ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Υλοποίηση συστήματος APNR (Automated Plate Number Recognition) με τη χρήση Μηχανικής Μάθησης και Computer Vision»

Του φοιτητή
Μπεσίρη Χρήστου
Αρ. Μητρώου: 2019240

Επιβλέπων
Δρ.Διαμαντάρας Κωνσταντίνος
Καθηγητής

Ημερομηνία 27/03/2026

Τίτλος Δ.Ε. Υλοποίηση συστήματος αυτόματης αναγνώρισης πινακίδων αυτοκινήτων με τη χρήση Μηχανικής Μάθησης.

Κωδικός Δ.Ε. 24193

Ονοματεπώνυμο φοιτητή ΜΠΕΣΙΡΗΣ ΧΡΗΣΤΟΣ

Ονοματεπώνυμο εισηγητή Δρ. ΚΩΝΣΤΑΝΤΙΝΟΣ ΔΙΑΜΑΝΤΑΡΑΣ

Ημερομηνία ανάληψης Δ.Ε.01/04/2024

Ημερομηνία περάτωσης Δ.Ε. 27/03/2026

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

«Αφιέρωση»

Στην οικογένειά μου που με στήριξε στην ανηφόρα, στη σύζυγο και τον υιό μου.

Στους καθηγητές μου που μου έδωσαν τη γνώση, που με ενέπνευσαν και με βοήθησαν όπου χρειάστηκε.

Στον συνεργάτη μου και μέντορα Καλαϊτζίδη Αναστάσιο που πίστεψε σε εμένα , και μου έδειξε τον σωστό δρόμο για την υλοποίηση μιας επαγγελματικής αρχιτεκτονικής λογισμικού.

Και ΚΥΡΙΩΣ σε όλους αυτούς που με αμφισβήτησαν όταν τους είπα ότι σχεδόν στα σαράντα μου θα ξανασπουδάσω κάτι που αγαπώ, και μου απάντησαν : «Δε μπορείς να κάνεις κάτι τέτοιο- δεν γίνεται.» Αυτή είναι η απάντησή μου : «κοιτάξτε να μαθαίνετε πως γίνεται»

Στους συμφοιτητές μου , που ξενυχτήσαμε ώρες ατελείωτες με προγράμματα και bugs για να καταφέρουμε τελικά να φτάσουμε εδώ.

Ψηλά το κεφάλι- με κόπο και υπομονή καταφέρνουμε το ακατόρθωτο.

Πρόλογος

Το επάγγελμά μου τα τελευταία 20 έτη της ζωής μου είναι αστυνομικός. Αστυνομικός της πρώτης γραμμής , στην υπηρεσία του πολίτη και πρώτος ανταποκρινόμενος στην στιγμή της ανάγκης. Όλα αυτά τα έτη έγινα μάρτυρας της άσχημης πλευράς της κοινωνίας μας. Σε πολλές περιπτώσεις κλήθηκα να βοηθήσω συμπολίτες μας οι οποίοι είχαν πέσει θύματα κλοπής των οχημάτων τους. Με σκοπό την υλοποίηση ενός συστήματος που μπορεί να βρίσκει κλεμμένα οχήματα ώστε αυτά να επιστρέψουν στους ιδιοκτήτες τους άμεσα επέλεξα το θέμα. Η υλοποίηση του συστήματος βασίζεται σε ένα prototyping board Raspberry Pi , έκδοση 3B+. Με μικρό κόστος και τρομερά μικρές υπολογιστικές δυνατότητες, η επίτευξη του στόχου φάνταζε αρχικά ουτοπική. Τώρα μετά την ολοκλήρωση του συστήματος κι έχοντας μείνει τρομερά ικανοποιημένος , αποδεικνύεται περίτρανα ότι δεν είναι πάντα απαραίτητες δαπάνες εκατομμυρίων για να βελτιώσουν τη ζωή των πολιτών. Σαν μηχανικός λογισμικού αντιμετώπισα τεράστιες δυσκολίες λόγω των υπολογιστικών πόρων, έμαθα όμως πολλά πράγματα στην προσπάθεια να τις ξεπεράσω. Εύχομαι το έργο αυτό να βοηθήσει κάποια στιγμή την κοινωνία μας.

Περίληψη

Σκοπός της εργασίας είναι η δημιουργία ενός ολοκληρωμένου συστήματος αυτόματης αναγνώρισης πινακίδων κυκλοφορίας (APNR) υλοποιημένο εξ' ολοκλήρου σε ένα prototyping board (Raspberry Pi) έκδοσης 3B+. Με τη χρήση μεθόδων και βιβλιοθηκών που εφαρμόζονται στην Υπολογιστική Όραση (Computer Vision) , οπτική αναγνώριση ενός οχήματος, της πινακίδας κυκλοφορίας του , εξαγωγή πληροφορίας από την εικόνα (αριθμός κυκλοφορίας οχήματος)με τη χρήση των κατάλληλων μοντέλων Μηχανικής Μάθησης και κατόπιν έλεγχος αυτής σε τοπική "dummy" βάση δεδομένων sqlite3 με καταχωρημένα οχήματα ως "κλαπέντα – αναζητούμενα". Ολόκληρο το θέμα μπορεί να αναλυθεί συνοπτικά με τον διαχωρισμό των επιμέρους λειτουργιών. Το μεγάλο πρόβλημα είναι η μικρή υπολογιστική ικανότητα της πλατφόρμας καθώς στερείται κάρτας γραφικών (GPU) και μεγάλης μνήμης RAM (1 GB). Η χρήση ενός τέτοιου συστήματος – απαιτεί την ύπαρξη δικλείδων ασφαλείας. Η ίδια η αρχιτεκτονική που υλοποιήθηκε εγγυάται την ασφάλεια των δεδομένων και των συναλλαγών. Η είσοδος στην εφαρμογή με διαπιστευτήρια, ο έλεγχος των ενεργειών με καταγραφή τους (logging) , ο έλεγχος πρόσβασης με βάση τους ρόλους του χρήστη – Role Based Access Control (RBAC) και η λογική του "Separation of Concerns" (SoC) στην υλοποίηση εγγυώνται την ασφάλεια της εφαρμογής. Η λειτουργικότητα μπορεί εν δυνάμει να επεκταθεί περαιτέρω με τη χρήση κατάλληλων "API calls" στις βάσεις δεδομένων της Ελληνικής Αστυνομίας, ώστε να εντοπίζονται άμεσα κλεμμένα οχήματα που τυχόν βρίσκονται σε κυκλοφορία. Για την υλοποίηση απαιτούνται : Prototyping board (Raspberry Pi) , συσκευή εισόδου οπτικού σήματος (camera), passive buzzer module για την ηχητική ανάδραση , καθώς και χρήση βιβλιοθηκών της γλώσσας προγραμματισμού Python.

«Implementation of an Automatic Plate Number Recognition System using Machine Learning and Computer Vision»

«Christos Besiris»

Abstract

This thesis presents the development and implementation of an Automatic Plate Number Recognition (APNR) system, also referred as Automatic License Plate Recognition (ALPR) system, with a prototyping board (Raspberry Pi 3B+) as the Edge Computing device. The use of Computer Vision, and Machine Learning is fundamental. The objective of the research is to design a robust and secure pipeline capable of accurately identifying the license plates of vehicles from a camera feed and using sophisticated mechanisms to ensure the safety of the application, the data integrity and functionality on such low computing resource devices- completely offline – minimizing also the exposure of the local database to unauthorized access. The approach starts with the video capture – passing the feed to a YOLO finetuned model for vehicle license plate detection , which afterwards passes through the regions of interest (ROI's) to a Compact Convolutional Transformer model (CCT) or a Mobile Vision Transformer (MobileViT). The result of the recognition is looked up in a local sqlite database and if there is a record for that vehicle an alert is returned to the user.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω την οικογένειά μου που με ανέχθηκε και μου συμπαραστάθηκε σε όλο αυτό το δύσκολο έργο καθώς και όλο το διδακτικό προσωπικό του Τμήματος που μου έδωσε τη γνώση και τη δύναμη να συνεχίσω.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία
APNR	Automatic Plate Number Recognition
ALPR	Automatic License Plate Recognition
GPU	Graphics Processing Unit
RAM	Random Access Memory
RBAC	Role-Based Access Control
SoC	Separation of Concerns
API	Application Programming Interface
YOLO	You Only Look Once
ROI	Region Of Interest
CCT	Compact Convolutional Transformer
MobileViT	Mobile Visual Transformer
CNN	Convolutional Neural Network
ΣΝΔ	Συνελικτικά Νευρωνικά Δίκτυα
R-CNN	Region-Based Convolutional Neural Network
WBF	Weighted Boxes Fusion
NMS	Non-Maximum Suppression
IoU	Intersection over Union
mAP	mean Average Precision
AI	Artificial Intelligence
ΤΝ	Τεχνητή Νοημοσύνη
HAT	Hardware Attached on Top
TOPS	Tera Operations Per Second
ΜΓΜ	Μεγάλα Γλωσσικά Μοντέλα
LLM	Large Language Model
ARM	Advanced RISC Machine
RISC	Reduced Instruction Set Computing
NVMe	Non-Volatile Memory Express

GPIO	General Purpose Input/ Output
V	Volt (Μονάδα μέτρησης ηλεκτρικής τάσης)
A	Ampere (Μονάδα μέτρησης ηλεκτρικού ρεύματος)
USB	Universal Serial Bus
HDMI	High Definition Multimedia Interface
F-F	Female- Female
SoC	Separation of Concerns
UI	User Interface
DAL	Data Access Layer
CRUD	Create Read Update Delete
MVC	Model View Controller
MCV	Model Controller View
ReLU	Rectified Linear Unit
IDE	Integrated Development Environment
ΕΛΑΚ	Ελεύθερο Λογισμικό Ανοιχτού Κώδικα
IAM	Identity and Access Management
NIST	National Institute of Standards and Technology
IAL	Identity Assurance Level
AAL	Authenticator Assurance Level
FAL	Federation Assurance Level
MFA	Multi Factor Authentication
HTTP	Hyper Text Transfer Protocol
JWT	JSON Web Token
JSON	JavaScript Object Notation
SoD	Separation of Duties
PoLP	Principle of Least Privilege
RNN	Recurrent Neural Network
LLM	Large Language Model
ViT	Vision Transformer
ONNX	Open Neural Network Exchange
DI	Dependency Injection
ORM	Object- Relational Mapping
RE	Regular Expressions

Κεφάλαιο 1ο: Εισαγωγή

1.1 Εισαγωγή

Από την πρώτη στιγμή που κυκλοφόρησαν τα πρώτα οχήματα , από τα πρώτα αυτοκίνητα και τις μοτοσυκλέτες, δημιουργήθηκε η ανάγκη της αναγνώρισης του οχήματος και κατ' επέκταση του ιδιοκτήτη ή οδηγού του. Η πιο άμεση κι εύκολη λύση ήταν η δημιουργία μεταλλικών πλαισίων με χαρακτηριστικά αλφαριθμητικά, ανάλογα με τον τόπο στον οποίο κυκλοφορεί – το κράτος και την περιφέρειά του. Αυτά τα πλαίσια είναι οι γνωστές σε όλους μας πινακίδες κυκλοφορίας οχήματος. Είναι μοναδικές για κάθε κράτος, εξασφαλίζοντας την επιτυχημένη ταυτοποίηση. Η απαίτηση της διάκρισης ήρθε από νωρίς για νομικούς λόγους πρωτίστως. Σε μια ευνομούμενη πολιτεία, η έλλειψη των πινακίδων κυκλοφορίας θα οδηγούσε σε αναρχία στους δρόμους, καθώς ο σεβασμός στο κοινωνικό σύνολο και τους υπόλοιπους χρήστες μιας οδού δεν είναι αυτονόητος. Εκεί έρχεται να προλάβει άσχημες καταστάσεις ο ρόλος του Κράτους, ώστε να μην έχουν τη δυνατότητα οι παραβάτες να μένουν α-ταυτοποίητοι και συνεπώς ατιμώρητοι.

Σε πολλές εφαρμογές στις μέρες μας, απαιτείται η αναγνώριση της πινακίδας ενός οχήματος κυκλοφορίας. Από τις σύγχρονες κάμερες εντοπισμού παραβάσεων , στις κάμερες APNR που ανοίγουν και κλείνουν μπάρες σε χώρους στάθμευσης, σε ευαίσθητες εγκαταστάσεις που επιτηρούνται με κλειστά κυκλώματα καμερών και σε πολλές ακόμη εφαρμογές, βλέπουμε να γίνεται χρήση συστημάτων αναγνώρισης πινακίδων κυκλοφορίας. Η αυτοματοποίηση και η χρήση Υπολογιστικής Όρασης σε συνδυασμό με Βαθειά Μάθηση και Νευρωνικά Δίκτυα συνετέλεσαν καθοριστικά στο σκοπό αυτό.

Τις δύο τελευταίες δεκαετίες ειδικά, καθώς η πρόοδος στην τεχνολογία των Νευρωνικών Δικτύων είναι συγκλονιστική οι χρήσεις τους έχουν αλλάξει την πραγματικότητα και την καθημερινότητά μας. Για να δουλέψει σωστά ένα σύστημα εντοπισμού και αναγνώρισης πινακίδων , πρέπει να υπάρξει μια σωστή επιλογή επιμέρους «τμημάτων» και σωστή συνεργασία μεταξύ τους. Το «στήσιμο» απαιτεί μια ροή πληροφορίας μέσω αγωγού ή "pipeline" όπως ονομάζεται. Ανάλογα με τις απαιτήσεις σε υπολογιστικούς πόρους του συστήματος και τις δυνατότητες του «υλισμικού» (hardware) γίνεται και η επιλογή των κατάλληλων μοντέλων Μηχανικής/ Βαθειάς Μάθησης και Νευρωνικών Δικτύων.

1.2 Βέλτιστη υλοποίηση συστήματος APNR.

Όπως προαναφέρθηκε, η αρχιτεκτονική της εφαρμογής παίζει πολύ σημαντικό ρόλο. Τον πρωταγωνιστικό ρόλο όμως κατέχουν τα μοντέλα Βαθειάς / Μηχανικής Μάθησης που χρησιμοποιούνται για την υλοποίηση του μηχανισμού «συμπεράσματος» ή αλλιώς "inference". Άλλα μοντέλα είναι κατάλληλα για την κατηγοριοποίηση σε κλάσεις -αντικειμένων που συμπεριλαμβάνονται στην εικόνα ή στη ροή εισόδου γενικά , ενώ άλλα δουλεύουν καλύτερα ψάχνοντας συγκεκριμένες κλάσεις τμηματικά. Σε μια υποδομή με άφθονους πόρους μπορεί να χρησιμοποιηθεί ένα ensemble, μια διαφορετική προσέγγιση στην αναγνώριση. Ενώ στο γραμμικό pipeline έχουμε τη διαδοχική μεταφορά πληροφορίας από επίπεδο σε επίπεδο και από μοντέλο σε μοντέλο, στο ensemble τα πράγματα είναι διαφορετικά [1] :η είσοδος- είτε πρόκειται για εικόνα , είτε για ροή – αντί να τροφοδοτήσει ένα μοντέλο και να περιμένουμε την έξοδο – μπαίνει ως είσοδος **ταυτόχρονα** σε διαφορετικά μοντέλα που το καθένα έχει ιδιαίτερα γνωρίσματα. Κι εδώ είναι η βελτιστοποίηση της εφαρμογής. Γενικά, σε ένα τέτοιο

σύστημα προτιμάται η χρήση μοντέλων Βαθιάς Μάθησης – όπως είναι τα Συνελικτικά Νευρωνικά Δίκτυα (Convolutional Neural Networks – CNN). Υπάρχουν διάφοροι αλγόριθμοι που χρησιμοποιούν τα ΣΝΔ, με κύρια διαφορά ανάμεσά τους τον τρόπο προσέγγισης του προβλήματος : Κάποια «κβαντίζουν» τον χώρο ελέγχου -την εικόνα / frame που δέχονται- σε μικρότερα τμήματα και αναζητούν «ανά περιοχή» τα αντικείμενα των κλάσεων που έχουν εκπαιδευτεί να βρίσκουν. Καθώς η αναζήτηση γίνεται τμηματικά και ανά περιοχή οι αλγόριθμοι αυτοί χαρακτηρίζονται ως Region-Based CNN ή R-CNN. Είναι αποδοτικοί στην ανεύρεση ενός αντικειμένου με ακρίβεια στο κάδρο τους, αλλά αυτό κοστίζει σε χρόνο και σε υπολογιστική δύναμη καθώς επαναλαμβάνουν την αναζήτησή τους πολλές φορές -όσες και οι κβαντισμένες περιοχές- για κάθε κλάση αντικειμένου. Στον αντίποδα έχουμε μοντέλα όπως το ιδιαίτερα δημοφιλές You Only Look Once (YOLO) το οποίο κάνει ακριβώς αυτό που περιγράφει η ονομασία του: Με μία και μόνο σάρωση της εισόδου – αναγνωρίζει τις κλάσεις αντικειμένων στις οποίες έχει εκπαιδευτεί. Οι εξελίξεις στις εκδόσεις των μοντέλων YOLO από το 2015 που εμφανίστηκαν μέχρι και σήμερα (έκδοση v11) είναι καταιγιστικές. Το ensemble μπορεί να πάρει τα δυνατά σημεία κάθε ομάδας αλγορίθμων και να τα εκμεταλλευτεί. Η διαφορά είναι ότι – δεν έχουμε μια έξοδο εξ'ορισμού- αλλά μπορούμε να έχουμε όσες και ο αριθμός των μοντέλων μας. Εδώ όμως επιλέγεται -είτε η μεγαλύτερη σε confidence – είτε αυτή που θεωρείται πιο αξιόπιστη -αφού επιβεβαιώνεται από την πλειοψηφία των μοντέλων.

1.3 Pipeline / Ensemble Learning

Η αλήθεια είναι ότι ένα σοβαρό σύστημα APNR σπάνια θα βασιστεί σε γραμμικό pipeline μονών μοντέλων. Καθώς η αναγνώριση πινακίδων οχημάτων είναι εξ' ορισμού ένα Multi-stage pipeline η χρήση Ensemble learning μπορεί να εκτοξεύσει την ακρίβεια εντοπισμού και ειδικά σε δύσκολες συνθήκες – όπως χαμηλός φωτισμός, περίεργες γωνίες λήψης, ταχύτητα εντοπισμού. Στο αρχικό στάδιο η εισαγωγή της εικόνας / ροής περνάει στο μοντέλο εντοπισμού οχημάτων / πινακίδων κυκλοφορίας οχημάτων. Σε σύστημα μονού μοντέλου – η έξοδος του σταδίου αυτού είναι «περιοχές ενδιαφέροντος» ή αλλιώς Regions of Interest (ROI's). Σε σύστημα που έχει Ensemble στο στάδιο του εντοπισμού όμως , αντί να χρησιμοποιηθεί ένα YOLOv8 ή ένα R-CNN μπορούν να τρέχουν δύο μοντέλα παράλληλα με την ίδια είσοδο. Σε αυτή την περίπτωση ,τα μοντέλα δίνουν bounding boxes χρησιμοποιώντας το λεγόμενο «Weighted Boxes Fusion (WBF)» αντί για το κλασικό «Non-Maximum Suppression (NMS)». Κατ' αυτόν τον τρόπο αν το ένα μοντέλο «χάσει» την πινακίδα του οχήματος λόγω μερικής σκίασης , το άλλο έχει αυξημένες πιθανότητες να τη βρει. Η έξοδος του πρώτου σταδίου περνάει στο κομμάτι του Optical Character Recognition (OCR). Εδώ ένα δεύτερο μοντέλο εκπαιδευμένο στην ανάγνωση χαρακτήρων προσπαθεί να «εξάγει» τους χαρακτήρες που αναγνωρίζει σε κάθε ROI που του δόθηκε ως όρισμα. Κι εδώ ακριβώς είναι που «λάμπει» η χρήση του ensemble. Η ταυτόχρονη χρήση διαφορετικών μοντέλων – διαφορετικών αρχιτεκτονικών μπορεί να δώσει και διαφορετικά αποτελέσματα με διαφορετικό πάντα confidence ανά μοντέλο. Σε ένα σύστημα με τρία μοντέλα, εάν δύο από τα τρία δώσουν έξοδο ότι ο χαρακτήρας που εντοπίστηκε είναι κλάσης μηδέν (0) και το τρίτο κλάσης όμικρον (O) τότε το σύστημα μπορεί να πάρει ως πιο σίγουρη έξοδο – αυτή της πλειοψηφίας. Ανάλογα με την υλοποίηση πάντα, μπορεί να συγκρίνει και τα confidence κάθε μοντέλου και με βάση αυτό να λάβει την απόφαση της εξόδου. Στο κομμάτι του OCR μπορεί να υπάρξει και το λεγόμενο Temporal Ensemble όταν εξετάζουμε ροή video. Το μοντέλο μας παίρνει διαδοχικές προβλέψεις από έναν **αριθμό συνεχόμενων** frames καθώς το όχημα πλησιάζει ή απομακρύνεται. Με αυτόν τον τρόπο μέσα από το ensemble των χρονικών στιγμών μπορούν να εξομαλυνθούν τα τυχαία λάθη – ο θόρυβος- που μπορεί να έχει εισαχθεί σε ένα frame [2].

Έτσι λοιπόν, βλέπουμε ότι η ακρίβεια των προβλέψεων αυξάνεται σημαντικά σε σχέση με ένα Pipeline μονών μοντέλων , αλλά: Όσο περισσότερα μοντέλα έχουμε στο σύστημα τόσο περισσότερο χρόνο επεξεργασίας θα ξοδέψουμε – με υπολογιστικό κόστος ομοίως πολλαπλάσιο. Συσκευές όπως το Raspberry Pi 3B+ της υλοποίησής μας Σχήμα 1.1 – στερούνται πόρων κι έτσι αναγκαζόμαστε να «εγκαταλείψουμε» το όνειρο μιας θωρακισμένης – δυνατής λύσης λόγω των περιορισμών του υλικού. Σε επόμενο κεφάλαιο θα δούμε την αρχιτεκτονική που ακολουθήθηκε στην προσέγγισή μας, με pipeline μονών μοντέλων.

Σχήμα 1.1 Πλατφόρμα προτυποποίησης Raspberry Pi 3B+

1.4 Δομή της Διπλωματικής Εργασίας

Στο εισαγωγικό αυτό κεφάλαιο -παρουσιάζονται γενικά τα θέματα που αγγίζει η εργασία. Στο δεύτερο κεφάλαιο γίνεται ανάλυση του υλικού (hardware) που απαιτείται για την υλοποίηση του APNR System ενώ στο τρίτο κεφάλαιο δίνεται έμφαση στο λογισμικό και την αρχιτεκτονική ενός προγράμματος. Κατόπιν , στο τέταρτο κεφάλαιο γίνεται μια πλήρης ανάλυση της δομής της εφαρμογής καθώς επίσης και των μεθόδων που υλοποιούνται. Στο πέμπτο κεφάλαιο προχωράμε στην εξαγωγή συμπερασμάτων και στα παραρτήματα Α, Β, Γ δίνονται αντίστοιχα – οδηγός υλοποίησης της εργασίας με αναλυτικά βήματα , χαρακτηριστικά των μοντέλων που χρησιμοποιήθηκαν και ο κώδικας της εφαρμογής.

1.5 Επίλογος

Στο εισαγωγικό αυτό κεφάλαιο είδαμε από την αρχή της ύπαρξης των οχημάτων – την ανάγκη ταυτοποίησης αυτών και τους νέους τρόπους αυτόματης αναγνώρισης των πινακίδων κυκλοφορίας με τη χρήση σύγχρονων συστημάτων , που ενσωματώνουν τεχνολογίες Υπολογιστικής Όρασής. Καθώς αναπτύσσεται η τεχνολογία , τα συστήματα αυτά μπορούν να βελτιωθούν , να απαιτούν λιγότερους υπολογιστικούς πόρους , να ενσωματώνονται σε edge devices χωρίς να χάνουν λειτουργικότητα και να αξιοποιηθούν στο μέγιστο βαθμό καθώς και το κόστος εξοπλισμού που μπορεί να τα φιλοξενήσει έχει αρχίσει να μειώνεται αισθητά. Βέβαια όπως είδαμε και στην τελευταία ενότητα, ένα σοβαρό σύστημα απαιτεί στις μέρες μας και σοβαρούς υπολογιστικούς πόρους , αλλά με την πρόοδο της τεχνολογίας φιλοδοξούμε ότι αυτό θα αλλάξει.

Κεφάλαιο 2ο: Προαπαιτούμενο υλικό υλοποίησης (hardware)

2.1 Εισαγωγή

Προκειμένου να μπορέσουμε να προχωρήσουμε στην δημιουργία αυτής της εφαρμογής θα χρειαστούμε την ίδια καθ' εαυτή πλατφόρμα προτυποποίησης. Η ανάγκη για αυξημένους πόρους κι εγκατάσταση ολόκληρων βιβλιοθηκών λογισμικού που θα αναλύσουμε σε μεταγενέστερο κεφάλαιο – σε συνδυασμό με το μικρότερο οικονομικό κόστος σε σχέση με άλλες πλατφόρμες πρωτοτυποποίησης όπως το BeagleBone , είναι ο λόγος που επιλέχθηκε το Raspberry Pi.

2.2 Οικοσύστημα Raspberry Pi

Το οικοσύστημα Raspberry Pi , είναι μια ολόκληρη οικογένεια με πολλές γενιές από πλατφόρμες προτυποποίησης. Με την αρχή να γίνεται - περίπου στο έτος 2006- στο Πανεπιστήμιο του Cambridge στο Ηνωμένο Βασίλειο, από συνεργάτες του που διαπίστωσαν ότι οι φοιτητές που εισάγονταν κάθε έτος στο Τμήμα Επιστήμης Υπολογιστών δεν διέθεταν τεχνικές δεξιότητες καθώς ήταν απλοί «καταναλωτές» των τεχνολογιών αντί να τις αναπτύξουν. Έτσι προσπάθησαν να δημιουργήσουν έναν υπολογιστή τόσο φθηνό που να μπορούν τα παιδιά να τον «πειράξουν» , ακόμη και να τον καταστρέψουν κάνοντας λάθη – χωρίς όμως να επιβαρύνουν τους οικογενειακούς προϋπολογισμούς με ένα σημαντικό κόστος. Στόχος τους ήταν το κόστος κάθε μονάδα να μην ξεπεράσει τα 25-35 δολάρια ώστε να εφοδιαστούν όσες περισσότερες οικογένειες γίνεται με αυτές τις πλατφόρμες.

Η πλατφόρμα ονομάστηκε «Raspberry» – καθώς η μόδα της εποχής ήταν τα υπολογιστικά συστήματα να παίρνουν ονόματα φρούτων όπως Apple, Tangerine, Apricot – και «Pi» καθώς αρχικά προοριζόταν για να «τρέχει» μόνο τη γλώσσα προγραμματισμού Python. Το 2009 ιδρύεται το Raspberry Pi Foundation – ένα φιλανθρωπικό ίδρυμα με σκοπό την προώθηση της μελέτης των υπολογιστών και της Πληροφορικής. Το 2012 κυκλοφόρησε το πρώτο μοντέλο του ιδρύματος, το Raspberry Pi 1 Model B. Ενώ οι προσδοκίες ήταν οι πωλήσεις να φτάσουν τα 10.000 τεμάχια σε όλη την ιστορία του, δεν επαληθεύθηκαν. Μέχρι σήμερα , έχουν ήδη πουληθεί πάνω από 50 εκατομμύρια συσκευές σε όλον τον πλανήτη, καθιστώντας το τον πιο επιτυχημένο υπολογιστή στην ιστορία- κατασκευασμένο στη Μεγάλη Βρετανία.

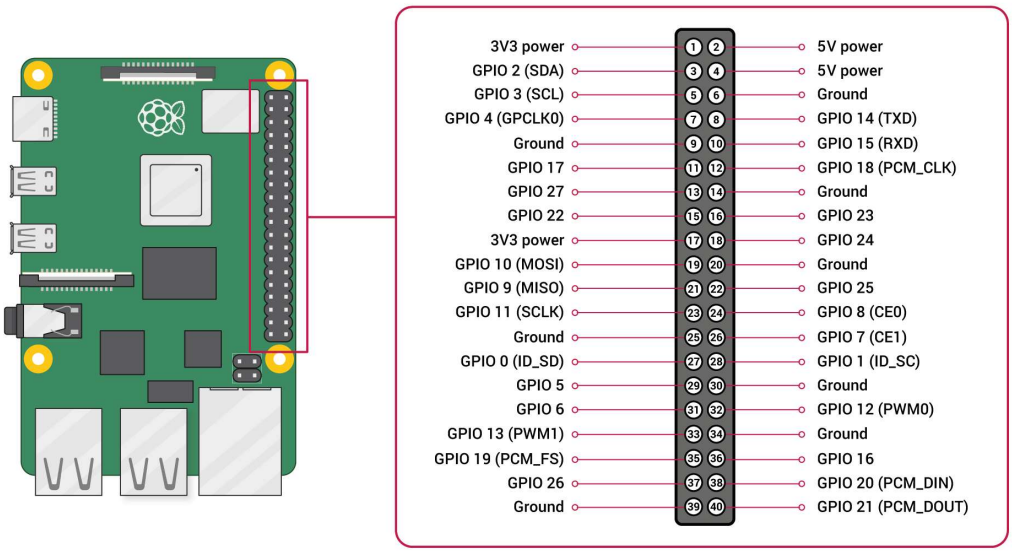
2.2.1 Βασικά Γενικά Τεχνικά Χαρακτηριστικά

Πρόκειται για έναν πλήρη υπολογιστή μονού κυκλώματος ή αλλιώς SBC (Single-Board Computer) σε μέγεθος λίγο μεγαλύτερο από πιστωτική κάρτα. Όλα τα επιμέρους στοιχεία του υπολογιστή βρίσκονται ενσωματωμένα στην ίδια μικρή πλακέτα – μνήμη RAM, θύρες USB , ελεγκτές δικτύου και έξοδος βίντεο. Εάν αγνοήσουμε το εξαιρετικά μικρό μέγεθος, δεν του υπολείπεται τίποτα έναντι ενός τυπικού επιτραπέζιου υπολογιστή (desktop) καθώς μπορεί να εκτελέσει επιτυχώς από απλές λειτουργίες όπως περιήγηση στο διαδίκτυο (internet browser) , επεξεργασία κειμένου και αναπαραγωγή βίντεο -ακόμη και υψηλής ανάλυσης – μέχρι και προγραμματισμό [3].

- Αρχιτεκτονική: Οι πλακέτες χρησιμοποιούν επεξεργαστές τεχνολογίας Advanced RISC Machine (ARM)- όμοιους με αυτούς που χρησιμοποιούνται στις συσκευές κινητής τηλεφωνίας, με κύριο κατασκευαστή αυτών την εταιρία Broadcom.
- Λειτουργικό Σύστημα: Τρέχουν κυρίως λειτουργικά συστήματα βασισμένα σε Linux. Το κύριο – επίσημο είναι το Raspberry Pi OS το οποίο βασίζεται σε Debian Linux.
- Αποθηκευτικός Χώρος : Καθώς στα περισσότερα μοντέλα – εξαιρούνται αυτά της τελευταίας γενιάς – δεν υπάρχει ενσωματωμένος δίσκος για την αποθήκευση δεδομένων, γίνεται χρήση

καρτών microSD. Σε αυτές φορτώνεται ολόκληρο το λειτουργικό σύστημα και τα αρχεία του χρήστη. Εκτός από το μέγεθός τους , οι κάρτες αυτές έχουν υψηλές ταχύτητες ανάγνωσης/ εγγραφής υποβοηθώντας σημαντικά την πλακέτα στην εκτέλεση των διεργασιών της. Στην τελευταία γενιά υπάρχει η επιλογή για Non-Volatile Memory Express (NVMe).

- Input / Output: Όπως σε κάθε άλλη πλακέτα για prototyping έτσι κι εδώ για να μπορεί να υπάρξει αμφίδρομη επικοινωνία του φυσικού κόσμου με αυτή, είναι απαραίτητη η ύπαρξη ακίδων (Pins) ώστε να μπορεί να δέχεται ερεθίσματα από τους αισθητήρες και να εκτελεί ενέργειες όπως εντολές σε κινητήρες /σερβομηχανισμούς και διακόπτες , LED κ.ά. Είναι τα λεγόμενα General Purpose Input/ Output (GPIO) Pins (Σχήμα 2.1).



Σχήμα 2.1 Raspberry Pi General/Purpose I/O

2.2.2 Οικογένειες Μοντέλων Raspberry Pi

Με την ευρεία αποδοχή του συστήματος από το κοινό και τον τεράστιο αριθμό πωλήσεων , το ίδρυμα προχώρησε στην περαιτέρω ανάπτυξή του. Για να μπορέσουν να καλυφθούν οι ανάγκες των διαφόρων κατηγοριών τεχνολογιών ήταν απαραίτητο να δημιουργηθούν διάφορες κατηγορίες – πιο «εξειδικευμένες»:

- **Mainline**. Η κύρια σειρά των συστημάτων του ιδρύματος. Είναι η σειρά που έχει τα πλήρη μοντέλα. Οι γενιές έχουν εξελιχθεί από τα Raspberry Pi 1 ,2 ,3 ,4 και στη στιγμή συγγραφής της παρούσης στη γενιά 5, το οποίο πλέον προσφέρει επιδόσεις κανονικού desktop υπολογιστή , ακόμη και με δυνατότητες χρήσης AI με την προσθήκη του κατάλληλου AI HAT+ 2. Με αυτό το έξτρα εξάρτημα υλικού προστίθεται η δυνατότητα εκτέλεσης μέχρι και 40 tera-operations per second (TOPS) – η χρήση έξτρα 8 GB RAM και ειδικά σχεδιασμένο για την εκτέλεση Μεγάλων Γλωσσικών Μοντέλων (LLM's).
- **Raspberry Pi Zero.** Πρόκειται για μια μικροσκοπική έκδοση του Raspberry Pi. Κυκλοφόρησε το 2015 με κόστος μικρότερο των 10 δολαρίων – ειδικά σχεδιασμένη εκδοχή που το καθιστά ιδανικό για οποιοδήποτε project που έχει περιορισμένο χώρο ή κατανάλωση ενέργειας.

- **Compute Module.** Είναι μια έκδοση με προσανατολισμό για χρήση στη βιομηχανία ή «βαριές» κατασκευές. Έχει τον πυρήνα του Raspberry Pi τοποθετημένο όμως σε μια πλακέτα παρόμοια με μνήμη RAM ενός υπολογιστή desktop ώστε να μπορεί να ενσωματωθεί εύκολα σε ειδικά κατασκευασμένες πλακέτες εταιριών. Πρόκειται για το βαρύ όπλο της σειράς.
- **Raspberry Pi Pico.** Είναι μια νέα σχετικά προσθήκη του ιδρύματος, εισαχθέν στη αγορά το 2021. Δεν είναι ένας υπολογιστής μικρού μεγέθους αλλά ένας μικροελεγκτής (microcontroller). Σκοπός του είναι το άνοιγμα στην αγορά των μικροελεγκτών όπως το Arduino και το ESP32. Χρησιμοποιείται σε αυτό ένας επεξεργαστής ειδικά σχεδιασμένος από το ίδρυμα ο RP2040.

Μετά από αυτή τη συνοπτική παρουσίαση των διαφορετικών κατηγοριών είναι καθαρά ορατό πλέον ότι για την υλοποίηση του project είναι απαραίτητη η χρήση μιας πλατφόρμας πρωτοτυποποίησης της κύριας οικογένειας Raspberry Pi. Επιλέχθηκε ένα μοντέλο της τρίτης γενιάς το Rasbperry Pi 3B+. Στον παρακάτω πίνακα (Πίνακας 2.1) παρουσιάζονται συνοπτικά τα κύρια τεχνικά χαρακτηριστικά του.

Πίνακας 2.1 Τεχνικά Χαρακτηριστικά του Raspberry Pi 3B+

Επεξεργαστής	BCM2837B0
Πυρήνας	ARM 64-bit @1.2 GHz
Μνήμη RAM	1GB
Πυρήνες	4
Συνδεσιμότητα	HDMI 4 × USB 2.0 standard 15-pin, 1.0 mm pitch, 16 mm width, CSI (camera) port standard 15-pin, 1.0 mm pitch, 16 mm width, DSI (display) port 3.5 mm AV jack 300 Mb/s Ethernet RJ45 with PoE support 2.4/5 GHz dual-band 802.11ac Wi-Fi (100 Mb/s) Bluetooth 4.2, Bluetooth Low Energy (BLE) microSD card slot micro-USB power

2.3 Υπολογιστική Όραση (Computer Vision)

2.3.1 Γενικά

Η Υπολογιστική Όραση είναι ένας ραγδαία αναπτυσσόμενος τομέας της Τεχνητής Νοημοσύνης. Είναι ο τομέας αυτός που δίνει την ικανότητα στα υπολογιστικά συστήματα να «βλέπουν» τον πραγματικό κόσμο και να εξάγουν πληροφορίες από την οπτική είσοδο που δέχονται.

2.3.2 Τρόπος λειτουργίας

Για τον άνθρωπο οι πληροφορίες που δέχεται από το οπτικό νεύρο μέσω των ματιών μεταφέρονται στο αντίστοιχο κέντρο επεξεργασίας του εγκεφάλου για γίνεται η ανάλυσή τους , εγγενώς , χωρίς να χρειάζεται κάποια επεξεργασία. Οι μηχανισμοί που εμπλέκονται έχουν διαμορφωθεί κατά την εξέλιξη του ανθρώπινου είδους. Μια μορφή «εκπαίδευσης» παρέχεται στον άνθρωπο με το πέρασμα του χρόνου, ξεκινώντας από τη στιγμή που ως βρέφη αρχίζουμε να αντιλαμβανόμαστε τον κόσμο με τα

μάτια μας. Αργότερα μπορούμε να ξεχωρίσουμε μορφές και αντικείμενα και να τις αποτυπώσουμε στη μνήμη μας, πχ η εικόνα των γονιών ενός παιδιού. Όταν το παιδί δει ξανά μια εικόνα με άτομα μπροστά του και ανάμεσά τους τους γονείς του, τότε μπορεί να τους ξεχωρίσει οπτικά – καθώς η πληροφορία ότι είναι «οι γονείς» του – οικεία σε αυτό άτομα – έχει αποθηκευτεί στις συνάψεις του εγκεφάλου του. Με αυτόν ακριβώς τον τρόπο, ξεπεράστηκε το πρόβλημα που είχαν οι προγραμματιστές σε όλη την υφήλιο μέχρι πρόσφατα – καθώς προσπαθούσαν να δώσουν «μοτίβα» που επαναλαμβάνονται σε έναν υπολογιστή για να αναγνωρίσει αντικείμενα.

2.3.3 Συνελικτικά Νευρωνικά Δίκτυα (Convolutional Neural Networks – CNN's) & Camera module

Σταδιακά εμφανίζονται τα ΣΝΔ στην εξέλιξη των υπολογιστών και ο προγραμματιστής αντιλαμβάνεται ότι αντί να καθίσει και να γράψει για κάθε αντικείμενο ή καλύτερα «κλάση» αντικειμένου ειδικές οδηγίες στο πως να αναγνωρίσει εάν μια είσοδος που δίνει στο σύστημα ανήκει στην κλάση «γάτα», τα πράγματα απλοποιούνται τρομερά. Η μέθοδος είναι εξαιρετικά απλούστερη: τροφοδοτούμε το σύστημα με εικόνες από αντικείμενα αυτής της κλάσης πχ εικόνες με γάτες, ώστε το σύστημα να πραγματοποιήσει μια ανάλυση και να αρχίσει να αναγνωρίζει μοτίβα που επαναλαμβάνονται στις φωτογραφίες που περιέχουν γάτες. Για να γίνει όμως κάτι τέτοιο πρέπει να περάσουμε την κάθε εικόνα από το ΣΝΔ μας και αυτό να προχωρήσει σε μια σειρά από βήματα επεξεργασίας κι εξαγωγής της χρήσιμης πληροφορίας περνώντας από ένα επίπεδο στο επόμενο. Εξετάζοντας την βασική αρχιτεκτονική ενός ΣΝΔ μπορούμε να καταλάβουμε καλύτερα τον τρόπο λειτουργίας του. Αρχικά, τα ΣΝΔ εκμεταλλεύονται τη χωρική συσχέτιση των pixels μιας εικόνας. Η εικόνα δεν αντιμετωπίζεται σαν ένα μονοδιάστατο διάνυσμα αλλά διατηρεί τη δομή της , είτε είναι δύο διαστάσεων ή και 3 εάν έχει και κανάλια χρώματος. Έτσι μια έγχρωμη εικόνα που περιέχει μια γάτα για παράδειγμα εισέρχεται ως είσοδος στο ΣΝΔ. Το πρώτο επίπεδο επεξεργασίας είναι και το πιο σημαντικό. Εδώ γίνεται η μαθηματική πράξη της Συνέλιξης (convolution) [4]. Το Δίκτυο έχει μικρούς πίνακες (τα λεγόμενα φίλτρα) με μικρές διαστάσεις οι οποίοι περιέχουν ως στοιχεία τους βάρη. Τα φίλτρα αυτά περνούν σταδιακά σαρώνοντας όλη την εικόνα τμηματικά. Κάθε φίλτρο που περνάει ανιχνεύει και κάποια συγκεκριμένα χαρακτηριστικά :τα πρώτα επίπεδα ανιχνεύουν απλά στοιχεία – στοιχειώδη όπως γωνίες και χρώματα , ενώ βαθύτερα επίπεδα συνδυάζουν αυτά τα απλά στοιχεία και προχωρούν στην αναγνώριση πολύπλοκων σχημάτων όπως πχ το περίγραμμα μιας γάτας. Το παράγωγο της συνέλιξης των φίλτρων με τα δεδομένα της εικόνας λέγεται χάρτης χαρακτηριστικών «Feature Map» καθώς «χαρτογραφήθηκε» ολόκληρη η εικόνα ως προς κάποια κοινά χαρακτηριστικά της [5]. Ακριβώς μετά από κάθε συνέλιξη πάνω στον χάρτη χαρακτηριστικών εφαρμόζουμε την συνάρτηση ενεργοποίησης για να πάρουμε τα βαθύτερα «κρυμμένα» δεδομένα , τις σχέσεις που βρίσκονται μέσα στο χάρτη. Η πιο διαδεδομένη συνάρτηση που χρησιμοποιείται είναι η ReLU (Rectified Linear Unit) , η οποία ορίζεται στη σχέση 2.1.

$$f(x) = \max(0, x). \qquad \text{[Σχέση 2.1]}$$

Η ReLU εξ' ορισμού για τιμές που είναι αρνητικές παίρνει επιστρέφει μηδενική τιμή επιτρέποντας έτσι στο νευρωνικό δίκτυο να εξάγει μη γραμμικές σχέσεις – πολύ πιο περίπλοκες. Συγκρίνοντάς την με άλλες συναρτήσεις ενεργοποίησης όπως είναι η Σιγμοειδής ή η Tanh, η ReLU επιτυγχάνει πολύ πιο γρήγορη εκπαίδευση.

Το αμέσως επόμενο στάδιο είναι αυτό της ομαδοποίησης ή αλλιώς γνωστό ως Pooling Layer. Εδώ «συμπιέζονται» οι χάρτες χαρακτηριστικών αφαιρώντας περιττή πληροφορία – μικραίνοντας τις διαστάσεις τους σε χώρο. Ο συνήθης τρόπος είναι ένα «πλαίσιο» μικρών διαστάσεων (πχ 2x2) να κάνει

ένα πέρασμα πάνω απ' όλο τον χάρτη και από κάθε περιοχή που διέρχεται κρατάει μόνο τη μέγιστη τιμή αυτής. Η μέθοδος ονομάζεται Max Pooling και βοηθάει σημαντικά στην μείωση του αριθμού των παραμέτρων που χρειάζεται να υπολογιστούν ενώ δεν αλλοιώνει τα χαρακτηριστικά που περιέχονται στην εικόνα, ακόμη κι αν αυτά μετατοπιστούν χωρικά [6].

Μετά από πολλά επίπεδα Συνέλιξης και ομαδοποίησης οι τελικοί χάρτες χαρακτηριστικών έχουν μικρύνει πολύ σε διάσταση αλλά έχουν μεγάλο βάθος. Τότε προχωρώντας στην «ισοπέδωσή» τους παράγεται ένα μονοδιάστατο διάνυσμα το οποίο το τροφοδοτούμε ως είσοδο σε πλήρως συνδεδεμένα νευρωνικά δίκτυα και στο τελευταίο αυτών με τη χρήση της συνάρτησης Softmax (Σχέση 2.2).

$$\mathrm{softmax}(x)_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}$$

Σχέση [2.2]

– ιδανική για προβλήματα ταξινόμησης – παίρνουμε και το τελικό διάνυσμα πιθανοτήτων για όλες τις κλάσεις [7].

Δεν πρέπει να ξεχνάμε όμως ότι το πρώτο και ίσως το πιο σημαντικό στάδιο για να πραγματοποιηθεί αυτή η λειτουργία από ένα σύστημα είναι να μπορέσει να «λάβει» το οπτικό ερέθισμα – το οπτικό σήμα. Και αυτό το κατορθώνουμε με τη χρήση ηλεκτρο-οπτικών αισθητήρων (E/O sensors) τις ευρέως διαδεδομένες στον κόσμο «κάμερες».

Έτσι δε θα μπορούσαμε να υλοποιήσουμε ποτέ ένα σύστημα APNR χωρίς να ενσωματώσουμε σε αυτό -σε κάποιο στάδιο εισαγωγής πληροφορίας- έναν τέτοιο αισθητήρα – μια κάμερα. Από το ίδιο το ίδρυμα δίνονται προδιαγραφές και εκδόσεις υλικού σε απόλυτη συμβατότητα με την ίδια την πλατφόρμα. Για λόγους απλότητας επιλέχθηκε το Camera Module 2 του οποίου τα τεχνικά χαρακτηριστικά φαίνονται στον παρακάτω σχήμα (Πίνακας 2.2 Camera module 2).

Πίνακας 2.2 Camera module 2

	Camera Module 1	Camera Module 2
Size	Around 25 × 24 × 9 mm	Around 25 × 24 × 9 mm
Weight	3 g	3 g
Still resolution	5 megapixels	8 megapixels
Video modes	1080p30, 720p60 and 640 × 480p60/90	1080p47, 1640 × 1232p41 and 640 × 480p206
Sensor	OmniVision OV5647	Sony IMX219
Sensor resolution	2592 × 1944 pixels	3280 × 2464 pixels
Sensor image area	3.76 × 2.74 mm	3.68 × 2.76 mm (4.6 mm diagonal)
Pixel size	1.4 µm × 1.4 µm	1.12 µm × 1.12 µm
Optical size	1/4"	1/4"
Focus	Fixed	Adjustable
Depth of field	Approx 1 m to ∞	Approx 10 cm to ∞
Focal length	3.60 mm ± 0.01	3.04 mm
Horizontal Field of View (FoV)	53.50 ± 0.13 degrees	62.2 degrees
Vertical Field of View (FoV)	41.41 ± 0.11 degrees	48.8 degrees
Focal ratio (F-Stop)	F2.9	F2.0
Maximum exposure time (seconds)	3.28	11.76
Lens Mount	N/A	N/A
NoIR version available?	Yes	Yes

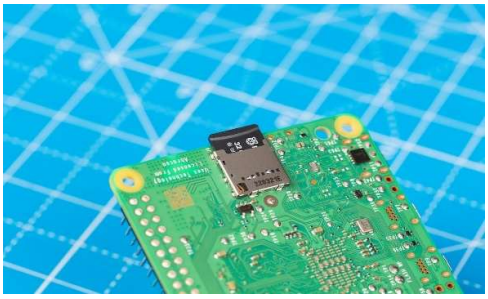
2.4 Απαραίτητα Περιφερειακά Υπολογιστικών Συστημάτων

Το Raspberry Pi ως υπολογιστικό σύστημα , φυσικά απαιτεί κάποιο περιφερειακό υλικό (hardware) ώστε να μπορέσει να λειτουργήσει και να εκμεταλλευτούμε στο έπακρο τις δυνατότητές του. Το υλικό αυτό μπορεί να θεωρηθεί αυτονόητο, πλην όμως για λόγους πληρότητας αυτής της υλοποίησης θα πρέπει να αναφερθεί – έστω και συνοπτικά όπως εδώ:

- Τροφοδοσία από παροχή δικτύου: από τα τεχνικά χαρακτηριστικά (Σχήμα 2.2) βλέπουμε ότι για την έκδοσή μας είναι απαραίτητη η χρήση ενός τροφοδοτικού τάσης εξόδου 5.1V και ρεύματος τουλάχιστον 2.5A με ακροδέκτη τύπου micro USB.
- Αποθηκευτικό μέσο για τη λειτουργία της πλακέτας: κάρτα microSD – σχετικά γρήγορη σε ταχύτητα ,ιδανικά σύμφωνα με τα απαιτούμενα χαρακτηριστικά- κλάση ταχύτητας : C10, U3, V30 , A2. (Σχήμα 2.3)
- Πληκτρολόγιο: για την εισαγωγή δεδομένων - απαιτείται χρήση πληκτρολογίου μέσω μιας από τις διαθέσιμες θύρες USB (Σχήμα 2.4)
- Ποντίκι: ομοίως με το πληκτρολόγιο – ενώ η πλατφόρμα μπορεί να λειτουργήσει και μόνο με το πληκτρολόγιο – για τη χρήση της εφαρμογής της υλοποίησής μας είναι απαραίτητη η χρήση ποντικιού μέσω θύρας USB (Σχήμα 2.4)
- Οθόνη: Η πλατφόρμα μας μπορεί να λειτουργήσει και σε headless mode – χωρίς να έχουμε συνδέσει κάποια συσκευή εξόδου. Για την δική μας υλοποίηση όμως – καθώς θα πρέπει το σύστημα εν δυνάμει να εγκατασταθεί σε ένα όχημα και να ενημερώνει τους χρήστες του για τυχόν εντοπισμό οχημάτων που έχουν κλαπεί ή αναζητούνται με οπτική ανάδραση τότε πρέπει να συνδεθεί μια οθόνη μέσω της ενσωματωμένης εξόδου HDMI που διαθέτει η πλακέτα(Σχήμα 2.5).

Product	Recommended PSU current capacity	Maximum total USB peripheral current draw	Typical bare-board active current consumption
Raspberry Pi 3 Model B+	2.5A	1.2A	500mA

Σχήμα 2.2 Τροφοδοσία Raspberry Pi 3B+.



Σχήμα 2.3 Αποθηκευτικό μέσο- κάρτα microSD.

Σχήμα 2.4 Πληκτρολόγιο & ποντίκι Raspberry Pi

Σχήμα 2.5 Οθόνη για Raspberry Pi

2.5 Επιπλέον υλικό για την υλοποίηση

Καθώς στην υλοποίηση έχουμε συμπεριλάβει και ηχητική ανάδραση για την ειδοποίηση του χρήστη αντί να χρησιμοποιήσουμε το ενσωματωμένο βοηθητικό jack των 3.5 χιλιοστών , για πρακτικούς λόγους επιλέχθηκε ένα passive buzzer module (Σχήμα 2.6). Η βοηθητική αυτή πλακέτα διαθέτει 3 ακροδέκτες (pins) για τη σύνδεση σε αντίστοιχους ακροδέκτες της πλατφόρμας μας , ώστε με την κατάλληλη διαχείριση μέσω κώδικα να παράγουμε τους ήχους που χρειαζόμαστε. Για να «συνεργαστούν» αποτελεσματικά αυτά μεταξύ τους πρέπει να συνδεθεί το module σε έναν ακροδέκτη της πλατφόρμας που υποστηρίζει Pulse Width Modulation (PWM) (Σχήμα 2.1). Η σύνδεση αυτή δεν απαιτεί κάποιο επιπλέον εξάρτημα για τον περιορισμό του ηλεκτρικού ρεύματος καθώς η βοηθητική πλακέτα διαθέτει ήδη αντιστάσεις ενσωματωμένες για το λόγο αυτό. Επιπλέον, θα χρειαστούν καλώδια dupont (Σχήμα 2.7) για τη σύνδεση ανάμεσα στα pins του Raspberry Pi και του passive buzzer module με αμφότερα θηλυκά άκρα (F-F)

Σχήμα 2.6 Passive Buzzer Module

Σχήμα 2.7 Καλώδια σύνδεσης Dupont

Τέλος, καθώς η πλατφόρμα μας θα χρειαστεί να δουλέψει όσο πιο αποδοτικά γίνεται προκειμένου να μπορέσουμε να εκτελούμε συνεχώς τα βήματα του pipeline μας. Για να το καταφέρουμε αυτό, ιδανικά

προσθέτουμε ένα κιτ για παθητική ψύξη των «ευαίσθητων» θερμικά εξαρτημάτων όπως είναι το κομμάτι του επεξεργαστή, αλλά και ένα enclosure , μια θήκη προστασίας με ενεργή ψύξη (με ενσωματωμένο ανεμιστήρα) για να «δροσίζουμε» όσο γίνεται περισσότερο το σύστημα (Σχήμα 2.8). Σε περίπτωση αύξησης της θερμοκρασίας πάνω από ένα κατώφλι το υλικό μας θα ζοριστεί με συνέπεια τα «κολλήματα» κατά τη φάση του OCR detection που είναι και το πιο απαιτητικό τμήμα. Στην ακραία υπερθέρμανση , η πλακέτα -με σκανδαλισμό από θερμικό αισθητήρα – απενεργοποιείται προκειμένου να αποφευχθεί μόνιμη βλάβη.

Σχήμα 2.8 Κιτ προστασίας και ενεργής ψύξης της πλατφόρμας

2.6 Επίλογος

Στο κεφάλαιο αυτό έγινε μια σύντομη παρουσίαση του υλικού που εμπλέκεται στην υλοποίηση του συστήματος APNR. Η εν δυνάμει ενσωμάτωση του συστήματος στα συστήματα της Ελληνικής Αστυνομίας θα απαιτούσε επιπλέον ένα τροφοδοτικό απ’ ευθείας από την τροφοδοσία του οχήματος με έξοδο 5V – όπως για παράδειγμα αυτό του σχήματος Σχήμα 2.9.Με αυτόν τον μετατροπέα έχουμε σταθερά 5V ως έξοδο και με ένα καλώδιο USB M σε micro USB M μπορούμε να τροφοδοτήσουμε τη συσκευή μας. Δεν θα επεκταθούμε όμως περισσότερο σε αυτό καθώς αφορά τις «εν δυνάμει» βελτιώσεις του συστήματος στις οποίες θα αναφερθούμε στο τελευταίο κεφάλαιο της εργασίας αυτής.

Σχήμα 2.9 Step-down-transformer

Κεφάλαιο 3ο: Λογισμικό της εφαρμογής

3.1 Εισαγωγή

Προσπερνώντας όλα τα βήματα για την αρχικοποίηση και τη ρύθμιση της πλακέτας Raspberry Pi 3B+ καθώς αυτά περιγράφονται αναλυτικά στο Παράρτημα Α προχωράμε στο κομμάτι του λογισμικού της εφαρμογής μας.

3.2 Σχεδιάζοντας την υλοποίηση

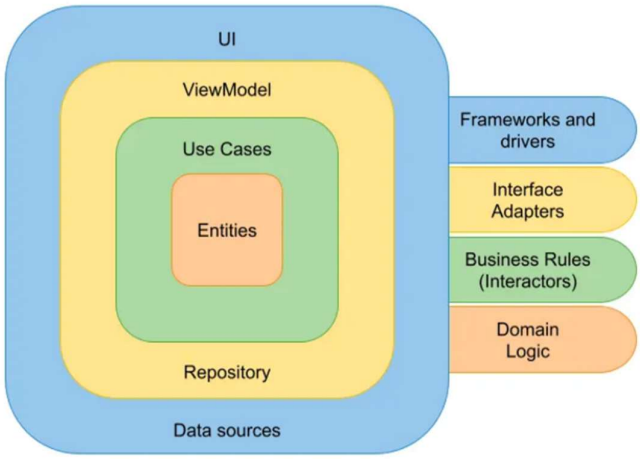
Σε κάθε εφαρμογή που δημιουργείται είτε από μεμονωμένους προγραμματιστές είτε από ομάδες αυτών δαπανώνται πόροι. Από τη φάση της «καταγραφής των αναγκών» μέχρι την τελική παράδοση του έργου παρεμβάλλονται πολλά στάδια. Το πιο σημαντικό όμως – είναι η επιλογή της κατάλληλης αρχιτεκτονικής που θα ακολουθηθεί για την υλοποίηση. Όσο περισσότερο χρόνο ξοδέψει η ομάδα στον σχεδιασμό τόσο καλύτερα αποτελέσματα θα έχει στο τέλος.

3.2.1 Αρχιτεκτονική του προγράμματος & SoC

Η φύση της εφαρμογής, η ανάγνωση των πινακίδων κυκλοφορίας των οχημάτων που διέρχονται από το οπτικό πεδίο του ηλεκτρο-οπτικού αισθητήρα της πλακέτας μας και η αναζήτησή τους σε βάση δεδομένων είναι ένα πρόβλημα εξ' ορισμού. Μια τέτοια εφαρμογή θα μπορούσε να «λειτουργήσει» με ένα ενιαίο αρχείο κώδικα γραμμένο στη γλώσσα Python. Αυτό όμως θα είχε ένα τελείως ερασιτεχνικό αποτέλεσμα με ένα «spaghetti code» όπως λέγεται στους κύκλους των προγραμματιστών και στις εταιρίες πληροφορικής. Η δημιουργία μιας εφαρμογής σε αυστηρά επαγγελματικά πρότυπα είναι αδιαπραγμάτευτη καθώς θα μπορούσε να χρησιμοποιηθεί από Οργανισμούς και φορείς όπως η Αστυνομία. Σε μια τέτοια περίπτωση – θα πρέπει ο χρήστης της εφαρμογής να μην έχει πρόσβαση στα δεδομένα. Επίσης, θα πρέπει να διασφαλιστεί η βάση δεδομένων και η πρόσβαση σε αυτή , καθώς τα δεδομένα που περιέχει είναι «ευαίσθητα».

Με βάση τα προηγούμενα, μία είναι η λύση: πρέπει να οδηγηθούμε σε μια «καθαρή Αρχιτεκτονική» που θα εφαρμόσει πιστά την αρχή του Διαχωρισμού των Αρμοδιοτήτων (Separation of Concerns- SoC). Σύμφωνα με την αρχή αυτή – το κάθε τμήμα της εφαρμογής μας πρέπει να είναι τελείως «χαζό» με αποκορύφωμα στη χαζομάρα το ίδιο το user interface (UI) , την διεπαφή χρήστη. [6]Χωρίζουμε την εφαρμογή μας σε τμήματα ή στρώσεις (modules ή layers) τα οποία έχουν το καθένα ένα συγκεκριμένο κομμάτι της λειτουργικότητας (concern).

Το UI , πρέπει να ξέρει μόνο ποια είναι τα στοιχεία του, πως «ζωγραφίζονται» στον καμβά της εφαρμογής – ανεξάρτητα της τεχνολογίας και του framework που μπορεί να χρησιμοποιηθεί – και το απαραίτητο , ποιες συναρτήσεις θα κληθούν και από που, από ποια αρχεία – ποιο προορισμό – χωρίς τίποτα παραπάνω.Σχήμα 3.1 Separation of Concern Περιορίζοντας τις πληροφορίες που έχει πρόσβαση η διεπαφή χρήστη χωρίς να είναι απαραίτητο , τότε περιορίζουμε τα «τρωτά» σημεία της ίδιας της εφαρμογής μας.



Σχήμα 3.1 Separation of Concern

Πλεονεκτήματα του SoC:

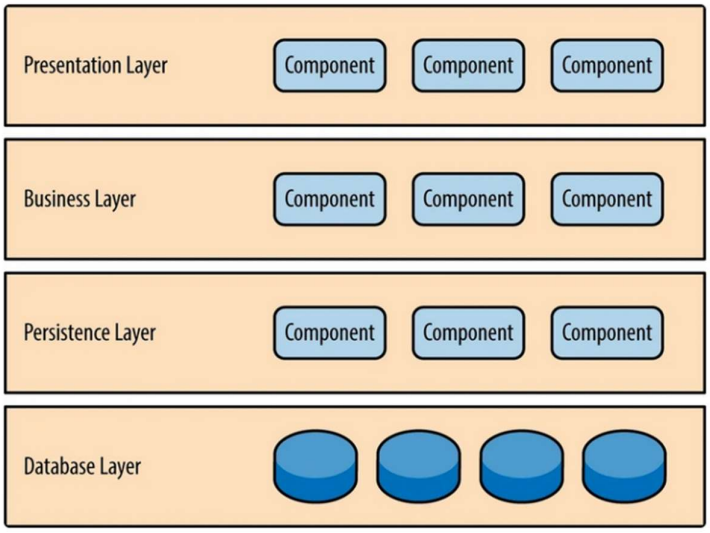
- Η συντήρηση του κώδικα και οι τυχόν αλλαγές που μπορεί να χρειαστούν παρακάτω στη χρήση της εφαρμογής είναι εύκολη. Εφόσον χρειαστεί να αλλάξει ένα τμήμα του συστήματος – πχ από PostgreSQL να περάσουμε σε MySQL ή κάτι παρόμοιο, θα γίνει μόνο μια αλλαγή στο τμήμα των δεδομένων μας, ή στο επίπεδο (Data Access Layer – DAL) όπως ονομάζεται. Δεν γίνεται καμία αλλαγή στον υπόλοιπο κώδικα , ούτε στο UI. Μια εφαρμογή που δούλευε τέλεια συνεχίζει να δουλεύει χωρίς να «σπάσουμε» κάτι σε αυτήν κατά τις αλλαγές. Η λογική παραμένει η ίδια και μεταβάλλουμε μόνο το τμήμα που απαιτείται.
- Επαναχρησιμοποίηση κώδικα. Ένα τμήμα του κώδικα που είναι υπεύθυνο για μια εργασία μπορεί να χρησιμοποιηθεί από διαφορετικούς μηχανισμούς που θα το καλέσουν. Μια υπηρεσία που καλεί μια συνάρτηση τελείως απομονωμένη μπορεί να κληθεί τόσο από μια έκδοση της εφαρμογής μας για κινητές συσκευές ή από μια web έκδοση. Το back-end θα δουλέψει με τον ίδιο τρόπο (agnostic execution).
- Ευκολότερη ανάπτυξη του λογισμικού. Ιδιαίτερα σε εταιρίες που εργάζονται πάνω σε enterprise λύσεις κι εφαρμογές, είναι απαραίτητη η αποτελεσματική συνεργασία πολλών ατόμων. Χωρίζοντας ολόκληρη την εφαρμογή σε τμήματα κάθε ομάδα προγραμματιστών μπορεί να εργάζεται στο back-end , ή στο front-end ανεμπόδιστα. Είναι μια πρακτική που ακολουθείται πιστά στις εταιρίες ανάπτυξης λογισμικού.
- Απλοποίηση ελέγχων. Πριν βγει στην παραγωγή μια εφαρμογή , πρέπει υποχρεωτικά να περάσει από ελέγχους. Χωρίζοντας όλη την εφαρμογή σε modules μπορούμε να κάνουμε πιο «συγκεντρωμένα» τεστ σε κάθε τμήμα του κώδικα , χωρίς να χρειάζεται να φορτώνουμε ολόκληρη την εφαρμογή.

Τα κοινά αρχιτεκτονικά πρότυπα που βασίζονται στη λογική διαχωρισμού των αρμοδιοτήτων παρουσιάζονται συνοπτικά παρακάτω.

3.2.2 Αρχιτεκτονική Πολλών Επιπέδων (Layered Architecture)

Πρόκειται για τον πιο διαδεδομένο τρόπο εφαρμογής του SoC στις εταιρικές λύσεις. Ολόκληρη η εφαρμογή χωρίζεται σε τρία κύρια στρώματα (layers) Σχήμα 3.2 Layered Architecture:

- UI: Αφορά μόνο ότι βλέπει και «αγγίζει» ο χρήστης της εφαρμογής. Τίποτε παραπάνω. Το πιο «χαζό» layer της εφαρμογής – δεν ξέρει τίποτε, δεν κάνει τίποτε. Απλά μεσολαβεί ανάμεσα στον χρήστη και στο κύριο στρώμα της εφαρμογής καλώντας συναρτήσεις.
- Business Logic Layer: Πρόκειται για το «έξυπνο» τμήμα της εφαρμογής. Εδώ «ζει» όλη η λογική του προγράμματός μας , όλες οι συναρτήσεις και οι υπηρεσίες. Οι περιορισμοί της εφαρμογής και οι έλεγχοι γίνονται όλοι εδώ.
- Data Access Layer: Η στρώση του προγράμματός μας που έχει μόνο έναν σκοπό και τίποτα περισσότερο. Ασχολείται με την αποθήκευση και ανάκτηση δεδομένων από τις βάσεις δεδομένων. (Όλο το CRUD – Create,Read,Update,Delete- και τα queries που χρησιμοποιούνται από την εφαρμογή θα περάσουν από εδώ σε επίπεδο συναλλαγής με βάση).

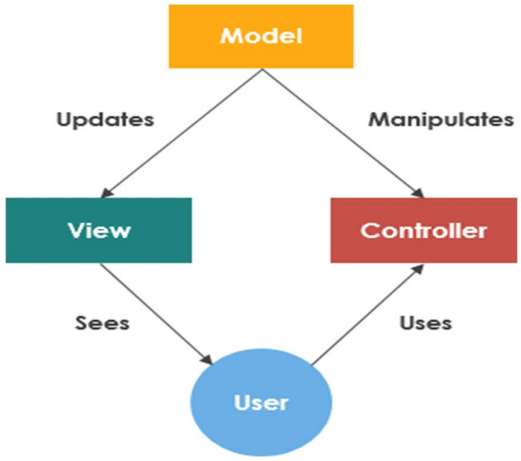


Σχήμα 3.2 Layered Architecture

3.2.3 Model – View – Controller

Επίσης γνωστό και ως MCV (Model-Controller-View) Σχήμα 3.3 Model-View-Controller, αποτελεί το πιο διάσημο πρότυπο για το σχεδιασμό UI. Εδώ χωρίζεται ομοίως η εφαρμογή σε τρία κύρια μέρη:

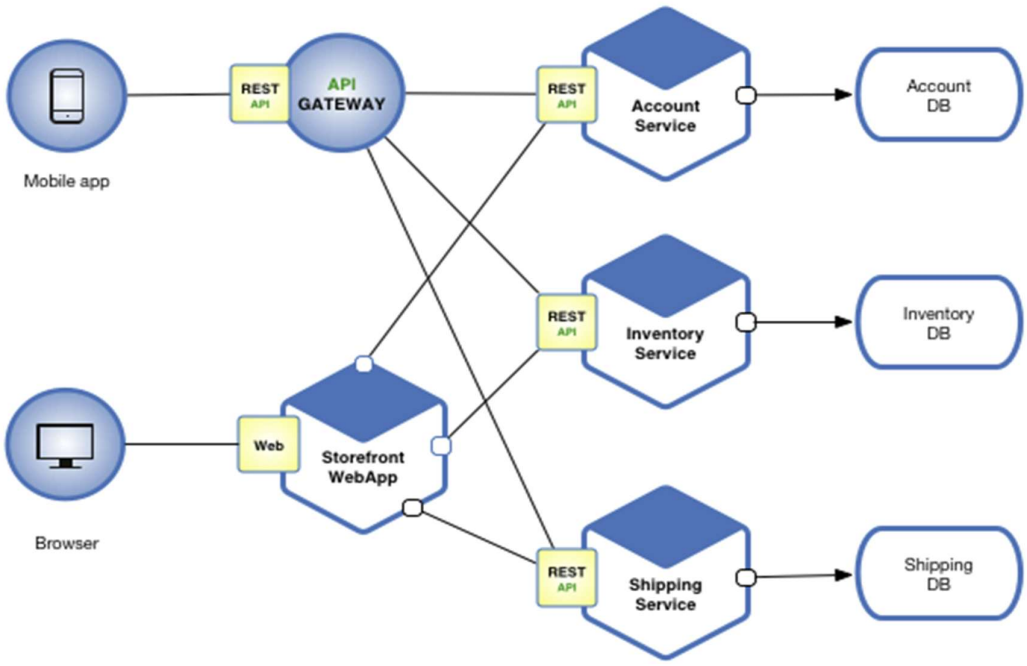
- Model (το «Μοντέλο»): Εδώ περιέχονται όλα τα δεδομένα, καθώς και όλοι οι κανόνες που πρέπει να τηρούνται σε σχέση με αυτά.
- View (η «Προβολή»): Είναι ολόκληρο το UI. Δεν εσωκλείει καθόλου business logic , είναι απλά ο μηχανισμός που ζωγραφίζει τον καμβά του χρήστη στην οθόνη.
- Controller (ο «Ελεγκτής): Όλα τα υπόλοιπα. Αποτελεί το «ενδιάμεσο» επίπεδο. Οποιαδήποτε ενέργεια κάνει ο χρήστης θα περάσει πρώτα από εδώ , θα ενημερώσει κατάλληλα το model και για οποιαδήποτε αλλαγή απαιτείται θα ζητήσει από το view να την «ζωγραφίσει».



Σχήμα 3.3 Model-View-Controller

3.2.4 Microservices

Πρόκειται για μια διαφορετική προσέγγιση καθώς ο Διαχωρισμός των Αρμοδιοτήτων δεν εφαρμόζεται με την τμηματοποίηση / στρωματοποίηση της εφαρμογής , αλλά με τον καταμερισμό της εφαρμογής σε πολλές μικρές , ανεξάρτητες μεταξύ τους μικρο-εφαρμογές(Σχήμα 3.4 Microservices. Με αυτόν τον τρόπο «τμηματοποιείται» προγραμματιστικά το σύστημα με «τμήματα» αρμοδιοτήτων. Δημιουργούνται επιμέρους υπηρεσίες, από τις οποίες η καθεμιά έχει δικό της καθαρό αντικείμενο. [9]



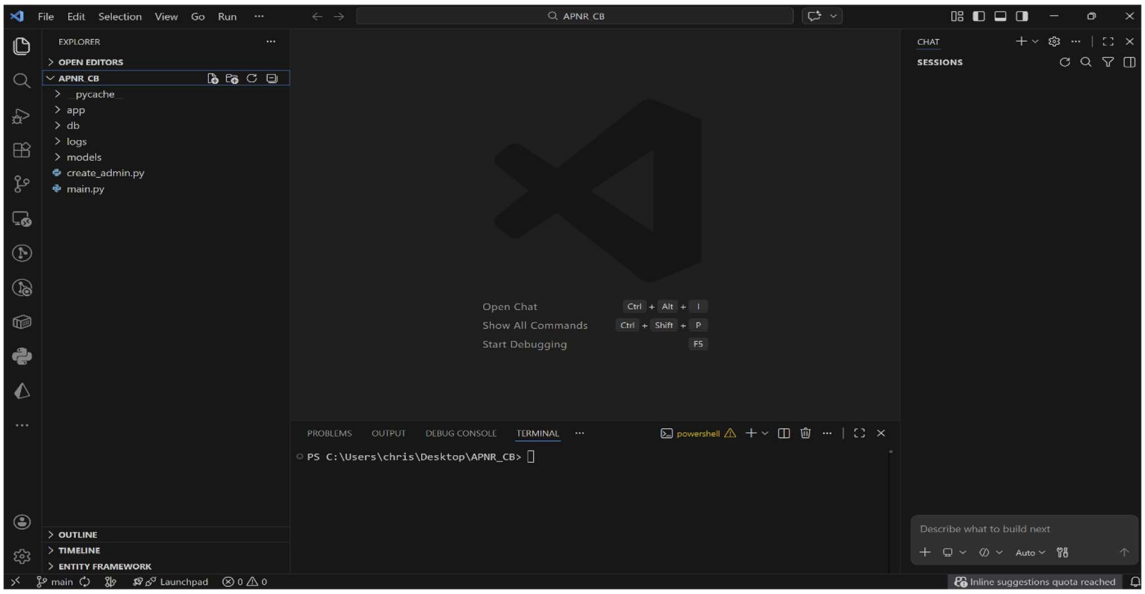
Σχήμα 3.4 Microservices

3.3 Περιβάλλον ανάπτυξης Visual Studio Code

Για να μπορέσουμε να δημιουργήσουμε την εφαρμογή μας – απαραίτητο συστατικό είναι ένα περιβάλλον ανάπτυξης λογισμικού. Είτε αυτό είναι ένας απλός επεξεργαστής κειμένου όπως το Notepad++ , είτε το πιο εξεζητημένο περιβάλλον , όλα καταλήγουν στο ίδιο έργο: την παραγωγή κώδικα. Για την ανάπτυξη του APNR System μας επιλέχθηκε ο text editor Visual Studio Code (Σχήμα 3.5) ο οποίος είναι κάτι διαφορετικό.

Σχήμα 3.5 Visual Studio Code logo

Δεν πρόκειται για ένα «βαρύ» πλήρες IDE (Integrated Development Environment) αλλά για έναν ιδιαίτερα ελαφρύ text editor ανοιχτού κώδικα με τρομερές επιλογές για την προσθήκη επεκτάσεων σε αυτών. Πρόκειται για δημιούργημα της εταιρίας Microsoft πλην όμως είναι ένα από τα έργα ΕΛΑΚ της γνωστής εταιρίας. Υποστηρίζεται από μεγάλη κοινότητα προγραμματιστών οι οποίοι συνεισφέρουν προσθέτοντας επεκτάσεις(extensions). Διαθέτει εργαλεία για την έξυπνη συμπλήρωση κώδικα με τους τύπους των μεταβλητών ή τους ορισμούς μεθόδων του χρήστη. Έχει ενσωματωμένους βοηθούς αποσφαλμάτωσης (debugger) για πολλές γλώσσες προγραμματισμού και πλέον στην εποχή της Τεχνητής Νοημοσύνης που διανύουμε διαθέτει πλήθος επιλογών για ψηφιακούς βοηθούς στην ανάπτυξη κώδικα. [[10]Το γεγονός ότι είναι πλήρως παραμετροποιήσιμο σύμφωνα με τις επιθυμίες του κάθε χρήστη είναι και ο λόγος που είναι το πιο δημοφιλές περιβάλλον ανάπτυξης κώδικα στον κόσμο(Σχήμα 3.6 Visual Studio Code UI).



Σχήμα 3.6 Visual Studio Code UI

3.4 Μηχανισμοί Προστασίας Δεδομένων Εφαρμογής

Καθώς ο σκοπός μας δεν είναι απλά η δημιουργία μιας εφαρμογής που να «τρέχει» αλλά μιας σοβαρής -επαγγελματικής κλάσης- εφαρμογής που να δουλεύει άριστα προστατεύοντας ταυτόχρονα τα δεδομένα της πρέπει να ενσωματώσουμε όσες περισσότερες δικλείδες ασφαλείας είναι δυνατόν. Αρχικά, όπως αναφέραμε και νωρίτερα στην ενότητα 3.2 η ίδια η αρχιτεκτονική μας με βάση το SoC είναι ένα μεγάλο

βήμα προς αυτή την κατεύθυνση. Με την ενσωμάτωση όμως όλων των παρακάτω μεθόδων ανεβάζουμε την εφαρμογή στην κορυφαία κατηγορία επαγγελματικών λύσεων.

3.4.1 Identity and Access Management (IAM)

Ολόκληρη η φιλοσοφία της αναγνώρισης ενός χρήστη που προσπαθεί να εκτελέσει την εφαρμογή μέχρι και την εκτέλεσή της τελικά – εσωκλείεται σε ένα «πλαίσιο» (framework) το οποίο μεταφράζεται ως «Διαχείριση Ταυτότητας και Πρόσβασης». Στις Ηνωμένες Πολιτείες Αμερικής το Εθνικό Ίδρυμα Προτύπων και Τεχνολογίας (National Institute of Standards and Technology -NIST) εκδίδει όποτε κρίνεται αναγκαίο «οδηγίες προς ναυτιλομένους» θα λέγαμε για τους προγραμματιστές που πλέουν σε θάλασσες κώδικα. Στην τελευταία μάλιστα που εκδόθηκε τον Ιούλιο του 2025 γίνεται πλήρης ανάλυση για τους τρόπους αναγνώρισης , διαχείρισης και επαλήθευσης της ψηφιακής ταυτότητας ενός χρήστη. Σύμφωνα με το NIST ο παγκόσμιος «Χρυσός κανόνας» γύρω από την πρόσβαση έχει τα εξής πολύ κρίσιμα σημεία [11] :

1. Διαχωρισμός των επιπέδων διασφάλισης.
- IAL (Identity Assurance Level) : Είναι ο «βαθμός σιγουριάς» για την πραγματική ταυτότητα του χρήστη. Είναι η ερώτηση που θα κάνουμε στον εαυτό μας «πόσο σίγουρος είμαι ότι ο χρήστης που θέλει να μπει στην εφαρμογή μου είναι αυτός που λέει;»
- AAL (Authenticator Assurance Level): Πρόκειται για τον «μετρητή ισχύος» της μεθόδου σύνδεσης στην εφαρμογή (πχ απλό password vs hashed_password)
- FAL (Federation Assurance Level): Η ερώτηση που κάνουμε στον εαυτό μας «πόσο αξιόπιστα μεταφέρεται η ταυτότητα του χρήστη;»
2. Η ακύρωση της περιοδικής αλλαγής των κωδικών των χρηστών.
3. Η υποχρεωτική χρήση πολλαπλών παραγόντων αυθεντικοποίησης (Multi-Factor Authentication -MFA)
4. Η διαχείριση συνεδριών και επαναυθεντικοποίησης.
5. Ο κύκλος ζωής της συνεδρίας και η τυχόν ανάκληση token.

Στο πλαίσιο αυτό εντάσσονται και οι 4 επόμενες υποενότητες. Ενώ είναι ουσιαστικά «κομμάτια» όλης της φιλοσοφίας του IAM στην περίπτωσή μας θα τα δούμε ξεχωριστά για να γίνει πιο εύκολα κατανοητό το κάθε τμήμα και η υλοποίησή του στην εφαρμογή μας. Είναι απαραίτητο να σημειωθεί εδώ , ότι στο APNR System ΔΕΝ έχουν υλοποιηθεί όλα τα κρίσιμα σημεία του χρυσού κανόνα του NIST , καθώς η υλοποίηση θα «ξέφευγε» υπερβολικά. Είναι όμως κάτι που μπορεί να συμπεριληφθεί άνετα στο τελευταίο κεφάλαιο αυτής της εργασίας με τις μελλοντικές βελτιώσεις.

3.4.2 Authentication

Η πιστοποίηση (authentication) είναι το πρώτο και το πιο κρίσιμο στάδιο στην ασφάλεια πληροφοριακών συστημάτων. Το να επιτρέψουμε σε ένα χρήστη την εκτέλεση της εφαρμογής ή όχι , το αποφασίζουμε ξεκινώντας από μια φόρμα πιστοποίησης της ταυτότητάς του (Log in). Γενικά, η διαδικασία του authentication είναι η διαδικασία της επιβεβαίωσης της ψηφιακής ταυτότητας ενός χρήστη, μιας συσκευής ή και μιας άλλης εφαρμογής που προσπαθεί να συνδεθεί στη δική μας.

Το πόσο ισχυρή είναι η πιστοποίηση του χρήστη ή της εφαρμογής που προσπαθεί να συνδεθεί εξαρτάται από κάποιον ή κάποιους σε περίπτωση που συνδυάσουμε περισσότερους από έναν παράγοντες:

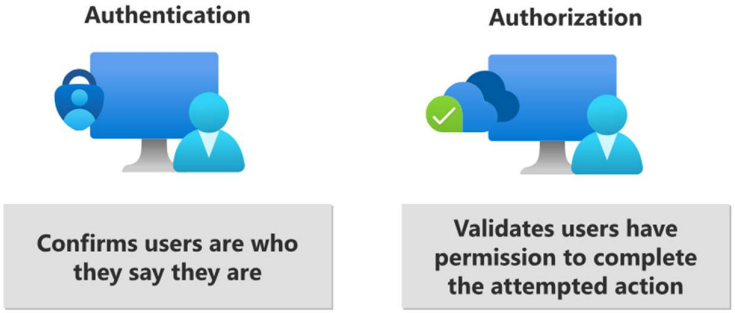
1. Κάτι που γνωρίζει ο αυθεντικός – πιστοποιημένος χρήστης (Γνώση): Κωδικοί, PINs, τυχόν απαντήσεις σε ερωτήσεις ασφαλείας που μπορεί να του κάνουμε. Σε σχέση με τους υπόλοιπους είναι ο πιο ευάλωτος παράγοντας.

2. Κάτι που κατέχει ο αυθεντικός – πιστοποιημένος χρήστης (Κατοχή): μια κινητή συσκευή που δέχεται push notifications ή συσκευή pin generator που του έχει δώσει η τράπεζα για παράδειγμα.
3. Κάτι που είναι ο αυθεντικός – πιστοποιημένος χρήστης (Βιομετρική ταυτοποίηση): αναγνώριση προσώπου , δαχτυλικά αποτυπώματα κλπ.

Πλέον για να πιστοποιηθεί ένας χρήστης στις περισσότερες εφαρμογές επαγγελματικού επιπέδου εμφανίζεται ένας συνδυασμός τουλάχιστον δύο **διαφορετικών** κατηγοριών από τις ανωτέρω (Multi Factor Authentication-MFA). Η χρήση μόνο μιας κατηγορίας στις μέρες μας δεν είναι ασφαλής για μια τέτοια εφαρμογή (Single Factor Authentication – SFA). Εναλλακτικά έχουν εμφανιστεί και είναι πλέον πολύ διαδεδομένοι -μηχανισμοί όπως το One Time Password (OTP) το οποίο αποστέλλεται στον χρήστη για περαιτέρω ασφάλεια όπως επίσης και η χρήση «passkeys» - κρυπτογραφικών κλειδιών που αποθηκεύονται στις συσκευές του χρήστη και όταν επαληθεύσει την ταυτότητά του με βιομετρική ταυτοποίηση (ξεκλείδωμα τηλεφώνου με αποτύπωμα ή αναγνώριση προσώπου) τότε ολοκληρώνεται η υπογραφή του αιτήματος σύνδεσης.

Η πιστοποίηση του χρήστη όμως δεν είναι μόνο η διαδικασία της σύνδεσης. Περιλαμβάνει και τη διαδικασία της επαλήθευσης του χρήστη πριν του δωθούν κωδικοί πρόσβασης, τις διαδικασίες που πρέπει να γίνουν στην περίπτωση που χαθεί η πρόσβαση στο MFA -πχ έχασε τη συσκευή που λάμβανε τα μηνύματα OTP- όπως επίσης – και είναι **πολύ** σημαντικό – τη διαδικασία ακύρωσης οποιασδήποτε πρόσβασης στην εφαρμογή μας σε περίπτωση που υπάρξει παραβίαση λογαριασμού [12].

Συμπερασματικά: Η αρχή και το κυριότερο στάδιο για την είσοδο στην εφαρμογή μας είναι η πιστοποίηση του χρήστη. Εάν ο έλεγχος αυτός αποτύχει και αποκτήσει πρόσβαση χρήστης που έχει «κρυφτεί» πίσω από τα πιστοποιητικά ενός νόμιμου χρήστη τότε όλοι οι υπόλοιποι μηχανισμοί που θα δούμε παρακάτω είναι ανεπαρκείς για την προστασία των δεδομένων μας. Θα πρέπει να έχουμε υπόψιν ότι η διαδικασία του authentication είναι διαφορετική από αυτή του authorization Σχήμα 3.7 Authentication & Authorizationto οποίο θα δούμε και σε επόμενη υποενότητα.



Σχήμα 3.7 Authentication & Authorization

3.4.3 Session Management

Η διαχείρηση της συνεδρίας (session) είναι κάτι που χρησιμοποιείται σε οποιαδήποτε σοβαρή εφαρμογή που «τρέχει» σε κάποιον εξυπηρετητή (server) και ένας ή περισσότεροι πελάτες (clients) αποκτούν πρόσβαση σε δεδομένα ή ενέργειες σε αυτόν. Για να μπορέσει ένας χρήστης, να «πλοηγηθεί» ανάμεσα στις διάφορες σελίδες μιας ιστοσελίδας θα έπρεπε κάθε φορά που επιλέγει μια διαφορετική σελίδα στον ίδιο ιστότοπο να κάνει εκ νέου log in με τα στοιχεία του. Αυτό πρακτικά είναι αδύνατο καθώς η ίδια η σελίδα θα ήταν απαίσια και αποτρεπτική. Προκειμένου να αποφευχθεί όλη αυτή η ταλαιπωρία , έχουν εφευρεθεί τρόποι για την διατήρηση μιας συνεχιζόμενης συνεδρίας από τη στιγμή που ο χρήστης θα

συνδεθεί την πρώτη φορά. Στην ουσία ο εξυπηρετητής δίνει ένα μοναδικό «εισιτήριο» στον κάθε χρήστη το session ID. Δεδομένα όπως η ταυτότητα του χρήστη και ο ρόλος του (περισσότερα στην αμέσως επόμενη ενότητα) αποθηκεύονται στον εξυπηρετητή. Η ροή της συνεδρίας έχει ως εξής:

- Authentication : Ο χρήστης συνδέεται στον εξυπηρετητή μέσω της διαδικασίας του login περνώντας τα διαπιστευτήριά του.
- Creation: Ο εξυπηρετητής παραλαμβάνει τα στοιχεία που του έδωσε ο χρήστης και ελέγχει τη γνησιότητά τους. Κατόπιν δημιουργεί μια συνεδρία την οποία αποθηκεύει είτε στη βάση δεδομένων του είτε στη μνήμη.
- Transmission: Ο εξυπηρετητής παίρνει το ID της συνεδρίας που μόλις δημιούργησε (session ID) και το στέλνει πίσω στον φυλλομετρητή του πελάτη – χρήστη, συνήθως με τη χρήση ενός cookie.
- Persistence: Σε κάθε επόμενη αίτηση του χρήστη που αφορά την ίδια σελίδα , ο χρήστης στέλνει αυτόματα το cookie που παρέλαβε. Ο εξυπηρετητής ελέγχει το session ID – επιβεβαιώνει ότι το έχει στα αποθηκευμένα του και θεωρεί τον χρήστη ως επιβεβαιωμένο για να μην επαναλάβει τη διαδικασία του login.

Αυτός είναι ο τρόπος που δουλεύει το session management σε μια τυπική συναλλαγή ανάμεσα σε χρήστη και εξυπηρετητή π.χ. για την ανάγνωση μιας ιστοσελίδας. [12]Ο μηχανισμός που γίνεται το authentication μπορεί να διαφέρει βέβαια. Υπάρχει και η περίπτωση – πιο δημοφιλής – όπου την έκδοση του session id δεν την κάνει ο εξυπηρετητής και δεν αποθηκεύει κάτι. Είναι οι λεγόμενες Client-Side Sessions (JWT). Τα δεδομένα στην έναρξη της συνεδρίας περνιούνται κωδικοποιημένα σε ένα “token” (JSON Web Token) [13] το οποίο παραμένει αποθηκευμένο στον ίδιο τον χρήστη. Με την κάθε συναλλαγή που πραγματοποιεί ο χρήστης , ο εξυπηρετητής απλά επαληθεύει την υπογραφή του token που περνάει κάθε φορά. Έτσι διατηρείται η συνεδρία και δεν χρειάζεται εκ νέου η διαδικασία της πιστοποίησης κάθε φορά.

Καθώς το APNR System πρόκειται να δουλέψει τοπικά και offline , η υλοποίηση με tokens αποτελεί υπερβολή. Ο σκοπός του είναι καθαρά η υποστήριξη του RBAC και μάλιστα είναι ο βέλτιστος τρόπος για να επιτευχθεί στην περίπτωσή μας. Διαφορετικά, θα έπρεπε σε κάθε κλάση που καλούμε και χρειάζεται η επιβεβαίωση του ρόλου του χρήστη να τρέχουμε σε κλήσεις προς την τοπική βάση δεδομένων ή σε ανάγνωση μεταβλητών από την δημιουργία τους στην Login page (εν προκειμένω gui_login.py) και το πέρασμά τους σε κάθε κλάση. Αυτό όμως θα αναιρούσε και την αρχιτεκτονική μας που στήνει την εφαρμογή με βάση το SoC – και θα έκανε το UI να έχει γνώση πληροφοριών και μεταβλητών που **απαγορεύεται** να κατέχει. Είπαμε και επιμένουμε – το UI πρέπει να είναι εντελώς «χαζό». Ο μηχανισμός μας -ως προς το session management- έχει ως εξής:

- Ο χρήστης παρέχει τα διαπιστευτήριά του στο login page - γίνεται το authentication.
- Με την επιτυχή επιβεβαίωση των στοιχείων που παρείχε ο χρήστης – διαβάζουμε το ρόλο που έχει αποδοθεί στον χρήστη με αυτά τα στοιχεία (εδώ μπορούσε να γίνει ακόμη πιο ασφαλές το σύστημα με υλοποίηση μιας κλάσης – policies).
- Ο ρόλος του χρήστη που μόλις συνδέθηκε με επιτυχία «περνάει» στο session που μόλις δημιουργήθηκε. Με αυτόν τον τρόπο – διαβάζοντας τα attributes του session από **οπουδήποτε** στον κώδικά μας έχουμε πρόσβαση στον ρόλο του χρήστη. [12] Οι υπόλοιπες κλάσεις δεν ξέρουν τίποτα- συνεχίζουν να αγνοούν οποιαδήποτε επιπλέον πληροφορία και απλά «διαβάζουν» τον ρόλο του χρήστη μόνο όταν υπάρχει λόγος.

3.4.4 RBAC (Role Based Access Control)

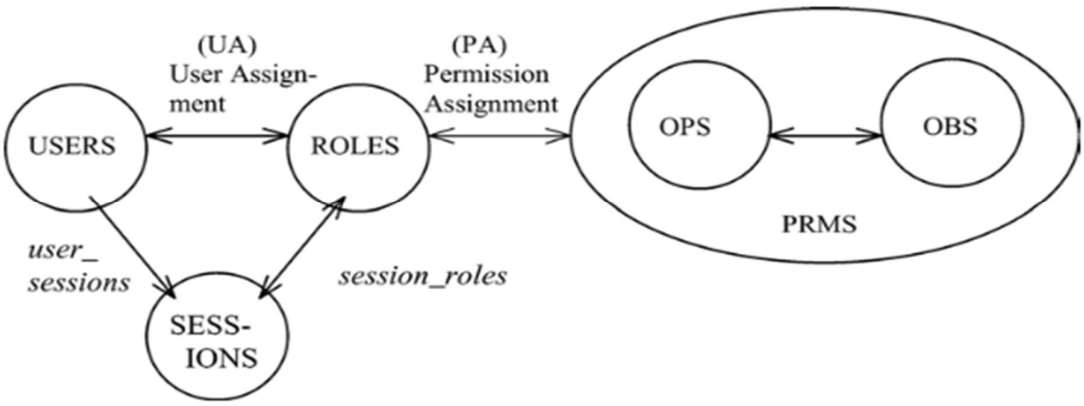
Το 1992 το Εθνικό Ινστιτούτο Προτύπων και Τεχνολογίας των Ηνωμένων Πολιτειών Αμερικής (NIST) παρουσίασε ένα μοντέλο για τον έλεγχο πρόσβασης με βάση τον ρόλο του χρήστη. [15, 16] Το αρχικό αυτό μοντέλο αναθεωρήθηκε και βελτιώθηκε και το έτος 2004 η Διεθνής Επιτροπή Προτύπων Τεχνολογίας Πληροφοριών και το Αμερικανικό Εθνικό Ινστιτούτο Προτύπων το τυποποίησαν [17].

Σκοπός του προτύπου είναι η καλύτερη διαχείριση πρόσβασης σε συστήματα πληροφορικής , αποσυνδέοντας τους χρήστες καθ' εαυτούς και συνδέοντας τα δικαιώματα που έχει ο κάθε ρόλος που κατέχουν. Έτσι, εισάγοντας την έννοια του «ρόλου» μπαίνουν στο παιχνίδι οι αρμοδιότητες που έχει κάποιος που κατέχει έναν ρόλο. Με αυτόν τον τρόπο μπορεί να γίνει άμεσα και με πολύ μεγάλη ασφάλεια η προσθήκη και η αφαίρεση χρηστών από ένα σύστημα, απλά αφαιρώντας ή προσθέτοντας τον αντίστοιχο ρόλο σε αυτόν.

Για την ολοκληρωμένη πρόσβαση το πρότυπο ορίζει ότι πρέπει πρώτα να δημιουργηθούν ρόλοι σε μια οντότητα. Κατόπιν, θα πρέπει να δοθούν συγκεκριμένα δικαιώματα σε κάθε ρόλο , ώστε να μπορούν να εκτελεστούν συγκεκριμένες ενέργειες όπου απαιτείται και τίποτα περισσότερο. Με αυτόν τον τρόπο περιορίζονται τα δικαιώματα στα απολύτως απαραίτητα. Με την καταχώρηση ενός νέου χρήστη, του αποδίδεται ένας ρόλος και αποκτά τα δικαιώματα και τις άδειες για εκτέλεση ενεργειών που αντιστοιχούν στον ρόλο αυτό. Όλη αυτή η αλληλεπίδραση «τυλίγεται» σε μια συνεδρία και με βάση αυτή πλέον προχωρά παρακάτω στις συναλλαγές ο χρήστης.

Το πρότυπο αυτό (ANSI/INCITS 359-2004) ορίζει τρία διακριτά επίπεδα τα οποία μπορούν να εφαρμοστούν σταδιακά , ξεκινώντας από αυτό που μόλις περιγράψαμε και ονομάζοντάς το «βασικό RBAC»(Core RBAC) και προχωρώντας στο «ιεραρχικό RBAC»(Hierarchical RBAC) το οποίο προσθέτει την ύπαρξη ιεραρχίας στους ρόλους. Με αυτόν τον τρόπο , επιτυγχάνεται η κληρονομικότητα των δικαιωμάτων πάντα όμως από κάτω προς τα επάνω. Ξεκινώντας από το πιο χαμηλό επίπεδο με τα χαμηλότερα δικαιώματα – πηγαίνουμε σε ανώτερο επίπεδο ρόλων στο οποίο οι χρήστες του έχουν κληρονομήσει τα δικαιώματα των κατώτερων κι επιπλέον αποκτούν κάποια επιπλέον. Τόσο απλά, μειώνεται δραματικά η ανάγκη για την απόδοση δικαιωμάτων δύο φορές, αφού πλέον είναι περιττή. Τέλος, εισάγονται περιορισμοί στο ανώτερο επίπεδο – εξ' ου και η ονομασία «περιορισμένο RBAC»(Constrained RBAC). Ο σημαντικότερος είναι ο διαχωρισμός των καθηκόντων (Separation of Duties – SoD). Εδώ αυξάνεται η ασφάλεια του συστήματος καθώς διασφαλίζεται ότι κανένας χρήστης δεν μπορεί να αποκτήσει υπερβολικά δικαιώματα. Ο διαχωρισμός μπορεί να γίνεται είτε δυναμικά(Dynamic Separation of Duties – DSD) , είτε στατικά(Static Separation of Duties). Στην πρώτη περίπτωση ο χρήστης μπορεί να έχει πολλαπλούς ρόλους οι οποίοι μπορούν να είναι και συγκρουόμενοι αλλά **ποτέ** στην ίδια συνεδρία (session). Στην δεύτερη, ο χρήστης απαγορεύεται να έχει δύο συγκρουόμενους ρόλους εξ ορισμού [16].

Στο APNR System έχει υλοποιηθεί το RBAC μόνο σε ένα επίπεδο – καθαρά για την αύξηση της ασφάλειας και μόνο. Στην ανάλυση του κώδικα σε μεταγενέστερο κεφάλαιο θα δούμε σε ποιο τμήμα και πως υλοποιήθηκε – αξίζει να αναφερθεί όμως ότι ο μοναδικός που μπορεί να διαχειριστεί τους χρήστες του συστήματος είναι κάποιος που έχει λογαριασμό με δικαιώματα «administrator». Επισημαίνεται ότι δεν έγινε πιο αναλυτική υλοποίηση καθώς αυτή διευρύνει πολύ περισσότερο απ' όσο απαιτείται τις λειτουργίες της εφαρμογής (Σχήμα 3.8).



Σχήμα 3.8 RBAC

3.4.5 Authorization

Όπως αναφέρθηκε νωρίτερα στην ενότητα 3.4.2 η διαδικασία της πιστοποίησης είναι αυτή που καλείται να απαντήσει στην ερώτηση «ποιος είσαι» ενώ η διαδικασία της εξουσιοδότησης(authorization) έρχεται να δώσει απάντηση στην ερώτηση «τί έχεις δικαίωμα να κάνεις δεδομένου του ρόλου που έχεις». Είναι το τελευταίο στάδιο του μηχανισμού που ελέγχει την πρόσβαση στην εφαρμογή μας και τις ενέργειες στις οποίες μπορεί να προχωρήσει ή απαγορεύεται να εκτελέσει ο χρήστης.

Γενικά, σε όλες τις επαγγελματικές εφαρμογές εφαρμόζεται η αρχή των ελάχιστων προνομίων(Principle of Least Privilege- PoLP). Αυτό συνεπάγεται ότι εκ προοιμίου ο χρήστης δεν έχει πρόσβαση σε κανέναν πόρο εκτός κι αν υπάρχει συγκεκριμένος κανόνας που να την επιτρέπει. Για την καλύτερη διαχείριση του μηχανισμού, γίνεται διαχωρισμός σε δύο σημεία: Στο «σημείο απόφασης της πολιτικής(Policy Decision Point- PDP) και στο «σημείο επιβολής πολιτικής(Policy Enforcement Point- PEP). Το πρώτο είναι το σημείο της εφαρμογής όπου η προσπάθεια του χρήστη για πρόσβαση σε δεδομένα ή ενέργεια, αξιολογείται με βάση το ρόλου που του έχει αποδοθεί και λαμβάνεται η απόφαση της έγκρισης ή της απόρριψης της ενέργειας αυτής. Το δεύτερο, είναι το σημείο όπου η ενέργεια που προσπαθεί να εκτελέσει ο χρήστης εξετάζεται αυστηρά και με βάση την απάντηση που θα επιστραφεί από το PDP προχωράει στην εκτέλεση ή στην απόρριψη.

Ειδικά στην περίπτωσή μας, καθώς το APNR System βασίζεται στο SoC και πρόκειται να εκτελείται τοπικά χωρίς πρόσβαση στο διαδίκτυο , όλη η διαδικασία του authorization έχει άμεση σχέση με το session management που είδαμε στην υποενότητα 3.4.3 και το βασικό RBAC (Core RBAC) της υποενότητας 3.4.4. Για λόγους συντομίας δεν υλοποιήθηκε ολόκληρος ο μηχανισμός αναλυτικά, όπως επίσης δεν δημιουργήθηκε ξεχωριστή κλάση «policy_management» για να ελέγχονται οι πολιτικές ως υπηρεσία. Η υλοποίηση του μηχανισμού εξουσιοδότησης γίνεται με τον απλό τρόπο: Αρχικά ο χρήστης συνδέεται μέσω του gui_login και αφού επαληθευθεί η ταυτότητά του – γίνεται αρχικοποίηση μιας συνεδρίας στην οποία προστίθεται ως attribute ο ρόλος του χρήστη που συνδέθηκε. Όταν ο χρήστης προσπαθήσει να διαχειριστεί λογαριασμούς χρηστών από το gui_main τότε ο κώδικας λειτουργεί απ' ευθείας ως σημείο επιβολής πολιτικής(PEP). Διαβάζει τον ρόλο του χρήστη από το session που τρέχει ήδη και μόνο εφόσον ο χρήστης έχει ρόλο «ADMINISTRATOR» του επιτρέπει να προχωρήσει σε CRUD λογαριασμών χρηστών. Σε οποιαδήποτε άλλη περίπτωση – εάν ο χρήστης έχει ρόλο «USER» ή «SUPERVISOR» το αίτημα για CRUD απορρίπτεται και η ενέργεια ακυρώνεται **πριν καν φτάσει στην λογική της εφαρμογής** ώστε να προστατευθεί η βάση δεδομένων και να αποτραπεί η εκτέλεση μη εξουσιοδοτημένου κώδικα. Έτσι συνεχίζει να είναι «χαζό» το UI μας(SoC Rules!) [12]

3.4.6 Dependency Injection (DI)

Το DI – ελληνιστί «έγχυση εξαρτήσεων» είναι ένα πρότυπο σχεδιασμού για την κατασκευή σύγχρονων εφαρμογών. Στην ουσία – αποτελεί ένα ενδιάμεσο επίπεδο στις κλήσεις μεθόδων από ένα αρχείο προς μια υπηρεσία ή μεταξύ υπηρεσιών και βασικών «ευαίσθητων» στοιχείων της εφαρμογής. Είναι ο τρόπος να «συνθέσουμε» ένα πολύπλοκο αντικείμενο απλά «εγχέοντας» τις εξαρτήσεις από άλλα αρχεία και υπηρεσίες σε αυτό. Το dependency είναι ένα αντικείμενο το οποίο για να μπορέσει να χρησιμοποιηθεί-να γίνει λειτουργικό – χρειάζεται μια κλάση. Το injection είναι η διαδικασία της εισόδου του αντικειμένου από εξωτερική πηγή , ώστε να το «βρει» έτοιμο. Έτσι δεν χρειάζεται να υλοποιηθεί κάθε φορά από την αρχή – παίρνουμε το παραγόμενο «σύνθετο» αντικείμενο και το αξιοποιούμε άμεσα [16]Κέρδος: Ο κώδικας της εφαρμογής γίνεται πιο ευέλικτος – αλλάζοντας μια κλάση που υλοποιεί ένα κομμάτι του σύνθετου αντικειμένου δεν χρειάζεται να αλλάξουμε όλα τα σύνθετα αντικείμενα. Κάνουμε τις αλλαγές που επιθυμούμε και μέσω του DI αυτές εφαρμόζονται αυτόματα. Επίσης , είναι πολύ πιο εύκολη η προσθήκη νέων χαρακτηριστικών χωρίς να υπάρχει κίνδυνος να «σπάσουμε» κάτι , ενώ το κομμάτι της επαλήθευσης της εφαρμογής γίνεται πολύ πιο εύκολο. Στη γλώσσα προγραμματισμού C# , το DI αποτελεί βασικό τμήμα του .NET Framework ,ενώ στην γλώσσα Python η δυναμικότητά της δεν καθιστά απαραίτητη τη χρήση DI. **ΟΜΩΣ** σε μεγάλες επαγγελματικές εφαρμογές χρησιμοποιείται. Η βιβλιοθήκη "Dependency Injector" περιέχει πολλά εργαλεία για άμεση χρήση [17]Στο APNR System το DI **υλοποιείται** , όχι όμως με τη χρήση βιβλιοθηκών, αλλά με «χειροκίνητο» τρόπο – για την χρήση του container που θα δούμε σε μεταγενέστερη ενότητα.

3.5 You Only Look Once (YOLO)

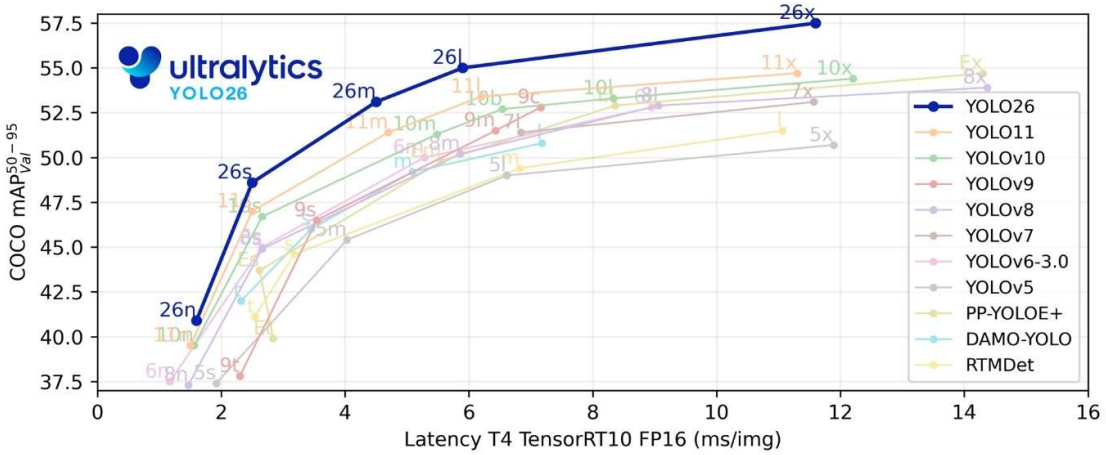
Στις ενότητες 1.2 και 2.3.3 είδαμε τον τρόπο λειτουργίας ενός ΣΝΔ και αναφερθήκαμε στα R-CNN. Το YOLO ήρθε να αλλάξει τελείως τα πράγματα [20, 21]. Με μόνο ένα πέρασμα όλης της εικόνας ο αλγόριθμός του αναγνωρίζει τα αντικείμενα και τα ταξινομεί ενώ παράλληλα εντοπίζει τη θέση τους στην εικόνα απευθείας, αναθέτοντας και bounding boxes γύρω από τα αντικείμενα αυτά. Η εικόνα «τεμαχίζεται» σε επιμέρους τμήματα διαστάσεων *S x S.* Κάθε τμήμα του «πλέγματος» που δημιουργείται έτσι έχει ως στόχο τον εντοπισμό των αντικειμένων των κλάσεων που έχει εκπαιδευτεί να αναγνωρίζει – την ανάθεση των bboxes και του βαθμού εμπιστοσύνης αυτών. Θα πρέπει να αναφέρουμε ότι ο βαθμός εμπιστοσύνης για την ύπαρξη μιας κλάσης ή όχι δεν είναι παρά μόνο το αποτέλεσμα της Σχέση 3.1 . Ουσιαστικά πρόκειται για το γινόμενο της πιθανότητας ύπαρξης της κλάσης του αντικειμένου (κανονικοποιημένη στο διάστημα [0,1]) με το Intersection over Union -IoU της επιφάνειας του «αληθούς» πλαισίου με την επιφάνεια του πλαισίου που προβλέφθηκε από το ίδιο το μοντέλο (Σχέση 3.2) . Δεν παύει να είναι κι αυτό ένα ΣΝΔ , απλά είναι ένα δίκτυο «ενός σταδίου» με πολλές δεκάδες ή και εκατοντάδες κρυφά επίπεδα. Η πληροφορία κινείται μόνο προς τα εμπρός χωρίς καμία αναδρομή όπως συμβαίνει στα Αναδρομικά Νευρωνικά Δίκτυα (Recurrent Neural Networks – RNN's).

$$confidence\ score = p(Object) * IOU_{pred}^{truth} \quad \text{[Σχέση 3.1]}$$

$$IoU = \frac{Area(Truth \cap Predictions)}{Area(Truth \cup Predictions)} \quad \text{[Σχέση 3.2]}$$

Η αρχή για το YOLO έγινε το 2015 στο πανεπιστήμιο της Ουάσιγκτον. Πρόκειται για ένα βαθύ νευρωνικό δίκτυο (Deep Learning).Η πρώτη έκδοση έκανε ανίχνευση σε πραγματικό χρόνο. Η

αρχιτεκτονική του από την έκδοση YOLOv3 άλλαξε σε (Darknet-53) το μετέτρεψε στο απόλυτο στάνταρ παγκοσμίως. Από το 2020 η ανάπτυξη πέρασε στην κοινότητα – κι εκεί κάπου έγινε η μεγάλη αλλαγή από την εταιρία Ultralytics όπου η έκδοση v5 ξαναγράφτηκε από την αρχή σε Python [18]. Πλέον έχουμε φτάσει στην έκδοση YOLO26 από την ίδια εταιρία , η οποία είναι ότι καλύτερο έχει υπάρξει μέχρι τώρα, με λιγότερες παραμέτρους από τις προηγούμενες εκδόσεις- με δυνατότητα να τρέχει σε κινητές συσκευές , με απίστευτες επιδόσεις. [21, 22] Η μεγάλη αλλαγή είναι ότι πλέον δεν υπάρχει NMS σε μια προσπάθεια να αυξηθεί ακόμη περισσότερο η ταχύτητά του και να γίνει πιο ελαφρύ. Σχεδιάστηκε ξανά από την αρχή με σκοπό να γίνει κατάλληλο για edge συσκευές και χαμηλή κατανάλωση ενέργειας και οι επιδόσεις του συγκριτικά με τις προηγούμενες εκδόσεις είναι εντυπωσιακές (43% γρηγορότερο inference με εκτέλεση CPU) **Error! Reference source not found.**(Σχήμα 3.9). Ο δείκτης mAP του σχήματος αναφέρεται στο mean Average Precision – την ακρίβεια εντοπισμού του μοντέλου μας.



Σχήμα 3.9 YOLO models Comparison in COCO

Κατά την ανάπτυξη της εφαρμογής APNR System , η βέλτιστη λύση για τον εντοπισμό μιας πινακίδας κυκλοφορίας ήταν ένα YOLO11 – βελτιστοποιημένο για να εντοπίζει την κλάση “LicensePlate”. Η έκδοση YOLO26 εμφανίστηκε μετά την ολοκλήρωση της υλοποίησης, αλλά σύμφωνα με τα benchmark αποτελεί μια πολύ καλύτερη λύση. Το μοντέλο που χρησιμοποιείται για τον εντοπισμό της πινακίδας κυκλοφορίας είναι το «license-plate-finetune-v1n.onnx» -η nano έκδοση- ήδη βελτιστοποιημένο από huggingface.co.

[23]Διατίθεται με άδεια χρήσης GNU AGPLv3 εξαιτίας της άδειας του YOLO11 – και οφείλουμε να αναφέρουμε την συνεισφορά του Roboflow Universe για το dataset 4.0, το MorseTechLab του huggingface.co για την διαδικασία της επανεκπαίδευσης- βελτιστοποίησης και φυσικά την Ultralytics για τη διάθεσή του.

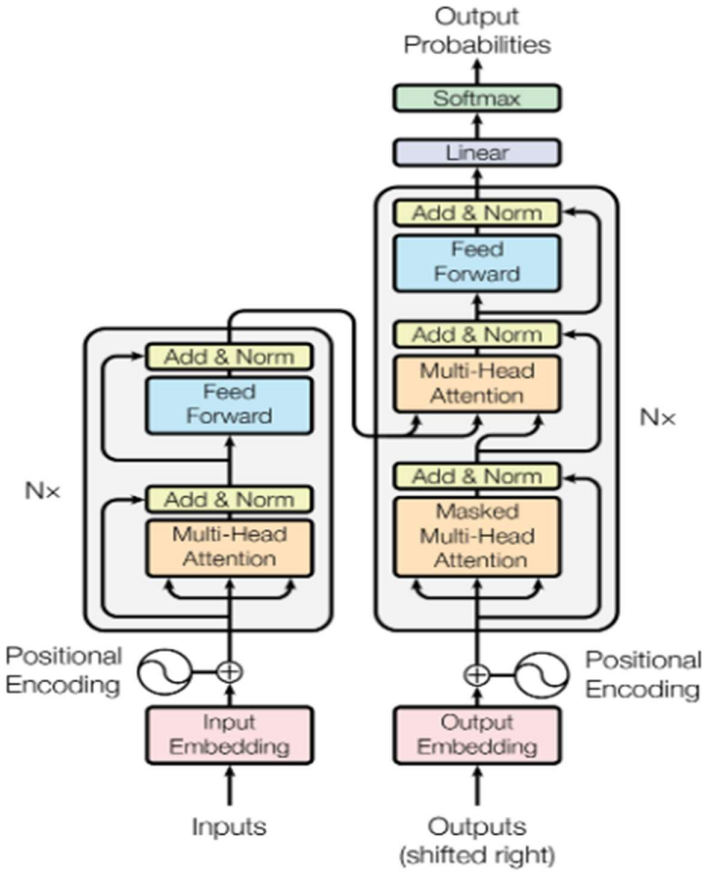
3.6 Transformers

Το 2017 ερευνητές της Google παρουσίασαν την αρχιτεκτονική Transformer. Πρόκειται για την βάση όλων των σημερινών Μεγάλων Γλωσσικών Μοντέλων (Large Language Models- LLMs). Η επεξεργασία κειμένου γίνονταν με αρχιτεκτονικές RNN όπου τα δεδομένα ενός κειμένου διαβαζόντουσαν κι επεξεργαζόντουσαν σειριακά. Εδώ είναι η μεγάλη διαφορά με τους transformers καθώς όλες οι λέξεις επεξεργάζονται παράλληλα – ταυτόχρονα.

Η επιτυχία τους βρίσκεται στον μηχανισμό που υλοποιούν: ονομάζεται μηχανισμός αυτοπροσοχής. Κατά την επεξεργασία μιας πρότασης, το μοντέλο δεν κοιτάζει μόνο μια λέξη – αλλά εστιάζει και σε άλλες σχετικές λέξεις της πρότασης για να μπορέσει να την κατανοήσει καλύτερα.

Η δομή ενός κλασσικού μοντέλου που ανήκει στην κατηγορία αυτή είναι η εξής: χωρίζεται σε δύο κύρια τμήματα , τον κωδικοποιητή (Encoder) και τον αποκωδικοποιητή (Decoder). Η δουλειά του κωδικοποιητή είναι να διαβάσει την είσοδο που του δίνεται (κείμενο) και να την αναπαραστήσει μαθηματικά με βάση τα συμφραζόμενα. Αποτελείται από μια στοίβα πανομοιότυπων επιπέδων τα οποία στον αρχικό paper που πρωτοπαρουσιάστηκαν ήταν 6. Ο αποκωδικοποιητής με τη σειρά του , παίρνει την μαθηματική αναπαράσταση που δημιουργήθηκε από τον κωδικοποιητή και αναπαράγει το τελικό αποτέλεσμα τμηματικά. Ομοίως αποτελείται από μια στοίβα επιπέδων τα οποία όμως έχουν και ένα ακόμη επιπλέον χαρακτηριστικό – ένα μηχανισμό προσοχής που κοιτάζει την έξοδο που παρήγαγε ο κωδικοποιητής.

Παρακάτω μπορούμε να δούμε αναλυτικά τα εξής σημαντικά σημεία που περιλαμβάνονται σε κάθε επίπεδο ενός Transformer (Σχήμα 3.10) :



Σχήμα 3.10 Αρχιτεκτονική των Transformers

- Input Embeddings(Ενσωματώσεις εισόδου): Εδώ γίνεται η μετατροπή των λέξεων σε διανύσματα πολύ μεγάλων διαστάσεων. Λέξεις που είναι παρόμοιες έχουν και παραπλήσια διανύσματα.

- Positional Encoding(Κωδικοποίηση θέσης): Η ανάγνωση όλων των λέξεων που δίνονται ως κείμενο στο μοντέλο γίνεται ταυτόχρονα. Για να μπορέσει λοιπόν να έχει επίγνωση της ακριβούς θέσης κάθε λέξης μέσα στο κείμενο , προσθέτει μια μαθηματική τιμή στο διάνυσμα (συνάρτηση ημιτόνου και συνημιτόνου) που την αντικατοπτρίζει.
- Self-Attention Mechanism – Q,K,V(Μηχανισμός προσοχής- Queries, Keys, Values): Η «καρδιά» του μοντέλου. Εδώ γίνεται η «μεταφορά και απορρόφηση» πληροφορίας από μια λέξη στο διάνυσμα μιας άλλης. Το μοντέλο για να μπορέσει να βγάλει την έξοδο προσοχής πρέπει να δημιουργήσει τρία διανύσματα για κάθε λέξη που του δόθηκε πολλαπλασιάζοντάς την με πίνακες βαρών που μπορούν να εκπαιδευτούν:
  1. Query(Ερώτημα -Q): Τί είδους πληροφορία ψάχνει η λέξη αυτή;
  2. Key(Κλειδί – K):Τί είδους πληροφορία περιέχει αυτή η λέξη;
  3. Value(Τιμή -V): Ποια πληροφορία περιέχει η λέξη πραγματικά;

  Η παρακάτω εξίσωση (Σχέση 3.3) ονομάζεται “Scaled Dot-Product Attention” με d_k να αντιστοιχεί στο μέγεθος της διάστασης των διανυσμάτων.

$$\text{Attention}(Q,K,V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

[Σχέση 3.3]

- Multi-Head Attention(Προσοχή πολλαπλών κεφαλών): το μοντέλο ,αντί να υπολογίζει την προσοχή μία μοναδική φορά , τρέχει τον μηχανισμό πολλές φορές παράλληλα – με διαφορετικές «κεφαλές». Κάθε κεφαλή δίνει βάρος σε διαφορετικές σχέσεις που βρίσκει μέσα στο κείμενο που δόθηκε – όπως πχ επίθετα με ουσιαστικά ή υποκείμενα με ρήματα.
- Feed Forward Neural Networks(Δίκτυα εμπρόσθιας τροφοδότησης): μετά από τον μηχανισμό προσοχής η έξοδος κάθε λέξης τροφοδοτείται ανεξάρτητη σε ένα κλασσικό πλήρως συνδεδεμένο νευρωνικό δίκτυο. Με αυτόν τον τρόπο η πληροφορία επεξεργάζεται βαθύτερα.
- Add & Norm(Προσθήκη και κανονικοποίηση):Σε κάθε βήμα το μοντέλο αυτοτροφοδοτείται με την έξοδο του προηγούμενου βήματος (auto-regression) ως πρόσθετη είσοδο. Αυτό βοηθάει στην κανονικοποίηση των δεδομένων καθώς μειώνεται το φαινόμενο της εξάλειψης του gradient descend κι επιτρέπει την εκπαίδευση σε βαθύτερο επίπεδο.

Με αυτόν τον τρόπο ένα μοντέλο που χρησιμοποιεί τον κωδικοποιητή είναι καλό στην κατανόηση κειμένου που δόθηκε σαν είσοδος , ενώ ένα μοντέλο που χρησιμοποιεί μόνο αποκωδικοποιητή είναι κατάλληλο για παραγωγή κειμένων.

3.7 Compact Convolutional Transformers – CCTs / Vision Transformers - ViTs

Το 2020 η εταιρία Google έφτιαξε το ViT(Vision Transformer) δείχνοντας στην πράξη ότι πλέον οι Transformers μπορούν να αντιληφθούν οπτικές εισόδους (εικόνες) αποδίδοντας το ίδιο με άλλα Συνελικτικά Νευρωνικά Δίκτυα. Υπήρχε όμως ένα τεράστιο πρόβλημα: Τα ViTs δεν μπορούν να κατανοήσουν ότι τα διάφορα εικονοστοιχεία μιας οπτικής εισόδου σχετίζονται με γειτνιάζοντα άλλα όταν περιγράφουν κάποιο σχήμα. Αυτό είναι πάρα πολύ δύσκολα αντιμετωπίσιμο καθώς για να μπορέσει να υπερκεραστεί θα πρέπει να εκπαιδευτούν σωστά με ένα τεράστιο σε μέγεθος σύνολο δεδομένων(dataset) αποτελούμενο από εκατομμύρια ή ακόμη και δισεκατομμύρια εικόνες. Σε οτιδήποτε λιγότερο , η αποτυχία σε σχέση με ένα ΣΝΔ είναι δεδομένη.

Λύση δόθηκε ένα χρόνο αργότερα με τη δημιουργία των CCTs. Συνδυάζοντας τα βέλτιστα τμήματα κάθε μοντέλου σε ένα νέο, φτιάχτηκε ένας νέος Transformer [26]. Τα ΣΝΔ είναι ιδανικά για να εντοπίζουν γραμμές και σχήματα. Τα ViT’s για να δουλέψουν μια εικόνα την «τεμαχίζουν» [27]σε

πολλά τετράγωνα και περνάνε το καθένα από αυτά σαν είσοδο σε έναν Transformer. Κι εδώ είναι που «σκάλωνε» κι αποτύγχανε όλη η διαδικασία καθώς στα τμήματα που γινόταν ο «τεμαχισμός» οι λεπτομέρειες χανόντουσαν εισάγοντας «ασυνέχειες» μεταξύ τους. Στο CCT η λύση είναι απλή: Αρχικά η εικόνα περνάει μέσα από ένα ΣΝΔ και αφού δημιουργηθούν τα feature maps τότε αυτά τα τμήματα γίνονται tokens περνώντας στον transformer ουσιαστική πληροφορία χωρίς θόρυβο γλυτώνοντας τεράστιο χρόνο από την εκπαίδευση. Αμέσως μετά προστίθεται η κωδικοποίηση θέσης των δεδομένων κι έπειτα εισέρχονται στον κλασσικό transformer (Self Attention βλ. ενότητα 3.6).

Τα μοντέλα CCT άλλαξαν δυναμικά τον τρόπο που αντιμετωπίζαμε την Υπολογιστική Όραση καθώς είναι τρομερά ελαφριά και γρήγορα λόγω μεγέθους. Συγκρινόμενα με αντίστοιχα μοντέλα ΣΝΔ , ViTs και Compact ViTs βλέπουμε ότι έχουν σαφέστατα προβάδισμα στην ακρίβεια (accuracy) ενώ ταυτόχρονα ο αριθμός των παραμέτρων τους είναι υπο-πολλαπλάσιος (Σχήμα 3.11) [28].

Model	C-10	C-100	Fashion	MNIST	# Params
Convolutional Networks (Designed for ImageNet)					
ResNet18	90.27%	66.46%	94.78%	99.80%	11.18 M
ResNet34	90.51%	66.84%	94.78%	99.77%	21.29 M
MobileNetV2/0.5	84.78%	56.32%	93.93%	99.70%	0.70 M
MobileNetV2/2.0	91.02%	67.44%	95.26%	99.75%	8.72 M
Convolutional Networks (Designed for CIFAR)					
ResNet56[16]	94.63%	74.81%	95.25%	99.27%	0.85 M
ResNet110[16]	95.08%	76.63%	95.32%	99.28%	1.73 M
ResNet1k-v2*[17]	95.38%	—	—	—	10.33 M
Proxyless-G[5]	97.92%	—	—	—	5.7 M
Vision Transformers					
ViT-12/16	83.04%	57.97%	93.61%	99.63%	85.63 M
ViT-Lite-7/16	78.45%	52.87%	93.24%	99.68%	3.89 M
ViT-Lite-7/8	89.10%	67.27%	94.49%	99.69%	3.74 M
ViT-Lite-7/4	93.57%	73.94%	95.16%	99.77%	3.72 M
Compact Vision Transformers					
CVT-7/8	89.79%	70.11%	94.50%	99.70%	3.74 M
CVT-7/4	94.01%	76.49%	95.32%	99.76%	3.72 M
Compact Convolutional Transformers					
CCT-2/3×2	89.75%	66.93%	94.08%	99.70%	**0.28 M**
CCT-7/3×2	95.04%	77.72%	95.16%	99.76%	3.85 M
CCT-7/3×1	96.53%	80.92%	**95.56%**	**99.82%**	3.76 M
CCT-7/3×1*	**98.00%**	**82.72%**	—	—	3.76 M

Σχήμα 3.11 CNN - ViT - CCT

Από τα ViT όμως εξελίχθηκαν τα MobileViT σε μια προσπάθεια να βελτιωθεί το inference σε συσκευές που δουλεύουν Edge AI και κινητές συσκευές. Η πρώτη προσπάθεια έδωσε καλά αποτελέσματα [29], όμως το μεγάλο βήμα ήρθε το 2022 με την παρουσίαση της δεύτερης έκδοσης , του MobileViT-v2 [30]. Η υβριδική αρχιτεκτονική που είχε συνδυάζοντας ομοίως Transformers και ΣΝΔ βοήθησε να συνδυαστούν τα δυνατά σημεία κάθε τμήματος. Η καινοτομία όμως στην έκδοση αυτή ήταν ότι στα βαθύτερα επίπεδα χρησιμοποιεί έναν διαχωρισμένο μηχανισμό αυτοπροσοχής επιτρέποντας στο δίκτυο να «καταλάβει» την γενικότερη εικόνα. Αυτό και μόνο επέτρεψε την αύξηση της ταχύτητας κατά 3.2 φορές σε σχέση με την πρώτη έκδοση και την αύξηση του accuracy. Μεγάλη βελτίωση όμως υπάρχει λόγω του τρόπου που είναι στημένο το μοντέλο. Ενώ στην v1 έχουμε διαφορετικές αρχιτεκτονικές ανάλογα με το μέγεθος, το v2 υλοποιεί έναν μηχανισμό «πολλαπλασιαστή» ο οποίος κλιμακώνει το

πλάτος του δικτύου ανάλογα με τους περιορισμούς που υπάρχουν στο υλικό. Οι εξαιρετικά χαμηλές απαιτήσεις σε ενέργεια και το ψηλό accuracy του χάρισαν την αναγνωρισιμότητα που έχει.

3.8 ONNX και ONNX Runtime

Το Open Neural Network Exchange (ONNX) μπορούμε να πούμε με πολύ απλά λόγια ότι είναι ένα πρότυπο ανοιχτού κώδικα που ανήκει στο Linux Foundation και αρχικά δημιουργήθηκε από την Microsoft και το Facebook. Η σημαντικότητά του είναι τεράστια καθώς επέτρεψε τη χρήση μοντέλων τεχνητής νοημοσύνης ανεξάρτητα από το framework στο οποίο δημιουργήθηκαν. Πριν το onnx ο δημιουργός ενός μοντέλου στο keras ή στο tensorflow ήταν υποχρεωμένος να χρησιμοποιήσει τα ίδια οικοσυστήματα για να το εκτελέσει. Με την έλευσή του , η εκπαίδευση του μοντέλου γίνεται οπουδήποτε και ο δημιουργός αρκεί να το μετατρέψει σε ένα αρχείο μορφής «*.onnx». Έτσι με αυτή την ενιαία μορφή μπορεί το μοντέλο να «τρέξει» σε οποιοδήποτε περιβάλλον χωρίς περιορισμούς. Η πραγματική δύναμη όμως του onnx δεν είναι ο τύπος του αρχείου ή ο γενικός χαρακτήρας του. Όταν το μοντέλο συνδυαστεί με το ONNX Runtime(ORT)- μια μηχανή εκτέλεσης (inference engine) τότε μπορεί να γίνει αντιληπτή η τεράστια ταχύτητά του καθώς το ORT είναι φτιαγμένο με τέτοιο τρόπο ώστε να παίρνει τα αρχεία .onnx και να τα τρέχει σε εφαρμογές (production). Οι βελτιστοποιήσεις του προς αυτή την κατεύθυνση το κάνουν να είναι το ιδανικό πακέτο εκτέλεσης μοντέλου TN. Ανεξάρτητα από το λειτουργικό σύστημα και τα τεχνικά χαρακτηριστικά του μηχανήματος – το ORT διαβάζει το υλικό και προσαρμόζει το μοντέλο ώστε να έχει τη μέγιστη δυνατή απόδοση κάθε φορά [31]Η ύπαρξη κάρτας γραφικών και οι διαφορετικοί τύποι επεξεργαστών δεν αποτελούν πλέον πρόβλημα καθώς ακόμη και σε σύστημα με χαμηλές υπολογιστικές δυνατότητες και στερούμενου κάρτας γραφικών – όπως για παράδειγμα ένα Raspberry Pi 3B+ - το ORT θα παραμετροποιήσει το μοντέλο κατάλληλα [[30]. Ο μηχανισμός του ORT παίρνει τα βάρη του μοντέλου μας και τα «κβαντίζει». Ειδικά στις περιπτώσεις υπολογισμών με αριθμούς float , αυτό βοηθά τρομερά στην αύξηση της ταχύτητας χωρίς να εξαντληθεί η – περιορισμένη στην περίπτωσή μας – πολύτιμη μνήμη RAM και χωρίς να «ψήσουμε» την πλατφόρμα μας από την υπερπροσπάθεια. Έτσι ο προγραμματιστής υλοποιεί το μοντέλο του σε όποιο framework του αρέσει , το μετατρέπει πολύ εύκολα σε ένα ενιαίο αρχείο της μορφής onnx και κατόπιν παίρνει αυτό το αρχείο και το ενσωματώνει σε οποιαδήποτε εφαρμογή – εγκαθιστώντας μόνο το ORT που θα κάνει ουσιαστικά την βελτιστοποίηση για την εκτέλεση.

Με βάση όλα τα παραπάνω η επιλογή του ONNX Runtime ήταν και είναι μονόδρομος. Τα μοντέλα που χρησιμοποιούνται στην υλοποίησή μας – τόσο στο στάδιο του detection («license-plate-finetune-v1n .onnx») όσο και στο στάδιο του OCR (1. «cct_xs_v1_global.onnx» , 2. «cct_xs_relu_v1_global.onnx» , 3. «cct_s_v1_global.onnx» , 4. «cct_s_relu_v1_global.onnx» , 5. «european_mobile_vit_v2_ocr.onnx») είναι **όλα** σε αυτή τη μορφή και ολόκληρο το inference βασίζεται πάνω στην εκτέλεση μέσω του ORT.

3.9 Τα μοντέλα της υλοποίησης

Το APNR System όπως είπαμε και σε προηγούμενο κεφάλαιο στο inference pipeline χρησιμοποιεί για τον εντοπισμό των πινακίδων ένα βελτιστοποιημένο μοντέλο nano YOLOv11 «yolov11-license-plate-detection.onnx [23]. Για το OCR όμως δίνεται η δυνατότητα στον χρήστη να επιλέξει ανάμεσα σε 5 μοντέλα αυτό που θα χρησιμοποιηθεί. Όλα τους είναι διαθέσιμα μέσω της βιβλιοθήκης «fast-plate-ocr». Ο προγραμματιστής μπορεί να κατεβάσει ένα από τα ήδη βελτιστοποιημένα μοντέλα από το FastPlateOCR HUB [31]επιλέγοντας αυτό που ταιριάζει στις ανάγκες του. Αναλυτικά χαρακτηριστικά τους στο Παράρτημα Β.

3.10 Επίλογος

Στο κεφάλαιο αυτό παρουσιάστηκαν αρχικά οι τρόποι υλοποίησης μιας αρχιτεκτονικής λογισμικού με βαρύτητα στο SoC. Είδαμε πως μπορούμε να διασφαλίσουμε το Separation of Concerns διαμορφώνοντας τον κώδικά μας σε layers ,χωρίζοντάς τον σε μικροϋπηρεσίες ή εφαρμόζοντας το πρότυπο MVC. Κατόπιν παρουσιάστηκε το περιβάλλον ανάπτυξης που χρησιμοποιήθηκε για την υλοποίηση και τα βήματα ενός σωστού IAM με RBAC, καθώς επίσης και μια σύντομη αναφορά στο DI. Μετά από μια συνοπτική παρουσίαση των αρχιτεκτονικών των μοντέλων που χρησιμοποιούνται στην υλοποίηση είδαμε το ORT. Στο επόμενο κεφάλαιο θα αναλύσουμε τη δομή της εφαρμογής καθώς επίσης και τις μεθόδους που υλοποιεί.

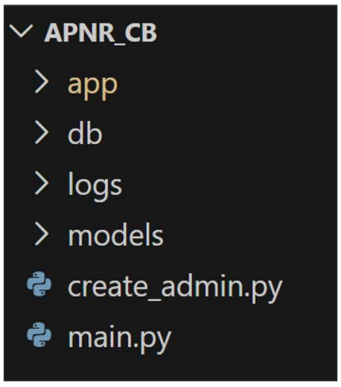
Κεφάλαιο 4ο: Δομή της εφαρμογής

4.1 Εισαγωγή

Στο κεφάλαιο αυτό πρόκειται να δούμε τη δομή της εφαρμογής καθώς επίσης και τις μεθόδους που υλοποιούνται από κάθε αρχείο.

4.2 Δομή

Το APNR System ως modular εφαρμογή και με βάση όλα τα προηγούμενα που αναλύσαμε στο κεφάλαιο 3 **δεν γίνεται** να είναι ένα και μοναδικό αρχείο. Στον αρχικό φάκελο της εφαρμογής (APNR_System) έχουμε τέσσερις φακέλους και δύο αρχεία (Σχήμα 4.1). Τα δύο αρχεία είναι το "create_admin.py" το οποίο εκτελούμε **μία φορά** πριν εκτελέσουμε το κυρίως πρόγραμμα ώστε να δημιουργηθεί μέσα από αυτό ο λογαριασμός του διαχειριστή – με ρόλο «ADMINISTRATOR». Μετά την επιτυχή εκτέλεση του προγράμματος μπορούμε να προχωρήσουμε στην εκτέλεση της κύριας εφαρμογής μέσω του "main.py". Το "main.py" είναι το σημείο εισόδου στην εφαρμογή. Θα τα δούμε αναλυτικότερα σε επόμενη ενότητα και τα δύο. Επίσης, στον root folder έχουμε τους εξής φακέλους: app/ , db/ , logs/ , models/. Παρακάτω θα δούμε αναλυτικά τα περιεχόμενα καθενός από αυτούς.



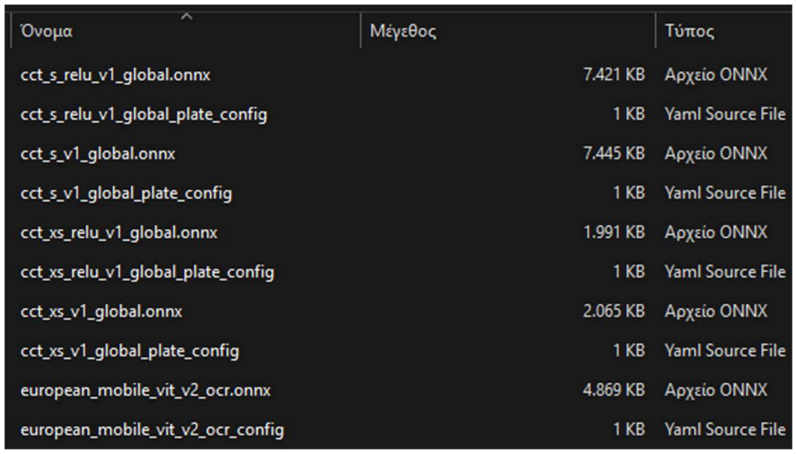
Σχήμα 4.1 Κορυφαίο επίπεδο δομής

4.2.1 models/

Ο φάκελος models όπως είναι και προφανές από την ονομασία του περιέχει δύο φακέλους με τα μοντέλα που χρησιμοποιεί η εφαρμογή στο inference pipeline τους : fast-plate-ocr/ και license_plate_detector/ .

4.2.1.1 fast-plate-ocr/

Στον φάκελο fast-plate-ocr/ περιέχονται τα 5 αρχεία onnx των μοντέλων που χρησιμοποιούνται για το OCR -προδήλως κατανοητό από την ονομασία του (Σχήμα 4.2). Είναι τα 4 CCT που είδαμε νωρίτερα καθώς επίσης και το ένα MobileViT-v2. Στον ίδιο φάκελο περιέχονται και τα αντίστοιχα αρχεία "**.yaml" που χρησιμοποιούνται για το configuration του κάθε μοντέλου μας.



Όνομα	Μέγεθος	Τύπος
cct_s_relu_v1_global.onnx	7.421 KB	Αρχείο ONNX
cct_s_relu_v1_global_plate_config	1 KB	Yaml Source File
cct_s_v1_global.onnx	7.445 KB	Αρχείο ONNX
cct_s_v1_global_plate_config	1 KB	Yaml Source File
cct_xs_relu_v1_global.onnx	1.991 KB	Αρχείο ONNX
cct_xs_relu_v1_global_plate_config	1 KB	Yaml Source File
cct_xs_v1_global.onnx	2.065 KB	Αρχείο ONNX
cct_xs_v1_global_plate_config	1 KB	Yaml Source File
european_mobile_vit_v2_ocr.onnx	4.869 KB	Αρχείο ONNX
european_mobile_vit_v2_ocr_config	1 KB	Yaml Source File

Σχήμα 4.2 Περιεχόμενα φακέλου models/fast-plate-ocr/

4.2.1.2 license_plate_detector/

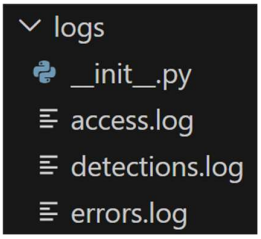
Στον φάκελο αυτό περιέχεται μόνο το αρχείο “license-plate-finetune-v1n.onnx”. Είναι το μοντέλο που χρησιμοποείται από το inference pipeline της εφαρμογής για τον εντοπισμό των πινακίδων κυκλοφορίας των οχημάτων ο “detector”.

4.2.2 db/

Σε αυτόν το φάκελο θα δημιουργηθεί η βάση δεδομένων “police_db.sqlite” όταν εκτελεστεί το αρχείο “create_admin.py”. Περισσότερα στην ενότητα που αφορά αυτό το αρχείο.

4.2.3 logs/

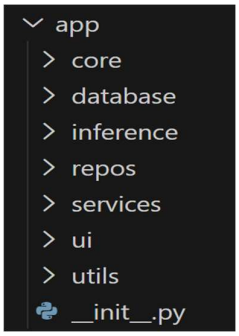
Ο φάκελος logs/ Περιέχει τα αρχεία καταγραφής – σφαλμάτων εφαρμογής , ελέγχου πρόσβασης στην εφαρμογή, ανιχνεύσεων οχημάτων (είτε είναι οχήματα που εμφανίζονται στο σύστημα ως «κλεμμένα ή αναζητούμενα» είτε είναι «καθαρά») (Σχήμα 4.3). Επίσης περιέχει κι ένα αρχείο “__init__.py”. (Περισσότερες πληροφορίες γι’ αυτό στην υποενότητα 4.3.9)



Σχήμα 4.3 Περιεχόμενα φακέλου logs/

4.3 app/

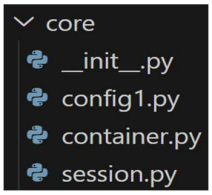
Ο φάκελος app/ περιέχει ουσιαστικά όλα εκείνα τα κομμάτια που συνθέτουν μια ολοκληρωμένη εφαρμογή (Σχήμα 4.4). Η διάρθρωσή τους δεν είναι τυχαία. Κάθε υποφάκελος περιέχει έναν αριθμό αρχείων που «συνθέτουν» την μεγάλη εικόνα με μόνο τα απαραίτητα στοιχεία (Need-to-know basis).



Σχήμα 4.4 Περιεχόμενα φακέλου app/

4.3.1 /core/

Στον υποφάκελο core/ περιέχονται τα ακόλουθα αρχεία (Σχήμα 4.5) : 1. “config1.py”, 2. “container.py”, 3. “session.py” , 4. “ __init__.py” (Περισσότερες πληροφορίες γι’ αυτό στην υποενότητα 4.3.9)



Σχήμα 4.5 Περιεχόμενα φακέλου core/

4.3.1.1 config1.py

Στην κορυφή του αρχείο γίνεται εισαγωγή της βιβλιοθήκης cv2 (opencv-python). Έπειτα αναθέτεται ένα CAMERA_ID που ενημερώνει το cv2 σε ποια πηγή να κοιτάξει για να λάβει το οπτικό σήμα και ορίζονται οι μεταβλητές FRAME_HEIGHT και FRAME_WIDTH που ορίζουν το μέγεθος της ανάλυσης που καταγράφει η κάμερα. Στα σχόλια δίπλα τους, είναι τιμές που υποδιπλασιάζουν τα μεγέθη – υποτετραπλασιάζουν την ανάλυση για να κερδίσουμε ταχύτητα. Στις επόμενες γραμμές δίνονται παράμετροι που θα χρησιμοποιηθούν στο inference από τα μοντέλα μας για το detection και το OCR όπως επίσης και η μια λίστα που περιέχει τα διαθέσιμα μοντέλα προς χρήση από τον χρήστη του συστήματος (GUI_FPS: πόσα Frames per second θέλουμε να «τρέξει» το GUI μας για την απεικόνιση του βίντεο στην οθόνη μας, DETECT_EVERY_N_FRAMES: κάθε πόσα Frames θέλουμε να γίνεται το detection – καθώς το Raspberry Pi 3B+ στερείται υπολογιστικής δύναμης -εάν αφήσουμε το detection να εκτελείται σε κάθε frame θα «παγώσει» η εφαρμογή αφού θα ζορίσουμε το υλικό , OCR_INTERVAL_SEC: κάθε πόσα δευτερόλεπτα θέλουμε να εκτελείται OCR καθώς έτσι «γλυτώνουμε» πόρους και αποφεύγουμε την ανίχνευση της ίδιας πινακίδας οχήματος που βρίσκετε στο πλάνο μας πολλές φορές, MAX_DETECTIONS_DRAW: μέγιστος αριθμός detections που θέλουμε να ζωγραφίσει το μοντέλο μας ταυτόχρονα- κρατώντας μόνο τις πιο «δυνατές» σε confidence , GREEK_CHARS: το σύνολο των αποδεκτών χαρακτήρων (ουσιαστικά κρατάμε μόνο αυτούς που εμφανίζονται σε ελληνικές πινακίδες κυκλοφορίας, DIGITS: τα ψηφία που αποδεχόμαστε, CONF_THRESHOLD : το κατώφλι στο confidence κάτω από το οποίο θα αγνοούμε τις ανιχνεύσεις- διατάζοντας το μοντέλο να απορρίπτει οτιδήποτε λιγότερο ως λάθος , NMS_THRESHOLD: το κατώφλι του NMS ώστε αν εντοπιστούν δύο ανιχνεύσεις που αλληλοκαλύπτονται σε αυτό το βαθμό να κρατάμε πάντα το καλύτερο αγνοώντας τα υπόλοιπα ,MODEL_INP: το μέγεθος στο οποίο θα μετατρέψουμε την

πηγή (εικόνα) πριν την περάσουμε ως είσοδο στο μοντέλο μας καθώς είναι εκπαιδευμένα σε συγκεκριμένες διαστάσεις , FAST_OCR_MODELS: η λίστα με τα μοντέλα και τέλος ένα μπλοκ κώδικα try-except για error handling στο οποίο λέμε στην βιβλιοθήκη OpenCV ότι μπορεί να χρησιμοποιήσει 2 threads από τον επεξεργαστή μας καθώς έχουμε σύνολο 4 πυρήνες. Με αυτό τον τρόπο αποφεύγουμε το «κόλλημα» της εφαρμογής μας (Σχήμα 4.6).

```
import cv2
# ================================
# CONFIGURATION FILE για το RPi 3B+
# ================================
CAMERA_ID = 0

# Capture video size - ιδιαίτερα σημαντικό
FRAME_WIDTH = 640 #320
FRAME_HEIGHT = 480 #240

# Display refresh (UI) and detection caden
GUI_FPS = 15
DETECT_EVERY_N_FRAMES = 3
OCR_INTERVAL_SEC = 0.45
MAX_DETECTIONS_DRAW = 3

# Greek plate whitelist / χαρακτήρες που επ
GREEK_CHARS = "ABEZHIKMNOPTYX"
DIGITS = "0123456789"

CONF_THRESHOLD = 0.45
NMS_THRESHOLD = 0.50
MODEL_INP = 320

FAST_OCR_MODELS = [      You, last week •
    "cct-xs-relu-v1-global-model",
    "cct-xs-v1-global-model",
    "cct-s-relu-v1-global-model",
    "cct-s-v1-global-model",
    "european-plates-mobile-vit-v2-model",
]
try:
    cv2.setNumThreads(2)
except:
    pass
```

Σχήμα 4.6 Περιεχόμενα αρχείου config1.py

4.3.1.2 container.py

Το αρχείο αυτό αποτελεί το σημείο της εφαρμογής που υλοποιείται το Dependency Injection. Αντί να φτιάχνουμε εκ νέου κάθε φορά που χρειάζεται αντικείμενα όπως είπαμε σε προηγούμενη ενότητα, τα «συνθέτουμε» εδώ μέσα και όπου χρειαστεί τα χρησιμοποιούμε. Στις πρώτες τρεις γραμμές του αρχείου γίνεται εισαγωγή κλάσεων από τρία διαφορετικά αρχεία (Σχήμα 4.7). Οι κλάσεις «InferencePipeline» και «DetectionService» από τα αντίστοιχα αρχεία “pipeline.py” και “detection_services.py”. Και τις δύο θα τις δούμε αναλυτικά σε επόμενες υποενότητες. Επίσης, εισάγουμε και το αρχείο “config1.py” ώστε να ξέρει το container ποιες ρυθμίσεις να εφαρμόσει στον μηχανισμό Inference.

def __init__(self): Είναι ο κατασκευαστής της κλάσης μας. Περιέχει όλο τον κώδικα που θα εκτελεστεί αυτόματα όταν ζητηθεί η δημιουργία ενός container. Εδώ ορίζεται ότι το μοντέλο που θα χρησιμοποιηθεί ως προεπιλογή είναι το πρώτο της λίστας μοντέλων του αρχείου ρυθμίσεών μας – ώστε να μην υπάρξει παρακάτω περίπτωση να αντιμετωπίσουμε Null εξαιρέσεις. Το ελέγχουμε επίσης κατά την εκτέλεση του κώδικα στο “gui_main.py”. Στις αμέσως επόμενες γραμμές έχουμε την υλοποίηση του DI:

self.pipeline = InferencePipeline(model_name) : Δημιουργούμε το αντικείμενό μας (μοντέλο) σύμφωνα με τις ρυθμίσεις μας και το αποθηκεύουμε στην μεταβλητή self.pipeline. Πλέον με την κλήση του container το μοντέλο έχει φορτωθεί στην μνήμη και μπορεί να προχωρήσει στο OCR.

self.detection_service = DetectionService() : Δημιουργούμε το αντικείμενο που τρέχει τα αιτήματά μας προς τη βάση δεδομένων και το αποθηκεύουμε στη μεταβλητή self.detection_service. Έτσι η σύνδεση γίνεται αυτόματα άμεσα και με ασφάλεια.

```python
from app.inference.pipeline import InferencePipeline
from app.services.detection_services import DetectionService
from app.core import config1

You, last week | 1 author (You)
class Container:
    def __init__(self):
        # 1. Αρχικοποίηση του AI Pipeline
        model_name = config1.FAST_OCR_MODELS[0]
        self.pipeline = InferencePipeline(model_name)

        # 2. Αρχικοποίηση του Service της βάσης
        self.detection_service = DetectionService()
```

Σχήμα 4.7 Περιεχόμενα αρχείου container.py

4.3.1.3 session.py

Στην ενότητα 3.4.3 είδαμε το session management και αναλύσαμε τον τρόπο λειτουργίας του. Στο APNR System όμως, χρησιμοποιούμε τον μηχανισμό διαχείρισης συνεδρίας καθαρά ως υποστήριξη του RBAC. Ο τρόπος λειτουργίας της κλάσης SessionManager που φαίνεται στο Σχήμα 4.8 είναι ο εξής: Η κλάση η ίδια συμπεριφέρεται ως μια «αποθήκη» πληροφοριών για τον τρέχοντα χρήστη. Οι εσωτερικές μεταβλητές "_user" και "_role" αρχικοποιούνται σε "None" καθώς με την έναρξη του προγράμματος δεν έχει συνδεθεί ακόμη κανένας χρήστης. Οι μεταβλητές αυτές θα χρησιμοποιηθούν ως "attributes" της κλάσης μας. Καθώς στην εφαρμογή μας υπάρχει μόνο ένας συνδεδεμένος χρήστης τη φορά και δεν πρόκειται να έχουμε ταυτόχρονα πολλαπλούς χρήστες χρησιμοποιούμε τον διακοσμητή "@classmethod" (με χρήση "cls" αντί για "self") για τις παρακάτω μεθόδους:

def set_user(cls, username, role) : Πρόκειται ουσιαστικά για τη λειτουργία του login. Παίρνει το username του χρήστη και το ρόλο του και τα περνάει στην ίδια την κλάση.

def get_current_user(cls) : επιστρέφει το username του συνδεδεμένου χρήστη.

def get_role(cls) : επιστρέφει τον ρόλο που έχει ο συνδεδεμένος χρήστης για να χρησιμοποιηθεί στο RBAC.

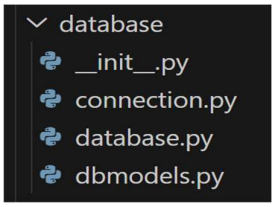
def clear(cls) : εκτελεί το "logout" του χρήστη. Στην παρούσα υλοποίηση καλείται μόνο κατά την έξοδο από το πρόγραμμα θα μπορούσαμε να το χαρακτηρίσουμε ως «περιττό» κώδικα.

```python
class SessionManager:
    _user = None
    _role = None

    @classmethod
    def set_user(cls, username, role):
        cls._user = username
        cls._role = role

    @classmethod
    def get_current_user(cls):
        return cls._user

    @classmethod
    def get_role(cls):
        return cls._role

    @classmethod
    def clear(cls):
        cls._user = None
        cls._role = None
```

Σχήμα 4.8 Περιεχόμενα αρχείου session.py

4.3.2 database/

Στον υποφάκελο database/ περιέχονται τα ακόλουθα αρχεία (Σχήμα 4.9) : 1. “connection.py”, 2. “database.py”, 3. “dbmodels.py” , 4. “ __init__.py” (Περισσότερες πληροφορίες γι’ αυτό στην υποενότητα 4.3.9)



Σχήμα 4.9 Περιεχόμενα φακέλου database/

4.3.2.1 connection.py

Το αρχείο “connection.py” είναι το κομμάτι της εφαρμογής που είναι υπεύθυνο για την επικοινωνία της εφαρμογής με την τοπική βάση δεδομένων μας. Αρχικά γίνεται εισαγωγή των κλάσεων “create_engine” και “sessionmaker” από την βιβλιοθήκη SQLAlchemy (Σχήμα 4.10). Πρόκειται για μια βιβλιοθήκη που λειτουργεί ως ενδιάμεσο στρώμα επικοινωνίας ανάμεσα στην βάση και των κώδικά μας. Το “engine” της SQLAlchemy είναι το σημείο εκκίνησης. Από εδώ γίνεται η διαχείριση της πραγματικής σύνδεσης. Το “session” η συνεδρία της εφαρμογής με τη βάση διαχειρίζεται από την κλάση “sessionmaker”. Για κάθε συναλλαγή που χρειάζεται να γίνει μεταξύ τους, μια νέα συνεδρία δημιουργείται , με τη βιβλιοθήκη να αναλαμβάνει όλη τη «βαριά» δουλειά κάνοντας την εφαρμογή πιο ασφαλή.

Επίσης εδώ εισάγονται και δύο κλάσεις από άλλα τοπικά αρχεία τα οποία θα δούμε αναλυτικά σε επόμενες ενότητες , η “Labyrinth” από το αρχείο “pathfinder.py” και η “Base” από το “dbmodels.py”.

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from app.utils.pathfinder import Labyrinth
from .dbmodels import Base
```

Σχήμα 4.10 Imports του αρχείου connection.py

Στην κλάση "DBConnectionManager" φτιάχνουμε αρχικά το "URL" της βάσης – έναν σύνδεσμο προς το τοπικό αρχείο , το οποίο παίρνει το σωστό μονοπάτι μέσω της βοηθητικής μεθόδου "get_db_path()" από την κλάση "Labyrinth". Έπειτα δημιουργούμε τη μηχανή (engine) με εξαιρετικά σημαντικό όρισμα το "check_same_thread": False , καθώς εξ' ορισμού η SQLite δεν επιτρέπει τις κλήσεις από διαφορετικά threads ενός προγράμματος ταυτόχρονα. Επειδή όμως εμείς έχουμε διαφορετικά threads να δουλεύουν ταυτόχρονα και χρειάζεται συναλλαγή με τη βάση σε κάποια από αυτά , είναι απαραίτητη η παράμετρος αυτή αλλιώς θα πάρουμε σφάλματα κατά τη διάρκεια της εκτέλεσης.

Αμέσως μετά το "sessionmaker" κατασκευάζει μια καινούρια «γραμμή παραγωγής συνεδριών» και το ενώνει με τη μηχανή που ήδη δημιουργήθηκε, βάζοντας τις εξής δύο παραμέτρους : "expire_on_commit=False" το οποίο σημαίνει ότι και μετά από ένα commit στη βάση μας μπορούμε να διαβάζουμε τα δεδομένα της χωρίς νέο αίτημα, και "autoflush=False" το οποίο αφήνει σε εμάς την επιλογή για το πότε θα γίνονται οι οποιεσδήποτε αλλαγές στη ΒΔ αντί να γίνονται αυτόματα και να αναλώνονται πόροι.

Επίσης περιέχονται δύο στατικές μέθοδοι (διακοσμητής «@staticmethod») (Σχήμα 4.11):

def setup_db(): Η οποία αρχικοποιεί τη ΒΔ , παίρνοντας τα μοντέλα των πινάκων μας από την κλάση "Base" και περνώντας τα στην μηχανή τα μετατρέπει σε πραγματικούς πίνακες.

def get_session(): Η οποία όταν καλείται δημιουργεί μια νέα συνεδρία με τη ΒΔ.

```
class DBConnectionManager:
    # Παίρνουμε το path της SQLite από το Pathfinder.py
    DB_URL = f"sqlite:///{Labyrinth.get_db_path()}"

    engine = create_engine(DB_URL, connect_args={"check_same_thread": False})
    SessionLocal = sessionmaker(bind=engine, expire_on_commit=False, autoflush=False)

    @staticmethod
    def setup_db():
        # Δημιουργούμε τους πίνακες στη βάση μας εάν δεν υπάρχουν.
        Base.metadata.create_all(DBConnectionManager.engine)

    @staticmethod
    def get_session():
        # Επιστρέφω ένα νέο session για να μπορέσουμε να κάνουμε queries στο police_db.sqlite
        return DBConnectionManager.SessionLocal()
```

Σχήμα 4.11 Κλάση DBConnectionManager

4.3.2.2 database.py

Στο αρχείο αυτό έχουμε την κλάση "DatabaseManager" η οποία ουσιαστικά είναι ο διαχειριστής της ΒΔ. Μέσω αυτής γίνονται οι κλήσεις των αποτελεσμάτων του inference προς τη βάση για να πάρουμε το τελικό αποτέλεσμα ως προς το εάν το όχημα που εντοπίστηκε είναι «καθαρό» ή όχι. Αρχικά έχουμε τα εξής imports (Σχήμα 4.12): Κλάσεις "StolenVehicle" ,"VehicleStatus" τις οποίες θα δούμε στην αμέσως επόμενη υποενότητα από το αρχείο "dbmodels.py", η κλάση "DBConnectionManager" που μόλις είδαμε από το αρχείο "connection.py" , τη βιβλιοθήκη "os" μέσω της οποίας το πρόγραμμα μπορεί

να αποκτήσει πληροφορίες του συστήματος και των αρχείων μας , καθώς επίσης και τη μέθοδο “log_error” από το αρχείο “logger.py” το οποίο θα δούμε σε μεταγενέστερη υποενότητα.

```
from app.database.dbmodels import StolenVehicle , VehicleStatus
from app.database.connection import DBConnectionManager
import os
from app.services.logger import log_error
```

Σχήμα 4.12 Imports του αρχείου database.py

Κατά την πρώτη αναφορά της κλάσης “DatabaseManager” αρχικοποιείται η βάση μας μέσω της μεθόδου **def __init__(self):** Εδώ καλείται η μέθοδος “DBConnectionManager.setup_db()” (Σχήμα 4.13)

```
class DatabaseManager:
    def __init__(self):
        # Κλήση της setup_db => Αρχικοποίηση της βάσης εάν δεν υπάρχει ήδη.
        DBConnectionManager.setup_db()
```

Σχήμα 4.13 ο “initializer” της κλάσης “DatabaseManager”

def check_plate(self , text): (Σχήμα 4.14) Είναι η μέθοδος η οποία δέχεται το κείμενο που έχει διαβάσει το OCR. Στην γραμμή 14 το κείμενο «καθαρίζεται» κρατώντας μόνο αλφαριθμητικά τα οποία μάλιστα μετατρέπει σε κεφαλαία ώστε ο έλεγχος στη βάση παρακάτω να γίνει σωστά. Εάν μετά το «καθάρισμα» δεν έχει μείνει αλφαριθμητικό τότε επιστρέφεται πινακίδα “None” , κατάσταση οχήματος “CLEAN” και το προεπιλεγμένο γκρι χρώμα για το status bar που θα δούμε παρακάτω στο UI (“gui_main.py”)- γραμμή 16. Αμέσως ανοίγουμε μια νέα συνεδρία με τη ΒΔ την οποία κλείνουμε για οικονομία πόρων αμέσως μόλις πάρουμε το αποτέλεσμά μας. Στις γραμμές 27-30 φτιάχνουμε ένα λεξικό όπου αναθέτονται χρώματα ανάλογα με την κατάσταση του οχήματος και επιστρέφονται (γραμμή 34): η πινακίδα κυκλοφορίας , η κατάσταση και το χρώμα του status bar- ανάλογα με την κατάσταση που αντιστοιχεί. Σε περίπτωση που το όχημα δεν βρεθεί στη βάση θεωρείται εξ ορισμού «καθαρό» με αντίστοιχα πεδία (γραμμή 36).

```python
    def check_plate(self, text):
        # Ελέγχουμε πρώτα αν το κείμενο της πινακίδας που αναγνωρίστηκε είναι έγκυρο.
        clean_text = ''.join(e for e in text if e.isalnum()).upper()
        if not clean_text:
            return None, "CLEAN", "#ecf0f1" # Default γκρι

        session = DBConnectionManager.get_session()
        # Ψάχνουμε στη βάση εάν το όχημα είναι καταχωρημένο ως STOLEN, ή WANTED.
        #  Εάν όχι, θεωρούμε ότι είναι CLEAN.
        vehicle = session.query(StolenVehicle).filter_by(plate_number=clean_text).first()
        session.close()

        if vehicle:
            # Δίνουμε χρώμα ανάλογα με το status του οχήματος στη βάση για να περάσει
            # ως alert στο UI παρακάτω
            colors = {
                VehicleStatus.STOLEN: "#c0392b", # Κόκκινο
                VehicleStatus.WANTED: "#d35400", # Πορτοκαλί
                VehicleStatus.CLEAN: "#27ae60"   # Πράσινο
            }
            # Το vehicle.status είναι  Enum object, οπότε το χρησιμοποιούμε ως key
            color = colors.get(vehicle.status, "#ecf0f1")
            return clean_text, vehicle.status.value, color

        return clean_text, "CLEAN", "#27ae60"
```

Σχήμα 4.14 database.check_plate()

def add_stolen_vehicle(self, plate, status_string): Με την κλήση αυτής της μεθόδου (από το "gui_main.py" που θα δούμε παρακάτω) προσθέτουμε μια πινακίδα ενός οχήματος ως νέα εγγραφή στη βάση μας (Σχήμα 4.15). Αρχικά ανοίγει μια νέα συνεδρία (γραμμή 40) κατά την οποία οι παράμετροι που δόθηκαν μέσω του UI αναθέτονται σε μεταβλητές (γραμμές 47-51). Έπειτα στην γραμμή 54 γίνεται η εισαγωγή της πινακίδας στη ΒΔ με τη χρήση της μεθόδου "merge(vehicle)". Έτσι, εάν το όχημα ΔΕΝ υπάρχει θα δημιουργηθεί μια νέα εγγραφή , ενώ αν το όχημα υπάρχει ήδη θα γίνει μια «ενημέρωση» της κατάστασής του αποφεύγοντας «διπλοεγγραφές» στη βάση μας. Με αυτόν τον τρόπο θα μπορούσαμε να προχωρήσουμε με τη χρήση της **ίδιας** μεθόδου και στον «αποχαρακτηρισμό» ενός οχήματος που ήταν «κλεμμένο» και ανευρέθηκε ως «καθαρό» ξεφεύγει όμως από το σκοπό της παρούσης. Στην γραμμή 55 επιχειρείται το "commit" της συναλλαγής στη βάση μας κι αν όλα πάνε καλά τότε επιστρέφεται ένα Boolean (True). Σε περίπτωση που υπάρξει αποτυχία για οποιοδήποτε λόγο, τότε γίνεται rollback στη βάση μας – ακύρωση όλης της διαδικασίας εγγραφής- ώστε να μην «χαλάσει». Επίσης, δυναμικά αναγνωρίζεται το όνομα του αρχείου, ο τύπος του σφάλματος και το ίδιο το σφάλμα, γίνεται καταγραφή στο αντίστοιχο log που θα δούμε παρακάτω και το μήνυμα σφάλματος ενημερώνει τον χρήστη ως alert window ώστε να καταλάβει ότι η ενέργεια απέτυχε (γραμμές 57-62). Τέλος, ανεξάρτητα από την επιτυχή ή ανεπιτυχή ενέργεια κλείνει η σύνδεση με τη βάση αποφεύγοντας φαινόμενα «διαρροής μνήμης» (γραμμή 66).

```
38      def add_stolen_vehicle(self, plate, status_string):
39          # Προσθήκη οχήματος που είναι STOLEN/WANTED στη βάση μας.
40          session = DBConnectionManager.get_session()
41          try:
42              # Διασφάλιση εγκυρότητας του status του οχήματος.
43              #  Στο combobox του UI ΔΕΝ έχουμε άλλη επιλογή,
44              # βεβαιωνόμαστε ότι σε περίπτωση σφάλματος, δε θα
45              # "χτυπήσει" η βάση ή δε θα κρασάρει η εφαρμογή.
46
47              status_enum = VehicleStatus(status_string.upper())
48
49              vehicle = StolenVehicle(
50                  plate_number=plate.upper(),
51                  status=status_enum
52              )
53
54              session.merge(vehicle)
55              session.commit()
56              return True
57          except Exception as e:
58              session.rollback()
59              file_name = os.path.basename(__file__)
60              error_type = type(e).__name__
61              log_error(error_type, file_name, str(e))
62              raise e
63
64
65          finally:
66              session.close()
```

Σχήμα 4.15 database.add_stolen_vehicle()

4.3.2.3 dbmodels.py

Στο αρχείο αυτό ορίζονται τα «μοντέλα» που θα χρησιμοποιηθούν από την SQLAlchemy για να φτιάξει τη ΒΔ μας με τους πίνακες που χρειαζόμαστε μέσα. Χρησιμοποιείται η τεχνική Object-Relational Mapping- (ORM) όπου ουσιαστικά για κάθε κλάση που έχουμε εδώ υλοποιείται ένας πίνακας στη ΒΔ και κατόπιν κάθε αντικείμενο που θα φτιάξουμε στον κώδικά μας και θα θελήσουμε να το αποθηκεύσουμε με τη βιβλιοθήκη μετατρέπεται σε νέα εγγραφή χωρίς πολύ κόπο και κυρίως χωρίς να χρειάζεται πουθενά κώδικας SQL. Στην αρχή του αρχείου έχουμε τα imports (Σχήμα 4.16). Εδώ περνάμε στο πρόγραμμα την ενσωματωμένη βιβλιοθήκη “enum” που εξυπηρετεί για τις κλειστές λίστες επιλογών, και μετά από την βιβλιοθήκη “sqlalchemy” εισάγουμε : “DeclarativeBase” : που είναι η «μητρική κλάση» αυτή από την οποία κληρονομούν όλες οι υπόλοιπες, “Mapped” :το οποίο μεταφράζει στο πρόγραμμά μας τον τύπο των δεδομένων ανάμεσα σε SQL και Python για να μπορούν να φτιαχτούν τόσο οι πίνακες αλλά και τα αντικείμενα παρακάτω και τέλος το “mapped_column” το οποίο δίνει οδηγίες προς τη βάση για τη δημιουργία των στηλών των πινάκων. Επιπλέον εισάγονται οι τύποι δεδομένων της, “String” και “Enum” – αποθηκευμένο ως “SQLEnum” για να μη γίνει κάποια σύγχυση με την enum της ενσωματωμένης βιβλιοθήκης που είδαμε στην αρχή.

```
import enum
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column
from sqlalchemy import String
from sqlalchemy import Enum as SQLEnum
```

Σχήμα 4.16 imports στο αρχείο dbmodels.py

Έπειτα, έχουμε τη δημιουργία της αρχικής κλάσης “Base” απ’ όπου θα δημιουργηθούν και όλες οι υπόλοιπες και τον ορισμό των enums που θα χρειαστούμε στα αντικείμενά μας παρακάτω καθώς αντικαθιστώντας την απλή εισαγωγή κειμένου με αυτά – εξαλείφουμε κάθε πιθανότητα σφαλμάτων εισαγωγής (Σχήμα 4.17):

```
11  class Base(DeclarativeBase):
12      pass
13
    You, last week | 1 author (You)
14  class UserRole(str , enum.Enum):
15      USER = "USER"
16      SUPERVISOR = "SUPERVISOR"
17      ADMIN = "ADMINISTRATOR"
18
    You, now | 1 author (You)
19  class VehicleStatus(str ,enum.Enum):
20      STOLEN = "STOLEN"
21      WANTED = "WANTED"
22      CLEAN = "CLEAN"
```

Σχήμα 4.17 Η «μητρική» κλάση “Base” και τα Enums του dbmodels.py

Γραμμή 14: ορίζουμε μια κλάση “UserRole” που μπορεί να έχει μόνο μια από τις προκαθορισμένες τιμές των γραμμών 15-17. Στις γραμμές 19-22 ομοίως ορίζουμε την κλάση “VehicleStatus” με παρόμοιο τρόπο. Κατόπιν προχωράμε στη δημιουργία δύο κλάσεων των μοντέλων μας, “User” και “StolenVehicle”. Και οι δύο κληρονομούν την κλάση “Base” , ορίζουν το όνομα του πίνακα που θα δημιουργηθεί στη ΒΔ -γραμμές 25 και 32 αντίστοιχα – καθώς και τα ονόματα των στηλών που θα περιέχει κάθε ένας από αυτούς. Ειδικά για το “password” (γραμμή 28) : εκχωρούμε ένα μέγεθος 255 χαρακτήρων – καθώς σε άλλη κλάση που θα δούμε αργότερα υλοποιούμε password hashing, απαγορεύοντας το πεδίο αυτό να είναι “null” – απαιτώντας ουσιαστικά να καταχωρηθεί κωδικός. Στις γραμμές 29 και 35 αντίστοιχα, οι δύο enum κλάσεις που είδαμε νωρίτερα περνούν σαν ορίσματα δίνοντας και προεπιλεγμένες τιμές – για να μην είναι null (Σχήμα 4.18).

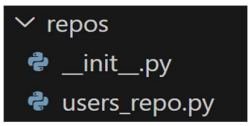
```
24  class User(Base):
25      __tablename__ = "users"
26      id: Mapped[int] = mapped_column(primary_key=True)
27      username: Mapped[str] = mapped_column(String(50), unique=True)
28      password: Mapped[str] = mapped_column(String(255), nullable=False) # Hashed
29      role: Mapped[UserRole] = mapped_column(SQLEnum(UserRole), default= UserRole.USER)
30
    You, last week | 1 author (You)
31  class StolenVehicle(Base):
32      __tablename__ = "stolen_vehicles"
33      plate_number: Mapped[str] = mapped_column(primary_key=True)
34      # Χρησιμοποιούμε SQLEnum για τη βάση και το VehicleStatus για την Python
35      status: Mapped[VehicleStatus] = mapped_column(SQLEnum(VehicleStatus), default=VehicleStatus.CLEAN)
```

Σχήμα 4.18 Κλάσεις “User” και “StolenVehicle” του dbmodels.py

4.3.3 repos/

Ο φάκελος repos/ περιέχει δύο αρχεία (Σχήμα 4.19) : 1. “users_repo.py” και 2. “ __init__.py” (Περισσότερες πληροφορίες γι’ αυτό στην υποενότητα 4.3.9).

Πρόκειται στην ουσία για έναν φάκελο που περιέχει τα αποθετήρια διαφόρων κλάσεων. Εδώ υλοποιείται μόνο ένα αποθετήριο που αφορά τους χρήστες της εφαρμογής με σκοπό το SoC και αποτελεί μια «χαλαρή» υλοποίηση σχεδιασμού “Repository” καθώς έχουμε κάποιες από τις ενέργειες που αφορούν χρήστες (CRUD) ενσωματωμένες στο αρχείο “user_services.py” και όχι εδώ. Εάν ακολουθούσαμε μια «αυστηρή» υλοποίηση, θα περνούσαμε όλες τις μεθόδους εδώ αφήνοντας το “user_services.py” ουσιαστικά να κάνει μόνο τον έλεγχο των δικαιωμάτων του χρήστη για τις αλλαγές που επιθυμεί να πραγματοποιήσει.



Σχήμα 4.19 Περιεχόμενα φακέλου repos/

4.3.3.1 users_repo.py

Το αποθετήριο χρησιμοποιείται μόνο για έναν σκοπό – τη διαδικασία της σύνδεσης ενός χρήστη στην εφαρμογή (login). Σε αυτό περιέχονται οι εισαγόμενες κλάσεις (DBConnectionManager, User) από τα αρχεία “connection.py” και “dbmodels.py” που είδαμε ήδη καθώς και οι μέθοδοι “log_access”, “log_error” από το αρχείο “logger.py” που θα δούμε σε μεταγενέστερη υποενότητα (Σχήμα 4.20):

```
1  from app.database.connection import DBConnectionManager
2  from app.database.dbmodels import User
3  from app.services.logger import log_access, log_error
```

Σχήμα 4.20 imports στο users_repo.py

Στο αρχείο αυτό περιέχεται η κλάση “UsersRepository” η οποία περιέχει μία στατική μέθοδο.

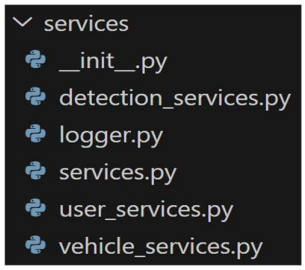
def get_user_by_username(username): δέχεται σαν όρισμα το username του χρήστη που προσπαθεί να συνδεθεί στην εφαρμογή (Σχήμα 4.21). Ξεκινά μια νέα συνεδρία με τη βάση και μέσα σε ένα block try- finally γίνεται το αίτημα σε αυτήν για την αναζήτηση του χρήστη(γραμμή13). Εάν αυτός βρεθεί τότε ο ρόλος που έχει δοθεί στον χρήστη επιστρέφεται μαζί με το username και τον κωδικό που βρίσκεται αποθηκευμένος στη βάση (κρυπτογραφημένος με hash) σαν ένα λεξικό και γίνεται η ανάλογη εγγραφή στο log_access(γραμμές 14-21). Σε περίπτωση που υπάρξει αποτυχία ή δεν βρεθεί χρήστης με αυτό το username , γίνονται εγγραφές στα log_access και log_error (γραμμές 22-24) κι επιστρέφει “None”. Τέλος , τερματίζεται η συνεδρία ώστε να αποφευχθούν φαινόμενα διαρροής μνήμης (γραμμή 27).

```python
class UsersRepository:
    @staticmethod
    def get_user_by_username(username):
        session = DBConnectionManager.get_session()
        try:
            user = session.query(User).filter_by(username=username).first()
            if user:
                actual_role = user.role.value
                log_access(username, actual_role, "Successful Login Attempt")
                return {
                    "username": user.username,
                    "password": user.password,
                    "role": actual_role
                }
            exception = f"User '{username}' not found in database."
            log_access(username, "UNKNOWN", f"Failed Login Attempt - {exception}")
            log_error("UserNotFound", "UsersRepository", exception)
            return None
        finally:
            session.close()
```

Σχήμα 4.21 users_repo.get_user_by_username()

4.3.4 services/

Μέσα στον φάκελο services/ περιέχονται τα εξής αρχεία (Σχήμα 4.22): 1. “detection_services.py”, 2. “logger.py”, 3. “services.py” , 4. “user_services.py”, 5. “vehicle_services.py”, 6. “ __init__.py” (Περισσότερες πληροφορίες γι’ αυτό στην υποενότητα 4.3.9)



Σχήμα 4.22 Περιεχόμενα φακέλου services/

4.3.4.1 detection_services.py

Στο αρχείο αυτό περιέχεται μόνο μία μέθοδος η σημαντικότητα της οποίας όμως είναι ύψιστη. Αρχικά, έχουμε την εισαγωγή της βιβλιοθήκης (Σχήμα 4.23) “re” η οποία είναι ενσωματωμένη στην Python και αφορά στη χρήση Regular Expressions (RE) καθώς επίσης και δύο κλάσεις από τα αντίστοιχα αρχεία “connection.py”, “dbmodels.py” η “DBConnectionManager” και η “StolenVehicle”. Με την αρχικοποίηση της κλάσης “DetectionService” που περιέχεται εδώ, δημιουργείται μια νέα συνεδρία με τη ΒΔ μας (γραμμή 13).

```python
import re
from app.database.connection import DBConnectionManager
from app.database.dbmodels import StolenVehicle

#region Comments ···
You, 10 seconds ago | 1 author (You)
class DetectionService:
    def __init__(self):
        self.session_factory = DBConnectionManager.get_session
```

Σχήμα 4.23 imports αρχείου detection_services.py

def process_detected_plate(sefl , raw_text): Εδώ γίνεται το «καθάρισμα» των detections (Σχήμα 4.24). Αρχικά το κείμενο που περάστηκε ως όρισμα κατά την κλήση της μεθόδου ελέγχεται ως προς την ορθότητα κρατώντας μόνο αλφαριθμητικά και μετατρέποντάς τα σε κεφαλαία (γραμμή 17). Στη γραμμή 25 θέτουμε την επιτρεπτή μορφή των πινακίδων που επιθυμούμε να αναγνωρίσουμε μέσω RegEx. Στην περίπτωσή μας – επιτρέπουμε πινακίδες κυκλοφορίας που αποτελούνται από τρείς **επιτρεπόμενους** χαρακτήρες ακολουθούμενους από έναν μέχρι και τέσσερις αριθμούς. Αυτό μας βοηθά να εντοπίσουμε ελληνικά οχήματα (αυτοκίνητα ή μοτοσυκλέτες). Έτσι φιλτράρονται οι αναγνωρίσεις και εάν δεν υπάρχει ταύτισή τους με το επιτρεπτό μοτίβο μας τότε επιστρέφεται “None” (γραμμές 25-27) γλυτώνοντας το υπολογιστικό κόστος αλλεπάλληλων «αχρείαστων» αιτημάτων προς τη βάση. Εάν το αποτέλεσμα όμως δεν είναι “None” αλλά μια έγκυρη πινακίδα κυκλοφορίας , τότε ξεκινά μια συνεδρία με τη βάση κι ακολουθεί ένα αίτημα αναζήτησης της πινακίδας στον πίνακα “stolen_vehicles”. Μόλις βρεθεί ένα αποτέλεσμα παύει η αναζήτηση κι επιστρέφεται ένα λεξικό με τον αριθμό κυκλοφορίας και την κατάσταση του οχήματος (γραμμές 30-36). Σε περίπτωση που δεν βρεθεί στη βάση όχημα με τέτοια πινακίδα τότε θεωρείται «καθαρό» και επιστρέφονται τα values της γραμμής 40. Τέλος, γίνεται κλείσιμο της συνεδρίας για να αποφευχθούν φαινόμενα διαρροής μνήμης (γραμμή 42).

```python
    def process_detected_plate(self, raw_text):
        # "Καθαρισμός" του κειμένου: Αφαιρούμε όλα τα μη-αλφαριθμητικά και μετατρέπουμε σε κεφαλαία
        clean_text = "".join(re.findall(r'[A-Z0-9]', raw_text.upper()))
#region  process_detected_plate() ---Comments ···
        pattern = r'^[ABEZHIKMNOPTYX]{3}\d{1,4}$'
        if not re.match(pattern, clean_text):
            return None # Δεν είναι έγκυρη ελληνική πινακίδα

        # Έλεγχος στη βάση
        session = self.session_factory()
        try:
            vehicle = session.query(StolenVehicle).filter_by(plate_number=clean_text).first()
            if vehicle:
                return {
                    "plate": clean_text,
                    "status": vehicle.status.value,

                }

            return {"plate": clean_text, "status": "CLEAN"}
        finally:
            session.close()
```

Σχήμα 4.24 detection_services.process_detected_plate()

4.3.4.2 logger.py

Στο αρχείο αυτό υλοποιούμε έναν μηχανισμό καταγραφών (logging) για την παρακολούθηση εντοπισμών, ελέγχου πρόσβασης στην εφαρμογή και διαχείρισης χρηστών καθώς επίσης και των σφαλμάτων που μπορεί να προκύψουν κατά την εκτέλεση της εφαρμογής. Είναι κάτι που ξεφεύγει από τα πλαίσια της εργασίας αυτής, πλην όμως πρόκειται για ένα **εξαιρετικά** κρίσιμο τμήμα μιας επαγγελματικής εφαρμογής. Αναλυτικότερα έχουμε την εισαγωγή της ενσωματωμένης βιβλιοθήκης "logging" της γλώσσας (Σχήμα 4.25) , της ενσωματωμένης βιβλιοθήκης "pathlib" που αναλαμβάνει να διαχειριστεί το «πρόβλημα» διαφορετικού ορισμού ενός μονοπατιού από λειτουργικό windows σε Linux και αντίστροφα, και μιας μεταβλητής από το βοηθητικό αρχείο "pathfinder.py". Στη γραμμή 9 ελέγχεται η ύπαρξη του φακέλου και εάν δεν υπάρχει δημιουργείται.

```
1  import logging
2  from pathlib import Path
3  from app.utils.pathfinder import BASE_DIR
4
5  #Εισάγουμε το BASE_DIR και πριν την δημιουργία του logger ελέγχουμε την ύπαρξη
6  # του φακέλου με τα logs.Στην περίπτωση που δεν υπάρχει - δημιουργείται εδώ.
7
8  LOGS_DIR = Path(BASE_DIR) / "logs"
9  LOGS_DIR.mkdir(exist_ok=True)
```

Σχήμα 4.25 imports logger.py

def _setup_logger(): Εδώ υλοποιούμε μια μέθοδο «δημιουργίας» loggers , αντίστοιχη των "Factories" που συναντάμε εγγενή στην γλώσσα C#. Με τη μέθοδο αυτή γράφουμε μόνο μία φορά τις απαιτήσεις της δημιουργίας ενός logger και παραμετρικά αρχικοποιούμε τα απαιτούμενα loggers παρακάτω. (Σχήμα 4.26) Στη μέθοδό μας περνάνε παραμετρικά δύο κείμενα -το όνομα του Logger και το όνομα του αρχείου- κι επιστρέφεται ένα αντικείμενο logging.Logger(γραμμές 11-13). Στη γραμμή 14 απενεργοποιείται το propagation καθώς χωρίς την απενεργοποίηση κάθε μήνυμα που θα δεχόταν ο κάθε logger που φτιάξαμε θα έστελνε τις «εγγραφές» του και στον κεντρικό logger του συστήματος και θα τα τύπωνε εις διπλούν στο τερματικό μας. Στη γραμμή 15 ελέγχουμε ότι ο logger έχει ρυθμιστεί καθώς αν δεν το ελέγξουμε και προσθέσουμε ξανά και ξανά τον ίδιο logger τότε θα έχουμε ως αποτέλεσμα πολλαπλές εγγραφές του κάθε μηνύματος στα αρχεία καταγραφών μας (μία φορά από κάθε logger που κλήθηκε). Στη γραμμή 16 ορίζουμε το ελάχιστο επίπεδο «σοβαρότητας» των μηνυμάτων που καταγράφονται. Με αυτή τη ρύθμιση παίρνουμε όλους τους τύπους εγγραφών αλλά όχι μηνύματα που αφορούν την αποσφαλμάτωση. Στις γραμμές 20-24 δίνεται η μορφοποίηση που θέλουμε να ακολουθεί η κάθε εγγραφή , δημιουργείται ο «διαχειριστής» του αρχείου εάν δεν υπάρχει ήδη και του ανατίθεται ο χειρισμός. Η μέθοδος επιστρέφει ολοκληρωμένο τον μηχανισμό logger που δημιουργήθηκε.

Έπειτα στις γραμμές 29-31 δημιουργούμε τρεις διαφορετικούς μηχανισμούς logging , έναν για τις αναγνωρίσεις πινακίδων -στον οποίο γίνεται και καταγραφή των νέων πινακίδων κυκλοφορίας που έχει καταχωρήσει ο χρήστης της εφαρμογής , έναν αποκλειστικά για την παρακολούθηση πρόσβασης στην εφαρμογή κι έναν για την καταγραφή των σφαλμάτων που τυχόν συμβαίνουν κατά την εκτέλεση.

```python
def _setup_logger(name: str, log_file: str) -> logging.Logger:
    #Δημιουργούμε ένα logger setup (generic).
    logger = logging.getLogger(name)
    logger.propagate = False
    if not logger.handlers:
        logger.setLevel(logging.INFO)

        # Ο formatter της βιβλιοθήκης βάζει το τρέχον timestamp όταν καλείται ο logger
        #  και μετά το μήνυμα που δέχεται στο αντίστοιχο αρχείο
        formatter = logging.Formatter('%(asctime)s, %(message)s', datefmt='%Y-%m-%d %H:%M:%S')

        file_handler = logging.FileHandler(LOGS_DIR / log_file, encoding='utf-8')
        file_handler.setFormatter(formatter)
        logger.addHandler(file_handler)

    return logger

# Αρχικοποίηση των 3 loggers όταν φορτώνεται το αρχείο
_det_logger = _setup_logger("detections", "detections.log")
_acc_logger = _setup_logger("access", "access.log")
_err_logger = _setup_logger("errors", "errors.log")
```

Σχήμα 4.26 Generic μέθοδος _setup_logger του logger.py

Παρακάτω (Σχήμα 4.27) , εμφανίζονται οι τέσσερις μέθοδοι που περιέχονται στο αρχείο logger.py.

def log_detection(plate: str, status: str, confidence: float, ocr_model: str): η συνάρτηση παίρνει ορίσματα τον αριθμό κυκλοφορίας που εντοπίστηκε, την κατάσταση του οχήματος, το confidence του εντοπισμού και το μοντέλο που χρησιμοποιήθηκε για το OCR κι έπειτα χρησιμοποιεί τον "_det_logger" για να περάσει το μορφοποιημένο κείμενο της γραμμής 40 στο αρχείο "detections.log"

def log_added_vehicle(plate: str, status: str): η μέθοδος παίρνει τα ορίσματα και με μορφοποιημένο κείμενο πραγματοποιεί νέα καταχώρηση στο "access.log" (γραμμή 44).

def log_access(username: str, role: str, action: str): κατ'αντιστοιχία με τις προηγούμενες μεθόδους περνιούνται οι παράμετροι κλήσης στο αρχείο "access.log" καταγράφοντας κάθε ενέργεια που αφορά χρήστες (CRUD) καθώς επίσης και τις απόπειρες εισόδου στο σύστημα (γραμμή 48).

def log_error(error_type: str, file_name: str, details: str= ""): με αυτή τη μέθοδο γίνονται οι καταγραφές σφαλμάτων στο αρχείο "error.log". Εάν υπάρχουν λεπτομέρειες σφάλματος – τότε προστίθενται στο μήνυμα καταγραφής (56-59)

```
33  def log_detection(plate: str, status: str, confidence: float, ocr_model: str):
34  >   #region Comments···
39      # Μορφοποίηση του confidence σε 2 δεκαδικά (π.χ. 0.95)
40      _det_logger.info(f"{plate}, {status}, {confidence:.2f}, {ocr_model}")
41
42  def log_added_vehicle(plate: str, status: str):
43      # Γράφει στο αρχείο access.log , την ΠΡΟΣΘΗΚΗ οχήματος στη βάση.
44      _acc_logger.info(f"{plate}, {status}, ADDED_TO_DB ")
45
46  def log_access(username: str, role: str, action: str):
47      # Γράφει στο access.log, ΚΑΘΕ ενέργεια που σχετίζεται με Χρήστες / απόπειρ
48      _acc_logger.info(f"{username}, {role}, {action}")
49
50  def log_error(error_type: str, file_name: str, details: str = ""):
51  >   #region···
56      msg = f"{error_type}, {file_name}"
57      if details:
58          msg += f", {details}"
59      _err_logger.error(msg)
```

Σχήμα 4.27 Μέθοδοι του αρχείου logger.py

4.3.4.3 services.py

Στο αρχείο αυτό πραγματοποιείται ο έλεγχος εισόδου του χρήστη. Πρόκειται για την υλοποίηση του service layer , το ενδιάμεσο επίπεδο «ασφαλείας» το οποίο αναλαμβάνει να διαχειριστεί δεδομένα από τη βάση δεδομένων, από το UI , καθώς επίσης και από την κρυπτογράφηση των κωδικών ασφαλείας.

Αρχικά εισάγονται (Σχήμα 4.28) οι κλάσεις “UsersRepository” και “SessionManager” τις οποίες είδαμε σε προγενέστερες υποενότητες από τα αρχεία που τις περιέχουν. Κατόπιν , στην κλάση AuthService υλοποιείται η μέθοδος του login. Επίσης εισάγεται η μέθοδος verify_password() από το αρχείο “hasher.py” το οποίο θα δούμε σε μεταγενέστερη ενότητα.

def login(username, password): Η μέθοδος παίρνει ως ορίσματα το username και το password- από το γραφικό περιβάλλον (συγκεκριμένα από το αρχείο “gui_login.py” που το δημιουργεί). Ανατίθεται ως user ο χρήστης τα στοιχεία του οποίου θα επιστραφούν από το UsersRepository και τη μέθοδό του get_user_by_username()(Γραμμή 10). Έπειτα γίνεται ο σημαντικός έλεγχος του κωδικού που έχει δώσει ο χρήστης με αυτόν που έχει καταχωρηθεί στο σύστημα στα στοιχεία αυτού του χρήστη. Εφόσον αυτά τα δύο ταιριάζουν , τότε αποθηκεύεται το username και ο ρόλος του χρήστη που συνδέθηκε στη συνεδρία (γραμμές 13-15). Εάν ο έλεγχος αποτύχει η μέθοδος επιστρέφει False στο σημείο που την κάλεσε.

Επίσης υλοποιείται η μέθοδος logout() στην οποία απλά σβήνει η συνεδρία (γραμμή 20).

```python
from app.repos.users_repo import UsersRepository
from app.utils.hasher import verify_password
from app.core.session import SessionManager

# Υπηρεσία που χειρίζεται το authentication των χρηστών.

You, last week | 1 author (You)
class AuthService:
    @staticmethod
    def login(username, password):
        user = UsersRepository.get_user_by_username(username)

        # Εδώ το user['password_hash'] περιέχει το user.password από τη βάση
        if user and verify_password(user['password'], password):
            SessionManager.set_user(user['username'], user['role'])
            return True
        return False

    @staticmethod
    def logout():
        SessionManager.clear()
```

Σχήμα 4.28 Περιεχόμενα αρχείου services.py

4.3.4.4 user_services.py

Στο αρχείο αυτό υλοποιούνται όλες οι λειτουργίες CRUD των χρηστών. Ξεκινώντας με τις εισαγωγές των κλάσεων και μεθόδων από τα αντίστοιχα αρχεία της εφαρμογής ξεχωρίζει μόνο η μέθοδος “hash_password” από το αρχείο “hasher.py” (Σχήμα 4.29).

```python
from app.database.connection import DBConnectionManager
from app.database.dbmodels import User, UserRole
from app.services.logger import log_access, log_error
from app.utils.hasher import hash_password
from app.core.session import SessionManager
> #region ···
You, 51 seconds ago | 1 author (You)
class UserManagementService:
    @staticmethod
    def require_admin():
        # 1. Αυστηρός έλεγχος δικαιωμάτων (Authorization Check)
        if SessionManager.get_role() != "ADMINISTRATOR":
            return False, "ACCESS DENIED : Μόνο ο Διαχειριστής μπορεί να εκτελέσει αυτή την ενέ
        return True, ""

    @staticmethod
    def get_manageable_users():
        session = DBConnectionManager.get_session()
        try:
            #Επιστρέφουμε όλους τους χρήστες που δεν είναι ADMINISTRATORS.
            users = session.query(User).filter(User.role != UserRole.ADMIN).all()
            return [{"id": u.id, "username": u.username, "role": u.role.value} for u in users]
        finally:
            session.close()
```

Σχήμα 4.29 user_services.py require_admin() και get_manageable_users()

def require_admin(): Στη μέθοδο αυτή πραγματοποιείται έλεγχος εάν ο χρήστης που είναι συνδεδεμένος στην παρούσα συνεδρία έχει ρόλο διαχειριστή. Εάν όχι, τότε η μέθοδος επιστρέφει False και μήνυμα άρνησης πρόσβασης στον χρήστη. Διαφορετικά επιστρέφει True (γραμμές 15-17).

def get_manageable_users():

Με την κλήση αυτής της μεθόδου, δημιουργείται μια νέα συνεδρία με τη ΒΔ και γίνεται ένα ερώτημα σε αυτή για να επιστραφούν όλοι οι χρήστες που ΔΕΝ έχουν ρόλο διαχειριστή – με την απάντηση να είναι διαμορφωμένη κατάλληλα (γραμμές 21-26). Τέλος κλείνει η συνεδρία για αποφυγή διαρροών μνήμης.

def create_user(new_username: str, new_password: str, role_to_assign: str):

Όπως γίνεται εύκολα κατανοητό από το όνομα της μεθόδου , εδώ υλοποιείται η δημιουργία ενός νέου χρήστη. Από προεπιλογή έχουμε δώσει το δικαίωμα επεξεργασίας χρηστών μόνο σε χρήστες με δικαιώματα διαχειριστή(Σχήμα 4.30). Αυτός ο έλεγχος πραγματοποιείται στις γραμμές 33-36 κι αν όντως ο ρόλος του χρήστη είναι διαφορετικός από του διαχειριστή παίρνει μήνυμα άρνησης πρόσβασης που περιγράφεται στη γραμμή 16. Στην περίπτωση που ο χρήστης έχει ρόλο διαχειριστή, μια νέα συνεδρία ανοίγει με τη ΒΔ και γίνεται έλεγχος εάν το username που έχει δώσει ο διαχειριστής χρησιμοποιείται ήδη ή όχι (γραμμές 38-42). Εάν δεν χρησιμοποιείται ο ρόλος που επέλεξε ο διαχειριστής επιλέγεται ως ρόλος του νέου χρήστη και ο κωδικός που έδωσε μετατρέπεται σε hashed_password με την κλήση της ομώνυμης μεθόδου (γραμμές 44-45). Ένας νέος χρήστης δημιουργείται με αυτά τα attributes (γραμμές 47-51) και ύστερα προστίθεται στη ΒΔ ενώ η συνεδρία οριστικοποιεί την εγγραφή και γίνεται καταγραφή στο αντίστοιχο log(γραμμές 47-56).Σε περίπτωση σφάλματος γίνεται επαναφορά στη βάση μας , καταγραφή του σφάλματος και μετά κλείνει η συνεδρία για αποφυγή διαρροών μνήμης(γραμμές 58-64).

```python
    def create_user(new_username: str, new_password: str, role_to_assign: str):
        # εάν όχι, τότε αρνούμαστε την εκτέλεση ενέργειας και καταγράφουμε στο log.
        is_admin, message = UserManagementService.require_admin()
        if not is_admin:
            log_error("ACCESS DENIED", "UserManagementService", f"Ο χρήστης {SessionManager.get_current_user()}
            return False, message

        session = DBConnectionManager.get_session()
        try:
            existing_user = session.query(User).filter_by(username=new_username).first()
            if existing_user:
                return False, f"Το όνομα χρήστη '{new_username}' χρησιμοποιείται ήδη."

            role_enum = UserRole(role_to_assign.upper())
            hashed_pw = hash_password(new_password)

            new_user = User(
                username=new_username,
                password=hashed_pw,
                role=role_enum
            )
            session.add(new_user)
            session.commit()
            # Καταγραφή της επιτυχούς ενέργειας
            log_access(SessionManager.get_current_user(), SessionManager.get_role(), f"Επιτυχής δημιουργία λογα
            return True, f"Ο λογαριασμός για τον χρήστη '{new_username}' με ρόλο '{role_enum.value}' δημιουργήθη

        except Exception as e:
            session.rollback()
            # Καταγραφή της αποτυχίας - λεπτομέρειες σφάλματος
            log_error(type(e).__name__, "UserManagementService", str(e))
            return False, f"Σφάλμα κατά την αποθήκευση στη βάση: Λάθος Ρόλος ή Δεδομένα. ({str(e)})"
        finally:
            session.close()
```

Σχήμα 4.30 create_user()

def delete_user(username): Η μέθοδος υλοποιεί τη διαγραφή ενός χρήστη από το σύστημα (Σχήμα 4.31). Όπως και με την δημιουργία , ο μόνος που μπορεί να εκτελέσει αυτές τις ενέργειες είναι ο διαχειριστής. Οπότε κατ' αντιστοιχία με την προηγούμενη μέθοδο , γίνεται ο έλεγχος δικαιωμάτων στις γραμμές 70- 73, κι εφόσον δεν έχει δικαιώματα διαχειριστή απαγορεύει την πρόσβαση στην ενέργεια. Στην περίπτωση που είναι διαχειριστής, νέα συνεδρία δημιουργείται αναζητείται ο χρήστης προς διαγραφή στη βάση με κριτήριο το "user_id" το οποίο παίρνουμε από το UI κι εφόσον βρεθεί πραγματοποιείται η διαγραφή του με τις αντίστοιχες εγγραφές στα logs. Τέλος, όπως πάντα κλείνει η συνεδρία για αποφυγή διαρροής μνήμης.

```python
    @staticmethod
    def delete_user(user_id: str):
        # Διαγραφή χρήστη με βάση το user_id του.
        is_admin, message = UserManagementService.require_admin()
        if not is_admin:
            log_error("ACCESS DENIED", "UserManagementService", f"Ο χρήστης {SessionManager.get_current_user()}
            return False, message

        session = DBConnectionManager.get_session()
        try:
            user_to_delete = session.query(User).filter_by(id=user_id).first()
            if  user_to_delete:
                session.delete(user_to_delete)
                session.commit()
                log_access(SessionManager.get_current_user(), SessionManager.get_role(), f"USER DELETED, ο χρήσ
                return True, f"Ο χρήστης με ID: {user_id} διαγράφηκε επιτυχώς."
            log_error("FailedDelete", "UserManagementService", f"FAILED DELETE ATTEMPT , ο χρήστης με ID: {user
            return False, f"Ο χρήστης με ID: {user_id} δεν βρέθηκε."
        finally:
            session.close()
```

Σχήμα 4.31 delete_user()

def update_user(user_id: str, new_password: str , new_role: str): Η μέθοδος υλοποιεί την ενημέρωση στοιχείων ενός χρήστη (Σχήμα 4.32). Η μόνη διαφορά από την δημιουργία του είναι ότι αφού επιλεχθεί ο χρήστης και ο διαχειριστής προχωρήσει στην τροποποίηση των στοιχείων του τότε : εάν ο διαχειριστής καταχωρήσει νέο κωδικό (γραμμή 100) – αυτός θα τροποποιήσει τον παλαιό. Εάν δεν καταχωρήσει κωδικό και αλλάξει μόνο τον ρόλο του τότε ο παλαιός κωδικός συνεχίζει να ισχύει. Σε περίπτωση που υπάρξει σφάλμα στην εκτέλεση τότε γίνεται επαναφορά χωρίς να τροποποιηθεί η βάση και στο τέλος τερματίζεται η συνεδρία για αποφυγή διαρροών μνήμης.

```
88      @staticmethod
89      def update_user(user_id: str, new_password: str, new_role: str):
90          # Ενημέρωση στοιχείων χρήστη - Αλλαγή κωδικού και / ή ρόλου. Απαιτείται ομοίως ρόλος Admin για εκτέλεση
91          is_admin, message = UserManagementService.require_admin()
92          if not is_admin:
93              log_error("ACCESS DENIED", "UserManagementService", f"Ο χρήστης {SessionManager.get_current_user()}
94              return False, message
95
96          session = DBConnectionManager.get_session()
97          try:
98              user_to_update = session.query(User).filter_by(id=user_id).first()
99              if user_to_update:
100                 if new_password: # Αν δόθηκε νέος κωδικός
101                     user_to_update.password = hash_password(new_password)
102
103                 user_to_update.role = UserRole(new_role.upper())
104
105                 session.commit()
106                 log_access(SessionManager.get_current_user(), SessionManager.get_role(), f"USER UPDATED, ο χρήστ
107                 return True, f"Ο χρήστης με ID: {user_id} ενημερώθηκε επιτυχώς."
108
109             log_error("FAILED USER UPDATE", "UserManagementService", f"FAILED UPDATE ATTEMPT, ο χρήστης με ID: 
110             return False, f"Ο χρήστης με ID: {user_id} δεν βρέθηκε."
111         except Exception as e:
112             session.rollback()
113             log_error(type(e).__name__, "UserManagementService", str(e))
114             return False, f"Σφάλμα κατά την ενημέρωση στη βάση: Λάθος Ρόλος ή Δεδομένα. ({str(e)})"
115         finally:
116             session.close()
```

Σχήμα 4.32 update_user()

4.3.4.5 vehicle_services.py

Στο αρχείο αυτό έχουμε την υλοποίηση μιας υπηρεσίας που μόνος της σκοπός είναι οι καταχωρήσεις οχημάτων στη βάση δεδομένων. Καθώς είναι ενδιάμεσο «στρώμα» κρατά απομονωμένη τη βάση και το UI και ολοκληρώνει την εργασία συγκεντρώνοντας τους πόρους που χρειάζεται. Πρέπει να επισημάνουμε ότι εάν θέλαμε να προχωρήσουμε σε πλήρη CRUD πινακίδων κυκλοφορίας οχημάτων θα έπρεπε κατ' αντιστοιχία κι εδώ να υλοποιήσουμε την υπηρεσία όπως στη διαχείριση χρηστών. Καθώς όμως ξεφεύγει από το σκοπό της παρούσης δεν υλοποιήθηκε. Εισάγονται η κλάση "DatabaseManager" από το αρχείο "database.py" και η μέθοδος log_added_vehicle από το αρχείο logger.py (Σχήμα 4.33).

def add_stolen_vehicle(plate: str, status: str) : Η μέθοδος επιστρέφει μια πλειάδα(μια Boolean μεταβλητή και ένα κείμενο. Αρχικά δημιουργούμε ένα νέο αντικείμενο DatabaseManager για να μπορέσουμε να προχωρήσουμε στη χρήση της μεθόδου της κλάσης , την "add_stolen_vehicle()" του αρχείου "database.py". Ο λόγος που υλοποιούμε έτσι τη μέθοδο είναι απλός αλλά εξαιρετικά σημαντικός: εάν η βάση μας έχει διαγραφεί ή αν είναι η πρώτη εκτέλεση του προγράμματος τότε έτσι διασφαλίζουμε τη σωστή δημιουργία της πριν την εισαγωγή του κλεμμένου οχήματος. Με αυτόν τον τρόπο δε θα «κρασάρει» η εφαρμογή μας. Έτσι το UI πέρασε τα στοιχεία του οχήματος στο επίπεδο

υπηρεσίας που βρίσκεται το αρχείο αυτό και κατόπιν περνάνε από εδώ μόνο στο Data Access Layer – DAL σαν ορίσματα οι απολύτως απαραίτητες πληροφορίες. Ανάλογα με το αποτέλεσμα της ενέργειας δίνεται και το αντίστοιχη ανατροφοδότηση στον χρήστη.

```
1  from app.database.database import DatabaseManager
2  from app.services.logger import log_added_vehicle
3
   You, yesterday | 1 author (You)
4  class VehicleService:
5      @staticmethod
6      def add_stolen_vehicle(plate: str, status: str) -> tuple[bool, str]:
7          # Υπηρεσία που αναλαμβάνει την προσθήκη ενός νέου οχήματος στη ΒΔ ως "STOLEN/WANTED"
8          try:
9              db_manager = DatabaseManager()
10             db_manager.add_stolen_vehicle(plate, status)
11             log_added_vehicle(plate, status)
12             return True, f"Το όχημα με πινακίδα {plate} προστέθηκε επιτυχώς!"
13         except Exception as e:
14             return False, f"Σφάλμα κατά την προσθήκη του οχήματος: {str(e)}"
```

Σχήμα 4.33 add_stolen_vehicle

4.3.5 utils/

Μέσα στον φάκελο utils/ περιέχονται αρχεία τα οποία έχουν υποστηρικτικό ρόλο στην υλοποίηση της εφαρμογής(Σχήμα 4.34). Αυτά είναι : 1. “hasher.py”, 2. “helpers.py” , 3. “pathfinder.py” , 4. “ __init__.py” (Περισσότερες πληροφορίες γι’ αυτό στην υποενότητα 4.3.9)

Σχήμα 4.34 Περιεχόμενα φακέλου utils/

4.3.5.1 hasher.py

Το αρχείο hasher είναι το σημείο στο οποίο στηρίζεται η εφαρμογή για την επιτυχή ταυτοποίηση ενός χρήστη. Κι ενώ η ταυτοποίηση δεν γίνεται εδώ , το γεγονός ότι η διαχείριση των κωδικών των χρηστών γίνεται με βάση τις μεθόδους που υλοποιούνται εδώ σημαίνει πολλά. Η μοναδική βιβλιοθήκη που εισάγεται στην αρχή του αρχείου (Σχήμα 4.35) η bcrypt αποτελεί ένα βιομηχανικό πρότυπο για εφαρμογές που θέτουν ψηλά τον πήχη στο κομμάτι της ασφάλειας. Ο λόγος είναι ότι η βιβλιοθήκη αυτή έχει ενσωματωμένη τη χρήση “salting” ώστε να μη βασιζόμαστε μόνο στο “hashing” του κωδικού.

def hash_password(password: str): Η μέθοδος παίρνει τον κωδικό που παρέχεται , τον περνάει από διαδικασία μετατροπής σε byte (γραμμή 8), τον «αλατίζει» αυτόματα (τον εμπλουτίζει με τυχαίους χαρακτήρες) και μετά προχωράει στη διαδικασία του hash (γραμμές 11-12). Τέλος επιστρέφει ένα σύνολο από χαρακτήρες (αφού έχει μετατρέψει τα τελικά bytes σε chars).

def verify_password(stored_hash: str, provided_password: str): Αρχικά γίνεται έλεγχος (γραμμές 19-20) εάν έχει δοθεί κωδικός από τον χρήστη κι αν έχει βρεθεί ο κωδικός που αντιστοιχεί για αυτόν τον χρήστη στη βάση δεδομένων μας. Εάν ένα από τα δύο δεν υπάρχει (κενό) τότε επιστρέφεται False από τη μέθοδο (απόρριψη εισόδου στην εφαρμογή – θα το δούμε στο “gui_login.py”). Διαφορετικά και

οι δύο παράμετροι μετατρέπονται σε bytes(γραμμές 22-23). Σε μία και μόνο γραμμή κώδικα φαίνεται η δύναμη της βιβλιοθήκης: στη γραμμή 27, γίνεται ο έλεγχος των δύο κωδικών σε μορφή byte πια. Εάν υπάρξει επιτυχία επιστρέφεται True αφήνοντας περαιτέρω τον έλεγχο να προχωρήσει. Σε περίπτωση που οι κωδικοί δεν ταυτίζονται ή υπάρχει σφάλμα τότε επιστρέφεται False.

```
1   import bcrypt
2   #region Comments···
6   def hash_password(password: str) -> str:
7       # Κάνουμε encode το password σε bytes- απαραίτητο για να λειτουργήσει το bcrypt.
8       password_bytes = password.encode('utf-8')
9
10      # Δημιουργούμε το salt και το hash
11      salt = bcrypt.gensalt()
12      hashed_bytes = bcrypt.hashpw(password_bytes, salt)
13
14      # Επιστρέφουμε string για τη συναλλαγή- καλύτερη διαχείριση στη ΒΔ μας.
15      return hashed_bytes.decode('utf-8')
16
17  def verify_password(stored_hash: str, provided_password: str) -> bool:
18      # Μετατρέπουμε το provided_password και το stored_hash σε bytes για να γίνει σύγκριση.
19      if not stored_hash or not provided_password:
20          return False  # Αν κάποιο από τα δύο είναι κενό, δεν μπορεί να ταιριάξει -ΑΠΟΡΡΙΨΗ
21
22      provided_password_bytes = provided_password.encode('utf-8')
23      stored_hash_bytes = stored_hash.encode('utf-8')
24
25      try:
26          # Η checkpw ελέγχει αυτόματα το salt και το hash
27          return bcrypt.checkpw(provided_password_bytes, stored_hash_bytes)
28      except ValueError:
29          # Σε περίπτωση που το stored_hash δεν είναι έγκυρο hash, απορρίπτουμε την προσπάθεια
30          return False
```

Σχήμα 4.35 hash_password και verify_password του hasher.py

4.3.5.2 helpers.py

Στο αρχείο περιέχονται βοηθητικές συναρτήσεις (Σχήμα 4.36). Αρχικά εισάγονται οι βιβλιοθήκες “time” και “cv2”.

def now_ts(): Συνάρτηση που όταν καλείται επιστρέφει μια χρονοσφραγίδα μορφοποιημένη σύμφωνα με τα περιεχόμενα της γραμμής 8.

def clamp(v , lo, hi): Παίρνει ως ορίσματα 3 αριθμούς. Πρόκειται για μαθηματική συνάρτηση η οποία περιορίζει τις τιμές που μπορεί να πάρει η τιμή “v” αυστηρά στο διάστημα που ορίζεται από τα “lo” και “hi”. Εξαιρετικά χρήσιμη κατά το Inference, καθώς εάν εντοπιστεί από το μοντέλο μας μια πινακίδα στα όρια της εικόνας, εάν το bounding box πάρει τιμή αρνητική το OpenCV θα κρασάρει. Η συνάρτησή μας βεβαιώνει ότι το bbox δεν θα βγει ποτέ εκτός ορίων.

def letterbox(image, new_shape=(320, 320), color=(114, 114, 114)): Η μέθοδος αποτελεί μια συνάρτηση προετοιμασίας για τα μοντέλα μας. Καθώς η εικόνα που περιμένουν τα μοντέλα μας πρέπει να είναι αυστηρά τετράγωνη με συγκεκριμένες διαστάσεις για να μπορέσουμε να μετατρέψουμε την εικόνα που έρχεται από την πηγή μας χωρίς να χάσουμε πληροφορία ή να την παραμορφώσουμε θα πρέπει να γίνουν κάποιες ενέργειες. Αυτή η μέθοδος τις πραγματοποιεί. Στη γραμμή 16 παίρνουμε τις διαστάσεις της εικόνας μας (αρχικές). Φτιάχνουμε νέο σχήμα στις διαστάσεις που έχουμε κατά την κλήση της συνάρτησης – οποίες εάν δεν αλλάξουν από το αρχείο

“config1.py” θα είναι (320 , 320)- γραμμή 17. Βρίσκεται ο μικρότερος λόγος ανάμεσα στα δύο κλάσματα «νέο πλάτος /πλάτος» και «νέο μήκος / μήκος» ώστε να μπορούν να χωράνε παρακάτω οι εικόνες στο νέο πλαίσιο χωρίς απώλεια πληροφορίας και κατόπιν γίνεται μια στρογγυλοποίηση στα νέα πλάτη και ύψη (γραμμές 19-20). Στην γραμμή 22 γίνεται μια αλλαγή μεγέθους της εικόνας με τη χρήση της μεθόδου “cv2.INTER_LINEAR”. Στις γραμμές 24-25 υπολογίζεται μαθηματικά πόσα εικονοστοιχεία λείπουν για να φτάσουμε τον στόχο του μοντέλου μας. Γίνεται ακέραια διαίρεση ώστε να μοιραστεί το κενό του πλάτους μας και του ύψους μας στη μέση στις γραμμές 24-29 και κατόπιν με την εντολή της γραμμής 31 «ζωγραφίζεται» το περίγραμμα που υπολείπεται με σταθερό το χρώμα που έχει οριστεί παραμετρικά κι επιστρέφεται η εικόνα **και** η αναλογία μετατροπής μαζί με το πόσο “padding” έχει προστεθεί ώστε παρακάτω στο pipeline να γίνουν άμεσα οι πράξεις που χρειάζεται.

```
1   import time
2   import cv2
3
4   #Βοηθητικές μέθοδοι
5
6   def now_ts():
7       # Δημιουργία timestamp
8       return time.strftime('%Y-%m-%d %H:%M:%S')
9
10  def clamp(v, lo, hi):
11      # Περιορισμός τιμής v στο διάστημα [lo, hi]
12      return max(lo, min(hi, v))
13
14  def letterbox(image, new_shape=(320, 320), color=(114, 114, 114)):
15      # Προσαρμογή μεγέθους της εικόνας /  φορμάρισμα της εικόνας με padding
16      h, w = image.shape[:2]
17      new_w, new_h = new_shape
18
19      r = min(new_w / w, new_h / h)
20      nw, nh = int(round(w * r)), int(round(h * r))
21
22      img_resized = cv2.resize(image, (nw, nh), interpolation=cv2.INTER_LINEAR)
23
24      dw = new_w - nw
25      dh = new_h - nh
26      left = dw // 2
27      right = dw - left
28      top = dh // 2
29      bottom = dh - top
30
31      img_lb = cv2.copyMakeBorder(img_resized, top, bottom, left, right,
32                                  cv2.BORDER_CONSTANT, value=color)
33      return img_lb, r, (left, top)
```

Σχήμα 4.36 Συναρτήσεις του αρχείου helpers.py

def preprocess_for_ocr(roi_bgr): Στην συνάρτηση αυτή γίνεται μια προετοιμασία της εικόνας που έχει περικοπεί από το μοντέλο εντοπισμού μας ώστε το OCR μοντέλο μας να μπορέσει να διαβάσει τα γράμματα της πινακίδας κυκλοφορίας χωρίς να γίνονται λάθη (Σχήμα 4.37).

Αρχικά στις γραμμές 37-38 ελέγχεται αν η περιοχή ενδιαφέροντος που έχει δοθεί σαν όρισμα είναι None ή κενή. Εάν ναι, τερματίζεται εδώ η προεπεξεργασία. Διαφορετικά: γραμμή 39: μετατρέπεται η έγχρωμη εικόνα σε ασπρόμαυρή. Γραμμή 40: μεγεθύνεται κατά 2.5 φορές το ROI καθώς τα μικροσκοπικά γράμματα δυσκολεύουν πολύ τα μοντέλα στην ταξινόμηση , και με τη μέθοδο “cv2.INTER_CUBIC” «γεμίζονται» τα κενά pixels που προκύπτουν από τον αλγόριθμο. Γραμμή 41: εφαρμόζεται ένα φίλτρο το οποίο αφαιρεί τον «θόρυβο» που έχει εισαχθεί από την κάμερα κρατώντας παράλληλα τις ακμές των γραμμάτων που περιέχονται αιχμηρές. Γραμμή 42: Εξισορροπεί την αντίθεση της εικόνας – καθαρίζοντάς την από το φόντο – τα γράμματα ξεχωρίζουν έντονα. Τέλος, στις γραμμές 43-44 , γίνεται η πλήρης μετατροπή της εικόνας σε ασπρόμαυρη. Ο αλγόριθμος “cv2.THRESH_OTSU” βρίσκει

εξαιρετικά το κατώφλι της φωτεινότητας για κάθε καρέ κάτω από το οποίο ότι βρίσκεται θα «ζωγραφιστεί» μαύρο και πάνω από αυτό λευκό. Σημαντικό: η συνάρτηση "cv2.threshold()" επιστρέφει δύο τιμές- την τιμή του «κατωφλιού» που έχει επιλεγεί και μετά την ίδια την επεξεργασμένη εικόνα. Καθώς δεν χρειαζόμαστε το κατώφλι την τιμή του την περνάμε στην προσωρινή μεταβλητή «_».

```
35  def preprocess_for_ocr(roi_bgr):
36      # Προ-επεξεργασία του Region of Interest (ROI) για τη χρήση του OCR.
37      if roi_bgr is None or roi_bgr.size == 0:
38          return None
39      gray = cv2.cvtColor(roi_bgr, cv2.COLOR_BGR2GRAY)
40      gray = cv2.resize(gray, None, fx=2.5, fy=2.5, interpolation=cv2.INTER_CUBIC)
41      gray = cv2.bilateralFilter(gray, 7, 50, 50)
42      gray = cv2.equalizeHist(gray)
43      _, th = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
44      return th
```

Σχήμα 4.37 preprocess_for_ocr()

4.3.5.3 pathfinder.py

Στο αρχείο αυτό περιέχονται διαδρομές αρχείων που χρειάζονται στην εφαρμογή και κάποιες μέθοδοι που τις επιστρέφουν (Σχήμα 4.38). Στην αρχή εισάγεται η βιβλιοθήκη "os" η οποία παρέχει τη δυνατότητα χειρισμού των διαδρομών. Κατόπιν, στις γραμμές 3-11 δημιουργούνται κάποιες μεταβλητές που αποθηκεύουν σημαντικές διαδρομές του προγράμματος.

def get_model_detector(): η μέθοδος επιστρέφει το μονοπάτι στο οποίο βρίσκεται το μοντέλο που χρησιμοποιείται στο pipeline για τον εντοπισμό πινακίδων.

def get_model_ocr(filename: str): ομοίως, η μέθοδος επιστρέφει τη διαδρομή του αρχείου που έχει περαστεί ως παράμετρος κατά την κλήση της.

```python
import os

CURRENT_DIR = os.path.dirname(os.path.abspath(__file__))
BASE_DIR = os.path.abspath(os.path.join(CURRENT_DIR, '..','..'))  # Μετάβαση σε επίπεδο root
MODEL_DETECTOR_PATH = os.path.join(BASE_DIR,   'models', 'license_plate_detector', 'license-plate-finet
MODEL_OCR_FOLDER_PATH = os.path.join(BASE_DIR, 'models', 'fast-plate-ocr')
DB_PATH = os.path.join(BASE_DIR, 'db','police_db.sqlite')
RESULTS_FILE_PATH = os.path.join(BASE_DIR,  'logs',  'detections.log')
ACCESS_LOG_FILE_PATH = os.path.join(BASE_DIR, 'logs', 'access.log')
ERROR_LOG_FILE_PATH = os.path.join(BASE_DIR, 'logs', 'errors.log')
LOGS_DIR_PATH = os.path.join(BASE_DIR, 'logs')
#region Comments···
You, 2 minutes ago | 1 author (You)
class Labyrinth:

    @staticmethod
    def get_model_detector() -> str:
        # Επιστρέφει το path του μοντέλου ανίχνευσης πινακίδων.
        target_path = MODEL_DETECTOR_PATH
        if not os.path.exists(target_path):
            raise FileNotFoundError(f"Detector model not found at path: {target_path}")
        return str(target_path)

    @staticmethod
    def get_model_ocr(filename : str) -> str:
        #Επιστρέφει το path ενός μοντέλου OCR - ελέγχοντας την ύπαρξή του.
        target_path = os.path.join(MODEL_OCR_FOLDER_PATH, filename)
        if not os.path.exists(target_path):
            raise FileNotFoundError(f"Model file '{filename}' not found in '{MODEL_OCR_FOLDER_PATH}'")
        return str(target_path)
```

Σχήμα 4.38 get_model_detector() & get_model_ocr()

def get_db_path(db_name): Η μέθοδος επιστρέφει τη διαδρομή στην οποία βρίσκεται η ΒΔ της εφαρμογής μας (Σχήμα 4.39). Αυτό που διαφοροποιεί αυτή τη μέθοδο από τις προηγούμενες είναι ότι στις γραμμές 41,42 γίνεται έλεγχος της ύπαρξης του γονικού φακέλου του αρχείου της βάσης και στην περίπτωση που αυτός δεν υπάρχει – τότε δημιουργείται.

def get_results_file_path(log_name): Κατ' αντιστοιχία με την προηγούμενη μέθοδο γίνεται έλεγχος της ύπαρξης του αρχείου των εντοπισμών και εάν ο γονικός του φάκελος δεν υπάρχει το αρχείο θα δημιουργηθεί(γραμμές 55-56). Επιστρέφεται η διαδρομή του αρχείου.

```python
    @staticmethod
    def get_db_path(db_name='police_db.sqlite')-> str:
        #Επιστρέφει το path της ΒΔ. Σε περίπτωση που ο φάκελος ΔΕΝ υπάρχει , δημιουργείται
        target_path = os.path.join(BASE_DIR, 'db',db_name)
        directory = os.path.dirname(target_path)

        if not os.path.exists(directory):
            os.makedirs(directory) # Δημιουργεί τον φάκελο 'db' αν λείπει

        return str(target_path)

    @staticmethod
    def get_results_file_path(log_name='detections.log') -> str:
        #Επιστρέφει το path του detections.log
        target_path = os.path.join(BASE_DIR,  'logs',  log_name)

        # Παίρνουμε μόνο τον φάκελο από το πλήρες μονοπάτι (π.χ. BASE_DIR/logs)
        directory = os.path.dirname(target_path)

        # Ελέγχουμε αν υπάρχει ο φάκελος. Αν όχι, τον δημιουργούμε.
        if not os.path.exists(directory):
            os.makedirs(directory) # Δημιουργείται ο φάκελος 'logs' αν λείπει

        return str(target_path)
```

Σχήμα 4.39 get_db_path() & get_results_file_path()

4.3.5.4 sound.py

Στο αρχείο αυτό περιλαμβάνεται η διαχείριση του υλικού που χρησιμοποιούμε για την ηχητική ανάδραση του χρήστη της εφαρμογής (Σχήμα 4.40). Αρχικά εισάγονται οι βιβλιοθήκες “time” , “threading” , όπως επίσης και οι μέθοδοι “TonalBuzzer” και “Tone” από τη βιβλιοθήκη διαχείρισης υλικού “gpiozero”. Η πρώτη είναι βιβλιοθήκη που χρησιμοποιούμε παρακάτω για τον έλεγχο του χρόνου, η δεύτερη είναι για τη διαχείριση νημάτων και η τελευταία είναι η επίσημη και πιο σύγχρονη βιβλιοθήκη για τον έλεγχο αλληλεπίδρασης με τον εξωτερικό κόσμο μέσω των ακίδων του Raspberry Pi. Στην αρχή της κλάσης SoundGenerator ορίζεται η μεταβλητή κλάσης “_instance” στην οποία θα αποθηκεύεται παρακάτω το αντικείμενο του ήχου.

def __new__(cls, *args, **kwargs): Στις γραμμές 12-16 γίνεται έλεγχος εάν υπάρχει ήδη κάτι στην μεταβλητή ή αν είναι άδεια. Στην περίπτωση που είναι άδεια δίνεται εντολή στην Python να δημιουργήσει το αντικείμενο στο υλικό και όταν θα ξανακληθεί η κλάση θα «δει» το ίδιο αντικείμενο και δε θα προσπαθήσει να φτιάξει εκ νέου. Έτσι αποτρέπουμε το υλικό να μπλοκάρει από την ύπαρξη διπλών αντικειμένων για το ίδιο Pin.

def __init__(self, pin=17): Στη γραμμή 19 γίνεται έλεγχος εάν το αντικείμενο έχει αρχικοποιηθεί. Εάν το attribute της γραμμής 15 είναι True, τότε παραλείπεται η εκ νέου ενεργοποίηση. Διαφορετικά προχωράμε σε «κλείδωμα» του νήματος ώστε να μην υπάρξει περίπτωση να εκτελεστούν δύο διαφορετικά νήματα για να παίξουν ηχητικά μηνύματα ταυτόχρονα (γραμμή 23). Μετά την εκτέλεση η σημαία της αρχικοποίησης του αντικειμένου παραμένει True ώστε να μην γίνει εκ νέου η διαδικασία σε επόμενη κλήση. Πρέπει να επισημάνουμε ότι στην περίπτωση σφάλματος επιλέχθηκε να μη γίνεται καταγραφή στα logs καθώς δεν κρίθηκε σκόπιμο – σε περίπτωση μιας κακής σύνδεσης με τα καλώδια dupont θα γέμιζε εγγραφές στο Log μας ήσσονος σημασίας.

```python
import time
import threading
from gpiozero import TonalBuzzer
from gpiozero.tones import Tone
#region Comments ···
You, last week | 1 author (You)
class SoundGenerator:
    _instance = None

    def __new__(cls, *args, **kwargs):
        if cls._instance is None:
            cls._instance = super(SoundGenerator, cls).__new__(cls)
            cls._instance._initialized = False
        return cls._instance

    def __init__(self, pin=17):
        if self._initialized:
            return

        self.ok = True
        self._lock = threading.Lock()

        try:
            self.buzzer = TonalBuzzer(pin)
        except Exception as e:
            print(f"[Sound Error] Buzzer Init Error: {e}")
            self.ok = False

        self._initialized = True
```

Σχήμα 4.40 το αρχείο sound.py με την κλάση SoundGenerator

def _run_in_thread(self, task, action_name= “Audio”): Η μέθοδος αυτή παίρνει σαν ορίσματα μια εργασία που πρέπει να εκτελεστεί και ένα όνομα. Η δουλειά της είναι να την εκτελέσει σωστά στο παρασκήνιο. Περιέχει μια συνάρτηση wrapper η οποία προσπαθεί να πάρει τον έλεγχο του κλειδωμένου νήματοςΣχήμα 4.41). Στην περίπτωση που το νήμα είναι ήδη απασχολημένο με άλλη εργασία τότε το (blocking=False) λέει στον έλεγχο να μην περιμένει σε ουρά εκτέλεσης αλλά να συνεχίσει προσπερνώντας ώστε να μην παγώσει η εκτέλεση του νήματος και ακούγονται ετεροχρονισμένα οι ήχοι από προηγούμενους εντοπισμούς. Έπειτα εκτελείται η εργασία και στο τέλος απελευθερώνεται το νήμα. Και φτάνουμε στη γραμμή 48 η οποία κάνει όλη τη σημαντική δουλειά: Η συνάρτηση που περιγράψαμε μόλις (η “wrapper”) εκτελείται σε ένα νέο νήμα στο παρασκήνιο – ώστε να μην επηρεάζεται καθόλου η εκτέλεση της εφαρμογής λόγω του ήχου. Το “daemon = True” λέει στο σύστημα ότι όσο η εφαρμογή είναι ενεργή – τόσο θα εκτελούνται οι ήχοι. Εάν ρυθμίσουμε έναν ήχο να ακούγεται για 10 δευτερόλεπτα για παράδειγμα, και στο πρώτο δευτερόλεπτο του ήχου κλείσουμε την εφαρμογή, χωρίς αυτή την παράμετρο θα είχαμε ήχο για ακόμη 9 δευτερόλεπτα καθώς εκτελείται σε ξεχωριστό – κλειδωμένο – νήμα. Με αυτόν τον τρόπο κλείνει και το νήμα ταυτόχρονα με την εφαρμογή μας.

Οι επόμενες τρεις συναρτήσεις δουλεύουν με τον ίδιο τρόπο. Ορίζουν εσωτερικά μια εργασία “_task” την οποία στέλνουν στην συνάρτηση που μόλις περιγράψαμε για να εκτελεστεί.

def play_scan_beep(self): Η συνάρτηση που παράγει τον ήχο μιας επιτυχούς αναγνώρισης πινακίδας κυκλοφορίας οχήματος. Στη γραμμή 52 γίνεται έλεγχος: εάν κατά την αρχικοποίηση της κλάσης δεν έχουμε “self.ok= True” τότε το υλικό μας δεν μπορεί να εκτελέσει τις εργασίες που θα θέλαμε να αναθέσουμε – οπότε επιστρέφουμε χωρίς να κάνουμε καμία ενέργεια. Στις γραμμές 54-58 ορίζουμε την εργασία που θέλουμε να εκτελέσουμε (συχνότητα – χρόνο διακοπής για ένα σύντομο και κοφτό «μπιπ») και κατόπιν την περνάμε σε μια κλήση της προηγούμενης συνάρτησης στη γραμμή 60.

```
33      def _run_in_thread(self, task, action_name="Audio"):
34          # Τρέχουμε τον ήχο σε ξεχωριστό thread.
35          def wrapper():
36              # Δοκιμάζει να πάρει την κλειδαριά
37              if self._lock.acquire(blocking=False):
38                  try:
39                      task()
40                  except Exception as e:
41                      print(f"[Sound Error] Σφάλμα κατά την αναπαραγωγή {action_name}: {e}")
42                  finally:
43                      self._lock.release()
44              else:
45                  # Αν βομβαρδιστεί με πολλές αναγνωρίσεις, απλά αγνοεί τις έξτρα
46                  pass
47
48          threading.Thread(target=wrapper, daemon=True).start()
49
50      def play_scan_beep(self):
51          #Ηχος σκαναρίσματος.
52          if not self.ok: return
53
54          def _task():
55
56              self.buzzer.play(Tone(frequency=600))
57              time.sleep(0.15)
58              self.buzzer.stop()
59
60          self._run_in_thread(_task, "Scan Beep (CLEAN)")
```

Σχήμα 4.41 _run_in_thread() & play_scan_beep()

def play_alert_beep(self): Ομοίως με παραπάνω με τη διαφορά ότι στις γραμμές 66-71 (Σχήμα 4.42) η εργασία που ορίζεται έχει μια «λούπα» με έναν ήχο λίγο χαμηλότερης συχνότητας και λίγο μεγαλύτερης διάρκειας. Η λούπα αυτή κάνει τρεις επαναλήψεις ώστε να ακουστεί ο ξεχωριστός ήχος της επισήμανσης εντοπισμού οχήματος που δεν είναι «καθαρό».

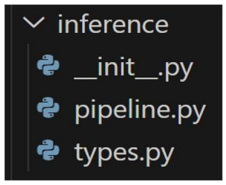
def play_welcome_intro(self): Ο τρόπος λειτουργίας είναι ακριβώς ο ίδιος με τις προηγούμενες δύο συναρτήσεις. Αλλάζει μόνο ο χρόνος που ενεργοποιείται το passive buzzer module μας (2 δευτερόλεπτα). Σκοπός είναι με την έναρξη της εφαρμογής να έχουμε κι ηχητική ανάδραση ώστε να επιβεβαιωθεί η σωστή λειτουργία του.

```
62      def play_alert_beep(self):
63          # Ήχος συναγερμού για "STOLEN/WANTED" οχήματα.
64          if not self.ok: return
65
66          def _task():
67              for _ in range(3):
68                  self.buzzer.play(Tone(frequency=500))
69                  time.sleep(0.3)
70                  self.buzzer.stop()
71                  time.sleep(0.1)
72
73          self._run_in_thread(_task, "Alert Beep (STOLEN/WANTED)")
74
75      def play_welcome_intro(self):
76          # Ήχος έναρξης εφαρμογής μετά το επιτυχημένο login του χρήστη. Σκοπός είναι
77          # να επιβεβαιωθεί η σωστή λειτουργία του passive buzzer module.
78          if not self.ok: return
79
80          def _task():
81              self.buzzer.play(Tone(frequency=600))
82              time.sleep(2.0)
83              self.buzzer.stop()
84
85          self._run_in_thread(_task, "Welcome Intro")
```

Σχήμα 4.42 play_alert_beep() & play_welcome_beep()

4.3.6 inference/

Πρόκειται για τον φάκελο που περιέχει τα αρχεία που κάνουν όλη τη «βαριά» δουλειά. Εδώ δουλεύει η μηχανή αναζήτησης πινακίδων και κατόπιν το OCR. Στον φάκελο αυτό περιέχονται (Σχήμα 4.43) : 1. “types.py”, 2. “pipeline.py” , 3. “ __init__.py” (Περισσότερες πληροφορίες γι’ αυτό στην υποενότητα 4.3.9)



Σχήμα 4.43 Περιεχόμενα φακέλου inference/

4.3.6.1 types.py

Το αρχείο αυτό αποτελεί ουσιαστικά μια βοηθητική κλάση αποκλειστικά για τον μηχανισμό του inference. Ο σκοπός της είναι να βοηθάει στην ανταλλαγή δεδομένων ανάμεσα στο UI και στο pipeline «πακετάροντας» τις πληροφορίες που χρειάζεται σε αντικείμενα. Τα αντικείμενα αυτά ονομάζονται Data Transfer Objects – DTO’s και οι κλάσεις που τα ορίζουν παίρνουν τον διακοσμητή “@dataclass”. Πρόκειται για ένα πολύ μικρό αρχείο πλην όμως εξαιρετικά χρήσιμο. Παρακάτω (Σχήμα 4.44) φαίνεται το περιεχόμενό του. Πρόκειται για μια κλάση αντικειμένου “DetectionResult” η οποία έχει ως ιδιότητες τη θέση και τις διαστάσεις του bounding box που εσωκλείει το ROI , το confidence του εντοπισμού, τον αριθμό της πινακίδας κυκλοφορίας καθώς και την κατάστασή του η οποία από προεπιλογή τίθεται σε «καθαρή».

```python
from dataclasses import dataclass
from typing import  Tuple

#Dataclass για να περνάμε τα αποτελέσματα της ανίχνευσης καθαρά.
You, 2 days ago | 1 author (You)
@dataclass
class DetectionResult:
    box: Tuple[int, int, int, int]  # [x, y, w, h]
    confidence: float
    plate_text: str = ""
    status: str = "CLEAN"
```

Σχήμα 4.44 Περιεχόμενα αρχείου types.py

4.3.6.2 pipeline.py

Πρόκειται για τον «πυρήνα» της εφαρμογής του APNR System. Όλη η «βαριά» και σημαντική δουλειά περιστρέφεται γύρω από αυτό το αρχείο καθώς εδώ η ΤΝ ζει κι εργάζεται σκληρά. Ξεκινώντας με τα imports (Σχήμα 4.45) έχουμε τη βιβλιοθήκη “cv2” η οποία είναι η βιβλιοθήκη που περιέχει τα πάντα για την Υπολογιστική Όραση (Computer Vision). Με τις μεθόδους της φορτώνουμε τα μοντέλα μας και γίνεται ο χειρισμός των δεδομένων της εικόνας. Η “numpy”, είναι μια μαθηματική βιβλιοθήκη για εξαιρετικά γρήγορους υπολογισμούς πράξεων πινάκων. Απαραίτητη καθώς τα μοντέλα στην ουσία κάνουν πράξεις πινάκων με τεράστια μεγέθη. Η “os” είναι υπεύθυνη για την διεπαφή με το λειτουργικό σύστημα. Επιπλέον εισάγεται το αρχείο ρυθμίσεων που είδαμε σε προγενέστερη υποενότητα “config1.py”, η κλάση “Labyrinth” του αρχείου “pathfinder.py” για τον εντοπισμό των θέσεων αρχείων , οι μέθοδοι “letterbox” και “clamp” από το βοηθητικό αρχείο “helpers.py” , η μέθοδος “log_error” από

το αρχείο “logger.py” για την καταγραφή σφαλμάτων στο log, και η κλάση “LicensePlateRecognizer” από τη βιβλιοθήκη “fast_plate_ocr” η οποία αναλαμβάνει να διαχειριστεί εξ ολοκλήρου το OCR.

def __init__(self, ocr_model_name): Με την αρχικοποίηση της κλάσης ζητείται να περαστεί παραμετρικά το μοντέλο που έχει επιλεγεί για το OCR (γραμμή 11). Στις επόμενες γραμμές (14-17) ζητείται η θέση του μοντέλου που ενεργεί τον εντοπισμό πινακίδων (detector) και το μοντέλο φορτώνεται στη μνήμη από το αρχείο *.onnx . Ειδικότερα ,επιλέγεται η χρήση του επεξεργαστή για τους υπολογισμούς αντί για κάποια κάρτα γραφικών – αφού ούτως ή άλλως το Raspberry Pi 3B+ δε διαθέτει- και η «μηχανή» που θα χρησιμοποιηθεί στο backend (χρησιμοποιείται η default για το OpenCV για συστήματα χωρίς κάρτα γραφικών). Έτσι παίρνουμε τις βέλτιστες παραμέτρους του μηχανισμού μας για να μην «κρασάρει» η εφαρμογή μας. Στην περίπτωση σφάλματος γίνεται η αντίστοιχη εγγραφή στο αρχείο “errors.log” – και «αδειάζει» από τη μνήμη ότι φορτώθηκε από το μοντέλο για το detection.

Έπειτα , στη γραμμή 26 φορτώνεται το μοντέλο για το OCR στη μνήμη , το οποίο έχει έρθει παραμετρικά με την αρχικοποίηση της κλάσης (από την επιλογή του χρήστη στο “gui_main.py”). Τέλος, στις γραμμές 27-32 σε περίπτωση σφάλματος γίνεται η καταγραφή αυτού στο “errors.log” και «αδειάζει» και το μοντέλο που προσπάθησε να φορτωθεί για το OCR από τη μνήμη μας.

def load_ocr(self , model_name): Η μέθοδος load_ocr() έχει μόνο ένα σκοπό: καλείται από το UI αφού ο χρήστης έχει επιλέξει το μοντέλο OCR που επιθυμεί να χρησιμοποιήσει , ώστε να το αναθέσει σαν “recognizer” στο μηχανισμό μας(γραμμή 36) Εάν αυτό αποτύχει τότε γίνεται σχετική εγγραφή στο “errors.log” και «αδειάζει» το μοντέλο που προσπάθησε να φορτωθεί από τη μνήμη (γραμμές 37-42).

```
1   import cv2
2   import numpy as np
3   import os
4   import app.core.config1 as config
5   from app.utils.helpers import letterbox, clamp
6   from app.utils.pathfinder import Labyrinth
7   from app.services.logger import log_error
8   from fast_plate_ocr import LicensePlateRecognizer
9
    You, 6 minutes ago | 1 author (You)
10  class InferencePipeline:
11      def __init__(self, ocr_model_name):
12          #Αρχικοποίηση του μοντέλου ανίχνευσης πινακίδων
13          try:
14              model_path = Labyrinth.get_model_detector()
15              self.net = cv2.dnn.readNetFromONNX(model_path)
16              self.net.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)
17              self.net.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)
18          except Exception as e:
19              file_name = os.path.basename(__file__)
20              error_type = type(e).__name__
21
22              log_error(error_type, file_name, str(e))
23              self.net = None
24
25          try:
26              self.recognizer = LicensePlateRecognizer(ocr_model_name)
27          except Exception as e:
28              file_name = os.path.basename(__file__)
29              error_type = type(e).__name__
30
31              log_error(error_type, file_name, str(e))
32              self.recognizer = None
33
34      def load_ocr(self, model_name):
35          try:
36              self.recognizer = LicensePlateRecognizer(model_name)
37          except Exception as e:
38              file_name = os.path.basename(__file__)
39              error_type = type(e).__name__
40
41              log_error(error_type, file_name, str(e))
42              self.recognizer = None
```

Σχήμα 4.45 Αρχικοποίηση της κλάσης “InferencePipeline” & load_ocr()

def detect_and_ocr(self , frame_bgr): Η καρδιά του συστήματος (Σχήμα 4.46): Στις γραμμές 45-46 ελέγχεται εάν έχει φορτωθεί το νευρωνικό μας δίκτυο κι εάν δέχεται εικόνα. Εάν όχι επιστρέφεται μια άδεια λίστα ώστε να μην «κρασάρει» η εφαρμογή μας. Έπειτα έχουμε την προετοιμασία της εικόνας που δεχόμαστε ως είσοδο- στη γραμμή 49, παίρνουμε την εικόνα και την περνάμε από τη μέθοδο letterbox() που είδαμε σε προγενέστερη υποενότητα. Έτσι αποθηκεύουμε την τετράγωνη «βελτιστοποιημένων διαστάσεων» εικόνα μας, την αναλογία μετατροπής (και το μέγεθος του padding που προστέθηκε πλάγια και πάνω) για να τα χρησιμοποιήσουμε παρακάτω. Στη γραμμή 50 κανονικοποιούνται οι τιμές των καναλιών των χρωμάτων ενώ γίνεται η αλλαγή τους από RGB σε BGR καθώς η βιβλιοθήκη OpenCV τα διαβάζει με αυτή τη σειρά. Το όρισμα "crop= False" ενημερώνει τη βιβλιοθήκη να μην προβεί σε περικοπή της εικόνας που δέχεται ως είσοδο -καθώς την έχουμε βελτιστοποιήσει εμείς στη μέθοδο "letterbox()". Οι γραμμές 51 και 52 – είναι το σημείο που «ταΐζουμε» το μοντέλο μας (detector) με την είσοδο -βελτιστοποιημένη γι' αυτό – και με την "forward()" δίνεται η εντολή εκτέλεσης. Το αποτέλεσμα του detection αποθηκεύεται στο outputs της γραμμής 52, το οποίο το «φιλτράρουμε» μέσα από την συνάρτηση "squeeze()" της numpy η οποία «αφαιρεί» τυχόν άδειες διαστάσεις από τον πίνακα. Οι γραμμές 54-70 αποτελούν «βελτίωση» του συστήματος – καθώς επιτρέπουν την επιλογή διαφορετικού μοντέλου detector- πχ αντί για το yolov11 που χρησιμοποιούμε ένα βελτιστοποιημένο yolo26 , ή και ένα yolov8. Στην υλοποίησή μας γίνεται χρήση μόνο **ενός** μοντέλου για το detection , παρ' όλα αυτά μπορεί με μια αλλαγή στο "config1.py" να χρησιμοποιηθεί ένα διαφορετικό μοντέλο και να συνεχίσει να λειτουργεί κανονικά το APNR System. Ο μηχανισμός του «αντάπτορα» θα λέγαμε λειτουργεί ως εξής: στη γραμμή 61 ελέγχεται εάν ο πίνακας που δίνεται ώς αποτέλεσμα από τη γραμμή 54 έχει δύο διαστάσεις. Εάν έχει δύο προχωράει παρακάτω στον έλεγχο. Αν το πλήθος των γραμμών είναι μικρότερο από το πλήθος των στηλών τότε είναι σχεδόν βέβαιο ότι έχει επιστραφεί πίνακας αποτελεσμάτων από μοντέλο που τον επιστρέφει ανάποδα (όπως το yolov8 που επιστρέφει αποτελέσματα με αυτή τη μορφή). Το δεύτερο κομμάτι του ελέγχου είναι το πλήθος των γραμμών του πίνακα που πήραμε ως έξοδο: το ελάχιστο πλήθος που χρειαζόμαστε είναι (x,y, w,h,) και το confidence οπότε 5. Εάν είναι εκπαιδευμένο να βρίσκει μία κλάση το μοντέλο (πχ LicensePlate) όπως στη δική μας περίπτωση με το βελτιστοποιημένο detector μας- τότε το πλήθος θα είναι 6. Εάν πρόκειται για ένα παλιότερο μοντέλο τότε σιγουρεύουμε ότι θα εντοπιστεί σωστά για να πάμε στην γραμμή 63 και να πάρουμε τον **αντίστροφο** πίνακα της εξόδου που πήραμε ώστε πλέον να μπορέσει να τον χειριστεί το πρόγραμμα χωρίς σφάλματα. Στην περίπτωση που η διάσταση του πίνακα της εξόδου είναι ίση με 1 (γραμμή 64) αυτό σημαίνει ότι στην ουσία έχουμε μια απλή λίστα- δεν εντοπίστηκε τίποτα στην εικόνα μας. Τότε επιστρέφεται μια άδεια λίστα και τερματίζεται η συνάρτηση χωρίς σφάλματα. Στην περίπτωση που οι γραμμές του πίνακα δεν ανήκουν στο εύρος τιμών της γραμμής 62, αλλά δεν είναι και ίσες με 1 τότε γίνεται μια «συμπίεση» στο μέγεθός του. Στα YOLOv5/v7 ο πίνακας εξόδου μπορεί να έχει 3 διαστάσεις εκ των οποίων η πρώτη είναι το μέγεθος της παρτίδας (batch size). Με τη γραμμή 67 αλλάζει το νούμερο των διαστάσεων. Μετά απ' όλες τις αλλαγές που μπορεί να έχουν γίνει – περνάμε στο σημείο που ελέγχεται το πλήθος των στηλών της εξόδου (γραμμή 69). Εάν είναι μικρότερο από 5 τότε επιστρέφεται μια κενή λίστα και τερματίζει η μέθοδος χωρίς να δημιουργηθεί πρόβλημα στην εφαρμογή μας. Στις γραμμές 72-77 βλέπουμε το δεύτερο και τελευταίο κομμάτι του «αντάπτορα». Καθώς κάποια μοντέλα επιστρέφουν τη θέση του αποτελέσματός τους σε ποσοστό επί του πλάτους και του ύψους της εικόνας, ενώ άλλα μοντέλα επιστρέφουν απόλυτες τιμές εικονοστοιχείων, εάν δεν γίνει αυτός ο έλεγχος και η μετατροπή τότε υπάρχει περίπτωση να τροφοδοτήσουμε τον κώδικα παρακάτω με σημεία που θα βρίσκονται «εκτός εικόνας» και θα οδηγήσουν σε κατάρρευση της εφαρμογής. Η λύση είναι απλή: αφού δημιουργήσουμε δύο άδειες λίστες με τα boxes και τα confidence τους για να τα χρησιμοποιήσουμε παρακάτω (γραμμές 72-73) προχωράμε σε μια «δειγματολειψία». Παίρνουμε μόνο τις πρώτες 50 γραμμές και από αυτές κρατάμε

μόνο τις πρώτες 4 στήλες (γραμμή 75). Μέσα σε αυτόν τον πίνακα δειγματοληψίας που έχουμε πάρει εντοπίζουμε τη μέγιστη τιμή που υπάρχει. Σε περίπτωση που ο πίνακας είναι άδειος αντί να πάρουμε σφάλμα και να καταρρεύσει ολόκληρο το inference θα επιστραφεί η τιμή 0.0. Το αποτέλεσμα μετατρέπεται σε αριθμό κινητής υποδιαστολής (γραμμή 76). Κατόπιν γίνεται ο εξής έλεγχος: εάν ο αριθμός αυτός είναι μικρότερος ή ίσος με 2.0. Εάν όχι , σίγουρα ο αριθμός «κριτήριο» είναι απόλυτη τιμή από εικονοστοιχείο πολύ μεγαλύτερος του 2. Εάν όμως το μοντέλο επέστρεφε σχετική θέση ύψους και πλάτους – οι τιμές θα πρέπει να είναι ανάμεσα στο 0.00 και στο 1.00. Καθώς όμως κάποιες φορές τα δίκτυα μπορεί να δώσουν μεγαλύτερες τιμές που θα βγάλουν τα κουτιά εκτός εικόνας για το λόγο αυτό με ένα «κατώφλι» στο 2.0 μπορούμε να είμαστε σίγουροι ότι δε θα επιστραφεί λάθος στην μεταβλητή της γραμμής 77 η οποία θα χρησιμοποιηθεί παρακάτω.

Στη γραμμή 79 ξεκινά η εκτέλεση ενός βρόγχου στον οποίο διατρέχονται όλα τα αποτελέσματα που έχουν δοθεί από το μοντέλο που ενεργεί το detection μας. Για καθένα απ' αυτά αποθηκεύονται οι τιμές των μεταβλητών μας (γραμμή 80). Στην γραμμή 82 γίνεται ένας έλεγχος για το μέγεθος των στηλών του πίνακά μας. Εάν είναι **ακριβώς** 5 όπως στα βελτιστοποιημένα μοντέλα νέας αρχιτεκτονικής (μετά το v8) που είναι εκπαιδευμένα στον εντοπισμό μιας κλάσης μόνο, τότε ξέρουμε ότι η τιμή στην τελευταία στήλη αντιστοιχεί στο confidence του εντοπισμού και ανατίθεται στη μεταβλητή conf της γραμμής 83. Εάν είναι μεγαλύτερος από 5, τότε πρόκειται για παλαιότερο μοντέλο , τα οποία στη θέση της στήλης 5 εξάγουν την πιθανότητα να υπάρχει **κάτι** στην περιοχή αυτή της εικόνας, με τις υπόλοιπες κλάσεις (στήλες) να δίνουν την πιθανότητα το αντικείμενο να ανήκει σε αυτή την κλάση. Οπότε σε αυτήν την περίπτωση ακολουθείται ο υπολογισμός της γραμμής 87 με το τμήμα "else obj" να αποτελεί δικλείδα ασφαλείας ώστε να μην κρασάρει το πρόγραμμα. Στη γραμμή 89 γίνεται ο έλεγχος ώστε το confidence του εντοπισμού να είναι πάνω από το «κατώφλι» που έχουμε ορίσει στο αρχείο ρυθμίσεών μας. Στις γραμμές 92-96 γίνεται η «αποκανονικοποίηση» των τιμών μας. Πολλαπλασιάζοντας πλέον κάθε μια από αυτές με το μέγεθος που έχουμε ορίσει στο αρχείο ρυθμίσεων , παίρνουμε πραγματικές τιμές εικονοστοιχείων της εικόνας μας αντί για ποσοστά. Καθώς τα μοντέλα στις τιμές x, y, δίνουν τις συντεταγμένες του κέντρου του bbox , πρέπει να τις μετατρέψουμε σε ακριβή σημεία της εικόνας. Αυτό ακριβώς γίνεται στις γραμμές 98-101. Πρόκειται ουσιαστικά για την αντιστροφή της διαδικασίας του letterbox που είδαμε σε προγενέστερη υποενότητα αναλυτικά.

```
44      def detect_and_ocr(self, frame_bgr):
45          if self.net is None or frame_bgr is None:
46              return []
47
48          # Letterbox & Inference
49          img_lb, r, (pad_x, pad_y) = letterbox(frame_bgr, (config.MODEL_INP, config.MODEL_INP))
50          blob = cv2.dnn.blobFromImage(img_lb, 1 / 255.0, (config.MODEL_INP, config.MODEL_INP), swapRB=True, crop=False)
51          self.net.setInput(blob)
52          outputs = self.net.forward()
53
54          out = np.squeeze(outputs)
55      #region Comments ···
61          if out.ndim == 2:
62              if out.shape[0] < out.shape[1] and out.shape[0] in (5, 6, 7, 8, 85, 84):
63                  out = out.T
64          elif out.ndim == 1:
65              return []
66          else:
67              out = out.reshape(-1, out.shape[-1])
68
69          if out.shape[1] < 5:
70              return []
71
72          boxes = []
73          confs = []
74
75          sample = out[:min(50, out.shape[0]), :4]
76          max_xywh = float(np.max(sample)) if sample.size else 0.0
77          normalized_xywh = max_xywh <= 2.0
78
79          for i in range(out.shape[0]):
80              x, y, w, h = out[i, 0], out[i, 1], out[i, 2], out[i, 3]
81
82              if out.shape[1] == 5:
83                  conf = float(out[i, 4])
84              else:
85                  obj = float(out[i, 4])
86                  cls_probs = out[i, 5:]
87                  conf = obj * float(np.max(cls_probs)) if cls_probs.size > 0 else obj
88
89              if conf < config.CONF_THRESHOLD:
90                  continue
91
92              if normalized_xywh:
93                  x *= config.MODEL_INP
94                  y *= config.MODEL_INP
95                  w *= config.MODEL_INP
96                  h *= config.MODEL_INP
97
98              left = (x - 0.5 * w - pad_x) / r
99              top = (y - 0.5 * h - pad_y) / r
100             right = (x + 0.5 * w - pad_x) / r
101             bottom = (y + 0.5 * h - pad_y) / r
```

Σχήμα 4.46 detect_and_ocr() (1/2)

Στις γραμμές 103-106 (Σχήμα 4.47) γίνεται ο «περιορισμός» των τιμών με την clamp() ώστε να μην έχουμε αρνητικές τιμές στα εικονοστοιχεία και η μετατροπή των τιμών τους σε ακέραιους. Έπειτα βρίσκουμε το πλάτος και το ύψος του bounding box (γραμμές 108-109) και τέλος προσθέτουμε τις τιμές των ορίων του κουτιού που εντοπίστηκε στη λίστα boxes και το αντίστοιχο confidence στη λίστα confs. Στη γραμμή 114 γίνεται ο τελικός έλεγχος ώστε σε περίπτωση που δεν έχουμε περιεχόμενα στη λίστα boxes να επιστραφεί μια κενή λίστα και να μην προχωρήσει σε περαιτέρω υπολογισμούς – αποφεύγοντας σφάλματα και κατάρρευση.

Από τη γραμμή 118 κι έπειτα λαμβάνει χώρα το NMS. Αρχικά γίνεται έλεγχος (γραμμές 119-120) ότι έχει επιστραφεί κάτι ως αποτέλεσμα της «σύγκρισης» των bboxes από τη γραμμή 118. Εάν όχι, επιστρέφεται κενή λίστα. Εάν υπάρχουν αποτελέσματα τότε εκτελείται ένας νέος βρόγχος για κάθε ένα από αυτά όπου: Στη γραμμή 124 εκτελείται μια «μετατροπή». Εξαιτίας διαφορών στις εξόδους της συνάρτησης "NMSBoxes" ανάλογα με την έκδοση του OpenCV κάποιες φορές επιστρέφεται ως αποτέλεσμα μια απλή λίστα , ενώ άλλες μια λίστα από λίστες. Για να το ξεπεράσουμε αυτό το πρόβλημα και το πρόγραμμα να εκτελείται ομαλά ανεξάρτητα από λειτουργικό σύστημα και την έκδοση του OpenCV που είναι εγκατεστημένη προχωράμε στην εξής λύση: η isinstance() ελέγχει εάν το item που πήρε ως όρισμα είναι ένα από τους τύπους που δίνονται σαν δεύτερο όρισμα. Εάν ναι, τότε η μεταβλητή k παίρνει την τιμή του item[0] «περασμένη» από μια μετατροπή σε ακέραιο. Εάν όχι, τότε πρόκειται για ήδη για αριθμό, οπότε τον περνάμε πάλι από την μετατροπή σε ακέραιο για να είμαστε σίγουροι και ανατίθεται στη μεταβλητή μας. Έπειτα , με τον δείκτη k παίρνουμε τις τιμές από τις λίστες των γραμμών 125-126 και προχωράμε στην «περικοπή» της εικόνας (γραμμή 133).

Εδώ ξεκινάει με έναν έλεγχο ασφαλείας και το κομμάτι του OCR: Στη γραμμή 136 ελέγχεται ότι η ROI δεν είναι None, ότι δεν είναι άδεια κι ότι έχει φορτωθεί στη μνήμη το μοντέλο OCR. Περνώντας απ'όλα αυτά τα βήματα προχωράμε στην εκτέλεσή του στη γραμμή 138. Και κάπου εδώ- καθώς οι διαφορετικές βιβλιοθήκες ή ακόμη και εκδόσεις ίδιων βιβλιοθηκών του OCR δεν μπορούν να διαχειριστούν μια εικόνα απ' ευθείας από τη μνήμη RAM και κρασάρουν υλοποιούμε μια μικρή παράκαμψη για να φτάσουμε στο αποτέλεσμα: με ένα exception πιάνουμε το σφάλμα και στον handler του φτιάχνουμε ένα προσωρινό αρχείο (γραμμές 141- 143). Σε αυτό αποθηκεύουμε την εικόνα από τη μνήμη και αμέσως μετά την ανοίγουμε (γραμμή 144), τρέχουμε το μοντέλο OCR και αμέσως μετά σβήνεται το προσωρινό αρχείο. Έτσι δεν υπάρχει περίπτωση να μη δουλέψει σωστά το OCR. Για την περίπτωση που αποτύχει ακόμη και η εγγραφή του προσωρινού αρχείου στο δίσκο τότε επιστρέφεται ένα κενό κείμενο για να μην κρασάρει η εφαρμογή μας (γραμμή 148).

Καθώς κάποια μοντέλα OCR επιστρέφουν το κείμενο του αποτελέσματος σε μορφή λίστας γίνεται ο έλεγχος των γραμμών 150-151 και η μεταβλητή raw_text παίρνει την τιμή του πρώτου στοιχείου της λίστας -εάν πρόκειται για λίστα που δεν είναι άδεια – αλλιώς παίρνει ένα κενό κείμενο. Έπειτα στις γραμμές 153- 155 γίνεται ένα επιπλέον «καθάρισμα» ώστε το αποτέλεσμα που θα έχουμε να είναι σίγουρα κείμενο και μάλιστα καθαρισμένο από κενά ή αλλαγές γραμμών (με την strip()).

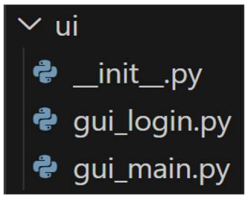
Τέλος, γίνεται η προσθήκη του αποτελέσματος στην κενή λίστα που δημιουργήσαμε στη γραμμή 122, αποθηκεύοντας τα στοιχεία του bbox, το κείμενο της πινακίδας που εντοπίστηκε με κεφαλαία για να μην έχουμε πρόβλημα ασυμβατότητας με τις εγγραφές της ΒΔ καθώς επίσης και το confidence του εντοπισμού. Στο τέλος, επιστρέφεται η λίστα με τα αποτελέσματα (γραμμή 163).

```
117         # NMS
118         indexes = cv2.dnn.NMSBoxes(boxes, confs, config.CONF_THRESHOLD, config.NMS_THRESHOLD)
119         if len(indexes) == 0:
120             return []
121 
122         results = []
123         for item in indexes:
124             k = int(item[0]) if isinstance(item, (list, tuple, np.ndarray)) else int(item)
125             bx = boxes[k]
126             conf = confs[k]
127 >           #region comments…
133             roi = frame_bgr[bx[1]:bx[1] + bx[3], bx[0]:bx[0] + bx[2]]
134             raw_text = ""
135 
136             if roi is not None and roi.size != 0 and self.recognizer is not None:
137                 try:
138                     raw_text = self.recognizer.run(roi)
139                 except Exception:
140                     try:
141                         import tempfile
142                         fd, tmp_path = tempfile.mkstemp(suffix=".jpg")
143                         os.close(fd)
144                         cv2.imwrite(tmp_path, roi)
145                         raw_text = self.recognizer.run(tmp_path)
146                         os.remove(tmp_path)
147                     except:
148                         raw_text = ""
149 
150             if isinstance(raw_text, list):
151                 raw_text = raw_text[0] if len(raw_text) > 0 else ""
152             raw_text = (raw_text or "")
153             if not isinstance(raw_text, str):
154                 raw_text = str(raw_text)
155             raw_text = raw_text.strip()
156 
157             results.append({
158                 'box': bx,
159                 'text': raw_text.upper(),
160                 'confidence': float(conf)
161             })
162 
163         return results
```

Σχήμα 4.47 detect_and_ocr() (2/2)

4.3.7 ui/

Στον φάκελο ui/ περιέχονται τα εξής αρχεία (Σχήμα 4.48) : 1. “gui_login.py”, 2. “gui_main.py” , 3. “ __init__.py” (Περισσότερες πληροφορίες γι’ αυτό στην υποενότητα 4.3.9)



Σχήμα 4.48 Περιεχόμενα φακέλου ui/

4.3.7.1 gui_login.py

Όπως θα δούμε αργότερα , με την έναρξη του προγράμματος ο έλεγχος περνάει στο LoginWindow. Για να μπορέσει κάποιος να περάσει στην κύρια εφαρμογή θα πρέπει πρώτα να πιστοποιηθεί ότι είναι νόμιμος χρήστης της. Αυτό θα γίνει εδώ. Με την εισαγωγή των βιβλιοθηκών στο αρχείο (Σχήμα 4.49) βλέπουμε ότι γίνεται χρήση του ενσωματωμένου εργαλείου της γλώσσας tkinter για τη δημιουργία των παραθύρων (και στο “gui_main.py” που θα δούμε αμέσως μετά) καθώς και του messagebox – υπεύθυνου για κάθε αναδυόμενο παράθυρο (pop-up window). Κατόπιν γίνεται εισαγωγή των κλάσεων

“AuthService” και “SoundGenerator” από τα αντίστοιχα αρχεία που εξετάσαμε νωρίτερα , τα “services.py” και “sound.py”.

Με την αρχικοποίηση της κλάσης βλέπουμε ότι έχει περαστεί παραμετρικά η παράμετρος “on_success”.Πρόκειται για μια συνάρτηση που δέχεται από την εκτέλεση του “main.py”. Θα δούμε τη σημασία της παρακάτω. Αρχικά δημιουργείται το κεντρικό παράθυρο (γραμμή 9). Του δίνονται τα χαρακτηριστικά όπως τίτλος και μέγεθος – μην επιτρέποντας στο χρήστη να κάνει προσαρμογή του μεγέθους του (γραμμές 9-12). Στην γραμμή 13 ανατίθεται η εξωτερική συνάρτηση στην συνάρτηση που καλείται με το επιτυχημένο login ενός χρήστη. Προστίθενται labels (γραμμές 15-17)και στις γραμμές 18 & 22 προστίθεται κουτιά εισαγωγής κειμένου (textbox) τα περιεχόμενα των οποίων αποθηκεύονται στα self.ent_user , self.ent_pass. Στη γραμμή 25 προστίθεται ένα κουμπί , το οποίο όταν πατηθεί εκτελείται η συνάρτηση handle_login() (Σχήμα 4.50).

def handle_login(self): Η συνάρτηση παίρνει με τις get() τις τιμές που έχουν αποθηκευτεί στο αντικείμενο της κλάσης ως properties και τις περνάει στις μεταβλητές user και pw τις οποίες δίνει ως ορίσματα με τη σειρά της στην συνάρτηση login() της κλάσης AuthService που είδαμε σε προγενέστερο κεφάλαιο. Εάν επιστραφεί True από εκεί , τότε το παράθυρο καταστρέφεται (γραμμή 36) καλείται να δημιουργηθεί ένα αντικείμενο της κλάσης SoundGenerator το οποίο είδαμε αναλυτικά επίσης νωρίτερα , αναπαράγεται ο ήχος έναρξης της εφαρμογής (γραμμή 40) και τέλος καλείται η self.on_success() η οποία επιστρέφει τον έλεγχο στο “main.py” (γραμμή 41). Σε περίπτωση σφάλματος κατά τη διαδικασία του login ο χρήστης παίρνει μήνυμα(γραμμή 43).

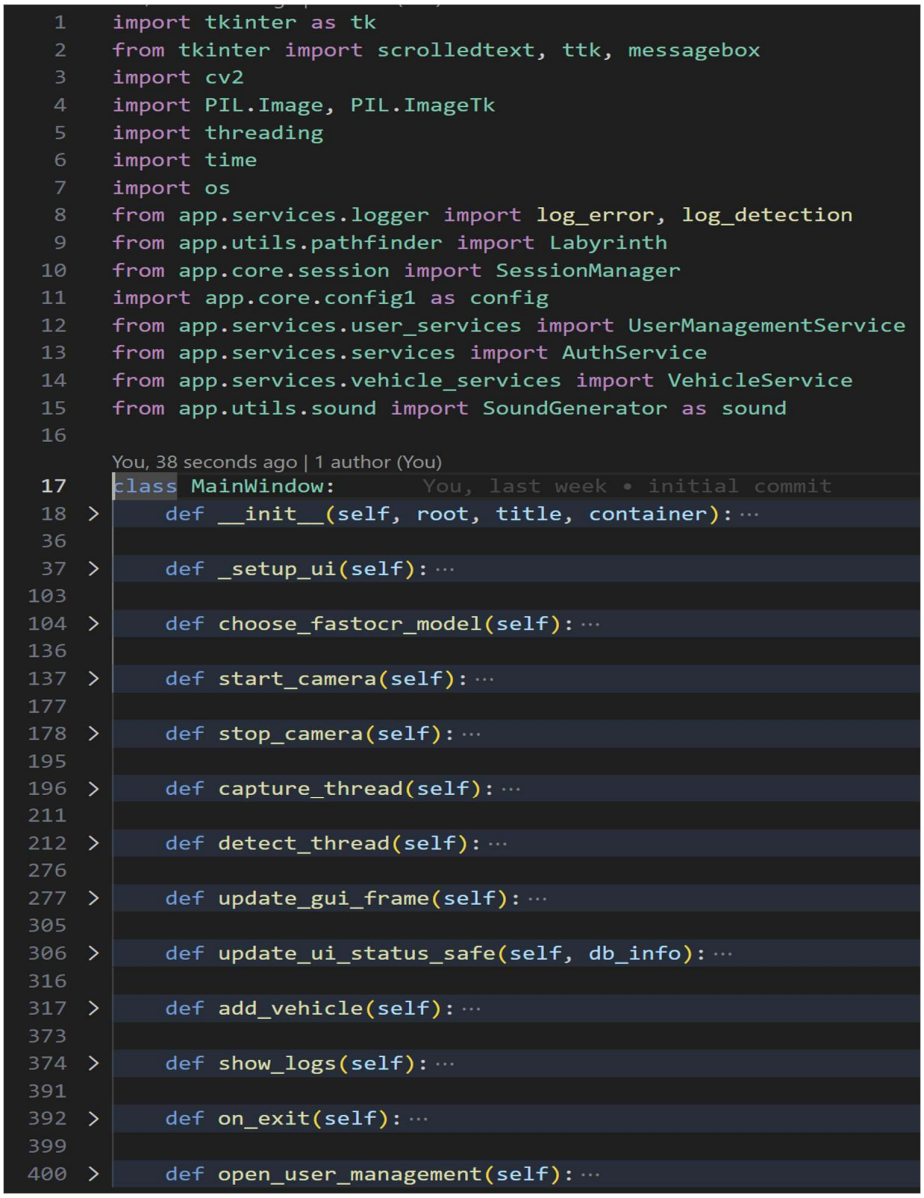
```
1   import tkinter as tk
2   from tkinter import messagebox
3   from app.utils.sound import SoundGenerator
4   from app.services.services import AuthService
5
    You, last week | 1 author (You)
6   class LoginWindow:
7       def __init__(self, on_success):
8           #Δημιουργία του login window
9           self.root = tk.Tk()
10          self.root.title("APNR System - Login")
11          self.root.geometry("350x250")
12          self.root.resizable(False, False)
13          self.on_success = on_success
14
15          tk.Label(self.root, text="ΣΥΣΤΗΜΑ APNR", font=("Arial", 14, "bold")).pack(pady=20)
16
17          tk.Label(self.root, text="Username:").pack()
18          self.ent_user = tk.Entry(self.root, width=25)
19          self.ent_user.pack(pady=5)
20
21          tk.Label(self.root, text="Password:").pack()
22          self.ent_pass = tk.Entry(self.root, show="*", width=25)
23          self.ent_pass.pack(pady=5)
24
25          tk.Button(self.root, text="LOGIN", bg="#27ae60", fg="white", width=15,
26                    command=self.handle_login).pack(pady=20)
27
28          self.root.mainloop()
29
30      def handle_login(self):
31          user = self.ent_user.get()
32          pw = self.ent_pass.get()
33
34          # Καλούμε το Service που ελέγχει τη βάση
35          if AuthService.login(user, pw):
36              self.root.destroy()
37
38              _sg= SoundGenerator()
39
40              _sg.play_welcome_intro()
41              self.on_success() # Αυτό θα ανοίξει το gui_main
42          else:
43              messagebox.showerror("Error", "Invalid credentials")
```

Σχήμα 4.49 Περιεχόμενα αρχείου gui_login.py

Σχήμα 4.50 Γραφικό περιβάλλον gui_login

4.3.7.2 gui_main.py

Πρόκειται για το «κέντρο ελέγχου» του APNR System. Από εδώ περνάνε όλα τα δεδομένα της εφαρμογής χωρίς το UI να καταλαβαίνει τί γίνεται. Ξεκινώντας με τις εισαγωγές των βιβλιοθηκών στο αρχείο (Σχήμα 4.51) έχουμε: Εισαγωγή της βιβλιοθήκης tkinter για το σχεδιασμό του UI στην οθόνη, μαζί με τα scrolledtext, ttk, messagebox που βελτιώνουν την εμπειρία χρήστη. Επίσης εισάγεται η βιβλιοθήκη του OpenCV , η cv2, καθώς επίσης και η PIL (Image & ImageTk) της (Pillow) η δουλειά της οποίας είναι το να παίρνει τα καρέ του OpenCV και να τα τοποθετεί στην οθόνη. Ακόμη εισήχθησαν οι threading, time και os τις οποίες είδαμε σε προγενέστερες ενότητες. Κατόπιν εισάγονται μέθοδοι και κλάσεις των αρχείων της εφαρμογής τις οποίες έχουμε όλες εξετάσει μέχρι τώρα.



Σχήμα 4.51 Οι μέθοδοι του αρχείου gui_main.py

def __init__(self, root, title, container): Αρχικά παρατηρούμε από την αρχικοποίηση της κλάσης "MainWindow" ότι παραμετρικά «φέρνει» και τον container (Σχήμα 4.52) . Αυτή είναι η υλοποίηση του DI που αναπτύξαμε στο προηγούμενο κεφάλαιο. Αντί να αφήσουμε το MainWindow να υλοποιήσει το inference και τη ΒΔ , τα δέχεται έτοιμα διαμέσου του container. Αφήνουμε έτσι την κλάση να χειρίζεται καθαρά το UI κάνοντας τον κώδικά μας αρθρωτό. Στις γραμμές 19-33 του αρχείου έχουμε την αποθήκευση του κεντρικού παραθύρου, προσθέτοντάς του έναν τίτλο και μέγεθος. Σημαντική είναι η γραμμή 22 με την οποία ρυθμίζουμε ένα σοβαρό πρόβλημα που μπορεί να προέκυπτε εάν ο χρήστης έκλεινε την εφαρμογή από το «Χ» επάνω δεξιά κι αφήναμε να το χειριστεί το λειτουργικό μας. Καθώς στην εφαρμογή τρέχουν πολλαπλά threads αυτά θα συνέχιζαν να τρέχουν επ' αόριστο – μέχρι να τερματιστεί η λειτουργία του Raspberry Pi. Έτσι, καλείται η μέθοδος "on_exit()" την οποία θα αναλύσουμε παρακάτω. Στη συνέχεια περνάμε τις εξαρτήσεις που

ετοιμάσαμε στο container στην ίδια την κλάση MainWindow , καλούμε την κλάση SoundGenerator για να πάρουμε το έτοιμο αρχείο που έχει δημιουργηθεί κατά το Login , κλειδώνουμε το νήμα για να είναι αφιερωμένο στη λειτουργία του ίδιου του MainWindow και αρχικοποιούμε μεταβλητές που θα χρειαστούμε παρακάτω- όπως το τελευταίο frame , ο πίνακας με τα τελευταία αποτελέσματα και δύο μεταβλητές χρονισμού (γραμμές 32 και 33). Τέλος καλείται η μέθοδος _setup_ui() που θα δούμε αμέσως.

```
17  class MainWindow:
18      def __init__(self, root, title, container):
19          self.root = root
20          self.root.title(title)
21          self.root.geometry("1000x750")
22          self.root.protocol("WM_DELETE_WINDOW", self.on_exit)
23          self.container = container
24          self.pipeline = container.pipeline
25          self.detection_service = container.detection_service
26          self.sound_gen = sound()
27          self.is_running = False
28          self.cap = None
29          self.lock = threading.Lock()
30          self.latest_frame = None
31          self.latest_results = []
32          self.last_ocr_time = 0
33          self.frame_count = 0
34
35          self._setup_ui()
```

Σχήμα 4.52 Η __init__() του αρχείου gui_main.py

def _setup_ui(self): Η μέθοδος αυτή χτίζει το γραφικό περιβάλλον. Αρχικά, δημιουργείται ένα πλαίσιο στο κεντρικό παράθυρο με πλάτος 250 pixels και ύψος όλο το διαθέσιμο του παραθύρου (fill) (Σχήμα 4.53) . Έπειτα στη γραμμή 43 απαγορεύεται στο παράθυρο να συρρικνωθεί σύμφωνα με τα περιεχόμενά του. Έπειτα προστίθενται τα επιμέρους γραφικά στοιχεία – ετικέτες και κουμπιά(1. “START CAMERA”, 2. ”STOP”, 3. ”Add Vehicle”, 4. ”View Logs”, 4.”Manage Users”, 5.”EXIT”), με ένα έξτρα πλαίσιο (γραμμή 61) το οποίο είναι για να διαχωρίζει το κομμάτι οπτικά το τμήμα της εφαρμογής που αφορά στον εντοπισμό πινακίδων -τον σκοπό υλοποίησης της παρούσης – από τις λοιπές βοηθητικές εργασίες. Η μορφοποίηση των κουμπιών γίνεται σύμφωνα με το λεξικό της γραμμής 51. Σε κάθε κουμπί σαν τελευταίο property βλέπουμε “**btn_style” το οποίο στην ουσία «ξεδιπλώνει» το λεξικό σε εκείνο το σημείο. Στην γραμμή 90 δημιουργούμε ένα δεύτερο πλαίσιο – το οποίο εκτείνουμε από επάνω και με το “fill=tk.BOTH” της γραμμής 91 του λέμε να «απλωθεί» σε όλο τον υπόλοιπο διαθέσιμο χώρο. Τέλος προστίθεται μια γραμμή «κατάστασης» στο χαμηλότερο μέρος του UI για να μπορέσουμε να δώσουμε οπτική ανατροφοδότηση του χρήστη χωρίς να επηρεάζεται καθόλου το υπόλοιπο τμήμα. Αυτό που πρέπει να επισημανθεί εδώ είναι η γραμμή 98 με το tk.StringVar. Πρόκειται για μια «ζωντανή» μεταβλητή , η οποία μόλις πάρει μια καινούρια τιμή (value) αυτή η τιμή θα «ζωγραφιστεί» αμέσως χωρίς να χρειαστεί να γίνει οποιαδήποτε αλλαγή σε ετικέτες. Επίσης, το relief= tk.SUNKEN δίνει ένα εφέ στη μπάρα που προσομοιάζει στις μπάρες των Windows. Το τελευταίο που ξεχωρίζει είναι το “anchor= tk.W” το οποίο σημαίνει «τοποθέτηση του κειμένου στα αριστερά» το αντίστοιχο του «text_alignment” σε πολλές άλλες γλώσσες προγραμματισμού.

```
17  class MainWindow:       You, last week • initial commit
37      def _setup_ui(self):
40          # Αριστερό πάνελ - σετάρισμα
41          self.left_frame = tk.Frame(self.root, width=250, bg="#2c3e50")
42          self.left_frame.pack(side=tk.LEFT, fill=tk.Y)
43          self.left_frame.pack_propagate(False)
44
45          tk.Label(self.left_frame, text="APNR System"+"\n"+"\n"+"Edge Computing Node"+
46                  "\n"+"on Raspberry Pi 3B+",
47                   font=("Helvetica", 14, "bold"),
48                   bg="#2c3e50", fg="#ecf0f1").pack(pady=25)
49
50
51          btn_style = {"font": ("Arial", 11), "pady": 5, "padx": 10}
52
53          self.btn_start = tk.Button(self.left_frame, text="START CAMERA", bg="#27ae60", fg="white",
54                                     command=self.start_camera, **btn_style)
55          self.btn_start.pack(pady=10, fill=tk.X, padx=15)
56
57          self.btn_stop = tk.Button(self.left_frame, text="STOP", bg="#c0392b", fg="white",
58                                    command=self.stop_camera, state=tk.DISABLED, **btn_style)
59          self.btn_stop.pack(pady=5, fill=tk.X, padx=15)
60
61          tk.Frame(self.left_frame, height=2, bg="grey").pack(fill=tk.X, pady=20, padx=10)
62
63          self.btn_add = tk.Button(self.left_frame, text="Add Vehicle", bg="#2980b9", fg="white",
64                                   command=self.add_vehicle, **btn_style)
65          self.btn_add.pack(pady=5, fill=tk.X, padx=15)
66
67          self.btn_log = tk.Button(self.left_frame, text="View Logs", bg="#8e44ad", fg="white",
68                                   command=self.show_logs, **btn_style)
69          self.btn_log.pack(pady=5, fill=tk.X, padx=15)
70
71          self.btn_users = tk.Button(self.left_frame, text="Manage Users", bg="#e67e22", fg="white",
72                                     command=self.open_user_management, **btn_style)
73          self.btn_users.pack(pady=5, fill=tk.X, padx=15)
74
75          tk.Label(self.left_frame, text="Διεθνές Πανεπιστήμιο της Ελλάδος" + "\n" + "Σχολή Μηχανικών" +
76                  "\n" + "\n" + "Τμήμα Μηχανικών Πληροφορικής " + "\n" + "& Ηλεκτρονικών Συστημάτων",
77                   font=("Helvetica", 10, "bold"),
78                   bg="#2c3e50", fg="#ecf0f1").pack(pady=20)
79
80          tk.Label(self.left_frame, text="Created by: "+"\n" +"Μπεσίρης Χρήστος (iee2019240)"+"\n"+"\n"+
81                  "Επιβλέπων:"+"\n"+"Δρ. Διαμαντάρας Κωνσταντίνος"+"\n"+"Καθηγητής",
82                   font=("Helvetica", 8, "bold"),
83                   bg="#2c3e50", fg="#ecf0f1").pack(pady=20)
84
85          self.btn_exit = tk.Button(self.left_frame, text="EXIT", bg="#34495e", fg="white",
86                                    command=self.on_exit, **btn_style)
87          self.btn_exit.pack(side=tk.BOTTOM, pady=20, fill=tk.X, padx=15)
88
89          # Σετάρισμα του κεντρικού πάνελ - frame ροής βίντεο
90          self.video_frame = tk.Frame(self.root, bg="black")
91          self.video_frame.pack(side=tk.TOP, fill=tk.BOTH, expand=True)
92
93          self.lbl_video = tk.Label(self.video_frame, bg="black", text="System Offline",
94                                    fg="#7f8c8d", font=("Arial", 18))
95          self.lbl_video.pack(expand=True)
96
97          # Εφαρμογή "Status Bar" στο κάτω μέρος της διεπαφής - για να δίνουμε ΟΠΤΙΚΟ feedback στον χρήστη
98          self.status_var = tk.StringVar(value="Waiting for start...")
99          self.lbl_status = tk.Label(self.root, textvariable=self.status_var, bd=1, relief=tk.SUNKEN,
100                                     anchor=tk.W, font=("Arial", 12, "bold"),
101                                     bg="#ecf0f1", padx=10, pady=5)
102         self.lbl_status.pack(side=tk.BOTTOM, fill=tk.X)
```

Σχήμα 4.53 46 _setup_ui()

def choose_fastocr_model(self): Η μέθοδος αυτή αποτελεί ένα επιπλέον feature του APNR System πέρα από τα προαπαιτούμενα για την υλοποίηση της παρούσης εργασίας. Όταν ο χρήστης πιέσει το κουμπί “START CAMERA” που είδαμε στο γραφικό περιβάλλον , τότε καλείται αυτή η μέθοδος η οποία (Σχήμα 4.54) : Δημιουργεί ένα νέο παράθυρο – «παιδί» αυτού που το κάλεσε(γραμμή 106). Του δίνεται τίτλος και σταθερό μέγεθος (γραμμές 107-108) και με τη γραμμή 109 «παντρεύεται» με το

γονικό του παράθυρο ώστε εάν ο χρήστης ελαχιστοποιήσει την εφαρμογή – να ελαχιστοποιηθεί αυτόματα κι αυτό. Κατόπιν προστίθενται , ταμπέλα(γραμμή 111), ένα νέο StringVar το οποίο παίρνει από προεπιλογή το όνομα του πρώτου μοντέλου OCR του αρχείου “config.py” κι ένα combobox το οποίο «δένεται» με την γραμμή 113 μέσω του “textvariable=var”. Επίσης δίνεται η παράμετρος “state=readonly” ώστε να μην επιτρέψουμε στον χρήστη να πληκτρολογήσει κάποιο κείμενο αντί να επιλέξει ένα από τα έτοιμα μοντέλα – καθώς εάν δεν το χειριζόμασταν θα προκαλούταν σφάλμα στο inference pipeline και δε θα δούλευε σωστά η εφαρμογή. Τέλος, στη γραμμή 116 δίνεται η εντολή να χρησιμοποιηθεί η πρώτη τιμή (0) από τις διαθέσιμες, ώστε με την εκκίνηση του παραθύρου να μην εμφανίζεται κενό και φορτώνει σε δεύτερο χρόνο. Στην γραμμή 118 υλοποιείται ένα λεξικό καθώς θέλουμε να αποθηκεύσουμε την τιμή που θα επιλέξει ο χρήστης σε αυτό. Η επιλογή του λεξικού αντί για μια απλή μεταβλητή έρχεται να λύσει το πρόβλημα του scope καθώς η αλλαγή από εσωτερική συνάρτηση θα προκαλούσε σφάλμα. Με αυτόν τον τρόπο δεν έχουμε πρόβλημα. Έτσι όταν ο χρήστης επιλέξει «οκ» στην εσωτερική συνάρτηση def ok() , η επιλογή του αποθηκεύεται ως το value του model στο λεξικό result. Στην περίπτωση που επιλέξει να πατήσει το «cancel» καλείται η εσωτερική συνάρτηση “cancel” και πριν κλείσει το παράθυρο (γραμμή 126) «αδειάζει» την τιμή που αντιστοιχεί στο κλειδί “model” του λεξικού result. Οι γραμμές 128-131 φτιάχνουν ένα πλαίσιο και τοποθετούν τα δύο κουμπιά μέσα του. Σημαντικό: στη γραμμή 133 δίνεται εντολή στο UI να αγνοήσει οτιδήποτε άλλο κάνει ο χρήστης εκτός του παραθύρου αυτού. Στην ουσία «δεσμεύει» ποντίκι και πληκτρολόγιο στο κομμάτι της εφαρμογής. Επίσης στην γραμμή 134 το πρόγραμμα ενημερώνεται να «παγώσει» μέχρι να καταστραφεί το παράθυρο – είτε όταν ο χρήστης επιλέξει «οκ» , είτε όταν επιλέξει «cancel». Η μέθοδος επιστρέφει το value που αντιστοιχεί στο κλειδί “model” του λεξικού result.

```
104     def choose_fastocr_model(self):
105         # Παράθυρο επιλογής μοντέλου για το OCR από τον χρήστη.
106         win = tk.Toplevel(self.root)
107         win.title("Select OCR model")
108         win.resizable(False, False)
109         win.transient(self.root)
110 
111         tk.Label(win, text="Choose fast-plate-ocr model:", font=("Arial", 11, "bold")).pack(padx=12, pady=(12, 6))
112 
113         var = tk.StringVar(value=config.FAST_OCR_MODELS[0])
114         combo = ttk.Combobox(win, values=config.FAST_OCR_MODELS, textvariable=var, state="readonly", width=42)
115         combo.pack(padx=12, pady=6, fill=tk.X)
116         combo.current(0)
117 
118         result = {"model": None}
119 
120         def ok():
121             result["model"] = var.get().strip()
122             win.destroy()
123 
124         def cancel():
125             result["model"] = None
126             win.destroy()
127 
128         btns = tk.Frame(win)
129         btns.pack(padx=12, pady=(8, 12), fill=tk.X)
130         tk.Button(btns, text="OK", command=ok, width=10).pack(side=tk.LEFT)
131         tk.Button(btns, text="Cancel", command=cancel, width=10).pack(side=tk.RIGHT)
132 
133         win.grab_set()
134         win.wait_window()
135         return result["model"]
```

Σχήμα 4.54 choose_fastocr_model()

def start_camera(self): Όταν ο χρήστης της εφαρμογής πατήσει το κουμπί “START CAMERA” από το UI καλείται αυτή η μέθοδος. Στη γραμμή 140 με την έναρξη της μεθόδου γίνεται ένας έλεγχος (Σχήμα

4.55) – στην περίπτωση που ο χρήστης μπόρεσε να πατήσει το κουμπί “START” περισσότερες από μια φορές , τότε αντί να ξεκινήσει εκ νέου η κάμερα και να κρασάρει ολόκληρο το σύστημα, επιστρέφεται ο έλεγχος. Έπειτα στις γραμμές 143-146 περνάει το μοντέλο που έχει επιστρέψει η συνάρτηση που μόλις είδαμε σε μια παράμετρο της μεθόδου ελέγχοντας πρώτα ότι δεν έχει επιστραφεί “None”. Έπειτα γίνεται ο έλεγχος ότι το pipeline έχει μέθοδο που λέγεται “load_ocr” και όταν αυτή βρεθεί καλείται με όρισμα το μοντέλο που έχει επιλεγεί από το χρήστη (γραμμές 148-149). Στις γραμμές 152-155 ενεργοποιείται η κάμερα μέσω της βιβλιοθήκης OpenCV. Ορίζεται ένα buffersize ίσο με ένα , ώστε να μην μένουν καθόλου καρέ στην μνήμη και να μην έχουμε καθυστέρηση στην εικόνα μας, και η ανάλυση της κάμερας σύμφωνα με τις ρυθμίσεις από το αρχείο “config1.py”. Στις γραμμές 157-161 έχουμε την αλλαγή σημαίας σε True, την απενεργοποίηση του κουμπιού “START CAMERA” και αντίστοιχα την ενεργοποίηση του κουμπιού “STOP” , την αλλαγή του κειμένου της ταμπέλας που είχαμε στο πλαίσιο της εικόνας σε άδειο κείμενο ώστε να δώσουμε χώρο στο βίντεο και την αλλαγή της μπάρας κατάστασης σε “Scanning…” για να δείξει ότι το inference πλέον είναι ενεργό. Οι γραμμές 168-170 είναι ζωτικής σημασίας καθώς σε αυτές «μοιράζεται» ο φόρτος εργασίας ανάλογα με τις διαδικασίες σε διαφορετικά νήματα: το πρώτο είναι το capture_thread το οποίο έχει σαν μοναδικό σκοπό να παίρνει φωτογραφίες από την κάμερα , το δεύτερο είναι το detect_thread το παίρνει τις φωτογραφίες που δίνει το πρώτο thread και τις περνάει στο pipeline για να γίνει ο εντοπισμός. Όπως είπαμε και νωρίτερα- η παράμετρος “daemon= True” σημαίνει ότι σε περίπτωση που τερματιστεί η εφαρμογή θα τερματιστούν και τα «παρελκόμενα» νήματα. Επιπλέον καλείται η μέθοδος update_gui_frame() την οποία θα δούμε παρακάτω. Σε περίπτωση που υπάρξει οποιαδήποτε εξαίρεση, τότε αυτή θα καταγραφεί στο αρχείο “errors.log” (γραμμές 172-175) και θα εμφανιστεί και σχετικό μήνυμα με messagebox στον χρήστη(γραμμή 176).

```
 17   class MainWindow:
137       def start_camera(self):
140           if self.is_running:
141               return
142
143           chosen = self.choose_fastocr_model()
144           if not chosen:
145               return
146           self.current_ocr_model = chosen
147
148           if hasattr(self.pipeline, 'load_ocr'):
149               self.pipeline.load_ocr(chosen)
150
151           try:
152               self.cap = cv2.VideoCapture(config.CAMERA_ID)
153               self.cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)
154               self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, config.FRAME_WIDTH)
155               self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, config.FRAME_HEIGHT)
156
157               self.is_running = True
158               self.btn_start.config(state=tk.DISABLED)
159               self.btn_stop.config(state=tk.NORMAL)
160               self.lbl_video.config(text="")
161               self.status_var.set("Scanning...")
162
163   > #region Comments···
168               threading.Thread(target=self.capture_thread, daemon=True).start()
169               threading.Thread(target=self.detect_thread, daemon=True).start()
170               self.update_gui_frame()
171
172           except Exception as e:
173               file_name = os.path.basename(__file__)
174               error_type = type(e).__name__
175               log_error(error_type, file_name, str(e))
176               messagebox.showerror("Camera Error", str(e))
```

Σχήμα 4.55 start_camera()

def stop_camera(self): Η μέθοδος αυτή που καλείται όταν ο χρήστης πιέσει το κουμπί “STOP” στο UI ξεκινώντας αλλάζει μια σημαία (γραμμή 180) σε False (Σχήμα 4.56). Με βάση τον έλεγχο αυτής της σημαίας παρακάτω θα δούμε ότι τα capture_thread και detect_thread τερματίζονται. Στις γραμμές 181 – 184 ελέγχεται εάν η κάμερα είναι ενεργοποιημένη και στη γραμμή 183 γίνεται απενεργοποίησή της. Εάν για κάποιο λόγο έχει αποσυνδεθεί λόγω υλικού τότε δεν αφήνει η γραμμή 184 το πρόγραμμα να καταρρεύσει. Στη γραμμή 186 κλειδώνεται το νήμα του UI, καθαρίζονται οι μεταβλητές των γραμμών 187 και 188 και στις γραμμές 190-194 γίνεται η ακριβώς αντίστροφη διαδικασία από αυτή που είχαμε στη μέθοδο “start_camera()”- δηλαδή ενεργοποιούμε το κουμπί “START CAMERA” , απενεργοποιείται το κουμπί “STOP”, αλλάζει το κείμενο στην ετικέτα στη θέση που καταλάμβανε το βίντεο και αλλάζει και το κείμενο στην γραμμή κατάστασης.

def capture_thread(self): Είδαμε νωρίτερα ότι όταν ο χρήστης πιέσει το κουμπί “START CAMERA” η μέθοδος “start_camera()” στο τέλος της περίπου καλεί αυτή τη συνάρτηση. Όσο η κάμερα είναι ενεργοποιημένη, αυτή η συνάρτηση τρέχει σε βρόγχο αρκεί να είναι η μεταβλητή “is_running” ίση με True και να έχει δεδομένα το capture. Τότε προκειμένου να μπορέσουμε να μειώσουμε το χρόνο επεξεργασίας της εικόνας είναι προτιμότερο να δουλέψουμε με τις μεθόδους grab() και retrieve() (γραμμές 200-201) καθώς αστραπιαία η εικόνα περνάει σε έναν buffer χωρίς επεξεργασία κι από εκεί μετατρέπεται μέσω της numpy σε πίνακα , ενώ η μεταβλητή ret γίνεται True αν όλα πήγαν καλά. Έτσι γίνεται η επεξεργασία πιο γρήγορα. Στην περίπτωση που το camera module δεν υποστηρίζει grab() & retrieve() θα επιστραφεί σφάλμα και με εναλλακτική τη γραμμή 203 παίρνουμε το αποτέλεσμα με τον κλασσικό τρόπο της μεθόδου read(). Στην περίπτωση που χαθεί κάποιο frame και δεν μπορέσει να επιστραφεί εικόνα η μεταβλητή ret θα γίνει False- και η γραμμή 206 θα επιβάλει στο πρόγραμμα να συνεχίσει να εκτελεί το βρόγχο χωρίς να καταρρεύσει. Στη γραμμή 208 «κλειδώνει» η πρόσβαση στο frame μέχρι να γίνει η εγγραφή στη μεταβλητή και κατόπιν «ξεκλειδώνει» καθώς η μεταβλητή “self.latest_frame” είναι μια μεταβλητή που χρησιμοποιείται και από τα τρία διαφορετικά νήματα κάποια στιγμή. Εάν κάποια από αυτά προσπαθήσουν να την επεξεργαστούν ταυτόχρονα τότε η εφαρμογή θα καταρρεύσει. Τέλος , στη γραμμή 210 δίνεται λίγος χρόνος στον επεξεργαστή του Raspberry Pi για να «ξεκουραστεί» και να τρέξει κι άλλες διεργασίες.

```
178     def stop_camera(self):
180         self.is_running = False
181         if self.cap:
182             try:
183                 self.cap.release()
184             except: pass
185 
186         with self.lock:
187             self.latest_frame = None
188             self.latest_results = []
189 
190         self.btn_start.config(state=tk.NORMAL)
191         self.btn_stop.config(state=tk.DISABLED)
192         self.lbl_video.config(image='', text="System Offline", bg="black")
193         self.status_var.set("Waiting...")
194         self.lbl_status.config(bg="#ecf0f1", fg="black")
195 
196     def capture_thread(self):
197         # Ορισμός και κλείδωμα του capture_thread
198         while self.is_running and self.cap:
199             try:
200                 self.cap.grab()
201                 ret, frame = self.cap.retrieve()
202             except:
203                 ret, frame = self.cap.read()
204 
205             if not ret:
206                 continue
207 
208             with self.lock:
209                 self.latest_frame = frame
210             time.sleep(0.001)
```

Σχήμα 4.56 stop_camera() & capture_thread()

def detect_thread(self): Εκτός από την μέθοδο “capture_thread()” η μέθοδος “start_camera()” καλεί και την συνάρτηση αυτή. Στην κλήση της άμεσα ελέγχει την μεταβλητή της γραμμής 215 μέσα σε έναν βρόγχο (Σχήμα 4.57). Εάν είναι True , τότε «κλειδώνει» την πρόσβαση στο “self.latest_frame” και δημιουργεί ένα αντίγραφο (γραμμή 218). Σε περίπτωση που δεν βρέθηκε φωτογραφία τότε το πρόγραμμα περιμένει για 0.01 δευτερόλεπτα και ξανακοιτάει (γραμμές 220-222). Στην γραμμή 224 βλέπουμε τη μεταβλητή που συναντήσαμε στην αρχικοποίηση της κλάσης MainWindow, η οποία αυξάνεται κατά ένα για κάθε νέο frame που περνάει. Έτσι – στις γραμμές 225-227 βλέπουμε ότι εάν το υπόλοιπο της διαίρεσης της τιμής της μεταβλητής με τον αριθμό που έχουμε δώσει στις ρυθμίσεις μας – είναι διαφορετικό του μηδενός δίνεται χρόνος στο πρόγραμμα να «ξεκουραστεί» και να περιμένει να αυξηθεί ο αριθμός των ληφθέντων φωτογραφιών. Στη γραμμή 229 ενημερώνεται η μεταβλητή με την τρέχουσα ώρα και στη γραμμή 230 η λίστα results είναι το αποτέλεσμα της μεθόδου detect_and_ocr() που είδαμε στο αρχείο “pipeline.py”.Τότε στη γραμμή 236 κλειδώνει την πρόσβαση για να πραγματοποιήσει την εγγραφή στα «κοινόχρηστα» αρχεία των αποτελεσμάτων (self.latest_results) και επιλέγεται ένας αριθμός αποτελεσμάτων ίσος με την μεταβλητή “MAX_DETECTIONS_DRAW” του αρχείου ρυθμίσεών μας, ώστε σε περίπτωση που εντοπιστεί μεγάλος αριθμός από αποτελέσματα να μην ζορίσουμε την πλατφόρμα μας για να τα ζωγραφίσει. Στη γραμμή 239 γίνεται ο έλεγχος του χρονικού κατωφλιού ώστε να μην γίνονται συνεχώς αιτήματα προς τη βάση μας για το ίδιο σταματημένο όχημα μπροστά στην οθόνη. Στις γραμμές 241-247: διατρέχονται τα αποτελέσματα – παίρνουμε τις πινακίδες κυκλοφορίας και προχωράμε στο OCR μέσω της γραμμής 247. Στις γραμμές 252-265 παίρνουμε τα αποτελέσματα του OCR και ως παράμετρο τα περνάμε στη μέθοδο update_ui_status_safe(). Ανάλογα με το status του οχήματος που ανιχνεύθηκε εκτελείται και ο κατάλληλος ήχος. Κατόπιν γίνεται καταχώρηση της εντοπισμένης πινακίδας στο “detection.log” και καταχωρείται το timestamp στην μεταβλητή τελευταίας ανίχνευσης OCR στη γραμμή 267. Σε περίπτωση που προκύψει εξαίρεση σε όλη τη διαδικασία γίνεται καταγραφή του τόσο στην κονσόλα μας όσο και στο “errors.log”. Τέλος, κλειδώνεται η πρόσβαση στο αρχείο αποτελεσμάτων – αδειάζεται- ώστε να μην περιέχει άχρηστη πληροφορία και δίνεται χρόνος ενός δευτερολέπτου για να μπορέσει να «ανασάνει» το υλικό μας και μετά ξαναπροσπαθεί.

```
212     def detect_thread(self):
215         while self.is_running:
216             try:
217                 with self.lock:
218                     frame = None if self.latest_frame is None else self.latest_frame.copy()
219
220                 if frame is None:
221                     time.sleep(0.01)
222                     continue
223
224                 self.frame_count += 1
225                 if (self.frame_count % config.DETECT_EVERY_N_FRAMES) != 0:
226                     time.sleep(0.001)
227                     continue
228
229                 current_time = time.time()
230 > #region Comments ···
234                 results = self.pipeline.detect_and_ocr(frame)
235
236                 with self.lock:
237                     # Καθαρίζει το box αν δεν βρει πινακίδα
238                     self.latest_results = results[:config.MAX_DETECTIONS_DRAW] if results else []
239
240                 if results and (current_time - self.last_ocr_time) >= config.OCR_INTERVAL_SEC:
241                     for res in results:
242                         plate = res['text']
243                         conf = res.get('confidence', 0.0)
244                         if not plate:
245                             continue
246 > #region Comments ···
250                         db_info = self.detection_service.process_detected_plate(plate)
251
252                         if db_info:
253                             self.update_ui_status_safe(db_info)
254                             if db_info['status'] != "CLEAN":
255                                 #Εάν η πινακίδα ΔΕΝ είναι "CLEAN" τότε στέλνουμε ALERT - ΗΧΗΤΙΚΟ FEEDBACK
256                                 self.sound_gen.play_alert_beep()
257                             else:
258                                 # Αλλιώς το buzzer αναπαράγει μόνο ένα "scan_beep" για ΗΧΗΤΙΚΟ FEEDBACK
259                                 self.sound_gen.play_scan_beep()
260                             log_detection(
261                                 plate=db_info['plate'],
262                                 status=db_info['status'],
263                                 confidence=conf,
264                                 ocr_model=getattr(self, 'current_ocr_model', 'Unknown')
265                             )
266
267                     self.last_ocr_time = time.time()
268             except Exception as e:
269                 print(f"\n[ΣΦΑΛΜΑ ΣΤΟ DETECT THREAD]: {e}")
270                 log_error("DetectThreadError", "gui_main.py", str(e))
271                 with self.lock:
272                     self.latest_results = []
273                 time.sleep(1)
```

Σχήμα 4.57 detect_thread()

def update_gui_frame(self): Στο Σχήμα 4.58 βλέπουμε τη μέθοδο η οποία είναι υπεύθυνη για το «ζωγράφισμα» του καμβά της εφαρμογής μας. Στην αρχή γίνεται έλεγχος αν κάποιος έχει πατήσει το κουμπί “STOP”. Σε θετική περίπτωση η μεταβλητή της γραμμής επιστρέφει false και η μέθοδος τερματίζει. Αλλιώς κλειδώνεται προσωρινά η πρόσβαση στα αρχεία frame και results ώστε να γίνει μια αντιγραφή της εικόνας και το πέρασμα των αποτελεσμάτων σε λίστα για τον χειρισμό περαιτέρω (γραμμές 280-282). Εάν βρεθεί εικόνα ξεκινάει ένας βρόγχος στη γραμμή 284 όπου: για κάθε αποτέλεσμα παίρνουμε τις συντεταγμένες , την πινακίδα κυκλοφορίας και το confidence. Διαμορφώνεται το κείμενο που θα εμφανιστεί σύμφωνα με τη γραμμή 290 και κατόπιν καλείται η OpenCV να σχεδιάσει ένα ορθογώνιο όπου σχηματίζεται από το πάνω αριστερό και κάτω δεξί σημείο με πράσινο χρώμα και πάχος 2 pixels. Καλείται ομοίως να τοποθετήσει το μορφοποιημένο κείμενο επάνω από το box διαφυλάσσοντας πάντα ότι το κείμενο δε θα βγει ποτέ εκτός πλαισίου (x, max(0, y-10)) της γραμμής 293. Έπειτα , γίνεται αλλαγή στα χρωματικά κανάλια για να μπορούν να περαστούν

στο tkinter, η εικόνα περνάει μέσα από τη βιβλιοθήκη Pillow (γραμμές 296-297) η οποία μετατρέπει τους πίνακες της εικόνας σε πραγματικά pixels και περνάει στην γραμμή 299 την εικόνα που μόλις δημιουργήσαμε στον καμβά αφού την αποθηκεύσαμε στη γραμμή 298. Η αποθήκευση γίνεται καθώς ο garbage collector της γλώσσας είναι γρήγορος -εάν παραλείψουμε αυτή τη γραμμή μόλις εμφανιστεί η εικόνα θα θεωρηθεί περιττή αμέσως με αποτέλεσμα να έχουμε ένα «τρεμόπαιγμα». Έτσι κάθε εικόνα μένει μέχρι να αντικατασταθεί από την επόμενη – συνθέτοντας ένα βίντεο χωρίς προβλήματα. Στη γραμμή 301 υπολογίζουμε το χρόνο που απαιτείται για κάθε FPS ανάλογα με τη μεταβλητή μας στο αρχείο ρυθμίσεων. Έτσι, στη γραμμή 302 ορίζουμε έναν βρόγχο ο οποίος θα τρέχει στο σωστό χρονικό πλαίσιο -καλώντας την μέθοδο εκ νέου για να ζωγραφιστεί ο καμβάς.

```
275     def update_gui_frame(self):
276         #UI Updater μέθοδος για 'ανανέωση' του frame (cap)
277         if not self.is_running:
278             return
279 
280         with self.lock:
281             frame = None if self.latest_frame is None else self.latest_frame.copy()
282             results = list(self.latest_results)
283 
284         if frame is not None:
285             for res in results:
286                 x, y, w, h = res['box']
287                 plate_text = res.get('text', '')
288                 conf = res.get('confidence', 0.0)
289 
290                 display_text = f"{plate_text} ({conf:.2f})" if plate_text else f"Conf: {conf:.2f}"
291 
292                 cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
293                 cv2.putText(frame, display_text, (x, max(0, y - 10)),
294                             cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
295 
296             img = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
297             img_tk = PIL.ImageTk.PhotoImage(image=PIL.Image.fromarray(img))
298             self.lbl_video.img_tk = img_tk
299             self.lbl_video.configure(image=img_tk)
300 
301         ms = int(1000 / config.GUI_FPS)
302         self.root.after(ms, self.update_gui_frame)
```

Σχήμα 4.58 update_gui_frame()

def update_ui_status_safe(self, db_info): (Σχήμα 4.59) Η μέθοδος δέχεται κατά την κλήση της τα πλήρη στοιχεία που έχουν επιστρέψει από τη ΒΔ ως παράμετρο. Στη γραμμή 306 γίνεται μια μορφοποίηση των στοιχείων αυτών και αρχικοποιείται η μεταβλητή της γραμμής 307 με το χρώμα πράσινο. Έπειτα, ανάλογα με την κατάσταση που έχει επιστραφεί από τη ΒΔ, αλλάζει και το χρώμα της μεταβλητής. Τέλος, ενημερώνεται το UI για να αλλάξει το status bar σύμφωνα με τα περιεχόμενα των γραμμών 312 και 313. Θα πρέπει να επισημανθεί ότι : με τον τρόπο αυτό «προτρέποντας» το root να κάνει μια εργασία με τη χρήση “after” δεν κρασάρει η εφαρμογή – καθώς η αλλαγή από ένα thread σε ένα άλλο στην περίπτωσή μας είναι επικίνδυνη. Επίσης μέσω της lamda μεθόδου και καλούνται οι μέθοδοι καθώς η “after” απαιτεί μόνο το όνομα μιας συνάρτησης κι όχι την εκτέλεσή της.

```
304     def update_ui_status_safe(self, db_info):
305         # UI Updater για το "Status Bar" - αναλαμβάνει την ΟΠΤΙΚΗ ΑΝΑΔΡΑΣΗ - του χρήστη!
306         status_text = f"PLATE: {db_info['plate']} | STATUS: {db_info['status']}"
307         color = "#27ae60" # Default Green
308 
309         if db_info['status'] == "STOLEN": color = "#c0392b"
310         elif db_info['status'] == "WANTED": color = "#d35400"
311 
312         self.root.after(0, lambda: self.status_var.set(status_text))
313         self.root.after(0, lambda: self.lbl_status.config(bg=color, fg="white"))
```

Σχήμα 4.59 update_ui_status_safe()

def add_vehicle(self): Η μέθοδος χρησιμοποιείται για την προσθήκη ενός νέου οχήματος στη ΒΔ μας. Καλείται όταν ο χρήστης πιέσει το κουμπί «Add Vehicle» στο πλευρικό μενού. Με την κλήση της ένα νέο παράθυρο δημιουργείται «παιδί» του κεντρικού παραθύρου (Σχήμα 4.60). Στις γραμμές 317-322 δίνονται παράμετροι για το παράθυρο και με την transient δεσμεύονται τα παράθυρα μεταξύ τους ενώ με την grab_set() ο έλεγχος κλειδώνεται στο παράθυρο παιδί μέχρι αυτό να καταστραφεί. Έτσι ο χρήστης δεν μπορεί να προβεί σε καμία ενέργεια μέχρι να εξαφανιστεί το παράθυρο «παιδί». Προστίθενται στοιχεία στο παράθυρο (ετικέτες, πεδίο εισαγωγής κειμένου, combobox και κουμπί για την αποθήκευση) και παράμετροι για όλα (γραμμές 324-333). Πρέπει να επισημανθεί ότι στο κουμπί δεν έχει προστεθεί κάποια μέθοδος που να καλείται κι αυτό γίνεται σκόπιμα, ο λόγος θα φανεί παρακάτω. Μέσα στη συνάρτησή μας έχουμε «φωλιασμένες» συναρτήσεις:

1 . def _bg_task(plate, status): (γραμμές 340-345) είναι μια συνάρτηση που παίρνει ως ορίσματα μια πινακίδα και την κατάσταση οχήματος. Με την κλήση της καλεί τη μέθοδο add_stolen_vehicle() που είδαμε στην κλάση VehicleService και περνάει την απάντησή της σε δύο μεταβλητές – η πρώτη είναι μια Boolean για την επιτυχία/ αποτυχία της συναλλαγής και η δεύτερη είναι ένα μήνυμα – το οποίο είδαμε αναλυτικά στην προαναφερθείσα κλάση. Στη γραμμή 343 καλείται με τη χρήση της “after” η συνάρτηση “_on_result()”(καθώς πρόκειται να εκτελεστεί από διαφορετικό νήμα) ώστε να περάσει μήνυμα στο UI. Σε περίπτωση εξαίρεσης καλείται πάλι η ίδια μέθοδος απλά με άλλα ορίσματα (345).

2. def _on_result(success, msg): (γραμμές 347-354) όταν κληθεί αυτή η συνάρτηση : εάν στην παράμετρο “success” που δέχτηκε κατά την κλήση της η τιμή της είναι “True” τότε ένα messagebox εμφανίζεται με το μήνυμα της γραμμής 349, απελευθερώνεται η χρήση του πληκτρολογίου και του ποντικιού με την “grab_release()” και το παράθυρο προσθήκης κλεμμένων οχημάτων κλείνει. Σε περίπτωση που δεν έχουμε “True” στην μεταβλητή, τότε εμφανίζεται μήνυμα σφάλματος και το κουμπί της αποθήκευσης ενεργοποιείται.

3. def on_save(): Στις γραμμές 357-358 περνάνε σε μεταβλητές: η πινακίδα που έχει καταχωρήσει ο χρήστης αφού «καθαριστεί» και μετατραπεί σε κεφαλαία , και η επιλογή του χρήστη από το combobox με την κατάσταση του οχήματος. Εάν δεν έχει καταχωρηθεί πινακίδα στο πλαίσιο εισαγωγής κειμένου τότε εμφανίζεται ένα messagebox με προειδοποίηση σφάλματος (γραμμές 360-362). Το κουμπί της αποθήκευσης απενεργοποιείται και το κείμενο μετατρέπεται σε αυτό της γραμμής 364. Στη γραμμή 367 , ξεκινά ένα νέο νήμα το οποίο θα καλέσει την μέθοδο “_bg_task()” με ορίσματα τις μεταβλητές που αποθηκεύτηκαν νωρίτερα, και με daemon= True βεβαιωνόμαστε ότι το νήμα αυτό θα τερματιστεί μαζί με την εφαρμογή – όπως κι αν κλείσει αυτή.

Μετά τις «φωλιασμένες» συναρτήσεις ανατίθεται η συνάρτηση που θα κληθεί όταν πατηθεί το κουμπί της αποθήκευσης (γραμμές 369-370).

```
315     def add_vehicle(self):
316         # Άνοιγμα παραθύρου εισαγωγής νέας πινακίδας οχήματος στη βάση
317         add_win = tk.Toplevel(self.root)
318         add_win.title("Προσθήκη Οχήματος")
319         add_win.geometry("300x250")
320         add_win.resizable(False, False)
321         add_win.transient(self.root)
322         add_win.grab_set()
323 
324         tk.Label(add_win, text="Αριθμός Πινακίδας (π.χ. ABC1234):", font=("Arial", 10)).pack(pady=(20, 5))
325         ent_plate = tk.Entry(add_win, width=20, font=("Arial", 12), justify="center")
326         ent_plate.pack()
327 
328         tk.Label(add_win, text="Κατάσταση Οχήματος:", font=("Arial", 10)).pack(pady=(15, 5))
329         cmb_status = ttk.Combobox(add_win, values=["STOLEN", "WANTED"], state="readonly", font=("Arial", 11), width=18)
330         cmb_status.current(0)
331         cmb_status.pack()
332 
333         btn_save = tk.Button(add_win, text="Αποθήκευση", bg="#c0392b", fg="white", font=("Arial", 10, "bold"), width=15)
334 > #region Comments ···
340         def _bg_task(plate, status):
341             try:
342                 success, msg = VehicleService.add_stolen_vehicle(plate, status)
343                 self.root.after(0, lambda: _on_result(success, msg))
344             except Exception as e:
345                 self.root.after(0, lambda: _on_result(False, f"Σφάλμα Βάσης: {str(e)}"))
346 
347         def _on_result(success, msg):
348             if success:
349                 messagebox.showinfo("Επιτυχία", msg, parent=add_win)
350                 add_win.grab_release()
351                 add_win.destroy()
352             else:
353                 messagebox.showerror("Σφάλμα", msg, parent=add_win)
354                 btn_save.config(state=tk.NORMAL, text="Αποθήκευση")
355 
356         def on_save():
357             plate = ent_plate.get().strip().upper()
358             status = cmb_status.get().strip()
359 
360             if not plate:
361                 messagebox.showwarning("Προσοχή", "Το πεδίο της πινακίδας δεν μπορεί να είναι κενό!", parent=add_win)
362                 return
363 
364             btn_save.config(state=tk.DISABLED, text="Παρακαλώ περιμένετε...")
365 
366             # Εκκίνηση του thread αποθήκευσης
367             threading.Thread(target=_bg_task, args=(plate, status), daemon=True).start()
368 
369         btn_save.config(command=on_save)
370         btn_save.pack(pady=25)
```

Σχήμα 4.60 add_vehicle()

def show_logs(self): Όταν καλείται η μέθοδος δημιουργείται ένα νέο παράθυρο (Σχήμα 4.61 γραμμή 374) κατόπιν ορίζεται τίτλος και προστίθεται scroll bar σε αυτό εκχωρώντας ένα μέγεθος πλάτους 70 και ύψους 20 **χαρακτήρων**. Του δίνεται η δυνατότητα να μπορεί να μεγεθυνθεί και ορίζεται (γραμμή 379) το αρχείο προέλευσης. Έπειτα στις γραμμές 381-384 ανοίγει το αρχείο με την "with open()" η οποία διασφαλίζει ότι αν κάτι πάει στραβά το αρχείο θα κλείσει σωστά- διαβάζεται και περνιέται στη μεταβλητή της γραμμής 383, και κατόπιν περνάει το περιεχόμενο της μεταβλητής στο παράθυρο που δημιουργήθηκε νωρίτερα. Στην περίπτωση που υπάρξει εξαίρεση τότε εισάγεται μήνυμα σφάλματος (γραμμή 386). Τέλος, στην γραμμή 387 : για να μην επιτρέψουμε στους χρήστες να μπορούν να επεξεργαστούν τα δεδομένα των εντοπισμών «κλειδώνουμε» το πληκτρολόγιο με αυτή την συνάρτηση καθώς δεσμεύουμε κάθε πάτημα πλήκτρου – αφού εκτελείται η συνάρτηση lamda που επιστρέφει "break". Στην βιβλιοθήκη Tkinter – η επιστροφή "break" ακυρώνει την ενέργεια του πλήκτρου. Με αυτόν τον τρόπο επιτρέπουμε στους χρήστες να πλοηγηθούν με το ποντίκι χωρίς να αλλοιώσουν το περιεχόμενο του αρχείου. Επίσης, με την γραμμή 388 με το άνοιγμα του log γίνεται μετάβαση στις τελευταίες εγγραφές.

def on_exit(self): Όταν ο χρήστης πατήσει το κουμπί "EXIT" στο UI μας , εκτελείται αυτή η μέθοδος ώστε να πραγματοποιηθεί η έξοδος από την εφαρμογή. Πρώτα εμφανίζεται ένα messagebox κι εφόσον ο χρήστης επιβεβαιώσει ότι επιθυμεί να κλείσει την εφαρμογή τότε – γίνεται logout, κλείνει η κάμερα, καταστρέφεται το παράθυρο και τερματίζεται η διεργασία (γραμμές 391- 396).

```
372     def show_logs(self):
373         # Εμφάνιση του detection.log στο UI σε νέο παράθυρο.
374         win = tk.Toplevel(self.root)
375         win.title("Detection History")
376         txt = scrolledtext.ScrolledText(win, width=70, height=20)
377         txt.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)
378 
379         logs_path = Labyrinth.get_results_file_path('detections.log')
380 
381         try:
382             with open(logs_path, "r") as f:
383                 data = f.read().strip()
384             txt.insert(tk.INSERT, data if data else "No logs yet.")
385         except:
386             txt.insert(tk.INSERT, "No logs found.")
387         txt.bind("<Key>", lambda e: "break")
388         txt.yview(tk.END)
389 
390     def on_exit(self):
391         # Exit button - για το ομαλό κλείσιμο της εφαρμογής.
392         if messagebox.askokcancel("Exit", "Quit system?"):
393             AuthService.logout()
394             self.stop_camera()
395             self.root.destroy()
396             os._exit(0)
```

Σχήμα 4.61 show_logs() & on_exit()

def open_user_management(self): Η μέθοδος αυτή καλείται όταν ο χρήστης πατήσει το κουμπί “Manage Users” από το UI μας. Αρχικά γίνεται ο έλεγχος ρόλου του συνδεδεμένου χρήστη καλώντας τον SessionManager που έχει αποθηκευμένα τα στοιχεία από το login. Εξ ορισμού επιτρέπεται το CRUD χρηστών μόνο από administrators. Οποιοσδήποτε προσπαθήσει να συνδεθεί με διαφορετικά δικαιώματα τότε ένα μήνυμα σφάλματος εμφανίζεται σε messagebox και τερματίζει η συνάρτηση (Σχήμα 4.62 γραμμές 405-407). Εάν ο χρήστης είναι διαχειριστής ανοίγει νέο παράθυρο «παιδί», περνάνε ρυθμίσεις – τίτλοι κλπ, «πακετάρεται» με το μητρικό παράθυρο και κλειδώνει ο έλεγχος σε αυτό μέχρι να τερματιστεί. Στις γραμμές 415-416 δημιουργείται ένα notebook με tabs στο παράθυρο «παιδί» και πιάνει όλο το διαθέσιμο χώρο του.

Στις γραμμές 419-433 γίνεται η σύνθεση του πρώτου tab (προστίθενται τα στοιχεία – ετικέτες, textboxes , combobox) και «κουμπώνουν» στο tab. Στις γραμμές 435 – 452 ορίζεται η «φωλιασμένη» συνάρτηση save_user().

def save_user(): Αποθηκεύει σε 3 μεταβλητές τα στοιχεία που έχει εισάγει ο χρήστης στα αντίστοιχα πεδία του UI (γραμμές 436-439). Γίνεται έλεγχος εάν τα πεδία του ονόματος χρήστη και του κωδικού είναι κενά και εάν είναι διακόπτεται η εκτέλεση της συνάρτησης. Εάν δεν είναι κενά, τότε καλείται η συνάρτηση create_user() από την κλάση UserManagementService με ορίσματα τις τιμές των μεταβλητών και το αποτέλεσμα , μαζί με το μήνυμα αποθηκεύονται στις αντίστοιχες μεταβλητές. Στις γραμμές 446 μετά τον επιτυχή έλεγχο (η success έχει τιμή True) εμφανίζεται ένα messagebox με την επιβεβαίωση, καθαρίζουν τα πεδία ονόματος χρήστη και κωδικού και κατόπιν καλείται η συνάρτηση load_users() που φορτώνει τους «μη διαχειριστές» και θα δούμε αμέσως μετά. Σε περίπτωση σφάλματος εμφανίζεται messagebox σφάλματος για να ενημερωθεί ο χρήστης (γραμμή 452).

Στη γραμμή 454 δημιουργείται το κουμπί αποθήκευσης χρήστη- του ανατίθεται η συνάρτηση που θα κληθεί όταν το πατήσει ο χρήστης και «κουμπώνεται» επάνω στο πλαίσιο.

```
398     def open_user_management(self):
399         #region Comments ···
405         if SessionManager.get_role() != "ADMINISTRATOR":
406             messagebox.showerror("Access Denied", "Μόνο ο ΔΙΑΧΕΙΡΙΣΤΗΣ έχει πρόσβαση.", parent=self.root)
407             return
408 
409         win = tk.Toplevel(self.root)
410         win.title("Manage Users")
411         win.geometry("550x450")
412         win.transient(self.root)
413         win.grab_set()
414 
415         tabs = ttk.Notebook(win)
416         tabs.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)
417 
418         # Πρώτο tab : CREATE
419         tab_create = tk.Frame(tabs)
420         tabs.add(tab_create, text="Create User")
421 
422         tk.Label(tab_create, text="Username:").pack(pady=(20, 5))
423         new_username_input = tk.Entry(tab_create, width=30)
424         new_username_input.pack()
425 
426         tk.Label(tab_create, text="Password:").pack(pady=5)
427         new_paswd_input = tk.Entry(tab_create, width=30, show="*")
428         new_paswd_input.pack()
429 
430         tk.Label(tab_create, text="Role:").pack(pady=5)
431         combobox_role = ttk.Combobox(tab_create, values=["supervisor", "user"], state="readonly")
432         combobox_role.current(0)
433         combobox_role.pack()
434 
435         def save_user():
436             # Αποθήκευση νέου χρήστη
437             u = new_username_input.get().strip()
438             p = new_paswd_input.get().strip()
439             r = combobox_role.get().strip()
440 
441             if not u or not p:
442                 messagebox.showwarning("Warning", "Τα πεδία δεν μπορεί να είναι κενά!", parent=win)
443                 return
444 
445             success, message = UserManagementService.create_user(u, p, r)
446             if success:
447                 messagebox.showinfo("Success", message, parent=win)
448                 new_username_input.delete(0, tk.END)
449                 new_paswd_input.delete(0, tk.END)
450                 load_users()
451             else:
452                 messagebox.showerror("Error", message, parent=win)
453 
454         tk.Button(tab_create, text="Save User", bg="#27ae60", fg="white", command=save_user).pack(pady=25)
```

Σχήμα 4.62 open_user_management() (1/3)

Στο Σχήμα 4.63(γραμμές 457-468) δημιουργείται ένα δεύτερο tab για τη διαχείριση των χρηστών. Στη γραμμή 460 τα αναγνωριστικά των στηλών και με την γραμμή 461 κατασκευάζεται ο πίνακας. Με την παράμετρο “show=headings” λέμε στο ttk να αγνοήσει την κλασσική προβολή “tree” και να δείξει τα δεδομένα απλά ως πίνακα. Βάζουμε το κείμενο σε κάθε κεφαλίδα (γραμμές 462-464) ρυθμίζουμε τη διάταξη των στηλών (γραμμές 465-467) και στο τέλος με τη γραμμή 468 τα συνθέτουμε όλα μαζί. Ακολουθούν οι «φωλιασμένες» μέθοδοι :

1. **def load_users():** με την κλήση της διατρέχουμε όλες τις εγγραφές του πίνακά μας και τις σβήνουμε μία-μία (γραμμές 472- 473). Έπειτα καλείται η μέθοδος “get_manageable_users()” από την κλάση UserManagementService και τα αποτελέσματα που επιστρέφει τα περνάμε στη μεταβλητή της

γραμμής 474. Τέλος , διατρέχουμε όλα τα αποτελέσματά μας και για κάθε ένα από αυτά γίνεται η εισαγωγή του στην αντίστοιχη στήλη του πίνακά μας (γραμμή 476).

2. **def delete_user():** Όταν ο διαχειριστής επιλέξει έναν από τους χρήστες από τον πίνακα, η επιλογή του αποθηκεύεται στη μεταβλητή της γραμμής 480. Στην περίπτωση που δεν έχει επιλεγεί κανένας χρήστης τότε η μέθοδος τερματίζεται. Στην περίπτωση που έχει επιλεγεί τουλάχιστον ένας χρήστης τότε στη γραμμή 482 επιλέγεται ο πρώτος από αυτούς, ανακτώνται κι αποθηκεύονται σε μεταβλητές τα uid και username του. Έπειτα, στη γραμμή 485 ζητείται επιβεβαίωση για τη διαγραφή μέσω messagebox. Εάν ο διαχειριστής επιβεβαιώσει , καλείται η μέθοδος “delete_user(uid)” από την κλάση UserManagementService και τα αποτελέσματά της αποθηκεύονται στις μεταβλητές της γραμμής 486. Σε περίπτωση επιτυχίας- έχουμε νέο messagebox που ενημερώνει τον χρήστη για την επιτυχή διαγραφή και κατόπιν φορτώνονται οι «μη διαχειριστές» εκ νέου στον πίνακα (γραμμή 489). Διαφορετικά εμφανίζεται messagebox με μήνυμα σφάλματος.

```
456         # Δεύτερο tab : MANAGE
457         tab_manage = tk.Frame(tabs)
458         tabs.add(tab_manage, text="Manage Users")
459
460         columns = ("id", "username", "role")
461         tree = ttk.Treeview(tab_manage, columns=columns, show="headings", height=10)
462         tree.heading("id", text="ID")
463         tree.heading("username", text="Username")
464         tree.heading("role", text="Role")
465         tree.column("id", width=50, anchor=tk.CENTER)
466         tree.column("username", width=200, anchor=tk.W)
467         tree.column("role", width=150, anchor=tk.CENTER)
468         tree.pack(pady=10, fill=tk.BOTH, expand=True)
469
470         def load_users():
471             # Φόρτωση των χρηστών της ΒΔ μας - ΟΧΙ ADMINS!! το χειριζόμαστε από το user_services.py
472             for item in tree.get_children():
473                 tree.delete(item)
474             users = UserManagementService.get_manageable_users()
475             for u in users:
476                 tree.insert("", tk.END, values=(u['id'], u['username'], u['role']))
477
478         def delete_user():
479             # Διαγραφή χρήστη από τη βάση μας
480             selected = tree.selection()
481             if not selected: return
482             item = tree.item(selected[0])
483             uid, uname = item['values'][0], item['values'][1]
484
485             if messagebox.askyesno("Confirm", f"Διαγραφή του χρήστη: {uname};", parent=win):
486                 success, message = UserManagementService.delete_user(uid)
487                 if success:
488                     messagebox.showinfo("Success", f"Ο χρήστης {uname} διαγράφηκε.", parent=win)
489                     load_users()
490                 else:
491                     messagebox.showerror("Error", message, parent=win)
```

Σχήμα 4.63 open_user_management() (2/3)

3. **def update_user():** Ακριβώς όπως στην περίπτωση της μεθόδου “delete_user()” έχουμε έλεγχο για την επιλογή ή όχι κάποιου χρήστη και στην περίπτωση που είναι αρνητική γίνεται έξοδος από τη μέθοδο ενώ διαφορετικά αποθηκεύονται τα στοιχεία του χρήστη στις μεταβλητές της γραμμής 498. (Σχήμα 4.64 γραμμές 495-498). Στις γραμμές 500- 504 δημιουργείται ένα νέο παράθυρο «παιδί» - του δίνονται ρυθμίσεις και «δένεται» με το μητρικό παράθυρο δεσμεύοντας τον έλεγχο. Εν συνεχεία , στις γραμμές 506-513 προστίθενται στοιχεία – ετικέτες, ένα textbox για την εισαγωγή κωδικού κι ένα

combobox για την επιλογή ρόλου- και στο τέλος συνθέτονται όλα μαζί. Το διαφορετικό είναι στη γραμμή 512 όπου σαν προεπιλογή στο combobox ορίζεται ο ρόλος που ήδη έχει ο χρήστης. Έπειτα στην ήδη «φωλιασμένη» μας μέθοδο συναντάμε ακόμη μια «φωλιασμένη».

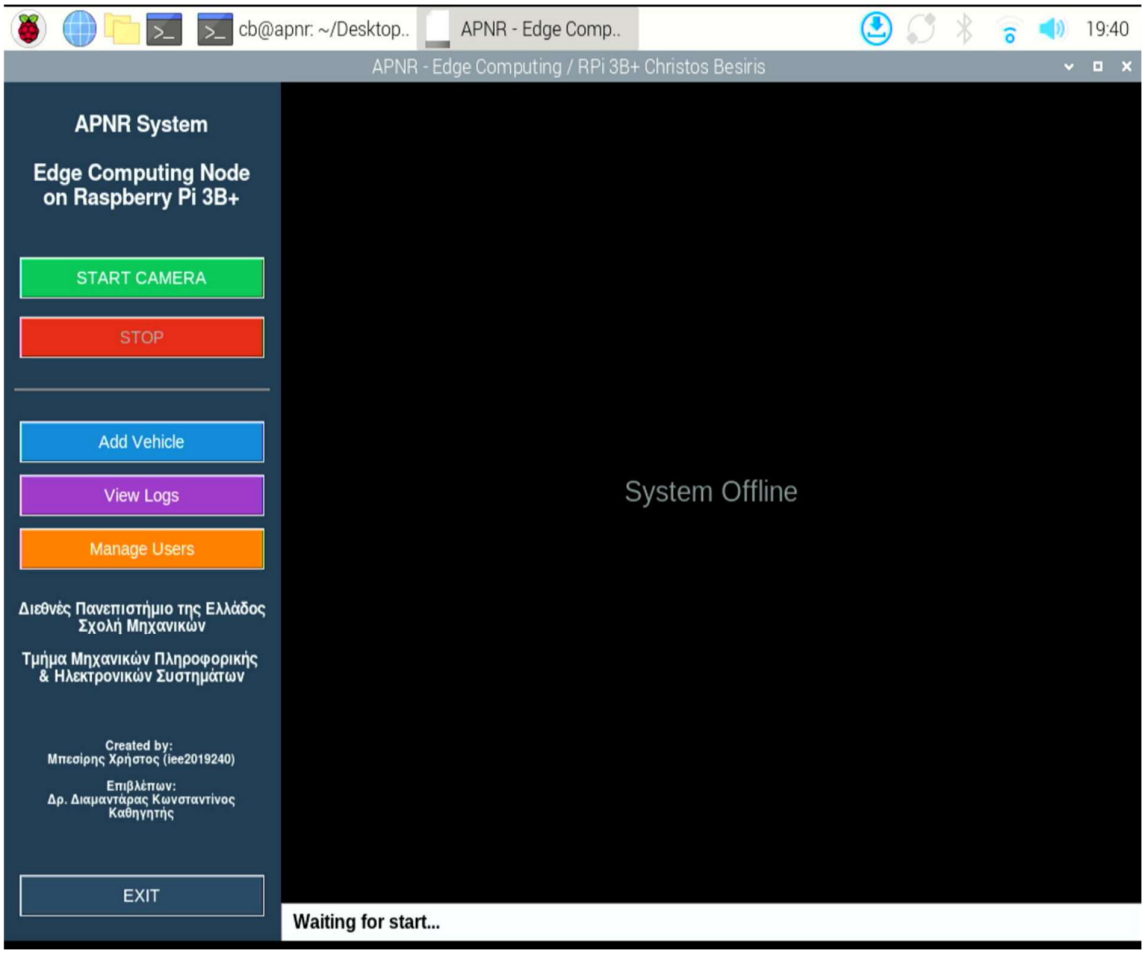
1. def save_update(): Στις γραμμές 517-518 αποθηκεύονται σε μεταβλητές τα στοιχεία που έχει εισάγει ο διαχειριστής, αφού «καθαρίστηκαν» από μη εκτυπώσιμους χαρακτήρες. Μετά την επιβεβαίωση στο messagebox που εμφανίζεται (γραμμή 520) καλείται η μέθοδος "update_user()" της κλάσης UserManagementService και το αποτέλεσμα αποθηκεύεται σε δύο μεταβλητές- μια Boolean και ένα κείμενο. Με την επιτυχία δημιουργείται ένα νέο messagebox το οποίο παρέχει ενημέρωση στον διαχειριστή, καταστρέφεται το παράθυρο που άνοιξε για την «ενημέρωση στοιχείων χρήστη» και φορτώνονται εκ νέου οι «μη διαχειριστές» στον πίνακα (γραμμές 522-525). Διαφορετικά ο διαχειριστής παίρνει μήνυμα σφάλματος (γραμμή 527).

Με τη γραμμή 529 προστίθεται ένα κουμπί για την αποθήκευση ενημέρωσης χρήστη το οποίο όταν πατηθεί καλείται η μέθοδος "save_update()".

Τέλος, με τις γραμμές 531 και 532 συντίθεται και κουμπώνει το tab για τη διαχείριση χρηστών ενώ με τις γραμμές 533 & 534 προστίθενται τα δύο κουμπιά της ενημέρωσης στοιχείων χρήστη και της διαγραφής χρήστη καλώντας τις αντίστοιχες μεθόδους όταν πατηθούν "update_user()" και "delete_user()". Κατόπιν , με τη γραμμή 536 καλείται η μέθοδος "load_users()". Εδώ τελειώνει η «σύνθεση» του γραφικού περιβάλλοντος με το τελικό αποτέλεσμα να αποτυπώνεται στο Σχήμα 4.65.

```
493         def update_user():
494             # Ενημέρωση στοιχείων χρήστη
495             selected = tree.selection()
496             if not selected: return
497             item = tree.item(selected[0])
498             uid, uname, urole = item['values'][0], item['values'][1], item['values'][2]
499 
500             upd_win = tk.Toplevel(win)
501             upd_win.title(f"Update: {uname}")
502             upd_win.geometry("300x250")
503             upd_win.transient(win)
504             upd_win.grab_set()
505 
506             tk.Label(upd_win, text="New Password\n(αφήστε κενό για διατήρηση παλιού):").pack(pady=10)
507             user_new_paswd = tk.Entry(upd_win, width=25, show="*")
508             user_new_paswd.pack()
509 
510             tk.Label(upd_win, text="Change Role:").pack(pady=10)
511             user_new_role = ttk.Combobox(upd_win, values=["USER", "SUPERVISOR"], state="readonly")
512             user_new_role.set(urole)
513             user_new_role.pack()
514 
515             def save_update():
516                 # Αποθήκευση ενημερωμένων στοιχείων
517                 new_p = user_new_paswd.get().strip()
518                 new_r = user_new_role.get().strip()
519 
520                 if messagebox.askyesno("Confirm", f"Ενημέρωση του χρήστη: {uname};", parent=win):
521                     success, message = UserManagementService.update_user(uid, new_p, new_r)
522                     if success:
523                         messagebox.showinfo("Success", f"Ο χρήστης {uname} ενημερώθηκε.", parent=win)
524                         upd_win.destroy()
525                         load_users()
526                     else:
527                         messagebox.showerror("Error", message, parent=win)
528 
529             tk.Button(upd_win, text="Save Update", bg="#2980b9", fg="white", command=save_update).pack(pady=20)
530 
531         btn_frame = tk.Frame(tab_manage)
532         btn_frame.pack(pady=5)
533         tk.Button(btn_frame, text="Edit", bg="#f39c12", fg="white", command=update_user).pack(side=tk.LEFT, padx=10)
534         tk.Button(btn_frame, text="Delete", bg="#c0392b", fg="white", command=delete_user).pack(side=tk.LEFT, padx=10)
535 
536         load_users()
```

Σχήμα 4.64 update_user()



Σχήμα 4.65 Γραφικό περιβάλλον gui_main

4.3.8 main.py

Το αρχείο main.py είναι το σημείο εισόδου στην εφαρμογή APNR System. Στην αρχή του εισάγεται η βιβλιοθήκη tkinter για τη δημιουργία του UI και οι κλάσεις “LoginWindow”, “MainWindow” και “Container” από τα αντίστοιχα αρχεία που είδαμε σε προγενέστερες ενότητες (Σχήμα 4.66). Υλοποιώντας αποτελεσματικά το DI βλέπουμε ότι στο σημείο εισόδου ο κώδικάς μας δεν υπερβαίνει τις 10 γραμμές. Στις γραμμές 7-23 ορίζεται η μέθοδος “launch_main_app()”.

def launch_main_app(): Στη γραμμή 9 «χτίζεται» ένα νέο παράθυρο από το tkinter. Μετά σε μια try-except γίνεται η προσπάθεια «χτισίματος» του κορμού της εφαρμογής- φτιάχνοντας το “Container()” στη γραμμή 13. Σε περίπτωση που κάτι αποτύχει δημιουργείται ένα messagebox το οποίο ενημερώνει τον χρήστη για το σφάλμα και διακόπτεται η εκτέλεση της εφαρμογής. Εάν όλα πάνε καλά, γίνεται εκκίνηση της εφαρμογής με το «πάντρεμα» στην MainWindow του UI και του app_container. Στην γραμμή 23 ξεκινάει ο βρόγχος του tkinter.

Με την γραμμή 25 – η γλώσσα δίνει το έναυσμα για εκτέλεση εφόσον το αρχείο καλείται απευθείας. Τότε εκτελείται ο κώδικας της γραμμής 28 ο οποίος στην ουσία περνάει τον έλεγχο στο “LoginWindow” με την προϋπόθεση ότι μόλις ο χρήστης της εφαρμογής κάνει επιτυχημένη είσοδο στο σύστημα τότε αυτόματα θα επιστρέψει ο έλεγχος στο main.py όπου και θα κληθεί η συνάρτηση “launch_main_app()”.

```python
import tkinter as tk
from app.ui.gui_login import LoginWindow
from app.ui.gui_main import MainWindow
from app.core.container import Container


def launch_main_app():
    # Η συνάρτηση που καλείται μετά από επιτυχημένο login και ξεκινάει το κύριο παράθυρο.
    root = tk.Tk()
    # Αρχικοποίηση Container (Inference + Services)
    # Εδώ φορτώνονται τα μοντέλα ONNX και η σύνδεση με τη ΒΔ
    try:
        app_container = Container()
    except Exception as e:
        from tkinter import messagebox
        messagebox.showerror("Initialization Error", f"Αποτυχία εκκίνησης συστήματος: {e}")
        return

    # Εκκίνηση MainWindow
    # Περνάμε το container ώστε το UI να έχει πρόσβαση στο AI και τη Βάση
    app = MainWindow(root, "APNR - Edge Computing / RPi 3B+ Christos Besiris", app_container)

    root.mainloop()

if __name__ == "__main__":
    # Το πρόγραμμα ξεκινάει πάντα από το Login
    # Το LoginWindow θα καλέσει τη launch_main_app μόνο αν τα credentials είναι σωστά
    LoginWindow(on_success=launch_main_app)
```

Σχήμα 4.66 main.py

4.3.9 create_admin.py

Το αρχείο αυτό περιέχει ουσιαστικά μόνο μια μέθοδο που καλείται με την εκτέλεση του αρχείου.

def create_admin_user(): Αρχικά εισάγονται η κλάση DBConnectionManager , οι κλάσεις αντικειμένων User και UserRole και η μέθοδος hash_password από τα αντίστοιχα αρχεία (Σχήμα 4.67). Στις γραμμές 11-15 γίνεται το «χτίσιμο» της ΒΔ , η δημιουργία μιας νέας συνεδρίας με αυτή και η ανάθεση τιμών στις μεταβλητές μας. Έπειτα γίνεται έλεγχος αν υπάρχει ήδη στη βάση μας χρήστης με το ίδιο username (γραμμές 17-21) και σε περίπτωση που υπάρχει -επιστρέφεται αντίστοιχο μήνυμα και η εκτέλεση του προγράμματος τερματίζεται. Εάν δεν υπάρχει τότε ο κωδικός περνάει μέσα από την μέθοδο hash_password και κατόπιν δημιουργείται ένας νέος χρήστης με τις αντίστοιχες ιδιότητες των γραμμών 25-29. Τέλος, γίνεται η προσθήκη του χρήστη στη βάση στις γραμμές 31-32 και τυπώνεται αντίστοιχο μήνυμα στο τερματικό μας. Σε περίπτωση οποιασδήποτε αποτυχίας τότε έχουμε αναδρομή της συνεδρίας στην προγενέστερη κατάσταση. Ανεξάρτητα του αποτελέσματος τερματίζεται η συνεδρία στη γραμμή 40. Στις γραμμές 42-43 ουσιαστικά με την εκτέλεση του αρχείου ελέγχεται το «σημείο εισόδου» - πως κλήθηκε το αρχείο και καλείται η συνάρτηση που μόλις περιγράψαμε.

```python
from app.database.connection import DBConnectionManager
from app.database.dbmodels import User, UserRole
from app.utils.hasher import hash_password

#Κλάση για την αρχικοποίηση της ΒΔ και τη δημιουργία ενός χρήστη ADMINISTRATOR με username:"admin"
#και password:"iee2019240". Η κλάση ελέγχει αν ο χρήστης υπάρχει ήδη και αν όχι, τον δημιουργεί με το ρόλο ADMIN.


def create_admin_user():
    print("Αρχικοποίηση σύνδεσης με τη βάση δεδομένων...")
    DBConnectionManager.setup_db()
    session = DBConnectionManager.get_session()

    new_username = "admin"
    new_password = "iee2019240"

    try:
        existing_user = session.query(User).filter_by(username=new_username).first()
        if existing_user:
            print(f"ERROR:  Ο χρήστης '{new_username}' υπάρχει ήδη στη βάση.")
            return

        hashed_pw = hash_password(new_password)

        new_admin = User(
            username=new_username,
            password=hashed_pw,
            role=UserRole.ADMIN
        )

        session.add(new_admin)
        session.commit()

        print(f"SUCCESS ! Ο χρήστης '{new_username}' δημιουργήθηκε.")

    except Exception as e:
        session.rollback()
        print(f"ERROR : Σφάλμα κατά τη δημιουργία του χρήστη: {e}")
    finally:
        session.close()

if __name__ == "__main__":
    create_admin_user()
```

Σχήμα 4.67 Περιεχόμενα αρχείου create_admin.py

4.3.10 __init__.py

Σε κάθε φάκελο του έργου μας , αναφέρεται ότι συμπεριλαμβάνεται ένα αρχείο με αυτό το όνομα χωρίς όμως να δείξουμε τα περιεχόμενά του. Ο λόγος είναι απλός: το αρχείο αυτό το έχουμε τελείως **άδειο.** Οι διάφορες εκδόσεις της γλώσσας και κάποιοι editors (κάποιες εκδόσεις τους) όπως είναι και το VS Code, δεν μπορούν να «δουν» τα περιεχόμενα ενός υποφακέλου ακόμη κι αν έχει εισαχθεί ολόκληρος σαν project. Αν και μετά την έκδοση 3.3 το πρόβλημα έχει ξεπεραστεί – εάν ο προγραμματιστής θέλει να χρησιμοποιήσει έτοιμα εργαλεία μετατροπής του προγράμματος σε αρχείο installer ή απευθείας ".exe" τότε χωρίς το αρχείο αυτό σε κάθε φάκελο δε θα μπορέσουν να συμπεριληφθούν τα περιεχόμενα και η μετατροπή θα αποτύχει. Η παρουσία του μετατρέπει τον απλό φάκελο σε Package για την Python.

4.3.11 Βιβλιοθήκες

Το APNR System χρησιμοποιεί ένα σύνολο βιβλιοθηκών για να γίνει πιο εύκολα και αποτελεσματικά η δουλειά. Κάποιες από αυτές είναι προεγκατεστημένες με τη γλώσσα ενώ κάποιες άλλες απαιτούν ξεχωριστή εγκατάσταση για να τις εκμεταλλευτούμε.

4.3.11.1 Προεγκατεστημένες βιβλιοθήκες

tkinter: χειρίζεται όλο το γραφικό περιβάλλον του χρήστη μας UI. Είναι εγγενής στην γλώσσα αλλά πρέπει να επιβεβαιώσουμε την εγκατάσταση στην πλατφόρμα μας- σε επίπεδο λειτουργικού συστήματος. (μέσω τερματικού "sudo apt-get install python3-tk")

threading: σηκώνει το βαρύ φορτίο του μοιράσματος του inference και των «βαριών» εργασιών της εφαρμογής. Χωρίς αυτή , δε θα ήταν δυνατή η εκτέλεση μιας εφαρμογής με αποτελέσματα.

time: η βιβλιοθήκη χρονόμετρο – τη χρησιμοποιούμε για να «ξεκουράζουμε» τον επεξεργαστή αλλά και για να ρυθμίζουμε τα χρονικά διαστήματα μεταξύ του OCR.

tempfile: την χρησιμοποιούμε για την δημιουργία προσωρινών αρχείων εικόνας για να μην «κρασάρουμε» το σύστημα προσπαθώντας να τα διαβάζουμε από την μνήμη RAM.

os: η βιβλιοθήκη «βοηθός» στην πλοήγηση της εφαρμογής στο σύστημα φακέλων μας.

4.3.11.2 Εξωτερικές βιβλιοθήκες

Οι παρακάτω βιβλιοθήκες πρέπει να εγκατασταθούν μέσω του package manager "pip" (ή "pip3") ανάλογα με την έκδοση.

opencv-python (cv2): το Α και το Ω της υπολογιστικής όρασης. Χωρίς αυτή δεν θα υλοποιούνταν ποτέ η εφαρμογή μας. Χειρίζεται την εικόνα και τα αποτελέσματα του inference. Εγκατάσταση μέσω του pip με "pip install opencv-python" ή σε επίπεδο λειτουργικού μέσω apt: "sudo apt install python3-opencv".

Pillow (PIL): το ενδιάμεσο κομμάτι – ο μεταφραστής ανάμεσα στο tkinter και στην numpy. Αλλάζει και μετατρέπει του πίνακες με τα χρωματικά κανάλια όποτε κι όπως απαιτείται. Εγκατάσταση μέσω pip με "pip install Pillow".

numpy: Ο μαθηματικός της εφαρμογής. Όλες οι πράξεις που γίνονται στο κομμάτι του inference απαιτούν εκτεταμένη χρήση πράξεων πινάκων. Ειδικότητά της.

onnxruntime: η «μηχανή» του inference. Παντρεύει υλικό και λογισμικό με τον καλύτερο και αποδοτικότερο τρόπο για να έχουμε αποτελέσματα χωρίς να «κάψουμε» το υλικό μας.

fast-plate-ocr: η βιβλιοθήκη που έχει μέσα της τις εξαρτήσεις και τα μοντέλα που χρειαζόμαστε για το κομμάτι του OCR.

bcrypt: η βιομηχανική βιβλιοθήκη ασφάλειας της εφαρμογής. Η διαχείριση κωδικών χωρίς αυτή θα ήταν μια ευπάθεια από μόνη της. Με το ενσωματωμένο salting αυξάνει την ασφάλεια σε επαγγελματικό επίπεδο με δυο τρεις γραμμές κώδικα.

gpiozero: η βιβλιοθήκη που χειρίζεται το υλικό μας, τις ακίδες I/O. Χωρίς αυτή δε θα είχαμε ηχητική ανάδραση στην εφαρμογή – κάτι που βελτιώνει την εμπειρία χρήστη.

SQLAlchemy: είναι ένας σύγχρονος «χαρτογράφος- μεταφραστής». Μπαίνοντας ανάμεσα στον κώδικα και στη βάση δεδομένων μας επιτρέπει την επικοινωνία ανεξάρτητα από το είδος βάσης που έχουμε. Το μεγάλο πλεονέκτημα όμως είναι ότι «απαγγιστρώνεται» η εφαρμογή από «ξερές» εντολές της SQL οπότε «σβήνει» την πιθανότητα να εκτεθεί η βάση μας σε καταστροφικά σφάλματα (από μια σύνδεση που δεν διαχειρίστηκε σωστά το πρόγραμμά μας) καθώς για όλα τα διαδικαστικά έχει λύσεις. Το μεγαλύτερο όφελος – εκτός από το προγραμματιστικό κομμάτι – είναι η εξάλειψη του SQL Injection.

4.3.12 Εκτέλεση της εφαρμογής

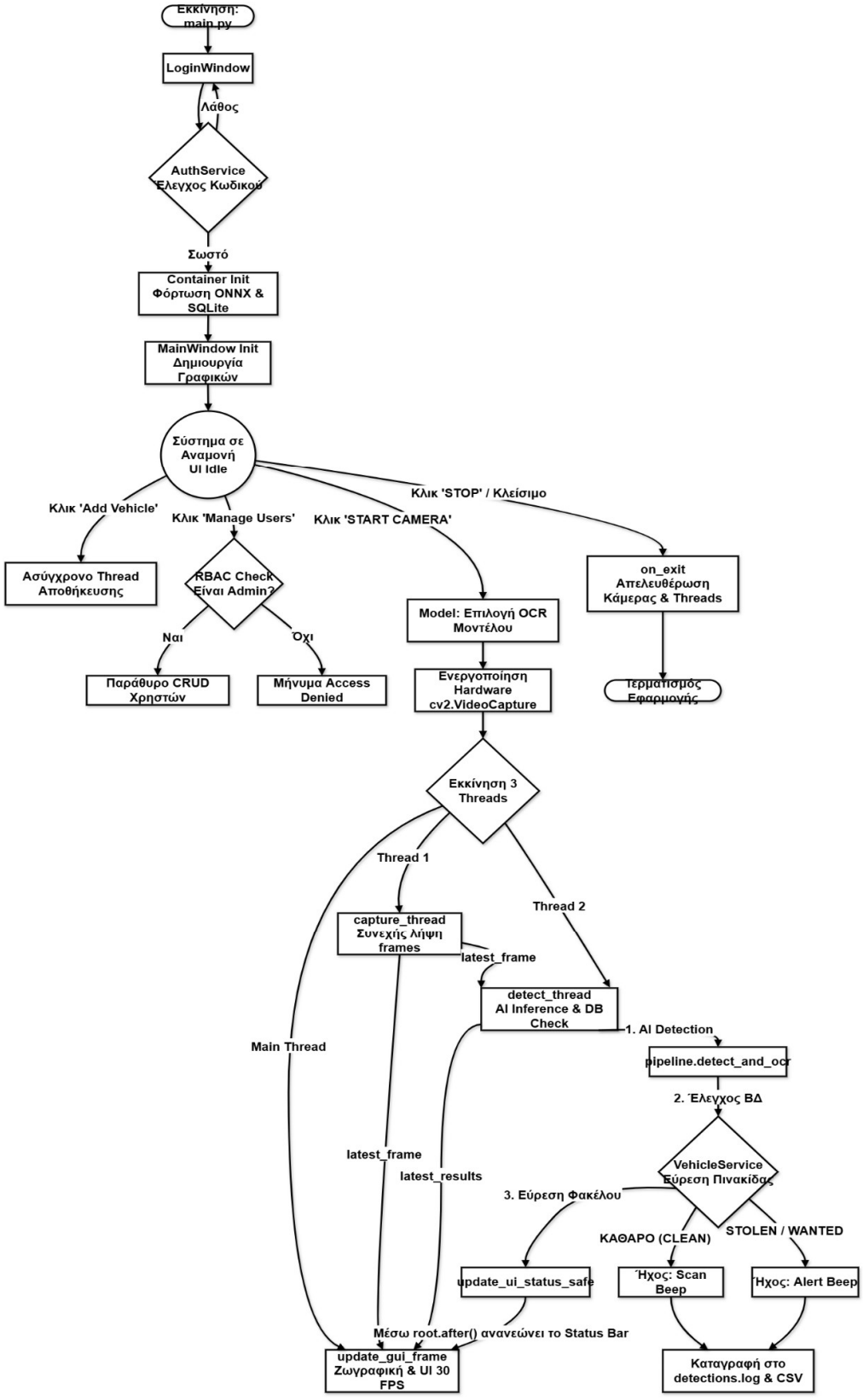
Καθώς η εφαρμογή έχει στηθεί με πολλαπλά επίπεδα φακέλων και αρχεία για να μπορέσει να τρέξει θα πρέπει ο χρήστης να πλοηγηθεί στον **ριζικό** φάκελο του APNR System και ανάλογα με την έκδοση της γλώσσας που έχει εγκατεστημένη στο σύστημά του να εκτελέσει την εξής εντολή: « python(3) -m main". Ο αριθμός «3» αντιστοιχεί για την έκδοση Python 3 την οποία χρησιμοποιήσαμε εμείς οπότε για εμάς η εντολή μετατρέπεται σε «python3 -m main».

Μια συνοπτική παρουσίαση της εφαρμογής είναι διαθέσιμη ως πόρος στην πλατφόρμα YouTube:

Βίντεο : «APNR System Short Presentation»

Απευθείας σύνδεσμος : **https://youtu.be/HZb-GXtwJ2Y**

4.4 Διάγραμμα Ροής του APNR System



Σχήμα 4.68 Διάγραμμα Ροής APNR System

4.5 Επίλογος

Στο κεφάλαιο αυτό έγινε μια αναλυτική παρουσίαση όλων των φακέλων και των αρχείων που συνθέτουν την εφαρμογή APNR System. Παρουσιάστηκαν όλες οι μέθοδοι του προγράμματος, οι κλάσεις ,η δομή της ,το διάγραμμα ροής της και οι βιβλιοθήκες που χρησιμοποιεί.

Κεφάλαιο 5ο: Συμπεράσματα – μελλοντικές βελτιώσεις

Ο σκοπός της εργασίας ήταν η δημιουργία μιας απλής εφαρμογής που θα «τρέχει» μοντέλα και με τη χρήση της υπολογιστικής όρασης θα εντοπίζει οχήματα που απασχολούν. Μετά από πολύ κόπο, πολλές ώρες με «κολλήματα» και «κρασαρίσματα» στην υπολογιστικά «φτωχή» πλατφόρμα μας και βουνά προβλημάτων καταφέραμε αυτό που θεωρούνταν δύσκολο. Ξεπερνώντας την απλή υλοποίηση , προστέθηκαν επιπλέον λειτουργίες στην εφαρμογή – προκειμένου να αγγίξουμε τα επαγγελματικά στάνταρ των ημερών μας. Κάποιες λειτουργίες μπορεί να φαίνονται στο άπειρο μάτι ως «υπερβολικές» πλην όμως κάθε αρχείο και κάθε συνάρτηση κατέληξαν σε αυτή τη μορφή μέσα από δοκιμές – πολλές δοκιμές – και άλλα τόσα λάθη. Μετά από μια ολόκληρη διετία ενασχόλησης μπορούμε πλέον να πούμε ότι τόσο η εφαρμογή όσο και το υλικό ξεπέρασαν κάθε προσδοκία. Ο επιμερισμός των εργασιών στα νήματα επέτρεψε την δημιουργία μιας εφαρμογής που δουλεύει απροβλημάτιστα , με μόλις 1 GB μνήμης RAM. Η απόκριση είναι στιγμιαία με αρκετά καλά αποτελέσματα στον εντοπισμό της σωστής πινακίδας.

5.1 Τί θα μπορούσε να βελτιωθεί;

Στην υλοποίηση επιλέχθηκαν έτοιμα βελτιστοποιημένα μοντέλα τόσο για το detection όσο και για το OCR. Και στα δύο τμήματα του inference pipeline θα μπορούσαν να χρησιμοποιηθούν νεότερα πιο εξελιγμένα μοντέλα- μετά από βελτιστοποίηση. Για παράδειγμα το YOLO26 με τις θεαματικές βελτιώσεις του μετά από μια εκπαίδευση θα μπορούσε να αποδίδει ακόμη καλύτερα. Ήδη με τους «αντάπτορες» που υλοποιήσαμε σε διάφορες φάσεις του pipeline, καλύπτουμε έναν αριθμό από διαφορετικές γενιές YOLO. Επίσης, νέες λειτουργίες μπορούν να προστεθούν στο λογισμικό. Ήδη σε κάποια τμήματα αναφέρθηκαν πιθανές επεκτάσεις – όπως για παράδειγμα το ολοκληρωμένο CRUD στη λίστα των οχημάτων. Αυτή τη στιγμή γίνεται μόνο προσθήκη ενός οχήματος που δεν είναι «καθαρό». Δυνητικά , μπορεί να αλλάξει το status του από “STOLEN” σε “WANTED” και το αντίθετο , χωρίς καμία αλλαγή στον κώδικα. Δεν επιτρέπουμε όμως την αλλαγή από μια «επικίνδυνη» κατάσταση σε «καθαρό». Θα μπορούσε να λυθεί εύκολα, δίνοντας στον χρήστη και την επιλογή του «καθαρού» οχήματος στην διαδικασία της καταχώρησης. Μια όμως- κατά λάθος- τροποποίηση εγγραφής με τέτοιο τρόπο θα ήταν επικίνδυνη για τους χρήστες. Οπότε θα ήταν καλό να προστεθεί μεν η δυνατότητα – υλοποιώντας πάλι τον μηχανισμό του RBAC- μόνο για διαχειριστές ή προϊσταμένους. Μια από τις κύριες αρχές του προγραμματισμού είναι ότι «ότι δεν μπορεί να μετρηθεί- δεν μπορεί να βελτιωθεί». Ο τρόπος που έχουμε «στήσει» την εφαρμογή – παίρνοντας εικόνα από την κάμερα, δεν επιτρέπει να κάνουμε συγκριτικές μετρήσεις ανάμεσα στα 5 μοντέλα που έχουμε για το OCR. Στο Παράρτημα Β , έχουμε τις μετρήσεις τους από “benchmarking”. Προσθέτοντας τη δυνατότητα να επιλέξει ο χρήστης την πηγή εισόδου(ανάμεσα σε κάμερα ή ένα αρχείο βίντεο) θα μπορούσαμε να τρέξουμε πολλά διαφορετικά σενάρια και πολλά διαφορετικά μοντέλα ώστε να βρούμε αυτό που είναι πιο αποδοτικό και αξιόπιστο. Προς το παρόν, στο “detections.log” έχουμε μια εικόνα των αναγνωρίσεων και του confidence κάθε μιας – μαζί με το μοντέλο που χρησιμοποιήθηκε για το OCR.

Μια άλλη λειτουργικότητα που – είναι και εύκολο να προστεθεί καθώς οι μέθοδοι είναι έτοιμοι- είναι η «αποσύνδεση» ενός χρήστη. Θα μπορούσαμε να προσθέσουμε ένα κουμπί για να κάνει απλά το “logout” και να επιστρέφει στο LoginWindow. Ακόμη θα μπορούσε να προστεθεί ένα κουμπί «ξέχασα τον κωδικό μου» στη διαδικασία του login, ή να υλοποιηθεί μια πιο αυστηρή διαδικασία authentication και authorization. Οι δυνατότητες είναι πολλές. Σε ένα ιδανικό σενάριο – θα μπορούσε το σύστημα να τοποθετηθεί σε οχήματα ή ακόμη και σε Συστήματα μη Επανδρωμένων Αεροσκαφών [34] και να «σαρώνει» ασταμάτητα τους δρόμους και αντί να γίνονται αιτήματα προς την τοπική ΒΔ μας να

γίνονται αιτήματα προς έναν κεντρικό εξυπηρετητή της Ελληνικής Αστυνομίας. Εάν προχωρούσε μια υλοποίηση προς αυτήν την κατεύθυνση , οι τρόποι μετάδοσης του αιτήματος είναι πολλοί. Εγώ όμως θα επέλεγα να χρησιμοποιήσω τεχνολογίες τύπου “LoRa”με υψηλή διαπερατότητα στον αστικό ιστό, με τεράστια εμβέλεια και αξιοπιστία. Ένας μηχανισμός «κρυπτογράφησης» της πληροφορίας θα ήταν απαραίτητος πριν ταξιδέψει στον αέρα και αποκρυπτογραφηθεί στην άλλη πλευρά. Με αυτόν τον τρόπο θα ήταν σίγουρος ο εντοπισμός πολλών «κλεμμένων» οχημάτων που μπορεί να περάσουν απαρατήρητα από το ανθρώπινο μάτι.

Το μόνο πράγμα που είναι σίγουρο σε αυτή τη φάση της υλοποίησης- είναι ότι μέσα από αυτή τη διαδικασία αποκτήθηκαν καινούριες δεξιότητες και αυξήθηκαν οι γνώσεις.

Ολόκληρος ο κώδικας της υλοποίησης είναι διαθέσιμος στο github στο παρακάτω repo:

https://github.com/Nyxteridass/APNR_CB/tree/main

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Z. H. Zhou, Ensemble methods: foundations and algorithms., CRC press ISBN-13: 978-1-4398-3005-5, (2012).

[2] L. Rokach, Ensemble-based classifiers, Springer . https://doi.org/10.1007/s10462-009-9124-7, 2009.

[3] "Raspberry Pi Foundation , Raspberry Pi Computer Hardware," [Online]. Available: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html. [Accessed 04 2025].

[4] Π. Γεωργιάδης, Κ. Ζαχαράκης and Κ. Διαμαντάρας, "Εφαρμογή Android για την αναγνώριση οδικών σημάτων," Thessaloniki, 2022.

[5] Κ. Διαμαντάρας and Δ. Μπότσης, "Συνελικτικά νευρωνικά δίκτυα," in *Μηχανική Μάθηση*, Κλειδάριθμος,, 2019, pp. 276-286.

[6] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press , ISBN: 9780262035613, 2016.

[7] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning," *Nature,* vol. 521, no. https://doi.org/10.1038/nature14539, p. 436–444, 2015.

[8] R. C. Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design, Prentice Hall: Pearson 1st Edition, 2017.

[9] R. C. Martin, "The Clean Code Blog," 13 August 2012. [Online]. Available: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html. [Accessed 01 2025].

[10] Visual Studio Code Docs, "Visual Studio Code documentation," [Online]. Available: https://code.visualstudio.com/docs. [Accessed 02 2025].

[11] National Institute of Standards and Technology,, "NIST Special Publication (SP) NIST SP 800-63-4," Gaithersburg.

[12] J. Manico, J. Mackowski, S. Zalman, K. Heigh and W. Wall, "OWASP/CheatSheetSeries," [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html. [Accessed 06 2025].

[13] M. Jones, J. Bradley and N. Sakimura, "JWT," Internet Engineering Task Force (IETF), May 2015. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7519. [Accessed 07 2024].

[14] A. Barth, "Internet Engineering Task Force (IETF)," 2011. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6265. [Accessed 07 2024].

[15] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn and R. Chandramouli, "Proposed NIST standard for role-based access control.," *ACM Transactions on Information and System Security (TISSEC),* vol. 4, no. 3, pp. 224-274, August 2001.

[16] American National Standards Institute, Inc., "Role Based Access Control," *INCITS 359-2004,* 3 February 2004.

[17] E. Coyne, T. Weil and D. Kuhn, "Role Engineering: Methods and Standards," *IT Professional,* vol. 13, pp. 54-57, 11 2011.

[18] M. Fowler, "Inversion of Control Containers and the Dependency Injection pattern," 2004. [Online]. Available: https://martinfowler.com/articles/injection.html.

[19] R. Mogylatov, "ets-labs.org," [Online]. Available: https://python-dependency-injector.ets-labs.org/. [Accessed 09 2025].

[20] G. Jocher, J. Qiu and A. Chaurasia, "Ultralytics YOLO (Version 11.0.0)," 2024. [Online]. Available: https://github.com/ultralytics/ultralytics. [Accessed 20 April 2025].

[21] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016.

[22] G. Jocher, "YOLOv5 by Ultralytics (Version 5.0)," 2020. [Online]. Available: https://github.com/ultralytics/ultralytics. [Accessed 20 April 2025].

[23] Ultralytics, "Ultralytics YOLO Docs," 11 2025. [Online]. Available: https://docs.ultralytics.com/models/yolo26/#overview. [Accessed 12 2025].

[24] G. Jocher, J. Qiu and A. Chaurasia, "Ultralytics YOLO (Version 8.0.0)," 2023. [Online]. Available: https://github.com/ultralytics/ultralytics. [Accessed 20 April 2025].

[25] morsetechlab, "Hugging Face," 04 2025. [Online]. Available: https://huggingface.co/morsetechlab/yolov11-license-plate-detection/tree/main. [Accessed 09 2025].

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, "Attention Is All You Need," Google, 2023.

[27] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit and N. Houlsby, "AN IMAGE IS WORTH 16X16 WORDS:TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE," in *ICLR*, 2021.

[28] A. Hassani, S. Walton, N. Shah, A. Abuduweili, J. Li and H. Shi, "Escaping the Big Data Paradigm with Compact Transformers," *ArXiv,* vol. abs/2104.05704, 2021.

[29] S. Mehta and M. Rastegari, "MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer," in *ICLR*, 2022.

[30] S. Mehta and M. Rastegari, "Separable Self-attention for Mobile Vision Transformers," 2022.

[31] H. Hongyuan, L. Xinyang and L. Guosheng, "Metrics and evaluations for computational and sustainable AI efficiency," 2025.

[32] Community, "ONNX Runtime," onnxruntime.ai, 02 2023. [Online]. Available: https://onnxruntime.ai/docs/get-started/with-python.html. [Accessed 01 2026].

[33] ankandrew.github.io, "fast-plate-ocr," [Online]. Available: https://ankandrew.github.io/fast-plate-ocr/latest/reference/inference/hub. [Accessed 02 2026].

[34] H. Wang, Z. Dang, H. Shi and e. al., "Portable Drone Detection with Optimized YOLOv11 for Edge Devices," Jiaotong Univ. (Sci.), Shanghai, 2025.

ΠΑΡΑΡΤΗΜΑ Α : ΟΔΗΓΟΣ ΥΛΟΠΟΙΗΣΗΣ APNR SYSTEM

Παρακάτω παρατίθενται αναλυτικά τα βήματα υλοποίησης του συστήματος:

Εγκατάσταση λογισμικού Raspberry Pi Imager στον υπολογιστή και τοποθέτηση της κάρτας microSD σε αυτόν για την μορφοποίησή της.

Από το λογισμικό Imager, επιλέγουμε Raspberry Pi OS (64-bit) **με** desktop περιβάλλον και αποθήκευση την κάρτα. Επιλέγουμε την εγγραφή και αφού ολοκληρωθεί η διαδικασία εισάγουμε την κάρτα στην πλατφόρμα μας. Συνδέουμε όλα τα περιφερειακά και την κάμερά μας (Camera module 2) με την αντίστοιχη ταινία και κατόπιν ενεργοποιούμε το σύστημα. Εδώ δεν πρέπει να παραλείψουμε τη σύνδεση του passive buzzer module που χρησιμοποιεί η εφαρμογή με τα αντίστοιχα καλώδια dupont. Στο πρόγραμμα έχουμε ορίσει ως pin για το buzzer μας το pin 17. **Σημαντικο:** η αρίθμηση των ακίδων του Raspberry Pi (φυσικές ακίδες) είναι διαφορετική από την αρίθμηση που καλείται στο πρόγραμμα. Αναλυτικά φαίνονται οι αντίστοιχες ακίδες με τους αριθμούς τους στο Σχήμα 2.1. Με βάση αυτό – συνδέουμε το καλώδιο του σήματος του buzzer μας στην ακίδα 11 , τη γείωση του module στο pin 14 και την παροχή τροφοδοσίας 3.3V από το pin 17. Αφού ενεργοποιηθεί το σύστημα συνδεόμαστε στο διαδίκτυο -είτε μέσω wifi – είτε μέσω καλωδίου UTP. Η πρόσβαση στο διαδίκτυο είναι απαραίτητη μέχρι την πρώτη φορά που θα εκτελεστεί το πρόγραμμα – καθώς θα γίνει λήψη των απαραίτητων βιβλιοθηκών. Από εκεί και μετά δεν χρειάζεται η πρόσβαση στο διαδίκτυο και μπορεί να απενεργοποιηθεί.

Από τερματικό εκτελούμε:

sudo apt update && sudo apt upgrade -y

Με την ολοκλήρωση προχωράμε στην αύξηση του swap καθώς θα το χρειαστούμε για έξτρα χώρο αντί για RAM.

sudo dphys-swapfile swapoff

sudo nano /etc/dphys-swapfile

Αναζητούμε τη γραμμή με την εγγραφή : CONF_SWAPSIZE=100 και αλλάζουμε το μέγεθος σε 1024. Κατόπιν αποθηκεύουμε (με Ctrl + O, επιβεβαιώνουμε με enter) και με Ctrl + X βγαίνουμε από τον nano. Έπειτα ενεργοποιούμε εκ νέου το swap με τις νέες ρυθμίσεις πληκτρολογώντας:

sudo dphys-swapfile setup

sudo dphys-swapfile swapon

Σε αυτή τη φάση το υλικό μας είναι έτοιμο να δουλέψει.

Εισάγουμε:

sudo apt install python3-venv python3-tk libgl1-mesa-glx libqt5x11extras5 -y

Πλοηγούμαστε στο φάκελο που θέλουμε να εγκαταστήσουμε το εικονικό περιβάλλον μας καθώς δεν επιτρέπεται εξ’ ορισμού η εγκατάσταση κάποιων πακέτων στο σύστημά μας (και να επιτρεπόταν δεν είναι ο ενδεδειγμένος τρόπος – είναι επικίνδυνο). Όταν επιλέξουμε και εισέλθουμε στο φάκελο που θα φιλοξενήσει το virtual environment τότε :

python3 -m venv venv Σημείωση: εάν επιθυμούμε να ονομάσουμε διαφορετικά το περιβάλλον μας τότε στη θέση του δεύτερου venv της εντολής – εισάγουμε το όνομα που θέλουμε. Με βάση αυτό το όνομα θα ενεργοποιείται από εδώ και πέρα το περιβάλλον, αλλάζοντας το όρισμα της παρακάτω εντολής:

source venv/bin/activate

Δημιουργούμε ένα αρχείο κειμένου με το όνομα "requirements.txt" στο φάκελό μας και αποθηκεύουμε τα παρακάτω:

opencv-python

Pillow

numpy

onnxruntime

fast-plate-ocr

SQLAlchemy

bcrypt

gpiozero

Έπειτα εισάγουμε στο τερματικό μας :

pip install -r requirements.txt

και περιμένουμε να εγκατασταθούν όλες οι βιβλιοθήκες στο εικονικό περιβάλλον μας.

Με οποιοδήποτε τρόπο , μεταφέρουμε το σύνολο των αρχείων – ολόκληρο το φάκελο με το project μας στον φάκελο που έχουμε εγκαταστήσει και το περιβάλλον και αφού μεταβούμε στον ριζικό φάκελο της εφαρμογής μας και βεβαιωθούμε ότι περιέχονται και τα κενά αρχεία "__init__.py" τότε τρέχουμε :

python3 create_admin.py

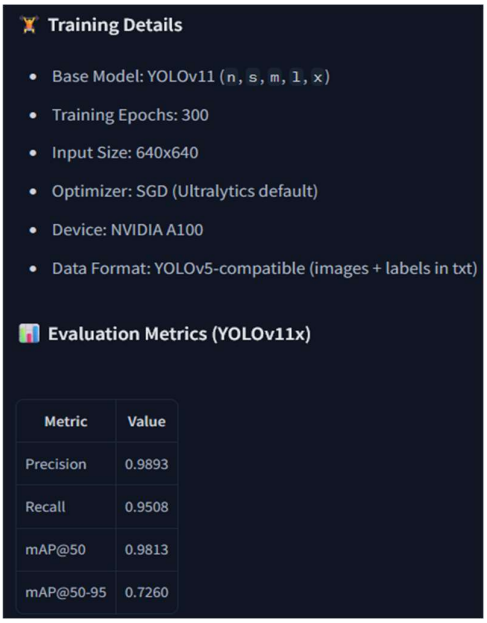
Με την ολοκλήρωση της εκτέλεσης θα πάρουμε μήνυμα επιτυχίας. Κατόπιν δίνουμε την εντολή :

python3 -m main

από τον **ριζικό κατάλογο** της εφαρμογής και **χωρίς** την κατάληξη «.py» καθώς καλούμε modular εκτέλεση του προγράμματος. Το σύστημα θα ενεργοποιηθεί – η είσοδος στην εφαρμογή γίνεται εισάγοντας username: admin και password : iee2019240 στο παράθυρο σύνδεσης. Εάν πριν την εκτέλεση του αρχείου create_admin.py έχει γίνει επεξεργασία στις παραμέτρους αυτές – τότε η σύνδεση πραγματοποιείται με βάση τις νέες παραμέτρους που έχουν επιλεγεί.

ΠΑΡΑΡΤΗΜΑ Β: ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΜΟΝΤΕΛΩΝ

B1 DETECTION : yolov11-license-plate-detection.onnx

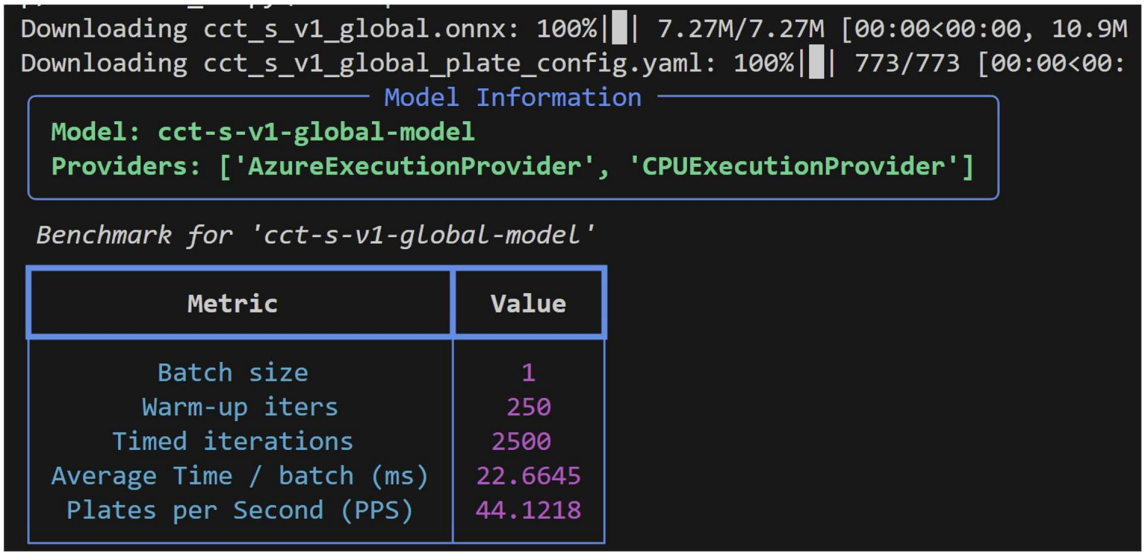


[https://huggingface.co/morsetechlab/yolov11-license-plate-detection]

[https://huggingface.co/morsetechlab/yolov11-license-plate-detection]

B2 OCR

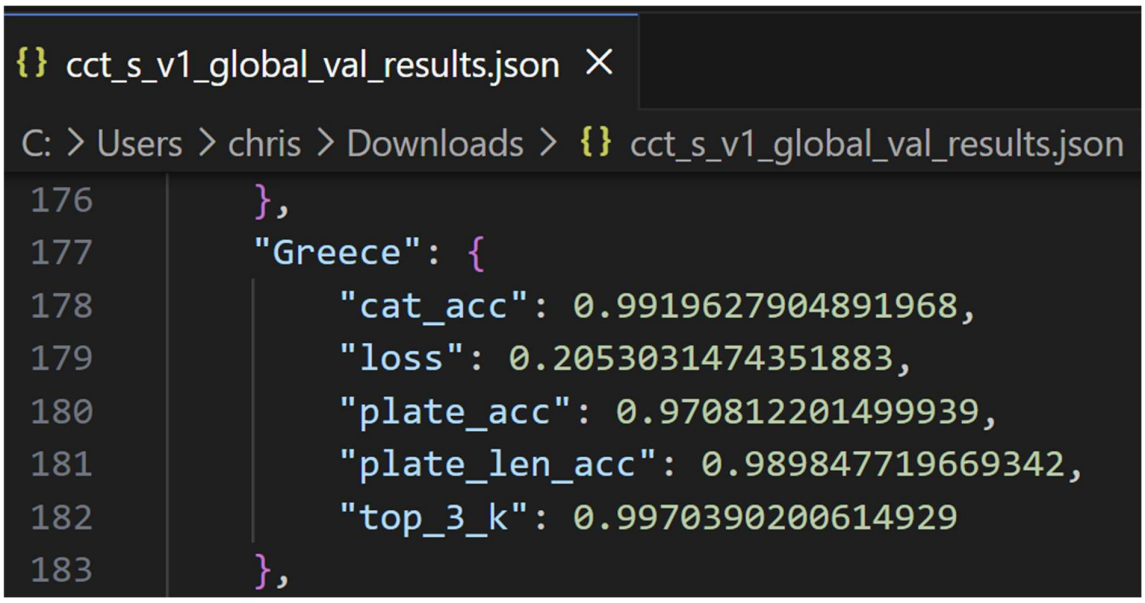
B2.1 cct-s-v1-global-model



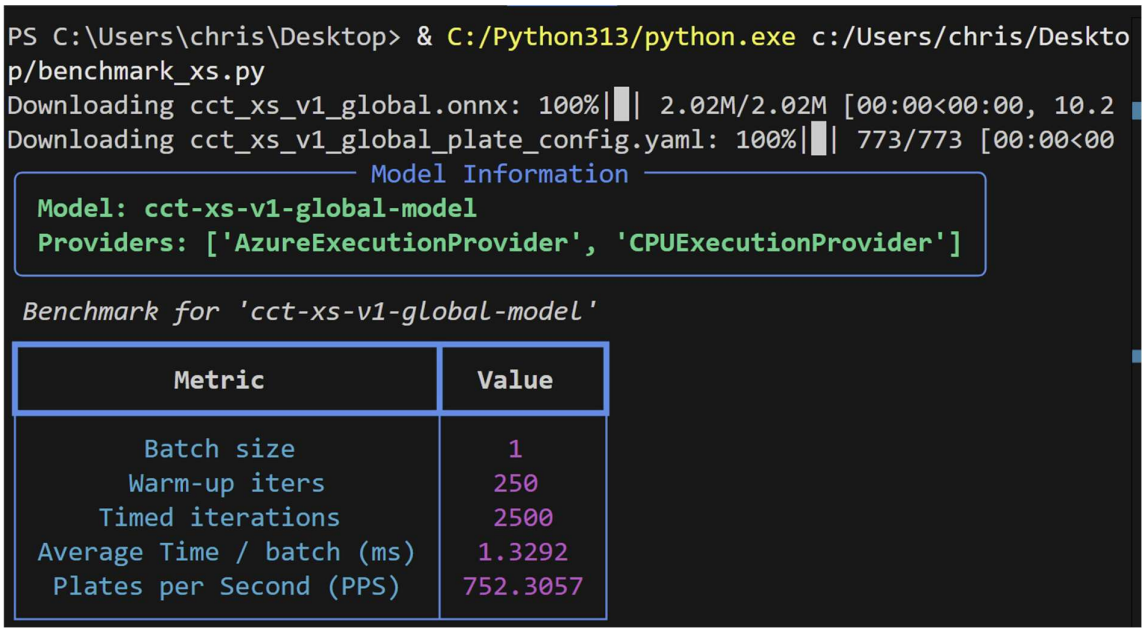
```
model: cct

rescaling:
  scale: 0.00392156862745098
  offset: 0.0

tokenizer:
  blocks:
    - { layer: Conv2D, filters: 48, kernel_size: 3, strides: 1, padding: same, activation: gelu, use_bias: false }
    - { layer: MaxBlurPooling2D, pool_size: 2, filter_size: 3 }
    - { layer: Conv2D, filters: 80, kernel_size: 3, strides: 1, padding: same, activation: gelu, use_bias: false }
    - { layer: Conv2D, filters: 96, kernel_size: 3, strides: 1, padding: same, activation: gelu, use_bias: false }
    - { layer: Conv2D, filters: 128, kernel_size: 3, strides: 1, padding: same, activation: gelu, use_bias: false }

  positional_emb: true
  patch_size: 2
  patch_mlp:
    layer: MLP
    hidden_units: [ 128 ]
    activation: gelu
    dropout_rate: 0.1

transformer_encoder:
  layers: 6
  heads: 2
  projection_dim: 128
  units: [ 128, 128 ]
  activation: gelu
  stochastic_depth: 0.12
  attention_dropout: 0.1
  mlp_dropout: 0.1
  head_mlp_dropout: 0.15
```

token_reducer_heads: 4

normalization: dyt



B2.2 cct-xs-v1-global-model

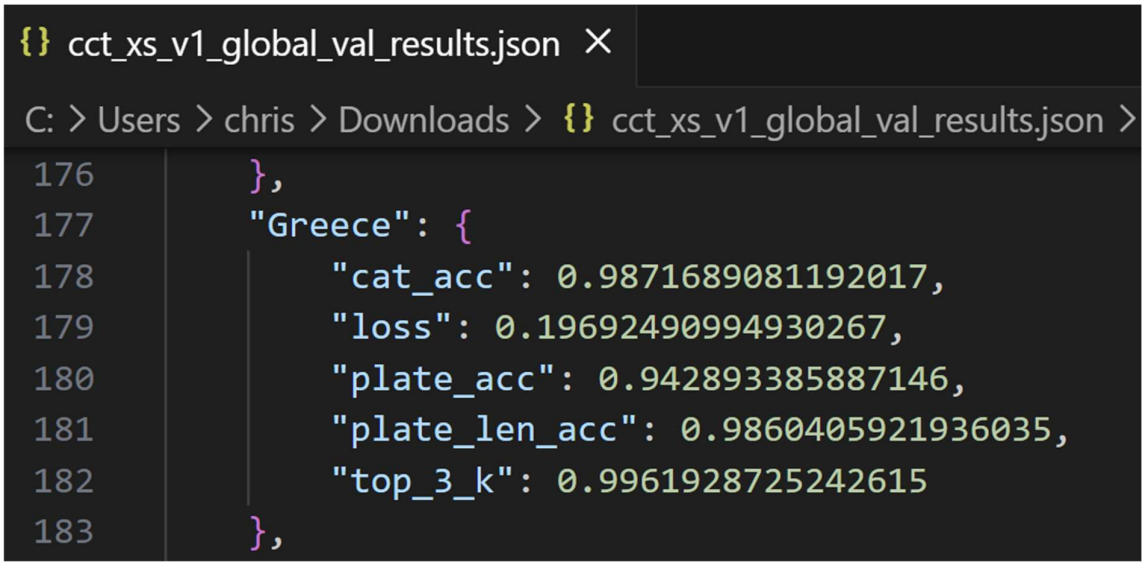


model: cct

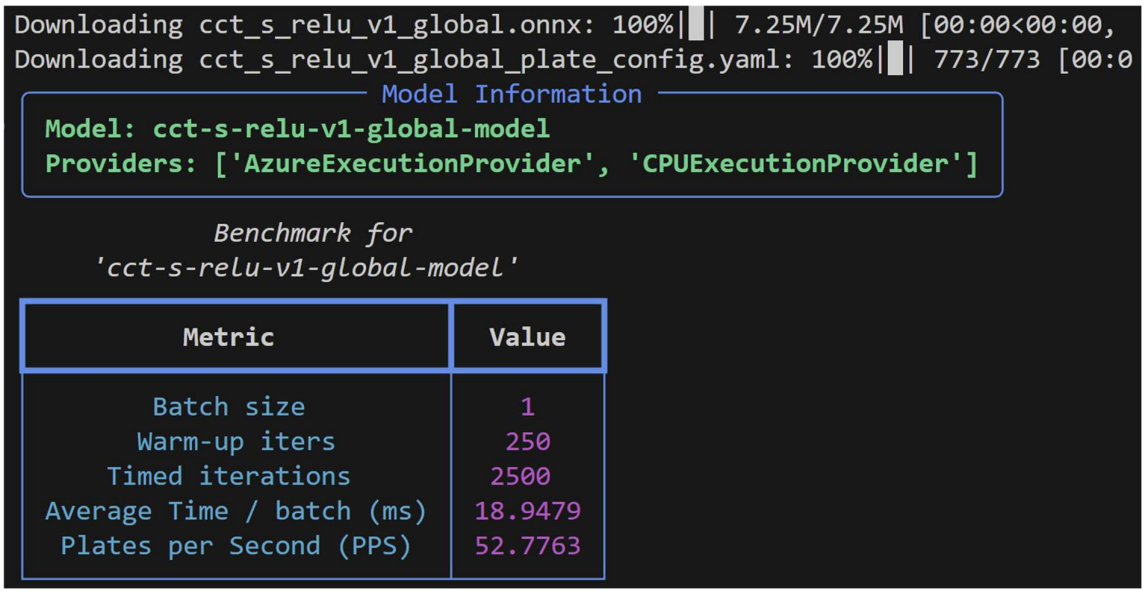
rescaling:

scale: 0.00392156862745098

```
  offset: 0.0

tokenizer:
  blocks:
    - { layer: Conv2D, filters: 32, kernel_size: 3, strides: 1, padding: valid, activation: relu, use_bias: true }
    - { layer: MaxBlurPooling2D, pool_size: 2, filter_size: 3 }
    - { layer: Conv2D, filters: 48, kernel_size: 3, strides: 1, padding: valid, activation: relu, use_bias: true }
    - { layer: Conv2D, filters: 64, kernel_size: 3, strides: 1, padding: valid, activation: relu, use_bias: true }
    - { layer: MaxBlurPooling2D, pool_size: 2, filter_size: 3 }
    - { layer: Conv2D, filters: 80, kernel_size: 3, strides: 1, padding: valid, activation: relu, use_bias: true }
    - { layer: Conv2D, filters: 96, kernel_size: 3, strides: 1, padding: valid, activation: relu, use_bias: true }

  positional_emb: true
  patch_size: 2
  patch_mlp:
    layer: MLP
    hidden_units: [64]
    activation: gelu
    dropout_rate: 0.05

transformer_encoder:
  layers: 4
  heads: 1
  projection_dim: 64
  units: [64, 64]
  activation: gelu
  stochastic_depth: 0.05
  attention_dropout: 0.05
  mlp_dropout: 0.1
  head_mlp_dropout: 0.05
  token_reducer_heads: 4
  normalization: dyt
```



B2.3 cct-s-relu-v1-global-model



model: cct

rescaling:

scale: 0.00392156862745098

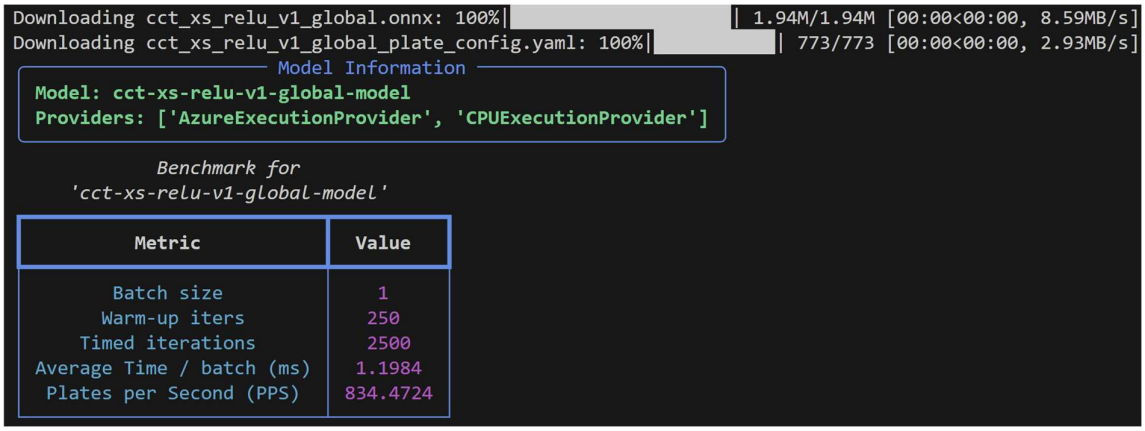
offset: 0.0

tokenizer:

blocks:

```
    - { layer: Conv2D, filters: 48, kernel_size: 3, strides: 1, padding: same, activation: relu, use_bias: false }
    - { layer: MaxBlurPooling2D, pool_size: 2, filter_size: 3 }
    - { layer: Conv2D, filters: 80, kernel_size: 3, strides: 1, padding: same, activation: relu, use_bias: false }
    - { layer: Conv2D, filters: 96, kernel_size: 3, strides: 1, padding: same, activation: relu, use_bias: false }
    - { layer: Conv2D, filters: 128, kernel_size: 3, strides: 1, padding: same, activation: relu, use_bias: false }

  positional_emb: true
  patch_size: 2
  patch_mlp:
    layer: MLP
    hidden_units: [ 128 ]
    activation: relu
    dropout_rate: 0.1

transformer_encoder:
  layers: 6
  heads: 2
  projection_dim: 128
  units: [ 128, 128 ]
  activation: relu
  stochastic_depth: 0.12
  attention_dropout: 0.1
  mlp_dropout: 0.1
  head_mlp_dropout: 0.15
  token_reducer_heads: 4
  normalization: dyt
```

B2.4 cct-xs-relu-v1-global-model



model: cct

rescaling:

scale: 0.00392156862745098

offset: 0.0

tokenizer:

blocks:

- { layer: Conv2D, filters: 32, kernel_size: 3, strides: 1, padding: valid, activation: relu, use_bias: true }
- { layer: MaxBlurPooling2D, pool_size: 2, filter_size: 3 }
- { layer: Conv2D, filters: 48, kernel_size: 3, strides: 1, padding: valid, activation: relu, use_bias: true }
- { layer: Conv2D, filters: 64, kernel_size: 3, strides: 1, padding: valid, activation: relu, use_bias: true }
- { layer: MaxBlurPooling2D, pool_size: 2, filter_size: 3 }
- { layer: Conv2D, filters: 80, kernel_size: 3, strides: 1, padding: valid, activation: relu, use_bias: true }
- { layer: Conv2D, filters: 96, kernel_size: 3, strides: 1, padding: valid, activation: relu, use_bias: true }

positional_emb: true

patch_size: 2

patch_mlp:

layer: MLP

hidden_units: [64]

activation: relu

dropout_rate: 0.05

transformer_encoder:

layers: 4

heads: 1

projection_dim: 64

units: [64, 64]

activation: relu

stochastic_depth: 0.05

attention_dropout: 0.05

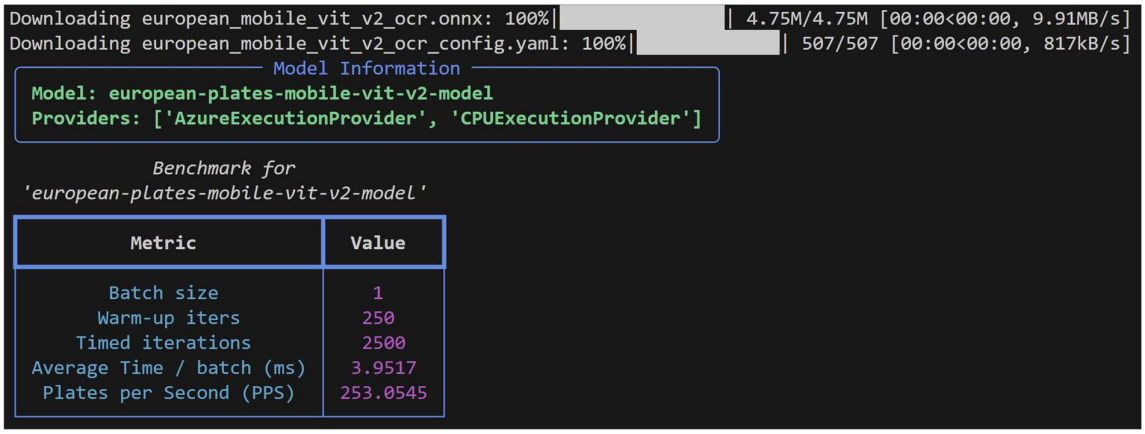
mlp_dropout: 0.1

head_mlp_dropout: 0.05

token_reducer_heads: 4

normalization: dyt

B2.5 european-plates-mobile-vit-v2-model



ΠΑΡΑΡΤΗΜΑ Γ: ΚΩΔΙΚΑΣ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

```
#config1.py
import cv2
# ================================
# CONFIGURATION FILE για το RPi 3B+
# ================================
CAMERA_ID = 0

# Capture video size - ιδιαίτερα σημαντικό για την απόδοση σε Pi 3B+
FRAME_WIDTH = 640 #320
FRAME_HEIGHT = 480 #240

# Display refresh (UI) and detection cadence (CPU) - ιδιαίτερα σημαντικό για την απόδοση σε Pi 3B+
GUI_FPS = 15
DETECT_EVERY_N_FRAMES = 3
OCR_INTERVAL_SEC = 0.45
MAX_DETECTIONS_DRAW = 3

# Greek plate whitelist / χαρακτήρες που επιτρέπονται στις Ελληνικές πινακίδες
GREEK_CHARS = "ABEZHIKMNOPTYX"
DIGITS = "0123456789"

CONF_THRESHOLD = 0.45
NMS_THRESHOLD = 0.50
MODEL_INP = 320

FAST_OCR_MODELS = [
    "cct-xs-relu-v1-global-model",
    "cct-xs-v1-global-model",
    "cct-s-relu-v1-global-model",
    "cct-s-v1-global-model",
    "european-plates-mobile-vit-v2-model",
```

```
]
try:
    cv2.setNumThreads(2)
except:
    pass

#container.py

from app.inference.pipeline import InferencePipeline
from app.services.detection_services import DetectionService
from app.core import config1

class Container:
    def __init__(self):
        # 1. Αρχικοποίηση του AI Pipeline
        model_name = config1.FAST_OCR_MODELS[0]
        self.pipeline = InferencePipeline(model_name)

        # 2. Αρχικοποίηση του Service της βάσης
        self.detection_service = DetectionService()

#session.py

class SessionManager:
    _user = None
    _role = None

    @classmethod
    def set_user(cls, username, role):
        cls._user = username
        cls._role = role

    @classmethod
```

```
    def get_current_user(cls):
        return cls._user

    @classmethod
    def get_role(cls):
        return cls._role

    @classmethod
    def clear(cls):
        cls._user = None
        cls._role = None
```

```
#connection.py

from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from app.utils.pathfinder import Labyrinth
from .dbmodels import Base

class DBConnectionManager:
    # Παίρνουμε το path της SQLite από το Pathfinder.py
    DB_URL = f"sqlite:///{Labyrinth.get_db_path()}"

    engine = create_engine(DB_URL, connect_args={"check_same_thread": False})
    SessionLocal = sessionmaker(bind=engine, expire_on_commit=False, autoflush=False)

    @staticmethod
    def setup_db():
        # Δημιουργούμε τους πίνακες στη βάση μας εάν δεν υπάρχουν.
        Base.metadata.create_all(DBConnectionManager.engine)
```

```
    @staticmethod
    def get_session():
        # Επιστρέφω ένα νέο session για να μπορέσουμε να κάνουμε queries στο police_db.sqlite
        return DBConnectionManager.SessionLocal()

#database.py



from app.database.dbmodels import StolenVehicle , VehicleStatus
from app.database.connection import DBConnectionManager
import os
from app.services.logger import log_error

class DatabaseManager:
    def __init__(self):
        # Κλήση της setup_db => Αρχικοποίηση της βάσης εάν δεν υπάρχει ήδη.
        DBConnectionManager.setup_db()

    def check_plate(self, text):
        # Ελέγχουμε πρώτα αν το κείμενο της πινακίδας που αναγνωρίστηκε είναι έγκυρο.
        clean_text = ''.join(e for e in text if e.isalnum()).upper()
        if not clean_text:
            return None, "CLEAN", "#ecf0f1" # Default γκρι

        session = DBConnectionManager.get_session()
        # Ψάχνουμε στη βάση εάν το όχημα είναι καταχωρημένο ως STOLEN, ή WANTED.
        #  Εάν όχι, θεωρούμε ότι είναι CLEAN.
        vehicle = session.query(StolenVehicle).filter_by(plate_number=clean_text).first()
        session.close()

        if vehicle:
            # Δίνουμε χρώμα ανάλογα με το status του οχήματος στη βάση για να περάσει
            # ως alert στο UI παρακάτω
```

```
        colors = {
            VehicleStatus.STOLEN: "#c0392b", # Κόκκινο
            VehicleStatus.WANTED: "#d35400", # Πορτοκαλί
            VehicleStatus.CLEAN: "#27ae60"   # Πράσινο
        }
        # Το vehicle.status είναι  Enum object, οπότε το χρησιμοποιούμε ως key
        color = colors.get(vehicle.status, "#ecf0f1")
        return clean_text, vehicle.status.value, color

    return clean_text, "CLEAN", "#27ae60"

def add_stolen_vehicle(self, plate, status_string):
    # Προσθήκη οχήματος που είναι STOLEN/WANTED στη βάση μας.
    session = DBConnectionManager.get_session()
    try:
        # Διασφάλιση εγκυρότητας του status του οχήματος.
        #  Στο combobox του UI ΔΕΝ έχουμε άλλη επιλογή,
        # βεβαιωνόμαστε ότι σε περίπτωση σφάλματος, δε θα
        # "χτυπήσει" η βάση ή δε θα κρασάρει η εφαρμογή.

        status_enum = VehicleStatus(status_string.upper())

        vehicle = StolenVehicle(
            plate_number=plate.upper(),
            status=status_enum
        )

        session.merge(vehicle)
        session.commit()
        return True
    except Exception as e:
        session.rollback()
        file_name = os.path.basename(__file__)
```

```
            error_type = type(e).__name__
            log_error(error_type, file_name, str(e))
            raise e


        finally:
            session.close()



#dbmodels.py

import enum
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column
from sqlalchemy import String
from sqlalchemy import Enum as SQLEnum

# Κλάσεις με τα entities που θα δημιουργηθούν στη ΒΔ μας.
#  Η SQLAlchemy αναλαμβάνει τη δημιουργία των πινάκων και
# των πεδίων τους, καθώς και τη διαχείριση των queries.

class Base(DeclarativeBase):
    pass

class UserRole(str , enum.Enum):
    USER = "USER"
    SUPERVISOR = "SUPERVISOR"
    ADMIN = "ADMINISTRATOR"

class VehicleStatus(str ,enum.Enum):
    STOLEN = "STOLEN"
    WANTED = "WANTED"
    CLEAN = "CLEAN"
```

```
class User(Base):
    __tablename__ = "users"
    id: Mapped[int] = mapped_column(primary_key=True)
    username: Mapped[str] = mapped_column(String(50), unique=True)
    password: Mapped[str] = mapped_column(String(255), nullable=False) # Hashed
    role: Mapped[UserRole] = mapped_column(SQLEnum(UserRole), default= UserRole.USER)


class StolenVehicle(Base):
    __tablename__ = "stolen_vehicles"
    plate_number: Mapped[str] = mapped_column(primary_key=True)
    # Χρησιμοποιούμε SQLEnum για τη βάση και το VehicleStatus για την Python
            status: Mapped[VehicleStatus] = mapped_column(SQLEnum(VehicleStatus), default=VehicleStatus.CLEAN)



#pipeline.py

import cv2
import numpy as np
import os
import app.core.config1 as config
from app.utils.helpers import letterbox, clamp
from app.utils.pathfinder import Labyrinth
from app.services.logger import log_error
from fast_plate_ocr import LicensePlateRecognizer

class InferencePipeline:
    def __init__(self, ocr_model_name):
        #Αρχικοποίηση του μοντέλου ανίχνευσης πινακίδων
        try:
            model_path = Labyrinth.get_model_detector()
            self.net = cv2.dnn.readNetFromONNX(model_path)
            self.net.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)
```

```
            self.net.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)
        except Exception as e:
            file_name = os.path.basename(__file__)
            error_type = type(e).__name__

            log_error(error_type, file_name, str(e))
            self.net = None

        try:
            self.recognizer = LicensePlateRecognizer(ocr_model_name)
        except Exception as e:
            file_name = os.path.basename(__file__)
            error_type = type(e).__name__

            log_error(error_type, file_name, str(e))
            self.recognizer = None

    def load_ocr(self, model_name):
        try:
            self.recognizer = LicensePlateRecognizer(model_name)
        except Exception as e:
            file_name = os.path.basename(__file__)
            error_type = type(e).__name__

            log_error(error_type, file_name, str(e))
            self.recognizer = None

    def detect_and_ocr(self, frame_bgr):
        if self.net is None or frame_bgr is None:
            return []

        # Letterbox & Inference
        img_lb, r, (pad_x, pad_y) = letterbox(frame_bgr, (config.MODEL_INP, config.MODEL_INP))
```

```
        blob = cv2.dnn.blobFromImage(img_lb, 1 / 255.0, (config.MODEL_INP, config.MODEL_INP), swapRB=True, crop=False)
        self.net.setInput(blob)
        outputs = self.net.forward()

        out = np.squeeze(outputs)
#region Comments
        # Ασφαλής Διαχείριση Output και προετοιμασία για NMS.
        # Εδώ διασφαλίζουμε ότι το output έχει τη σωστή μορφή,
        # ανεξάρτητα από το αν το μοντέλο επέστρεψε 1D ή 2D array,
        #  ή αν τα δεδομένα είναι σε xywh format ή normalized.
#endregion
        if out.ndim == 2:
            if out.shape[0] < out.shape[1] and out.shape[0] in (5, 6, 7, 8, 85, 84):
                out = out.T
        elif out.ndim == 1:
            return []
        else:
            out = out.reshape(-1, out.shape[-1])

        if out.shape[1] < 5:
            return []

        boxes = []
        confs = []

        sample = out[:min(50, out.shape[0]), :4]
        max_xywh = float(np.max(sample)) if sample.size else 0.0
        normalized_xywh = max_xywh <= 2.0

        for i in range(out.shape[0]):
            x, y, w, h = out[i, 0], out[i, 1], out[i, 2], out[i, 3]
```

```
    if out.shape[1] == 5:
        conf = float(out[i, 4])
    else:
        obj = float(out[i, 4])
        cls_probs = out[i, 5:]
        conf = obj * float(np.max(cls_probs)) if cls_probs.size > 0 else obj

    if conf < config.CONF_THRESHOLD:
        continue

    if normalized_xywh:
        x *= config.MODEL_INP
        y *= config.MODEL_INP
        w *= config.MODEL_INP
        h *= config.MODEL_INP

    left = (x - 0.5 * w - pad_x) / r
    top = (y - 0.5 * h - pad_y) / r
    right = (x + 0.5 * w - pad_x) / r
    bottom = (y + 0.5 * h - pad_y) / r

    x0 = int(clamp(left, 0, config.FRAME_WIDTH - 1))
    y0 = int(clamp(top, 0, config.FRAME_HEIGHT - 1))
    x1 = int(clamp(right, 0, config.FRAME_WIDTH - 1))
    y1 = int(clamp(bottom, 0, config.FRAME_HEIGHT - 1))

    bw = max(1, x1 - x0)
    bh = max(1, y1 - y0)

    boxes.append([x0, y0, bw, bh])
    confs.append(conf)

if not boxes:
```

```
            return []


        # NMS
                    indexes = cv2.dnn.NMSBoxes(boxes, confs, config.CONF_THRESHOLD, config.NMS_THRESHOLD)
        if len(indexes) == 0:
            return []


        results = []
        for item in indexes:
            k = int(item[0]) if isinstance(item, (list, tuple, np.ndarray)) else int(item)
            bx = boxes[k]
            conf = confs[k]
#region comments
            # OCR Logic - Προσπαθούμε να αναγνωρίσουμε την πινακίδα μέσα στο ανιχνευμένο
            # πλαίσιο. Εάν το απευθείας ROI δεν δουλέψει (π.χ. λόγω μεγέθους ή μορφής),
            # αποθηκεύουμε προσωρινά την εικόνα και προσπαθούμε ξανά,
            #  για να αυξήσουμε τις πιθανότητες επιτυχίας.
#endregion
            roi = frame_bgr[bx[1]:bx[1] + bx[3], bx[0]:bx[0] + bx[2]]
            raw_text = ""


            if roi is not None and roi.size != 0 and self.recognizer is not None:
                try:
                    raw_text = self.recognizer.run(roi)
                except Exception:
                    try:
                        import tempfile
                        fd, tmp_path = tempfile.mkstemp(suffix=".jpg")
                        os.close(fd)
                        cv2.imwrite(tmp_path, roi)
                        raw_text = self.recognizer.run(tmp_path)
                        os.remove(tmp_path)
```

```
                except:
                    raw_text = ""

            if isinstance(raw_text, list):
                raw_text = raw_text[0] if len(raw_text) > 0 else ""
            raw_text = (raw_text or "")
            if not isinstance(raw_text, str):
                raw_text = str(raw_text)
            raw_text = raw_text.strip()

            results.append({
                'box': bx,
                'text': raw_text.upper(),
                'confidence': float(conf)
            })

        return results
```

#types.py

```
from dataclasses import dataclass
from typing import  Tuple

#Dataclass για να περνάμε τα αποτελέσματα της ανίχνευσης καθαρά.
@dataclass
class DetectionResult:
    box: Tuple[int, int, int, int]  # [x, y, w, h]
    confidence: float
    plate_text: str = ""
    status: str = "CLEAN"
```

#users_repo.py

```
from app.database.connection import DBConnectionManager
from app.database.dbmodels import User
from app.services.logger import log_access, log_error

# Repository για την ανάκτηση των στοιχείων των χρηστών από τη βάση μας.
# Χρησιμοποιείται για τον έλεγχο ταυτότητας κατά το login.

class UsersRepository:
    @staticmethod
    def get_user_by_username(username):
        session = DBConnectionManager.get_session()
        try:
            user = session.query(User).filter_by(username=username).first()
            if user:
                actual_role = user.role.value
                log_access(username, actual_role, "Successful Login Attempt")
                return {
                    "username": user.username,
                    "password": user.password,
                    "role": actual_role
                }
            exception = f"User '{username}' not found in database."
            log_access(username, "UNKNOWN", f"Failed Login Attempt - {exception}")
            log_error("UserNotFound", "UsersRepository", exception)
            return None
        finally:
            session.close()

#detection_services.py

import re
from app.database.connection import DBConnectionManager
from app.database.dbmodels import StolenVehicle
```

```
#region Comments
# Υπηρεσία που ελέγχει την πινακίδα ως προς την ορθότητα για τον περιορισμό των αχρείαστων queries
# προς τη βάση και μετά φέρνει την "κατάσταση" του οχήματος πάλι προς το UI. Εάν ΔΕΝ εντοπιστεί τέτοια
# πινακίδα, επιστρέφει status "CLEAN" και default χρώμα πράσινο. Εάν εντοπιστεί, επιστρέφει το status
# του οχήματος στη βάση μας (STOLEN/WANTED) και το αντίστοιχο χρώμα για να περάσει ως alert στο UI.
#endregion
class DetectionService:
    def __init__(self):
        self.session_factory = DBConnectionManager.get_session

    def process_detected_plate(self, raw_text):
        # "Καθαρισμός" του κειμένου: Αφαιρούμε όλα τα μη-αλφαριθμητικά και μετατρέπουμε σε κεφαλαία
        clean_text = "".join(re.findall(r'[A-Z0-9]', raw_text.upper()))
#region  process_detected_plate() ---Comments
        # Regex για ελληνικό format (3 γράμματα που συνοδεύονται από 1- 4 αριθμούς ,
        # για να συμπεριλάβουμε και  πινακίδες δικύκλων που έχουν 3 γράμματα αλλά
        # από 1 έως 3 αριθμούς). Με αλλαγή στο regex μπορούμε να συμπεριλάβουμε και
        # πινακίδες με 2 γράμματα (π.χ. για παλιά οχήματα) ή να επεκτείνουμε την αναζήτηση
        # σε πινακίδες άλλων τύπων οχημάτων (πχ φορτηγά).
#endregion
        pattern = r'^[ABEZHIKMNOPTYX]{3}\d{1,4}$'
        if not re.match(pattern, clean_text):
            return None # Δεν είναι έγκυρη ελληνική πινακίδα

        # Έλεγχος στη βάση
        session = self.session_factory()
        try:
            vehicle = session.query(StolenVehicle).filter_by(plate_number=clean_text).first()
            if vehicle:
```

```
            return {
                "plate": clean_text,
                "status": vehicle.status.value,

            }

        return {"plate": clean_text, "status": "CLEAN"}
    finally:
        session.close()
```

```
#logger.py

import logging
from pathlib import Path
from app.utils.pathfinder import BASE_DIR

#Εισάγουμε το BASE_DIR και πριν την δημιουργία του logger ελέγχουμε την ύπαρξη
# του φακέλου με τα logs.Στην περίπτωση που δεν υπάρχει - δημιουργείται εδώ.

LOGS_DIR = Path(BASE_DIR) / "logs"
LOGS_DIR.mkdir(exist_ok=True)

def _setup_logger(name: str, log_file: str) -> logging.Logger:
    #Δημιουργούμε ένα logger setup (generic).
    logger = logging.getLogger(name)
    logger.propagate = False
    if not logger.handlers:
        logger.setLevel(logging.INFO)

        # Ο formatter της βιβλιοθήκης βάζει το τρέχον timestamp όταν καλείται ο logger
        #  και μετά το μήνυμα που δέχεται στο αντίστοιχο αρχείο
        formatter = logging.Formatter('%(asctime)s, %(message)s', datefmt='%Y-%m-%d %H:%M:%S')
```

```
        file_handler = logging.FileHandler(LOGS_DIR / log_file, encoding='utf-8')
        file_handler.setFormatter(formatter)
        logger.addHandler(file_handler)

    return logger

# Αρχικοποίηση των 3 loggers όταν φορτώνεται το αρχείο
_det_logger = _setup_logger("detections", "detections.log")
_acc_logger = _setup_logger("access", "access.log")
_err_logger = _setup_logger("errors", "errors.log")

# Συναρτήσεις για τους διάφορους τύπους καταγραφών:

def log_detection(plate: str, status: str, confidence: float, ocr_model: str):
    #region Comments
    # Γράφει στο αρχείο detections.log : timestamp, πινακίδα που ανιχνεύθηκε, status ,
    #  confidence του μοντέλου που χρησιμοποιήθηκε και ποιο μοντέλο OCR έχει χρησιμοποιηθεί
    # στη συγκεκριμένη ανίχνευση για λόγους στατιστικής ανάλυσης.
    #endregion
    # Μορφοποίηση του confidence σε 2 δεκαδικά (π.χ. 0.95)
    _det_logger.info(f"{plate}, {status}, {confidence:.2f}, {ocr_model}")

def log_added_vehicle(plate: str, status: str):
    # Γράφει στο αρχείο access.log , την ΠΡΟΣΘΗΚΗ οχήματος στη βάση.
    _acc_logger.info(f"{plate}, {status}, ADDED_TO_DB ")

def log_access(username: str, role: str, action: str):
     # Γράφει στο access.log, ΚΑΘΕ ενέργεια που σχετίζεται με Χρήστες / απόπειρες εισόδου στο σύστημα
    _acc_logger.info(f"{username}, {role}, {action}")

def log_error(error_type: str, file_name: str, details: str = ""):
    #region
```

```
    # Γράφει στο errors.log, κάθε σφάλμα που προκύπτει κατά τη λειτουργία του συστήματος.
    # Εκτός από το timestamp  δίνει τον τύπο σφάλματος και το αρχείο που το προκάλεσε και μπορεί
    # να συμπεριλάβει και λεπτομέρειες.
    #endregion
    msg = f"{error_type}, {file_name}"
    if details:
        msg += f", {details}"
    _err_logger.error(msg)

#services.py

from app.repos.users_repo import UsersRepository
from app.utils.hasher import verify_password
from app.core.session import SessionManager

# Υπηρεσία που χειρίζεται το authentication των χρηστών.

class AuthService:
    @staticmethod
    def login(username, password):
        user = UsersRepository.get_user_by_username(username)

        # Εδώ το user['password_hash'] περιέχει το user.password από τη βάση
        if user and verify_password(user['password'], password):
            SessionManager.set_user(user['username'], user['role'])
            return True
        return False

    @staticmethod
    def logout():
        SessionManager.clear()

#user_services.py
```

```
from app.database.connection import DBConnectionManager
from app.database.dbmodels import User, UserRole
from app.services.logger import log_access, log_error
from app.utils.hasher import hash_password
from app.core.session import SessionManager
#region
# Υπηρεσία διαχείρισης χρηστών , με υλοποίηση Role-Based Access Control (RBAC). Χωρίζοντας και αναθέτοντας
#δυνατότητες ανάλογα με το ρόλο που έχει ο εκάστοτε χρήστης, μπορούμε να διασφαλίσουμε ότι MONO αυτοί που
#έχουν κατάλληλα δικαιώματα μπορούν να πραγματοποιήσουν κάποιες ενέργειες - πχ CRUD χρηστών.
#endregion
class UserManagementService:
    @staticmethod
    def require_admin():
        # 1. Αυστηρός έλεγχος δικαιωμάτων (Authorization Check)
        if SessionManager.get_role() != "ADMINISTRATOR":
            return False, "ACCESS DENIED : Μόνο ο Διαχειριστής μπορεί να εκτελέσει αυτή την ενέργεια."
        return True, ""

    @staticmethod
    def get_manageable_users():
        session = DBConnectionManager.get_session()
        try:
            #Επιστρέφουμε όλους τους χρήστες που δεν είναι ADMINISTRATORS.
            users = session.query(User).filter(User.role != UserRole.ADMIN).all()
            return [{"id": u.id, "username": u.username, "role": u.role.value} for u in users]
        finally:
            session.close()

    @staticmethod
    def create_user(new_username: str, new_password: str, role_to_assign: str):
```

```
# Δημιουργία νέου χρήστη με επιλογή Ρόλου από ADMINISTRATOR. Ελέγχουμε αν ο τρέχων χρήστης είναι Admin
# εάν όχι, τότε αρνούμαστε την εκτέλεση ενέργειας και καταγράφουμε στο log.
is_admin, message = UserManagementService.require_admin()
if not is_admin:
                log_error("ACCESS DENIED", "UserManagementService", f"Ο χρήστης {SessionManager.get_current_user()} προσπάθησε να φτιάξει λογαριασμό χωρίς άδεια.")
    return False, message

session = DBConnectionManager.get_session()
try:
    existing_user = session.query(User).filter_by(username=new_username).first()
    if existing_user:
        return False, f"Το όνομα χρήστη '{new_username}' χρησιμοποιείται ήδη."

    role_enum = UserRole(role_to_assign.upper())
    hashed_pw = hash_password(new_password)

    new_user = User(
        username=new_username,
        password=hashed_pw,
        role=role_enum
    )
    session.add(new_user)
    session.commit()
    # Καταγραφή της επιτυχούς ενέργειας
        log_access(SessionManager.get_current_user(), SessionManager.get_role(), f"Επιτυχής δημιουργία λογαριασμού ({role_enum.value}): {new_username}")
    return True, f"Ο λογαριασμός για τον χρήστη '{new_username}' με ρόλο '{role_enum.value}' δημιουργήθηκε με επιτυχία."

except Exception as e:
    session.rollback()
    # Καταγραφή της αποτυχίας - λεπτομέρειες σφάλματος
```

```
            log_error(type(e).__name__, "UserManagementService", str(e))
            return False, f"Σφάλμα κατά την αποθήκευση στη βάση: Λάθος Ρόλος ή Δεδομένα. ({str(e)})"
        finally:
            session.close()

    @staticmethod
    def delete_user(user_id: str):
        # Διαγραφή χρήστη με βάση το user_id του.
        is_admin, message = UserManagementService.require_admin()
        if not is_admin:
                        log_error("ACCESS DENIED", "UserManagementService", f"Ο χρήστης {SessionManager.get_current_user()} προσπάθησε να διαγράψει χρήστη χωρίς άδεια.")
            return False, message

        session = DBConnectionManager.get_session()
        try:
            user_to_delete = session.query(User).filter_by(id=user_id).first()
            if  user_to_delete:
                session.delete(user_to_delete)
                session.commit()
                    log_access(SessionManager.get_current_user(), SessionManager.get_role(), f"USER DELETED, ο χρήστης με ID: {user_id} διαγράφηκε!")
                return True, f"Ο χρήστης με ID: {user_id} διαγράφηκε επιτυχώς."
            log_error("FailedDelete", "UserManagementService", f"FAILED DELETE ATTEMPT , ο χρήστης με ID: {user_id} δεν βρέθηκε!")
            return False, f"Ο χρήστης με ID: {user_id} δεν βρέθηκε."
        finally:
            session.close()

    @staticmethod
    def update_user(user_id: str, new_password: str, new_role: str):
        # Ενημέρωση στοιχείων χρήστη - Αλλαγή κωδικού και / ή ρόλου. Απαιτείται ομοίως ρόλος Admin για εκτέλεση.
        is_admin, message = UserManagementService.require_admin()
```

```
    if not is_admin:
                    log_error("ACCESS DENIED", "UserManagementService", f"Ο χρήστης {SessionManager.get_current_user()} προσπάθησε να ενημερώσει χρήστη χωρίς άδεια.")
        return False, message


    session = DBConnectionManager.get_session()
    try:
        user_to_update = session.query(User).filter_by(id=user_id).first()
        if user_to_update:
            if new_password: # Αν δόθηκε νέος κωδικός
                user_to_update.password = hash_password(new_password)


            user_to_update.role = UserRole(new_role.upper())


            session.commit()
                log_access(SessionManager.get_current_user(), SessionManager.get_role(), f"USER UPDATED, ο χρήστης με ID: {user_id} ενημερώθηκε (Νέος Ρόλος: {user_to_update.role.value})!")
            return True, f"Ο χρήστης με ID: {user_id} ενημερώθηκε επιτυχώς."


             log_error("FAILED USER UPDATE", "UserManagementService", f"FAILED UPDATE ATTEMPT, ο χρήστης με ID: {user_id} δεν βρέθηκε!!!")
        return False, f"Ο χρήστης με ID: {user_id} δεν βρέθηκε."
    except Exception as e:
        session.rollback()
        log_error(type(e).__name__, "UserManagementService", str(e))
        return False, f"Σφάλμα κατά την ενημέρωση στη βάση: Λάθος Ρόλος ή Δεδομένα. ({str(e)})"
    finally:
        session.close()



#vehicle_services.py


from app.database.database import DatabaseManager
from app.services.logger import log_added_vehicle
```

```
class VehicleService:
    @staticmethod
    def add_stolen_vehicle(plate: str, status: str) -> tuple[bool, str]:
        # Υπηρεσία που αναλαμβάνει την προσθήκη ενός νέου οχήματος στη ΒΔ ως "STOLEN/WANTED"
        try:
            db_manager = DatabaseManager()
            db_manager.add_stolen_vehicle(plate, status)
            log_added_vehicle(plate, status)
            return True, f"Το όχημα με πινακίδα {plate} προστέθηκε επιτυχώς!"
        except Exception as e:
            return False, f"Σφάλμα κατά την προσθήκη του οχήματος: {str(e)}"



#gui_login.py

import tkinter as tk
from tkinter import messagebox
from app.utils.sound import SoundGenerator
from app.services.services import AuthService

class LoginWindow:
    def __init__(self, on_success):
        #Δημιουργία του login window
        self.root = tk.Tk()
        self.root.title("APNR System - Login")
        self.root.geometry("350x250")
        self.root.resizable(False, False)
        self.on_success = on_success

        tk.Label(self.root, text="ΣΥΣΤΗΜΑ APNR", font=("Arial", 14, "bold")).pack(pady=20)

        tk.Label(self.root, text="Username:").pack()
```

```
        self.ent_user = tk.Entry(self.root, width=25)
        self.ent_user.pack(pady=5)

        tk.Label(self.root, text="Password:").pack()
        self.ent_pass = tk.Entry(self.root, show="*", width=25)
        self.ent_pass.pack(pady=5)

        tk.Button(self.root, text="LOGIN", bg="#27ae60", fg="white", width=15,
                  command=self.handle_login).pack(pady=20)

        self.root.mainloop()

    def handle_login(self):
        user = self.ent_user.get()
        pw = self.ent_pass.get()

        # Καλούμε το Service που ελέγχει τη βάση
        if AuthService.login(user, pw):
            self.root.destroy()

            _sg= SoundGenerator()

            _sg.play_welcome_intro()
            self.on_success() # Αυτό θα ανοίξει το gui_main
        else:
            messagebox.showerror("Error", "Invalid credentials")

#gui_main.py

import tkinter as tk
from tkinter import scrolledtext, ttk, messagebox
import cv2
import PIL.Image, PIL.ImageTk
```

```
import threading
import time
import os
from app.services.logger import log_error, log_detection
from app.utils.pathfinder import Labyrinth
from app.core.session import SessionManager
import app.core.config1 as config
from app.services.user_services import UserManagementService
from app.services.services import AuthService
from app.services.vehicle_services import VehicleService
from app.utils.sound import SoundGenerator as sound

class MainWindow:
    def __init__(self, root, title, container):
        self.root = root
        self.root.title(title)
        self.root.geometry("1000x750")
        self.root.protocol("WM_DELETE_WINDOW", self.on_exit)
        self.container = container
        self.pipeline = container.pipeline
        self.detection_service = container.detection_service
        self.sound_gen = sound()
        self.is_running = False
        self.cap = None
        self.lock = threading.Lock()
        self.latest_frame = None
        self.latest_results = []
        self.last_ocr_time = 0
        self.frame_count = 0

        self._setup_ui()

    def _setup_ui(self):
```

```
# "Στήσιμο" του UI.

# Αριστερό πάνελ - σετάρισμα
self.left_frame = tk.Frame(self.root, width=250, bg="#2c3e50")
self.left_frame.pack(side=tk.LEFT, fill=tk.Y)
self.left_frame.pack_propagate(False)

tk.Label(self.left_frame, text="APNR System"+"\n"+"\n"+"Edge Computing Node"+
       "\n"+"on Raspberry Pi 3B+",
        font=("Helvetica", 14, "bold"),
        bg="#2c3e50", fg="#ecf0f1").pack(pady=25)



btn_style = {"font": ("Arial", 11), "pady": 5, "padx": 10}

self.btn_start = tk.Button(self.left_frame, text="START CAMERA", bg="#27ae60", fg="white",
                    command=self.start_camera, **btn_style)
self.btn_start.pack(pady=10, fill=tk.X, padx=15)

self.btn_stop = tk.Button(self.left_frame, text="STOP", bg="#c0392b", fg="white",
                    command=self.stop_camera, state=tk.DISABLED, **btn_style)
self.btn_stop.pack(pady=5, fill=tk.X, padx=15)

tk.Frame(self.left_frame, height=2, bg="grey").pack(fill=tk.X, pady=20, padx=10)

self.btn_add = tk.Button(self.left_frame, text="Add Vehicle", bg="#2980b9", fg="white",
                    command=self.add_vehicle, **btn_style)
self.btn_add.pack(pady=5, fill=tk.X, padx=15)

self.btn_log = tk.Button(self.left_frame, text="View Logs", bg="#8e44ad", fg="white",
                    command=self.show_logs, **btn_style)
self.btn_log.pack(pady=5, fill=tk.X, padx=15)
```

```
        self.btn_users = tk.Button(self.left_frame, text="Manage Users", bg="#e67e22", fg="white",
                                   command=self.open_user_management, **btn_style)
        self.btn_users.pack(pady=5, fill=tk.X, padx=15)

        tk.Label(self.left_frame, text="Διεθνές Πανεπιστήμιο της Ελλάδος" + "\n" + "Σχολή Μηχανικών"
+
                 "\n" + "\n" + "Τμήμα Μηχανικών Πληροφορικής " + "\n" + "& Ηλεκτρονικών Συστημάτων",
                 font=("Helvetica", 10, "bold"),
                 bg="#2c3e50", fg="#ecf0f1").pack(pady=20)

                    tk.Label(self.left_frame, text="Created by: "+"\n" +"Μπεσίρης Χρήστος
(iee2019240)"+"\n"+"\n"+
                 "Επιβλέπων:"+"\n"+"Δρ. Διαμαντάρας Κωνσταντίνος"+"\n"+"Καθηγητής",
                 font=("Helvetica", 8, "bold"),
                 bg="#2c3e50", fg="#ecf0f1").pack(pady=20)

        self.btn_exit = tk.Button(self.left_frame, text="EXIT", bg="#34495e", fg="white",
                                  command=self.on_exit, **btn_style)
        self.btn_exit.pack(side=tk.BOTTOM, pady=20, fill=tk.X, padx=15)

        # Σετάρισμα του κεντρικού πάνελ - frame ροής βίντεο
        self.video_frame = tk.Frame(self.root, bg="black")
        self.video_frame.pack(side=tk.TOP, fill=tk.BOTH, expand=True)

        self.lbl_video = tk.Label(self.video_frame, bg="black", text="System Offline",
                                  fg="#7f8c8d", font=("Arial", 18))
        self.lbl_video.pack(expand=True)

        # Εφαρμογή "Status Bar" στο κάτω μέρος της διεπαφής - για να δίνουμε ΟΠΤΙΚΟ feedback στον
χρήστη
        self.status_var = tk.StringVar(value="Waiting for start...")
        self.lbl_status = tk.Label(self.root, textvariable=self.status_var, bd=1, relief=tk.SUNKEN,
                                   anchor=tk.W, font=("Arial", 12, "bold"),
                                   bg="#ecf0f1", padx=10, pady=5)
```

```
        self.lbl_status.pack(side=tk.BOTTOM, fill=tk.X)


    def choose_fastocr_model(self):
        # Παράθυρο επιλογής μοντέλου για το OCR από τον χρήστη.
        win = tk.Toplevel(self.root)
        win.title("Select OCR model")
        win.resizable(False, False)
        win.transient(self.root)


          tk.Label(win, text="Choose fast-plate-ocr model:", font=("Arial", 11, "bold")).pack(padx=12, pady=(12, 6))


        var = tk.StringVar(value=config.FAST_OCR_MODELS[0])
              combo = ttk.Combobox(win, values=config.FAST_OCR_MODELS, textvariable=var, state="readonly", width=42)
        combo.pack(padx=12, pady=6, fill=tk.X)
        combo.current(0)


        result = {"model": None}


        def ok():
            result["model"] = var.get().strip()
            win.destroy()


        def cancel():
            result["model"] = None
            win.destroy()


        btns = tk.Frame(win)
        btns.pack(padx=12, pady=(8, 12), fill=tk.X)
        tk.Button(btns, text="OK", command=ok, width=10).pack(side=tk.LEFT)
        tk.Button(btns, text="Cancel", command=cancel, width=10).pack(side=tk.RIGHT)


        win.grab_set()
```

```
        win.wait_window()
        return result["model"]

    def start_camera(self):
        # Button έναρξης ροής από την κάμερα. Με την έναρξη - προτρέπεται ο
        # χρήστης να επιλέξει μοντέλο OCR
        if self.is_running:
            return

        chosen = self.choose_fastocr_model()
        if not chosen:
            return
        self.current_ocr_model = chosen

        if hasattr(self.pipeline, 'load_ocr'):
            self.pipeline.load_ocr(chosen)

        try:
            self.cap = cv2.VideoCapture(config.CAMERA_ID)
            self.cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)
            self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, config.FRAME_WIDTH)
            self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, config.FRAME_HEIGHT)

            self.is_running = True
            self.btn_start.config(state=tk.DISABLED)
            self.btn_stop.config(state=tk.NORMAL)
            self.lbl_video.config(text="")
            self.status_var.set("Scanning...")

#region Comments
            # SOS !!! Πολύ σημαντικό - διαχείριση threads γιατί χωρίς
            # διάκριση - ο έλεγχος από το OCR "παγώνει" την εκτέλεση
            # του υπόλοιπου κώδικα.
```

```
#endregion
            threading.Thread(target=self.capture_thread, daemon=True).start()
            threading.Thread(target=self.detect_thread, daemon=True).start()
            self.update_gui_frame()

        except Exception as e:
            file_name = os.path.basename(__file__)
            error_type = type(e).__name__
            log_error(error_type, file_name, str(e))
            messagebox.showerror("Camera Error", str(e))

    def stop_camera(self):
        #Διακοπή ροής της κάμερας (feed)
        self.is_running = False
        if self.cap:
            try:
                self.cap.release()
            except: pass

        with self.lock:
            self.latest_frame = None
            self.latest_results = []

        self.btn_start.config(state=tk.NORMAL)
        self.btn_stop.config(state=tk.DISABLED)
        self.lbl_video.config(image='', text="System Offline", bg="black")
        self.status_var.set("Waiting...")
        self.lbl_status.config(bg="#ecf0f1", fg="black")

    def capture_thread(self):
        # Ορισμός και κλείδωμα του capture_thread
        while self.is_running and self.cap:
            try:
```

```
                self.cap.grab()
                ret, frame = self.cap.retrieve()
            except:
                ret, frame = self.cap.read()

            if not ret:
                continue

            with self.lock:
                self.latest_frame = frame
            time.sleep(0.001)

    def detect_thread(self):
        # Ορισμός και "κλείδωμα " του detect_thread

        while self.is_running:
            try:
                with self.lock:
                    frame = None if self.latest_frame is None else self.latest_frame.copy()

                if frame is None:
                    time.sleep(0.01)
                    continue

                self.frame_count += 1
                if (self.frame_count % config.DETECT_EVERY_N_FRAMES) != 0:
                    time.sleep(0.001)
                    continue

                current_time = time.time()
#region Comments
        #-------------------------------------------------------------#
        # SOS !!! - Πέρασμα του ελέγχου στο inference.pipeline.py
```

```
#endregion
            results = self.pipeline.detect_and_ocr(frame)

            with self.lock:
                # Καθαρίζει το box αν δεν βρει πινακίδα
                self.latest_results = results[:config.MAX_DETECTIONS_DRAW] if results else []

            if results and (current_time - self.last_ocr_time) >= config.OCR_INTERVAL_SEC:
                for res in results:
                    plate = res['text']
                    conf = res.get('confidence', 0.0)
                    if not plate:
                        continue
#region Comments
            #-------------------------------------------------------
             # SOS ! Περνάμε τον έλεγχο στο detection_service για να τρέξει τον έλεγχο της βάσης με τη σειρά του
#endregion
                    db_info = self.detection_service.process_detected_plate(plate)

                    if db_info:
                        self.update_ui_status_safe(db_info)
                        if db_info['status'] != "CLEAN":
                                #Εάν η πινακίδα ΔΕΝ είναι "CLEAN" τότε στέλνουμε ALERT - ΗΧΗΤΙΚΟ FEEDBACK
                            self.sound_gen.play_alert_beep()
                        else:
                            # Αλλιώς το buzzer αναπαράγει μόνο ένα "scan_beep" για ΗΧΗΤΙΚΟ FEEDBACK
                            self.sound_gen.play_scan_beep()
                        log_detection(
                            plate=db_info['plate'],
                            status=db_info['status'],
                            confidence=conf,
                            ocr_model=getattr(self, 'current_ocr_model', 'Unknown')
```

```
                )

                self.last_ocr_time = time.time()
            except Exception as e:
                print(f"\n[ΣΦΑΛΜΑ ΣΤΟ DETECT THREAD]: {e}")
                log_error("DetectThreadError", "gui_main.py", str(e))
                with self.lock:
                    self.latest_results = []
                time.sleep(1)

    def update_gui_frame(self):
        #UI Updater μέθοδος για 'ανανέωση' του frame (cap)
        if not self.is_running:
            return

        with self.lock:
            frame = None if self.latest_frame is None else self.latest_frame.copy()
            results = list(self.latest_results)

        if frame is not None:
            for res in results:
                x, y, w, h = res['box']
                plate_text = res.get('text', '')
                conf = res.get('confidence', 0.0)

                display_text = f"{plate_text} ({conf:.2f})" if plate_text else f"Conf: {conf:.2f}"

                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
                cv2.putText(frame, display_text, (x, max(0, y - 10)),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)

            img = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            img_tk = PIL.ImageTk.PhotoImage(image=PIL.Image.fromarray(img))
```

```
            self.lbl_video.img_tk = img_tk
            self.lbl_video.configure(image=img_tk)

        ms = int(1000 / config.GUI_FPS)
        self.root.after(ms, self.update_gui_frame)

    def update_ui_status_safe(self, db_info):
        # UI Updater για το "Status Bar" - αναλαμβάνει την ΟΠΤΙΚΗ ΑΝΑΔΡΑΣΗ - του χρήστη!
        status_text = f"PLATE: {db_info['plate']} | STATUS: {db_info['status']}"
        color = "#27ae60" # Default Green

        if db_info['status'] == "STOLEN": color = "#c0392b"
        elif db_info['status'] == "WANTED": color = "#d35400"

        self.root.after(0, lambda: self.status_var.set(status_text))
        self.root.after(0, lambda: self.lbl_status.config(bg=color, fg="white"))

    def add_vehicle(self):
        # Άνοιγμα παραθύρου εισαγωγής νέας πινακίδας οχήματος στη βάση
        add_win = tk.Toplevel(self.root)
        add_win.title("Προσθήκη Οχήματος")
        add_win.geometry("300x250")
        add_win.resizable(False, False)
        add_win.transient(self.root)
        add_win.grab_set()

        tk.Label(add_win, text="Αριθμός Πινακίδας (π.χ. ABC1234):", font=("Arial", 10)).pack(pady=(20,
5))
        ent_plate = tk.Entry(add_win, width=20, font=("Arial", 12), justify="center")
        ent_plate.pack()

        tk.Label(add_win, text="Κατάσταση Οχήματος:", font=("Arial", 10)).pack(pady=(15, 5))
            cmb_status = ttk.Combobox(add_win, values=["STOLEN", "WANTED"], state="readonly",
font=("Arial", 11), width=18)
```

```
        cmb_status.current(0)
        cmb_status.pack()

        btn_save = tk.Button(add_win, text="Αποθήκευση", bg="#c0392b", fg="white", font=("Arial", 10, "bold"), width=15)
#region Comments
# ------------------------------------------------------
#   SOS !!!  Δημιουργία functions για να τρέξουν ΑΣΥΓΧΡΟΝΑ  σε δικό τους thread (lib threading) ώστε η αποθήκευση
#            των δεδομένων που έχει εισάγει ο χρήστης σαν νέο "STOLEN/WANTED" όχημα , να γίνει χωρίς να "κολλάει" η
#          υπόλοιπη διεπαφή.
#endregion
        def _bg_task(plate, status):
            try:
                success, msg = VehicleService.add_stolen_vehicle(plate, status)
                self.root.after(0, lambda: _on_result(success, msg))
            except Exception as e:
                self.root.after(0, lambda: _on_result(False, f"Σφάλμα Βάσης: {str(e)}"))

        def _on_result(success, msg):
            if success:
                messagebox.showinfo("Επιτυχία", msg, parent=add_win)
                add_win.grab_release()
                add_win.destroy()
            else:
                messagebox.showerror("Σφάλμα", msg, parent=add_win)
                btn_save.config(state=tk.NORMAL, text="Αποθήκευση")

        def on_save():
            plate = ent_plate.get().strip().upper()
            status = cmb_status.get().strip()

            if not plate:
```

```
            messagebox.showwarning("Προσοχή", "Το πεδίο της πινακίδας δεν μπορεί να είναι κενό!",
parent=add_win)
            return

        btn_save.config(state=tk.DISABLED, text="Παρακαλώ περιμένετε...")

        # Εκκίνηση του thread αποθήκευσης
        threading.Thread(target=_bg_task, args=(plate, status), daemon=True).start()

    btn_save.config(command=on_save)
    btn_save.pack(pady=25)

  def show_logs(self):
    # Εμφάνιση του detection.log στο UI σε νέο παράθυρο.
    win = tk.Toplevel(self.root)
    win.title("Detection History")
    txt = scrolledtext.ScrolledText(win, width=70, height=20)
    txt.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)

    logs_path = Labyrinth.get_results_file_path('detections.log')

    try:
        with open(logs_path, "r") as f:
            data = f.read().strip()
        txt.insert(tk.INSERT, data if data else "No logs yet.")
    except:
        txt.insert(tk.INSERT, "No logs found.")
    txt.bind("<Key>", lambda e: "break")
    txt.yview(tk.END)

  def on_exit(self):
    # Exit button - για το ομαλό κλείσιμο της εφαρμογής.
    if messagebox.askokcancel("Exit", "Quit system?"):
```

```
        AuthService.logout()
        self.stop_camera()
        self.root.destroy()
        os._exit(0)

    def open_user_management(self):
        #region Comments
        # Διαχείριση χρηστών - User Management
        # Ελεγχος πρώτα με βάση το RBAC - το policy είναι MONO user.role.ADMINISTRATOR Μπορεί να το χειριστεί!
        # ΔΕΝ έχουμε τα Policies σε ξεχωριστό αρχείο policy.py λόγω μεγέθους εφαρμογής- ΚΑΝΟΝΙΚΑ ΠΡΕΠΕΙ.
        # Εδώ το χειριζόμαστε "μπακαλίστικα" - απλουστευμένα - ακατάλληλο για Enterprize class εφαρμογή!
        #endregion
        if SessionManager.get_role() != "ADMINISTRATOR":
            messagebox.showerror("Access Denied", "Μόνο ο ΔΙΑΧΕΙΡΙΣΤΗΣ έχει πρόσβαση.", parent=self.root)
            return

        win = tk.Toplevel(self.root)
        win.title("Manage Users")
        win.geometry("550x450")
        win.transient(self.root)
        win.grab_set()

        tabs = ttk.Notebook(win)
        tabs.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        # Πρώτο tab : CREATE
        tab_create = tk.Frame(tabs)
        tabs.add(tab_create, text="Create User")

        tk.Label(tab_create, text="Username:").pack(pady=(20, 5))
```

```
new_username_input = tk.Entry(tab_create, width=30)
new_username_input.pack()

tk.Label(tab_create, text="Password:").pack(pady=5)
new_paswd_input = tk.Entry(tab_create, width=30, show="*")
new_paswd_input.pack()

tk.Label(tab_create, text="Role:").pack(pady=5)
combobox_role = ttk.Combobox(tab_create, values=["supervisor", "user"], state="readonly")
combobox_role.current(0)
combobox_role.pack()

def save_user():
    # Αποθήκευση νέου χρήστη
    u = new_username_input.get().strip()
    p = new_paswd_input.get().strip()
    r = combobox_role.get().strip()

    if not u or not p:
        messagebox.showwarning("Warning", "Τα πεδία δεν μπορεί να είναι κενά!", parent=win)
        return

    success, message = UserManagementService.create_user(u, p, r)
    if success:
        messagebox.showinfo("Success", message, parent=win)
        new_username_input.delete(0, tk.END)
        new_paswd_input.delete(0, tk.END)
        load_users()
    else:
        messagebox.showerror("Error", message, parent=win)

            tk.Button(tab_create, text="Save User", bg="#27ae60", fg="white",
command=save_user).pack(pady=25)
```

```
# Δεύτερο tab : MANAGE
tab_manage = tk.Frame(tabs)
tabs.add(tab_manage, text="Manage Users")

columns = ("id", "username", "role")
tree = ttk.Treeview(tab_manage, columns=columns, show="headings", height=10)
tree.heading("id", text="ID")
tree.heading("username", text="Username")
tree.heading("role", text="Role")
tree.column("id", width=50, anchor=tk.CENTER)
tree.column("username", width=200, anchor=tk.W)
tree.column("role", width=150, anchor=tk.CENTER)
tree.pack(pady=10, fill=tk.BOTH, expand=True)

def load_users():
   # Φόρτωση των χρηστών της ΒΔ μας - OXI ADMINS!! το χειριζόμαστε από το user_services.py
    for item in tree.get_children():
        tree.delete(item)
    users = UserManagementService.get_manageable_users()
    for u in users:
        tree.insert("", tk.END, values=(u['id'], u['username'], u['role']))

def delete_user():
    # Διαγραφή χρήστη από τη βάση μας
    selected = tree.selection()
    if not selected: return
    item = tree.item(selected[0])
    uid, uname = item['values'][0], item['values'][1]

    if messagebox.askyesno("Confirm", f"Διαγραφή του χρήστη: {uname};", parent=win):
        success, message = UserManagementService.delete_user(uid)
        if success:
```

```
            messagebox.showinfo("Success", f"Ο χρήστης {uname} διαγράφηκε.", parent=win)
            load_users()
        else:
            messagebox.showerror("Error", message, parent=win)

    def update_user():
        # Ενημέρωση στοιχείων χρήστη
        selected = tree.selection()
        if not selected: return
        item = tree.item(selected[0])
        uid, uname, urole = item['values'][0], item['values'][1], item['values'][2]

        upd_win = tk.Toplevel(win)
        upd_win.title(f"Update: {uname}")
        upd_win.geometry("300x250")
        upd_win.transient(win)
        upd_win.grab_set()

        tk.Label(upd_win, text="New Password\n(αφήστε κενό για διατήρηση παλιού):").pack(pady=10)
        user_new_paswd = tk.Entry(upd_win, width=25, show="*")
        user_new_paswd.pack()

        tk.Label(upd_win, text="Change Role:").pack(pady=10)
        user_new_role = ttk.Combobox(upd_win, values=["USER", "SUPERVISOR"], state="readonly")
        user_new_role.set(urole)
        user_new_role.pack()

        def save_update():
            # Αποθήκευση ενημερωμένων στοιχείων
            new_p = user_new_paswd.get().strip()
            new_r = user_new_role.get().strip()
```

```
            if messagebox.askyesno("Confirm", f"Ενημέρωση του χρήστη: {uname};", parent=win):
                success, message = UserManagementService.update_user(uid, new_p, new_r)
                if success:
                    messagebox.showinfo("Success", f"Ο χρήστης {uname} ενημερώθηκε.", parent=win)
                    upd_win.destroy()
                    load_users()
                else:
                    messagebox.showerror("Error", message, parent=win)

                        tk.Button(upd_win, text="Save Update", bg="#2980b9", fg="white", command=save_update).pack(pady=20)

    btn_frame = tk.Frame(tab_manage)
    btn_frame.pack(pady=5)
                          tk.Button(btn_frame, text="Edit", bg="#f39c12", fg="white", command=update_user).pack(side=tk.LEFT, padx=10)
                        tk.Button(btn_frame, text="Delete", bg="#c0392b", fg="white", command=delete_user).pack(side=tk.LEFT, padx=10)

    load_users()

#hasher.py

import bcrypt
#region Comments
# Χρήση της βιβλιοθήκης bcrypt για την βέλτιστη αφάλεια κωδικών.
# η ενσωματωμένη χρήση salt αυξάνει την ασφάλεια.
#endregion
def hash_password(password: str) -> str:
    # Κάνουμε encode το password σε bytes- απαραίτητο για να λειτουργήσει το bcrypt.
    password_bytes = password.encode('utf-8')

    # Δημιουργούμε το salt και το hash
    salt = bcrypt.gensalt()
```

```
    hashed_bytes = bcrypt.hashpw(password_bytes, salt)

    # Επιστρέφουμε string για τη συναλλαγή- καλύτερη διαχείριση στη ΒΔ μας.
    return hashed_bytes.decode('utf-8')

def verify_password(stored_hash: str, provided_password: str) -> bool:
    # Μετατρέπουμε το provided_password και το stored_hash σε bytes για να γίνει σύγκριση.
    if not stored_hash or not provided_password:
        return False  # Αν κάποιο από τα δύο είναι κενό, δεν μπορεί να ταιριάξει -ΑΠΟΡΡΙΨΗ

    provided_password_bytes = provided_password.encode('utf-8')
    stored_hash_bytes = stored_hash.encode('utf-8')

    try:
        # Η checkpw ελέγχει αυτόματα το salt και το hash
        return bcrypt.checkpw(provided_password_bytes, stored_hash_bytes)
    except ValueError:
        # Σε περίπτωση που το stored_hash δεν είναι έγκυρο hash, απορρίπτουμε την προσπάθεια.
        return False



#helpers.py

import time
import cv2

#Βοηθητικές μέθοδοι

def now_ts():
    # Δημιουργία timestamp
    return time.strftime('%Y-%m-%d %H:%M:%S')

def clamp(v, lo, hi):
```

```
    # Περιορισμός τιμής v στο διάστημα [lo, hi]
    return max(lo, min(hi, v))

def letterbox(image, new_shape=(320, 320), color=(114, 114, 114)):
    # Προσαρμογή μεγέθους της εικόνας /  φορμάρισμα της εικόνας με padding
    h, w = image.shape[:2]
    new_w, new_h = new_shape

    r = min(new_w / w, new_h / h)
    nw, nh = int(round(w * r)), int(round(h * r))

    img_resized = cv2.resize(image, (nw, nh), interpolation=cv2.INTER_LINEAR)

    dw = new_w - nw
    dh = new_h - nh
    left = dw // 2
    right = dw - left
    top = dh // 2
    bottom = dh - top

    img_lb = cv2.copyMakeBorder(img_resized, top, bottom, left, right,
                                cv2.BORDER_CONSTANT, value=color)
    return img_lb, r, (left, top)

def preprocess_for_ocr(roi_bgr):
    # Προ-επεξεργασία του Region of Interest (ROI) για τη χρήση του OCR.
    if roi_bgr is None or roi_bgr.size == 0:
        return None
    gray = cv2.cvtColor(roi_bgr, cv2.COLOR_BGR2GRAY)
    gray = cv2.resize(gray, None, fx=2.5, fy=2.5, interpolation=cv2.INTER_CUBIC)
    gray = cv2.bilateralFilter(gray, 7, 50, 50)
    gray = cv2.equalizeHist(gray)
    _, th = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
```

```
    return th

#pathfinder.py

import os

CURRENT_DIR = os.path.dirname(os.path.abspath(__file__))
BASE_DIR = os.path.abspath(os.path.join(CURRENT_DIR, '..','..'))  # Μετάβαση σε επίπεδο root
MODEL_DETECTOR_PATH = os.path.join(BASE_DIR,   'models', 'license_plate_detector', 'license-plate-finetune-v1n.onnx')
MODEL_OCR_FOLDER_PATH = os.path.join(BASE_DIR, 'models', 'fast-plate-ocr')
DB_PATH = os.path.join(BASE_DIR, 'db','police_db.sqlite')
RESULTS_FILE_PATH = os.path.join(BASE_DIR,  'logs',  'detections.log')
ACCESS_LOG_FILE_PATH = os.path.join(BASE_DIR, 'logs', 'access.log')
ERROR_LOG_FILE_PATH = os.path.join(BASE_DIR, 'logs', 'errors.log')
LOGS_DIR_PATH = os.path.join(BASE_DIR, 'logs')
#region Comments

# Η κλάση Labyrinth έχει στατικές μεθόδους που επιστρέφουν τα paths για , ΒΔ, μοντέλα , Logs κλπ.
# εάν οι φάκελοι ΔΕΝ υπάρχουν, τότε δημιουργούνται αυτόματα.
#endregion
class Labyrinth:

    @staticmethod
    def get_model_detector() -> str:
        # Επιστρέφει το path του μοντέλου ανίχνευσης πινακίδων.
        target_path = MODEL_DETECTOR_PATH
        if not os.path.exists(target_path):
            raise FileNotFoundError(f"Detector model not found at path: {target_path}")
        return str(target_path)

    @staticmethod
    def get_model_ocr(filename : str) -> str:
```

```
        #Επιστρέφει το path ενός μοντέλου OCR - ελέγχοντας την ύπαρξή του.
        target_path = os.path.join(MODEL_OCR_FOLDER_PATH, filename)
        if not os.path.exists(target_path):
                        raise FileNotFoundError(f"Model file '{filename}' not found in '{MODEL_OCR_FOLDER_PATH}'")
        return str(target_path)

    @staticmethod
    def get_db_path(db_name='police_db.sqlite')-> str:
        #Επιστρέφει το path της ΒΔ. Σε περίπτωση που ο φάκελος ΔΕΝ υπάρχει , δημιουργείται.
        target_path = os.path.join(BASE_DIR, 'db',db_name)
        directory = os.path.dirname(target_path)

        if not os.path.exists(directory):
            os.makedirs(directory) # Δημιουργεί τον φάκελο 'db' αν λείπει

        return str(target_path)

    @staticmethod
    def get_results_file_path(log_name='detections.log') -> str:
        #Επιστρέφει το path του detections.log
        target_path = os.path.join(BASE_DIR,  'logs',  log_name)

        # Παίρνουμε μόνο τον φάκελο από το πλήρες μονοπάτι (π.χ. BASE_DIR/logs)
        directory = os.path.dirname(target_path)

        # Ελέγχουμε αν υπάρχει ο φάκελος. Αν όχι, τον δημιουργούμε.
        if not os.path.exists(directory):
            os.makedirs(directory) # Δημιουργείται ο φάκελος 'logs' αν λείπει

        return str(target_path)
```

```
#sound.py

import time
import threading
from gpiozero import TonalBuzzer
from gpiozero.tones import Tone
#region Comments
#Κλάση που περιέχει μεθόδους για την αναπαραγωγή ήχων από το passive buzzer module.
# Γίνεται χρήση threading για να μην μπλοκάρει το κύριο πρόγραμμα κατά την αναπαραγωγή ήχων.
#endregion
class SoundGenerator:
    _instance = None

    def __new__(cls, *args, **kwargs):
        if cls._instance is None:
            cls._instance = super(SoundGenerator, cls).__new__(cls)
            cls._instance._initialized = False
        return cls._instance

    def __init__(self, pin=17):
        if self._initialized:
            return

        self.ok = True
        self._lock = threading.Lock()

        try:
            self.buzzer = TonalBuzzer(pin)
        except Exception as e:
            print(f"[Sound Error] Buzzer Init Error: {e}")
            self.ok = False
```

```
    self._initialized = True

def _run_in_thread(self, task, action_name="Audio"):
    # Τρέχουμε τον ήχο σε ξεχωριστό thread.
    def wrapper():
        # Δοκιμάζει να πάρει την κλειδαριά
        if self._lock.acquire(blocking=False):
            try:
                task()
            except Exception as e:
                print(f"[Sound Error] Σφάλμα κατά την αναπαραγωγή {action_name}: {e}")
            finally:
                self._lock.release()
        else:
            # Αν βομβαρδιστεί με πολλές αναγνωρίσεις, απλά αγνοεί τις έξτρα
            pass

    threading.Thread(target=wrapper, daemon=True).start()

def play_scan_beep(self):
    #Ηχος σκαναρίσματος.
    if not self.ok: return

    def _task():

        self.buzzer.play(Tone(frequency=600))
        time.sleep(0.15)
        self.buzzer.stop()

    self._run_in_thread(_task, "Scan Beep (CLEAN)")

def play_alert_beep(self):
    # Ήχος συναγερμού για "STOLEN/WANTED" οχήματα.
```

```
        if not self.ok: return

        def _task():
            for _ in range(3):
                self.buzzer.play(Tone(frequency=500))
                time.sleep(0.3)
                self.buzzer.stop()
                time.sleep(0.1)

        self._run_in_thread(_task, "Alert Beep (STOLEN/WANTED)")

    def play_welcome_intro(self):
        # Ήχος έναρξης εφαρμογής μετά το επιτυχημένο login του χρήστη. Σκοπός είναι
        # να επιβεβαιωθεί η σωστή λειτουργία του passive buzzer module.
        if not self.ok: return

        def _task():
            self.buzzer.play(Tone(frequency=600))
            time.sleep(2.0)
            self.buzzer.stop()

        self._run_in_thread(_task, "Welcome Intro")


#main.py

import tkinter as tk
from app.ui.gui_login import LoginWindow
from app.ui.gui_main import MainWindow
from app.core.container import Container

def launch_main_app():
    # Η συνάρτηση που καλείται μετά από επιτυχημένο login και ξεκινάει το κύριο παράθυρο.
```

```
    root = tk.Tk()
    # Αρχικοποίηση Container (Inference + Services)
    # Εδώ φορτώνονται τα μοντέλα ONNX και η σύνδεση με τη ΒΔ
    try:
        app_container = Container()
    except Exception as e:
        from tkinter import messagebox
        messagebox.showerror("Initialization Error", f"Αποτυχία εκκίνησης συστήματος: {e}")
        return

    # Εκκίνηση MainWindow
    # Περνάμε το container ώστε το UI να έχει πρόσβαση στο AI και τη Βάση
    app = MainWindow(root, "APNR - Edge Computing / RPi 3B+ Christos Besiris", app_container)

    root.mainloop()

if __name__ == "__main__":
    # Το πρόγραμμα ξεκινάει πάντα από το Login
    # Το LoginWindow θα καλέσει τη launch_main_app μόνο αν τα credentials είναι σωστά
    LoginWindow(on_success=launch_main_app)
```

```
#create_admin.py

from app.database.connection import DBConnectionManager
from app.database.dbmodels import User, UserRole
from app.utils.hasher import hash_password

#Κλάση για την αρχικοποίηση της ΒΔ και τη δημιουργία ενός χρήστη ADMINISTRATOR με username:"admin"
#και password:"iee2019240". Η κλάση ελέγχει αν ο χρήστης υπάρχει ήδη και αν όχι, τον δημιουργεί με το ρόλο ADMIN.

def create_admin_user():
    print("Αρχικοποίηση σύνδεσης με τη βάση δεδομένων...")
```

```
    DBConnectionManager.setup_db()
    session = DBConnectionManager.get_session()

    new_username = "admin"
    new_password = "iee2019240"

    try:
        existing_user = session.query(User).filter_by(username=new_username).first()
        if existing_user:
            print(f"ERROR:  Ο χρήστης '{new_username}' υπάρχει ήδη στη βάση.")
            return

        hashed_pw = hash_password(new_password)

        new_admin = User(
            username=new_username,
            password=hashed_pw,
            role=UserRole.ADMIN
        )

        session.add(new_admin)
        session.commit()

        print(f"SUCCESS ! Ο χρήστης '{new_username}' δημιουργήθηκε.")

    except Exception as e:
        session.rollback()
        print(f"ERROR : Σφάλμα κατά τη δημιουργία του χρήστη: {e}")
    finally:
        session.close()

if __name__ == "__main__":
    create_admin_user()
```