώδικας αυτός ορίζει το bit rate για τη σειριακή μετάδοση, καθώς και τα pins που θα χρησιμοποιηθούν για τα εξαρτήματα που είναι συνδεδεμένα στο Arduino. Ακόμα, ορίζει το σημείο που θα βρίσκεται η μπάρα μέσω του servo motor όπως και το αρχικό μήνυμα της οθόνης.

```
void setup()
{
  // Sets the data rate in bits per second (baud) for serial data transmission
  Serial.begin(9600);

  pinMode(pinIRin, INPUT);
  pinMode(pinIRout, INPUT);
  pinMode(buzzer, OUTPUT);

  // servo motor - bar carrier
  myservo.attach(motor);

  // init servo motor
  //myservo.write(10);

  // init lcd screen
  lcd.init();
  lcd.backlight();

  init_message();

}
```

Παρακάτω, ο κώδικας έχει χωριστεί σε αρκετές μεθόδους ώστε να είναι ευανάγνωστος και εύκολα διαχειρίσιμος.

```
void loop()
{
  delay(1000);

  IRin = digitalRead(pinIRin); //
  IRout = digitalRead(pinIRout);

  entrance_ir();
  exit_ir();
}
```

Όταν ξεκινάει η loop() «βλέπει» αν υπάρχει κάποιο όχημα σε αναμονή (είτε στην είσοδο, είτε στην έξοδο) και εκτελούνται οι δύο παρακάτω μέθοδοι για να εκκινήσουν οι διαδικασίες:

```
void entrance_ir() {
  // Entrance IR
  if (IRin == 0 && previousIRin != IRin) {
    check_car("1");
    previousIRin = IRin;
  }

  if (IRin == 1 && previousIRin != IRin) {
    close_bar();
    previousIRin = IRin;
  }
}
```

Στη μέθοδο entrance_ir() γίνεται έλεχος αν υπάρχει όχημα στην είσοδο του χώρου στάθμευσης και εφόσον υπάρχει, τότε καλείται η check_car() με παράμετρο «1» για να πραγματοποιηθεί λήψη φωτογραφίας. Η παράμετρος στέλνεται μέσω σειριακής θύρας στο ESP32-CAM και χρησιμοποιείται για να γνωρίζει από ποιόν αισθητήρα προήλθε η επικοινωνία. Αντίστοιχος είναι ο κώδικας της exit_ir() με τη μόνη διαφορά ότι η παράμετρος είναι «2». Επιπλέον, γίνεται ένας έλεγχος αν είναι ανοικτή η μπάρα και καλείται η close_bar() για να κλείσει.

```
void open_bar() {

  if (servo_pos < 20)
    return;

  // open bar carrier
  for (servo_pos = 94; servo_pos >= 10; servo_pos -= 1) {
    myservo.write(servo_pos);
    delay(30);
  }
  // delay 1 sec
  delay(1000);

}
```

Το άνοιγμα και το κλείσιμο της μπάρας πραγματοποιείται από τις μεθόδους open_bar() και close_bar() αντίστοιχα. Έχουν παρόμοια λειτουργικότητα με τη διαφορά ότι δίνεται η εντολή να γυρίσει αντίστροφα το servo motor.

```
void show_message(String msg, int line) {
  lcd.setCursor(0, line);
  lcd.print(msg);
}

void clear_message() {
  lcd.clear();
  lcd.setCursor(0, 0);
}

void init_message() {
  clear_message();
  show_message("Smart Parking", 0);
  show_message("Free slots: " + String(freeSlots), 1);
}
```

Οι μέθοδοι της παραπάνω εικόνας αφορούν την ψηφιακή οθόνη και τις ενδείξεις που θα εμφανίσει. Η show_message() ορίζει από πού θα ξεκινήσει ο cursor για να εμφανίσει το μήνυμα και σε ποια γραμμή. Η οθόνη είναι 16x2 που σημαίνει ότι μπορεί να εμφανίσει 16 χαρακτήρες ανά γραμμή έχοντας 2 γραμμές.

Η clear_message() καθαρίζει την οθόνη, και τέλος η init_message() αρχικοποιεί την οθόνη με ένα σταθερό κείμενο καθώς και τον αριθμό ελεύθερων θέσεων που λαμβάνει από το API.

```
void check_car(String gate) {
  clear_message();

  if (gate.indexOf("1") >= 0) {
    clear_message();
    show_message("Vehicle", 0);
    show_message("has arrived", 1);
  } else {
    clear_message();
    show_message("Vehicle", 0);
    show_message("comes out", 1);
  }

  Serial.print(gate);

  String responseData = "";

  delay(50); // wait for the serial port
  while (Serial.available() <= 0) {}
  responseData = Serial.readString();

  StaticJsonDocument<200> json;
  DeserializationError error = deserializeJson(json, responseData);
  // check if parsing succeeds
  if (error) {
    clear_message();
    show_message("Fix car position", 0);
    error_sound();
    return;
  }
```

```
  // Fetch values
  bool succeeded = json["succeeded"];

  if (succeeded) {
    // Fetch values
    String name = json["name"];
    String surname = json["surname"];
    String plate = json["plate"];
    String status = json["status"];
    int cost = json["cost"];
    bool lowCoins = json["lowCoins"];
    freeSlots = json["freeSlots"];

    if (status.equals("InParking")) {
      clear_message();
      show_message("Welcome", 0);
      show_message(name, 1);
    }else {
      clear_message();
      show_message("Goodbye", 0);
      show_message(name, 1);
    }
    success_sound();
    open_bar();
  }else {
    // Fetch value
    String message = json["message"];

    clear_message();
    show_message(message, 0);
    error_sound();
  }

  delay(1000);
}
```

Η check_car() ελέγχει την παράμετρο που λαμβάνει και εμφανίζει μήνυμα στην οθόνη για το καλωσόρισμα ή την αποχώρηση του οχήματος. Στη συνέχεια, στέλνει στο ESP32-CAM την παράμετρο ώστε να ξεκινήσει τη λήψη και αποστολή φωτογραφίας στον server. Εφόσον ολοκληρωθεί αυτή η διαδικασία, το αποτέλεσμα επιστρέφει από το ESP32-CAM μέσω σειριακής επικοινωνίας στο Arduino. Το αποτέλεσμα είναι ένα JSON όπου μέσω της βιβλιοθήκης που υπάρχει στην αρχή του κώδικα, γίνεται αναγνώριση και ορίζεται σε μεταβλητές για να χρησιμοποιηθούν στη συνέχεια. Αν υπάρξει κάποιο πρόβλημα με την

αναγνώριση της απάντησης, τότε εμφανίζεται κατάλληλο μήνυμα στην οθόνη, διαφορετικά, γίνεται έλεγχος αν το όχημα είναι εντός ή εκτός χώρου στάθμευσης ώστε να εμφανίσει αντίστοιχο μήνυμα στην οθόνη. Τέλος, με τη χρήση buzzer υπάρχει ηχητική ειδοποίηση για μια επιτυχημένη ή αποτυχημένη προσπάθεια.

```
void success_sound() {
  tone(buzzer, 1000); //Send 1KHz sound signal
  delay(700); // delay 1sec
  noTone(buzzer); // stop sound
}

void error_sound() {
  tone(buzzer, 1500); //Send 1KHz sound signal
  delay(300); // delay 1sec
  noTone(buzzer); // stop sound

  delay(300); // delay 1sec

  tone(buzzer, 1500); //Send 1KHz sound signal
  delay(300); // delay 1sec
  noTone(buzzer); // stop sound
}
```

Για να μπορέσει το buzzer να ηχήσει με κατάλληλους ήχους, έχουν οριστεί δύο μέθοδοι για την επιτυχή και αποτυχή προσπάθεια. Η success_sound() δηλώνει στο buzzer να ηχήσει στη συχνότητα 1KHz κι έχει οριστεί μια καθυστέρηση 700ms όπου είναι η χρονική διάρκεια του σήματος. Τέλος, με τη noTone() δηλώνει στο buzzer να σταματήσει. Με τον ίδιο τρόπο λειτουργεί και η error_sound() με τη διαφορά ότι γίνεται μια μικρή παύση της τάξης των 300ms ανάμεσα στους δύο ήχους και ότι οι ήχοι είναι σε διαφορετική συχνότητα ώστε να γίνεται κατανοητό από τον επισκέπτη-πελάτη ότι πρόκειται για μια άλλη κατάσταση.

2.6.2 Κώδικας ESP32-CAM

Για την ανάπτυξη του κώδικα στο εξάρτημα ESP32-CAM χρησιμοποιήθηκε ένα νέο project και φορτώθηκε ο κώδικας μέσω καλωδίου USB όπως περιγράφεται στα προηγούμενα κεφάλαια.

```
#include <Arduino.h>
#include <WiFi.h>
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"
#include "esp_camera.h"

const char* ssid = "Casperakos";
const char* password = "@6993934230@";

String serverName = "192.168.1.20";   // REPLACE WITH YOUR IP ADDRESS
//String serverName = "example.com";   // OR REPLACE WITH YOUR DOMAIN NAME
String serverPath = "/api/arduino/ocr-space/recognize-car";
const int serverPort = 44313; // port

WiFiClient client;
```

Αρχικά, κι εδώ τοποθετήθηκαν οι βιβλιοθήκες που χρησιμοποιούνται μέσα στον κώδικα. Στη συνέχεια, ορίστηκαν κάποιες σταθερές όπου είναι το όνομα του δρομολογητή του δικτύου, ο κωδικός σύνδεσης, το όνομα του server, το path του server που θα γίνεται η αποστολή της φωτογραφίας (end point).

```
void setup() {
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);
  Serial.begin(9600);

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }

  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
```

Στη συνάρτηση setup() αρχικοποιήθηκε η κάμερα με τις παραμέτρους της, το μέγεθος της εικόνας, η ποιότητα της κ.ά.

```
void loop() {
  while (Serial.available()) {
    String carHasArrived = Serial.readString();

    if (carHasArrived.indexOf("1") >= 0 || carHasArrived.indexOf("2") >= 0)
      String response = sendPhoto();

      int startsJson = response.indexOf('{');
      int endsJson = response.indexOf('}');
      response = response.substring(startsJson, endsJson + 1);

      Serial.println(response);
      delay(50);
    }
  }
}
```

Στη μέθοδο loop() που επαναλαμβάνεται, γίνεται ένας ατέρμονος έλεγχος στη σειριακή επικοινωνία για να ελεγχθεί αν έχουν έρθει δεδομένα από το Arduino. Στην περίπτωση που διαβάσει κάποια δεδομένα, γίνεται έλεγχος αν αυτά είναι «1» ή «2» όπου είναι ο αισθητήρας εισόδου και εξόδου αντίστοιχα. Εφόσον ισχύουν αυτά, καλείται η sendPhoto() και κρατάει μόνο το JSON καθώς υπάρχουν και κάποιοι ειδικοί χαρακτήρες όπου καθαρίζονται πριν αποσταλεί στο Arduino.

```
String sendPhoto() {
  String getAll;
  String getBody;

  camera_fb_t * fb = NULL;
  fb = esp_camera_fb_get();
  if(!fb) {
    Serial.println("Camera capture failed");
    delay(1000);
    ESP.restart();
  }

  if (client.connect(serverName.c_str(), serverPort)) {
    String head = "--SmartParkingSolutions\r\nContent-Disposition: form-data; name=\"files\"; filename=\"esp32-cam.jpg\"\r\nContent-Type: image/jpeg\r\n\r\n";
    String tail = "\r\n--SmartParkingSolutions--\r\n";

    uint32_t imageLen = fb->len;
    uint32_t extraLen = head.length() + tail.length();
    uint32_t totalLen = imageLen + extraLen;

    client.println("POST " + serverPath + " HTTP/1.1");
    client.println("Host: " + serverName);
    client.println("Content-Length: " + String(totalLen));
    client.println("Content-Type: multipart/form-data; boundary=SmartParkingSolutions");
    client.println();
    client.print(head);

    uint8_t *fbBuf = fb->buf;
    size_t fbLen = fb->len;
    for (size_t n=0; n<fbLen; n=n+1024) {
      if (n+1024 < fbLen) {
        client.write(fbBuf, 1024);
        fbBuf += 1024;
      }
      else if (fbLen%1024>0) {
        size_t remainder = fbLen%1024;
        client.write(fbBuf, remainder);
      }
    }
    client.print(tail);

    esp_camera_fb_return(fb);
```

Στη μέθοδο sendPhoto() υλοποιείται η κλήση POST, και προστίθενται σε αυτήν τα κατάλληλα δεδομένα όπως το όνομα του αρχείου, τα δεδομένα της φωτογραφίας, το μήκος κ.ά. Μετά την αποστολή της κλήσης, υπάρχει μια μικρή αναμονή για τη λήψη της απάντησης και γίνεται επιστροφή της απάντησης στην μέθοδο loop() από όπου καλείται η sendPhoto().

ΚΕΦΑΛΑΙΟ 3: ΔΙΑΔΙΚΤΥΑΚΕΣ ΕΦΑΡΜΟΓΕΣ ΚΑΙ ΥΠΗΡΕΣΙΕΣ

3.1 Εισαγωγή

Στο συγκεκριμένο κεφάλαιο θα αναλυθεί το σύστημα αυτοματοποιημένων υπηρεσιών οχημάτων με χρονοχρέωση σε επίπεδο κώδικα. Πιο συγκεκριμένα, όλες εκείνες οι ενέργειες και λειτουργίες που χρειάζονται για τη διαχείριση, λειτουργικότητα και αποθήκευση δεδομένων ενός τέτοιου συστήματος.

Αρχικά, αναφέρονται οι απαιτήσεις που καλείται να αντιμετωπίσει ένα τέτοιο σύστημα, για πιο λόγο θα πρέπει να επιλεγούν συγκεκριμένες αρχιτεκτονικές σχεδίασης αντί άλλων και ποιες λύσεις θα μπορούσαν να διευκολύνουν την επίλυση συγκεκριμένων προβλημάτων.

Στη συνέχεια γίνεται μια μικρή αναφορά στις τεχνολογίες που χρησιμοποιήθηκαν σε επίπεδο κώδικα και πιο συγκεκριμένα, στις γλώσσες προγραμματισμού και τα πακέτα πάνω στα οποία υλοποιήθηκε το συγκεκριμένο σύστημα.

Σειρά έχουν τα εργαλεία που χρειάστηκαν κατά τη δημιουργία του συστήματος, όπως τα εργαλεία που δίνουν τη δυνατότητα εκτέλεσης κώδικα και διαχείρισης βάσεων δεδομένων και εξυπηρετητών.

Στο επόμενο κεφάλαιο εξετάζονται οι αρχιτεκτονικές που ακολουθήθηκαν κατά τη συγγραφή του κώδικα. Με τη χρήση αυτών, δύναται να αποφευχθούν συγκεκριμένες δυσκολίες που αντιμετωπίζουν συστήματα τέτοιας φύσεως και βελτιώνουν την ποιότητα του κώδικα.

Ακολουθεί η περιγραφή της διαδικασίας αναγνώρισης μιας πινακίδας ενός οχήματος. Εξηγείται ο όρος OCR και γίνεται μια ιστορική αναδρομή στις πιο σημαντικές ημερομηνίες του. Στη συνέχεια παρουσιάζεται η υπηρεσία OCR Space που χρησιμοποιείται για την αναγνώριση κειμένου μέσα από εικόνες μαζί με παραδείγματα αιτημάτων και απαντήσεων προς την υπηρεσία αυτή.

Τέλος, γίνεται μια ανάλυση των σημαντικότερων σημείων του κώδικα, όπως η διαχείριση κλήσεων και σφαλμάτων, η αυθεντικοποίηση ενός χρήστη, η βάση δεδομένων, τα APIs και οι εφαρμογές χρήστη.

3.2 Απαιτήσεις

Οι απαιτήσεις κάθε συστήματος είναι ανάλογες με την πολυπλοκότητα των εργασιών που έχουν να αντιμετωπίσουν και του πλήθος των χρηστών που καλούνται να εξυπηρετήσουν. Το μέγεθος των απαιτήσεων που πρέπει να καλυφθούν οδηγεί και στις τεχνολογίες που θα χρησιμοποιηθούν.

Ένα σύστημα μικρού βεληνεκούς μπορεί να αρκεστεί σε λύσεις που συνδυάζουν front-end, back-end, database όλα σε ένα σημείο. Αυτά τα συστήματα ονομάζονται μονολιθικά και μπορούν να υλοποιηθούν σχετικά γρήγορα. Τα προβλήματα επικοινωνίας μεταξύ των επιπέδων του συστήματος είναι μειωμένα και χρησιμοποιούνται πιο απλές τεχνικές και εργαλεία κατά τη δημιουργία τους. Βασικό τους μειονέκτημα είναι ο περιορισμός που έχουν ως προς την αναβάθμιση και επέκτασή τους καθώς επίσης και στις τεχνολογίες που μπορούν να χρησιμοποιήσουν.

Ένα πιο δυναμικό σύστημα όπως αυτό της διαχείρισης ενός παρκινγκ δε θα μπορούσε να ανήκει στην παραπάνω κατηγορία, γιατί θα πρέπει να μπορεί να υποστηρίξει μικρού και μεγάλου μεγέθους υπηρεσίες. Για τον λόγο αυτό, είναι αναγκαίο να χρησιμοποιηθούν τεχνικές που επιτρέπουν την επέκταση και αναβάθμισή του όπως επίσης και τη χρήση πολλαπλών τεχνολογιών. Πάνω σε αυτό το πρόβλημα έρχονται να δώσουν τεχνικές λύσεις, τα microservices και τα gateways.

Όταν αυξάνεται η κίνηση και η πολυπλοκότητα ενός συστήματος, αυξάνεται και η χρήση βασικών πόρων όπως η μνήμη, η επεξεργαστική ισχύ και η ταχύτητα δικτύου. Συχνά αυτό αντιμετωπίζεται με κάθετη αναβάθμιση, προσθέτοντας δηλαδή περισσότερη μνήμη ή επεξεργαστική ισχύ στο σύστημα. Επειδή, όμως, αυτού του είδους η αναβάθμιση δεν είναι πανάκεια και έχει ένα τέλμα, τα μεγάλα συστήματα έχουν την ανάγκη να υποστηρίζουν και οριζόντιες αναβαθμίσεις.

Υποστηρίζοντας μια οριζόντια αναβάθμιση δημιουργείται το πρόβλημα της διαχείρισης των εξουσιοδοτημένων χρηστών. Μια παραδοσιακή μέθοδος διαχείρισής τους είναι η χρήση sessions. Αυτά δημιουργούνται κατά την εισαγωγή του χρήστη στο σύστημα και

θα πρέπει να παραμένουν ενεργά για όση ώρα ο χρήστης χρησιμοποιεί την εφαρμογή, έτσι ώστε να μην χρειάζεται να κάνει είσοδο σε κάθε του ενέργεια. Σε περίπτωση πολλαπλών server κάτι τέτοιο περιπλέκει την κατάσταση. Θα πρέπει να βρεθεί ένας τρόπος, όπου ο εξυπηρετητής θα μπορεί να αναγνωρίσει τον χρήστη, ο οποίος έκανε το ερώτημα και αν έχει συνδεθεί, να του παρέχει πρόσβαση στις υπηρεσίες του.

Μερικοί από τους τρόπους επίλυσης του προβλήματος της αυθεντικοποίησης ενός χρήστη βασίζονται σε Balancers με Sticky Sessions, χρήση βάσης δεδομένων ή τεχνικές όπως η JTW. Οι δυο πρώτοι τρόποι επιβαρύνουν εξ' ολοκλήρου τους εξυπηρετητές, οπότε απαιτείται σωστή διαχείριση πόρων. Ο τελευταίος, μοιράζει εν μέρει το βάρος σε front-end και back-end με βασική προϋπόθεση ότι οι εφαρμογές χρήστη είναι υπεύθυνες για τη σωστή αποθήκευση και χρήση του Token αυθεντικοποίησης.

Μια ακόμα απαίτηση ενός συστήματος που δύναται να αυξήσει τις λειτουργίες του στο μέλλον είναι αυτή της εγγύησης της σωστής λειτουργίας των βασικών διαδικασιών. Αυτό μπορεί να επιτευχθεί με τη χρήση ελέγχων οι οποίοι θα εκτελούνται σε κάθε νέα αλλαγή. Οι έλεγχοι αυτοί συνήθως γράφονται τόσο από τους ίδιους τους προγραμματιστές που υλοποίησαν το σύστημα (unit tests) όσο και από εξωτερικούς (integration tests), για να διασφαλίσουν, ότι το πρόγραμμα ανταποκρίνεται σε ενέργειες που πρέπει να κάνει και ταυτόχρονα διαχειρίζεται ακραίες καταστάσεις. Μπορούν, επίσης, να χρησιμοποιήσουν εικονικά συστήματα ή καταστάσεις αντί των πραγματικών, όπως για παράδειγμα μια εικονική βάση δεδομένων ή μια υποτιθέμενη απάντηση από μια REST κλήση, για να βελτιώσουν την αποτελεσματικότητα και την ακρίβεια των ελέγχων.

Στις απαιτήσεις ελέγχου ενός συστήματος έρχονται να δώσουν λύσεις αρχιτεκτονικές και πρότυπα, όπως η Clean Architecture, το Dependency Injection, το SOLID pattern και άλλες παρόμοιες τεχνικές. Με τη χρήση τους γίνεται πιο εύκολη η δημιουργία ελέγχων και πιο ξεκάθαρο το αντικείμενο που θα πρέπει να ελεγχθεί.

3.3 Τεχνολογίες

Στο κεφάλαιο αυτό θα γίνει μια μικρή αναφορά των τεχνολογιών που χρησιμοποιήθηκαν τόσο σε front-end όσο και σε back-end αλλά και σε επίπεδο βάσεων δεδομένων για τη δημιουργία του συστήματος αυτοματοποιημένων υπηρεσιών στάθμευσης.

3.3.1 .NET Core

Η πλατφόρμα .NET είναι ένα πλαίσιο λογισμικού που δημιουργήθηκε από την Microsoft με σκοπό τη δημιουργία πολλών διαφορετικών εφαρμογών. Είναι γραμμένη πάνω σε C++ και C# και υποστηρίζει πλήρως γλώσσες προγραμματισμού όπως η C#, F# και η VB.NET. Χρησιμοποιεί μια πληθώρα γλωσσών προγραμματισμού, εργαλείων και βιβλιοθηκών για τη δημιουργία ιστοσελίδων, desktop και mobile εφαρμογών καθώς επίσης και IoT.

Παλαιότερες εκδόσεις του χρησιμοποιούνταν κυρίως από περιβάλλοντα Windows της Microsoft. Αυτό όμως έχει αλλάξει στις νεότερες εκδόσεις του, στις εκδόσεις .NET Core και πλέον υποστηρίζεται σε πολλές πλατφόρμες (cross-platform) όπως Linux και macOS. Ο κώδικάς του είναι διαθέσιμος στο GitHub με άδεια χρήσης του MIT και μπορεί να χρησιμοποιηθεί από οποιονδήποτε προγραμματιστή.

3.3.2 C#

Η C# είναι μια γλώσσα προγραμματισμού ασφαλούς τύπου βασισμένη στον αντικειμενοστραφή προγραμματισμό. Χρησιμοποιείται για τη δημιουργία εφαρμογών που εκτελούνται στο οικοσύστημα της πλατφόρμας .NET. Προέρχεται από την οικογένεια γλωσσών προγραμματισμού της C και μπορεί εύκολα να χρησιμοποιηθεί από προγραμματιστές που έχουν γνώση της C, C++, Java και JavaScript.

Αξιοσημείωτα χαρακτηριστικά αυτής της γλώσσας είναι η υποστήριξη εκφράσεων τύπου lambda (lambda expressions), generics, σύνταξη LINQ για επικοινωνία με διάφορες πηγές δεδομένων όπως βάσεις δεδομένων, αυτόματο καθαρισμό «σκουπιδιών» (automatic garbage collection) και αυτόματη διαχείριση μνήμης.

3.3.3 MS SQL Server

Η Microsoft SQL Server είναι ένα σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων (RDBMS) που αναπτύχθηκε από την Microsoft. Έχει χτιστεί πάνω στην SQL, μια βασική γλώσσα προγραμματισμού για αλληλεπίδραση με σχεσιακές βάσεις δεδομένων και η κύρια γλώσσα που χρησιμοποιείται για την εκτέλεση ερωτημάτων προς τη βάση είναι η Transact-SQL (T-SQL).

Η T-SQL είναι μια επέκταση που δημιούργησε η Microsoft για τους SQL Servers. Παρέχει οδηγίες που επεκτείνουν το σύνολο των εντολών της SQL όσον αφορά τον ορισμό και χειρισμό δεδομένων, συμπεριλαμβανομένων των ρυθμίσεων για τον SQL Server, της ασφάλειας και της διαχείρισης στατιστικών βάσεων δεδομένων.

3.3.4 Node.js

Η Node.js είναι μια πλατφόρμα ανοιχτού κώδικα, σχεδιασμένη να εκτελείται σε πολλά περιβάλλοντα και βασισμένη στην Javascript. Χαρακτηρίζεται από την ασύγχρονη επικοινωνία μεταξύ των υπολογιστικών πόρων, η οποία με τη σειρά της διευκολύνει την εκτέλεση του κώδικα χωρίς να μένει ανενεργός ο επεξεργαστής. Αν και έχει σχεδιαστεί να χρησιμοποιεί ένα νήμα διεργασιών (thread), μπορεί να χρησιμοποιήσει πολλαπλούς επεξεργαστές.

Χρησιμοποιείται κυρίως για τη δημιουργία επεκτάσιμων διακομιστών χρησιμοποιώντας ένα απλοποιημένο μοντέλο event-driven προγραμματισμού. Το μοντέλο αυτό χρησιμοποιεί callbacks για να σηματοδοτήσει την ολοκλήρωση μιας εργασίας.

3.3.5 Vue Js

Η Vue είναι ένα πλαίσιο λογισμικού της JavaScript, βασισμένη στο μοντέλο του mode-view-viewmodel, με σκοπό τη δημιουργία διεπαφών χρήστη. Έχει σχεδιαστεί εξ’ αρχής για να καλύπτει τις ανάγκες ενός συστήματος με ολοένα και αυξανόμενες απαιτήσεις. Μπορεί δηλαδή να χρησιμοποιηθεί για δημιουργία μιας βιβλιοθήκης έως και μιας πολύπλοκης ιστοσελίδας, όπως οι εφαρμογές μιας σελίδας (sigle-page applications).

Βασικά χαρακτηριστικά του πλαισίου αυτού είναι τα συστατικά στοιχεία (components), μικρές μονάδες επαναχρησιμοποιήσιμου κώδικα, και τα πρότυπα (templates) που είναι βασισμένα στην HTML. Συνδέοντάς τα, επιτυγχάνεται το δέσιμο του φορτωμένου εγγράφου με τα δεδομένα του αντικειμένου της Vue.

Επιπλέον χαρακτηριστικά της Vue είναι επίσης το σύστημα “Reactivity”, που διαθέτει τα εφέ μετάβασης όταν αντικείμενα εισέρχονται, ενημερώνονται και αφαιρούνται από την εφαρμογή, και το σύστημα δρομολόγησης (routing), που βοηθάει στην πλοήγηση μέσα σε εφαρμογές μονής σελίδας.

3.4 Εργαλεία

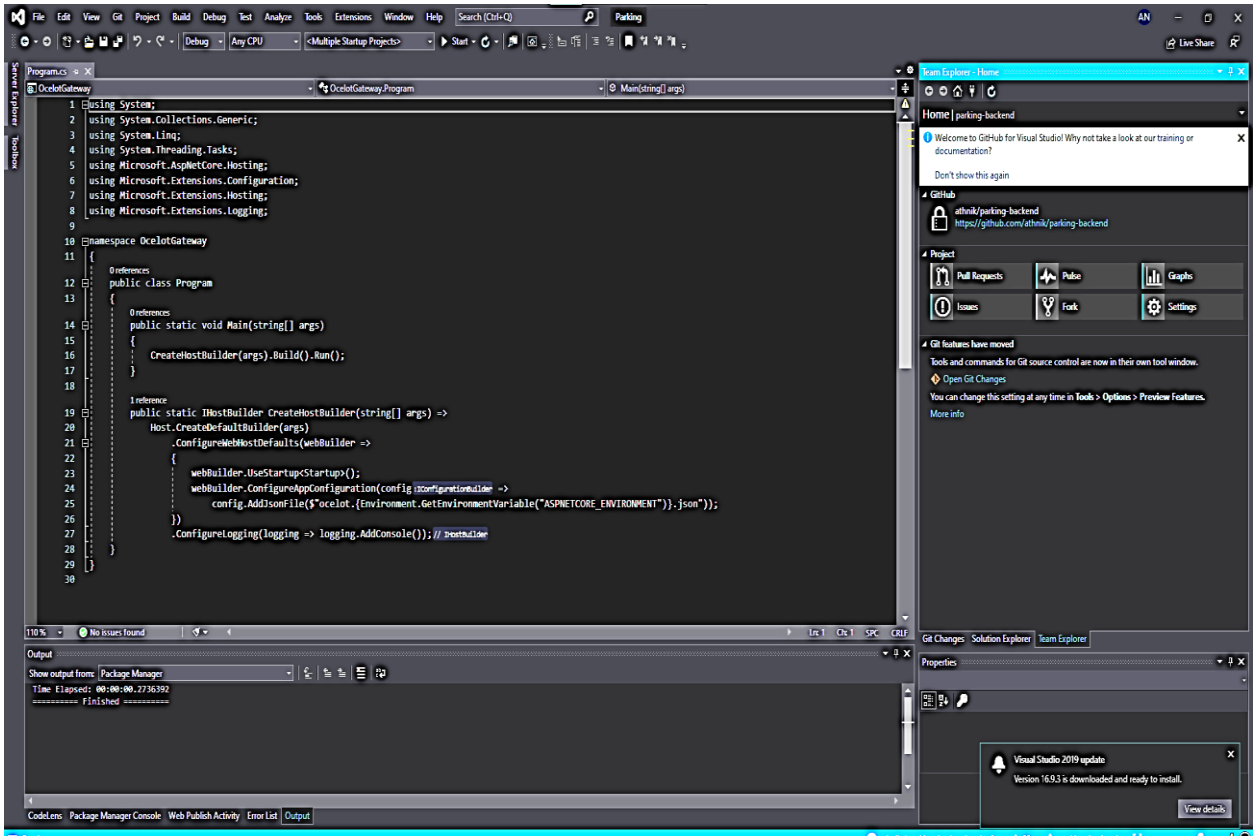
Το κεφάλαιο αυτό περιγράφει τα εργαλεία που χρησιμοποιήθηκαν για τη δημιουργία των υπηρεσιών διαδικτύου, των βάσεων δεδομένων καθώς επίσης και των ιστοσελίδων διαχείρισης χρήστη και διαχειριστή.

3.4.1 Microsoft Visual Studio

Το Microsoft Visual Studio είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) της Microsoft. Προσφέρει μια μεγάλη γκάμα εργαλείων και λειτουργιών για την κατασκευή ιστοσελίδων, WEB APIs και εφαρμογών υπολογιστών κυρίως για Windows περιβάλλοντα. Χρησιμοποιεί κυρίως γλώσσες προγραμματισμού, που αναπτύσσει η ίδια η εταιρεία όπως οι C#, F#, Visual Basic και με επιπλέον βιβλιοθήκες μπορεί να υποστηρίξει επιπλέον γλώσσες όπως VueJs, PHP κ.ά.

Το εργαλείο αυτό, στις τελευταίες του εκδόσεις και πιο συγκεκριμένα στην έκδοση 2019, κάνει χρήση του GitHub, ενός συστήματος που διευκολύνει την καταγραφή των εκδόσεων ενός προγράμματος (version control system).

Άλλα βασικά χαρακτηριστικά του εργαλείου αυτού είναι ότι μπορεί να διαχειριστεί βάσεις δεδομένων, όπως επίσης και να συνδεθεί με το Azure. Έτσι, προσφέρει μια ενιαία εμπειρία στον προγραμματιστή και τον διευκολύνει να διαχειριστεί πολύπλοκες εργασίες που απαιτούν τη χρήση πολλαπλών εργαλείων.

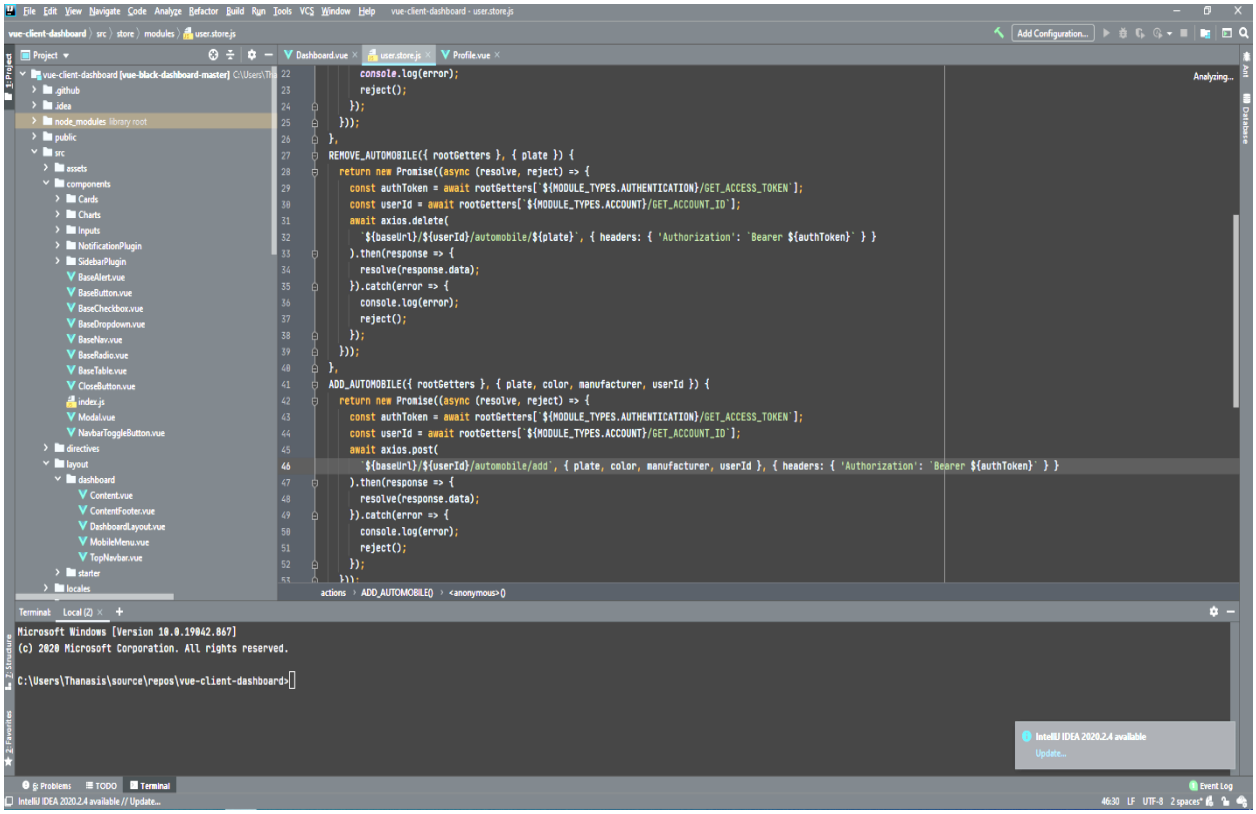


Εικόνα 3.1: Περιβάλλον εργασίας MS Visual Studio

3.4.2 IntelliJ IDEA

Το IntelliJ IDEA είναι και αυτό ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) λογισμικού, το οποίο το αναπτύσσει η εταιρεία Jetbrain. Υποστηρίζει κυρίως γλώσσες προγραμματισμού ανοιχτού κώδικα και για την χρήση επιπλέον γλωσσών είναι απαραίτητη η εγκατάσταση κάποιας βιβλιοθήκης.

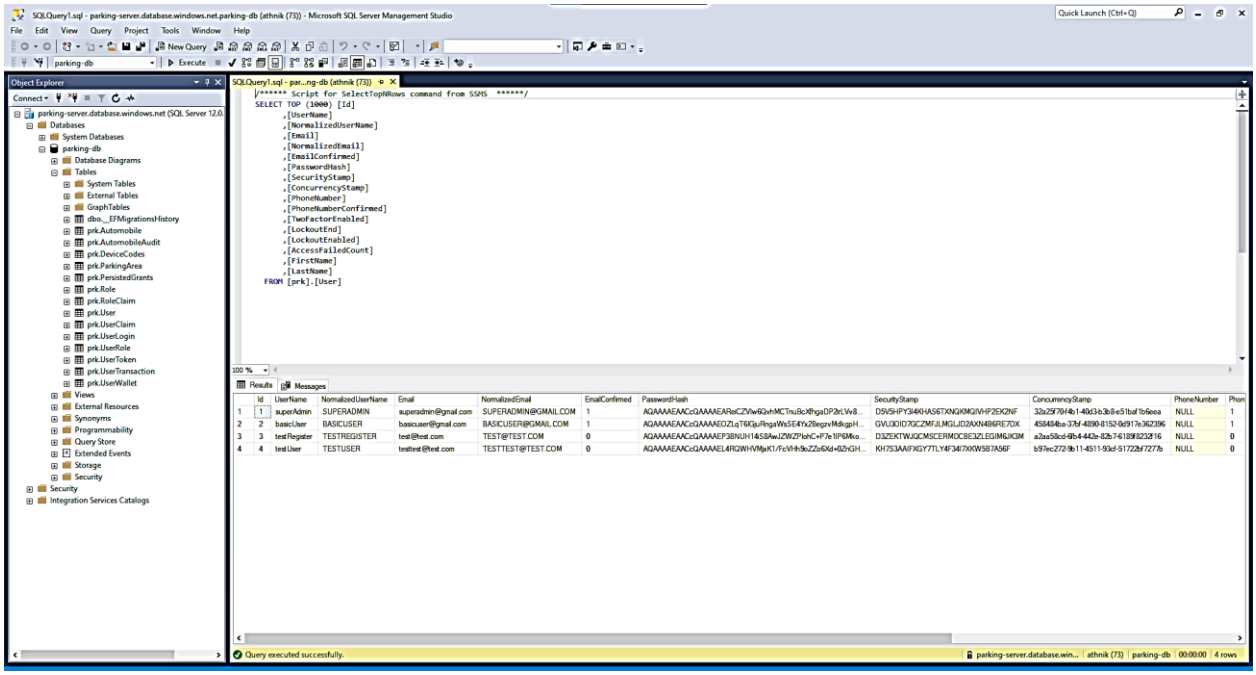
Στα προεγκατεστημένα εργαλεία του συμπεριλαμβάνονται λειτουργίες version control για συστήματα όπως Git, Mercurial, Perforce και SVN, σύνδεση με βάσεις δεδομένων όπως MS SQL Server, Oracle, PostgreSQL, SQLite και MySQL και τέλος build/packaging εργαλεία για grunt, gradle και SBT εφαρμογές.



Εικόνα 3.2: Περιβάλλον Εργασίας IntelliJ IDEA

3.4.3 SQL Server Management Studio (SSMS)

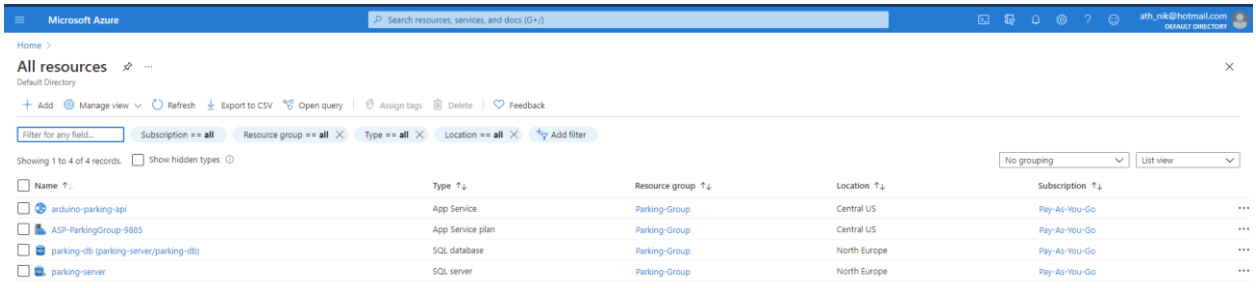
Το SQL Server Management Studio είναι ένα ολοκληρωμένο περιβάλλον της Microsoft και χρησιμοποιείται για τη διαμόρφωση και διαχείριση οποιασδήποτε βάσης SQL. Χρησιμοποιώντας εμπλουτισμένους επεξεργαστές σεναρίων και γραφικές απεικονίσεις παρέχει πρόσβαση σε προγραμματιστές και διαχειριστές βάσεων δεδομένων διαφόρων επιπέδων δεξιοτήτων.



Εικόνα 3.3: Περιβάλλον εργασίας SSMS

3.4.4 Microsoft Azure

Το Microsoft Azure είναι μια δημόσια πλατφόρμα υπολογιστικού νέφους (cloud computing) της Microsoft. Υποστηρίζει διάφορες γλώσσες προγραμματισμού, εργαλεία και πλατφόρμες είτε της Microsoft είτε άλλων κατασκευαστών. Χρησιμοποιείται για τη δημιουργία, τον έλεγχο, την ανάπτυξη και τη διαχείριση εφαρμογών και υπηρεσιών. Μέσω του γραφικού του περιβάλλοντος παρέχει εύκολη πρόσβαση και διαχείριση των υπηρεσιών του χρήστη.



Εικόνα 3.4: Πάνελ διαχείρισης MS Azure

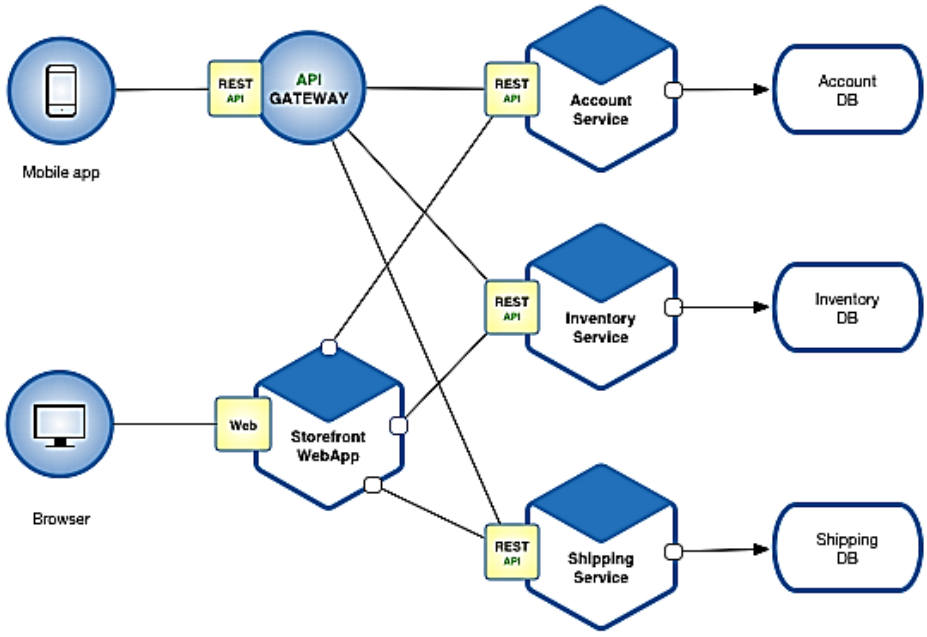
3.5 Αρχιτεκτονική

Στο κεφάλαιο αυτό θα αναλυθούν μερικές τεχνικές ανάπτυξης κώδικα και αυθεντικοποίησης, που χρησιμοποιήθηκαν στο σύστημα αυτοματοποιημένων υπηρεσιών στάθμευσης. Οι τεχνικές αυτές επιλέχθηκαν με βασικό γνώμονα την καλύτερη διαχείριση δεδομένων και ανάπτυξης-συντήρησης του συστήματος. Επίσης δόθηκε βάση και στην επεκτασιμότητα που μπορεί να έχει ένα τέτοιο σύστημα.

3.5.1 Microservices

Τα Microservices είναι μια τεχνική ανάπτυξης εφαρμογών με τρόπο τέτοιο, έτσι ώστε το συνολικό σύστημα να αποτελείται από μικρότερες ανεξάρτητες μονάδες λογισμικού που αλληλοεπιδρούν μεταξύ τους. Κύριος σκοπός της τεχνικής αυτής είναι να διαχωρίσει το σύστημα σε αυτόνομες μονάδες υπηρεσιών, έτσι ώστε να διευκολύνει τη διαχειρισιμότητά τους.

Συνήθως η κάθε μονάδα/υπηρεσία διαχειρίζεται πληροφορίες που αφορούν μια συγκεκριμένη οντότητα. Για παράδειγμα σε ένα σύστημα e-shop η κάθε υπηρεσία θα διαχειριζόταν ξεχωριστά τις πληροφορίες του λογαριασμού χρήστη (Account Service), του καταλόγου προϊόντων (Inventory Service), των αποστολών (Shipping Service) κ.ά., όπως φαίνεται στην παρακάτω εικόνα.



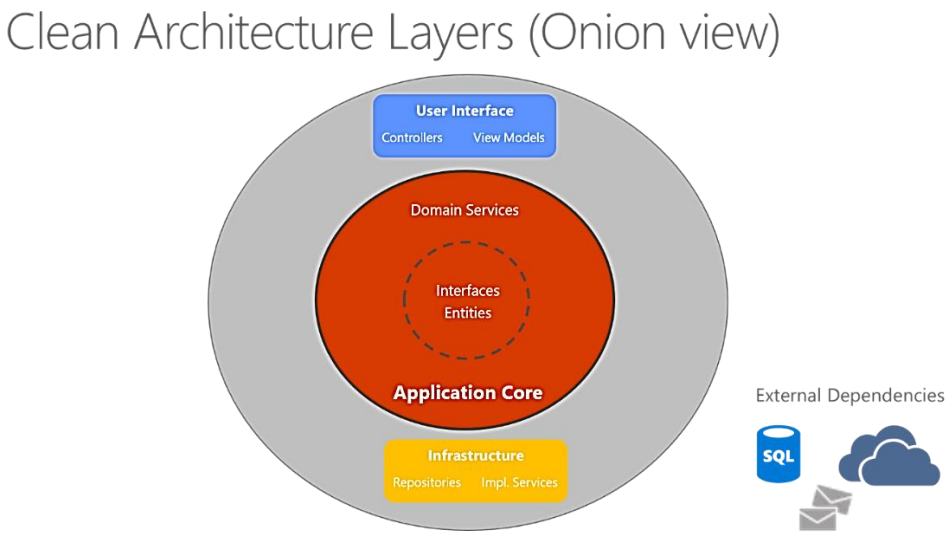
Εικόνα 3.5: Παράδειγμα microservices

Με τον τρόπο αυτό επιτυγχάνεται η πλήρης ανεξαρτητοποίηση των υπηρεσιών μεταξύ τους, κάτι που δημιουργεί πολλαπλά οφέλη. Αρχικά, διαφορετικές ομάδες μπορούν να αναπτύσσουν ταυτόχρονα την κάθε μια υπηρεσία σε όποια γλώσσα προγραμματισμού είναι πιο εξειδικευμένη η ομάδα αυτή. Η αναβάθμιση ή παύση μιας υπηρεσίας δεν απαιτεί πάντα τη συνολική παύση της εφαρμογής. Αν κάποια από τις υπηρεσίες δέχεται περισσότερο φόρτο εργασίας, μπορεί να εφαρμοστεί κάθετη ή οριζόντια αναβάθμιση υλικοτεχνικού υλικού μόνο για τη συγκεκριμένη υπηρεσία. Ο εντοπισμός σφαλμάτων και η ανάλυση προβλημάτων γίνεται πιο γρήγορα και στοχευμένα.

3.5.2 Clean architecture

Η καθαρή αρχιτεκτονική (Clean Architecture) ή αλλιώς η αρχιτεκτονική του κρεμμυδιού (Onion Architecture) είναι ένας τρόπος οργάνωσης του κώδικα, των οντοτήτων και των συνδέσμων μεταξύ τους, που έχει ως στόχο την ανεξαρτητοποίηση των επιπέδων ενός συστήματος καθώς επίσης και τον ευκολότερο έλεγχό τους.

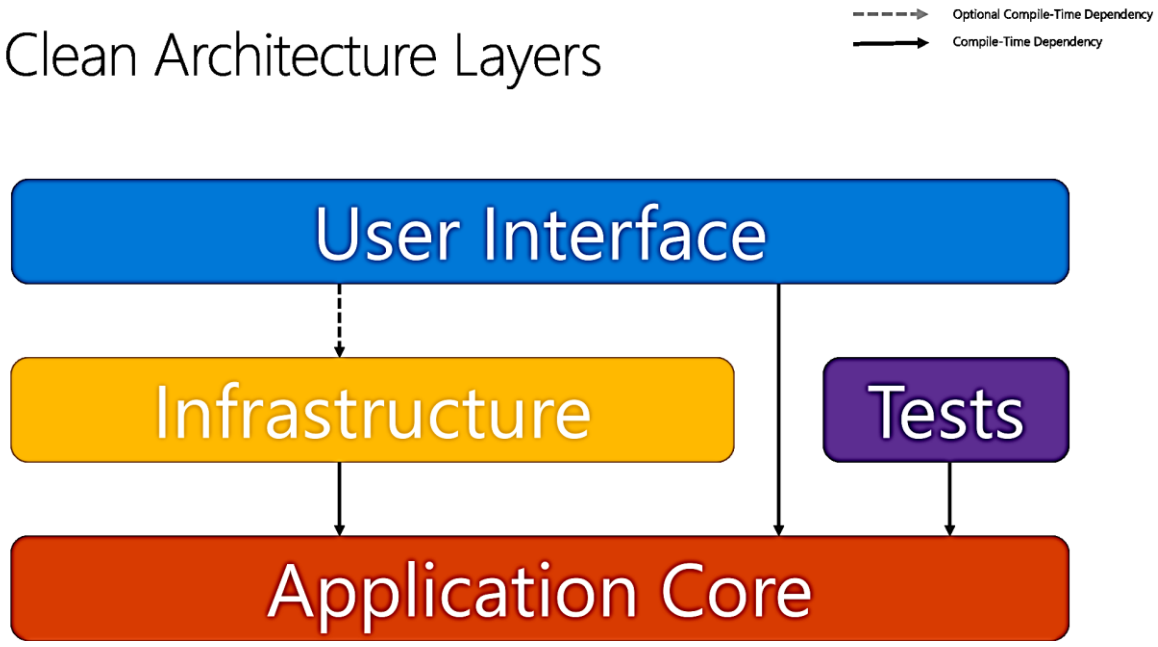
Πιο συγκεκριμένα περιγράφει μια αρχιτεκτονική που αποτελείται από στρώσεις από μέσα προς τα έξω για τα διάφορα επίπεδα που καλείται να διαχειριστεί και γι' αυτό πήρε και τον χαρακτηρισμό «Αρχιτεκτονική κρεμμυδιού». Καμία από τις εσωτερικές στρώσεις δεν ξέρει ή δεν μπορεί να αναφερθεί σε κάτι από τις εξωτερικές στρώσεις, με τις τελευταίες να αναφέρονται πιο πολύ σε μηχανισμούς και τις πρώτες σε πολιτικές.



Εικόνα 3.6: Clean architecture

Μια εφαρμογή συνήθως αποτελείται από διάφορα επίπεδα και είναι σημαντικό το καθένα από αυτά να είναι όσο το δυνατόν πιο απομονωμένο από τα υπόλοιπα, για να μη δημιουργούνται περιορισμοί στο πώς μπορούν να υλοποιηθούν ή ν' αναπτυχθούν. Με τη συγκεκριμένη αρχιτεκτονική επιτυγχάνονται τα εξής:

- Ανεξαρτητοποίηση των Frameworks που θα χρησιμοποιηθούν
- Ανεξαρτητοποίηση των διεπαφών χρηστών
- Ανεξαρτητοποίηση των βάσεων δεδομένων που χειρίζεται το σύστημα
- Ανεξαρτητοποίηση των εξωτερικών διασυνδέσεων
- Ελεγξιμότητα



Εικόνα 3.7: Επίπεδα Clean Architecture

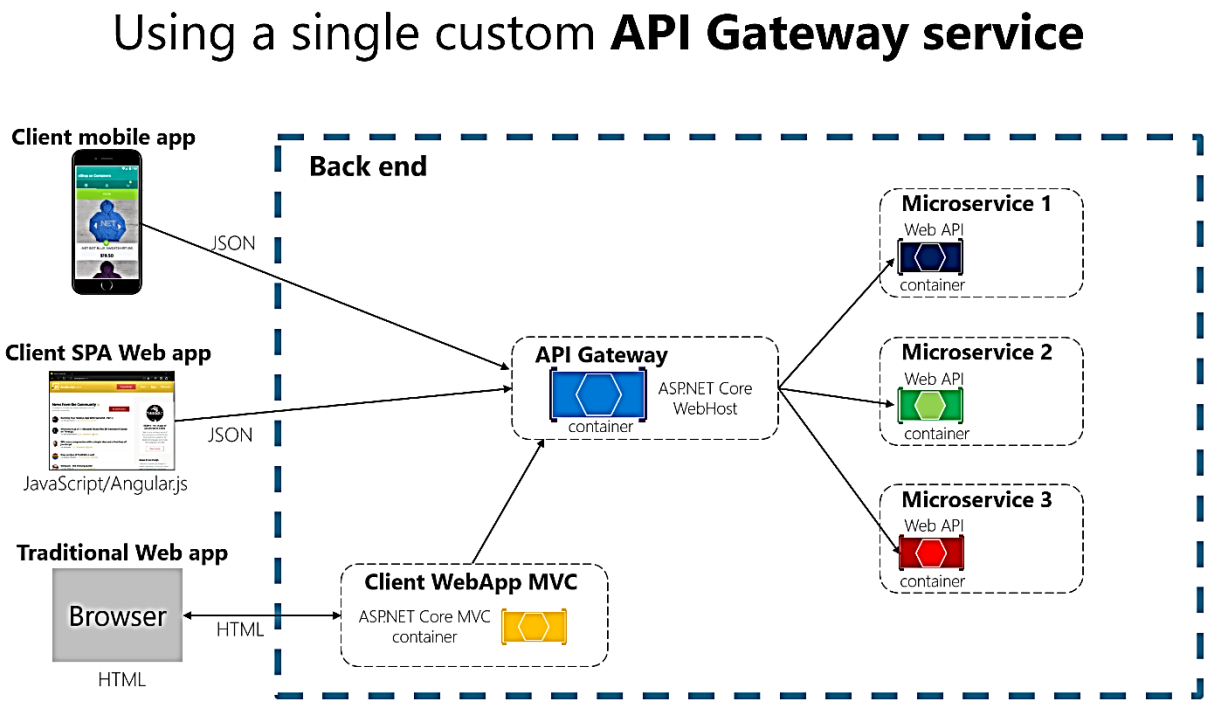
3.5.3 API Gateways

Η χρήση των API Gateways είναι πολύ συχνή στα Microservices. Λειτουργούν κυρίως ως ένας "ενδιάμεσος" ανάμεσα στον χρήστη και τις υπηρεσίες backend που τον εξυπηρετούν. Επειδή τα συστήματα Microservices χωρίζονται σε πολλές μικρό-υπηρεσίες η χρήση τους από μια εφαρμογή χρήστη (Client app) μπορεί να γίνει δύσκολη ειδικά αν υπάρχουν συνεχείς αναβαθμίσεις. Για τον λόγο αυτό χρησιμοποιείται ένα ή

πολλά API Gateways που οργανώνουν και ανακατευθύνουν τις κλήσεις του χρήστη προς τις υπηρεσίες.

Τα κύρια πλεονεκτήματα χρήσης των API Gateways είναι τα παρακάτω:

1. Οι εφαρμογές χρηστών δε χρειάζεται να γνωρίζουν πώς είναι διαμορφωμένα τα microservices (Coupling).
2. Πολλαπλές αιτήσεις (requests) μπορούν να ομαδοποιηθούν σε μια προσφέροντας ταχύτερη απάντηση προς το χρήστη.
3. Ασφάλεια ως προς το πώς είναι δομημένο το σύστημα.
4. Γενικές λειτουργίες που χρειάζεται το κάθε microservice, όπως για παράδειγμα η αυθεντικοποίηση, μπορούν να γίνουν σε προηγούμενο στάδιο και να είναι κοινές για όλες.



Εικόνα 3.8: Gateways σε συνδιασμό με microservices

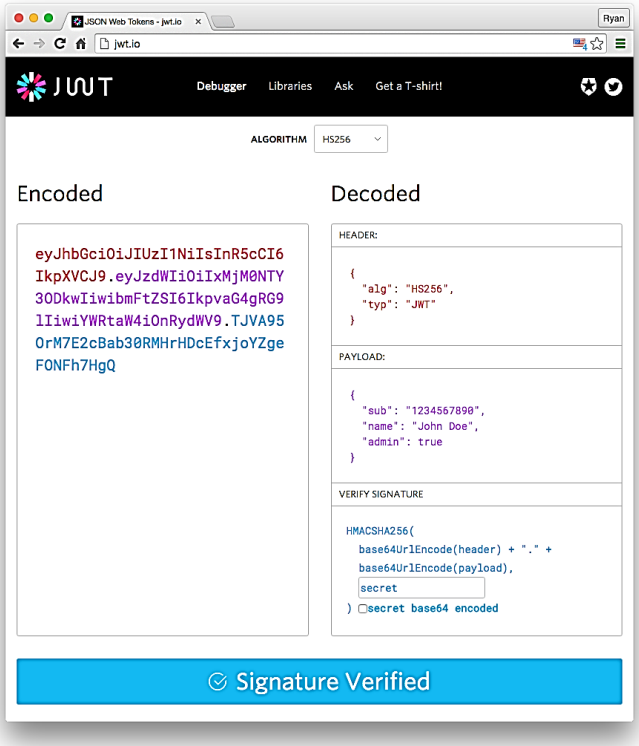
3.5.4 JWT Authentication

Μερικά από τα προβλήματα που αντιμετωπίζουν στις μέρες μας μεγάλοι εξυπηρετητές που απαντούν σε ταυτόχρονα αιτήματα πολλαπλών χρηστών είναι η καθυστέρηση και το πλήθος των πελατών που καλούνται να διαχειριστούν. Μια από τις εκδοχές αναγνώρισης

και εξυπηρέτησης ενός εγγεγραμμένου χρήστη απαιτεί μια βάση δεδομένων με τα στοιχεία του και τη δημιουργία ενός αρχείου ή εγγραφής (session), έτσι ώστε το σύστημα να τον αναγνωρίζει σε κάθε αίτημά του. Συνεπώς, οι απαιτήσεις ενός τέτοιου συστήματος αυξάνονται και η ταχύτητα απόκρισης μειώνεται ανάλογα με το πλήθος των χρηστών.

Μια λύση στο πρόβλημα αυτό ήρθε να δώσει το JWT (JSON Web Token). Ένα ανοιχτό πρότυπο για την ασφαλή μεταφορά πληροφοριών χρησιμοποιώντας JSON αντικείμενα. Χρησιμοποιείται κυρίως για την αυθεντικοποίηση κάποιου χρήστη ή την μεταφορά πληροφοριών με ασφαλή τρόπο.

Ένα JWT αποτελείται από τρία μέρη· την κεφαλίδα, το φορτίο και την υπογραφή (Header, Payload, Signature) χωρισμένα μεταξύ τους με τελεία (xxxxx.yyyyy.zzzzz). Το πρώτο μέρος αποτελείται τυπικά από δύο μέρη που καθορίζουν τον τύπο και τον αλγόριθμο που χρησιμοποιείται. Το δεύτερο μέρος περιλαμβάνει πληροφορίες που αφορούν είτε τον χρήστη είτε πρόσθετα δεδομένα. Το τελευταίο μέρος περιέχει την υπογραφή, έτσι ώστε να διασφαλιστεί ότι ο αποστολέας είναι πράγματι αυτός που το έστειλε και το μήνυμα δεν τροποποιήθηκε.



Εικόνα 3.9: Παράδειγμα JWT Token

Ο τρόπος λειτουργίας του προτύπου αυτού είναι πολύ απλός. Κατά την επιτυχή είσοδο του χρήστη στο σύστημα θα του επιστραφεί ένα JSON Web Token. Θα πρέπει να δοθεί ιδιαίτερη σημασία στη φύλαξη αυτού του Token καθώς θα χρησιμοποιηθεί για την περαιτέρω αυθεντικοποίηση του χρήστη. Η συχνή ανανέωσή του και η ασφαλής φύλαξή του θα μπορούσε να αποτρέψει προβλήματα ασφαλείας.

Στη συνέχεια, όταν ο χρήστης επιθυμεί να στείλει αίτημα (request) σε μια προστατευμένη υπηρεσία, θα πρέπει να στείλει και το συγκεκριμένο Token, συνήθως ως «Authorization header» και «Bearer» σχήμα. Ένα παράδειγμα μιας τέτοιας κεφαλίδας θα μπορούσε να είναι το παρακάτω:

```
Authorization: Bearer <token>
```

Τα οφέλη χρήσης του προτύπου αυτού είναι:

- Αποφυγή διαχείρισης sessions (stateless): Οι πληροφορίες του χρήστη καθώς επίσης και του ίδιου του Token βρίσκονται κωδικοποιημένες μέσα σε αυτό κάτι που μειώνει επιπλέον κλήσεις προς τη βάση δεδομένων για την ανάκτηση πληροφοριών του χρήστη.
- Φορητότητα: Ένα Token μπορεί να χρησιμοποιηθεί για την πρόσβαση σε πολλαπλές υπηρεσίες που έχουν γραφτεί σε διαφορετικές γλώσσες προγραμματισμού.
- Δεν απαιτεί χρήση «Cookies»: Φιλικό προς εφαρμογές για κινητές συσκευές.
- Αποκεντροποίηση (Decouple/Decentralize): Το Token μπορεί να παραχθεί οπουδήποτε ακόμα και σε έναν απομονωμένο, δικό του server, όπου δε θα επηρεάζει τη λειτουργία των υπόλοιπων υπηρεσιών.

3.6 Αναγνώριση πινακίδας οχήματος

Το κεφάλαιο αυτό αναφέρεται στον τρόπο με τον οποίο γίνεται η αναγνώριση της πινακίδας ενός οχήματος. Εξηγείται αρχικά ο όρος OCR και ποιες είναι οι κύριες χρησιμότητές του, οι κλάδοι στους οποίους αξιοποιείται αποτελεσματικά και τα ερευνητικά πεδία στα οποία αποτελεί βασικό συστατικό.

Στη συνέχεια γίνεται μια ιστορική αναδρομή στις πιο κύριες ημερομηνίες που αποτέλεσαν και σημείο εξέλιξης της τεχνολογίας αυτής. Από τις πρώτες εφαρμογές που αναπτύχθηκαν με βάση την οπτική αναγνώριση χαρακτήρων έως το «Tessaract» της γνωστής εταιρίας Hewlett-Packard.

Τέλος, παρουσιάζεται η υπηρεσία OCR.Space που χρησιμοποιήθηκε για την αναγνώριση πινακίδων στο σύστημα αυτοματοποιημένων υπηρεσιών στάθμευσης οχημάτων. Πιο συγκεκριμένα, αναλύεται το τι χρειάζεται ένα αίτημα προς τη συγκεκριμένη διαδικτυακή υπηρεσία και η απάντησή του.

3.6.1 Οπτική Αναγνώριση Χαρακτήρων

Ως Οπτική Αναγνώριση Χαρακτήρων (Optical Character Recognition - OCR) ορίζεται η ηλεκτρονική ή μηχανική μετατροπή εικόνων σε χαρακτήρες επεξεργάσιμους για τον ηλεκτρονικό υπολογιστή. Η αναγνώριση και αντιστοίχιση δηλαδή των χαρακτήρων που περιέχει μια εικόνα σε ψηφιακή μορφή, έτσι ώστε το κείμενο να μπορεί να χρησιμοποιηθεί από τον υπολογιστή.

Χρησιμοποιείται κυρίως στην ψηφιοποίηση έντυπων κειμένων, στην εξόρυξη δεδομένων και στη μετατροπή κειμένων σε ακουστικό υλικό. Με την τεχνική αυτή, πολλά βιβλία έχουν περάσει σε ψηφιακή μορφή και έγιναν διαθέσιμα σε πολλούς περισσότερους αναγνώστες. Πλέον, εκτός από εκδοτικούς οίκους, υπάρχουν ειδικά ψηφιακά καταστήματα, τα οποία προμηθεύουν τους χρήστες με συσκευές “e-books” και παρέχουν ειδικές πλατφόρμες, όπου μπορούν να προμηθευτούν και να διαβάσουν ένα βιβλίο.

Ένας ακόμη κλάδος που αξιοποιεί αρκετά την οπτική αναγνώριση χαρακτήρων είναι αυτός που ασχολείται με τη μετατροπή κειμένου σε φωνή. Με την ολοένα και μεγαλύτερη βελτίωση των αλγορίθμων αναγνώρισης είναι εφικτό τα άτομα που αντιμετωπίζουν κάποιου είδους δυσκολία με την όραση να ακούσουν ηχητικά μηνύματα με σκοπό τη διευκόλυνση της καθημερινότητά τους.

Αποτελεί βασικό πεδίο έρευνας σε τομείς όπως η αναγνώριση προτύπων, η τεχνητή νοημοσύνη και η επεξεργασία εικόνας από τον υπολογιστή. Η κατανόηση κειμένου και εικόνων από τον υπολογιστή ήταν πάντα μια δύσκολη διαδικασία που βελτιώθηκε ως προς την αποδοτικότητα και ευστοχία της με τη χρήση του OCR.

3.6.2 Ιστορική αναδρομή

Οι πρώτες εφαρμογές OCR χρονολογούνται περίπου το 1913 όπου ο Fournier D'Albe κατασκεύασε μια φορητή συσκευή, το ονομαζόμενο optophone, για την αναπαραγωγή ήχων. Η σάρωση της εικόνας γινόταν κατά μήκος της εκτυπωμένης γραμμής και δημιουργούσε ήχους, όπου ο καθένας τους αντιστοιχούσε σε ένα συγκεκριμένο χαρακτήρα. Με τη χρήση της συσκευής αυτής μπορούσαν άτομα με προβλήματα όρασης να διαβάσουν οποιοδήποτε έντυπο υλικό.

Η ανάπτυξη της τεχνολογίας για την οπτική αναγνώριση χαρακτήρων συνεχίστηκε από τον Goldberg με σκοπό την εισαγωγή δεδομένων. Η πρότασή του ήταν η φωτογράφιση δεδομένων και στη συνέχεια η χρήση φωτοκυττάρων, έτσι ώστε με τη σειρά τους να αντιστοιχηθούν σε κάποια πρότυπα αναγνώρισης.

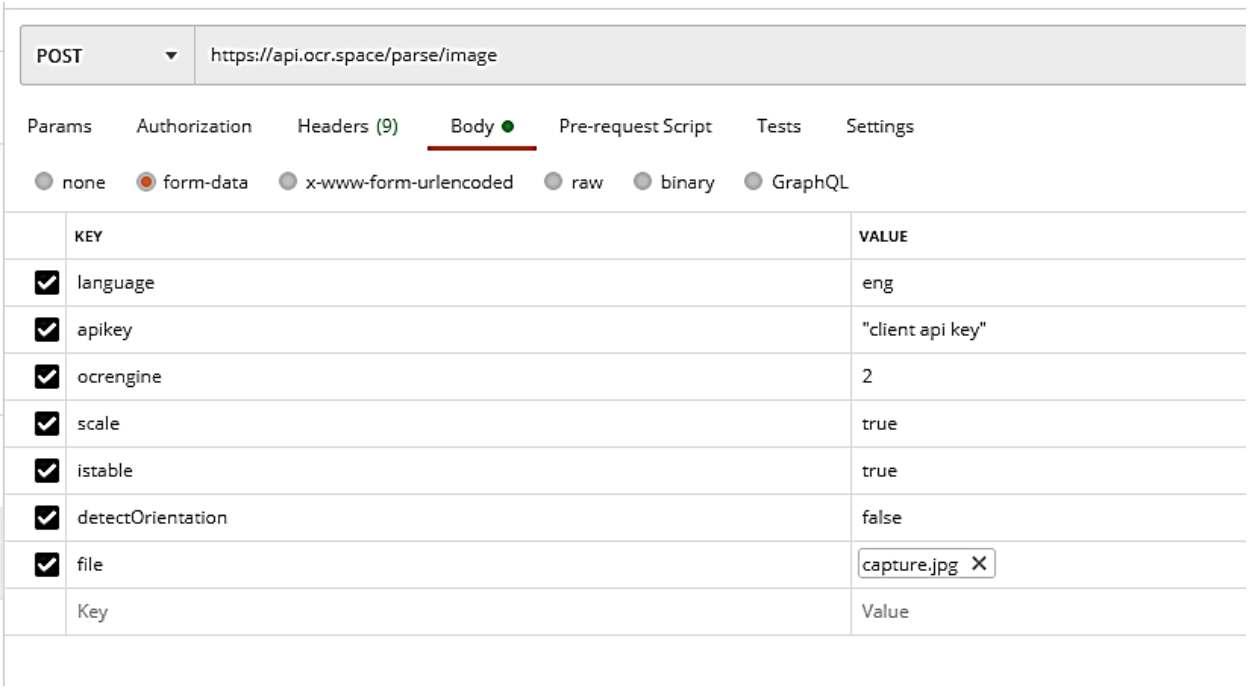
Η χρήση ηλεκτρονικού υπολογιστή συνδέθηκε με εφαρμογές οπτικής αναγνώρισης χαρακτήρων στα τέλη της δεκαετίας του 1940. Πιο συγκεκριμένα, οι ερευνητές του RCA (Radio Corporation of America) εργάστηκαν σε ένα πρωτοποριακό σύστημα αναγνώρισης χαρακτήρων με τη χρήση υπολογιστή, ο οποίος στη συνέχεια τα εκφωνούσε. Ακολούθησαν ο David H. Shepard το 1951 με την κατασκευή του «Gismo», μια μηχανή αναγνώρισης εκτυπωμένων μηνυμάτων, με σκοπό την επεξεργασία τους από υπολογιστές και ο καθηγητής John Linvill το 1962 με το Optacon, την πρώτη φορητή συσκευή ανάγνωσης για τυφλούς.

Το 2006 η Hewlett-Packard δημιουργεί το «Tessaract», ένα δωρεάν λογισμικό ανοιχτού κώδικα με την άδεια της Apache, για διάφορα λειτουργικά συστήματα και με την υποστήριξη της Google.

3.6.3 OCR.Space

Στις μέρες μας πολλές εταιρίες προσφέρουν δυνατότητες αναγνώρισης χαρακτήρων μέσω λογισμικών ως υπηρεσίες (Software as a Service - SaaS). Μια τέτοια υπηρεσία χρησιμοποιήθηκε και στην παρούσα εργασία για την αναγνώριση της πινακίδας των αυτοκινήτων. Η συγκεκριμένη υπηρεσία, μέσω μιας εύκολης διεπαφής, δίνει τη δυνατότητα στους χρήστες της να στείλουν κάποια εικόνα και με βάση τις παραμέτρους που θα επιλέξουν θα τους επιστραφεί το κείμενο που θα μπορέσει να διαβάσει ο αλγόριθμος μέσα από την εικόνα.

Για να γίνει αρχικά χρήση της υπηρεσίας θα πρέπει να δημιουργηθεί ένας λογαριασμός, έτσι ώστε να δοθεί στον χρήστη ένα API KEY. Το συγκεκριμένο κλειδί θα πρέπει να στέλνεται σε κάθε κλήση που γίνεται προς την υπηρεσία. Η επικοινωνία με την υπηρεσία είναι απλή και γρήγορη. Χρησιμοποιώντας μια φόρμα και στέλνοντάς την με τη μέθοδο POST, η υπηρεσία μάς επιστρέφει τα αποτελέσματα που βρήκε για την εικόνα. Ένα παράδειγμα κλήσης προς την υπηρεσία OCR.Space φαίνεται παρακάτω:



Εικόνα 3.10: Request προς OCR.Space

Στο παραπάνω request οι τιμές των πεδίων έχουν επιλεχθεί με βάση την αποδοτικότητά τους ως προς την αναγνώριση μιας πινακίδας αυτοκινήτου. Οι τιμές αυτές μπορεί να διαφέρουν ανάλογα με το φωτισμό, τη θέση και την ανάλυση της κάμερας και άλλους εξωτερικούς παράγοντες.

Παράμετροι κλήσης

- language: γλώσσα αναγνώρισης κειμένου μέσα στην εικόνα. Αποτελείται από τρία γράμματα και επιλέγεται η αγγλική αν δεν συμπληρωθεί το πεδίο αυτό.
- apikey: το μοναδικό κλειδί που έχει κάθε χρήστης της εφαρμογής στην κατοχή του.

- ocrengine: στη συγκεκριμένη υπηρεσία παρέχονται δύο μηχανές οπτικής αναγνώρισης χαρακτήρων. Η πρώτη (“ocrengine = 1”) υποστηρίζει πολλαπλές γλώσσες, είναι ταχύτερη, δέχεται μεγαλύτερες εικόνες και υποστηρίζει πολλαπλές σελίδες. Η δεύτερη υποστηρίζει μόνο δυτικές γλώσσες με λατινικούς χαρακτήρες όπως Αγγλικά, Γερμανικά, Γαλλικά κ.ά., έχει μεγαλύτερη ευστοχία στην εύρεση ειδικών χαρακτήρων (@,#,-,+,…) και στις ανάποδες εικόνες, αλλά δέχεται εικόνες με ανώτατο μέγεθος 5000*5000px
- scale: με την παράμετρο αυτή δίνεται η δυνατότητα στον μηχανισμού του OCR.Space να κάνει εσωτερικές αυξομειώσεις στην εικόνα για να βελτιώσει το παραγόμενο αποτέλεσμα. Σε εικόνες με μικρή ανάλυση είναι σχεδόν απαραίτητο.
- istable: σε περίπτωση που το κείμενο είναι χωρισμένο σε γραμμές, όπως για παράδειγμα τιμολόγια, η παράμετρος αυτή θα πρέπει να τεθεί «true» ώστε το επιστρεφόμενο αποτέλεσμα να είναι χωρισμένο σε γραμμές.
- detectOrientation: δίνει τη δυνατότητα στο σύστημα να επιλέξει μόνο του το πόσο, που θα περιστρέψει την εικόνα για να έχει καλύτερο αποτέλεσμα.

Απάντηση

Με κάθε αίτημα επιστρέφεται και μια απάντηση ανάλογα με τις παραμέτρους που έχουμε επιλέξει. Η απάντηση αυτή μπορεί να είναι είτε κάποιο σφάλμα στο ερώτημα που έχει σταλεί είτε το αποτέλεσμα της διαδικασίας αναγνώρισης κειμένου μέσα από την εικόνα. Το αποτέλεσμα θα μπορούσε να είναι και κενό κείμενο σε περίπτωση που δεν αναγνωριστεί κανένας χαρακτήρας.

Παρακάτω φαίνεται η απάντηση (“response”) που επιστρέφει το ερώτημα προς την υπηρεσία της OCR.Space με τα χαρακτηριστικά που παρουσιάστηκαν στην προηγούμενη ενότητα. Πιο αναλυτικά, μας επιστρέφει ένα JSON με όλο το εντοπισμένο κείμενο (“ParsedText”), τις γραμμές του κειμένου (“LineText”) καθώς επίσης και τις λέξεις (“WordText”) μια προς μια που εντοπίστηκαν μέσα στην εικόνα.

```
"ParsedResults": [
    {
        "TextOverlay": {
            "Lines": [
                {
                    "LineText": "NKI 3564",
                    "Words": [
                        {
                            "WordText": "NKI",
                            "Left": 360.0,
                            "Top": 288.33331298828127,
                            "Height": 54.999996185302737,
                            "Width": 101.66666412353516
                        },
                        {
                            "WordText": "3564",
                            "Left": 481.6666564941406,
                            "Top": 288.33331298828127,
                            "Height": 54.999996185302737,
                            "Width": 102.49999237060547
                        }
                    ],
                    "MaxHeight": 54.999996185302737,
                    "MinTop": 288.33331298828127
                }
            ],
            "HasOverlay": true,
            "Message": "Total lines: 1"
        },
        "TextOrientation": "0",
        "FileParseExitCode": 1,
        "ParsedText": "NKI 3564\t\r\n",
        "ErrorMessage": "",
        "ErrorDetails": ""
    }
],
"OCRExitCode": 1,
"IsErroredOnProcessing": false,
"ProcessingTimeInMilliseconds": "671"
```


Εικόνα 3.11: Response απο OCR.Space

Η απάντηση περιέχει επίσης πληροφορίες σχετικά με τη θέση που βρέθηκε κάθε λέξη μέσα στην εικόνα, αν υπήρξε κάποιο πρόβλημα κατά τη διάρκεια ανίχνευσης κειμένου, αν μπόρεσε να επεξεργαστεί όλες τις εικόνες ή μερικές από αυτές κ.ά.

3.7 Ανάλυση κώδικα

Προκειμένου να υλοποιηθεί ένα σύστημα αυτοματοποιημένων υπηρεσιών στάθμευσης οχημάτων, θα πρέπει να συνεργαστούν διάφορες τεχνολογίες σε πολλαπλά επίπεδα όπως βάσεις δεδομένων, πύλες, διαδικτυακές υπηρεσίες και εφαρμογές χρήστη.

Στα παρακάτω υποκεφάλαια θα γίνει μια εκτενής ανάλυση των σημαντικότερων σημείων του κώδικα που διαχειρίζεται το σύστημα αυτοματοποιημένων υπηρεσιών.

3.7.1 Διαχείριση κλήσεων (Request handling)

Η διαχείριση των κλήσεων περιλαμβάνει τις πύλες (gateways), του διαχειριστές (Controllers) και τις λειτουργίες (Actions) που χρειάζονται προκειμένου τα αιτήματα να εξυπηρετηθούν από κάποια υπηρεσία. Παρακάτω αναλύονται τα τρία αυτά κύρια συστατικά που χρησιμοποιήθηκαν στο σύστημα αυτοματοποιημένων υπηρεσιών στάθμευσης οχημάτων.

Ocelot

Οι διαδικτυακές πύλες όπως αναφέραμε και στο κεφάλαιο των “API Gateways” δημιουργούν ένα πρόσθετο βήμα σε μια κλήση καθώς θα πρέπει να περάσει από έναν ακόμη κόμβο, αλλά προσθέτει σημαντικά οφέλη στο σύνολο του συστήματος. Μερικά από αυτά είναι η πιο κεντρική διαχείριση των κλήσεων προς τις υπηρεσίες του συστήματος, η απόκρυψη της δομής των εξυπηρετητών κ.ά.

Στο εν λόγω σύστημα χρησιμοποιήθηκε το middleware Ocelot της Microsoft. Το πακέτο αυτό είναι ανοιχτού κώδικα σχεδιασμένο για αρχιτεκτονικές βασισμένες στα microservices που χρειάζονται ένα ενοποιημένο σημείο εισόδου. Μπορεί να χρησιμοποιηθεί μόνο από εφαρμογές που είναι γραμμένες σε .NET Core και προσφέρει ένα ελαφρύ λογισμικό για δρομολόγηση και έλεγχο ταυτότητας κ.ά.

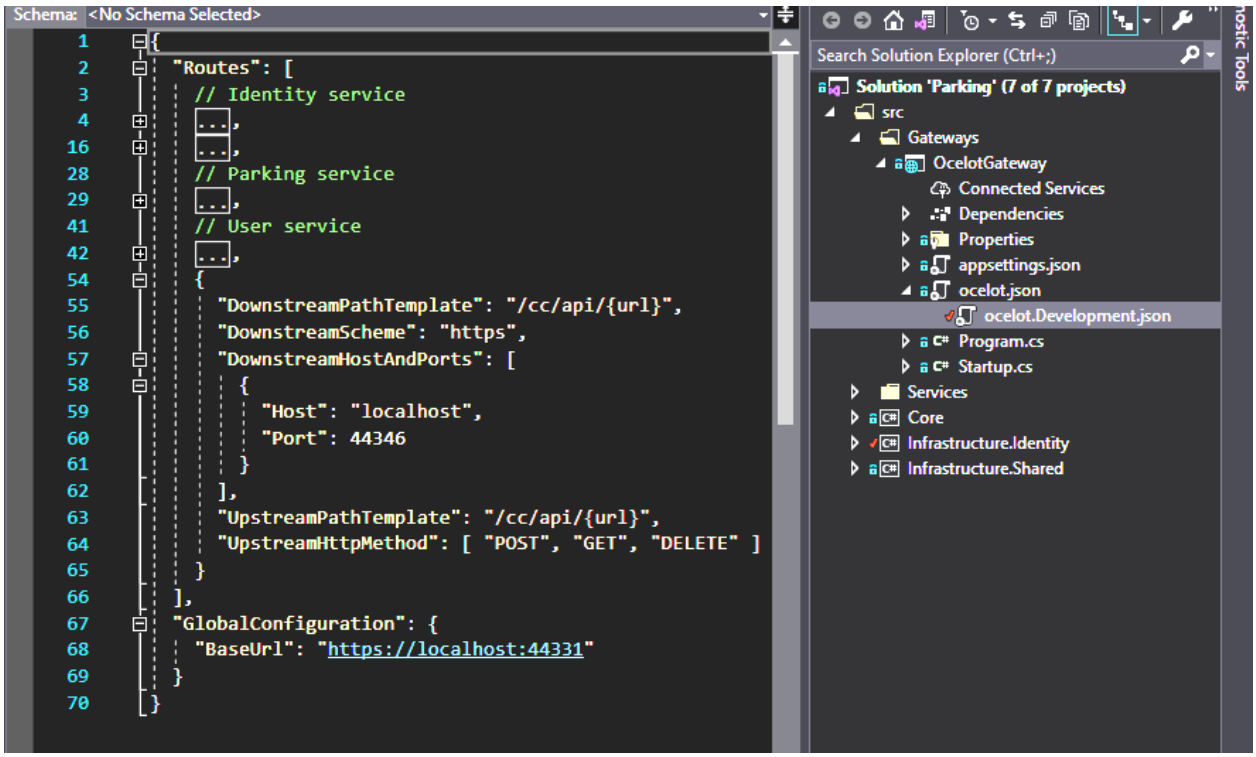
Οι κύριοι λόγοι επιλογής του πακέτου Ocelot ως δρομολογητή ήταν το μέγεθος του, η ταχύτητα δρομολόγησης των αιτημάτων προς τα web-services και η cross-platform δυνατότητα χρήσης του καθώς είναι ένα πακέτο βασισμένο στο .NET Core.

Για να γίνει χρήση του συγκεκριμένου middleware θα χρειαστεί να δημιουργηθεί μια εφαρμογή .NET Core App, ιδανικά σε έκδοση 3.1 και να γίνει εγκατάσταση του πακέτου. Έπειτα, σειρά έχει ο σχηματισμός των κανόνων που δηλώνονται στο αρχείο ocelot.json όπως φαίνεται και στην εικόνα παρακάτω.

Μέσα στο αρχείο ocelot.json θα πρέπει να δηλωθούν αρχικά η διεύθυνση στην οποία ακούει το συγκεκριμένο service κι έπειτα τα επιμέρους routes. Κάθε route αποτελείται από:

- τη διεύθυνση, στην οποία λαμβάνει αιτήματα (UpstreamPathTemplate).
- τις Http μεθόδους που επιτρέπει στο εισερχόμενο αίτημα (UpstreamHttpMethod).

- τον συνδυασμό των DownstreamScheme, DownstreamHostAndPorts, DownstreamPathTemplate για τη δημιουργία της διεύθυνσης ανακατεύθυνσης του αιτήματος.



Εικόνα 3.12: Configuration για δρομολόγηση μέσω Ocelot

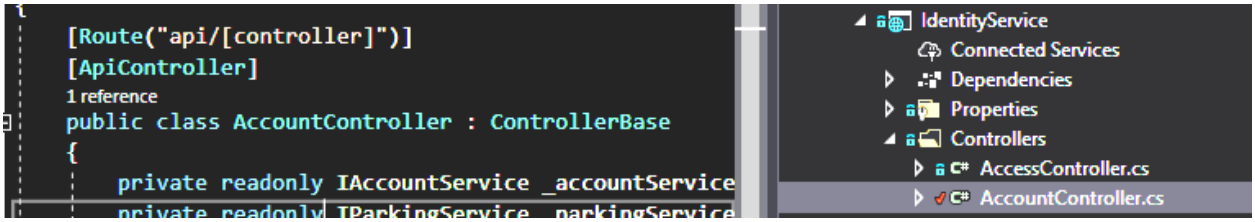
Αφότου δημιουργηθούν οι καταχωρήσεις σχετικά με τις διευθύνσεις και τους εξυπηρετητές, σειρά έχει η δήλωση χρήσης του πακέτου μέσα στις υπηρεσίες που θα χρησιμοποιεί το συγκεκριμένο service, η οποία γίνεται χρησιμοποιώντας τους μηχανισμούς Dependency Injection της πλατφόρμας .Net Core.

Controllers

Για τη διαχείριση των Http αιτημάτων, το .Net Core framework χρησιμοποιεί τους λεγόμενους Controllers. Το όνομα του αρχείου που τους αντιπροσωπεύει τελειώνει με τη λέξη “Controller” και το υπόλοιπο χρησιμοποιείται συχνά ως μέρος της διεύθυνσης που εξυπηρετεί. Βρίσκονται ως συνήθως μέσα στον φάκελο “ Controllers”.

Ένα από τα βασικά του χαρακτηριστικά ενός Controller είναι η δήλωση της διεύθυνσης την οποία καλείται να διαχειριστεί. Η διεύθυνση αυτή είναι μοναδική για κάθε εφαρμογή και δηλώνεται με το χαρακτηριστικό [Route(“/route”)].

Ένα παράδειγμα τέτοιου Controller φαίνεται στην παρακάτω εικόνα. Ο συγκεκριμένος Controller διαχειρίζεται όλα εκείνα τα αιτήματα που έχουν διεύθυνση “…/api/account/….”. Οι αγκύλες με τη λέξη “controller” μέσα στη δήλωση του Route, αντικαθίστανται από το όνομα του αρχείου.



Εικόνα 3.13: Ορισμός διεύθυνσης Controller

Με τον συγκεκριμένο τρόπο όλες οι λειτουργίες που αφορούν μια ενότητα, όπως για παράδειγμα ο χρήστης, η διαχείριση του προφίλ, η δημιουργία αναφορών κ.ά. μπορούν να ομαδοποιηθούν κάτω από ένα αρχείο. Αυτό διευκολύνει τη διαχείριση της εφαρμογής, την ανάγνωση του κώδικα και την εύρεση πιθανών προβλημάτων.

Ένα ακόμα βασικό συστατικό των Controllers είναι η δήλωση «[ApiController]» πάνω από την κλάση. Το χαρακτηριστικό αυτό ενεργοποιεί λειτουργίες που έχουν ως βάση τα API συστήματα. Μερικές από αυτές είναι:

- η υποχρεωτική δήλωση μονοπατιού
- οι αυτόματες απαντήσεις σε ερωτήματα με κωδικό 400
- ευκολότερη σύνδεση παραμέτρων

Actions

Τα actions είναι μέθοδοι των Controllers που εκτελούν διεργασίες και επιστρέφουν πίσω αποτελέσματα. Καταχωρούνται στο σύστημα κατά την εκκίνησή του και εξυπηρετούν μια και μοναδική διεύθυνση. Σε συνδυασμό με τους Controllers σχηματίζουν τη διευθυνσιοδότηση του συστήματος και των URLs που θα εξυπηρετεί.

Μιας action μέθοδος επιστρέφει ως συνήθως ένα αντικείμενο τύπου IActionResult. Το αντικείμενο αυτό είναι από τα πιο κατάλληλα, όταν πρόκειται να επιστραφούν περισσότεροι από έναν τύποι ActionResult. Τα ActionResult αντιπροσωπεύουν διάφορους τύπους κατάστασης HTTP αιτημάτων. Κάποιοι από αυτούς τους τύπους είναι οι BadRequestResult (400), OkObjectResult (200) και πολλοί άλλοι.

3.7.2 Αυθεντικοποίηση

Για την αυθεντικοποίηση του χρήστη στο σύστημα χρησιμοποιήθηκε το πακέτο Identity της Microsoft σε συνδυασμό με τα JWT Tokens. Η χρήση των δυο αυτών πακέτων διευκολύνει την αναγνώριση και διαχείριση των χρηστών σε σελίδες SPA (single page applications).

Οι σελίδες SPA αποφεύγουν τη χρήση της επαναφόρτωσης για τη λήψη νέων δεδομένων και εκτελούν ασύγχρονες κλήσεις (AJAX) προς τους εξυπηρετητές τους. Επίσης υπάρχει μεγάλο ενδεχόμενο οι εφαρμογές αυτές να μην βρίσκονται στον ίδιο server με τις υπηρεσίες που εκτελούν τα ερωτήματα ή να υπάρχουν πολλαπλοί servers που να εξυπηρετούν την ίδια εφαρμογή πελάτη.

Για τους παραπάνω λόγους, είναι επιτακτική η ανάγκη για μια ανεξάρτητη υπηρεσία αυθεντικοποίησης του χρήστη που θα του δίνει πρόσβαση στις υπόλοιπες υπηρεσίες του συστήματος. Μια τέτοια υπηρεσία συνήθως λειτουργεί με Tokens τα οποία δίνει στον χρήστη κατά την είσοδό του στο σύστημα. Από τη στιγμή που ο χρήστης θα λάβει το Token καθίσταται υπεύθυνος για τη φύλαξη και προστασία του από κακόβουλα λογισμικά ή χρήστες.

Identity

Το πακέτο αυτό προσφέρει τα μοντέλα και τις βασικές λειτουργίες που μπορεί να χρειαστεί ένα σύστημα διαχείρισης χρηστών όπως η εγγραφή, η είσοδος ενός χρήστη, η ταυτοποίησή του κ.ά. Μπορεί να χρησιμοποιηθεί σε συνδυασμό με μια βάση δεδομένων ή με τη μνήμη του συστήματος για σκοπούς ελέγχου.

Τα βασικά του μοντέλα είναι:

- IdentityUser
- IdentityRole

- IdentityUserRole
- IdentityUserClaim
- IdentityUserLogin
- IdentityRoleClaim
- IdentityUserToken

Παρότι έχουν συγκεκριμένη δομή είναι πάρα πολύ εύκολο να παραμετροποιηθούν στις ανάγκες του εκάστοτε συστήματος. Το μόνο που χρειάζεται είναι να δημιουργηθεί μια νέα κλάση που θα κληρονομεί το βασικό μοντέλο προς αλλαγή και εκεί μέσα μπορούν να του εκχωρηθούν νέα χαρακτηριστικά ή και μέθοδοι.

Μια ακόμα λειτουργία του πακέτου αυτού είναι η παροχή κωδικοποίησης σε ευαίσθητα δεδομένα όπως email, password κ.ά. Διασφαλίζει με τον τρόπο αυτό ότι ακόμα και αν κάποιος αποκτήσει πρόσβαση στη βάση δεδομένων χωρίς άδεια, δεν θα μπορεί να διαβάσει δεδομένα χρηστών. Η κωδικοποίηση της έκδοσης Core είναι τύπου HMAC-SHA256.

Δημιουργία Tokens

Για να μπορέσει να επικοινωνήσει ο χρήστης με τις υπηρεσίες του συστήματος, θα πρέπει να έχει στην κατοχή του ένα Token αυθεντικοποίησης. Για τη δημιουργία του χρησιμοποιείται το πρότυπο JWT. Αυτό παρέχεται στον χρήστη κατά την επιτυχή είσοδό του στο σύστημα και έχει περιορισμένο χρόνο χρήσης.

Με το Token αυτό αναγνωρίζει το σύστημα ποιος είναι ο χρήστης που εκτελεί το ερώτημα και μπορεί να ανασύρει πληροφορίες γι’ αυτόν. Μπορεί επίσης να απαγορεύσει τη χρήση συγκεκριμένων λειτουργιών.

Μαζί με τον JWT Token, παρέχεται ένα ακόμα Token που χρησιμοποιείται για την ανανέωση του πρώτου σε περίπτωση που επέλθει η περίοδος χρήσης του. Το Token ανανέωσης αποθηκεύεται στη βάση δεδομένων και έχει μεγαλύτερη διάρκεια χρήσης. Σε περίπτωση που ο χρήστης χάσει αυτό το αναγνωριστικό, το σύστημα δε δύναται να τον αναγνωρίσει και θα πρέπει να γίνει είσοδος από την αρχή.

Η επικοινωνία του Token ανανέωσης με τον χρήστη γίνεται μέσω Cookies. Ο χρήστης, μετά την είσοδό του στο σύστημα λαμβάνει μαζί με την απάντηση που περιέχει το JWT Token και ένα Cookie με συγκεκριμένο όνομα που του έχει δοθεί από το σύστημα. Το

Cookie αυτό δεν έχει χαρακτηριστεί ως HttpOnly, ώστε οι εφαρμογές διαδικτύου να μπορούν να διαβάσουν την τιμή του.

Την τιμή του παραπάνω Cookie θα πρέπει η εφαρμογή να την κρατήσει και να την προσθέτει στους header κάθε αιτήματος που κάνει. Με τον τρόπο αυτό μειώνονται οι πιθανότητες να υποκλέψει κάποιος την τιμή του Token αυθεντικοποίησης και να κάνει κλήσεις με ψεύτικα στοιχεία. Διασφαλίζεται, δηλαδή, ότι ο χρήστης που εκτελεί το ερώτημα είναι και αυτός που έκανε την είσοδο στο σύστημα.

Στις δυο παρακάτω εικόνες φαίνεται η απάντηση που μας επιστρέφει μια επιτυχής είσοδος ενός χρήστη στο σύστημα. Στο σώμα της απάντησης υπάρχει ένα JSON αντικείμενο με το JWT Token του χρήστη και παράλληλα έχει επιστραφεί κι ένα Cookie με όνομα «XSRF-TOKEN», που περιέχει την τιμή του Token ανανέωσης.


```
{
    "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJzdWIiOiJiYXNpY1VzZXIiLCJqdGkiOiJhYTZlYzA4Zi02YTdlLTQ1YTI
        tOTZkMy0yNTg2MDkyY2IyNWYiLCJlbWFpbCI6ImJhc2ljdXNlck8nbWFpbC
        5jb20iLCJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L
        2lkZW50aXR5L2NsYWltcy9uYW1laWRlbnRpZmllciI6IjIiLCJ1aWQiOiIy
        IiwiaXAiOiIxOTIuMTY4LjEuMiIsInJvbGVzIjoiQmFzaWMiLCJleHAiOjE
        2MjMyNjAyMTIsImlzcyI6IkNvcmVJZGVudGl0eSIsImF1ZCI6IkNvcmVJZG
        VudGl0eVVzZXIifQ.
        b-L5yr_d2yykPnz4JD_-yK1cS8pPNZW9IzhISRcuvP8",
    "expires": "2021-06-09T17:36:52Z"
}
```


Εικόνα 3.14: JSON Response από επιτυχημένη ταυτοποίηση


```
▼ Response Headers
  Cache-Control: "no-cache"
  Pragma: "no-cache"
  Content-Length: "502"
  Content-Type: "application/json; charset=utf-8"
  Expires: "Thu, 01 Jan 1970 00:00:00 GMT"
  Server: "Microsoft-IIS/10.0"
  Set-Cookie: ".AspNetCore.Identity.Application=CfDJ8NRbd8ZyTn5CgAKn4ENwonwUwJ9IStgYz8bxxKzqxRptWYgwfY6l7D3WwPR5bwpA_
  Qcu-V0zeNicFtdzgDpdzaRacg9Y5Jn-OTEojDX_05R7ZJZ5QA24TIMhrrsByTQq8MFb8zEFuQGNQ6oegRGa9iirh677IKnIkFbluOW1tikO_MAUi5qe
  LYP-jA2sKWaQBtScitdEkGIDwN1PqXdXYPXUkX1D_ygBBpg5TPMOajErDdzf_9drNJNVOhEe6ll7R6zFldnLU-6UuB3F_XcFkpoXbdvbNA3Ei92lsuO
  M6sUNANbg3pCaGQtpOh3uprf2I_R6Oq2kAxuhQ00CAuE05Cedw4BT7Oa6bKYm1K7ZYSCMQavZ6IfZlRjhTaU8TASCRzH2UN7YpYiOJR9MG3bbKvbxW3
  ZDF8H8e-OcWR4Hm-y_GSCpzu19D9n59N9wUnVFqXtfTWTGZsStNCQRg2ewS5-00mOl4n1FOfxzToR2ATLQFGBUsCCsAgt4da456r-6cO_vo3RrSt9b3
  m82H97rJT7R7Cf0B-C8z5Iwnuluiqho-5z7ueoLSdttVdEXBn86FmigQmmAvka0yIgWb4EjOX-T1o7-6DBBBtZykqNrrqOkJ3WpvpgH5ANjPM1sQV-8
  I2FhVqSLeSuKbUgNPSwhc-8h67wM5zk355c2RFVElSwj; path=/; secure; samesite=lax; httponly"
  Set-Cookie: "XSRF-TOKEN=2978744F58FA53C4FFBCC5A1678AFCF1FE2E49B29E8D5217E2692A8CA99540A492EBFAE53F4FC75C576F048C724
  7CBCBE8CB43528E4EFB292409E34FF712ED9D; expires=Wed, 16 Jun 2021 17:31:52 GMT; path=/; samesite=strict"
  X-Powered-By: "ASP.NET"
  X-Powered-By: "ASP.NET"
  Date: "Wed, 09 Jun 2021 17:31:52 GMT"
▸ Response Body
```


Εικόνα 3.15: XSRF-TOKEN Cookie

3.7.3 Διαχείριση σφαλμάτων (Error handling)

Ένα σημαντικό σημείο για κάθε εφαρμογή είναι η διαχείριση των σφαλμάτων της. Πολλές είναι οι φορές που κάτι μπορεί να πάει στραβά κατά την εκτέλεση του κώδικα και αυτό θα πρέπει να αντιμετωπιστεί χωρίς να σταματήσει η λειτουργία της εφαρμογής. Υπάρχει έτσι η ανάγκη για ένα κεντρικό σύστημα πρόληψης και διαχείρισης σφαλμάτων πριν αυτά εμφανιστούν στον τελικό χρήστη.

Ένα ακόμα πρόβλημα μη σωστής διαχείρισης σφαλμάτων θα μπορούσε να δημιουργήσει κενά ασφαλείας στο σύστημα. Αυτό μπορεί να συμβεί, αν κάποιος αξιοποιήσει τις πληροφορίες που βρίσκονται στα περιεχόμενα του σφάλματος.

Η διαχείριση αυτή των σφαλμάτων στις υπηρεσίες διαδικτύου γίνεται συνήθως με κάποιο ενδιάμεσο «middleware». Ο σκοπός του είναι να επιβλέπει την εκτέλεση του κάθε αιτήματος και μόλις δημιουργηθεί κάποιο σφάλμα να αναλάβει να το μεταφράσει σε ένα μήνυμα πιο φιλικό προς τον χρήστη.

Στην παρακάτω εικόνα φαίνεται ο κώδικας που διαχειρίζεται τα σφάλματα των υπηρεσιών διαδικτύου της εφαρμογής. Πιο συγκεκριμένα, περιμένει μέχρι να τελειώσει η εκτέλεση των διεργασιών του εκάστοτε αιτήματος και αν εντοπιστεί κάποιο σφάλμα, τότε δημιουργεί ένα νέο αντικείμενο με τις κατάλληλες πληροφορίες.

```
0 references
public async Task Invoke(HttpContext context)
{
    try
    {
        await _next(context);
    }
    catch (Exception error)
    {
        var response = context.Response;
        response.ContentType = "application/json";
        var responseModel = new Response<string> { Succeeded = false, Message = error.Message };

        switch (error)...
        var result = JsonSerializer.Serialize(responseModel);

        await response.WriteAsync(result);
    }
}
```

Εικόνα 3.16: Σύλληψη και επεξεργασία σφαλμάτων

Ένα από τα θετικά της τεχνικής αυτής είναι, ότι αν παραμετροποιηθεί σωστά δίνει μεγάλη ελευθερία στην οργάνωση της δομής του προγράμματος. Αυτό γιατί από οποιοδήποτε σημείο και αν εμφανιστεί κάποιο σφάλμα θα μεταφραστεί κατάλληλα και θα

παρέχει τις απαραίτητες πληροφορίες στον χρήστη. Μπορεί επίσης να συνδυαστεί και με ένα παράλληλο σύστημα καταγραφής ή ενημέρωσης σφαλμάτων για να βελτιώσει την ταχύτητα εντοπισμού τους.

3.7.4 Βάση δεδομένων (Database)

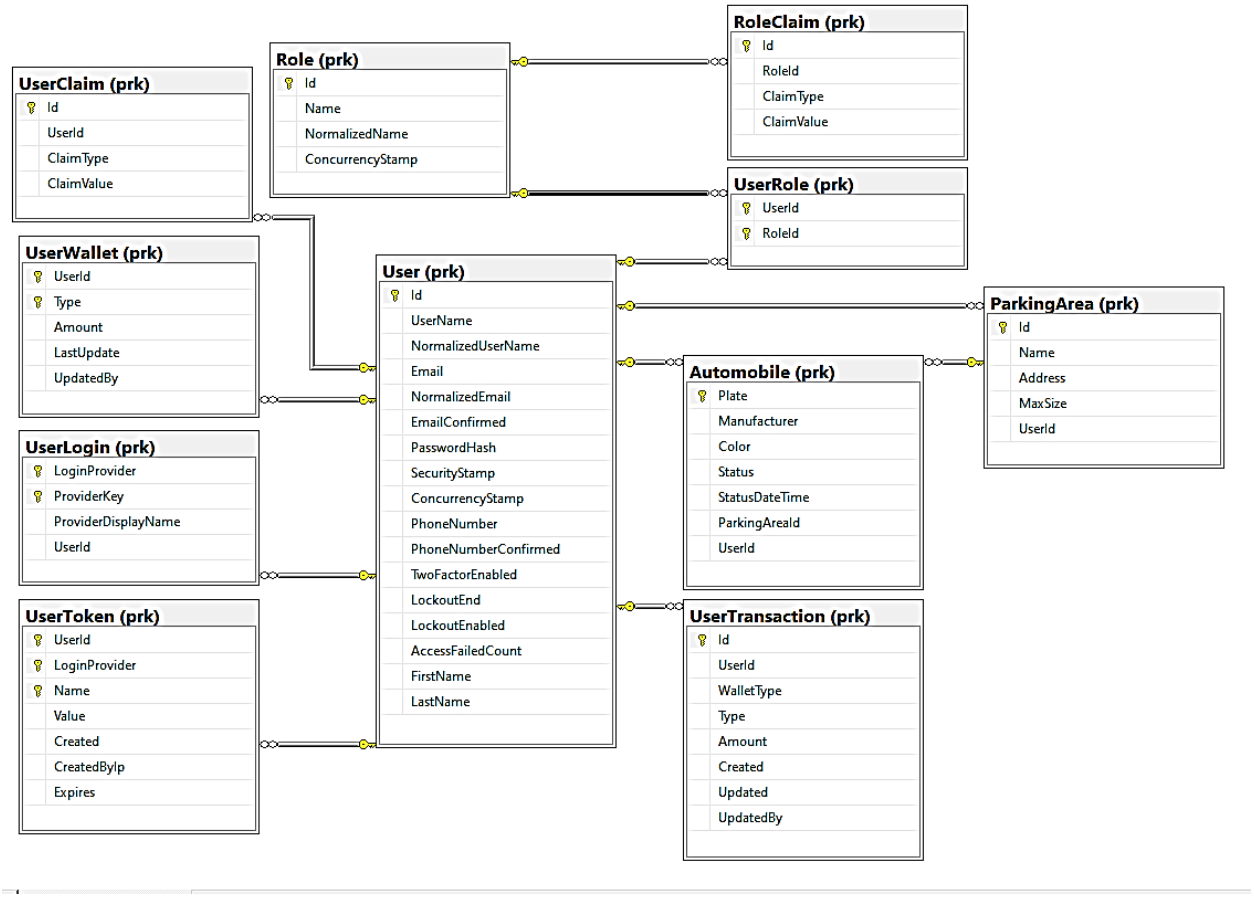
Στη βάση δεδομένων αποθηκεύονται όλες οι πληροφορίες σχετικά με τους χρήστες, τους χώρους στάθμευσης, τα αυτοκίνητα και τις κινήσεις τους. Η ανάκτηση αυτών γίνεται μέσω ερωτημάτων SQL. Για τη δημιουργία των ερωτημάτων γίνεται χρήση του πακέτου «Entity Framework» και της γλώσσας LINQ (Language Integrated Query).

Η γλώσσα LINQ χρησιμοποιεί εκφράσεις lambda για να αναπαραστήσει ένα ερώτημα προς τη βάση. Οι εκφράσεις αυτές είναι γνωστές και από άλλες γλώσσες όπως Javascript, Java κ.ά. Είναι σύντομα μπλοκ κώδικα που λαμβάνουν παραμέτρους και επιστρέφουν ανάλογα αποτελέσματα. Θα μπορούσαν να χαρακτηριστούν και μέθοδοι με τη μόνη διαφορά ότι δεν χρειάζονται όνομα (Anonymous Functions).

Το πακέτο «Entity Framework» είναι μια βιβλιοθήκη που προσφέρει εργαλεία διαχείρισης βάσης δεδομένων. Με την εγκατάσταση του πακέτου αυτού γίνονται διαθέσιμα τα βασικά αντικείμενα και μέθοδοι δημιουργίας μιας βάσης δεδομένων και σχηματισμού ερωτημάτων προς αυτή. Για την αναπαράσταση των πινάκων χρησιμοποιεί κλάσεις αντικειμένων, τα λεγόμενα μοντέλα.

Η δημιουργία των μοντέλων μιας βάσης δεδομένων μπορεί να γίνει με δυο βασικούς τρόπους τους Database First και Code First. Ο πρώτος προϋποθέτει να υπάρχει ήδη έτοιμη η βάση δεδομένων και μέσω των εργαλείων του «Entity Framework» να δημιουργηθούν τα κατάλληλα αντικείμενα στο πρόγραμμα. Ο δεύτερος, απαιτεί πρώτα να δημιουργηθούν τα μοντέλα κι έπειτα να γίνει ανανέωση της βάσης με αυτά.

Στην εφαρμογή χρησιμοποιήθηκε η μέθοδος Code First για τη δημιουργία της βάσης δεδομένων καθώς ενδείκνυται για μικρές εφαρμογές και με το «migration» εργαλείο μπορεί να διατηρηθεί έλεγχος ως προς τις εκδόσεις της. Στην παρακάτω εικόνα φαίνεται το διάγραμμα της βάσης και οι πίνακες – μοντέλα που την αποτελούν.



Εικόνα 3.17: Σχεδιάγραμμα βάσης δεδομένων

Το παραπάνω διάγραμμα ακολουθεί τη μεθοδολογία των microservices, ώστε να μπορεί να χωριστεί σε περισσότερες ενότητες, αν αυτό χρειαστεί. Πιο συγκεκριμένα, έχει τη δυνατότητα να χωριστεί σε δύο ενότητες, μια που έχει ως επίκεντρο τον χρήστη και μια που αφορά τους χώρους στάθμευσης. Με τη μέθοδο αυτή, μπορούν ακόμα και να χωριστούν οι βάσεις για να μειώσουν τον φόρτο εργασίας που έχουν να διαχειριστούν και να βελτιώσουν τον χρόνο απόκρισης.

3.7.5 Χρονοχρέωση

Για τη χρέωση των πελατών ακολουθήθηκε η λογική του ηλεκτρονικού πορτοφολιού. Το πορτοφόλι αυτό δημιουργείται κατά την εγγραφή του χρήστη στο σύστημα. Για να γίνει χρήση της υπηρεσίας από κάποιο όχημα θα πρέπει πρώτα να υπάρχουν χρήματα στο πορτοφόλι του χρήστη στο οποίο αυτό είναι καταχωρημένο.

Κατά τη διαδικασία αναγνώρισης της πινακίδας γίνεται έλεγχος του πορτοφολιού του χρήστη για να διαπιστωθεί αν μπορεί να χρησιμοποιήσει την υπηρεσία. Αν ο χρήστης του οχήματος έχει λιγότερο από ένα ευρώ στο λογαριασμό του, τότε επιστρέφεται ένα μήνυμα λάθους που πληροφορεί τα υπόλοιπα συστήματα να μην εκτελέσουν κάποια ενέργεια. Επίσης, όταν ένα όχημα εξέρχεται από την περιοχή στάθμευσης, γίνεται η χρέωση του εκάστοτε χρήστη. Για λόγους ευκολίας έχουν γίνει οι εξής παραδοχές: μια ώρα ισούται με ένα ευρώ και η ελάχιστη χρέωση χρήστη είναι η μια ώρα.

3.7.6 Διεπαφές προγραμματισμού χρήστη (APIs)

Για τις ανάγκες της εφαρμογής και για την κάλυψη ενδεχόμενων επεκτάσεών της, χρειάστηκε να δημιουργηθούν τρία APIs. Ένα για τη διαχείριση της εγγραφής και της αυθεντικοποίησης του χρήστη, ένα για τις εργασίες σχετικά με τον χρήστη, όπως η αλλαγή των στοιχείων του και η εξαγωγή reports κι ένα για την επικοινωνία του ελεγκτή (Arduino).

Η πρώτη διεπαφή «IdentityService» διεκπεραιώνει εργασίες σχετικά με τη δημιουργία νέων χρηστών, επαλήθευσης στοιχείων εισόδου, δημιουργία JWT και Refresh Token. Έχει ξεχωριστό ρόλο, γιατί είναι αυτό το API που θα δώσει τη δυνατότητα τόσο σε χρήστες, όσο και στην εξυπηρέτηση πελατών να εισέλθουν στο σύστημα και να χρησιμοποιήσουν τις λειτουργίες του. Επομένως, θα πρέπει να μπορεί να ενεργήσει αυτόνομα ακόμα και αν οι υπόλοιπες υπηρεσίες δε λειτουργούν.

Η επόμενη διεπαφή «UserService» επεξεργάζεται όλα τα αιτήματα που γίνονται για την ανάκτηση πληροφοριών σχετικών με τους χρήστες του συστήματος. Υποστηρίζει εφαρμογές χρηστών, όπως ιστοσελίδες ή εφαρμογές κινητών, αλλά και εφαρμογές που απευθύνονται σε εξυπηρέτηση πελατών. Μερικές από τις διεργασίες της είναι η ανάκτηση πληροφοριών κάποιου χρήστη και η ενημέρωσή τους, η διαχείριση των αυτοκινήτων ενός χρήστη και η δημιουργία reports.

Τέλος, η διεπαφή «ParkingService» έχει αποκλειστικό σκοπό να εξυπηρετεί αιτήματα που έρχονται από τον ελεγκτή του πάρκινγκ (Arduino). Ο διαχωρισμός αυτός έγινε με βάση τη συνέχεια (continuity) του συστήματος σε ακραίες καταστάσεις και την αποδοτικότερη απόκρισή του στα αιτήματα που καλείται να διαχειριστεί. Η απομόνωση αυτή μειώνει τον κίνδυνο να γεμίσει η ουρά της υπηρεσίας με αιτήματα που θα έχουν ως αποτέλεσμα την αυξανόμενη καθυστέρηση των απαντήσεων. Επίσης, ως αυτόνομη

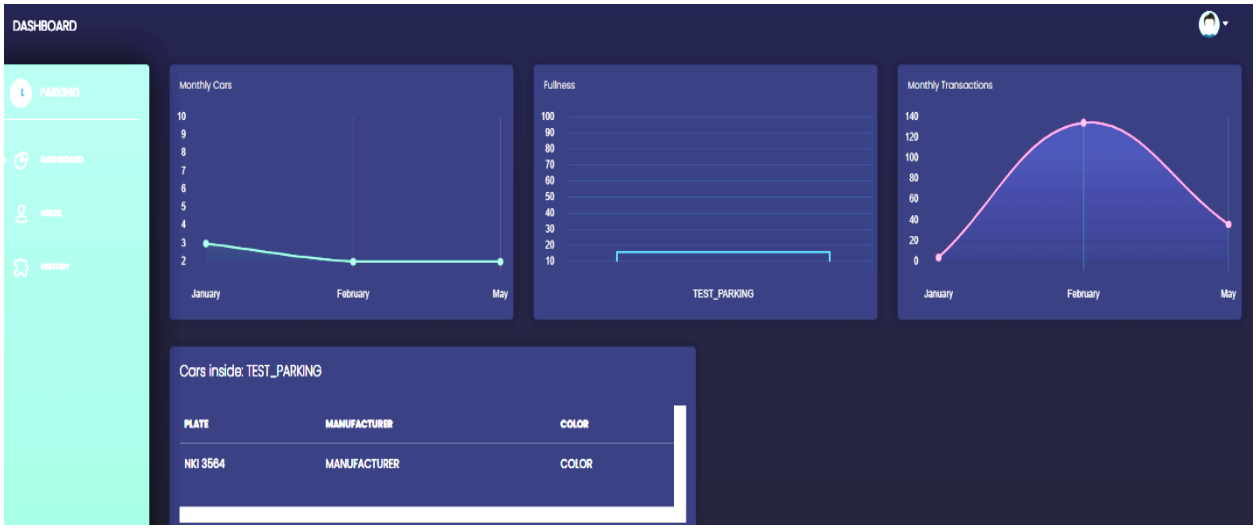
υπηρεσία, μπορεί να έχει τους δικούς της κανόνες σε επίπεδο server (firewall rules) για να αυξήσει την ασφάλειά της.

3.7.7 Εφαρμογές χρηστών

Για τις ανάγκες της εργασίας χρειάστηκε να δημιουργηθούν δύο εφαρμογές χρήστη. Μια που απευθύνεται αποκλειστικά στον διαχειριστή και μια για τους χρήστες του συστήματος.

Οι εφαρμογές αυτές χτίστηκαν πάνω στο framework της VueJS. Επιλέχθηκε το συγκεκριμένο framework, επειδή είναι ανοιχτού κώδικα και έχει ολοένα και αυξανόμενο κοινό που το χρησιμοποιεί. Συνεπώς, οι αναβαθμίσεις, η υποστήριξη και η αντιμετώπιση προβλημάτων είναι ανάλογες του κοινού που απευθύνονται.

Η πρώτη εφαρμογή καλύπτει τις ανάγκες του διαχειριστή του συστήματος. Παρέχει βασικές λειτουργίες και πληροφορίες, όπως δείκτες απόδοσης KPIs, διαχείριση χρηστών και αυτοκινήτων και ιστορικό κινήσεων. Παρακάτω φαίνεται η αρχική οθόνη του διαχειριστή, αφότου εισέλθει στο σύστημα με τους κωδικούς του.



Εικόνα 3.18: Panel διαχειριστή

Ανάλογη σελίδα έχουν και οι χρήστες του συστήματος. Οι βασικές τους λειτουργίες περιορίζονται στη διαχείριση του προφίλ τους, των αμαξιών που έχουν στην κατοχή τους και το ιστορικό κινήσεων τους.

Πακέτα

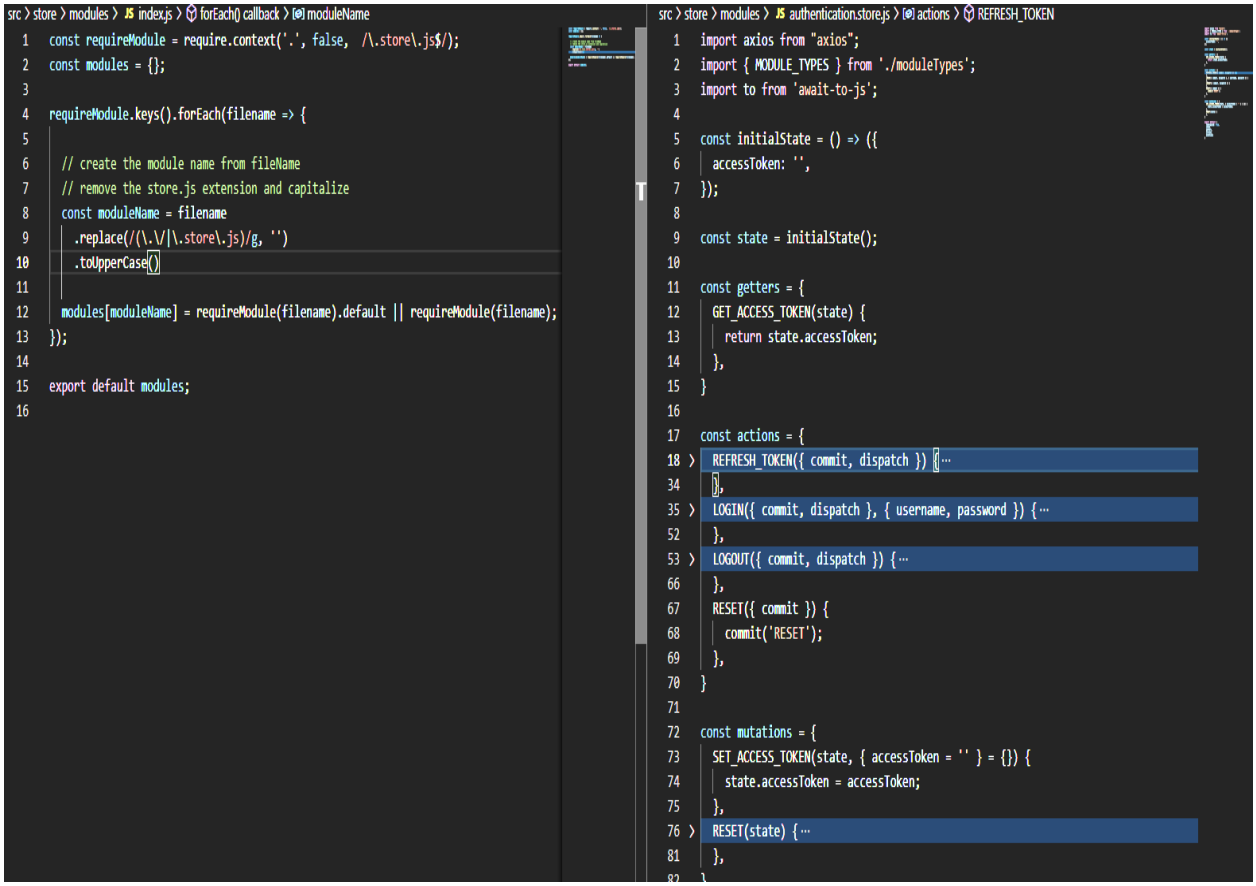
Για την VueJS υπάρχουν επίσης πολλά πακέτα (packages) που μπορούν να εγκατασταθούν για να υποστηρίξουν πολύπλοκες διαδικασίες. Τα βασικότερα πακέτα που χρησιμοποιήθηκαν στο χτίσιμο των εφαρμογών ήταν τα Vue CLI, Vuex και axios.

Το Vue CLI είναι ένα πακέτο που δίνει τη δυνατότητα χρήσης εντολών «vue» από τερματικό για γρήγορη δημιουργία εφαρμογών. Παρέχει όλα εκείνα τα πακέτα διαχείρισης, σύνδεσης και μετατροπής των αρχείων της VueJS σε μορφή κατανοητή από τα προγράμματα περιήγησης με έναν πιο οργανωμένο τρόπο για μεγαλύτερη διευκόλυνση.

Η βιβλιοθήκη Vuex είναι ένα πρότυπο διαχείρισης της κατάστασης της εφαρμογής. Ουσιαστικά είναι ένα κεντρικό σημείο αποθήκευσης των καίριων στοιχείων της που διασφαλίζει την ορθότητά τους και την αλλαγή τους μέσω συγκεκριμένων διαδικασιών. Διασφαλίζει έτσι την ομαλή λειτουργία της εφαρμογής, ελαττώνοντας τις απρόβλεπτες καταστάσεις που μπορεί να δημιουργηθούν από μη αναμενόμενες αλλαγές.

Ένα πολύ βασικό χαρακτηριστικό της Vuex είναι η οργάνωση των δεδομένων σε ενότητες. Είθισται να διαχωρίζεται η πληροφορία σε μικρές οντότητες ανάλογα με τον σκοπό που προορίζονται για ευκολότερη διαχείρισή τους.

Παρακάτω φαίνεται ένα κοινό στιγμιότυπο των εφαρμογών και αφορά την αυθεντικοποίηση του χρήστη. Το JWT Token του χρήστη αποθηκεύεται στη μεταβλητή «accessToken» και παρέχεται οπουδήποτε χρειαστεί μέσω της μεθόδου «GET_ACCESS_TOKEN». Για να αλλάξει η τιμή του θα πρέπει να χρησιμοποιηθεί η μέθοδος «SET_ACCESS_TOKEN» που βρίσκεται κάτω από τα «mutations», η οποία με τη σειρά της μπορεί μόνο να καλεστεί από κάποιο action.



Εικόνα 3.19: Διαχείριση Tokens μέσω Vuex

Το πακέτο axios είναι ένα εργαλείο για χρήση HTTP κλήσεων σε node.js και προγράμματα περιήγησης βασισμένο στα Promises. Με τον ίδιο κώδικα μπορεί να εκτελεστεί και στα δυο περιβάλλοντα γι' αυτό έχει χαρακτηριστεί και ισομορφικό. Για το καθένα από τα περιβάλλοντα χρησιμοποιεί είτε native node.js, είτε XMLHttpRequests.

ΚΕΦΑΛΑΙΟ 4: ΥΛΟΠΟΙΗΣΗ ΣΥΣΤΗΜΑΤΟΣ

4.1 Εισαγωγή

Στο παρακάτω κεφάλαιο, περιγράφεται αρχικά ο σκοπός της παρούσας μεταπτυχιακής διπλωματικής εργασίας κι έπειτα ακολουθεί η περιγραφή του συστήματος, δηλαδή ο τρόπος με τον οποίο λειτουργεί, όπως και ο τρόπος υλοποίησής του με τη δημιουργία μακέτας.

4.2 Σκοπός εργασίας

Αντικείμενο μελέτης, υλοποίησης, αλλά και σκοπός της παρούσας μεταπτυχιακής διπλωματικής εργασίας είναι η κατασκευή ενός «έξυπνου» συστήματος πάρκινγκ, το οποίο πρόκειται να διευκολύνει τους χρήστες του μέσω αυτοματοποιημένων διαδικασιών οι οποίες συμβάλλουν στη βελτίωση της ποιότητας των παρεχόμενων υπηρεσιών για τους διαθέσιμους χώρους στάθμευσης.

Μέσω του συστήματος αυτού παρέχονται εκτενείς πληροφορίες στον ιδιοκτήτη-διαχειριστή του πάρκινγκ καθώς και βασικές πληροφορίες στους χρήστες-πελάτες μέσω μιας web διεπαφής.

4.3 Περιγραφή συστήματος

Για να μπορέσει να γίνει χρήση της υπηρεσίας του smart parking θα πρέπει ο χρήστης-πελάτης να πραγματοποιήσει εγγραφή μέσω της ιστοσελίδας όπου θα καταχωρίσει τα απαραίτητα στοιχεία, όπως ονοματεπώνυμο, πινακίδα, αριθμό πιστωτικής κάρτας κ.λπ.

Στη συνέχεια, ο πελάτης έχει τη δυνατότητα να επισκεφθεί οποιοδήποτε πάρκινγκ σε οποιοδήποτε μέρος του πλανήτη με την προϋπόθεση ότι αυτός ο χώρος στάθμευσης θα είναι ήδη ενταγμένος μέσα στην υπηρεσία.

Η διαδικασία που ακολουθείται για την ένταξη του οχήματος εντός του χώρου στάθμευσης, αναλύεται εκτενώς παρακάτω:

1. Το αυτοκίνητο πλησιάζει τη μπάρα στην είσοδο του πάρκινγκ.
2. Ο αισθητήρας εντοπίζει ότι ένα όχημα είναι σε αναμονή και στέλνει σήμα στο Arduino για να προβεί στις κατάλληλες ενέργειες.
3. Το Arduino δίνει εντολή στο ESP32-CAM για τη λήψη φωτογραφίας της πινακίδας του οχήματος.
4. Στην ψηφιακή οθόνη αναγράφεται κατάλληλο μήνυμα ότι ένα όχημα εισέρχεται εντός του χώρου του πάρκινγκ, προς αποφυγή ατυχημάτων.
5. Με την ολοκλήρωση της λήψης φωτογραφιών από το ESP32-CAM, γίνεται η κατάλληλη κλήση στο API όπου στέλνει τη φωτογραφία στον server.
6. Ο server μέσω κατάλληλων διαδικασιών στέλνει τη φωτογραφία δια μέσου κλήσης API σε ένα 3rd party service, το οποίο με την τεχνική του image to text recognition, εντοπίζει το κείμενο μέσα στη φωτογραφία και επιστρέφει στον server το αποτέλεσμα.
7. Ο server ψάχνει στη λίστα πελατών να βρει αν έχει γίνει εγγραφή του οχήματος στην υπηρεσία κι εφόσον το εντοπίσει, ξεκινάει ένας μετρητής χρέωσης (χρονοχρέωση) και στέλνει στο ESP32-CAM όλες τις απαραίτητες πληροφορίες για το όχημα αυτό.
8. Το ESP32-CAM, μόλις λάβει την απάντηση, την προωθεί στο Arduino μέσω σειριακής πύλης.
9. Το Arduino με βάση τα αποτελέσματα που έχει λάβει, αποφασίζει αν θα ανοίξει τη μπάρα και θα εμφανίσει ένα μήνυμα καλωσορίσματος ή θα βγάλει κατάλληλο μήνυμα στην οθόνη σε περίπτωση κάποιου σφάλματος (π.χ. μη εντοπισμός οχήματος).
10. Κατά τη διαδικασία εξόδου του οχήματος, ακολουθείται η ίδια διαδικασία με τη μόνη διαφορά ότι πραγματοποιείται η χρέωση με βάση τη χρονομέτρηση που έχει γίνει και τον ισχύοντα τιμοκατάλογο, μέσω της πιστωτικής κάρτας.
11. Το σύστημα ενημερώνει τον χρήστη μέσω e-mail για αυτήν την χρέωση.

Αξίζει να σημειωθεί ότι ο χρήστης έχει πρόσβαση στην ιστοσελίδα όπου μπορεί να δει το ιστορικό κινήσεων και χρεώσεων, να διαχειριστεί τα οχήματά του και πλήθος άλλων λειτουργιών.

4.4 Υλοποίηση συστήματος

Για την υλοποίηση του smart parking, κρίθηκε αναγκαία η δημιουργία μακέτας. Πιο συγκεκριμένα, για τη δημιουργία μακέτας χρησιμοποιήθηκαν τα παρακάτω υλικά:

- Ξύλινη βάση διαστάσεων 50cm x 50cm, για την τοποθέτηση όλων των εξαρτημάτων πάνω στη βάση αυτή.

Εικόνα 4.1: Ξύλινη βάση μακέτας

- Πλαστικοί διάδρομοι καλωδίων (κανάλι καλωδίων), για την εισαγωγή όλων των καλωδίων που ενώνουν τα επιμέρους εξαρτήματα.

Εικόνα 4.2: Κανάλι καλωδίωσης

- Αυτοκόλλητα

Εικόνα 4.3: Κεντρικό αυτοκόλλητο

Εικόνα 4.4: Αυτοκόλλητο σήμανσης

Εικόνα 4.5: Αυτοκόλλητο βάσης

- Χαρτόνια

Εικόνα 4.6: Διαχωριστικό χαρτόνι

- Αυτοκινητάκια

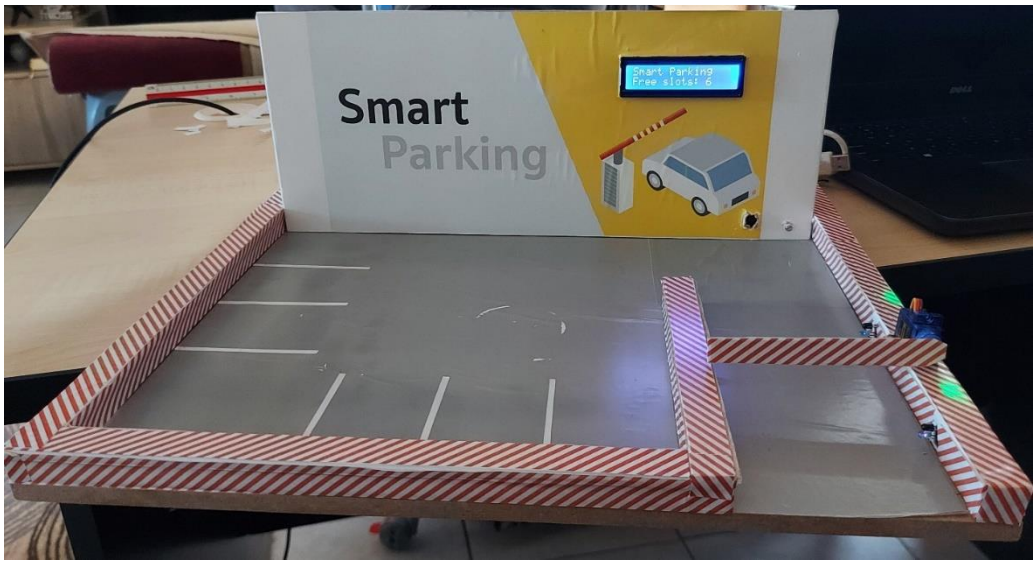
Εικόνα 4.7: Μεταλλικά αυτοκινητάκια

- Γλωσσοπίεστρα χειροτεχνιών (για τη μπάρα)

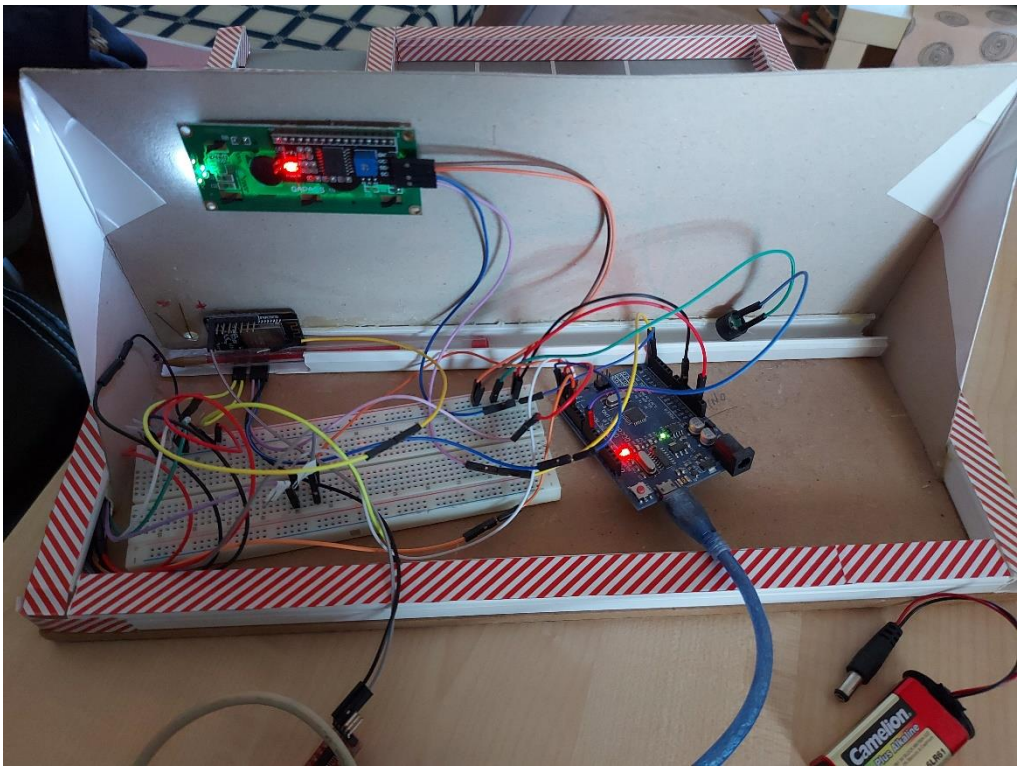
Εικόνα 4.8: Γλωσσοπίεστα χειροτεχνιών

Επιπλέον, χρησιμοποιήθηκαν τα ηλεκτρονικά μέρη της εργασίας (βλ. κεφ.2.3.2, 2.3.3 της παρούσας εργασίας).

- Μακέτα



Εικόνα 4.9: Μακέτα κατασκευής



Εικόνα 4.10: Ηλεκτρονικός εξοπλισμός

Εικόνα 4.11: Εισαγωγή οχήματος

ΚΕΦΑΛΑΙΟ 5: Επίλογος

5.1 Σύνοψη

Η ανάγκη για τη δημιουργία έξυπνων συστημάτων τα οποία πρόκειται να διευκολύνουν τη ζωή των ανθρώπων, ενέπνευσε την υλοποίηση της παρούσας διπλωματικής εργασίας, η οποία στοχεύει στη σχεδίαση και στην ανάπτυξη ενός συστήματος παροχής αυτοματοποιημένων υπηρεσιών στάθμευσης οχημάτων με χρονοχρέωση.

Αρχικά, στο πρώτο κεφάλαιο έγινε αναφορά στα συστήματα ANPR και ακολούθησε η συνοπτική περιγραφή του συστήματος που δημιουργήθηκε στο πλαίσιο της εν λόγω εργασίας.

Στο δεύτερο κεφάλαιο πραγματοποιήθηκε μία ιστορική αναδρομή για τους μικροελεγκτές και ειδικότερα για τον μικροελεγκτή Arduino. Έπειτα, αναλύθηκαν οι εκδόσεις του Arduino και τα συγκεκριμένα παρελκόμενά του τα οποία χρησιμοποιήθηκαν σε αυτή την εργασία. Ακολούθησε η περιγραφή του περιβάλλοντος ανάπτυξης και η επεξήγηση του κώδικα που δημιουργήθηκε.

Στο τρίτο κεφάλαιο αναλύθηκε η διαδικτυακή εφαρμογή. Πιο συγκεκριμένα, αναλύθηκε το σύστημα αυτοματοποιημένων υπηρεσιών οχημάτων με χρονοχρέωση σε επίπεδο κώδικα. Έγινε αναφορά στις απαιτήσεις που έχουν παρόμοια συστήματα, στις τεχνολογίες που χρησιμοποιούνται, στα εργαλεία και στην αρχιτεκτονική που ακολουθείται. Επιπρόσθετα, αναλύθηκε ο τρόπος με τον οποίο πραγματοποιείται η αναγνώριση της πινακίδας του κάθε οχήματος, αναλύεται ο κώδικας και η διαχείριση κλήσεων. Στη συνέχεια του κεφαλαίου, έγινε αναφορά στον τρόπο με τον οποίο πραγματοποιήθηκε η αυθεντικοποίηση στην πράξη. Επιπλέον, στο κεφάλαιο αυτό αναφέρθηκαν η διαχείριση σφαλμάτων, η βάση δεδομένων, οι διεπαφές προγραμματισμού χρήστη και οι διαδικτυακές εφαρμογές.

Στο τέταρτο κεφάλαιο, εκτός από τον σκοπό της εργασίας, περιγράφηκε ο τρόπος λειτουργίας του συγκεκριμένου συστήματος -τα βήματα που ακολουθούνται-, ο τρόπος υλοποίησής του και όσα υλικά χρησιμοποιήθηκαν για τη δημιουργία μακέτας.

Τέλος, ακολουθεί ο επίλογος ο οποίος αναφέρεται στα προβλήματα που παρουσιάστηκαν και πώς αυτά αντιμετωπίστηκαν, στις μελλοντικές επεκτάσεις και στα συμπεράσματα που προέκυψαν από τη συγκεκριμένη διπλωματική εργασία.

5.2 Προβλήματα και αντιμετώπιση

Σε αυτή την ενότητα παρατίθονται τα προβλήματα που εμφανίστηκαν καθώς επίσης και η αντιμετώπισή τους κατά τη διάρκεια υλοποίησης του συστήματος αυτοματοποιημένων υπηρεσιών στάθμευσης οχημάτων με χρονοχρέωση.

Η αναγνώριση πινακίδας από ένα σύστημα οπτικής αναγνώρισης χαρακτήρων δεν είναι πάντα μια εύκολη διαδικασία. Θα πρέπει να ελεγχθούν αρκετοί παράμετροι που επηρεάζουν μια εικόνα έτσι ώστε να μειωθεί στο ελάχιστο το ποσοστό εκείνων που δεν μπορούν να διαβαστούν. Στις παραμέτρους αυτές συγκαταλέγονται η ποιότητα της φωτογραφίας, η απόσταση του αυτοκινήτου από την κάμερα, ο φωτισμός της λήψης και τέλος, η εύρεση της πινακίδας μέσα από ένα σύνολο χαρακτήρων.

Σχετικά με την ποιότητα της φωτογραφίας, αυτό καθορίζεται από την εκάστοτε κάμερα που κάνει τη λήψη. Μεγάλης ποιότητας εικόνες αυξάνουν το μέγεθος του αρχείου κάτι που δημιουργεί καθυστέρηση στην επικοινωνία με τις υπόλοιπες υπηρεσίες. Για το λόγο αυτό στο συγκεκριμένο σύστημα αυτοματοποιημένων υπηρεσιών επιλέχθηκε ένα μέγεθος ικανό να αναγνωριστεί από την OCR υπηρεσία σε συνδυασμό με την «scaling» λειτουργία της.

Εκτός από την ποιότητα μιας εικόνας, βασικό ρόλο στην αναγνώριση κειμένου παίζει και το μέγεθός του. Ένα πολύ μικρό κείμενο θα είναι δύσκολο να απομονωθεί και να ξεχωρίσουν οι χαρακτήρες του. Άρα, για να είναι σε ένα ικανοποιητικό μέγεθος το κείμενο, θα πρέπει η φωτογραφία να ληφθεί όσο το δυνατόν πιο κοντά έτσι ώστε να είναι ευανάγνωστοι οι χαρακτήρες της πινακίδας του αυτοκινήτου.

Τέλος, για την ολοκλήρωση της αναγνώρισης μιας πινακίδας, από το σύνολο των χαρακτήρων που επιστρέφει η υπηρεσία OCR.Space θα πρέπει να διαχωριστούν αυτοί που ανήκουν στην πινακίδα του αυτοκινήτου. Το πρόβλημα αυτό δημιουργείται γιατί αρκετές φορές σύμβολα ή και αντικείμενα στο χώρο της εικόνας μπορεί να αναγνωρισθούν εσφαλμένα ως χαρακτήρες. Για τον καθαρισμό των επιπλέον

χαρακτήρων χρησιμοποιήθηκαν Regular Expressions που αντιστοιχούν σε πινακίδες αυτοκινήτων.

5.3 Μελλοντικές επεκτάσεις συστήματος

Στην ενότητα αυτή αναφέρονται κάποιες από τις πιθανές επεκτάσεις του συστήματος αυτοματοποιημένων υπηρεσιών στάθμευσης οχημάτων με χρονοχρέωση. Οι βελτιώσεις ή αναβαθμίσεις αυτές έχουν ως στόχο τη βελτίωση της εμπειρίας του χρήστη.

Αρχικά για την πιο εύκολη είσοδο και έξοδο των οχημάτων θα μπορούσαν να δημιουργηθούν πολλαπλά σημεία εισόδου-εξόδου, ανάλογα με τις ανάγκες του χώρου στάθμευσης, έτσι ώστε να γίνεται παράλληλα η εισαγωγή και εξαγωγή αυτοκινήτων. Η συγκεκριμένη βελτίωση προϋποθέτει σαφώς και την ανάλογη αύξηση των εξαρτημάτων για αναγνώριση και εξυπηρέτηση των χρηστών. Σε σχέση με τις υπηρεσίες back-end θα πρέπει να υπάρξουν μερικές αλλαγές για λόγους ασφαλείας. Ο ρόλος του κάθε αιτήματος θα πρέπει να είναι διακριτός, αν αναφέρεται δηλαδή σε είσοδο ή έξοδο ενός οχήματος, για να αποφευχθούν περιπτώσεις λάθους.

Ένα ακόμη σημείο που μπορεί να βελτιωθεί είναι η αναγνώριση άφιξης ενός οχήματος. Εκτός του αισθητήρα, θα μπορούσε να χρησιμοποιηθεί κι ένας μετρητής βάρους ο οποίος με τη σειρά του θα ενεργοποιεί το σύστημα ανίχνευσης πινακίδας. Με τον τρόπο αυτό μπορούν να ελαχιστοποιηθούν οι επιπλέον κλήσεις και να μειωθεί η συνολική επιβάρυνση του συστήματος καθώς οι μόνες λήψεις που θα πηγαίνουν προς επεξεργασία θα είναι αυτές που έχουν και το αντικείμενο προς αναγνώριση.

Για την αύξηση της ασφάλειας των χρηστών του συστήματος θα πρέπει να ακολουθηθεί το μοντέλο αυθεντικοποίησης δύο σημείων. Αυτό προϋποθέτει τα οχήματα να εξοπλιστούν με ένα NFC tag, συγκεκριμένο για κάθε πινακίδα αυτοκινήτου. Με τον τρόπο αυτό αποτρέπεται η χρήση πλαστής πινακίδας από άλλο πρόσωπο.

Μια εναλλακτική λύση στο θέμα ασφάλειας θα ήταν και αυτή της ειδοποίησης μέσω εφαρμογής κινητού. Σε κάθε είσοδο ή έξοδο του οχήματος από τον χώρο στάθμευση, ο χρήστης θα μπορούσε να λαμβάνει κάποιο μήνυμα ή να του ζητείται επιβεβαίωση. Η άμεση αυτή ενημέρωση του χρήστη θα μπορούσε να προλάβει ανεπιθύμητες καταστάσεις ή μη εξουσιοδοτημένες χρήσεις του συγκεκριμένου οχήματος.

Τέλος, θα μπορούσε να υλοποιηθεί ένας συνδυασμός αισθητήρων θέσεως και φωτισμού με απώτερο σκοπό την ταχύτερη εξυπηρέτηση των χρηστών του συστήματος. Οι αισθητήρες θέσεως θα μπορούσαν να δίνουν πληροφορίες για το ποιες ακριβώς θέσεις είναι ελεύθερες κι έτσι να ενημερώνουν τους οδηγούς των οχημάτων προς τα που θα πρέπει να κινηθούν για να τις βρουν. Η φωτεινή σήμανση, θα μπορούσε να ενημερώνει τους οδηγούς σε περιπτώσεις όπου στο χώρο στάθμευσης υπάρχει μόνο μια είσοδος και έξοδος, ότι ο διάδρομος είναι κατειλημμένος και θα πρέπει να περιμένουν. Έτσι, θα αποφεύγονταν καταστάσεις εμπλοκής όπου δυο ή και περισσότερα οχήματα βρίσκονταν ταυτόχρονα στο ίδιο σημείο και προσπαθούσαν να κάνουν χρήση της κάμερας και της μπάρας εισόδου-εξόδου.

5.4 Συμπεράσματα

Σκοπός της παρούσας μελέτης ήταν η δημιουργία ενός συστήματος παροχής αυτοματοποιημένων υπηρεσιών στάθμευσης οχημάτων με χρονοχρέωση.

Το πρώτο ερώτημα που τέθηκε ήταν ποια είναι τα πραγματικά συστήματα αναγνώρισης αριθμών κυκλοφορίας. Τα συστήματα, λοιπόν, τα οποία χρησιμοποιούνται με σκοπό την αναγνώριση αριθμών κυκλοφορίας είναι τα λεγόμενα ANPR και μπορούν να δεχτούν εικόνες από οποιαδήποτε κάμερα, συνοδευόμενα από υπέρυθρο φωτισμό. Επιπρόσθετα, για την αναγνώριση εφαρμόζονται τεχνικές οπτικής αναγνώρισης χαρακτήρων (Optical Character Recognition – OCR).

Το δεύτερο ερώτημα που τέθηκε ήταν ποια πλακέτα πρόκειται να χρησιμοποιηθεί για τη σχεδίαση και ανάπτυξη του συστήματος παροχής αυτοματοποιημένων υπηρεσιών στάθμευσης οχημάτων με χρονοχρέωση. Η πλακέτα που επιλέχθηκε με σκοπό την υλοποίηση του συγκεκριμένου συστήματος είναι ο μικροελεγκτής Arduino και συγκεκριμένα ο Arduino Uno. Η πλακέτα αυτή αποτελεί πλατφόρμα ανοικτού κώδικα και μπορεί κανείς να τη διαχειριστεί εύκολα. Η πλακέτα έχει τη δυνατότητα να διαβάζει εισόδους και να τις μετατρέπει σε εξόδους. Υπάρχει, επιπλέον, η δυνατότητα προγραμματισμού του μικροελεγκτή μέσω κώδικα, χρησιμοποιώντας συγκεκριμένη γλώσσα προγραμματισμού. Υπάρχουν αρκετές εκδόσεις Arduino, αλλά στην παρούσα κατασκευή χρησιμοποιήθηκε το Arduino Uno. Η εν λόγω έκδοση είναι η πιο διαδεδομένη από τις υπόλοιπες εκδόσεις και περιέχει όλα τα απαραίτητα στοιχεία για να είναι

λειτουργική. Εκτός από την ανάλυση της πλακέτας Arduino, στην εργασία περιγράφονται λεπτομερώς και τα εξαρτήματα που χρησιμοποιήθηκαν, το περιβάλλον ανάπτυξης και ο κώδικας προγραμματισμού.

Το τρίτο ερώτημα που τέθηκε ήταν ποιες τεχνολογίες, εργαλεία και αρχιτεκτονική θα πρέπει να χρησιμοποιηθούν, σύμφωνα με τις απαιτήσεις του συστήματος, ώστε να είναι λειτουργικό. Ειδικότερα, χρησιμοποιήθηκαν τεχνολογίες σε front-end, σε back-end και σε βάσεις δεδομένων. Χρησιμοποιήθηκαν εργαλεία για τη δημιουργία των υπηρεσιών διαδικτύου, των βάσεων δεδομένων και των ιστοσελίδων που πλοηγείται ο χρήστης, αλλά και ο διαχειριστής. Για την αρχιτεκτονική χρησιμοποιήθηκαν τεχνικές ανάπτυξης κώδικα και αυθεντικοποίησης, οι οποίες επιλέχθηκαν με βάση την καλύτερη διαχείριση δεδομένων και ανάπτυξης/συντήρησης του συγκεκριμένου συστήματος.

Το τέταρτο και τελευταίο ερώτημα που τέθηκε ήταν πώς υλοποιήθηκε πρακτικά το σύστημα παροχής αυτοματοποιημένων υπηρεσιών στάθμευσης οχημάτων με χρονοχρέωση. Αρχικά, δημιουργήθηκε η μακέτα του πάρκινγκ, ώστε να γίνει έπειτα ο προγραμματισμός της πλακέτας Arduino Uno. Στη συνέχεια, υλοποιήθηκε η δομή SaaS (Software as a service) ώστε να υποστηρίξει την υπηρεσία. Για να μπορέσει ο πελάτης του πάρκινγκ να χρησιμοποιήσει την υπηρεσία «Smart parking», θα πρέπει να προβεί σε κάποιες ενέργειες, όπως να κάνει εγγραφή μέσω της ιστοσελίδας και να καταχωρίσει τα απαραίτητα στοιχεία. Έπειτα, αφού γίνει η εγγραφή του, μπορεί να επισκεφθεί το πάρκινγκ και καθώς πλησιάζει το αυτοκίνητό του στην είσοδο, ο αισθητήρας θα εντοπίσει ότι υπάρχει ένα όχημα στην είσοδο και θα δώσει την εντολή στο Arduino να προβεί στις κατάλληλες ενέργειες. Εφόσον καταγραφεί από την κάμερα και αναγνωριστεί η πινακίδα του οχήματος ώστε να γίνει ταυτοποίηση του πελάτη, τότε η μπάρα ανοίγει και το όχημα μπορεί να εισέλθει στο πάρκινγκ. Παρόμοια είναι και η διαδικασία κατά την έξοδο, με τη μόνη διαφορά ότι κατά την έξοδο πραγματοποιείται και η χρέωση σύμφωνα με τη χρονομέτρηση που έγινε. Και στις δύο περιπτώσεις, η οθόνη προβάλλει το κατάλληλο μήνυμα καλωσορίσματος, ευχαριστήριο ή ενημερωτικό. Επιπλέον, δίνεται η δυνατότητα στον πελάτη να ελέγχει το ιστορικό κινήσεων-χρεώσεων, να διαχειρίζεται τα οχήματά του κ.ά.

Στις μελλοντικές επεκτάσεις θα μπορούσαν να γίνουν ορισμένες αλλαγές, ώστε να διευκολύνουν την είσοδο και την έξοδο των πελατών. Επίσης, θα μπορούσε να βελτιωθεί

η αναγνώριση άφιξης του κάθε οχήματος, όπου με τη βοήθεια ενός μετρητή βάρους θα ενεργοποιούνταν αυτόματα το σύστημα ανίχνευσης πινακίδας με σκοπό την αποτροπή επιπλέον κλήσεων από αντικείμενα που μπορεί να βρεθούν στην εμβέλεια του αισθητήρα. Ακόμα, για την καλύτερη ασφάλεια των πελατών, θα μπορούσε να ακολουθηθεί το μοντέλο αυθεντικοποίησης δύο σημείων (Two step authentication) με σκοπό την αποφυγή χρήσης πλαστών πινακίδων από άλλους χρήστες. Επιπρόσθετα, για την ασφάλεια των πελατών, καλή λύση είναι να ειδοποιούνται οι χρήστες μέσω εφαρμογής κινητού τηλεφώνου και να επιβεβαιώνουν ή όχι τις ενέργειες που προτείνονται. Ολοκληρώνοντας, θα μπορούσε να δημιουργηθεί ένας συνδυασμός αισθητήρων θέσεως και φωτισμού με σκοπό την ταχύτερη εξυπηρέτηση των χρηστών.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Ελληνόγλωσσες βιβλιογραφικές αναφορές

[1] Βυσανσιώτης Σ. & Καρβουνίδου Α. (χ.χ.). Υλοποίηση ασύρματης επικοινωνίας με χρήση μικροελεγκτών. Αδημοσίευτη πτυχιακή εργασία. ΤΕΙ Ηπείρου: Ιωάννινα.

[2] Γεωργιάδης Α. & Κλεφτογιώργης Ρ.Χ. (2015). Μελέτη και υλοποίηση συστήματος συλλογής και καταγραφής δεδομένων με Arduino. Αδημοσίευτη πτυχιακή εργασία. ΤΕΙ Σερρών: Σέρρες.

[3] Δημητρακόπουλος Α. (2017). Έξυπνα σπίτια με χρήση Arduino. Αδημοσίευτη πτυχιακή εργασία. ΤΕΙ Δυτικής Ελλάδας: Πάτρα.

[4] Νικολαρέας Π. (2019). Κατασκευή αυτόνομου πυροσβεστικού οχήματος με δυνατότητα ανίχνευσης και κατάσβεσης πύρινης εστίας. Αδημοσίευτη πτυχιακή εργασία. ΠΑΔΑ: Αθήνα.

Ξενόγλωσσες βιβλιογραφικές αναφορές

[5] Goldish L.H. & Taylor H.E. (1974). The Optacon: A Valuable Device for Blind Persons. New Outlook for the Blind. American Foundation for the Blind. pp. 49-56.

[6] Martin, C. M. (2018). Clean Architecture: A Craftsman’s Guide to Software Structure and Design. THE CLEAN ARCHITECTURE. Prentice Hall.

[7] Rajshree D. & Dharmendra R. (2014), “Automatic Number Plate Recognition System”, IJCSMC, Vol. 3, pg. 6-12.

[8] Schantz, Herbert F. (1982). The history of OCR, optical character recognition. Recognition Technologies Users Association.

Διαδικτυακές αναφορές

[9] https://www.arduino.cc/en/Guide/Introduction

[10] https://www.arduino.cc/en/Guide/Introduction

[11] https://azure.microsoft.com/en-us/

[12] https://docs.microsoft.com/el-gr/lifecycle/faq/dotnet-core

[13] https://docs.microsoft.com/el-gr/dotnet/csharp/tour-of-csharp/

[14]https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15

[15]https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern

[16] https://go.java/

[17] https://grobotronics.com/arduino-pro-mini-328-5v-16mhz.html

[18] https://i1.wp.com/randomnerdtutorials.com/wp-content/uploads/2019/03/ESP32-CAM-pinout-1.png?quality=100&strip=all&ssl=1

[19] https://jwt.io/

[20] https://microservices.io/

[21] https://nodejs.org/en/

[22] https://opensource.google/projects/tesseract

[23] https://threemammals.com/ocelot/

[24] https://visualstudio.microsoft.com/vs/

[25] https://vuejs.org/

[26] https://www.electroschematics.com/arduino-mega-adk-pinout/

[27] https://www.jetbrains.com/idea/

[28] https://www.microsoft.com/en-us/sql-server/

[29]https://www.seeedstudio.com/media/catalog/product/cache/ef3164306500b1080e8560b2e8b5cc0f/b/a/bazaar1003542_esp32cam2.jpg

[30] https://www.skroutz.gr/s/8231728/Arduino-Uno-SMD-Rev3.html

[31] https://www.skroutz.gr/s/8854949/FT232RL-FTDI-Serial-Adapter-Module.html

[32]https://www.why.gr/%CE%BA%CE%B1%CF%84%CE%B1%CF%83%CF%84%CE%B7%CE%BC%CE%B1/open-hardware/arduino/arduino-main-boards/arduino-esplora/

[33] https://xenyltechbd.com/shop/development-board/arduino-lilypad-at-mega328p/