ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ ΓΙΑ ΚΙΝΗΤΕΣ ΣΥΣΚΕΥΕΣ ΜΕ ΧΡΗΣΗ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ ΓΙΑ ΤΗΝ ΑΝΙΧΝΕΥΣΗ ΤΗΣ ΤΟΠΟΘΕΤΗΣΗΣ ΤΟΥ ΣΩΜΑΤΟΣ ΚΑΤΑ ΤΗΝ ΕΚΓΥΜΝΑΣΗ

Παπαπάσχος Θωμάς

Αριθμός Μητρώου: 71346789

Επιβλέπων καθηγητής:

Μιχαηλίδης Εμμανουήλ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ ΓΙΑ ΚΙΝΗΤΕΣ ΣΥΣΚΕΥΕΣ ΜΕ ΧΡΗΣΗ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ ΓΙΑ ΤΗΝ ΑΝΙΧΝΕΥΣΗ ΤΗΣ ΤΟΠΟΘΕΤΗΣΗΣ ΤΟΥ ΣΩΜΑΤΟΣ ΚΑΤΑ ΤΗΝ ΕΚΓΥΜΝΑΣΗ

Θωμάς Παπαπάσχος
Α.Μ. 71346789

Εισηγητής: Μιχαηλίδης Εμμανουήλ

Εξεταστική Επιτροπή:

Α/Α	ΟΝΟΜΑΤΕΠΩΝΥΜΟ	ΒΑΘΜΙΔΑ/ΙΔΙΟΤΗΤΑ/ΤΜΗΜΑ	ΨΗΦΙΑΚΗ ΥΠΟΓΡΑΦΗ
1	Μιχαηλίδης Εμμανουήλ	Ακαδημαϊκός Υπότροφος	Emmanouil Michailidis Digitally signed by Emmanouil Michailidis Date: 2023.07.17 14:58:53 +03'00'
2	Παναγιώτης Γιαννακόπουλος	Καθηγητής	Panagiotis Giannakopoulos Digitally signed by Panagiotis Giannakopoulos Date: 2023.07.18 17:57:40 +03'00'
3	Νικόλαος Μυριδάκης	Επίκουρος Καθηγητής	Nikolaos Myridakis Digitally signed by Nikolaos Myridakis Date: 2023.07.17 15:54:06 +03'00'

Ημερομηνία εξέτασης: 14/07/2023

Δήλωση Συγγραφέα Διπλωματικής Εργασίας

Ο κάτωθι υπογεγραμμένος Παπαπάσχος Θωμάς του Θεοδώρου, με αριθμό μητρώου 71346789 φοιτητής του Πανεπιστημίου Δυτικής Αττικής της Σχολής Μηχανικών του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών, δηλώνω υπεύθυνα ότι:

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

Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

Ο Δηλών

Ευχαριστίες

Η διπλωματική αυτή εργασία εκπονήθηκε υπό την καθοδήγηση και υποστήριξη του καθηγητή μου Δρ. Εμμανουήλ Μιχαηλίδη, τον οποίο ευχαριστώ θερμά.

Επίσης, θα ήθελα να ευχαριστήσω την οικογένεια μου για την υποστήριξη της όλα αυτά τα χρόνια.

Περιεχόμενα

Περίληψη

Στις μέρες μας η τεχνολογία είναι ένα αναπόσπαστο κομμάτι της καθημερινής ζωής του ανθρώπου. Ένα κομμάτι λογισμικού ή υλικού υπάρχει σε κάθε τομέα ώστε να τον βοηθήσει είτε ως προς το να επιτεύξει ένα στόχο είτε ως προς το να βελτιώσει κάποιο αποτέλεσμα. Ειδικότερα, παρατηρείται μια αύξησή ως προς την χρήση κινητών συσκευών και εφαρμογών για την διευκόλυνση εργασιών ή ακόμα και την μετατροπή αυτών των εργασιών σε μια πιο ενδιαφέρουσα ενασχόληση.

Αδιαμφισβήτητα, η σωματική άθληση είναι μια ενασχόληση η οποία έχει πολλαπλά οφέλη για έναν άνθρωπο διαχρονικά αλλά ακόμα περισσότερο στις μέρες μας όπου υπάρχει αύξησή στην καθιστική ζωή. Για τον λόγο αυτό προχωρήσαμε σε ανάλυση και υλοποίηση εφαρμογής, βασιζόμενη στο ML Kit και την εφαρμογή ML Kit vision quickstart, για κινητές συσκευές, η οποία με χρήση μηχανικής μάθησης και της κάμερας του κινητού, είναι ικανή να εντοπίζει την στάση του σώματος του αθλούμενου κατά την διάρκεια της άσκησης, να μετράει τις επαναλήψεις που έχουν εκπονηθεί και να τις αποθηκεύει.
Οι στόχοι της χρήσης της παραπάνω εφαρμογής είναι:

- Ο ασκούμενος να βρίσκει την άσκηση πιο ενδιαφέρουσα και να επιθυμεί να ασκηθεί ξανά ώστε να περάσει τα προηγουμένως αποθηκευμένα στατιστικά επαναλήψεων της άσκησης.
- Να έχει πρόσβαση σε μια κινητή βάση δεδομένων με το προσωπικό του ιστορικό εκτέλεσης της άσκησης.
- Ο ασκούμενος να δέχεται ανατροφοδότηση για το για το πότε έχει καταφέρει την επίτευξη μιας επανάληψης. Αυτό έχει ως συνέπεια την καλύτερη εκτέλεση της άσκησης.

Λέξεις-Κλειδιά: Εφαρμογή κινητών συσκευών, Android, Τεχνητή Νοημοσύνη, Μηχανική Μάθηση, Kotlin.

Abstract

Nowadays technology is an integral part of everyday life. A piece of software or hardware exists in every area to help humanity to either achieve a goal or improve an outcome.
Specifically, there is an increase in the use of mobile devices and applications, to facilitate tasks or even turn these tasks into a more interesting pastime.

Undoubtedly, physical exercise is an activity that has multiple benefits for a person over time but even more so nowadays where there is an increase in sedentary lifestyle. For this reason, we proceeded to the analysis and implementation of an application, based on ML Kit and the sample app ML Kit vision quickstart, for mobile devices, which, using machine learning and the phone's camera, is able to detect the posture of an athlete during the exercise, to count the repetitions that have been performed and to store them in a mobile database.
The goals of using the above application are:

- The trainee finds the exercise more interesting and wishes to practice again to surpass the previously saved repetition statistics of the exercise.
- To have access to a mobile database with their personal log of performing the exercise.
- The trainee receives feedback on when they have achieved a repetition. This results in the best execution of the exercise.

Keywords: Mobile App, Android, Artificial Intelligence, Machine Learning, Kotlin.

Κατάλογος εικόνων

Κεφάλαιο 1ο: Εισαγωγή

1.1 Εισαγωγή

Στην απλούστερη μορφή της, η τεχνητή νοημοσύνη είναι ένα πεδίο που συνδυάζει την επιστήμη των υπολογιστών και ισχυρά σύνολα δεδομένων, για να επιτρέψει την επίλυση προβλημάτων. Περιλαμβάνει επίσης υποπεδία της μηχανικής μάθησης (machine learning) και της βαθιάς μάθησης (deep learning), τα οποία αναφέρονται συχνά σε συνδυασμό με την τεχνητή νοημοσύνη. Αυτοί οι κλάδοι αποτελούνται από αλγόριθμους τεχνητής νοημοσύνης που επιδιώκουν να δημιουργήσουν έμπειρα συστήματα που κάνουν προβλέψεις ή ταξινομήσεις βάσει των δεδομένων εισόδου. **[1]**

Η μηχανική μάθηση (machine learning) είναι ένας κλάδος της τεχνητής νοημοσύνης που εστιάζει στη χρήση δεδομένων και αλγορίθμων για τη μίμηση του τρόπου με τον οποίο μαθαίνουν οι άνθρωποι, βελτιώνοντας σταδιακά την ακρίβειά της. **[2]**

Οι τεχνικές μηχανικής μάθησης που χρησιμοποιούνται ανάλογα με την φύση του κάθε προβλήματος ανήκουν είτε στην μάθηση με επίβλεψη είτε στην μάθηση χωρίς επίβλεψη. Στη μάθηση με επίβλεψη (supervised learning) το σύστημα καλείται να "μάθει" μια έννοια ή συνάρτηση από ένα σύνολο δεδομένων, η οποία αποτελεί περιγραφή ενός μοντέλου. Και ονομάζεται έτσι διότι θεωρείται ότι υπάρχει κάποιος επιβλέπον ο οποίος παρέχει την σωστή τιμή εξόδου της συναρτήσεως.**[3]**

Σε αντίθεση με το παραπάνω, στην μάθηση χωρίς επίβλεψη (unsupervised learning) το σύστημα καλείται να φτιάξει από μόνο του συσχετίσεις και να δημιουργήσει πρότυπα χωρίς να είναι γνωστό εάν αυτά υπάρχουν.

Μια εφαρμογή (app ή application) για κινητές συσκευές είναι ένα πρόγραμμα λογισμικού που έχει σχεδιαστεί με σκοπό την εκτέλεση σε φορητές συσκευές όπως έξυπνο τηλέφωνο (smartphone), έξυπνη ταμπλέτα (tablet) , έξυπνο ρολόι (smartwatch) αλλά και έξυπνο αυτοκίνητο (smart auto) και έξυπνη τηλεόραση (smart tv).

Το μεγαλύτερο πλεονέκτημα μιας εφαρμογής κινητού, είναι η ικανότητα της φορητότητας της ίδιας της συσκευής, η οποία μας επιτρέπει να έχουμε μια συσκευή πολλαπλών χρήσεων που δέχεται τις εισόδους μέσω από κάποιους από τους αισθητήρες που κατέχει όπως της κάμερας, του μικροφώνου, του γυροσκόπιού κ.ά. Τις επεξεργάζεται με το CPU της και βγάζει συμπεράσματα τα οποία τα βλέπουμε στην οθόνη (τυπική έξοδος) ή αποθηκεύονται σε μια τοπική στο κινητό βάση ή ακόμα και απομακρυσμένη βάση δεδομένων με σύνδεση στο διαδίκτυο και όλα αυτά ενώ τρέχει χωρίς σύνδεση σε τροφοδοσία αλλά με χρήση της μπαταρίας.

1.2 Κίνητρο για τη διεξαγωγή της εργασίας

Τις δυνατότητες που μας παρέχονται πλέον από μια μικρή συσκευή που χωράει στην τσέπη μας ή φοριέται στον καρπό μας δύσκολα θα μπορούσαμε να τις φανταστούμε πριν από 30 χρονιά, όπου για το απλούστερο των εργασιών θα χρειαζόμασταν ένα ολόκληρο δωμάτιο γεμάτο από καλώδια.

Πλέον διαθέτουμε φορητές συσκευές εξοπλισμένες με διάφορα είδη αισθητηρίων με μεγάλη επεξεργαστική ισχύ που παρέχουν την δυνατότητα συνθέτων υπολογισμών, μεγάλη διάρκεια ζωής και σύνδεση στο διαδίκτυο με μεγάλη ταχύτητα επικοινωνίας χρησιμοποιώντας πρωτοκολλά όπως το 4G, 5G.

Αυτό είναι ακριβώς και το κίνητρο για την διεξαγωγή της εργασίας αυτής, δηλαδή η εξερεύνηση των δυνατοτήτων μας και η διευκόλυνση που αυτές μας παρέχουν στο πεδίο της άθλησης και της εκγύμνασης με την δημιουργία μιας εφαρμογής βοηθού εκγύμνασης.

1.3 Στόχος της εργασίας

Ο στόχος της εργασίας είναι η ανάπτυξη μιας εφαρμογής κινητού και συγκεκριμένα συστήματος Android, η οποία θα βοηθά τον χρήστη να πραγματοποιήσει και να μετρήσει έλξεις σε μονόζυγο. Αυτό θα το κάνει μετρώντας και αποθηκεύοντας τις επαναλήψεις του χρήστη εάν αυτές είναι αποδέκτες.
Το μέτρημα των επαναλήψεων θα γίνεται με χρήση της κάμερας η οποία θα είναι στραμμένη προς τον χρήστη και το κινητό σε μια απόσταση έτσι ώστε να φαίνεται ολόκληρο το σώμα του χρήστη. Μια επανάληψη θα θεωρείται επιτυχημένη όταν το σώμα έχει εισέλθει στην ζητούμενη πόζα/τοποθέτηση και έχει εξέλθει από αυτήν, πρακτικά ο χρήστης θα έχει βρεθεί στο χαμηλότερο σημείο του εύρους κίνησης για αυτήν την άσκηση και έπειτα στο υψηλότερο.

Η εφαρμογή λαμβάνοντας καρέ ή πλαίσια (frames) από την κάμερα τα αναλύει σε πραγματικό χρόνο και εντοπίζει την τοποθέτηση του σώματος και αν αυτή ταιριάζει σε κάποιο σημείο της άσκησης. Το τελευταίο γίνεται χρησιμοποιώντας μηχανικό μοντέλο το οποίο έχει εκπαιδευτεί ώστε να αναγνωρίζει συγκεκριμένες πόζες. Κατά την επίτευξη μίας πλήρους κίνησης αυξάνεται ο μετρητής και ηχεί διακριτικός ήχος που σηματοδοτεί την επίτευξη της άσκησης. Ο χρήστης όταν σταματήσει να ασκείται στο πάτημα ενός κουμπιού σταματάει την μέτρηση και έχει την ευκαιρία να διορθώσει τον αριθμό των επαναλήψεων που έχει εκτελέσει σε περίπτωση λάθους στην ανίχνευση όπως ψευδή αρνητικά ή ψευδή θετικά αποτελέσματα (false negatives ή false positives), πρώτου αποθηκεύσει το αποτέλεσμα. Έχει επίσης και την δυνατότητα να μοιραστεί τον αριθμό των επαναλήψεων που έκανε με email ή στα μέσα κοινωνικής δικτύωσης.

Πηγαίνοντας από το κεντρικό μενού στην οθόνη του ιστορικού μπορεί να δει τα αποθηκευμένα στην συσκευή στατιστικά στοιχεία του, όπως τις συνολικές επαναλήψεις που έχει εκτελέσει, το μέγιστο αριθμό επαναλήψεων που έχει εκτελέσει σε ένα μόνο σετ όπως και αλλά στατιστικά και λίστα με τις ημερομηνίες, για κάθε ημέρα που έχει κάνει χρήση της εφαρμογής, οπού μπορεί αν δει τα σετ και τις επαναλήψεις που εκτέλεσε.

1.4 Τεχνολογίες που χρησιμοποιήθηκαν

Το Android είναι ένα λειτουργικό σύστημα, το οποίο αρχικά είχε δημιουργηθεί από την εταιρία Android Inc. και αργότερα αποκτήθηκε από την Google. Η αρχική έκδοση του έγινε διαθέσιμη το 2008 και σήμερα είναι ένα από τα πιο διαδεδομένα λειτουργικά συστήματα. Είναι βασισμένο στον πυρήνα του Linux και σχεδιασμένο έτσι ώστε να λειτουργεί σε διαφορά είδη συσκευών όπως έξυπνα κινητά (smartphones), έξυπνες ταμπλέτες (tablets) όπως και τηλεοράσεις (smart TVs) αλλά και έξυπνα ρολόγια (smartwatches) με την ειδική έκδοση ονόματι WearOS και αυτοκίνητα (Android Auto).**[4]**

Το Android Framework είναι ένα σύνολο στοιχείων λογισμικού που αποτελούν τη βάση για τη δημιουργία εφαρμογών στο Android. Περιλαμβάνει το λειτουργικό σύστημα Android, καθώς και ένα σύνολο βιβλιοθηκών και APIs (Application Programming Interfaces) που μπορούν να χρησιμοποιηθούν από τους προγραμματιστές για τη δημιουργία εφαρμογών κινητών συσκευών. Το Android Framework παρέχει στους προγραμματιστές πρόσβαση σε πόρους υλικού και λογισμικού της συσκευής, όπως η κάμερα, το GPS και οι αισθητήρες, καθώς και εργαλεία για το χειρισμό κοινών εργασιών όπως η αποθήκευση δεδομένων, η δικτύωση και η σχεδίαση διεπαφής χρήστη (UI).

Εικόνα 1: Λογότυπο λειτουργικού συστήματος Android

Σαν προγραμματιστικό περιβάλλον υλοποίησης/ανάπτυξης χρησιμοποιήθηκε το Android Studio, το οποίο είναι το επίσημο εργαλείο ανάπτυξης εφαρμογών Android της Google, το οποίο είναι χτισμένο πάνω στο περιβάλλον IntelliJ IDEA της εταιρίας JetBrains.

Εικόνα 2: Λογότυπο περιβάλλοντος ανάπτυξης εφαρμογών, Android Studio

Ως γλώσσες υλοποίησης της εφαρμογής χρησιμοποιήθηκε η γλώσσα προγραμματισμού, Kotlin.Ωστόσο πολλές αν όχι όλες οι συναρτήσεις του Android που χρησιμοποιούνται είναι γραμμένες σε

Java. Η Java είναι μια υψηλού επιπέδου, αντικειμενοστραφής γλώσσα προγραμματισμού η οποία σχεδιάστηκε από την εταιρεία πληροφορικής Sun Microsystems το 1995 και αργότερα πέρασε στην Oracle. Εξακολουθεί να είναι μια από τις πιο διαδεδομένες γλώσσες προγραμματισμού και αποτελεί βάση για πολλές νεότερες γλώσσες. Ο κώδικας Java επιδέχεται μεταγλώττιση σε μια μορφή κώδικα που ονομάζεται bytecode, ο οποίος μπορεί να εκτελεστεί ανεξαρτήτως αρχιτεκτονικής του συστήματος από την εικονική μηχανή της Java (Java Virtual Machine).**[5]**

Εικόνα 3: Λογότυπο γλώσσας προγραμματισμού Java

Η Kotlin είναι μια γλώσσα προγραμματισμού που χρησιμοποιείται σε μεγάλο εύρος εφαρμογών και συστημάτων (cross-platform) και είναι και αυτή μια από τις πιο διαδεδομένες γλώσσες προγραμματισμού. Παίρνει το όνομα της από το νησί Kotlin που βρίσκεται στην Βαλτική Θάλασσα κοντά στην Αγία Πετρούπολη της Ρωσίας. Έχει σχεδιαστεί από την JetBrains, όπως και το Android Studio, έτσι ώστε να δια λειτουργεί με την Java και την εικονική της μηχανή, Java Virtual Machine. Είναι συνεπώς βασισμένη στην Java αλλά και στην γλώσσα προγραμματισμού Scala. Μεταξύ άλλων φέρει χρήσης και σε web περιβάλλον αφού μπορεί να μεταγλωττιστεί και σε JavaScript. Η Kotlin παρεχόταν στο Android Studio από το 2017. Το 2019 η Google η οποία είναι η κυριά υπεύθυνη για το λειτουργικό σύστημα Android, ανακοίνωσε ότι η Kotlin θα αντικαταστήσει την Java ως την προτεινομένη γλώσσα προγραμματισμού Android εφαρμογών.**[6]**

Εικόνα 4: Λογότυπο γλώσσας προγραμματισμού Kotlin

Η εφαρμογή βασίστηκε στο ML Kit της Google και στην εφαρμογή ML Kit vision quickstart sample **[7]**. Το ML Kit είναι ένα SDK για κινητά που παρέχεται από την Google και επιτρέπει στους προγραμματιστές εφαρμογών για κινητές συσκευές να προσθέτουν εύκολα λειτουργίες μηχανικής εκμάθησης στις εφαρμογές τους για κινητά. Παρέχει ένα σύνολο προ εκπαιδευμένων μοντέλων για κοινές εργασίες όπως η αναγνώριση κειμένου, η επισήμανση εικόνων και η ανίχνευση προσώπου, καθώς και API για προσαρμοσμένα μοντέλα που μπορούν να εκπαιδευτούν χρησιμοποιώντας το TensorFlow Lite. Επιπλέον, παρέχει λειτουργίες όπως εκτέλεση μοντέλου στη συσκευή και αυτόματη ενημέρωση μοντέλου, καθιστώντας εύκολη την προσθήκη δυνατοτήτων μηχανικής εκμάθησης σε εφαρμογές για κινητά χωρίς να απαιτείται σύνδεση διακομιστή ή Διαδίκτυο.**[8]**

Εικόνα 5: Λογότυπο ML Kit SDK

1.5 Δομή εργασίας

Κεφάλαιο 1ο – Εισαγωγικό κεφάλαιο: Σε αυτό το κεφάλαιο παρέχεται μια γενική εισαγωγή για την διπλωματική εργασία όπως και οι στόχοι και οι τεχνολογίες που χρησιμοποιήθηκαν.

Κεφάλαιο 2ο – Ανάλυση και Σχεδίαση Εφαρμογής: Κεφάλαιο στο οποίο θα ξεκινήσουμε με αναφορά στην μεθοδολογία της εφαρμογής και τα επιμέρους στάδια της ανάλυσης και σχεδίασης της.

Κεφάλαιο 3ο – Παρουσίαση Εφαρμογής: Κεφάλαιο στο οποίο θα παρουσιάσουμε την εφαρμογή που φτιάξαμε ως προς την λειτουργία της και θα εξηγήσουμε τον τρόπο χρήσης της .

Κεφάλαιο 4ο – Εκπαίδευση μοντέλου: Κεφάλαιο στο οποίο θα κάνουμε αναφορά στην διαδικασία εκπαίδευσης, στο εργαλείο το οποίο κάναμε χρήση για την εκπαίδευση του μοντέλου της εφαρμογής όπως και την δομή του αρχείου της πληροφορίας.

Κεφάλαιο 5ο – Υλοποίηση Εφαρμογής: Κεφάλαιο στο οποίο θα παρουσιάσουμε την εφαρμογή που φτιάξαμε ως προς την δομή της και θα παρουσιάσουμε τον κώδικα της.

Κεφάλαιο 6ο – Επίλογος: Κεφάλαιο στο οποίο θα αναφερθούμε στα συμπεράσματα της εργασίας, στα προβλήματα που αντιμετωπίσαμε κατά την ανάπτυξη της εφαρμογής και στις πιθανές βελτιώσεις που μπορούμε να κάνουμε.

Κεφάλαιο 2ο: Ανάλυση & Σχεδίαση Εφαρμογής

2.1 Εισαγωγή

Στο κεφάλαιο αυτό θα γίνει μια επισκόπηση των βημάτων για την ανάλυση και σχεδίαση της εφαρμογής.

2.2 Ανάλυση και σχεδίαση κύριου στόχου

Ο κύριος στόχος της εφαρμογής είναι η έξυπνη μέτρηση των επαναλήψεων της άσκησης των έλξεων στο μονόζυγο. Ήτοι να μπορέσουμε να χρησιμοποιήσουμε ένα έξυπνο κινητό(smartphone) για μέτρηση των επαναλήψεων και να μην την κάνουμε εμείς (ο αθλούμενος). Ο παραπάνω στόχος θα μπορούσε να επιτευχθεί με χρήση ηλεκτρονικού εξοπλισμού ειδικού σκοπού, για παράδειγμα ένα ειδικά διαμορφωμένο κύκλωμα μικροελεγκτή ο οποίος διαθέτει αισθητήρες για την ανίχνευση της τοποθέτησης του σώματος σε σχέση με το μονόζυγο ή το έδαφος. Εναλλακτικά θα μπορούσε να γίνει χρήση των αισθητήρων κίνησης και τοποθέτησης του ιδίου του κινητού.

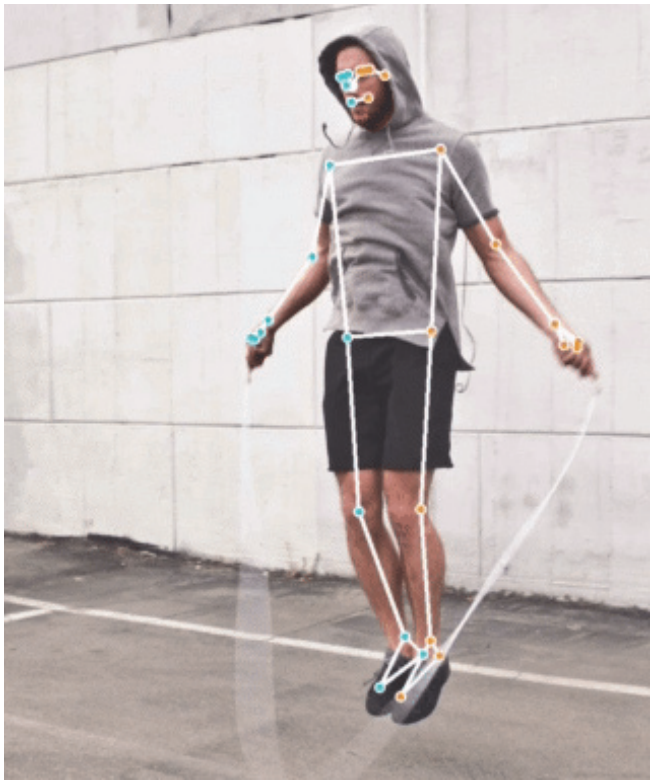
Τα παραπάνω όμως αποτελούν δαπανηρή ή και μερική λύση διότι με τις παραπάνω δεν μπορούμε να είμαστε βέβαιοι (η τουλάχιστον όσο βέβαιοι όσο θα επιθυμούσαμε) για την σωστή εκπλήρωση της άσκησης, επίσης δεν λαμβάνουν υπόψη τις ιδιομορφίες και την διαφορετικότητα του κάθε σώματος όπως πχ διαφορετικό ύψος χεριού, απόσταση ωμοπλάτης κλπ.

Με χρήση όμως της κάμερας του κινητού και ενός μοντέλου μηχανικής μάθησης θα μπορούσαμε να επιτύχουμε μια πιο ακριβή και ολοκληρωμένη λύση. Με τη χρήση ενός αλγορίθμου μηχανικής μάθησης που έχει εκπαιδευτεί σε εικόνες ανθρώπων να κάνουν την άσκηση, θα μπορούσαμε να αναγνωρίζουμε την κίνηση και τη θέση του σώματος του χρήστη και να μετρούμε τον αριθμό των επαναλήψεων της άσκησης με μεγαλύτερη ακρίβεια.

Επιπλέον, η χρήση ενός μοντέλου μηχανικής μάθησης προσφέρει τη δυνατότητα προσαρμογής του στις διαφορετικές ιδιομορφίες του κάθε σώματος και προσφέρει μεγαλύτερη ακρίβεια στη μέτρηση των επαναλήψεων. Δεδομένου ότι το μοντέλο έχει εκπαιδευτεί με διάφορα είδη σώματος. Η χρήση της κάμερας του κινητού σε συνδυασμό με ένα μοντέλο μηχανικής μάθησης για τη μέτρηση έλξεων προσφέρει μια ευέλικτη και οικονομικά αποδοτική λύση που μπορεί να βοηθήσει τους ανθρώπους να παρακολουθούν τους στόχους της φυσικής τους κατάστασης και να βελτιώσουν την απόδοσή τους. Όσον αφορά το κύριο στόχο της εφαρμογής η λύση επρόκειτο να δοθεί με χρήση του λογισμικού (SDK) ML Kit.

2.2.1 Δυνατότητες ML Kit

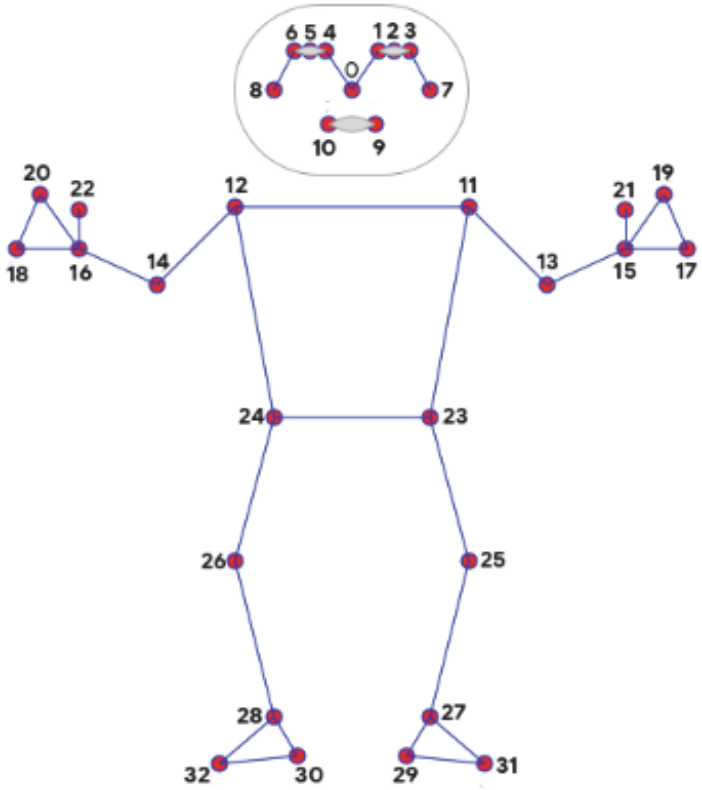
Το ML Kit μας δίνει την δυνατότητα χρησιμοποιώντας φωτογραφίες ή βίντεο να μπορέσουμε να εντοπίσουμε σε αυτά, σημεία κλειδιά του σώματος του ανθρώπου όπως τους αγκωνές, τους ωμούς, τα γόνατα κοκ.



Εικόνα 6: Αναγνώριση σκελετικών σημείων ορόσημων από το ML Kit

πηγή: https://developers.google.com/ml-kit/vision/pose-detection

Το ML Kit πραγματοποιεί ταίριασμα 33 σημαντικών σημείων/σημείων οροσήμων του σώματος τα οποία είναι αριθμημένα ξεκινώντας από το 0 που είναι η μύτη έως το 32 που είναι ο δείκτης του δεξιού ποδιού, το οποίο περιλαμβάνει τα κύρια χαρακτηριστικά σημεία του προσώπου (μάτια , στόμα, μύτη και αυτιά) και σημεία των ακρών του σώματος. Στο παρακάτω σχήμα είναι μια κατοπτρική εικόνα στην οποία φαίνονται τα σημεία ορόσημα, η δεξιά πλευρά του χρήστη εμφανίζεται στα αριστερά της εικόνας όπως δηλαδή θα φαινόταν ένας χρήστης μέσα από κάμερα.**[9]**



Εικόνα 7: Σχεδιάγραμμα σημείων ορόσημων

πηγή: https://developers.google.com/ml-kit/vision/pose-detection

Παρακάτω είναι η αντιστοίχιση των σημείων του σώματος με την αρίθμηση της εικόνας 7:

0. Μύτη
1. Αριστερό μάτι (εσωτερικό)
2. Αριστερό μάτι
3. Αριστερό μάτι (εξωτερικό)
4. Δεξί μάτι (εσωτερικό)
5. Δεξί μάτι
6. Δεξί μάτι (εξωτερικό)
7. Αριστερό αυτί
8. Δεξί αυτί
9. Στόμα (αριστερό σημείο)
10. Στόμα (δεξιό σημείο)
11. Αριστερός ώμος
12. Δεξιός ώμος

13. Αριστερός αγκώνας
14. Δεξιός αγκώνας
15. Αριστερός καρπός
16. Δεξιός καρπός
17. Αριστερό μικρό δάχτυλο
18. Δεξιό μικρό δάχτυλο
19. Αριστερός δείκτης
20. Δεξιός δείκτης
21. Αριστερός αντίχειρας
22. Δεξιός αντίχειρας
23. Αριστερό ισχίο
24. Δεξί ισχίο
25. Αριστερό γόνατο
26. Δεξί γόνατο
27. Αριστερός αστράγαλος
28. Δεξιός αστράγαλος
29. Αριστερή φτέρνα
30. Δεξιά φτέρνα
31. Δείκτης αριστερού ποδιού
32. Δείκτης δεξιού ποδιού

Το κάθε ένα από τα παραπάνω σημεία παρέχουν πολλές πληροφορίες όπως την θέση στον x, την θέση στον y και την θέση στον z άξονα. Το τρίτο βοηθά στον υπολογισμού βάθους και υπολογίζεται σε σχέση με τους γοφούς του χρήστη και την κάμερα. Οι θέσεις αυτές έχουν και την πληροφορία **InFrameLikelihood** που μας φανερώνει την υπολογισμένη πιθανότητα για το συγκεκριμένο σημείο ενδιαφέροντος να βρίσκεται πράγματι μέσα στην εικόνα που ελέγχθηκε. Εάν κάποιο σημείο δεν βρεθεί εντός πλαισίου θεωρείται εκτός πλαισίου και παίρνει τιμές έξω από το πλαίσιο.

Το SDK δεν μπορεί να ανιχνεύσει παραπάνω από ένα άτομο σε μια εικόνα και αν ανιχνεύσει περισσότερα απλά επιστρέφει τα σημεία του ατόμου με την μεγαλύτερη αξιοπιστία ανίχνευσης.**[10]**

Συγκρίνοντας και κατηγοριοποιώντας τα παραπάνω προκαθορισμένα σημεία – ορόσημα του σώματος του χρήστη τα οποία μας επιστρέφει το SDK με κάποιες αποδέκτες θέσεις τις οποίες γνωρίζουμε μέσω ενός εκπαιδευμένου μοντέλου, μπορούμε κατά την εκτέλεση της άσκησης των έλξεων, να βγάλουμε το συμπέρασμα για το αν έχει πραγματοποιηθεί μια πλήρης επανάληψη. Μια επανάληψη έλξης μπορεί να

θεωρηθεί εκπληρωμένη όταν το σώμα του αθλητή έχει εισέλθει στην ζητούμενη πόζα/τοποθέτηση και έχει εξέλθει από αυτήν.

Εικόνα 8: Χαμηλότερο (αριστερά) και υψηλότερο (δεξιά) σημείο εκτέλεσης έλξης

Το ML Kit για χρήση της ανίχνευσης του ανθρωπίνου σώματος μας παρέχει
δυο είδη λειτουργίας:

- Την γρήγορη λειτουργία (**fast**) για περιπτώσεις που η ταχύτητα της ανίχνευσης παίζει σημαντικό ρόλο στον στόχο της εφαρμογής.
- Την ακριβή λειτουργία (**accurate**) ανίχνευσης για περιπτώσεις οπού η ταχύτητα της ανίχνευσης δεν είναι σημαντικός παράγοντας.**[11]**

Για τους στόχους της εφαρμογής μας, θα χρειαστεί να ‘‘θυσιάσουμε’’ λίγη ακρίβεια προκειμένου η ανίχνευση να είναι όσο γρηγορότερη γίνεται μιας και θέλουμε να μετράμε ζωντανά τα πλαίσια (frames) που μας δίνει η κάμερα του κινητού. Να ανιχνεύουμε δηλαδή σε αυτά μια στάση του σώματος να κατηγοριοποιούμε την ανιχνευμένη στάση ανάλογα με την γνωσιακή βάση που έχουμε και να μετράμε τις επαναλήψεις.

Θα πρέπει λοιπόν να μπορεί το πρόγραμμα να ανιχνεύσει πότε ο αθλητής βρίσκεται σε μια αποδεκτή θέση ως προς την άσκηση στόχο.
Το πρόβλημα που προκύπτει είναι πως θα ορίσουμε ποια είναι αποδεκτή θέση.

2.2.2 Μάθηση με επίβλεψη

Η λύση εδώ θα βρεθεί με χρήση της μηχανικής μάθησης και συγκεκριμένα της μάθησης με επίβλεψη ή αλλιώς μάθησης με παραδείγματα.

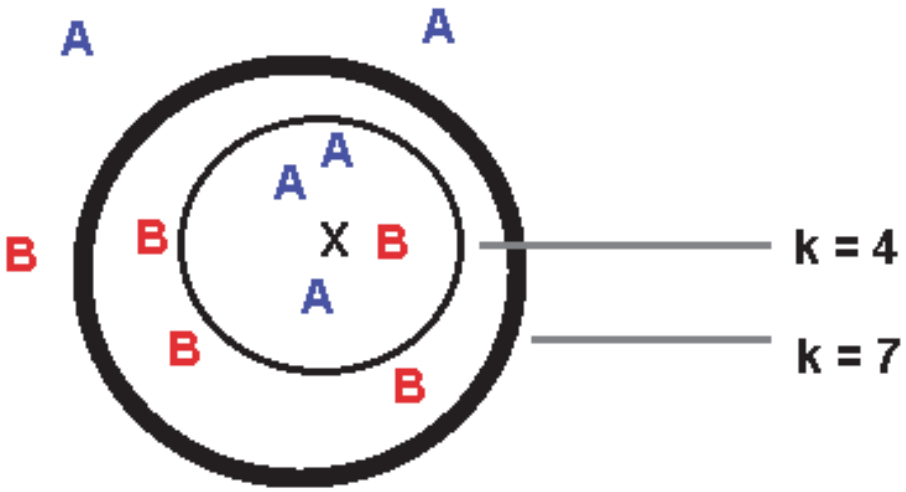
Ανάλογα με το πρόβλημα που έχουμε να λύσουμε, στην μάθηση με επίβλεψη, χρησιμοποιούμε και την κατάλληλη μεθοδολογία. Στην μάθηση με επίβλεψη συναντάμε δυο είδη προβλημάτων, την ταξινόμηση ή κατηγοριοποίηση και την παρεμβολή. **[3]**

Στην ταξινόμηση έχουμε ως στόχο να δημιουργήσουμε μοντέλο πρόβλεψης διακριτών κατηγοριών. Ενώ στην παρεμβολή θέλουμε να προβλέψουμε, μέσω του μοντέλου που δημιουργήσαμε, κάποια αριθμητική τιμή. Στο εγχείρημα μας θέλουμε να μπορέσουμε να ξεχωρίσουμε την πάνω θέση και την κάτω θέση των έλξεων. Έχουμε συνεπώς ένα πρόβλημα ταξινόμησης δυο κατηγοριών (πάνω θέση έλξης, κάτω θέση έλξης). Αρχικά, θα χρειαστεί λοιπόν να κάνουμε μια κατηγοριοποίηση ώστε να μπορούμε να διακρίνουμε την πάνω θέση και την κάτω θέση της κίνησης των έλξεων.

Στην μάθηση κατά περίπτωση (instance - based learning) η οποία είναι μια μέθοδος μηχανικής μάθησης, τα δεδομένα εκπαίδευσης διατηρούνται αυτούσια. Ένα τέτοιο σύστημα καλείται να αποφασίσει για την κατηγορία μιας νέας περίπτωσης και θα εξετάσει εκείνη την στιγμή τη σχέση της νέας περίπτωσης με τα αποθηκευμένα παραδείγματα που έχει. Αυτή η μέθοδος έχει χαρακτηριστεί ως αναβλητική μάθηση (lazy learning) σε αντίθεση με τις άλλες μεθόδους οι οποίες μαθαίνουν το μοντέλο, χωρίς να περιμένουν την άφιξη νέας περίπτωσης, από τα παραδείγματα του συνόλου εκπαίδευσης και ονομάζονται έγκαιρες (eager learners).

Ο αλγόριθμος των k - πλησιέστερων γειτόνων ή αλλιώς αλγόριθμος k -nn (k-Nearest Neighbors) είναι γνωστός αλγόριθμος αυτής της κατηγορίας. Στον αλγόριθμο αυτό θεωρούμε ότι τα παραδείγματα μπορούν να αναπαρασταθούν ως σημεία σε κάποιον n-διάστατο Ευκλείδειο χώρο Rn οπού n ο αριθμός των χαρακτηριστικών (ανεξάρτητων μεταβλητών) . Κάθε νέα περίπτωση τοποθετείται στον χώρο ως ένα καινούργιο σημείο και προσδιορίζεται η κατηγορία του με βάση την κατηγορία τον k πλησιέστερων γειτόνων. **[3]**

Παρακάτω βλέπουμε ένα απλό παράδειγμα κατηγοριοποίησης μιας νέας περίπτωσης X μεταξύ των κατηγοριών A και B. Αν ληφθούν υπόψη οι 4 πλησιέστεροι γείτονες 4-nn δηλαδή για k = 4 τότε τα σημεία μέσα στον εσωτερικό λεπτό κύκλο το X θα κατηγοριοποιηθεί ως A ενώ αν λάβουμε υπόψη τους 7 πλησιέστερους γείτονες 7-nn δηλαδή για k =7 αρά όλα τα σημεία εντός τους εξωτερικού παχύ κύκλου τότε το X κατηγοριοποιείται ως B.



Εικόνα 9: Λειτουργία αλγορίθμου k-nn

2.3 Ανάλυση και σχεδίαση δευτερευόντων στόχων

Σαν δευτερεύοντες στόχοι τέθηκαν ζητήματα όπως:

- Αποθήκευση ιστορικού των επαναλήψεων της εφαρμογής ώστε ο χρήστης να μπορεί ανά πάσα στιγμή να ανατρέξει στην σελίδα ιστορικού και να δει την πρόοδο του σε προηγούμενες ημέρες και στατιστικά όπως το μέγιστο των επαναλήψεων που έχει επιτύχει σε ένα set (maximum), τις συνολικές επαναλήψεις που έχει επιτύχει καθ' όλη την διάρκεια από την ημέρα που ξεκίνησε να κάνει χρήση της εφαρμογής (total) και άλλα στατιστικά.
- Ζητήματα διεπαφής χρήστη (User Interface) για εύκολη πλοήγηση μέσα στην εφαρμογή, εύκολη και συνεπή χρήση.

Σε μια εφαρμογή βοηθό εκγύμνασης θα ήταν ιδιαίτερα βοηθητικό να μπορούμε να αποθηκεύουμε χρήσιμες πληροφορίες όπως το πότε ο χρήστης εκτέλεσε την άσκηση όπως και τον αριθμό των επαναλήψεων αλλά και άλλα συμπληρωματικά στατιστικά στοιχεία. Για την αποθήκευση ιστορικού των επαναλήψεων της εφαρμογής ώστε ο χρήστης να μπορεί ανά πάσα στιγμή να ανατρέξει στην σελίδα ιστορικού και να δει την πρόοδο του σε προηγούμενες ημέρες όπως και κάποια στατιστικά στοιχεία χρησιμοποιήθηκε η βιβλιοθήκη Realm για διαχείριση μιας τοπικής βάσης στο κινητό.

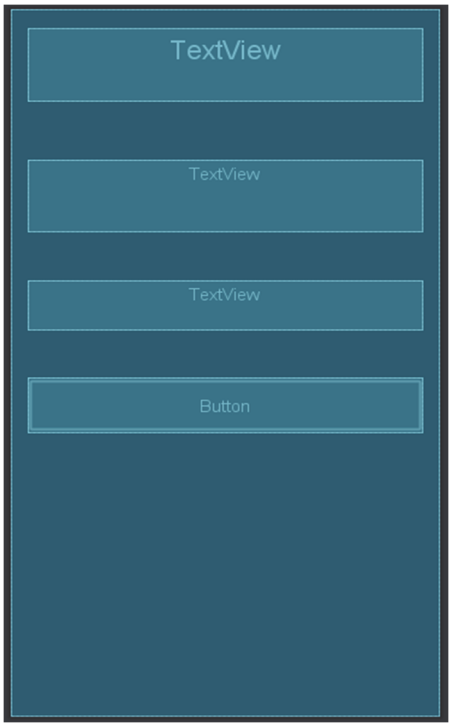
Εικόνα 10: Λογότυπο Realm

Η Realm είναι μια μη σχεσιακή, NoSQL βάση δεδομένων για φορητές έξυπνες συσκευές που έχει σχεδιαστεί για να παρέχει μια εναλλακτική λύση στην βάση δεδομένων SQLite του Android οικοσυστήματος. Είναι μια βάση δεδομένων πολλαπλών πλατφορμών που προοριζόταν αρχικά για χρήση σε πλατφόρμες Android και iOS αλλά έπειτα έγινε διαθέσιμο και για πλατφόρμες όπως το Xamarin, React Native και άλλες συμπεριλαμβανομένων των desktop εφαρμογών. Οι βάσεις δεδομένων της Realm έχουν σχεδιαστεί για να είναι πιο γρήγορες και πιο αποτελεσματικές από τις παραδοσιακές βάσεις δεδομένων SQLite, με τη δυνατότητα να εκτελούν ταχύτερα queries και transactions. **[12]**.

Είναι ένα έργο ανοιχτού κώδικα που μπορεί να χρησιμοποιηθεί ελεύθερα με την άδεια Apache 2.0. Παρέχει ένα απλό API για να χρησιμοποιηθεί από προγραμματιστές, καθιστώντας εύκολη την ενσωμάτωση με υπάρχουσες βάσεις κώδικα. Οι προγραμματιστές μπορούν να χρησιμοποιήσουν την Realm για να αποθηκεύσουν δομημένα δεδομένα σε μια τοπική βάση δεδομένων στη συσκευή, επιτρέποντας ταχύτερες και πιο αποκριτικές εφαρμογές. Η Realm είναι ένα χρήσιμο εργαλείο για προγραμματιστές Android που αναζητούν έναν γρήγορο και αποτελεσματικό τρόπο αποθήκευσης δεδομένων τοπικά σε κινητές συσκευές.

Για την διεπαφή χρήστη θα χρησιμοποιήσουμε το κλασικό σύστημα του Android, το View System. Το View System χρησιμοποιεί την κλάση View η οποία αποτελεί το βασικό δομικό στοιχείο για όλα τα στοιχεία διεπαφής χρήστη. Το σύστημα αυτό περιλαμβάνει ένα σύνολο στοιχείων διεπαφής χρήστη, όπως TextViews, EditTexts, Buttons και άλλα πιο σύνθετα όπως RecyclerViews, ListViews, CardViews, τα οποία μπορούν να οργανωθούν μέσα σε μια ιεραρχία για να δημιουργήσουν την επιθυμητή διάταξη (layout). Τα παραπάνω στοιχεία είναι όλα επέκταση της κλάσης View. Μπορούμε ακόμα να φτιάξουμε και δικά μας custom views κάνοντας επέκταση από την κλάση View ή από κάποια άλλη κλάση. Μπορούμε να δημιουργήσουμε διεπαφή χρήστη χρησιμοποιώντας κώδικα XML ή κώδικα Java/Kotlin. Παρέχει ευελιξία και έλεγχο στη σχεδίαση Το View System είναι μια αξιόπιστη και αποδεδειγμένη προσέγγιση για τη δημιουργία διεπαφής χρήστη σε εφαρμογές Android. Μια νεότερη τεχνολογία δημιουργίας διεπαφών χρήστη είναι το Jetpack Compose.

Στην παρακάτω εικόνα μπορούμε να δούμε μια απλή διάταξη βασικών στοιχείων διεπαφής χρήστη όπως TextViews και Button όπως φαίνεται στην καρτέλα Design του Android Studio.



Εικόνα 11: Διάταξη στοιχείων διεπαφής χρήστη

Όσον αφορά την πλοήγηση της εφαρμογής, θα ενσωματωθεί ένα **Navigation Drawer**. Επιτρέπει στους χρήστες να εναλλάσσονται μεταξύ της αρχικής οθόνης και της οθόνης Ιστορικού, προσφέροντας αρκετά πλεονεκτήματα, ένα από αυτά είναι το οικείο μοτίβο πλοήγησης. Εφαρμόζοντάς αυτό, οι χρήστες μπορούν να έχουν πρόσβαση σε διαφορετικές οθόνες με ευκολία, καθώς είναι συνηθισμένοι σε αυτό το ευρέως χρησιμοποιούμενο τρόπο πλοήγησης σε πολλές δημοφιλείς εφαρμογές.

Επιπλέον, εφαρμοστήκαν έλεγχοι για τα δικαιώματα χρήσης της κάμερας εντός της εφαρμογής. Εάν ο χρήστης αρνηθεί την πρόσβαση στην κάμερα, θα ανακατευθυνθεί σε μια οθόνη που εξηγεί τον λόγο για τον οποίο είναι απαραίτητη η άδεια της κάμερας για τη λειτουργικότητα της εφαρμογής. Σε αυτήν την οθόνη, ο χρήστης καλείται να πατήσει το κουμπί ρυθμίσεων, το οποίο ανοίγει τις ρυθμίσεις της εφαρμογής απευθείας. Καθοδηγώντας τον χρήστη στις ρυθμίσεις της εφαρμογής, διασφαλίζουμε τη διαφάνεια και του παρέχουμε την ευκαιρία να ελέγξει και να αλλάξει την απόφασή του σχετικά με την πρόσβαση στην κάμερα. Μόλις ο χρήστης επιστρέψει στην εφαρμογή μετά την επιτυχή αποδοχή των αδειών της κάμερας, η κάμερα θα ανοίξει αυτόματα, επιτρέποντάς του να συνεχίσει απρόσκοπτα τη χρήση της λειτουργίας εισαγωγής κάμερας της εφαρμογής. Εξηγώντας τη σημασία των αδειών της κάμερας σε μια ξεχωριστή οθόνη, ο χρήστης καταλαβαίνει γιατί είναι απαραίτητη η παραχώρηση πρόσβασης. Αυτό ενθαρρύνει τους χρήστες να χορηγήσουν τα απαιτούμενα δικαιώματα.

Τέλος, το αυτόματο άνοιγμα της κάμερας μετά την επιτυχή αποδοχή των αδειών βελτιώνει την εμπειρία του χρήστη. Οι χρήστες δεν χρειάζεται να επιστρέψουν με μη αυτόματο τρόπο στη λειτουργία της κάμερας.

η εφαρμογή επανέρχεται απρόσκοπτα στην επιθυμητή λειτουργικότητα, παρέχοντας μια ομαλή και αποτελεσματική εμπειρία. Κατά αυτόν τον τρόπο αυτή η εφαρμογή ευθυγραμμίζεται με τις πρακτικές του Android για το χειρισμό των αδειών. Ακολουθώντας αυτές τις οδηγίες, διασφαλίζουμε ότι η εφαρμογή μας πληροί τα πρότυπα που ορίζει η πλατφόρμα Android και διατηρεί συμβατότητα με μελλοντικές ενημερώσεις.

Κεφάλαιο 3ο: Παρουσίαση Εφαρμογής

3.1 Εισαγωγή

Στο κεφάλαιο αυτό θα γίνει μια παρουσίαση για την εφαρμογή που δημιουργήθηκε και ανάλυση του τρόπου λειτουργίας της. Η εφαρμογή έχει πολλαπλά σενάρια χρήσης και θα δούμε κάθε ένα από αυτά.

3.2 Γενική περιγραφή

Ένα από τα σημαντικότερα στοιχεία στον προγραμματισμό μιας εφαρμογής Android είναι η κλάση Activity (δραστηριότητα). Η κλάση αυτή αποτελεί μια οθόνη οπού ο χρήστης μπορεί να δει και να αλληλοεπιδράσει με διάφορα στοιχεία όπως πεδία εισόδου κειμένου, κουμπιά, εικόνες κλπ. Συνήθως μια Activity είναι το σημείο εισόδου στην εφαρμογή και ονομάζεται MainActivity. Τα Activities έχουν το δικό τους κύκλο ζωής (lifecycle) και περιέχει μεθόδους όπως onCreate() , onResume(), onDestroy() κ.α. Μέσα από ένα Activity μπορούμε να μετακινηθούμε σε άλλο Activity δημιουργώντας έτσι την ροή μιας εφαρμογής.

Μια άλλη σημαντική κλάση είναι η κλάση Fragment (κομμάτι). Η κλάση αυτή αναπαριστά ένα κομμάτι σε ένα Activity το οποίο είναι επαναχρησιμοποιούμενο. Ένα Activity μπορεί να περιέχει μέσα του πολλά Fragments. Ένα Fragment έχει επίσης κύκλο ζωής (lifecycle).
Στην εφαρμογή μας υπάρχουν οι εξής οθόνες :
Κλάσεις Activities:

- MainActivity (φιλοξενεί το HomeFragment και το HistoryFragment)
- CameraActivity (φιλοξενεί το CameraFragment)
- CameraAccessActivity
- CorrectResultActivity

Και τα ακόλουθα fragments:

- CameraFragment (φιλοξενείται στο CameraActivity)
- HomeFragment (φιλοξενείται στο MainActivity)
- HistoryFragment (φιλοξενείται στο MainActivity)

Το κύριο σημείο εισόδου στην εφαρμογή είναι το MainActivity. Φιλοξενεί δύο fragments: το HomeFragment και το HistoryFragment. Από το HomeFragment, ο χρήστης μπορεί να πατήσει ένα κουμπί με την ένδειξη "START" και θα του ζητηθεί να παραχωρήσει πρόσβαση στην κάμερα. Εάν ο χρήστης παραχωρήσει πρόσβαση, μπορεί να προχωρήσει στο CameraActivity, όπου τα pull-ups εντοπίζονται, ταξινομούνται και καταμετρώνται. Εάν δεν παραχωρηθεί πρόσβαση στην κάμερα, ο χρήστης θα

κατευθυνθεί στο CameraAccessActivity, το οποίο εξηγεί στον χρήστη γιατί απαιτείται πρόσβαση στην κάμερα και παρέχει ένα κουμπί για απευθείας μετάβαση στις ρυθμίσεις της συσκευής. Αφού παραχωρήσει επιτυχώς πρόσβαση στην κάμερα από τις ρυθμίσεις της συσκευής και επιστρέψει στο CameraAccessActivity, ο χρήστης θα ανακατευθυνθεί αυτόματα στο CameraActivity.

Από το HomeFragment, ο χρήστης μπορεί επίσης να πλοηγηθεί στο HistoryFragment χρησιμοποιώντας το συρτάρι πλοήγησης(Navigation Drawer) στην αριστερή πλευρά της οθόνης, στο οποίο μπορεί να χρησιμοποιηθεί κάνοντας κίνηση swipe ή πατώντας στο hamburger button (τρεις παράλληλες οριζόντιες γραμμές) που υπάρχει πάνω αριστερά. Το συρτάρι πλοήγησης περιέχει τα δύο fragments ως προορισμούς (HomeFragment, HistoryFragment) και ο χρήστης μπορεί να πατήσει σε καθένα από αυτά.

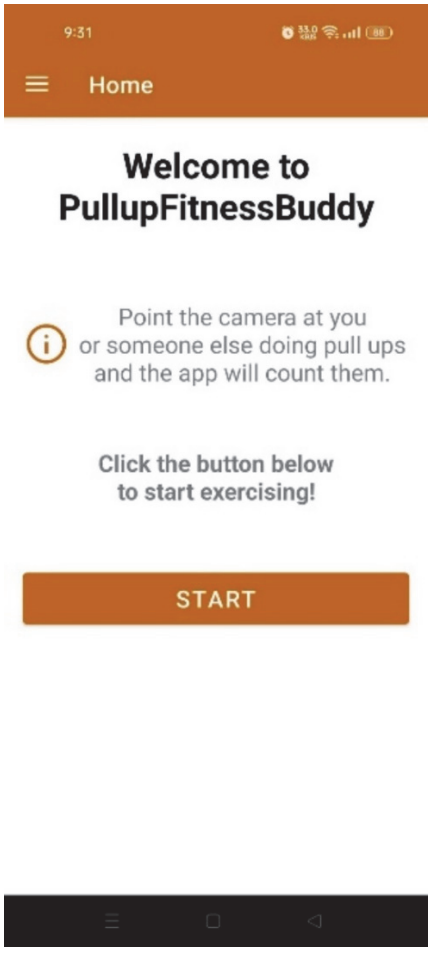
Στο HistoryFragment, ο χρήστης μπορεί να βρει μια λίστα φορτωμένη από μια τοπική βάση δεδομένων, στην οποία εμφανίζονται επαναλήψεις και σετ, καθώς και στατιστικά στοιχεία όπως μέγιστες επαναλήψεις και συνολικές επαναλήψεις. Από το HistoryFragment, ο χρήστης μπορεί να προηγηθεί πίσω στο HomeFragment μόνο χρησιμοποιώντας το συρτάρι πλοήγησης.

Από το CameraActivity, όταν ο χρήστης πατήσει το κουμπί "DONE", κατευθύνεται στο CorrectResultActivity. Εδώ, ο χρήστης μπορεί να ελέγξει τον αριθμό των pull-ups που καταμετρήθηκαν από την εφαρμογή και έχει τη δυνατότητα να επεξεργαστεί το πλήθος για να υπολογίσει ψευδώς θετικά ή ψευδώς αρνητικά αποτελέσματα. Υπάρχει επίσης ένα κουμπί κοινής χρήσης (Share) το οποίο, όταν πατηθεί, πραγματοποίει ένα αίτημα στο λειτουργικό σύστημα για μια εφαρμογή που μπορεί να δεχτεί το κείμενο: "I just did X pull-ups with the help of PullupFitnessBuddy", επιτρέποντας στον χρήστη να το μοιραστεί σε άλλες εφαρμογές. Η κύρια ενέργεια στο CorrectResultActivity είναι το κουμπί "SAVE", το οποίο, όταν πατηθεί, αποθηκεύει τις μετρημένες επαναλήψεις στην τοπική βάση δεδομένων και ανακατευθύνει τον χρήστη στο HomeFragment.

3.3 Παρουσίαση με στιγμιότυπα οθονών

Ξεκινάμε με το MainActivity που εμφανίζει το HomeFragment.

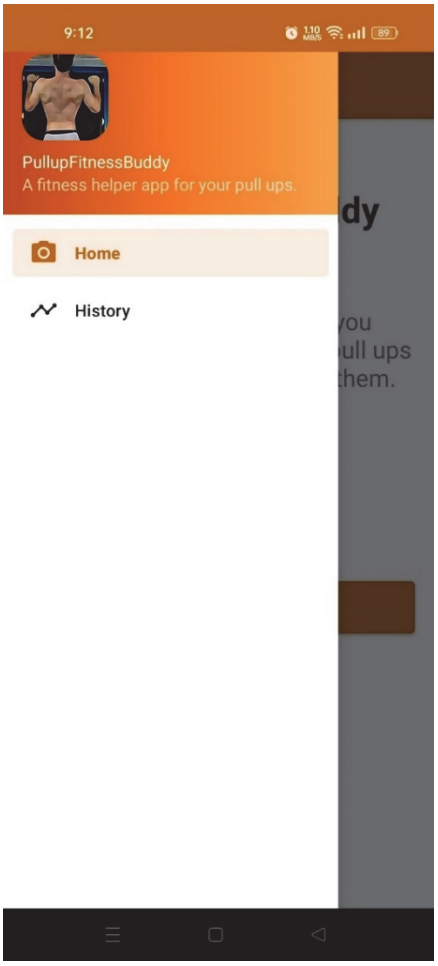
MainActivity (HomeFragment):



Εικόνα 12: Αρχική οθόνη

Σε αυτήν την οθόνη μπορούμε να δούμε ένα μήνυμα που μας καλωσορίζει, ένα σύντομο επεξηγηματικό μήνυμα το κουμπί START και το hamburger button που μας ανοίγει το Navigation Drawer.

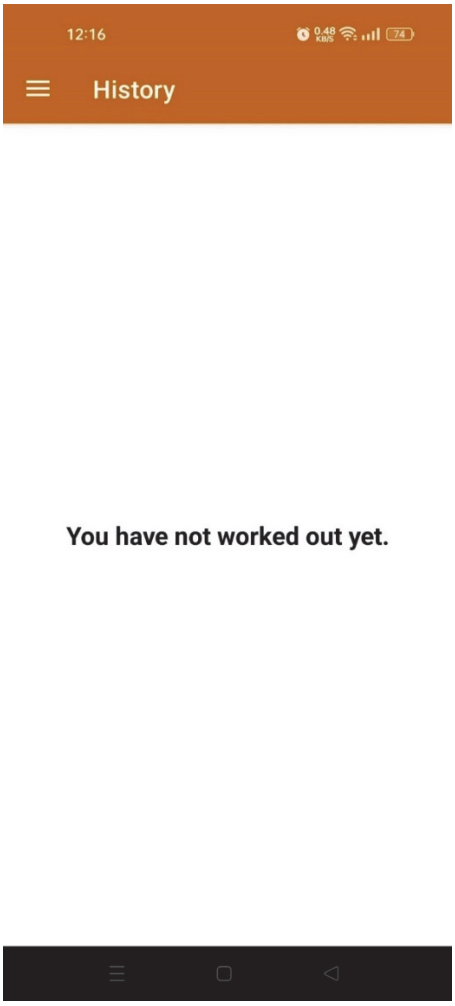
Navigation Drawer:



Εικόνα 13: Navigation Drawer

Αν σε αυτό το σημείο πατήσουμε το History θα κατευθυνθούμε στο HistoryFragment όμως επειδή είναι η πρώτη φορά χρήσης της εφαρμογής η τοπική βάση δεδομένων μας είναι άδεια θα δούμε ενημερωτικό μήνυμα για αυτήν την περίπτωση.

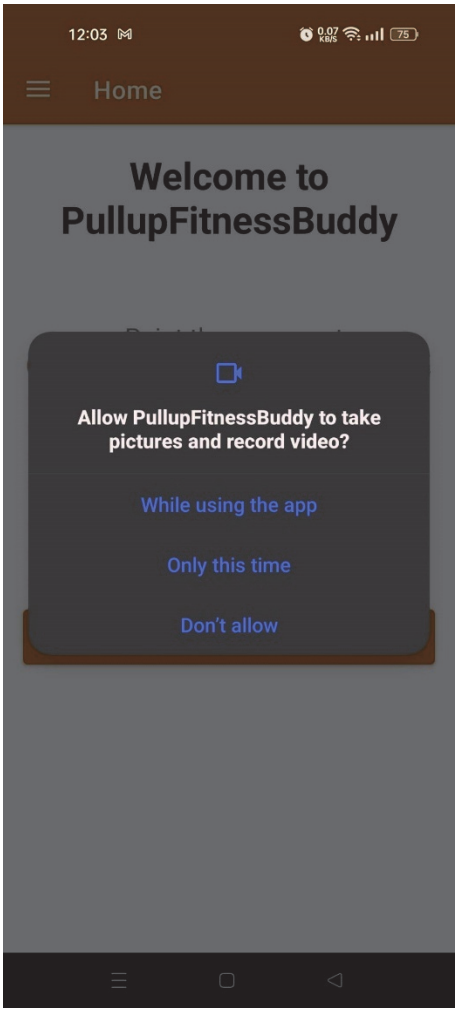
HistoryFragment με κενή τοπική βάση δεδομένων:



Εικόνα 14: Οθόνη ιστορικού (κενή)

Αν μεταφερθούμε πίσω στο HomeFragment και πατήσουμε το κουμπί START θα εμφανιστεί το παράθυρο του λειτουργικού συστήματος για να ζητήσει από τον χρήστη άδεια για χρήση της κάμερας.

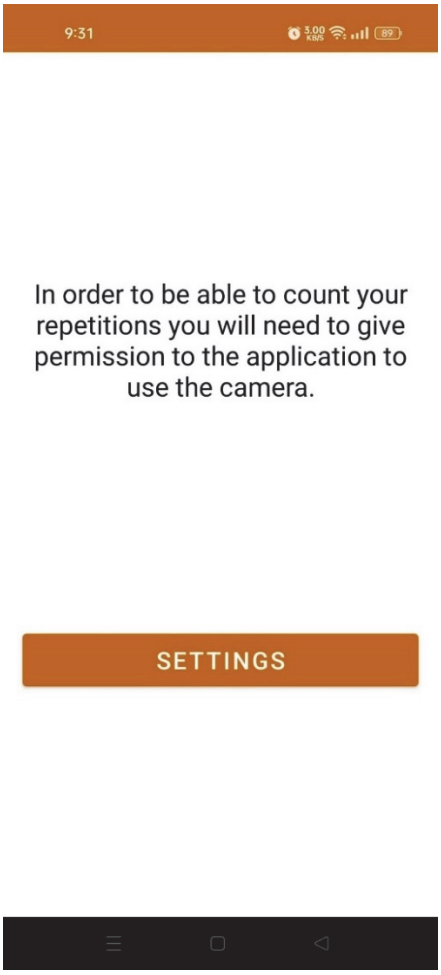
Αδειοδότηση κάμερας:



Εικόνα 15: Παράθυρο λειτουργικού συστήματος

Σε αυτό το σημείο αν δώσουμε θετική απάντηση θα κατευθυνθούμε στην οθόνη CameraActivity.Εάν αρνηθούμε (Don’t allow) θα μεταφερθούμε στην οθόνη CameraAccessActivity.

CameraAccessActivity:



Εικόνα 16: Οθόνη άδειας κάμερας

Η οθόνη αυτή είναι απαραίτητη διότι αν για κάποιο λόγο ο χρήστης δεν αποδεχθεί πολλαπλές φορές ή αφαιρέσει την άδεια της κάμερας από τις ρυθμίσεις της συσκευής το λειτουργικό σύστημα, θα τον αποτρέψει αυτόματα από το να συνεχίσει την ροή της εφαρμογής. Οπότε με αυτόν τον τρόπο του δίνουμε την οδηγία πως να ενεργοποιήσει πάλι την κάμερα, τον λόγο που χρειάζεται και του παρέχουμε εύκολο τρόπο για να δώσει την άδεια. Πατώντας το κουμπί SETTINGS μεταβιβάζεται στις ρυθμίσεις της συσκευής για την εφαρμογή μας. Όταν δώσει άδεια με επιτυχία και επιστρέψει πίσω η εφαρμογή θα το αντιληφθεί αυτόματα με σωστή χρήση του lifecycle του Activity. Πιο συγκεκριμένα, μέσα στην μέθοδο onResume() θα γίνει έλεγχος αν τελικά έχει δοθεί η άδεια για την κάμερα. Σε επιτυχημένη λήψη αδείας θα μεταβεί αυτόματα στην οθόνη CameraActivity.

Στην CameraActivity που φαίνεται στην Εικόνα 17 παρακάτω φτάνουμε πάραυτα και την πρώτη φορά με θετική απάντηση αλλά και τις επόμενες φορές όπου δεν θα μας ξαναζητηθεί.

CameraActivity (CameraFragment):

Εικόνα 17: Ανίχνευση άσκησης (κάτω θέση)

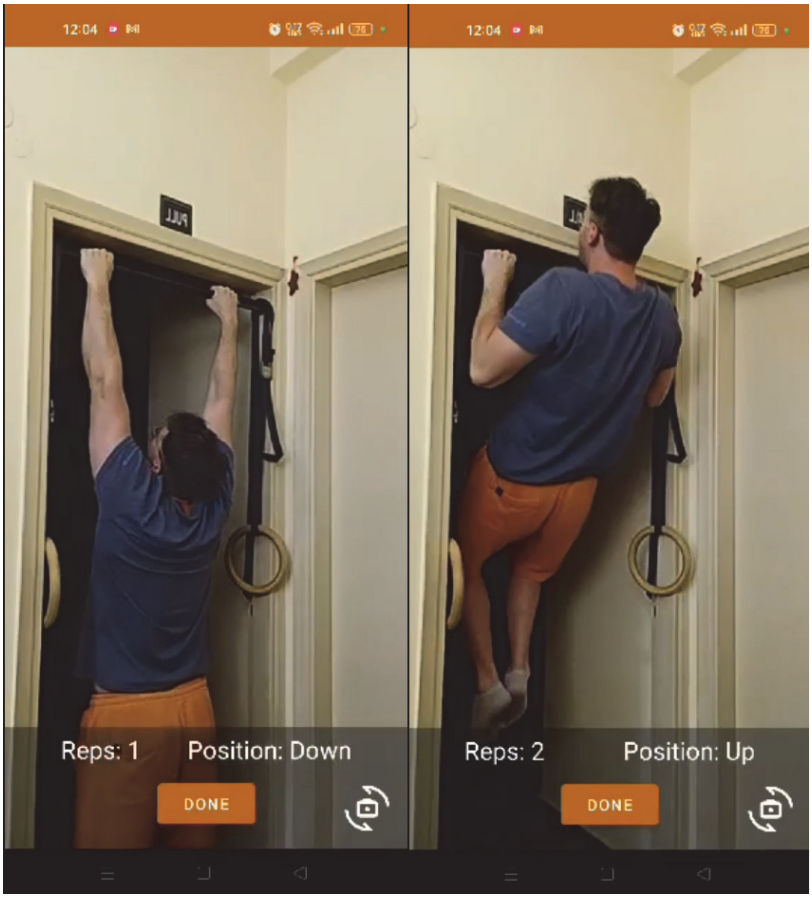
Στο προηγούμενο στιγμιότυπο οθόνης βλέπουμε την κύρια λειτουργία της εφαρμογής. Η κάμερα διαβάζει ζωντανά εικόνες και ανιχνεύει την τοποθέτηση σε αυτές, κατηγοριοποιεί και τέλος μετράει τις επαναλήψεις. Κάτω δεξιά βλέπουμε τις επαναλήψεις που έχουν μετρηθεί μέχρι τώρα και κάτω δεξιά την τωρινά ανιχνευμένη θέση στην έλξη. Αυτήν την στιγμή ο αθλούμενος φαίνεται να βρίσκεται στην κάτω θέση της έλξης οπότε και εμφανίζεται η ένδειξη Down.Υπάρχουν δυο κουμπιά, το κουμπί DONE και το κουμπί εναλλαγής κάμερας. Το πρώτο μας οδηγεί στην οθόνη CorrectResultActivity την οποία θα δούμε αργότερα. Το κουμπί εναλλαγής κάμερας, αλλάζει την επιλεγμένη κάμερα της συσκευής από την πίσω, στην μπροστά και το αντίθετο.
Συνεχίζοντας θα δούμε τον αθλούμενο να βρίσκεται στην πάνω θέση.

Ανίχνευση άνω θέσης:

Εικόνα 18: Ανίχνευση άσκησης (άνω θέση)

Στην προηγουμένη εικόνα βλέπουμε ο αθλούμενος να βρίσκεται στην άνω θέση της έλξης, το Position να είναι **Up** και το νούμερο των έλξεων να έχει γίνει 1 (ένα).

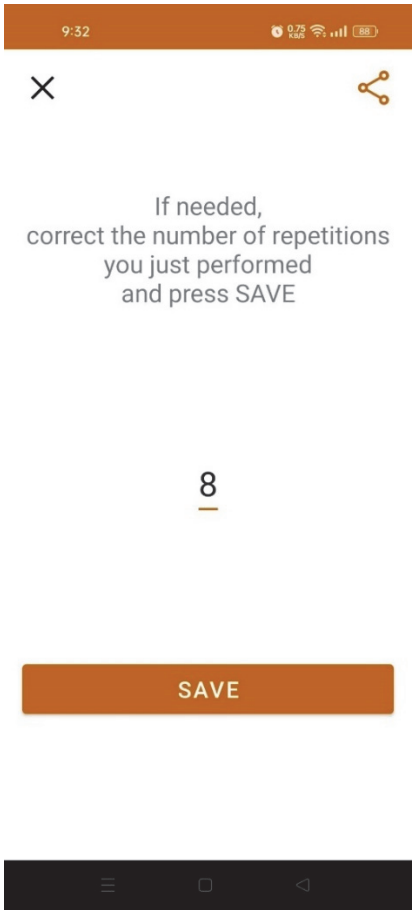
Σύνοψη άσκησης:



Εικόνα 19: Πραγματοποίηση εκγύμνασης με μέτρηση

Στην παραπάνω εικόνα βλέπουμε την λειτουργία της εφαρμογής όπου από την επανάληψη 2 θα μετάβουμε στην επανάληψη 3 κοκ. Όταν ο χρήστης πατήσει το κουμπί **DONE** θα μεταφερθεί στην επόμενη οθόνη που είναι η CorrectResultActivity.

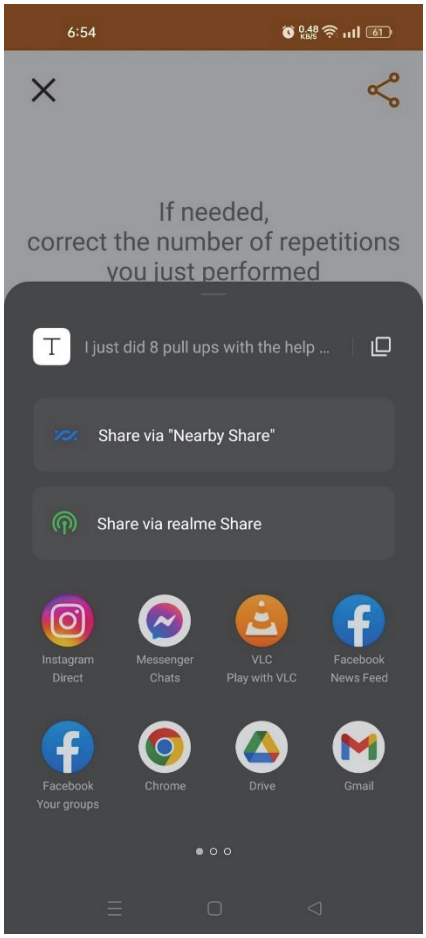
CorrectResultActivity:



Εικόνα 20: CorrectResultActivity

Σε αυτήν την οθόνη έχει την ευκαιρία να κάνει **Share** (να διαμοιραστεί) το αποτέλεσμα του σε άλλες εφαρμογές πατώντας το κουμπί πάνω δεξιά.

Παράθυρο διαμοιρασμού από το λειτουργικό σύστημα:

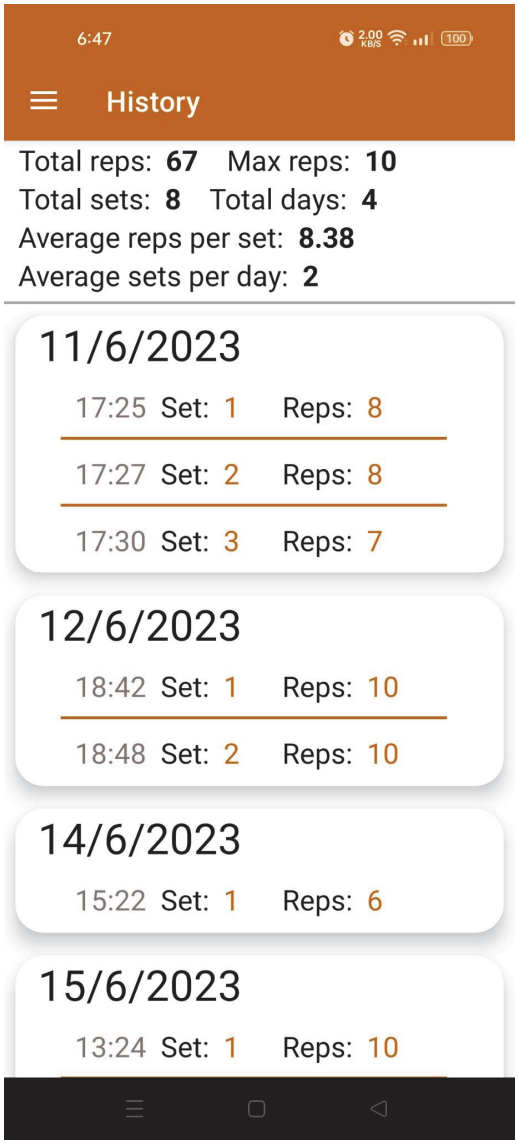


Εικόνα 21: Λειτουργία διαμοιρασμού (share)

Μπορεί επίσης να πατήσει το κουμπί **X** πάνω αριστερά ώστε να επιστρέψει στην αρχική οθόνη χωρίς να αποθηκεύσει το αποτέλεσμα του.

Πατώντας πάνω στο νούμερο των επαναλήψεων που μετρήθηκαν έχει την ευκαιρία να το επεξεργαστεί ώστε να διορθώσει τυχόν περιπτώσεις λάθος μέτρησης όπως ψευδής αρνητικές (false negatives) ή ψευδής θετικές (false positives) μετρήσεις.. Τέλος, πατώντας στο κουμπί **SAVE** το αποτέλεσμα του αποθηκεύεται στην τοπική βάση δεδομένων και μπορεί πλέον να το ξαναδεί πηγαίνοντας στην οθόνη του Ιστορικού, την οποία θα δούμε επομένως.

HistoryFragment:



Εικόνα 22: Οθόνη Ιστορικού

Στην οθόνη ιστορικού μπορούμε να δούμε στατιστικά στοιχεία που αφορούν το σύνολο των προπονήσεων που έχουμε καταγράψει μέσω της εφαρμογής, όπως τις συνολικές επαναλήψεις που έχουμε κάνει (Total reps) το οποίο βρίσκουμε προσθέτοντας όλα τα **Set** όλων των ημέρων που υπάρχουν μέσα στην βάση δεδομένων αλλά και τις περισσότερες συνεχόμενες επαναλήψεις, δηλαδή επαναλήψεις μέσα σε ένα σετ (Max reps) το οποίο βρίσκουμε ψάχνοντας το μέγιστο από τα Set που υπάρχουν μέσα στην βάση δεδομένων. Σύνολο των σετ (Total sets) το οποίο βρίσκουμε προσθέτοντας όλα τα Set όλων των ημερών. Σύνολο των ημέρων που πραγματοποιήθηκε εξάσκηση (Total days) το οποίο βρίσκουμε μετρώντας το μέγεθος της λίστας των ημέρων δηλαδή του αντικειμένου **Day** που επιστρέφετε από την βάση δεδομένων.

Το μέσο όρο των επαναλήψεων μέσα στα σετ (Average reps per set) που είναι η διαίρεση του συνόλου των επαναλήψεων δια τον αριθμό των σετ και το μέσο όρο των σετ μέσα σε μια ημέρα (Average sets per day) που είναι η διαίρεση του συνόλου των σετ δια τον αριθμό των ημερών. Στην οθόνη αυτή βλέπουμε επίσης μια λίστα από ημέρες και τα set που έχουν πραγματοποιηθεί μέσα σε κάθε ημέρα και την ώρα που αποθηκευτήκαν. Αρχικά προτού αποθηκευτεί ο αριθμός ενός set στην βάση ελέγχουμε εάν η συγκεκριμένη ημέρα υπάρχει στην βάση. Εάν δεν υπάρχει, δημιουργούμε μια καινούργια ημέρα (Day) και το Set θα αποθηκευτεί μέσα στην νέο-δημιουργηθείσα ημέρα, εάν υπάρχει θα προστεθεί στην ήδη υπάρχουσα ημέρα.

Παρατηρούμε τέσσερις ημέρες οπότε το Total days αναγραφεί τον αριθμό τέσσερα (4), με την πρώτη να έχει τρία(3) Set με οκτώ (8), οκτώ (8) και επτά (7) επαναλήψεις για κάθε σετ. Το δέκα (10) είναι ο μέγιστος αριθμός επαναλήψεων οπότε είναι και το Max reps που φαίνεται. Στο σύνολο τους όλες οι ημέρες έχουν εξήντα επτά (67) επαναλήψεις και μπορούμε να το δούμε στην ένδειξη Total reps.Έχουν οκτώ (8) σετ το οποίο φαίνεται στην ένδειξη Total sets. Τέλος ο μέσος όρος των σετ είναι Total sets / Total days = 2 και ο μέσος όρος των επαναλήψεων είναι 67 επαναλήψεις / 8 σετ = 8.38 τα οποία αναγράφονται αντίστοιχα.

Κεφάλαιο 4ο: Εκπαίδευση μοντέλου

4.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα γίνει μια επισκόπηση της διαδικασίας και του εργαλείου που χρησιμοποιήθηκε για την εκπαίδευση του μοντέλου.

4.2 Συλλογή εικόνων

Για την εκπαίδευση του αλγόριθμου k-πλησιέστερων γειτόνων (k-nn) ο οποίος είναι υπεύθυνος για την κατηγοριοποίηση/ταξινόμηση της άσκησης των έλξεων βάσει δεδομένης της θέσης του σώματος του αθλούμενου μια συγκεκριμένη χρονική στιγμή, δημιουργήθηκε ένα σύνολο δεδομένων με παραδείγματα με ετικέτα. Το αρχικό βήμα περιλάμβανε τη συλλογή ενός επιμελημένου συνόλου εικόνων και φωτογραφιών από το διαδίκτυο και μη, στιγμιότυπων από βίντεο γυμναστικής που απεικονίζουν άτομα να εκτελούν έλξη στην ανώτερη και στην κατώτερη θέση του εύρους κίνησης της άσκησης. Αυτές οι εικόνες επιλέχθηκαν και διαχωρίστηκαν σε δύο κατηγορίες χειροκίνητα για να καταγράψουν τα δυο διαφορετικά στάδια της άσκησης, συμπεριλαμβανομένων και των φάσεων προς τα πάνω και προς τα κάτω. **[13]**

4.3 Χειροκίνητη επισήμανση

Μετά τη συλλογή εικόνων, οι εικόνες υποβλήθηκαν σε μια διαδικασία χειροκίνητου διαχωρισμού σε δύο ομάδες. Χειροκίνητα εξετάσθηκαν όλες οι εικόνες. Με βάση τις γνώσεις και την κατανόηση της άσκησης, κάθε εικόνα χαρακτηρίστηκε είτε ως "pullups_up" είτε ως "pullups_down", υποδηλώνοντας τις αντίστοιχες φάσεις της άσκησης των έλξεων. Οι εικόνες στη συνέχεια χωρίστηκαν σε δύο ξεχωριστές ομάδες με βάση αυτόν τον χαρακτηρισμό. Έτσι κάθε εικόνα τοποθετήθηκε στην ομάδα που ανήκει.

Εικόνα 23: Οι δυο ομάδες εικόνων

Ο αριθμός, η ποικιλία και η ποιότητα στα δεδομένα εκπαίδευσης παίζουν μεγάλο ρόλο στην απόδοση του αλγορίθμου. Ωστόσο η διαδικασία αυτή είναι χρονοβόρα. Συνολικά χρησιμοποιήθηκαν πενήντα (50) εικόνες, είκοσι οκτώ (28) για την άνω θέση και είκοσι δύο (22) για την κάτω θέση. Είναι προφανές λοιπόν ότι θα αποτελούσε βελτίωση η συλλογή περισσοτέρων δεδομένων εκπαίδευσης.

4.4 Εξαγωγή τοποθέτησης

Μετά τη χειροκίνητη διαδικασία επισήμανσης, οι εικόνες υποβλήθηκαν σε επεξεργασία χρησιμοποιώντας έναν αλγόριθμο ανίχνευσης τοποθέτησης. Αυτός ο αλγόριθμος χρησιμοποίησε τεχνικές όρασης υπολογιστή (computer vision) για την ανίχνευση των 33 σημείων του σώματος, με αποτέλεσμα την εξαγωγή των πληροφοριών της τοποθέτησης του σώματος σε ένα CSV αρχείο. Ένα αρχείο CSV (Comma Separated Values) είναι ένα αρχείο που περιέχει τιμές, συνήθως αριθμητικές, οι οποίες διαχωρίζονται συνήθως από το κόμμα (,) από οπού παίρνουν και την ονομασία τους αλλά πολλές φορές ο διαχωριστικός χαρακτήρας είναι και ο χαρακτήρας (;).Το αρχείο αυτό περιέχει μια λίστα από πόζες. Οι γραμμές του CSV είναι όσες και οι εικόνες που τροφοδοτήσαμε τον αλγόριθμο πλην αυτών που δεν εντοπίστηκε η παρουσία ανθρωπίνου σώματος, αν υπήρξε τέτοια περίπτωση. Οι στήλες αποτελούνται από το όνομα του αρχείου, το αποτέλεσμα της χειροκίνητης κατηγοριοποίησης του και τα σημεία x, y, z για κάθε ένα από τα 33 ορόσημα που αντιπροσωπεύουν σημαντικά σημεία μέσα στη τοποθέτηση του σώματος

4.5 Εργαλείο παραγωγής αρχείου

Η εκτέλεση του αλγορίθμου για την ανάγνωση εικόνων και τη δημιουργία του αρχείου CSV επιτεύχθηκε με τη χρήση ενός εργαλείου Google Colab. **[14]**

Συγκεκριμένα, οι αλγοριθμικές διαδικασίες εκτελέστηκαν χρησιμοποιώντας το εργαλείο Google Colab που είναι διαθέσιμο στον παρακάτω σύνδεσμο:

https://colab.research.google.com/drive/19txHpN8exWhstO6WVkfmYYVC6uug_oVR

Αυτό το εργαλείο ήταν ζωτικής σημασίας για την διαδικασία μετατροπής των εικόνων σε μια δομημένη μορφή CSV αξιοποιώντας τη δύναμη των βιβλιοθηκών προγραμματισμού της γλώσσας προγραμματισμού Python και μηχανικής μάθησης. Η χρήση αυτού του πόρου διευκόλυνε την εξαγωγή των πληροφοριών τοποθέτησης του σώματος από τις εικόνες, συμβάλλοντας έτσι σημαντικά στη διαδικασία δημιουργίας δεδομένων που θα χρησιμοποιηθούν αργότερα για την εκπαίδευση του αλγόριθμου k-πλησιέστερων γειτόνων.

4.6 Μορφή αρχείου

Το αρχείο έχει όπως αναφέρθηκε νωρίτερα μια γραμμή για κάθε εικόνα εκπαίδευσης που ανιχνεύτηκε η τοποθέτηση του σώματος. Στο δημιουργημένο αρχείο CSV παρατηρούμε ότι έχει συνολικά 47 γραμμές. Είκοσι πέντε (25) από αυτές έχουν την επισήμανση “pullups_up” και είκοσι δύο (22) την επισήμανση “pullups_down”. Αυτό σημαίνει ότι σε τρείς από τις εικόνες για την άνω θέση των έλξεων δεν ανιχνεύτηκε ανθρώπινο σώμα. Όσον αφορά τις στήλες του αρχείου. Έχει εκατό και μια (101) στήλες οπού η πρώτη στήλη είναι το όνομα του αρχείου πχ image_001.jpg. Η δεύτερη στήλη είναι η ετικέτα που έχει δοθεί ανάλογα με την κατηγορία “ pullups_up ” ή “pullups_down”. Οι υπόλοιπες στήλες δηλαδή από

την τρίτη μέχρι και την εκατοστή πρώτη στήλη είναι ενενήντα εννέα (99) σε αριθμό. Αυτές αποτελούνται από τα τρία σημεία των αξόνων (X,Y,Z) επί κάθε ένα από τα 33 σημαντικά σημεία του σώματος.

20	image_023.jpg	pullups_up	1512.15	811.258	-197.032	1569.33	740.052
21	image_024.jpg	pullups_up	1701.67	493.664	-810.17	1755.58	434.267
22	image_025.jpg	pullups_up	1912.84	2038.96	-1319.49	1951.3	2014.44
23	image_027.jpg	pullups_up	1298.73	1662.53	-497.207	1303.58	1591.48
24	image_028.jpg	pullups_up	1414.49	1400.47	-980.215	1450.81	1326.68
25	image_029.jpg	pullups_up	1659.2	663.993	-766.077	1723.42	592.898
26	image_001.jpg	pullups_down	286.625	135.446	-199.553	292.457	129.675
27	image_002.jpg	pullups_down	99.3774	82.0746	-161.833	99.5117	78.9608
28	image_003.jpg	pullups_down	127.199	169.127	-173.938	133.797	161.798
29	image_004.jpg	pullups_down	567.613	142.743	52.8993	564.109	139.792
30	image_005.jpg	pullups_down	567.613	142.743	52.8993	564.109	139.792

Εικόνα 24: Δείγμα αρχείου csv

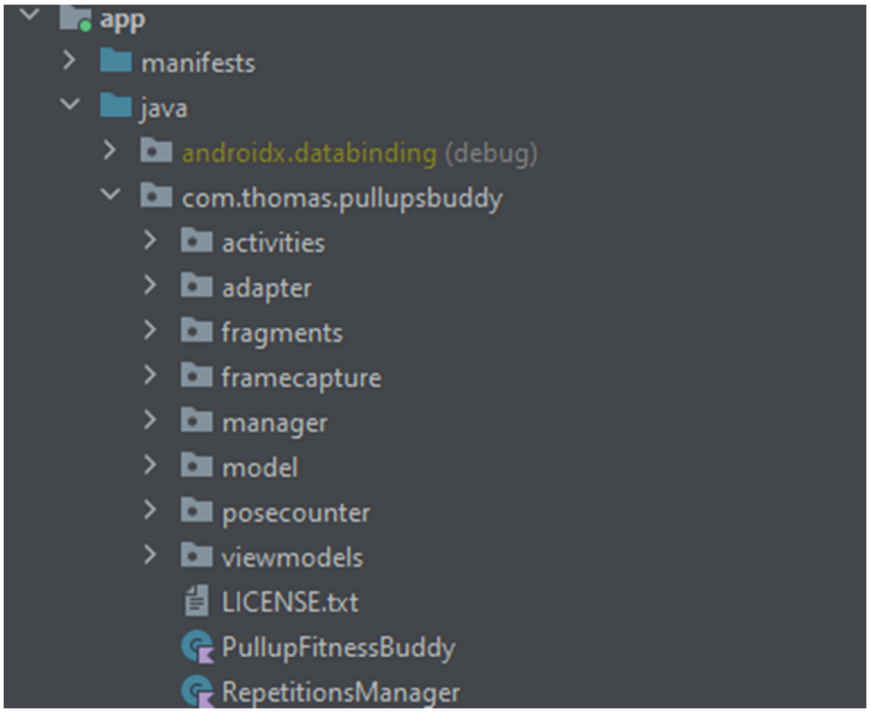
Κεφάλαιο 5ο: Υλοποίηση Εφαρμογής

5.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα πραγματοποιηθεί παρουσίαση της δομής του κώδικα της εφαρμογής, και τα κυριότερα και σημαντικότερα σημεία του κώδικα της.

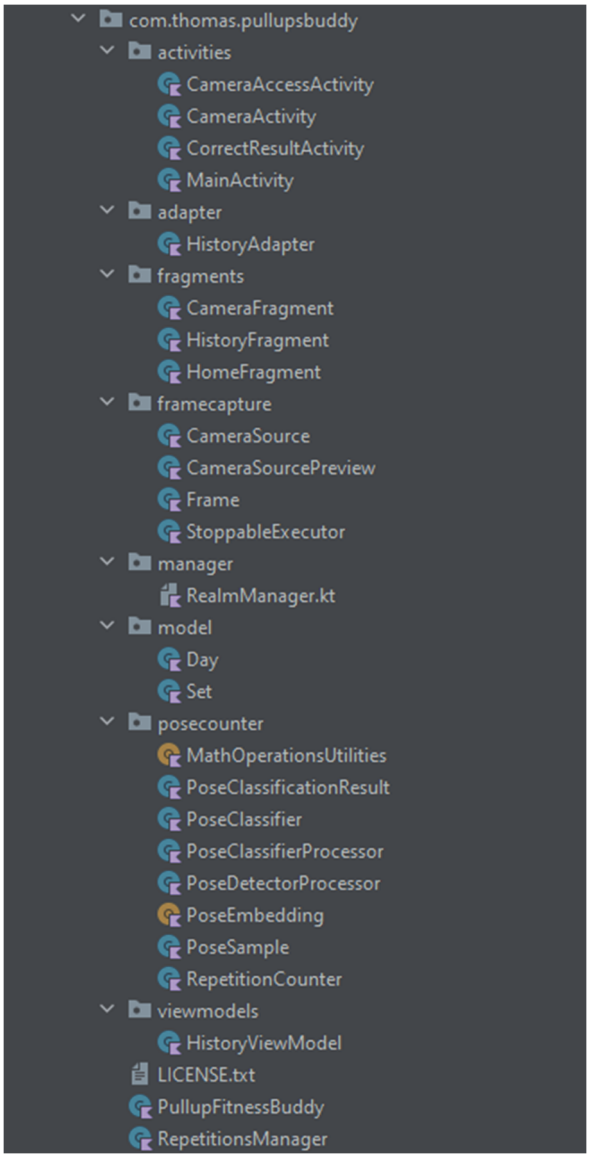
5.2 Εισαγωγή στην δομή της εφαρμογής.

Η εφαρμογή έχει διαχωριστεί σε packages (πακέτα) που εμπεριέχουν αρχεία κώδικα, μια πρακτική που συνηθίζεται στην ανάπτυξη λογισμικού και ειδικότερα στην Java.



Εικόνα 25: Δομή κώδικα εφαρμογής

Στην επόμενη εικόνα θα δούμε όλα τα αρχεία τα οποία βρίσκονται σε packages μέσα στην εφαρμογή.



Εικόνα 26: Δομή κώδικα εφαρμογής ανεπτυγμένη όψη

Στο package activities υπάρχουν όλα τα activities της εφαρμογής, οθόνες τις οποίες ο χρήστης μπορεί να δει και να αλληλοεπιδράσει.

Στο package adapter υπάρχει ο HistoryAdapter ο οποίος είναι ένας RecyclerView adapter. Είναι υπεύθυνος για την εμφάνιση του ιστορικού υπό την μορφή recycler view το οποίο είναι ένα αποδοτικό είδος λίστας αντικειμένων διεπαφής χρήστη.

Στο package fragments υπάρχουν όλα τα fragments της εφαρμογής τα οποία είναι επαναχρησιμοποιούμενα στοιχεία διεπαφής χρήστη παρόμοια με τα activities.

Στο package framecapture υπάρχουν οι κλάσεις κώδικα οι οποίες είναι υπεύθυνες για την λήψη και εμφάνιση πλαισίων(frames) από την κάμερα και τροφοδότηση αυτών στο posecounter package.

Στο package manager βρίσκεται το αρχείο RealmManager.kt το οποίο περιέχει συναρτήσεις διαχείρισης της τοπικής βάσης όπως αποθήκευση και φόρτωση του ιστορικού.

Στο package model υπάρχουν τα αντικείμενα μοντέλα της βάσης δεδομένων. Περιέχει την κλάση Day που αναπαριστά μια ημέρα προπόνησης όπως και την κλάση Set που αναπαριστά ένα Set μέσα σε μια ημέρα προπόνησης.

Στο package posecounter υπάρχουν όλα τα αρχεία και κλάσεις που έχουν ως στόχο την ανίχνευση, κατηγοριοποίηση και μέτρηση των pull ups και θα αναφερθούμε σε αυτά με μεγαλύτερη λεπτομέρεια στο υποκεφάλαιο που ακολουθεί.

Στο package viewmodels υπάρχει το HistoryViewModel. Το HistoryViewModel είναι μια κλάση που κληρονομεί από την κλάση ViewModel του Android. Το ViewModel είναι υπεύθυνο για την διαχείριση ενός στοιχείου διεπαφής χρήστη όπως είναι ένα activity ή ένα fragment. Χρησιμεύει στον διαχωρισμό του κώδικα της εφαρμογής με τα στοιχεία που εμφανίζονται στην οθόνη.

Στον αρχικό φάκελο της εφαρμογής υπάρχει ένα αρχείο LICENSE το οποίο είναι απαραίτητο διότι έχουν χρησιμοποιηθεί κομμάτια κώδικα της Google που αφορούν το ML Kit, όπως για παράδειγμα το αρχείο CameraSource που διαχειρίζεται την κάμερα ή το αρχείο MathOperationsUtilities το οποίο είναι ένα αρχείο με αριθμητικές πράξεις συγκεκριμένα για το PointF3D object που μας επιστρέφει το ML Kit.

Στον αρχικό φάκελο υπάρχει και το αρχείο PullupsFitnessBuddy που κληρονομεί από την κλάση Application. Η κλάση Application είναι μια κλάση βάση και η δική μας κλάση που θα το κληρονομήσει θα είναι η αρχική κλάση της εφαρμογής μας και χρησιμοποιείται συνήθως για κάποιες αρχικοποιήσεις όπως την αρχικοποίηση της βάσης δεδομένων.

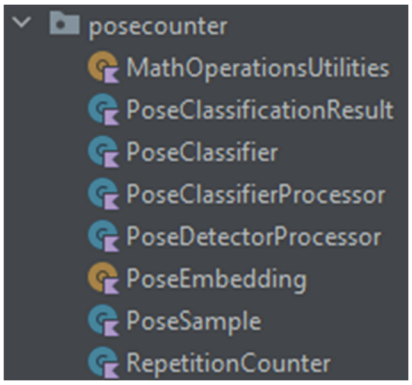
Τέλος, ο RepetitionsManager είναι μια κλάση Singleton υπεύθυνη για την αποθήκευση του αριθμού των επαναλήψεων. Οι Singleton κλάσεις είναι κλάσεις που σε αντίθεση με άλλες κλάσεις που μπορούμε να δημιουργήσουμε πολλαπλά αντικείμενα μιας κλάσης (πολλαπλά στιγμιότυπα), δημιουργούνται μια φόρα και παρέχουν πρόσβαση στο ίδιο μεμονωμένο αντικείμενο.

5.3 Παρουσίαση κώδικα ροής μέτρησης έλξεων.

Σε αυτό το υποκεφάλαιο θα παρουσιάσουμε τα κυριότερα σημεία του κώδικα της εφαρμογής και συγκεκριμένα το πως πραγματοποιείται η ανίχνευση, η κατηγοριοποίηση και η μέτρηση των επαναλήψεων.

5.3.1 Επισκόπηση πακέτου posecounter

Όπως βλέπουμε και από την ιεραρχική όψη της εφαρμογής στο προγραμματιστικό περιβάλλον του Android Studio το πακέτο pose counter στην Εικόνα 27 παρακάτω έχει τα εξής αρχεία:



Εικόνα 27 Πακέτο posecounter

- PoseDetectorProcessor: Αυτή η κλάση λειτουργεί ως γέφυρα μεταξύ του πακέτου λήψης καρέ (frame capture) και του πακέτου μετρητή πόζας (posecounter). Είναι δηλαδή το σημείο που λαμβάνονται τα frames από την υπόλοιπη εφαρμογή και ξεκινά η ανίχνευση μιας πόζας. Χρησιμοποιεί το MLKit Pose Detection API για να ανιχνεύει πόζες στα καρέ της κάμερας. Λαμβάνει δεδομένα ενός καρέ, τα επεξεργάζεται και μεταβιβάζει την ανιχνευμένη πόζα στον PoseClassifierProcessor για ταξινόμηση και καταμέτρηση.

- PoseClassifierProcessor: Αυτή η κλάση δέχεται ως είσοδο μια ροή (stream) αντικειμένων Pose και εκτελεί ταξινόμηση και μέτρηση επαναλήψεων. Φορτώνει δείγματα πόζας της μορφής της κλάσης PoseSample από ένα αρχείο CSV, το οποίο περιέχει προκαθορισμένες στάσεις με τις αντίστοιχες κλάσεις τους. Χρησιμοποιεί την κλάση PoseClassifier για να ταξινομήσει μια νέα ανιχνευμένη πόζα (Pose) που επιστρέφει ο ML Kit Detector και ενημερώνει τον αριθμό των επαναλήψεων με βάση τα αποτελέσματα της ταξινόμησης.

- PoseEmbedding: Αυτή η κλάση δημιουργεί ενσωματώσεις, δηλαδή αριθμητικές αναπαραστάσεις, για πόζες που είναι λίστες με τα 33 σημεία ορόσημα που επιστρέφονται από τον ML Kit Detector. Πραγματοποιεί κανονικοποίηση στα σημεία ορόσημα, υπολογίζει το μέγεθος της πόζας και υπολογίζει τις αποστάσεις μεταξύ διαφόρων ορόσημων για να σχηματίσει την τελική ενσωμάτωση της πόζας, επιστρέφοντας έτσι μια λίστα από τρισδιάστατα σημεία.

- PoseSample: Αυτή η κλάση αντιπροσωπεύει μια αναπαράσταση πόζας με το όνομα, την τάξη και τα 33 τρισδιάστατα σημεία εκείνης. Διαβάζει δείγματα πόζας από ένα αρχείο CSV και δημιουργεί αντικείμενα της ίδιας κλάσης PoseSample χρησιμοποιώντας το κατασκευαστικό πρότυπο σχεδίασης factory method.

- RepetitionCounter: Αυτή η κλάση μετράει τις επαναλήψεις για μια κατηγορία πόζας (άνω θέση ή κάτω θέση). Λαμβάνει τα αποτελέσματα ταξινόμησης από τον PoseClassifierProcessor και ενημερώνει τον αριθμό των επαναλήψεων με βάση προκαθορισμένα όρια για την είσοδο και την έξοδο από μια στάση.

- PoseClassifier: Αυτή η κλάση εκτελεί ταξινόμηση πόζας με βάση τα αποθηκευμένα δείγματα πόζας. Για την ταξινόμηση, χρησιμοποιεί τον αλγόριθμο K-Nearest Neighbors με φιλτράρισμα των ακραίων τιμών για να βρει τους πλησιέστερους γείτονες στη ανιχνευμένη στάση που της δόθηκε σαν είσοδος. Υπολογίζει τις αποστάσεις μεταξύ της ενσωμάτωσης της θέσης εισόδου και των ενσωματώσεων των δειγμάτων πόζας και επιστρέφει ένα αντικείμενο PoseClassificationResult.

- PoseClassificationResult: Αυτή η κλάση αντιπροσωπεύει το αποτέλεσμα της ταξινόμησης μιας δοσμένης πόζας. Αποθηκεύει τις τιμές εμπιστοσύνης για τις διάφορες τάξεις πόζας(άνω θέση και κάτω θέση) και παρέχει μεθόδους πρόσβασης και χειρισμού των αποτελεσμάτων ταξινόμησης.

- Τέλος, το αντικείμενο MathOperationsUtilities παρέχει βοηθητικές μεθόδους για την εκτέλεση μαθηματικών πράξεων σε αντικείμενα PointF3D, τα οποία αντιπροσωπεύουν τρισδιάστατα σημεία με συντεταγμένες X, Y και Z.

Αυτές οι κλάσεις αλληλοεπιδρούν ως εξής:

Η κλάση PoseDetectorProcessor λαμβάνει ζωντανά καρέ/πλαίσια από την κάμερα και χρησιμοποιεί την κλάση του ML Kit, PoseDetector για να ανιχνεύει πόζες. Διαβιβάζει τις ανιχνευμένες πόζες στην κλάση PoseClassifierProcessor, η οποία εκτελεί ταξινόμηση και μέτρηση χρησιμοποιώντας την κλάση PoseClassifier. Η κλάση PoseClassifier χρησιμοποιεί την κλάση PoseEmbedding για να δημιουργήσει τις ενσωματώσεις για την ανιχνευμένη πόζα και συγκρίνει αυτές τις ενσωματώσεις της ανιχνευμένης πόζας με τα αποθηκευμένα δείγματα πόζας για να προσδιορίσει την κλάση της ανιχνευμένης πόζας.

Η κλάση RepetitionCounter ενημερώνει τον αριθμό των επαναλήψεων με βάση τα αποτελέσματα της ταξινόμησης. Συνολικά, αυτή η ροή λαμβάνει πλαίσια/καρέ από την κάμερα, ανιχνεύει πόζες μέσα σε αυτά, ταξινομεί την ανιχνευμένη πόζα και να μετράει επαναλήψεις των έλξεων σε πραγματικό χρόνο.

5.3.2 Παρουσίαση κώδικα του πακέτου posecounter

Το κύριο σημείο εισόδου στην εφαρμογή είναι το MainActivity. Το MainActivity είναι υπεύθυνο ως προς τα δυο fragments που φιλοξενεί HomeFragment και HistoryFragment αλλά και για την αρχικοποίηση του Navigation Drawer.

```
class MainActivity : AppCompatActivity() {

    private lateinit var appBarConfiguration: AppBarConfiguration
    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        setSupportActionBar(binding.appBarMain.toolbar)

        val drawerLayout: DrawerLayout = binding.drawerLayout
        val navView: NavigationView = binding.navView
        val navController = findNavController(R.id.nav_host_fragment_content_main)
        appBarConfiguration = AppBarConfiguration(
            setOf(
                R.id.nav_home, R.id.nav_history
            ), drawerLayout
        )
        setupActionBarWithNavController(navController, appBarConfiguration)
        navView.setupWithNavController(navController)
    }

    override fun onSupportNavigateUp(): Boolean {
        val navController = findNavController(R.id.nav_host_fragment_content_main)
        return navController.navigateUp(appBarConfiguration) || super.onSupportNavigateUp()
    }
}
```

Εικόνα 28: Κώδικας MainActivity

Το αρχικό fragment που εμφανίζεται είναι το HomeFragment.

```
private fun setCameraPermissionLauncher() {
    requestPermissionLauncher = registerForActivityResult(
        ActivityResultContracts.RequestPermission()
    ) { isGranted: Boolean ->
        if (isGranted) {
            // Permission is granted.
            startCameraActivity()
        } else {
            // Permission denied. Show something and lead user to settings.
            startCameraAccessActivity()
        }
    }
}

private fun setListeners() {
    binding.start.setOnClickListener { it: View!
        checkForPermission()
    }
}

private fun startCameraActivity() {
    val intent = Intent(requireActivity(), CameraActivity::class.java)
    startActivity(intent)
}

private fun startCameraAccessActivity() {
    val intent = Intent(requireActivity(), CameraAccessActivity::class.java)
    startActivity(intent)
}

private fun checkForPermission() {
    if (ContextCompat.checkSelfPermission(
            requireActivity(), Manifest.permission.CAMERA
        ) == PackageManager.PERMISSION_GRANTED
    ) {
        // Permission is granted.
        startCameraActivity()
    } else {
        // You can directly ask for the permission.
        // The registered ActivityResultCallback gets the result of this request.
        requestPermissionLauncher.launch(
            Manifest.permission.CAMERA
        )
    }
}
```

Εικόνα 29: HomeFragment

Στο HomeFragment όταν πατηθεί το κουμπί START θα τρέξει η συνάρτηση checkForPermission() οπού ελέγχουμε αν υπάρχει ήδη πρόσβαση στην κάμερα. Εάν δεν υπάρχει ζητάμε μέσω κλήσης στο λειτουργικό. Αν ο χρήστης δώσει ή έχει ήδη δώσει πρόσβαση θα οδηγηθεί μέσω intent στο CameraActivity αν όχι θα οδηγηθεί στο CameraAccessActivity.

Το αρχείο CameraActivity:

```
class CameraActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_camera)

        if (savedInstanceState == null) {
            supportFragmentManager.beginTransaction()
                .replace(R.id.fragment_container, CameraFragment())
                .commit()
        }
    }
}
```

Εικόνα 30: CameraActivity

Η κλάση CameraActivity φιλοξενεί το CameraFragment που φαίνεται στην παρακάτω εικόνα:

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    setRepetitionsToZero()
    setupViews()
    setupListeners()
    setupObservers()
    if (allPermissionsGranted()) {
        createCameraSource()
    } else {
        requestPermissions()
    }
}
```

Εικόνα 31: CameraFragment

Το CameraFragment καθαρίζει την μέτρηση των pull ups μέσα στην κλάση RepetitionsManager

```
private fun setRepetitionsToZero() {
    val repetitionsManager = RepetitionsManager.getInstance()
    repetitionsManager.setReps(0)
    repetitionsManager.setPose("")
}
```

Εικόνα 32: Συνάρτηση setRepetitionsToZero

και αρχικοποιεί την κλάση CameraSource

```
private fun createCameraSource() {
    if (cameraSource == null) {
        cameraSource = CameraSource(requireActivity())
    }
    try {
        val poseDetectorOptions = getPoseDetectorOptionsForLivePreview()
        cameraSource!!.setMachineLearningFrameProcessor(
            PoseDetectorProcessor(
                requireContext(), poseDetectorOptions
            )
        )
    } catch (e: Exception) {
        Log.e( tag: "CameraFragment",  msg: "Can not create image processor:", e)
        Toast.makeText(
            requireContext(),  text: "Can not create image processor: ${e.message}", Toast.LENGTH_LONG
        ).show()
    }
}
```

Εικόνα 33: Συνάρτηση createCameraSource

Η κλάση CameraSource αρχικοποιεί εμφανίζει και διαχειρίζεται την κάμερα και τα πλαίσια της κάμερας της συσκευής. Μετά την αρχικοποίηση της CameraSource αρχικοποιούμε και το attribute frameProcessor της κλάσης με την κλάση PoseDetectorProcessor. Η κλάση CameraSource είναι στο framecapture package και η κλάση PoseDetectorProcessor είναι στο posecounter package συνεπώς είναι το σημείο επικοινωνίας αυτών των δυο packages. Η κλάση CameraSource στέλνει συνεχώς frames και η κλάση PoseDetectorProcessor καλείται να ανιχνεύσει μέσα σε αυτά την στάση του σώματος κατά την εκγύμναση.

Η κλάση Frame:

```
class Frame(val data: ByteBuffer, val width: Int, val height: Int, val rotation: Int)
```

Εικόνα 34: Κλάση Frame

data: τα δεδομένα της εικόνας της κάμερας.

Width, height, rotation: metadata που μας ζητάει η κλάση ανιχνευτής, Detector του ML Kit, δηλαδή επιπλέον πληροφορίες για τα data, στην περίπτωση μας για το frame.

Για την δημιουργία της κλάσης PoseDetectorProcessor:

```
private fun getPoseDetectorOptionsForLivePreview(): PoseDetectorOptionsBase {
    return PoseDetectorOptions.Builder().setDetectorMode(PoseDetectorOptions.STREAM_MODE)
        .build()
```

Εικόνα 35: Συνάρτηση getPoseDetectorOptionsForLivePreview

Χρησιμοποιούμε το builder που παρέχεται από το ML Kit, εδώ μπορούμε να θέσουμε επιλογές όπως την λειτουργία της ανίχνευσης Accurate ή Fast, έχουμε επιλέξει την λειτουργία Fast για λόγους που αναφέρθηκαν σε προηγούμενο κεφάλαιο.

Μέσα στην κλάση PoseDetectorProcessor υπάρχει η συνάρτηση detectPose() :

```
    private fun detectPose(frame: Frame) {

        val mlImage =
            ByteBufferMlImageBuilder(
                frame.data,
                frame.width,
                frame.height,
                MlImage.IMAGE_FORMAT_NV21
            )
                .setRotation(frame.rotation).build()

        detector
            .process(mlImage)
            .addOnSuccessListener(classificationExecutor) { pose ->

                if (poseClassifierProcessor == null) {
                    poseClassifierProcessor = PoseClassifierProcessor(context)
                }
                poseClassifierProcessor!!.getPoseResult(pose)

                //continue with another frame
                processLatestFrameFromBuffer()

            }
            .addOnFailureListener(classificationExecutor) { e ->
                onFailure(e)
            }

        mlImage.close()
    }
```

Εικόνα 36: Συνάρτηση detectPose()

Η συνάρτηση detectPose() καλείται κάθε φόρα που λαμβάνεται νέο frame και ξανακαλείται μόλις τελειώσει σε κάποιο επόμενο διαθέσιμο frame. Χρησιμοποιώντας τον Detector που αρχικοποιήσαμε νωρίτερα ανιχνεύει την θέση του σώματος μέσα σε ένα frame.

```
private val detector: PoseDetector = PoseDetection.getClient(options)
```

Εικόνα 37: ML Kit Detector

Το MlImage χρησιμοποιείται ως περιτύλιγμα για τα δεδομένα του frame το οποίο θα μπορούσε να είναι κοινόχρηστο σε πολλαπλές πλατφόρμες μεταξύ διαφορετικών πλαισίων ODML(On Device Machine Learning) της Google (TFLite, MLKit), αν χρειαζόταν. Χρησιμοποιείται για την επεξεργασία των frames που καταγράφονται από την κάμερα και μεταβιβάζονται στο ML Kit Pose Detection API για ανίχνευση πόζας. Ο PoseDetectorProcessor παίρνει ένα αντικείμενο Frame και κατασκευάζει από αυτό ένα αντικείμενο MlImage. Για να δημιουργηθεί ένα MlImage, χρησιμοποιείται ένα από τα παρεχόμενα builders. Στην κλάση PoseDetectorProcessor χρησιμοποιείται το ByteBufferMlImageBuilder. **[15]** Έπειτα το κατασκευασμένο MlImage υποβάλλεται σε επεξεργασία από τον ανιχνευτή πόζας και ως αποτέλεσμα λαμβάνεται το αντικείμενο πόζας που ανιχνεύεται. Στην συνέχεια αυτή η ανιχνευμένη θέση (Pose) που περιέχει μια λίστα από σημεία ενδιαφέροντος του σώματος (μιλήσαμε για αυτό σε προηγούμενο κεφάλαιο) περνάει στο επόμενο βήμα, αυτό της κατηγοριοποίησης.

Μετά από την ανίχνευση το επόμενο βήμα της διαδικασίας είναι η κλάση PoseClassifierProcessor. Η κλάση PoseClassifierProcessor αποτελεί το πρώτο βήμα της κατηγοριοποίησης της στάσης του σώματος. Όταν δημιουργηθεί ένα αντικείμενο της κλάσης αυτής θα τρέξει μέσα στο init block της κλάσης η συνάρτηση loadPosesFromCsv(). Το init block στην γλώσσα Kotlin είναι ιδιαίτερα χρήσιμο όταν θέλουμε να τρέξουμε κώδικα στην αρχικοποίηση της κλάσης. Οπότε όταν φτιάξουμε ένα αντικείμενο της κλάσης PoseClassifierProcessor θα τρέξει το init block, έτσι θα δημιουργηθεί επίσης ένα αντικείμενο repCounter.Τότε η συνάρτηση loadPosesFromCsv() θα διαβάσει το csv αρχείο που έχουμε περάσει στα assets της εφαρμογής (στα assets βρίσκονται συνήθως αρχεία με πληροφορίες που δεν εμπεριέχουν κώδικα) και θα αγνοήσει της άδειες (null) γραμμές του csv αρχείου. Αυτή η συνάρτηση με την σειρά της θα φτιάξει ένα αντικείμενο της κλάσης PoseClassifier.

```
class PoseClassifierProcessor @WorkerThread constructor(context: Context) {


    companion object {
        private const val POSE_SAMPLES_FILE_PULLUPS = "pose/pullups_samples.csv"
        private const val className = "pullups_down"
    }

    private var repCounter: RepetitionCounter
    private var poseClassifier: PoseClassifier? = null
    private var lastRepResult: Int = 0

    init {
        Preconditions.checkState( expression: Looper.myLooper() != Looper.getMainLooper())
        repCounter = RepetitionCounter(className)
        lastRepResult = 0
        loadPosesFromCsv(context)
    }

    private fun loadPosesFromCsv(context: Context) {
        val poseSamples: MutableList<PoseSample> = ArrayList()
        try {
            val reader = BufferedReader(
                InputStreamReader(context.assets.open(POSE_SAMPLES_FILE_PULLUPS))
            )
            var csvLine = reader.readLine()
            while (csvLine != null) {
                val poseSample: PoseSample? = PoseSample.getPoseSample(csvLine, separator: ",")
                if (poseSample != null) {
                    poseSamples.add(poseSample)
                }
                csvLine = reader.readLine()
            }
        } catch (e: IOException) {
            Log.e( tag: "PoseClassifierProcessor", msg: "Error when loading pose samples.\n$e")
        }
        poseClassifier = PoseClassifier(poseSamples)

    }
```

Εικόνα 38: Κώδικας PoseClassifierProcessor

Το επόμενο βήμα της διαδικασίας της κατηγοριοποίησης και μέτρησης είναι η κλάση PoseClassifier η οποία με την σειρά της χρησιμοποιεί τις κλάσεις PoseSample και PoseEmbedding.

Η κλάση PoseSample αντιπροσωπεύει μια τοποθέτηση ή πόζα και εμπεριέχει τις πληροφορίες σχετικά με το όνομα, την τάξη ή κλάση και τα 33 σημεία ορόσημα της πόζας. Η κλάση PoseSample παρέχει μια εργοστασιακή μέθοδο, το getPoseSample, που παίρνει μια γραμμή από ένα CSV αρχείο από την κλάση PoseClassifierProcessor και ένα διαχωριστικό, στην προκείμενη περίπτωση το κόμμα, ως είσοδο. Αναλύει τη γραμμή CSV και εξάγει το όνομα, την κλάση και τις πληροφορίες των σημείων και δημιουργεί ένα αντικείμενο PoseSample.Όσες γραμμές περιέχει το CSV τόσα και τα αντικείμενα PoseSample που θα

δημιουργηθούν. Η μέθοδος getPoseSample εξάγει τα δεδομένα της τοποθέτησης βάση μιας συγκεκριμένης μορφής της γραμμής CSV: "Όνομα, Κλάση,X1,Y1,Z1,X2,Y2,Z2,...". Κάθε συντεταγμένη (X, Y, Z) αντιπροσωπεύει τη θέση ενός σημείου στον τρισδιάστατο χώρο. Η μέθοδος χωρίζει τη γραμμή CSV και μετατρέπει τις τιμές σε αντικείμενα PointF3D για να συμπληρωθεί η λίστα των σημείων.

```
class PoseSample(val name: String, val className: String, landmarks: List<PointF3D>) {

    val embedding: List<PointF3D?> = PoseEmbedding.getPoseEmbedding(landmarks)

    companion object {
        private const val TAG = "PoseSample"
        private const val NUM_LANDMARKS = 33
        private const val NUM_DIMS = 3

        fun getPoseSample(csvLine: String?, separator: String?): PoseSample? {
            val tokens = Splitter.onPattern(separator.toString()).splitToList(csvLine.toString())
            if (tokens.size != NUM_LANDMARKS * NUM_DIMS + 2) {
                Log.e(TAG, msg: "Invalid number of tokens for PoseSample")
                return null
            }
            val name = tokens[0]
            val className = tokens[1]
            val landmarks: MutableList<PointF3D> = ArrayList()
            var i = 2
            while (i < tokens.size) {
                try {
                    landmarks.add(
                        PointF3D.from(
                            tokens[i].toFloat(),
                            tokens[i + 1].toFloat(),
                            tokens[i + 2].toFloat()
                        )
                    )
                } catch (e: NullPointerException) {
                    Log.e(TAG, msg: "Invalid value " + tokens[i] + " for landmark position.")
                    return null
                } catch (e: NumberFormatException) {
                    Log.e(TAG, msg: "Invalid value " + tokens[i] + " for landmark position.")
                    return null
                }
                i += NUM_DIMS
            }
            return PoseSample(name, className, landmarks)
        }
    }
}
```

Εικόνα 39: Κλάση PoseSample

Η κλάση PoseEmbedding έχει ως σκοπό τον υπολογισμό σημαντικών αποστάσεων μεταξύ μια στάσης σώματος. Χρησιμοποιώντας την μετατρέπουμε μια λίστα που αναπαριστά τα 33 σημαντικά σημεία του σώματος σε μια λίστα με 23 υπολογισμένες αποστάσεις μεταξύ κάποιων από αυτών των 33 σημείων. Στην θεωρία θα μπορούσαμε να πάρουμε πολύ περισσότερες μοναδικές αποστάσεις για 33 σημεία. Για να υπολογίσουμε πόσες μοναδικές αποστάσεις θα μπορούσαμε συνολικά να πάρουμε χρησιμοποιούμε τον τύπο:

$$\frac{n \times (n-1)}{2}$$

Για n = 33 το σύνολο των μοναδικών αποστάσεων μεταξύ των σημείων είναι: 33 x (33 -1) / 2 = 528 μοναδικές αποστάσεις. Όμως αυτό θα ήταν και περιττό και υπολογιστικά επιβαρυντικό στο πρόγραμμα.

```
embedding.add(
    MathOperationsUtilities.subtract(
        MathOperationsUtilities.average(
            lm[PoseLandmark.LEFT_HIP],
            lm[PoseLandmark.RIGHT_HIP]
        ),
        MathOperationsUtilities.average(
            lm[PoseLandmark.LEFT_SHOULDER],
            lm[PoseLandmark.RIGHT_SHOULDER]
        )
    )
)
embedding.add(
    MathOperationsUtilities.subtract(
        lm[PoseLandmark.LEFT_SHOULDER], lm[PoseLandmark.LEFT_ELBOW]
    )
)
embedding.add(
    MathOperationsUtilities.subtract(
        lm[PoseLandmark.RIGHT_SHOULDER], lm[PoseLandmark.RIGHT_ELBOW]
    )
)
embedding.add(
    MathOperationsUtilities.subtract(
        lm[PoseLandmark.LEFT_ELBOW],
        lm[PoseLandmark.LEFT_WRIST]
    )
)
```

Εικόνα 40: Αποστάσεις PoseEmbedding

Ένας λόγος για τη χρήση αποστάσεων κατά ζεύγη είναι ότι είναι αμετάβλητες ως προς τη μεταφορά και την περιστροφή. Εάν αποφασίζαμε να συγκρίνουμε τις συντεταγμένες των 33 σημείων, τότε δύο πανομοιότυπες στάσεις θα μπορούσαν να φαίνονται διαφορετικές εάν η μία από αυτές μετατοπιζόταν ή είχε περιστρέφει σε σχέση με την άλλη. Αντίθετα, συγκρίνοντας αποστάσεις ανά ζεύγη, κάνουμε κανονικοποίηση των στάσεων για να αφαιρέσετε το παραπάνω ενδεχόμενο παραλλαγών. Ένας άλλος λόγος για τη χρήση αποστάσεων κατά ζεύγη είναι ότι μπορούν να τονίσουν τα σημαντικά χαρακτηριστικά που έχουν την πραγματική σημασία για την άσκηση των έλξεων που δεν είναι εμφανή από τις ακατέργαστες συντεταγμένες. Για παράδειγμα, εάν μια στάση καταγράφεται από μια ελαφρώς διαφορετική γωνία ή εάν το άτομο κινείται πιο κοντά ή πιο μακριά από την κάμερα, τότε οι μετατοπίσεις των σημείων θα μπορούσαν να αλλάξουν, ακόμα κι αν το άτομο στην πράξη βρίσκεται στην ιδιά θέση. Στην περίπτωση αυτή, η σύγκριση των μετατοπίσεων θα μπορούσε να οδηγήσει σε λανθασμένα αποτελέσματα ταξινόμησης.

Η κλάση PoseClassifier έχει ως στόχο την κατηγοριοποίηση ή ταξινόμηση μιας ανιχνευμένης στάσης σώματος, η οποία ανιχνεύτηκε νωρίτερα, με βάση μια λίστα δειγμάτων που επίσης έλαβε νωρίτερα από το csv αρχείο μέσα στην κλάση PoseClassifierProcessor. Χρησιμοποιεί τον αλγόριθμο K-Nearest Neighbors (k - πλησιέστερων γειτόνων), στον οποίο αναφερθήκαμε και σε προηγούμενο κεφάλαιο, με φιλτράρισμα ακραίων τιμών. Όταν δημιουργηθεί η κλάση PoseClassifier λαμβάνει μια λίστα αντικειμένων PoseSample, αυτό συμβαίνει στην κλάση PoseClassifierProcessor που συζητήσαμε προηγουμένως, ο σκοπός εδώ είναι να φορτωθούν οι αποθηκευμένες πόζες που θα χρησιμοποιηθούν για τον υπολογισμό της απόστασης μετά. Οι δυο σταθερές (maxDistanceTopK και meanDistanceTopK) ορίζουν τον αριθμό των "γειτόνων" που λαμβάνονται υπόψη στον αλγόριθμο σε δυο βήματα. Η μεταβλητή axesWeights χρησιμοποιείται για να λάβουμε λιγότερο υπόψη τα νούμερα του άξονα Z σε σύγκριση με τον X και τον Y όταν γίνονται υπολογισμοί λόγω της γενικά χαμηλότερης ακρίβειάς του στην ταξινόμηση πόζας χρησιμοποιώντας το MLKit, καθώς όπως αναφέρει και το ίδιο η τιμή που μας επιστρέφει για τον Z άξονα σε κάθε σημείο είναι σε πειραματικό στάδιο.

Η μέθοδος extractPoseLandmarks(pose: Pose): Είναι μια βοηθητική μέθοδος που δέχεται μια ανιχνευμένη στάση σαν αντικείμενο και επιστρέφει αυτήν την ανιχνευμένη στάση σαν λίστα από 3d σημεία. Εξάγει τα σημεία ορόσημα από την στάση που ανιχνεύτηκε και τα επιστρέφει σε μια λίστα αντικειμένων PointF3D που είναι αντικείμενα που αντιπροσωπεύουν ένα τρισδιάστατο σημείο στο χώρο, που σημαίνει ότι έχουν μια δεκαδική τιμή για τον άξονα x μια για το y και μια για τον z, τα οποία επιστρέφονται σε μια λίστα και αντιπροσωπεύουν τα 33 ορόσημα σώματος σε μια ανιχνευμένη στάση.**[16]**

```
class PoseClassifier(private val poseSamples: List<PoseSample>) {

    private val maxDistanceTopK = 30
    private val meanDistanceTopK = 10
    private val axesWeights = PointF3D.from( x: 1f,  y: 1f,  z: 0.2f)

    private fun extractPoseLandmarks(pose: Pose): List<PointF3D> {
        val landmarks: MutableList<PointF3D> = ArrayList()
        for (poseLandmark in pose.allPoseLandmarks) {
            landmarks.add(poseLandmark.position3D)
        }
        return landmarks
    }
```

Εικόνα 41: Κλάση PoseClassifier

Ακολουθεί η μέθοδος classify(pose: Pose) η οποία είναι η κύρια συνάρτηση που χρησιμοποιείται για την ταξινόμηση της στάσης και χρησιμοποιεί την βοηθητική μέθοδο extractPoseLandmarks().

- Εξάγει τα σημεία της εντοπισμένης στάσης.
- Εάν δεν έχουν εντοπιστεί σημεία, η λίστα θα είναι άδεια, τότε επιστρέφει αμέσως ένα κενό αποτέλεσμα.

- Παίρνει ένα αντικείμενο PoseEmbedding από τα σημεία ορόσημα της ανιχνευμένης πόζας δηλαδή μια λίστα από αποστάσεις μεταξύ των σημείων.
- Χρησιμοποιώντας αυτό το αντικείμενο ενσωμάτωσης πόζας, υπολογίζει τη μέγιστη απόσταση μεταξύ της στάσης και κάθε δείγματος στα αποθηκευμένα δείγματα πόζας, διατηρώντας εκείνα με τη μικρότερη απόσταση (εξαιρουμένων των ακραίων τιμών). Οι μέγιστες αποστάσεις σταθμίζονται από τα βάρη των αξόνων.

```
fun classify(pose: Pose): PoseClassificationResult {

    val landmarks = extractPoseLandmarks(pose)
    val result = PoseClassificationResult()
    if (landmarks.isEmpty()) {
        return result
    }

    val embedding = PoseEmbedding.getPoseEmbedding(landmarks)

    val maxDistances = PriorityQueue(
        maxDistanceTopK
    ) { o1: Pair<PoseSample, Float?>, o2: Pair<PoseSample, Float?> ->
        -(o1.second!!).compareTo(o2.second!!)
    }
    for (poseSample in poseSamples) {
        val sampleEmbedding = poseSample.embedding
        var max = 0f
        for (i in embedding.indices) {
            max = max.coerceAtLeast(MathOperationsUtilities.maxAbs(MathOperationsUtilities.multiply(MathOperationsUtilities.subtract(embedding[i], sampleEmbedding[i]),
                axesWeights
                    )
                )
            )
        }
        maxDistances.add(Pair(poseSample, max))
        if (maxDistances.size > maxDistanceTopK) {
            maxDistances.poll()
        }
    }
```

Εικόνα 42: Μέθοδος classify (υπολογισμός μέγιστων αποστάσεων)

- Στη συνέχεια, υπολογίζει τη μέση απόσταση μεταξύ της στάσης και των δειγμάτων που διατηρήθηκαν από το προηγούμενο βήμα. Διατηρεί και πάλι αυτά με τη μικρότερη μέση απόσταση (εξαιρουμένων των ακραίων τιμών). Αυτές οι μέσες αποστάσεις σταθμίζονται επίσης από τα βάρη των αξόνων.
- Τέλος, αυξάνει την εμπιστοσύνη της κλάσης για καθεμία από τις στάσεις που αντιπροσωπεύονται στα υπόλοιπα δείγματα. Η τάξη με τις περισσότερες εμφανίσεις θα έχει την υψηλότερη εμπιστοσύνη.

```
    val meanDistances = PriorityQueue(
        meanDistanceTopK
    ) { o1: Pair<PoseSample, Float?>, o2: Pair<PoseSample, Float?> ->
        -(o1.second!!).compareTo(o2.second!!)
    }
    for (sampleDistances in maxDistances) {
        val poseSample = sampleDistances.first
        val sampleEmbedding = poseSample.embedding
        var sum = 0f
        for (i in embedding.indices) {
            sum += MathOperationsUtilities.sumAbs(MathOperationsUtilities.multiply(MathOperationsUtilities.subtract(embedding[i], sampleEmbedding[i]),
                axesWeights))
        }
        val meanDistance = sum / (embedding.size)
        meanDistances.add(Pair(poseSample, meanDistance))
        if (meanDistances.size > meanDistanceTopK) {
            meanDistances.poll()
        }
    }
    for (sampleDistances in meanDistances) {
        val className = sampleDistances.first.className
        result.incrementClassConfidence(className)
    }
    return result
}
```

Εικόνα 43: Μέθοδος classify (υπολογισμός μέσων αποστάσεων)

Το αντικείμενο PoseClassificationResult κρατά την εμπιστοσύνη για κάθε τάξη πόζας με βάση τον αριθμό των πλησιέστερων γειτόνων της. Όσο περισσότεροι γείτονες από την ίδια τάξη, τόσο μεγαλύτερη είναι η εμπιστοσύνη για αυτήν την τάξη. Αυτό είναι το χαρακτηριστικό για ταξινομητές οι οποίοι βασίζονται στον αλγόριθμο k-NN, όπου η κλάση της πλειοψηφίας των πλησιέστερων γειτόνων χρησιμοποιείται για τον προσδιορισμό της κλάσης του δείγματος εισόδου. Στην παρακάτω εικόνα βλέπουμε τον κώδικα της κλάσης PoseClassificationResult:

```
class PoseClassificationResult {

    private var pullUpsUpConfidence: Float = 0f
    private var pullUpsDownConfidence: Float = 0f

    fun getClassConfidence(className: String): Float {
        var localConfidence = 0f
        if (className == "pullups_up") {
            localConfidence = pullUpsUpConfidence
        } else if (className == "pullups_down") {
            localConfidence = pullUpsDownConfidence
        }
        return localConfidence
    }

    fun maxConfidenceClass(): String {
        return if (pullUpsUpConfidence > pullUpsDownConfidence) {
            "pullups_up"
        } else {
            "pullups_down"
        }
    }

    fun incrementClassConfidence(className: String) {
        when (className) {
            "pullups_up" -> pullUpsUpConfidence++
            "pullups_down" -> pullUpsDownConfidence++
        }
    }
}
```

Εικόνα 44: Κλάση PoseClassificationResult

Εμπεριέχει μεθόδους οπού αυξάνουν και επιστρέφουν την εμπιστοσύνη της ζητούμενης κλάσης, incrementClassConfidence και getClassConfidence αντίστοιχα, είτε αυτή είναι η κλάση ‘pullups_up’ είτε είναι η κλάση ‘pullups_down’. Όπως και την μέθοδο maxConfidenceClass που επιστρέφει την κλάση με την μέγιστη εμπιστοσύνη μεταξύ των δυο κλάσεων. Η εμπιστοσύνη των δυο κλάσεων αποθηκεύεται σε μια floating point μεταβλητή και έχει δεκαδική τιμή από 0 μέχρι Κ δηλαδή 10.0 .

```
34          Log.d( tag: "confidence", msg: "pullUpsUpConfidence $pullUpsUpConfidence")
35          Log.d( tag: "confidence", msg: "pullUpsDownConfidence $pullUpsDownConfidence")
36          Log.d( tag: "confidence", msg: "confidence $localConfidence")
37          return localConfidence
```

Εικόνα 45: Logging μηχανισμός στην έξοδο της μεθόδου.

Προσθέτοντας logging μηχανισμό ώστε να δούμε τις τιμές που επιστρέφει η getClassConfidence στην κάτω θέση της κίνησης βλέπουμε την έξοδο στην κονσόλα:

```
22:51:57.310 11146-11328 confidence            com.thomas.pullupsbuddy          D  pullUpsUpConfidence 0.0
22:51:57.310 11146-11328 confidence            com.thomas.pullupsbuddy          D  pullUpsDownConfidence 10.0
22:51:57.310 11146-11328 confidence            com.thomas.pullupsbuddy          D  confidence 10.0
```

Εικόνα 46: Έξοδος στην κονσόλα

Παρατηρούμε ότι στην κάτω θέση η εμπιστοσύνη της pullups_down κλάσης είναι 10.0 το μέγιστο δυνατό.

```
class RepetitionCounter(private val className: String) {

    private val enterThreshold = 6f
    private val exitThreshold = 4f

    var numRepeats = 0
        private set
    private var poseEntered = false

    fun addClassificationResult(poseClassificationResult: PoseClassificationResult): Int {
        val poseConfidence = poseClassificationResult.getClassConfidence(className)
        if (!poseEntered) {
            poseEntered = poseConfidence > enterThreshold
            return numRepeats
        }
        if (poseConfidence < exitThreshold) {
            numRepeats++
            poseEntered = false
        }
        return numRepeats
    }

}
```

Εικόνα 47: Κλάση RepetitionCounter

Η κλάση RepetitionCounter έχει ως σκοπό την μέτρηση των επαναλήψεων μιας συγκεκριμένης πόζας. Όταν αρχικοποιείται η κλάση αυτή δέχεται σαν είσοδο την κλάση pullups_down, δηλαδή ελέγχει την εμπιστοσύνη της κλάσης αυτής ώστε να πραγματοποιεί την μέτρηση.. Η μέθοδος addClassificationResult καλείται με το αποτέλεσμα της ταξινόμησης που λαμβάνει από την κλάση PoseClassifierProcessor.

Η μέτρηση των επαναλήψεων βασίζεται σε ορισμένα κατώφλια (thresholds): Το enterThreshold και το exitThreshold. Αυτά τα κατώφλια καθορίζουν πότε μια στάση θεωρείται ότι έχει εισέλθει ή εξέλθει. Το κατώφλι εισόδου enterThreshold είναι μια τιμή εμπιστοσύνης η οποία, όταν ξεπεραστεί από την

εμπιστοσύνη της πόζας, υποδεικνύει ότι έχει πραγματοποιηθεί η αναζητούμενη πόζα. Το κατώφλι εξόδου exitThreshold είναι μια τιμή εμπιστοσύνης η οποία, όταν η πόζα έχει μικρότερη εμπιστοσύνη από αυτήν, υποδεικνύει ότι ο άνθρωπος έχει βγει από αυτήν την πόζα αρά έχει γίνει μια επανάληψη της άσκησης.

5.4 Παρουσίαση κώδικα ροής αποθήκευσης και εμφάνισης ιστορικού/στατιστικών.

Σε αυτό το υποκεφάλαιο θα παρουσιάσουμε τα κυριότερα σημεία του κώδικα στα οποία πραγματοποιείται η αποθήκευση και εμφάνιση των επαναλήψεων που πραγματοποιήθηκαν.

5.4.1 Επισκόπηση σχετικών αρχείων

Τα αρχεία που έχουν ως σκοπό την αποθήκευση, φόρτωση και εμφάνιση των στατιστικών στοιχείων και του ιστορικού των επαναλήψεων είναι τα αρχεία:

- RealmManager: Αρχείο υπεύθυνο για την διαχείριση τον δεδομένων από την βάση δεδομένων.
- Day: Μοντέλο μιας ημέρας γυμναστικής.
- Set: Μοντέλο ενός σετ μέσα σε μια ημέρα.
- HistoryFragment: Κλάση διεπαφής χρήστη για εμφάνιση στην οθόνη.
- HistoryViewModel: Κλάση που περιέχει υπολογισμούς και λογική του HistoryFragment.
- HistoryAdapter: Κλάση ρύθμισης και διαχείρισης RecyclerView

5.4.2 Παρουσίαση κώδικα αρχείων

Το αρχείο RealmManager.kt:

Περιέχει δυο συναρτήσεις για αποθήκευση και φόρτωση των επαναλήψεων αντίστοιχα. Η πρώτη ονομάζεται saveSetToRealm , φαίνεται στην παρακάτω εικόνα και λαμβάνει ως είσοδο των αριθμό των επαναλήψεων που πραγματοποιήθηκαν. Πρώτου γίνει η αποθήκευση φορτώνονται σε μεταβλητές ο χρόνος, ο μήνας, η ημέρα, η ώρα και τα λεπτά. Ο συνδυασμός ημέρα-μήνα-χρόνου μας δίνει ένα μοναδικό κλειδί για να αποτελέσει το χαρακτηριστικό μιας μοναδικής εγγραφής ημέρας. Είναι λοιπόν ένα σύνθετο κύριο κλειδί. Όλες οι μεταβλητές αποθηκεύονται και για εμφάνιση μετέπειτα στον χρήστη. Συνεχίζοντας θα πραγματοποιήσουμε μια αναζήτηση μέσα στο Realm, την βάση δηλαδή. Εδώ υπάρχουν δυο περιπτώσεις, η μια είναι να επιστραφεί μη κενό αποτέλεσμα, αυτό σημαίνει ότι υπάρχει ήδη αυτή η ημέρα συνεπώς υπάρχει ήδη κάποιο Set μέσα σε αυτήν την ημέρα. Οπότε προσθέτουμε το νέο σετ μέσα στην λίστα των σετ αυξάνοντας των αριθμό του σετ κατά ένα και τέλος αποθηκεύουμε την αυξημένη λίστα των σετ στην ήδη υπάρχουσα ημέρα. Η δεύτερη περίπτωση είναι να επιστραφεί κενό αποτέλεσμα, το οποίο σημαίνει ότι δεν έχει δημιουργηθεί η σημερινή ημέρα άρα δεν έχει πραγματοποιηθεί και εκγύμναση για την σημερινή ημέρα. Οπότε δημιουργούμε μια λίστα με ένα σετ, την τοποθετούμε σε μια νέα ημέρα και

αποθηκεύουμε την ημέρα στην βάση. Εάν αποθηκευτούν ξανά δεδομένα για την τωρινή ημέρα πλέον θα επιστραφεί το αντικείμενο της ημέρας και θα συμβαίνει η πρώτη περίπτωση.

```
val realm: Realm = Realm.getDefaultInstance()

fun saveSetToRealm(reps: Int) {

    val calendar: Calendar = Calendar.getInstance(TimeZone.getDefault())
    val currentYear: String = calendar.get(Calendar.YEAR).toString()
    val currentMonth: String = (calendar.get(Calendar.MONTH) + 1).toString()
    val currentDay: String = calendar.get(Calendar.DAY_OF_MONTH).toString()
    val hourOfDay: String = calendar.get(Calendar.HOUR_OF_DAY).toString().padStart( length: 2,  padChar: '0')
    val minute: String = calendar.get(Calendar.MINUTE).toString().padStart( length: 2,  padChar: '0')

    val timeOfDay = "$hourOfDay:$minute"
    val dayMonthYear = currentDay + currentMonth + currentYear

    val result = realm.where(Day::class.java).equalTo( fieldName: "dayMonthYear", dayMonthYear).findFirst()
    if (result != null) {
        val listOfSet = result.sets
        val setNumber = listOfSet?.last()?.setNumber?.plus( other: 1)
        val newSetId = dayMonthYear + setNumber.toString()
        realm.beginTransaction()
        listOfSet?.add(Set(newSetId, setNumber, reps, timeOfDay))
        val dayObject = Day(dayMonthYear, currentDay, currentMonth, currentYear, listOfSet)
        realm.copyToRealmOrUpdate(dayObject)
        realm.commitTransaction()
    } else {
        val listOfSet = RealmList<Set>()
        val setNumber = 1
        val newSetId = dayMonthYear + "1"
        realm.beginTransaction()
        listOfSet.add(Set(newSetId, setNumber, reps, timeOfDay))
        val dayObject = Day(dayMonthYear, currentDay, currentMonth, currentYear, listOfSet)
        realm.copyToRealm(dayObject)
        realm.commitTransaction()
    }
}
```

Εικόνα 48: Μέθοδος saveSetToRealm

Η δεύτερη μέθοδος ονομάζεται getDaysFromRealm και είναι υπεύθυνη για να φορτώσει τα αποθηκευμένα δεδομένα από την βάση. Πραγματοποιείται μια αναζήτηση μέσα στην βάση και τελικά επιστρέφονται όλες οι αποθηκευμένες ημέρες με την μορφή λίστας.

```
fun getDaysFromRealm(): MutableList<Day> {
    var days: MutableList<Day> = mutableListOf()
    realm.executeTransaction { it: Realm
        val results: RealmResults<Day> = realm.where(Day::class.java).findAll()
        days = results
    }
    return days
}
```

Εικόνα 49: Μέθοδος getDaysFromRealm

Το αρχείο Day.kt:

Είναι το μοντέλο της ημέρας που στεγάζει μια λίστα από Set. Ένα αντικείμενο Day δημιουργείται την πρώτη φορά που θα εκπονηθεί ένα σετ σε μια καινούργια ημέρα. Έχει ένα αλφαριθμητικό σύνθετο κύριο κλειδί τον συνδυασμό ημέρας-μήνα-χρόνου πχ ‘’22112023’’ για να διασφαλιστεί η μοναδικότητα μιας ημέρας.

```
open class Day(
    @PrimaryKey
    var dayMonthYear: String? = null,
    var day: String? = null,
    var month: String? = null,
    var year: String? = null,
    var sets: RealmList<Set>? = RealmList<Set>()
) : RealmObject()
```

Εικόνα 50: Αρχείο Day.kt

Το αρχείο Set.kt:

Είναι το μοντέλο ενός σετ και στεγάζει τον αριθμό επαναλήψεων που πραγματοποιήθηκαν μέσα σε αυτό το σετ όπως και την πληροφορία για την ώρα που αποθηκεύτηκε αυτό το σετ. Ως κύριο κλειδί έχει το αλφαριθμητικό setId το οποίο λαμβάνει τιμή από το κύριο κλειδί της ημέρας συν τον αριθμό του σετ πχ 221120231 οπού το τελικό 1 είναι ο αριθμός του σετ.

```
open class Set(
    @PrimaryKey
    var setId: String? = null,
    var setNumber: Int? = null,
    var reps: Int? = null,
    var timeOfDay: String? = null
) : RealmObject()
```

Εικόνα 51: Αρχείο Set.kt

Το αρχείο HistoryFragment.kt

Το HistoryFragment είναι μια κλάση που κληρονομεί την κλάση Fragment του Android το οποίο είναι ένα στοιχείο διεπαφής χρήστη και εμφανίζει στατιστικά στοιχεία και λίστα με ιστορικό επαναλήψεων. Κάνει χρήση και αλληλοεπιδρά με το HistoryViewModel το οποίο εμπεριέχει την λογική για την φόρτωση του ιστορικού και τον υπολογισμό των στατιστικών. Χρησιμοποιεί το HistoryAdapter για να εμφανίσει το ιστορικό μέσα σε ένα RecyclerView.

Εικόνα 52: Στοιχεία διεπαφής χρήστη στατιστικών

```
override fun onResume() {
    super.onResume()
    setDaysObserver()
    historyViewModel.getDays()
}

private fun setDaysObserver() {
    historyViewModel.days.observe(viewLifecycleOwner) { daysList ->
        if (daysList.isNotEmpty()) {
            showStats(daysList)
            setAdapter(daysList)
        } else {
            binding.noWorkouts.visibility = View.VISIBLE
        }
    }
}
```

Εικόνα 53: Μέρος του κώδικα του HistoryFragment.

Στην μέθοδο onResume του HistoryFragment η οποία καλείται όταν το fragment γίνεται ορατό στον χρήστη. Δημιουργείται ένας observer (παρατηρητής) για τις ημέρες που είναι της μορφής LiveData μέσα στο HistoryViewModel. Καλείται και η μέθοδος getDays για να ανακτηθούν οι ημέρες προπόνησης.

Όταν η λίστα των ημερών ενημερώνεται, ενεργοποιείται η επανάκληση (callback) του παρατηρητή. Εάν η λίστα δεν είναι κενή, καλείται η μέθοδος showStats για εμφάνιση στατιστικών στοιχείων και η μέθοδος setAdapter για να ρυθμιστεί η κλάση HistoryAdapter με τη λίστα των ημερών. Εάν η λίστα είναι κενή, εμφανίζεται ένα μήνυμα που υποδεικνύει ότι δεν έχει εκπονηθεί καμία προπόνηση ακόμα.

```
private fun setAdapter(daysList: List<Day>) {
    historyAdapter = null
    historyAdapter = HistoryAdapter(daysList)
    binding.daysRecycler.setItemViewCacheSize(300)
    binding.daysRecycler.adapter = historyAdapter
}
```

Εικόνα 54: Μέθοδος setAdapter

Η μέθοδος setAdapter ρυθμίζει το HistoryAdapter με τη λίστα των αντικειμένων Day. Η κλάση HistoryAdapter είναι μια κλάση που διαμορφώνει την λειτουργία και συμπεριφορά του RecyclerView, το οποίο όντας στοιχείο διεπαφής χρήστη αποτελεί μια λίστα από πολλαπλά αντικείμενα.

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/days_recycler"
    android:background="@color/white"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:overScrollMode="never"
    android:scrollbars="vertical"
    app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
    app:layout_constraintTop_toBottomOf="@id/separatorView"
    app:layout_constraintBottom_toBottomOf="parent" />
```

Εικόνα 55: RecyclerView στην xml του HistoryFragment

Το αρχείο HistoryViewModel.kt:

Το HistoryViewModel είναι μια κλάση ViewModel του Android το οποίο παρέχει δεδομένα και μεθόδους που σχετίζονται με το ιστορικό και τα στατιστικά. Λειτουργεί ως γέφυρα επικοινωνίας μεταξύ της πηγής δεδομένων, δηλαδή στην περίπτωση μας την βάση δεδομένων, και των στοιχείων διεπαφής χρήστη, συγκεκριμένα του HistoryFragment. Το HistoryViewModel ορίζει ένα αντικείμενο LiveData που ονομάζεται days, το οποίο περιέχει μια λίστα με τα αντικείμενα που αντιπροσωπεύουν τις ημέρες προπόνησης. Αυτό το αντικείμενο LiveData επιτρέπει σε άλλα στοιχεία, όπως το HistoryFragment, να παρατηρούν αλλαγές στην τιμή του και να ενημερώνονται όταν αυτές συμβαίνουν ώστε να τρέξει κάποιο κομμάτι κώδικα και να διαχειριστεί αυτές τις αλλαγές όπως να τις εμφανίσει κλπ. Η μέθοδος getDays ανακτά τις ημέρες προπόνησης από την Realm βάση και ενημερώνει την τιμή της LiveData μεταβλητής days. Χρησιμοποιεί τη συνάρτηση getDaysFromRealm την οποία είδαμε νωρίτερα στο αρχείο RealmManager.Η μέθοδος getStatistics υπολογίζει στατιστικά στοιχεία με βάση την λίστα που επιστέφεται από την προηγουμένη μέθοδο. Πραγματοποιεί επανάληψη μέσα στην λίστα των ημέρων και την λίστα των σετ μέσα στην κάθε ημέρα για να υπολογίσει τις συνολικές επαναλήψεις, τα συνολικά σετ, τις μέγιστες επαναλήψεις σε ένα σετ, τον μέσο ορό των σετ ανά ημέρα, των μέσο όρο των επαναλήψεων ανά σετ και τις συνολικές ημέρες προπόνησης. Αυτά τα στατιστικά στοιχεία επιστρέφονται στο HistoryFragment όλα μαζί μέσα σε μια κλάση με την βοήθεια της εσωτερικής κλάση δεδομένων Stats.

```kotlin
class HistoryViewModel : ViewModel() {

    private var _days = MutableLiveData<MutableList<Day>>()
    val days: LiveData<MutableList<Day>>
        get() = _days

    fun getDays() {
        _days.value = getDaysFromRealm()
    }

    data class Stats(
        val totalReps: Int, val totalSets: Int, val maxRepsInASet: Int,
        val averageSetsPerDay: Double, val averageRepsPerSet: Double, val totalWorkoutDays: Int
    )

    fun getStatistics(daysList: List<Day>): Stats {
        var totalReps = 0
        var totalSets = 0
        var maxRepsInASet = 0

        for (day in daysList) {
            day.sets?.let { sets ->
                totalSets += sets.size
                for (set in sets) {
                    set.reps?.let { reps ->
                        totalReps += reps
                        if (reps > maxRepsInASet) {
                            maxRepsInASet = reps
                        }
                    }
                }
            }
        }
        val averageSetsPerDay =
            if (daysList.isNotEmpty()) totalSets.toDouble() / daysList.size else 0.0
        val averageRepsPerSet = if (totalSets != 0) totalReps.toDouble() / totalSets else 0.0
        return Stats(
            totalReps = totalReps, totalSets = totalSets, maxRepsInASet = maxRepsInASet,
            averageSetsPerDay = averageSetsPerDay, averageRepsPerSet = averageRepsPerSet, totalWorkoutDays = daysList.size
        ) }
}
```

Εικόνα 56: Κλάση HistoryViewModel

Το HistoryViewModel λειτουργεί ως διαμεσολαβητής μεταξύ του επιπέδου δεδομένων (βάση δεδομένων) και του επιπέδου διεπαφής χρήστη (fragment), παρέχοντας τα απαραίτητα δεδομένα και τους υπολογισμούς που απαιτούνται για την εμφάνιση του ιστορικού και των στατιστικών στο HistoryFragment. Ακολουθεί το μοτίβο αρχιτεκτονικής MVVM (Model-View-ViewModel) για να διαχωρίσει την λογική διαχείρισης της διεπαφής χρήστης με την λογική διαχείρισης των δεδομένων και να διατηρήσει έναν καθαρό διαχωρισμό μεταξύ των δυο.

Το αρχείο HistoryAdapter.kt:

```
class HistoryAdapter(private var data: List<Day>) : RecyclerView.Adapter<HistoryAdapter.ViewHolder>() {

    private lateinit var context: Context

    inner class ViewHolder(binding: ViewDayBinding) : RecyclerView.ViewHolder(binding.root) {
        var bindingDay: ViewDayBinding? = binding
    }

    override fun getItemCount(): Int = data.size

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        context = parent.context
        val mInflater: LayoutInflater = LayoutInflater.from(context)
        val binding: ViewDataBinding = DataBindingUtil.inflate(
            mInflater, R.layout.view_day, parent, attachToParent: false
        )
        return ViewHolder(binding as ViewDayBinding)
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        val currentItem = data[position]
        val binding = holder.bindingDay
        setupDay(binding, currentItem)
    }

    private fun setupDay(binding: ViewDayBinding?, currentItem: Day) {
        binding?.date?.text = "${currentItem.day}/${currentItem.month}/${currentItem.year}"
        for (set in currentItem.sets!!) {
            val setView = LayoutInflater.from(PullupFitnessBuddy.instance)
                .inflate(R.layout.view_set, binding?.setsLayout, attachToRoot: false)
            val timeTv: TextView = setView.findViewById(R.id.time)
            val setNumberTv: TextView = setView.findViewById(R.id.set_number)
            val repNumberTv: TextView = setView.findViewById(R.id.numberOfReps)
            timeTv.text = set.timeOfDay.toString()
            setNumberTv.text = set.setNumber.toString()
            repNumberTv.text = set.reps.toString()
            if (set == currentItem.sets!!.last()) {
                val setSeparatorView: View = setView.findViewById(R.id.set_separator)
                setSeparatorView.visibility = View.INVISIBLE
            }
            binding?.setsLayout?.addView(setView)
        }
    }
}
```

Εικόνα 57: Κλάση HistoryAdapter

Η κλάση HistoryAdapter είναι μια κλάση τύπου RecyclerView.Adapter του Android και είναι υπεύθυνη για τη σύνδεση των δεδομένων του ιστορικού στις αντίστοιχες προβολές στο HistoryFragment ώστε να εμφανιστεί η λίστα ιστορικού. Λαμβάνει μια λίστα με αντικείμενα Day στον κατασκευαστή, η οποία αντιπροσωπεύει το ιστορικό. Η κλάση ViewHolder περιέχει αναφορές στα στοιχεία διεπαφής χρήστη όπως περιγράφονται στη διάταξη του αρχείου view_day.xml. Το view_day.xml αντιπροσωπεύει την διάταξη και το πως θα εμφανίζεται ένα μοναδικό αντικείμενο της λίστας. Εφόσον όλα τα αντικείμενα εδώ είναι ιδίου τύπου αυτή η διάταξη θα φαίνεται για κάθε αντικείμενο της λίστας με διαφορετικά δεδομένα. Στην επόμενη εικόνα φαίνεται το αρχείο view_day.xml, η διάταξη των στοιχείων διεπαφής χρήστη που αποτελούν ένα αντικείμενο της λίστας.

```xml
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">
    <androidx.cardview.widget.CardView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="8dp"
        app:cardBackgroundColor="@android:color/white"
        app:cardCornerRadius="20dp"
        app:cardElevation="10dp">
        <androidx.constraintlayout.widget.ConstraintLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content">
            <TextView
                android:id="@+id/date"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginStart="16dp"
                android:textColor="@color/black"
                android:textSize="30sp"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toTopOf="parent"
                tools:text="14/5/2022">
            </TextView>

            <LinearLayout
                android:id="@+id/sets_layout"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_marginStart="32dp"
                android:layout_marginEnd="40dp"
                android:layout_marginBottom="10dp"
                android:orientation="vertical"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toBottomOf="@id/date" />
        </androidx.constraintlayout.widget.ConstraintLayout>
    </androidx.cardview.widget.CardView>
</layout>
```

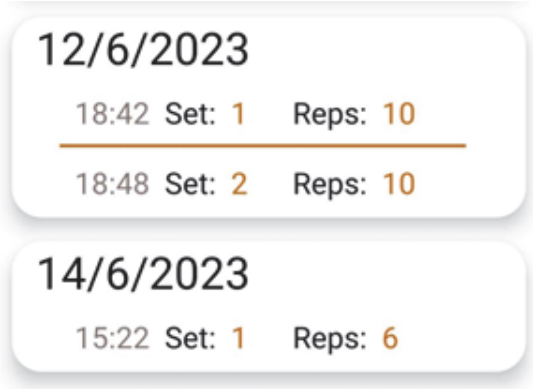
Εικόνα 58: Αρχείο view_day.xml

Εικόνα 59: view_day.xml όπως εμφανίζεται τελικά στην εφαρμογή

Η κλάση ViewHolder επεκτείνει την κλάση RecyclerView.ViewHolder και περιλαμβάνει ένα αντικείμενο ViewDayBinding για πρόσβαση στα στοιχεία διεπαφής χρήστη. Επεκτείνοντας την κλάση RecyclerView.ViewHolder πρέπει να ορίσουμε τις μεθόδους onCreateViewHolder, onBindViewHolder

και getItemCount. Η μέθοδος getItemCount επιστρέφει το μέγεθος της λίστας δεδομένων, που αντιστοιχεί στον αριθμό των ημερών προπόνησης. Στη μέθοδο onCreateViewHolder, ο adapter δημιουργεί τη διάταξη των στοιχείων (view_day.xml), δημιουργεί ένα νέο αντικείμενο της κλάσης ViewHolder και δεσμεύει τη διάταξη. Στη μέθοδο onBindViewHolder, ο adapter συνδέει τα δεδομένα ιστορικού με τις στοιχεία διεπαφής χρήστη στη διάταξη των στοιχείων. Πιο συγκεκριμένα, ανακτά το τρέχον αντικείμενο της λίστας των αντικειμένων Day με βάση τη θέση του στοιχείου και καλεί τη μέθοδο setupDay για να συμπληρώσει τα στοιχεία διεπαφής χρήστη της διάταξης με τα αντίστοιχα δεδομένα.

Η μέθοδος setupDay ρυθμίζει τα στοιχεία διεπαφής χρήστη από το αντικείμενο ViewDayBinding με βάση το παρεχόμενο αντικείμενο Day. Ορίζεται η ημερομηνία. Λόγω ότι ο αριθμός των σετ δεν είναι σταθερός και μπορεί να διαφέρει ανά ημέρα πρέπει τα σετ να εμφανιστούν δυναμικά, μέσα σε επαναληπτική δομή λοιπόν, διαβάζονται όλα τα σετ της ημέρας και με βάση αυτά δημιουργούνται δυναμικά διατάξεις view_set.xml για κάθε σετ, οπού συμπληρώνονται με τα δεδομένα του σετ (αριθμός σετ, επαναλήψεις, ώρα της ημέρας).



Εικόνα 60: Δυναμικός αριθμός σετ (view_set.xml) ανά ημέρα.

Κεφάλαιο 6ο: Επίλογος

6.1 Συμπεράσματα

Η εφαρμογή παρουσιάζει μια ιδιαίτερη χρήση της τεχνητής νοημοσύνης και ειδικότερα της μηχανικής μάθησης με επίβλεψη σε εφαρμογές για κινητές συσκευές, οντάς εφαρμογή βοηθός στη φυσική κατάσταση, παρέχει αναγνώριση και μέτρηση της σωστής εκτέλεσης της άσκησης των έλξεων σε πραγματικό χρόνο χρησιμοποιώντας ζωντανά την κάμερα για είσοδο. Κατά αυτόν τον τρόπο παρέχει δυνητικά επαγγελματική βοήθεια στον αθλούμενο, αν υποθέσουμε ότι τα δεδομένα εκπαίδευσης έχουν επιλεχθεί από έναν έμπειρο επαγγελματία και ότι είναι αρκετά στο πλήθος τους.

Χρησιμοποιώντας τα ήδη εκπαιδευμένα μοντέλα ανίχνευσης του ML Kit ώστε να διασφαλίσουμε όσο δυνατόν καλύτερη ανίχνευση. Όπως και χρησιμοποιώντας τον αλγόριθμο k-πλησιέστερων γειτόνων για ταξινόμηση και προσφέροντας την δυνατότητα διόρθωσης του αποτελέσματος μετά το πέρας του σετ, η εφαρμογή παρέχει μια απλοϊκή αλλά και αποτελεσματική λύση για την ορθή μέτρηση της άσκησης, πάρα τις πιθανές αστοχίες της ανίχνευσης λόγω χαμηλής ποιότητας της κάμερας ή μη ιδανικού φωτισμού αλλά και τις πιθανές αστοχίες της κατηγοριοποίησης λόγω ελλείπων δεδομένων εκπαίδευσης.

Είναι φιλική προς τον χρήστη παρέχοντας του ευκολία στην διαχείριση της άδειας της κάμερας και επεξηγηματικά μηνύματα. Έχοντας επιλογή διαμοιρασμού του αποτελέσματος σε άλλες εφαρμογές και οθόνη ιστορικού με στατιστικά στοιχεία αποθηκευμένα στην τοπική βάση δεδομένων του κινητού προσελκύει τον χρήστη να αθλείται περισσότερο και να ξεπερνάει το προηγούμενο του αποτέλεσμα.

Δεν χρησιμοποιεί το διαδίκτυο γιατί το ιστορικό αποθηκεύεται τοπικά και η ανίχνευση και κατηγοριοποίηση επίσης δεν χρησιμοποιούν το διαδίκτυο.

Έτσι, λοιπόν, καταλήγουμε στη διαπίστωση των θετικών σημείων της χρήσης μηχανικής μάθησης σε καθημερινές εφαρμογές κινητών.

6.2 Προβλήματα και προκλήσεις κατά την ανάπτυξη

Μια από τις προφανείς δυσκολίες είναι η ακρίβεια ανίχνευσης της τοποθέτησης του ανθρωπίνου σώματος. Η ποιότητα της κάμερας και ο ανεπαρκής ή υπερβολικός φωτισμός αλλά ακόμα και ο χώρος που πραγματοποιείται η άσκηση επηρεάζουν την ακρίβεια της ανίχνευσης. Επίσης η ακρίβεια της ταξινόμησης, η οποία είναι ανάλογη της ποσότητας και ποιότητας των δεδομένων εκπαίδευσης. Από την άλλη, αν είχαμε μεγάλο μέγεθος δεδομένων ο αλγόριθμος k-πλησιέστερων γειτόνων θα μπορούσε να αργεί να ανταπεξέλθει διότι όπως αναφέραμε σε προηγούμενο κεφάλαιο χρησιμοποιεί αναβλητική μάθηση δηλαδή εκτελεί τους υπολογισμούς του την στιγμή της εξέτασης μιας νέας περίπτωσης. Αυτό μπορεί να οδηγήσει σε υπερβολική χρήση της μνήμης στην οποία φορτώνονται τα δεδομένα και υπερβολική χρήση του επεξεργαστή κατά τους υπολογισμούς.

6.3 Πιθανές βελτιώσεις

Πιθανές βελτιώσεις της εφαρμογής συμπεριλαμβάνουν την βελτίωση της διεπαφής χρήστη με τις τελευταίες τάσεις στην σχεδίαση και την βελτιωμένη εκπαίδευση του υπάρχοντος μοντέλου ταξινόμησης.. Προσθήκη ρυθμίσεων οπού ο χρήστης θα μπορεί να διαλέξει την ποιότητα της κάμερας και τα frames per second (πλαίσια ανά δευτερόλεπτο). Επίσης σημαντική βελτίωση θα ήταν η προσθήκη νέων ασκήσεων είτε από τον προγραμματιστή μέσω ενημερώσεων είτε από τον χρήστη. Το δεύτερο σημαίνει ότι ο χρήστης θα έχει δυνατότητα να εκπαιδεύσει ο ίδιος το μοντέλο και να προσθέσει νέες ασκήσεις. Αυτό θα ήταν επίσης ιδιαίτερα χρήσιμο για έναν γυμναστή και τον εκπαιδευόμενο του. Όπου ο έμπειρος στην άσκηση γυμναστής, εκπαιδεύει το μοντέλο για χρήση από τον εκπαιδευόμενο.

Σύστημα ανατροφοδότησης οπού ο χρήστης θα μπορούσε να ενημερώσει την εφαρμογή για συγκεκριμένο λάθος μέτρησης ή ανίχνευσης. Κατά αυτόν τον τρόπο θα μπορούσαμε να έχουμε πληροφορίες για την απόδοση την ανίχνευσης και ταξινόμησης της εφαρμογής και να αναγνωρίσουμε τα αδύναμα σημεία βάση του συνόλου τον δεδομένων που θα λαμβάνουμε. Αν επιλέγαμε να χρησιμοποιήσουμε κάποιον άλλο αλγόριθμο διαφορετικής νοοτροπίας από τον k-nn δηλαδή κάποιον ο οποίος να μην ανήκει στην κατηγορία της μάθησης κατά περίπτωση οπού δεδομένα εκπαίδευσης διατηρούνται αυτούσια, αυτός ο νέος αλγόριθμος θα μπορούσε να μαθαίνει κατά την διάρκεια εκτέλεσης της άσκησης και αυτό θα ήταν ιδιαίτερα χρήσιμο.

Περισσότερα στατιστικά στοιχεία όπως για παράδειγμα αριθμό περισσοτέρων συνεχόμενων ημέρων γυμναστικής, διάρκεια του κάθε σετ, σε ποια ώρα της ημέρας ο χρήστης καταφέρνει περισσότερες επαναλήψεις ή ακόμα και γραφικά στοιχεία όπως η πρόοδος του αριθμού επαναλήψεων του χρήστη με το πέρασμα του χρόνου, γράφημα (αριθμός επαναλήψεων – χρόνος) και χρήση αυτών σαν συμβουλές ως προς τον χρήστη.

Τέλος, ένταξη κοινωνικών στοιχείων μέσω διαδικτύου όπως δημιουργία προσωπικού προφίλ και χρήση κάποιου τύπου scoreboard (πίνακα αποτελεσμάτων) ώστε να μπορεί να συγκρίνει την επίδοση του ενάντια σε άλλους ανθρώπους από όλο τον κόσμο ή ακόμα φίλους και συγγενείς, συμμετοχή σε challenges (προκλήσεις) κοκ.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] IBM. "What is Artificial Intelligence?". [Online].
Διαθέσιμο: https://www.ibm.com/cloud/learn/what-is-artificial-intelligence.

[2] IBM. "Machine Learning". [Online]. Διαθέσιμο: https://www.ibm.com/topics/machine-learning.

[3] I. Vlachavas, P. Kefalas, N. Vassiliadis, F. Kokkoras, and H. Sakellariou, Τεχνητή Νοημοσύνη - Γ' Έκδοση. Thessaloniki, Greece: Εκδόσεις Πανεπιστημίου Μακεδονίας, 2011. (ISBN: 978-960-8396-64-7)

[4] "Android (operating system)". Wikipedia. [Online].
Διαθέσιμο: https://en.wikipedia.org/wiki/Android_(operating_system).

[5] "Java (software platform)". Java.com. [Online].
Διαθέσιμο: https://www.java.com/en/download/help/whatis_java.html.

[6] B. Lutkevich, "Kotlin". TechTarget. [Online].
Διαθέσιμο: https://www.techtarget.com/whatis/definition/Kotlin.

[7] Google Samples. "ML Kit: Vision Quickstart". [Online].
Διαθέσιμο: https://github.com/googlesamples/mlkit/tree/master/android/vision-quickstart.

[8] Google Developers. "ML Kit for Firebase". [Online].
Διαθέσιμο: https://developers.google.com/ml-kit/guides.

[9] Google Developers. "Pose Detection". [Online].
Διαθέσιμο: https://developers.google.com/ml-kit/vision/pose-detection.

[10] Google Developers. "Pose Detection in Android". [Online].
Διαθέσιμο: https://developers.google.com/ml-kit/vision/pose-detection/android.

[11] Google Developers. "FaceDetectorOptions". [Online].
Διαθέσιμο:
https://developers.google.com/android/reference/com/google/mlkit/vision/face/FaceDetectorOptions.

[12] MongoDB. "MongoDB Realm: Introduction". [Online].
Διαθέσιμο: https://www.mongodb.com/docs/realm/introduction/.

[13] Google Developers. "Classifying Poses". [Online].
Διαθέσιμο: https://developers.google.com/ml-kit/vision/pose-detection/classifying-poses.

[14] Google Colaboratory. "Pose Estimation with ML Kit". [Online].
Διαθέσιμο: https://colab.research.google.com/drive/19txHpN8exWhstO6WVkfmYYVC6uug_oVR.

[15] Google Developers. "MlImage". [Online].
Διαθέσιμο: https://developers.google.com/android/reference/com/google/android/odml/image/MlImage.

[16] Google Developers. "PointF3D". [Online].
Διαθέσιμο: https://developers.google.com/android/reference/com/google/mlkit/vision/common/PointF3D.