ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ, ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΗΜΙΟΥΡΓΙΑ ΠΛΑΤΦΟΡΜΑΣ ΓΙΑ ΤΗΝ ΚΑΤΑΓΡΑΦΗ ΔΙΑΤΡΟΦΗΣ ΚΑΙ ΔΡΑΣΤΗΡΙΟΤΗΤΑΣ, ΠΡΟΒΛΕΨΗ ΜΕΛΛΟΝΤΙΚΩΝ ΑΣΘΕΝΕΙΩΝ ΜΕ ΤΗΝ ΧΡΗΣΗ ΔΕΝΤΡΩΝ ΑΠΟΦΑΣΗΣ

Πτυχιακή Εργασία του

Λαμπρακόπουλου Δημητρίου (113744)

Επιβλέπων: Ευ. Αντωνίου, Καθηγητής

ΘΕΣΣΑΛΟΝΙΚΗ, ΣΕΠΤΕΜΒΡΙΟΣ 2023

Υπεύθυνη Δήλωση *: Βεβαιώνω ότι είμαι συγγραφέας αυτής της πτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην πτυχιακή εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης, βεβαιώνω ότι αυτή η πτυχιακή εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τις απαιτήσεις του προγράμματος σπουδών του Τμήματος Μηχανικών Πληροφορικής, Υπολογιστών και Τηλεπικοινωνιών του Διεθνούς Πανεπιστημίου της Ελλάδας.*

Περίληψη

Στη παρούσα πτυχιακή ερευνάται ο τρόπος παρουσίασης και διαχείρισης δεδομένων διατροφής και άσκησης που προέρχονται από την καθημερινότητα του ανθρώπου, καθώς και η σχέση τους με την ύπαρξη της νόσου του μυοκαρδίου. Η δυσκολία εξαγωγής συμπερασμάτων από πρωτογενή πληροφορία που μπορεί να παραχθεί από την καθημερινότητα μας απαιτεί την υλοποίηση μιας πλατφόρμας συλλογής και διαχείρισης δεδομένων. Για την υλοποίηση αυτής της πλατφόρμας, χρησιμοποιήθηκαν τεχνολογίες ανάπτυξης διαδικτυακών υπηρεσιών, πιο συγκεκριμένα η χρήση του πλαισίου ανάπτυξης εφαρμογών Spring Boot και συμβατών βιβλιοθηκών μέσω maven. Όσον αφορά τη δομή της εφαρμογής λήφθηκαν υπόψη μεθοδολογίες που την καθιστούν επεκτάσιμη ως προς τις διεπαφές και τις υπηρεσίες που προσφέρει. Παράλληλα ερευνήθηκε το σύνολο δεδομένων Heart Disease του πανεπιστημίου του Irvine της Καλιφόρνια για την δημιουργία μοντέλου μηχανικής μάθησης. Η βιβλιοθήκη που χρησιμοποιήθηκε για την ανάλυση των δεδομένων και εκπαίδευση του μοντέλου είναι η scikit learn με χρήση της γλώσσας Python. Η μέθοδος που εξήγαγε τα καλύτερα αποτελέσματα ήταν η RandomForest που χρησιμοποιεί ταξινομητές CART και η εύρεση των καλύτερων παραμέτρων με τη μέθοδο RandomizedSearchCV, το ποσοστό επιτυχίας ξεπερασε το **88%**. Επισης υλοποιήθηκε διεπαφή με χρήση του πλαισίου React που χρησιμοποιεί τις διαδικτυακές υπηρεσίες που προαναφέρθηκαν. Μελλοντικά ένα ερώτημα που προκύπτει από την έρευνα αυτής της πτυχιακής είναι αν θα μπορούσαμε να αυτοματοποιήσουμε την διαδικασία πρόβλεψης και γενικότερα παραγωγής συμπερασμάτων χωρίς να μεσολαβεί ανθρώπινη παρέμβαση.

Περιεχόμενα

Εισαγωγή

Η καθημερινότητα του μέσου ανθρώπου τα τελευταία χρόνια έχει αλλάξει, ο περιορισμένος χρόνος έχει συμβάλει αρνητικά στην υγεία, το άγχος και το στρες εχουν κατακλύσει κάθε μέρος της ζωής μας και τα προβλήματα υγείας πληθαίνουν, αλλά πόσα από αυτά τα προβλήματα προέρχονται από την ελάχιστη άσκησης, την κακή διατροφή και την κακή ποιότητα των προϊόντων. Μελέτες έχουν δείξει ότι οι θρεπτικές ουσίες στα φρούτα και λαχανικά έχουν μειωθεί από τα προηγούμενα χρόνια έως και 38% [8].

Η έλλειψη βασικών θρεπτικών ουσιών γίνεται ορατή καθώς συμπληρώματα διατροφής είναι πλέον απαραίτητα και αναπόφευκτα στην καθημερινή ζωή. Η Ελλάδα κρατάει την θέση της στις τρεις πρώτες χώρες με τα υψηλότερα ποσοστά παχυσαρκίας μέσα στην Ευρωπαϊκή Ένωση [9]. Σύμφωνα με τα δεδομένα που αναφέρθηκαν και είναι λογικό να συμπεράνουμε και προβλήματα υγείας που προκύπτουν, όπως παρουσιάστηκε μέσω ερευνών τα ποσοστά θνησιμότητας στην Ελλάδα, προέρχονται κατά 51.1% από νόσο του μυοκαρδίου.

Η έρευνα αυτή επιδιώκει την μοντελοποίηση και τεκμηρίωση μιας πλατφόρμας συλλογής δεδομένων με σκοπό την ευρύτερη ανάλυση της καθημερινής ζωής του ανθρώπου ως προς την διατροφή και δραστηριότητα. Μοντελοποιώντας μια πλατφόρμα συλλογής δεδομένων δίνεται το πλαίσιο ανάπτυξης όμοιων έργων από τεχνολογικά ινστιτούτα για σκοπούς συνεισφοράς στην υγεία. Στα πλαίσια αυτής της πτυχιακής ερευνώνται τεχνολογίες και πρακτικές που επιλύουν προβλήματα ανάλυσης δεδομένων, μεγάλου όγκου δεδομένων, αρχιτεκτονικής υπηρεσιών-εφαρμογών νέφους και διεπαφών χρήστη καθώς και η δημιουργία και χρήση μοντέλου μηχανικής μάθησης.

Σε αυτή την εργασία θα βρείτε τα μέρη που διέπουν το πρακτικό μέρος της πτυχιακής και την εκτενή περιγραφή τους, η εργασία έχει συνταχθεί από τρια βασικά μέρη και την περιγραφή μερικών βοηθητικών τεχνολογιών που χρησιμοποιήθηκαν.

Στο πρώτο μέρος γίνεται μια αναφορά στο Δέντρο αποφάσεων και στα μοντέλα ταξινόμησης και παλινδρόμησης (CART), με βασικούς ορισμού για να κατανοήσουμε για ποιους λόγους έγινε η επιλογή αυτού του μοντέλου για την παρούσα πτυχιακή. Επίσης παρουσιάζονται μέθοδοι που αξιοποιούν τα μοντέλα CART και βελτιστοποιεί τις λειτουργίες τους και βελτιώνουν τις αδυναμίες του.

Στο δεύτερο μέρος περιγράφεται το σύνολο δεδομενων (Heart Disease dataset) [1] που χρησιμοποιήθηκε για την υλοποίηση του μοντέλου, περιλαμβάνει την περιγραφή των χαρακτηριστικών, την προετοιμασία, προεπεξεργασία και την χρήση μεθόδων για την βελτιστοποίηση της απόδοσης του μοντέλου.

Στο τρίτο μέρος παρουσιάζεται η δομή του έργου, η μορφή των διαδικτυακών υπηρεσιών και των τεχνολογιών που χρησιμοποιήθηκαν. Δίνεται βηματικά η διαδικασία ανάπτυξης μιας εφαρμογής με την χρήση της τεχνολογίας Spring Boot και των απαραίτητων βιβλιοθηκών. Περιέχεται το σχήμα ER που δομεί την βάση δεδομένων, η δημιουργία των οντοτήτων με την χρήση JPA και τών διεπαφών που επεκτείνουν Data Manipulation Language (DML) λειτουργίες και παράγουν Ερωτήματα (Queries), το Service layer που περιλαμβάνει την επιχειρησιακή λογική και τους Controllers που διαχειρίζονται τα Request. Το σύνολο αυτων τον μεθοδων έχει παραχθεί σύμφωνα με το documentation της Spring Boot.

Επιπλέον αναφέρονται τεχνολογίες που συντέλεσαν στην γρηγορότερη ανάπτυξη της πλατφόρμας και επικεντρώνονται στην λύση κοινών προβλημάτων, όπως η υπηρεσία διαχείρισης χρηστών, πιστοποίησης και εξουσιοδότηση τους με την παροχή ειδικών πιστοποιητικων (token).

Γίνεται συνοπτική αναφορά στην δημιουργία της διεπαφής χρηστών και τών βιβλιοθηκών που χρησιμοποιήθηκαν και τέλος παρέχονται παραρτήματα για την διαχείριση και την επέκταση της εφαρμογής.

1. Δέντρο Αποφάσεων

Πριν από το τεχνικό μέρος είναι σημαντικό να κατανοήσουμε για ποιους λόγους έχει γίνει επιλογή του δέντρου αποφάσεων, στη συνέχεια θα αναλυθούν μερικοί λόγοι για την κατανόηση του, απαραίτητοι ορισμοί καθώς και περιγραφή των ιδιαίτερων χαρακτηριστικών του αλγορίθμου.

Τα **Δέντρο απόφασης** είναι μοντέλο μηχανικής μάθησης που ανήκει στην κατηγορία επιβλεπόμενης μάθησης, χρησιμοποιείται για ταξινόμησης και παλινδρόμησης, λειτουργούν με αναδρομική κατανομή των δεδομένων σε υποσύνολα, με βάση τις τιμές των χαρακτηριστικών εισόδου και στη συνέχεια κάνοντας προβλέψεις με βάση την τάξη που υπερισχύει (στην ταξινόμηση) ή τη μέση τιμή (σε παλινδρόμηση) της μεταβλητής στόχου σε κάθε υποσύνολο.

Στα πλαίσια αυτής της πτυχιακής έχουν προτιμηθεί τα μοντέλα που βασίζονται στο δέντρο αποφάσεων και στις κατηγορίες Ensemble που τα χρησιμοποιούν για τους παρακάτω λόγους :

Τα μοντέλα CART μπορούν να αναπαρασταθούν οπτικά ως δομές δέντρων, όπου κάθε κόμβος αντιπροσωπεύει μια απόφαση που βασίζεται σε ένα χαρακτηριστικό και τα κλαδιά δείχνουν τα πιθανά αποτελέσματα αυτής της απόφασης. Αυτή η οπτική αναπαράσταση διευκολύνει τους ανθρώπους να κατανοήσουν πώς το μοντέλο κάνει προβλέψεις ακολουθώντας μια σειρά αποφάσεων.

Σε κάθε κόμβο του δέντρου, λαμβάνεται μια δυαδική απόφαση να χωριστούν τα δεδομένα σε δύο υποσύνολα με βάση την τιμή ενός συγκεκριμένου χαρακτηριστικού. Αυτό μιμείται τον τρόπο με τον οποίο οι άνθρωποι λαμβάνουν συχνά αποφάσεις λαμβάνοντας υπόψη έναν παράγοντα κάθε φορά.

Η αναδρομική φύση αυτών των αποφάσεων θυμίζει τον τρόπο με τον οποίο αναλύουμε σύνθετα προβλήματα σε μικρότερα, διαχειρίσιμα κομμάτια.

Οι κανόνες απόφασης που δημιουργούνται από τα μοντέλα CART είναι απλοί στην ερμηνεία. Κάθε διαδρομή από τη ρίζα κόμβο σε έναν κόμβο φύλλου αντιπροσωπεύει μια σειρά από αποφάσεις που οδηγούν σε μια πρόβλεψη. Αυτοί οι κανόνες μπορούν συχνά να μεταφραστούν σε **κατανοητή γλώσσα**, επιτρέποντας στους χρήστες να αντιληφθούν το σκεπτικό του μοντέλου πίσω από τις προβλέψεις του.

Τα μοντέλα CART παρέχουν επίσης ένα μέτρο της σημασίας των χαρακτηριστικών, υποδεικνύοντας ποια χαρακτηριστικά έχουν τη μεγαλύτερη επιρροή στη δημιουργία προβλέψεων. Αυτές οι πληροφορίες μπορούν να βοηθήσουν τους χρήστες να κατανοήσουν ποιοι παράγοντες καθοδηγούν τις αποφάσεις του μοντέλου.

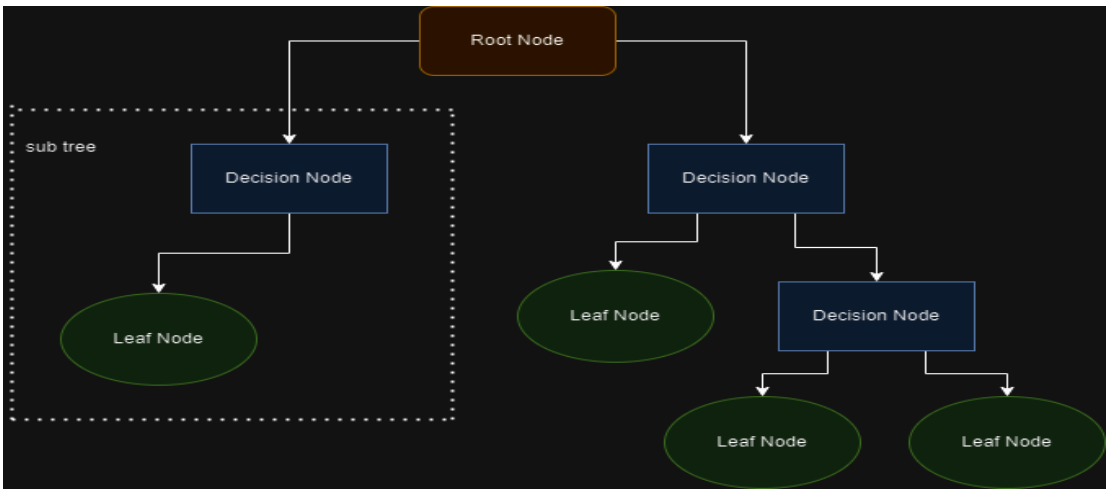
Οι διαχωρισμοί στο δέντρο μπορούν εύκολα να οπτικοποιηθούν, δείχνοντας πώς χρησιμοποιούνται οι τιμές των χαρακτηριστικών για την κατάτμηση των δεδομένων. Αυτή η οπτικοποίηση μπορεί να παρέχει πληροφορίες για τις σχέσεις μεταξύ των χαρακτηριστικών και της μεταβλητής στόχου.

Η έννοια του κλαδέματος, όπου το δέντρο απλοποιείται με την αφαίρεση περιττών κλαδιών, για την αποφυγή της περιττής πολυπλοκότητας. Αυτό αντικατοπτρίζει το πώς συχνά αναζητούμε απλότητα στις διαδικασίες λήψης αποφάσεων.

Τα μοντέλα CART είναι ιδιαίτερα διαισθητικά για εργασίες δυαδικής ταξινόμησης, όπου κάθε κλαδί οδηγεί σε ένα από τα δύο πιθανά αποτελέσματα. Αυτό αντικατοπτρίζει πολλά σενάρια αποφάσεων του πραγματικού κόσμου όπου οι επιλογές συχνά συνοψίζονται σε δύο επιλογές.

1.1 Δέντρα ταξινόμησης και παλινδρόμησης (CART)

Ένα δέντρο αποφάσεων αποτελείται από κόμβους (nodes), κλαδιά (branches) και φυλλώματα (leaves). Οι κόμβοι αντιπροσωπεύουν μια απόφαση ή μια δοκιμή σε ένα συγκεκριμένο χαρακτηριστικό, ενώ τα κλαδιά αντιπροσωπεύουν τα πιθανά αποτελέσματα αυτής της απόφασης. Το δέντρο ξεκινά με έναν μόνο κόμβο που ονομάζεται κόμβος ρίζας και μεγαλώνει με κάθε απόφαση σε πολλαπλά κλαδιά.



Εικόνα 1. *Μορφή δέντρου απόφασης*

Για τη δημιουργία ενός δέντρου αποφάσεων, ο αλγόριθμος αναζητά το καλύτερο χαρακτηριστικό για να χωρίσει τα δεδομένα σε κάθε κόμβο. Το καλύτερο χαρακτηριστικό επιλέγεται με βάση ορισμένα κριτήρια, όπως ο συντελεστής Gini (για προβλήματα ταξινόμησης) ή το μέσο τετράγωνο σφάλμα (για προβλήματα παλινδρόμησης).

Ο αλγόριθμος του δέντρου αποφάσεων αξιολογεί διαφορετικά κριτήρια διαχωρισμού για να προσδιορίσει τον καλύτερο τρόπο για τη διαίρεση των δεδομένων σε υποσύνολα σε κάθε κόμβο. Ο στόχος είναι να βρεθούν οι διαχωρισμοί που δημιουργούν τα πιο ομοιογενή υποσύνολα, που σημαίνει ότι περιέχουν παρόμοια σημεία δεδομένων σε σχέση με τη μεταβλητή στόχο.

Μόλις επιλεχθεί ένα χαρακτηριστικό και ένα σημείο διαχωρισμού, τα δεδομένα χωρίζονται σε δύο ή περισσότερα υποσύνολα σε αυτόν τον κόμβο. Στη συνέχεια, η διαδικασία επαναλαμβάνεται αναδρομικά για κάθε υποσύνολο, δημιουργώντας νέους κόμβους και περαιτέρω διαχωρισμούς μέχρι να ολοκληρωθεί μια συνθήκη διακοπής.

Η διαδικασία ανάπτυξης δέντρων συνεχίζεται μέχρι να ικανοποιηθεί ένα από τα προκαθορισμένα κριτήρια διακοπής. Τα κοινά κριτήρια διακοπής περιλαμβάνουν την επίτευξη ενός μέγιστου βάθους δέντρου, τον ελάχιστο αριθμό δειγμάτων σε έναν κόμβο ή τη μη περαιτέρω βελτίωση στον διαχωρισμό των δεδομένων.

Όταν η διαδικασία αναδρομικής διαίρεσης σταματά, οι τελικοί κόμβοι του δέντρου ονομάζονται κόμβοι φύλλων. Κάθε κόμβος φύλλου αντιπροσωπεύει μια κλάση (στην ταξινόμηση) ή μια προβλεπόμενη τιμή (σε παλινδρόμηση). Η κλάση πλειοψηφίας ή η μέση τιμή της μεταβλητής στόχου σε έναν κόμβο φύλλου χρησιμοποιείται για να γίνουν προβλέψεις για νέα σημεία δεδομένων που εμπίπτουν σε αυτήν την περιοχή.

Τα δέντρα αποφάσεων προσφέρουν πολλά πλεονεκτήματα, όπως η εύκολη ερμηνεία, η ικανότητα χειρισμού τόσο αριθμητικών όσο και κατηγορικών χαρακτηριστικών και αντοχή σε ακραίες τιμές. Ωστόσο, μπορεί να είναι επιρρεπείς σε **Overfitting**, ειδικά όταν το δέντρο αφήνεται να αναπτυχθεί πολύ βαθιά. Για την αντιμετώπιση αυτού του ζητήματος, συχνά χρησιμοποιούνται τεχνικές όπως το κλάδεμα ή η χρήση μεθόδων συνόλου όπως το Random Forest ή Gradient Boosting.

1.2 Κλάδεμα

Το κλάδεμα είναι μια διαδικασία που εφαρμόζεται στα δέντρα ταξινόμησης και παλινδρόμησης (CART) με σκοπό την δημιουργία ενός απλουστευμένου δέντρου που αποδίδει καλύτερα σε άγνωστα δεδομένα. Γενικεύει το δέντρο αφαιρώντας κλαδιά κόμβους από τις άκρες του προς την ρίζα με αναδρομική διαδικασία, αυτό έχεις ως αποτέλεσμα την δημιουργία ενός πιο συμπαγούς δέντρου με καλύτερη προγνωστική απόδοση.

Περιγράφεται η διαδικασία κλαδέματος συνοπτικά.

Το πρώτο βήμα είναι να δημιουργήσετε ένα δέντρο αποφάσεων χρησιμοποιώντας τον αλγόριθμο CART, ο οποίος διαχωρίζει αναδρομικά τα δεδομένα με βάση τις τιμές χαρακτηριστικών για να δημιουργήσει μια δομή δέντρου. Το δέντρο αναπτύσσεται έως ότου κάθε κόμβος φύλλων περιέχει ένα μικρό υποσύνολο των δεδομένων εκπαίδευσης, που ενδεχομένως οδηγεί σε ένα σύνθετο και υπερ-προσαρμοσμένο δέντρο.

Το κλάδεμα ξεκινά από το κάτω μέρος του δέντρου και ανεβαίνει προς τη ριζα κόμβο. Κάθε κόμβος στο δέντρο αξιολογείται για να προσδιοριστεί εάν η αφαίρεση του (και του υποδέντρου του) θα οδηγούσε σε βελτιωμένη απόδοση σε μη ορατά δεδομένα. Αυτή η αξιολόγηση συνήθως περιλαμβάνει την απόδοση από την μέθοδο Pruning.

Η μέτρηση κλαδέματος χρησιμοποιείται για τη σύγκριση της απόδοσης του αρχικού δέντρου (πριν από το κλάδεμα) με την απόδοση μιας κλαδευμένης έκδοσης. Εάν η κλαδευμένη έκδοση αποδίδει τουλάχιστον το ίδιο καλά με την αρχική ή βελτιώνει την απόδοση, ο κόμβος κλαδεύεται. Διαφορετικά, διατηρείται ως έχει.

Η διαδικασία κλαδέματος γίνεται με επαναληπτική διαδικασία από τα άκρα του δέντρου μέχρι την ρίζα, όπου σε κάθε επανάληψη παράγεται η απόδοση του κλαδεμένου δέντρου με το αρχικό χρησιμοποιώντας τον συντελεστή α’ που παράγει το άθροισμα των τετραγωνικών σφαλμάτων.

Μετά την ολοκλήρωση του κλαδέματος το δέντρο που παράγεται αποτελεί ένα πιο απλουστευμένο και συμπαγή δέντρο με μεγαλύτερη ικανότητα γενίκευσης.

Πλέον έχουμε αποτρέψει το δέντρο να γίνει πολύ συγκεκριμένο για τα δεδομένα εκπαίδευσης και του μπορεί να συλλαμβάνει πιο γενικά μοτίβα που μπορούν να εφαρμοστούν σε μη ορατά δεδομένα.

1.3 Ensemble Αλγόριθμοι

Οι μέθοδοι συνόλου (ensemble) στη μηχανική μάθηση ακολουθούν μια διαδικασία λήψης απόφασης με συλλογική προσέγγιση. Αντί να βασιζόμαστε σε ένα μόνο μοντέλο - δέντρο για να κάνουμε προβλέψεις, συγκεντρώνουμε πολλαπλούς ταξινομητές που έχουν προσαρμοστεί σε συγκεκριμένο υποσύνολο δεδομένων (βασικά μοντέλα) για να συνεργαστούν και να κάνουν καλύτερη πρόβλεψη.

Τα πλεονεκτήματα των μεθόδων συνόλου περιγράφονται από τον συνδυασμό των μοντέλων, ως μια ομάδα. Με τη συνεργασία, η ομάδα μπορεί να κάνει πιο ακριβείς προβλέψεις από οποιοδήποτε μεμονωμένο μέλος. Η ομαδική εργασία βοηθά στη μείωση των λαθών και στην πραγματοποίηση πιο αξιόπιστων προβλέψεων. Τα σύνολα επηρεάζονται λιγότερο από ακραίες τιμές ή θορυβώδη σημεία δεδομένων, καθιστώντας τις προβλέψεις τους πιο σταθερές. Οι μέθοδοι συνόλου μπορούν να χρησιμοποιούν διαφορετικούς τύπους μοντέλων ή διαμορφώσεων, παρέχοντας ευελιξία στην επιλογή της καλύτερης προσέγγισης για ένα δεδομένο πρόβλημα.

Ωστόσο υπάρχουν μερικοί περιορισμοί όσον αφορά τις μεθόδους συνόλου. Οι μέθοδοι συνόλου απαιτούν μεγάλο όγκο δεδομένων καθώς και τον συνδυασμό πολλαπλών μοντέλων, τα οποία είναι πιο απαιτητικά υπολογιστικά. Η ερμηνεία των

συνδυασμένων προβλέψεων ενός συνόλου μπορεί να είναι πιο δύσκολη σε σύγκριση με μεμονωμένα μοντέλα.

Με απλά λόγια, οι μέθοδοι συνόλου αφορούν την ομαδική εργασία, όπου πολλά μοντέλα συνεργάζονται για να βελτιώσουν τις προβλέψεις μειώνοντας τα λάθη και αυξάνοντας την ακρίβεια.

Οι δύο βασικοί τύποι μεθόδων συνόλου Bagging και Boosting.

1.3.1 Bagging

Τα μοντέλο του συνόλου (βασικό μοντέλο) λαμβάνει ανεξάρτητα μέρη του προβλήματος. Στη συνέχεια, η πρόβλεψη λαμβάνεται κατα ποσοστό ψήφου του κάθε μοντέλου. Αυτό βοηθά στη μείωση των λαθών και στη βελτίωση της ακρίβειας.

1.3.2 Boosting

Τα μέλη του συνόλου συνεργάζονται με διαδοχικό τρόπο. Κάθε μέλος μαθαίνει από τα λάθη των προηγούμενων μελών και προσπαθεί να τα βελτιώσει. Συνεχίζουν να προσαρμόζουν τις γνώσεις τους μέχρι να φτάσουν σε μια καλή πρόβλεψη.

2. Heart Disease Dataset

Στα πλαίσια αυτής της πτυχιακής έχει γίνει η χρήση τών βιβλιοθηκών scikit-learn [2] numpy [3] , Pandas [4] , matplotlib [5], skl2onnx [6] , onnxruntime [7] καθώς παρέχουν λειτουργίες επεξεργασίας δεδομένων και δημιουργίας μοντέλων μηχανικής μάθησης.

2.1 Τα δεδομένα

Το σύνολο δεδομένων για τις καρδιακές παθήσεις UCI (Πανεπιστήμιο της Καλιφόρνια, Irvine) [1] είναι ένα σύνολο δεδομένων στον τομέα της μηχανικής μάθησης και της ανάλυσης δεδομένων. Περιλαμβάνει δεδομένα που σχετίζονται με τη διάγνωση της καρδιακής νόσου και περιέχει διάφορα χαρακτηριστικά για την πρόβλεψη καρδιακής πάθησης.

Ακολουθεί μια επισκόπηση του συνόλου δεδομένων και των ιδιοτήτων του:

Ονομα μεταβλητης	Τύπος	Τιμές	Περιγραφή
Ηλικία	numeric	σε χρόνια	Η ηλικία του ασθενούς.
Φύλο	numeric	Γυναικα = 0, Ανδρας = 1	Το φύλο του ασθενούς .
Τύπος πόνου στο στήθος	Ordinal	1 = τυπική στηθάγχη, 2 = άτυπη στηθάγχη, 3 = μη στηθάγχη, 4=ασυμπτωματική	Ο τύπος του πόνου στο στήθος που βιώνει ο ασθενής, που κατηγοριοποιείται σε τέσσερις τύπους .
Αρτηριακή πίεση σε ηρεμία	numeric		Η αρτηριακή πίεση ηρεμίας του ασθενούς (σε mmHg).

Χοληστερόλη	numeric		Το επίπεδο χοληστερόλης του ασθενούς (σε mg/dl).
Σάκχαρο αίματος νηστείας	numeric	(> 120 mg/dl = 1, <= 120 mg/dl = 0)	Το επίπεδο σακχάρου αίματος νηστείας του ασθενούς.
Ηλεκτροκαρδιογραφικά αποτελέσματα ηρεμίας	Ordinal	0 = φυσιολογικό, 1 = με ανωμαλία του κύματος ST-T, 2 = εμφάνιση πιθανής ή οριστικής υπερτροφίας της αριστερής κοιλίας	Αποτελέσματα του ηλεκτροκαρδιογραφήματος ηρεμίας, κατηγοριοποιημένα σε τρεις τιμές.
Μέγιστος καρδιακός ρυθμός που επιτυγχάνεται	numeric		Ο μέγιστος καρδιακός ρυθμός του ασθενούς που επιτυγχάνεται κατά τη διάρκεια της άσκησης
Στηθάγχη που προκαλείται από την άσκηση	numeric	1 = ναι, 0 = όχι	Εάν ο ασθενής εμφάνισε στηθάγχη που προκλήθηκε από την άσκηση .
Κατάθλιψη ST που προκαλείται από την άσκηση			Η κατάθλιψη ST που προκαλείται από την άσκηση σε σχέση με την ανάπαυση.
Κλίση του τμήματος ST άσκησης κορυφής	Ordinal	1 = κλίση, 2 = επίπεδο,	Η κλίση του τμήματος ST άσκησης κορυφής .

		3 = κατωφέρεια	
Αριθμός μεγάλων αγγείων	Ordinal	(0-3)	Ο αριθμός των μεγάλων αγγείων χρωματισμένα με ακτινοσκόπηση.
Thal	Ordinal	3 = φυσιολογικό, 6 = σταθερό ελάττωμα, 7 = αναστρέψιμο ελάττωμα	Μια διαταραχή του αίματος που ονομάζεται θαλασσαιμία, που κατηγοριοποιείται σε τρεις τύπους .
Στόχος	numeric	0 = όχι, 1 = ναι	Η παρουσία καρδιακής νόσου.

Ο στόχος της χρήσης αυτού του συνόλου δεδομένων είναι η δημιουργία ενός μοντέλου μηχανικής μάθησης που μπορεί να προβλέψει εάν ένας ασθενής έχει καρδιακή νόσο με βάση τα δεδομένα χαρακτηριστικά. Οι ερευνητές και οι επιστήμονες δεδομένων μπορούν να εφαρμόσουν διάφορους αλγόριθμους ταξινόμησης σε αυτό το σύνολο δεδομένων για να αναπτύξουν μοντέλα πρόβλεψης και να μελετήσουν τις σχέσεις μεταξύ των χαρακτηριστικών και της παρουσίας καρδιακών παθήσεων.

Είναι σημαντικό να σημειωθεί ότι το σύνολο δεδομενων προέρχεται από το Cleveland dataset που είναι μια προ επεξεργασμένη μορφή του heart disease dataset με σκοπό την χρήση του σε μοντέλα μηχανικής μάθησης. Για την περαιτέρω κατανόηση των χαρακτηριστικών που δεν έχουν χρησιμοποιηθεί στα πλαίσια αυτής της πτυχιακής μπορείτε να ανατρέξετε στην πηγή του UCI datasets Heart Disease [1].

Εισαγωγή των δεδομένων με τη την χρήση της βιβλιοθήκης Pandas και δημιουργία Dataframe:

```
df = pd.read_csv("../datasets/heart/processed.cleveland.data",
header=None)
df.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	hd
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	0
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	108.0	1.0	1.5	2.0	3.0	3.0	2
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	129.0	1.0	2.6	2.0	2.0	7.0	1
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	0

Το Dataframe έχει **303 γραμμές** και **14 στήλες**.

2.2 Προετοιμασία δεδομένων

Δήλωση ονομάτων χαρακτηριστικών στις στήλες του Dataframe:

```
df.columns =
["age","sex","cp","trestbps","chol","fbs","restecg","thalach",
 "exang","oldpeak","slope","ca","thal", "hd"]
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	hd
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	0
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	108.0	1.0	1.5	2.0	3.0	3.0	2
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	129.0	1.0	2.6	2.0	2.0	7.0	1
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	0
4	41.0	0.0	2.0	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	0

2.1.1 Missing values

Τα χαρακτηριστικά που έχουν null τιμές είναι το ca και thal, με τον παρακάτω κώδικα αφαιρούμε τις γραμμές που έχουν τις ελλιπείς τιμές.

```
df_no_missing = df.loc[(df["ca"] != "?") & (df["thal"] != "?")]
df_no_missing
```

με την αφαίρεση των null τιμών οι γραμμές του Dataframe είναι 297.

2.1.2 Dependent and Independent values

Ξεχωρίζουμε τις Εξαρτημένες από ανεξάρτητες μεταβλητές.

```
X = df_no_missing.drop('hd', axis=1).copy()
y = df_no_missing["hd"].copy()
```

Η μεταβλητη στόχος (target label) έχει 4 τιμές, στη δημιουργία αυτού του μοντέλου θα λάβουμε υπόψη τον στόχο ως δυαδική τάξη, με τιμή 0 για την απουσία καρδιακής νόσου και 1 για την ύπαρξη καρδιακής νόσου.

```
y_not_zero_index = y>0
y[y_not_zero_index] = 1
target_labels = ["No HD","Yes HD"]
```

2.1.3 One-hot encoding

Το One-hot encoding είναι ένας τύπος κωδικοποίηση που χρησιμοποιείται για την αναπαράσταση κατηγορικών μεταβλητών ή χαρακτηριστικών ως δυαδικών διανυσμάτων. Εφαρμόζεται συνήθως στη μηχανική μάθηση για κατηγορικά δεδομένα, επειδή τα περισσότερα μοντέλα απαιτούν αριθμητικές εισόδους.

Στην κωδικοποίηση one-hot, κάθε μοναδική κατηγορία σε ένα κατηγορικό χαρακτηριστικό μετατρέπεται σε ένα δυαδικό διάνυσμα που έχει το ίδιο μήκος με τον αριθμό των μοναδικών κατηγοριών. Το δυαδικό διάνυσμα περιέχει όλα τα μηδενικά εκτός από ένα μεμονωμένο στοιχείο, το οποίο έχει οριστεί σε 1 για να αντιπροσωπεύει την παρουσία μιας συγκεκριμένης κατηγορίας.

Η κωδικοποίηση One-hot χρησιμοποιείται ευρέως, σε εργασίες όπως η ταξινόμηση, όπου οι κατηγορικές μεταβλητές πρέπει να μετατραπούν στις αριθμητικές τους αναπαραστάσεις ως είσοδοι σε μοντέλα. Μερικά από αυτά τα μοντέλα είναι η λογιστική παλινδρόμηση, τα δέντρα αποφάσεων ή τα νευρωνικά δίκτυα.

```
df_encoded = pd.get_dummies(X, columns=["cp","restecg","thal","slope"])
```

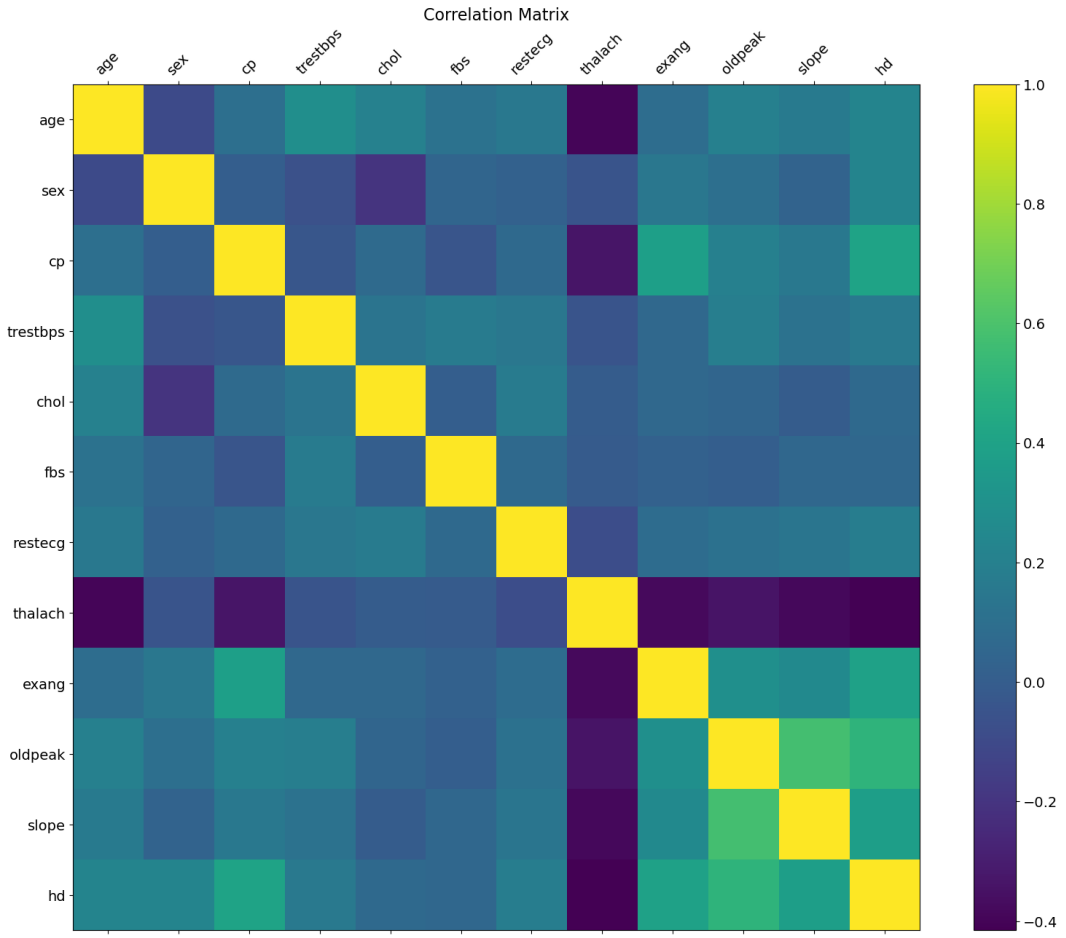
Με τον παραπάνω κώδικα μετατρέπουμε τις ονοματικές τιμες σε αριθμητικές, τα χαρακτηριστικά που ανήκουν σε κατηγορίες είναι cp, restecg, thal, slop.

2.1.2 Correlation Matrix

Καλό είναι να έχουμε μια οπτικοποίηση των συσχετίσεων των μεταβλητών για περεταίρω συμπεράσματα.

```
plt.figure(figsize=(19, 15))
plt.matshow(df.corr(), fignum=f.number)
plt.xticks(range(df_no_missing.select_dtypes(['number']).shape[1]),
df.select_dtypes(['number']).columns, fontsize=14, rotation=45)
plt.yticks(range(df_no_missing.select_dtypes(['number']).shape[1]),
df.select_dtypes(['number']).columns, fontsize=14)
cb = plt.colorbar()
cb.ax.tick_params(labelsize=14)
plt.title('Correlation Matrix', fontsize=16);
```

Ο παραπάνω κώδικας μας δίνει τον πίνακα συσχετίσεων σε **heatmap**:



Εικόνα 1. *Ο Πίνακας συσχετίσεων των παραμέτρων*

Διαχωρισμός σε δεδομένα εκπαίδευσης και δεδομένα ελέγχου. Ο διαχωρισμός γίνεται με σκοπό να δημιουργήσουμε άγνωστα δεδομένα για το εκπαιδευμένο μοντέλο με σκοπό την εκτίμηση της απόδοσης του στα δεδομένα ελέγχου και την ικανότητα γενίκευσης.

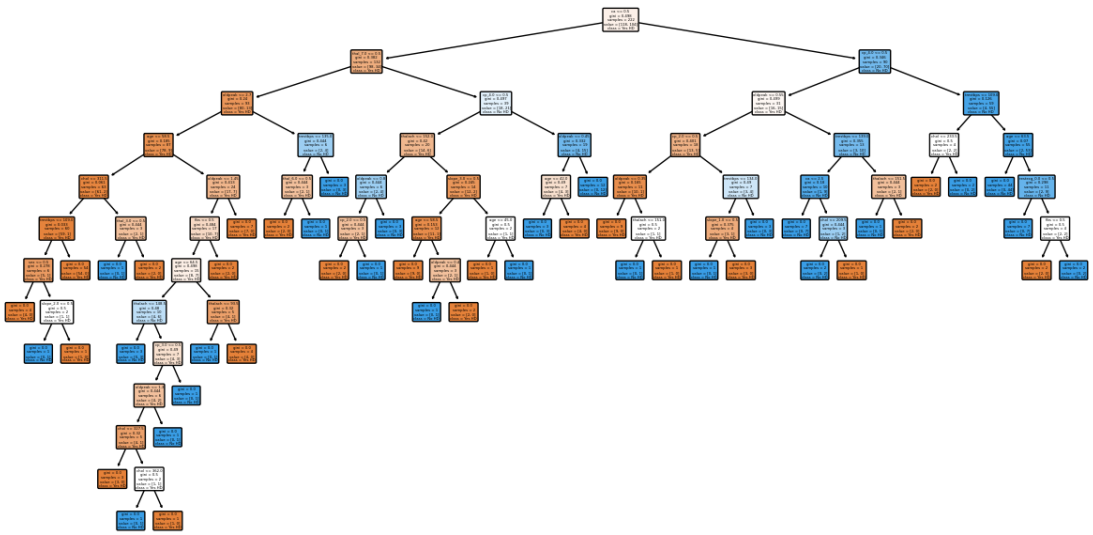
```
X_train, X_test, Y_train, Y_test = train_test_split(df_encoded, y,
train_size = 0.2,random_state=42)
```

2.3 DecisionTreeClassifier

θα χρησιμοποιήσουμε τον ταξινομητη για να δημιουργήσουμε ένα αρχικό δέντρο. Χρησιμοποιώντας την βιβλιοθήκη matplotlib θα δημιουργήσουμε ένα plot του δέντρου για να οπτικοποιήσουμε τη μορφή και να βγάλουμε συμπεράσματα.

```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
from sklearn.tree import DecisionTreeClassifier

clf_dt = DecisionTreeClassifier(random_state=(42))
clf_dt = clf_dt.fit(X_train, Y_train)

plt.figure(figsize=(15, 7.5))
plot_tree(clf_dt,
          filled=True,
          rounded=True,
          class_names=target_labels,
          feature_names=df_encoded.columns)
```

Το δεντρο που παράγεται είναι ένα **περίπλοκο** και **υπερ-προσαρμοσμένο** δέντρο.



Εικόνα 2. *Το πλήρες δέντρο που λαμβάνουμε από τον ταξινομητή*

Στη συνέχεια θα απλουστεύσουμε το δέντρο με τη μέθοδο κλαδέματος.

2.3.1 Cost complexity Pruning

Το κριτήριο κόστος πολυπλοκότητας (Cost complexity Pruning), που συχνά δηλώνεται ως α’ (άλφα), ελέγχει την αντιστάθμιση μεταξύ της πολυπλοκότητας του δέντρου και της προσαρμογής των δεδομένων εκπαίδευσης. Καθορίζει την ποσότητα κλαδέματος που εφαρμόζεται στο δέντρο. Μεγαλύτερες τιμές άλφα οδηγούν σε πιο επιθετικό κλάδεμα, οδηγώντας σε απλούστερα δέντρα με λιγότερα κλαδιά. Μικρότερες τιμές του άλφα επιτρέπουν στο δέντρο να συλλαμβάνει περισσότερες λεπτομέρειες από τα δεδομένα εκπαίδευσης.

Το κριτήριο κόστους-πολυπλοκότητας είναι ένα μαθηματικό μέτρο που χρησιμοποιείται για την αξιολόγηση της ποιότητας ενός δέντρου. Συνδυάζει το ποσοστό εσφαλμένης ταξινόμησης (για ταξινόμηση) ή το μέσο τετράγωνο σφάλμα (για παλινδρόμηση) με έναν όρο ποινής που αντιπροσωπεύει την πολυπλοκότητα του δέντρου.

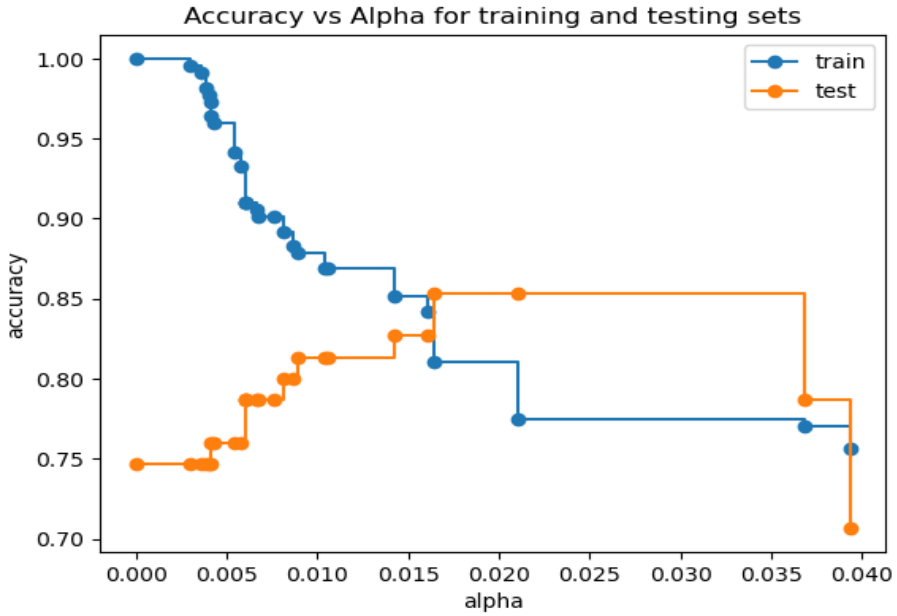
Γενικός τύπος: Κόστος-Πολυπλοκότητα = Ποσοστό εσφαλμένης ταξινόμησης (ή MSE) + α * Πολυπλοκότητα δέντρου

Βρίσκουμε το α’ που ανταποκρίνεται καλύτερα για την εκπαίδευση και για την δοκιμή του μοντέλου.

```
path = clf_dt.cost_complexity_pruning_path(X_train, Y_train)
ccp_alphas = path.ccp_alphas #extracting different values for alphas
ccp_alphas = ccp_alphas[:-1] #exclude maximum value from alphas

clf_dts = [] # creating an array to put decision trees

for ccp_alpha in ccp_alphas:
    clf_dt = DecisionTreeClassifier(random_state=(0),
ccp_alpha=ccp_alpha)
    clf_dt.fit(X_train, Y_train)
    clf_dts.append(clf_dt)
```

```
train_scores = [clf_dt.score(X_train, Y_train) for clf_dt in clf_dts]
test_scores = [clf_dt.score(X_test, Y_test) for clf_dt in clf_dts]

fig, ax = plt.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs Alpha for training and testing sets")
ax.plot(ccp_alphas,train_scores, marker="o", label="train",
drawstyle="steps-post")
ax.plot(ccp_alphas,test_scores, marker="o", label="test",
drawstyle="steps-post")
ax.legend()
plt.show()
```

Χρησιμοποιουμε τον παραπάνω κώδικα για να βρούμε ποιο α' ανταποκρίνεται καλύτερα και για την εκπαίδευση και για τον έλεγχο του μοντέλου.



Εικόνα 3. *Η ακριβεια του α' σε σχεση με το train και τεστ set.*

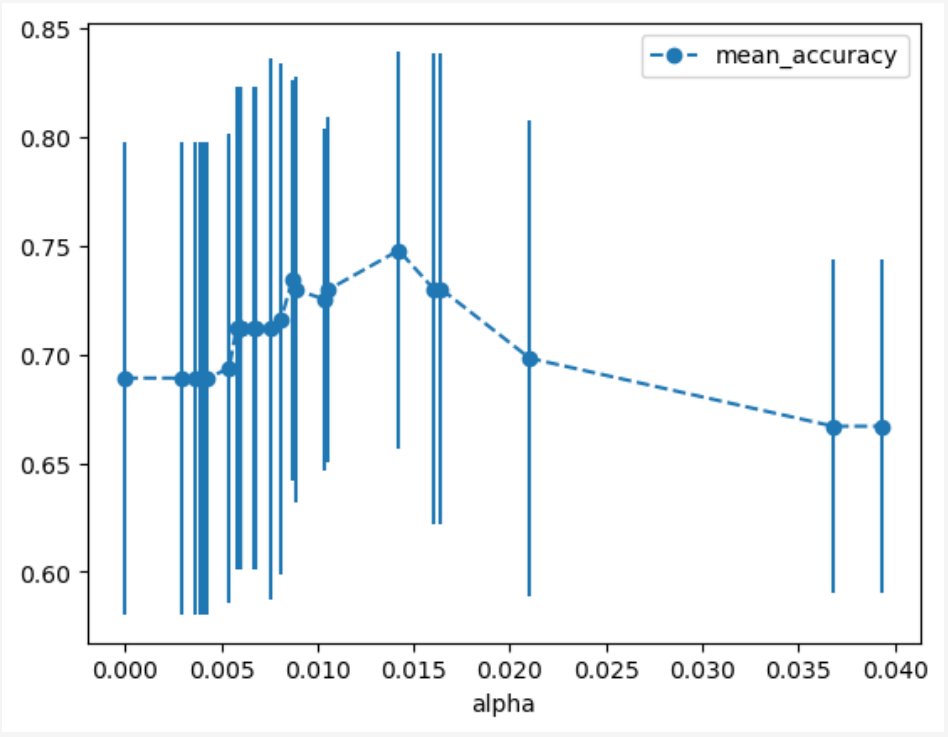
```
import numpy as np
alpha_loop_values = []
```

```
for ccp_alpha in ccp_alphas:
    clf_dt = DecisionTreeClassifier(random_state=(42),
ccp_alpha=ccp_alpha)
    scores = cross_val_score(clf_dt, X_train, Y_train, cv=5)
    alpha_loop_values.append([ccp_alpha, np.mean(scores),
np.std(scores)])

alpha_results = pd.DataFrame(alpha_loop_values, columns=['alpha',
'mean_accuracy', 'std'])
```

Για κάθε ένα α’ που έχουμε κρατήσει στον πίνακα ccp_alphas καταγράφουμε την απόδοση δημιουργώντας ταξινομητές και κρατάμε την μέση απόδοση και την απόκλιση στον πίνακα alpha_loop_values.

Το ιδανικό α’ βρίσκεται μεταξύ των τιμών 0.014 και 0.015. Με τον παρακάτω κώδικα βρίσκουμε και κρατάμε το ιδανικό α΄ στην ματαβλιτή ideal_ccp_alpha.

```
alpha_results.plot(x='alpha',
                   y='mean_accuracy',
                   yerr='std',
                   marker='o',
                   linestyle='--')

ideal_ccp_alpha = alpha_results[(alpha_results['alpha'] > 0.014) &
(alpha_results['alpha'] < 0.015)]['alpha']
ideal_ccp_alpha = ideal_ccp_alpha.astype(np.float64).iloc[0]
print(ideal_ccp_alpha)
```



Εικόνα 4. *Βρισκουμε το α' που δίνει τη καλύτερη ακρίβεια.*

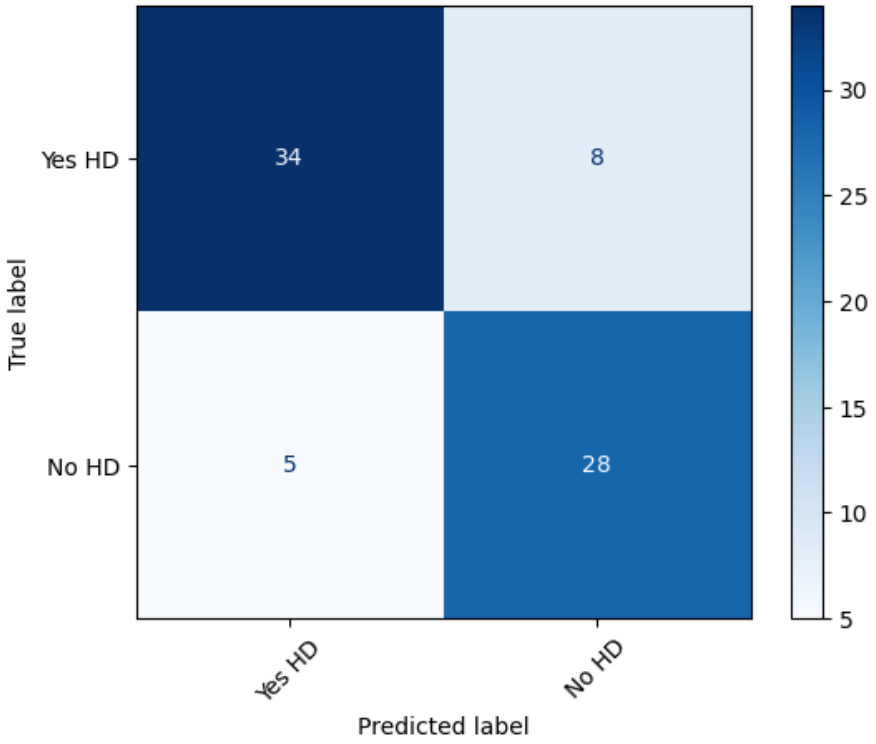
Το καλύτερο ccp_a: **`0.014224751066856332`**

```
clf_dt_pruned = DecisionTreeClassifier(random_state=(42),
ccp_alpha=ideal_ccp_alpha)
clf_dt_pruned = clf_dt_pruned.fit(X_train, Y_train)
predictions = clf_dt_pruned.predict(X_test)
```

Στη συνέχεια δημιουργούμε το confusion matrix σύμφωνα με το βέλτιστο α'.

```
clf_dt_pruned = DecisionTreeClassifier(random_state=(42),
ccp_alpha=ideal_ccp_alpha)
clf_dt_pruned = clf_dt_pruned.fit(X_train, Y_train)
predictions = clf_dt_pruned.predict(X_test)
cm = confusion_matrix(Y_test, predictions,
labels=clf_dt_pruned.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=target_labels)
disp.plot(cmap=plt.cm.Blues, xticks_rotation=45)
```

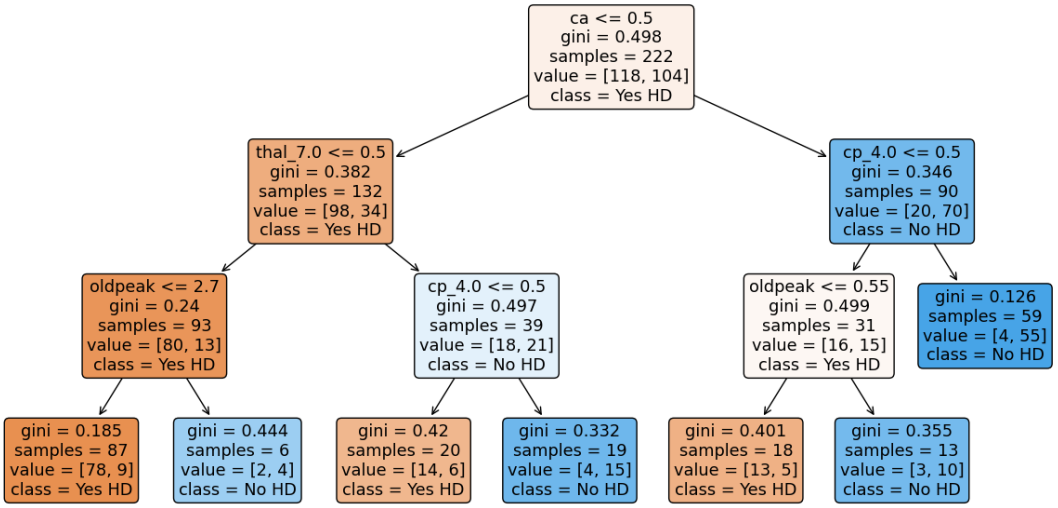
```
plt.show()
```



Εικόνα 5. *Το confusion matrix χρησιμοποιώντας το α΄ .*

Παρουσιάζουμε το διάγραμμα του δέντρου μετά τη διαδικασία κλαδέματος.

```
from sklearn.tree import plot_tree
plt.figure(figsize=(15, 7.5))
plot_tree(clf_dt_pruned,
          filled=True,
          rounded=True,
          class_names=target_labels,
          feature_names=df_encoded.columns)
```



Εικόνα 6.*Το δέντρο μετα τη διαδικασία Pruning.*

```
from sklearn.model_selection import cross_val_score
cvs = cross_val_score(clf_dt_pruned, X, y, cv=10)
np.average(cvs)
```

```
Accuracy: 0.7239080459770115
```

Αν και το μοντέλο αποδίδει πλεον καλύτερα σε άγνωστα δεδομένα με ποσοστό 72%, δεν είναι εύκολο να μεγιστοποιήσουμε την επιτυχία μόνο με την χρήση ενός ταξινομητή CART. Ωστόσο μπορούμε να χρησιμοποιήσουμε τις μεθόδους συνόλου για να επιτύχουμε καλύτερα αποτελέσματα.

Είναι επίσης σημαντικό να αναφέρουμε ότι με την χρήση των μεθοδων συνόλου δεν είναι αναγκαίο να χρησιμοποιήσουμε τη μέθοδο κλαδέματος καθώς ένα σύνολο ταξινομητών έχουν εκπαιδευτεί σε ένα **κομμάτι του αρχικού συνόλου δεδομένων** (Bootstraped dataset) και το **συμπέρασμα** (Inference) παράγεται κατά ποσοστό ψήφων των ταξινομητών.

2.4 RandomForestClassifier

Γίνεται μια δοκιμή με την παράμετρο ccp_a για σύγκριση αποτελεσμάτων.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
base_rf = RandomForestClassifier(
    n_estimators = 1000,
    random_state = 1,
    criterion = "gini",
    ccp_alpha=0.0142247,
    max_samples=0.11,
)
base_rf.fit(X_train, Y_train)

acc = base_rf.score(X_test,Y_test)*100
print("Random Forest Algorithm Accuracy Score : {:.2f}%".format(acc))
print("Cross Validation Score: ",np.mean(cross_val_score(base_rf,
X_test, Y_test, cv=5)))


acc = base_rf.score(X_test,Y_test)*100
```

Random Forest Algorithm Accuracy with ccp_a
Cross Validation Score: 0.8231382978723405

2.5 RandomizedSearchCV

Χρησιμοποιούμε μια μεθοδο για hyperparameter tuning από τη βιβλιοθήκη sklearn. Μας δίνεται η δυνατότητα να δώσουμε ένα σύνολο - διάστημα παραμέτρων για να γίνουν οι δοκιμές στον μοντέλο.

```
from sklearn.model_selection import RandomizedSearchCV
from pprint import pprint

n_estimators = [int(x) for x in np.linspace(start = 1800, stop = 1900,
num = 10)]
criterion = ['gini','entropy']
max_features = ['sqrt','log2']
bootstrap = [True]
ccp_alpha = [ 0, 0.0142247]

max_depth = [int(x) for x in np.linspace(38, 42, num = 2)]
min_samples_split = [2, 5, 10, 12, 13, 14, 15]
min_samples_leaf = [1, 2, 4, 6]

#min_impurity_decrease = [0.03,0.1,0.2,0.4]

max_samples= [0.6, 0.7, 0.8]
oob_score = [True]
class_weight = ["balanced","balanced_subsample"]

random_grid = {'n_estimators': n_estimators,
               'criterion':criterion,
               'max_features': max_features,
               'bootstrap': bootstrap,
               #'ccp_alpha': ccp_alpha,
               'max_depth' : max_depth,
               'min_samples_split':min_samples_split,
               'min_samples_leaf':min_samples_leaf,
               #'min_impurity_decrease':min_impurity_decrease,
               'max_samples': max_samples,
               'oob_score' : oob_score,
               'class_weight' : class_weight
              }
pprint(random_grid)
```

Στον παραπάνω κώδικα δηλώνονται το πλέγμα (grid) των παραμέτρων που θα RandomizedSearchCV για να βρεθεί ο καλύτερος συνδυασμός. Παρέχεται η ικανότητα δήλωσης ενός μήκους τιμών σε κάθε μεταβλητή.
Λαμβάνουμε τις βέλτιστες παραμέτρους με τη μέθοδο best_params_ του αντικειμένου rf_randomCV που δημιουργίσαμε.

```
print(rf_randomCV.best_params_)

#Best params for RandomCV

{'oob_score': True, 'n_estimators': 1800, 'min_samples_split': 15,
'min_samples_leaf': 6, 'max_samples': 0.6, 'max_features': 'log2',
'max_depth': 38, 'criterion': 'gini', 'class_weight': 'balanced',
'bootstrap': True}
```

Ακολουθεί μια σύντομη περιγραφή των καλύτερων παραμέτρων [2].

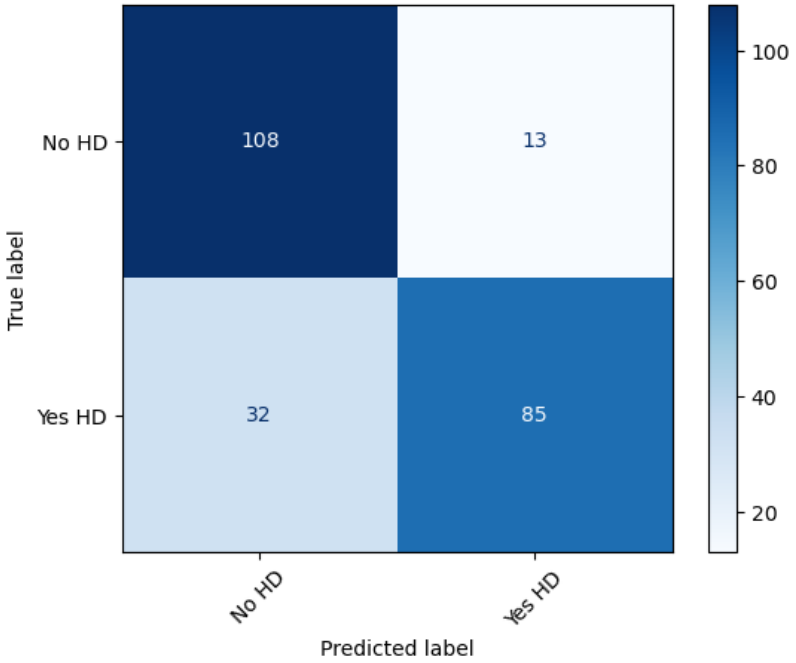
n_estimators 1800 , το πλήθος των ταξινομητών CART που χρησιμοποιήθηκαν.
min_samples_split 15, το ελάχιστο πλήθος δειγμάτων για τον διαχωρισμό εσωτερικού κόμβου.
min_samples_leaf 6, το ελαχιστον πλήθος δειγμάτων που μπορεί να βρεθεί σε ένα φύλλο, βοηθάει στην εξομάλυνση του μοντέλου ειδικά στην παλινδρόμηση.
max_samples 0.6, Όταν γίνεται χρήση του Bootstrap, περιγράφει το ποσοστό των δειγμάτων που θα χρησιμοποιηθεί για την εκπαίδευση κάθε ταξινομητή.
max_features 'log2', Το μέγιστο πλήθος χαρακτηριστικών, όταν ψάχνουμε για τον καλύτερο διαχωρισμό, δίνεται από max_features = log2(n_features).
max_depth 38, Το μέγιστο βάθος του δέντρου, αν δεν δοθεί τιμή οι κόμβοι διαχωρίζονται μέχρι τα φύλλα να περιέχουν μια κλάση ή μέχρι να περιέχουν λιγότερα δείγματα από το min_samples_split.
criterion, 'gini', Το κριτήριο ως συνάρτηση που προσδιορίζει τον καλύτερο διαχωρισμό.
class_weight 'balanced', Η βαρύτητα που προσδιορίζει την κάθε κλάση.

bootstrap 'True', στην περίπτωση που είναι ενεργό, χρησιμοποιεί δείγματα του αρχικούν dataset για την εκπαίδευση των ταξινομητών.

Λαμβάνουμε το Cross Validation Score από τον ταξινομητή που δώσαμε τις καλύτερες παραμέτρους. Με την παράμετρο param_distribution δεσμεύουν τις καλύτερες παραμέτρους για να χρησιμοποιηθούν.

```
rf = RandomForestClassifier()
rf_randomCV = RandomizedSearchCV(
    estimator = rf,
    param_distributions = random_grid,
    n_iter = 100,
    cv = 3,
    verbose=2,
    random_state=1,
    n_jobs = -1,
    return_train_score = True
)
rf_randomCV.fit(X_train, Y_train)
print("best score: ",rf_randomCV.best_score_)
```

best score: 0.8815789473684211



Εικόνα 6. *Το confusion matrix από το RandomForest Classifier.*

Έχοντας τα αποτελέσματα από το pruned tree, το RandomForest με χρηση ccp_a και το καλύτερο ταξινομητή του RandomizedSearchCV μπορούμε να κάνουμε σύγκριση αποτελεσμάτων.

Classifier	**Accuracy**
Pruned Tree	**72%**
RandomForest with ccp_a parameter	**82%**
RandomizedSearchCV Best Estimator	**88%**

Αν και έχουμε αυξήσει το ποσοστό επιτυχίας είναι σημαντικό να αναφέρουμε ότι στις περιπτώσεις των **false negative** το ποσοστό έχει αυξηθεί, μελλοντικά θα πρέπει να λάβουμε υπόψιν, δηλαδή τις περιπτώσεις όπου ο πελάτης έχει καρδιοπάθεια αλλά έχει ταξινομηθεί λανθασμένα.

2.3 Open Neural Network Exchange (Onnx)

Σκοπός μας είναι να εξάγουμε το μοντέλο για να το χρησιμοποιήσουμε σε μια διαδικτυακή υπηρεσία. Υπάρχουν διάφορες επιλογές για την λύση αυτου του θέματος. Το onnx παρέχει δυνατότητα κωδικοποίησης του μοντέλου έτσι ώστε να είναι συμβατό με διαφορετικές πλατφόρμες και γλωσσες προγραμματισμου.

Η βιβλιοθήκη για το serialisation του μοντέλου είναι η skl2onnx που περιέχει converter για την βιβλιοθήκη sklearn.

Ο παρακάτω κώδικα μας δίνει το μοντέλο σε μορφή τένσορα (Tensor) για την χρήση του σε διαφορετικό περιβάλλον.

```
# Convert into ONNX format
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType
initial_type = [('float_input', FloatTensorType([None,
X_train.shape[1]]))]
onx = convert_sklearn(best_estimator, initial_types=initial_type)
with open("rf_heart.onnx", "wb") as f:
    f.write(onx.SerializeToString())
```

Οπως φαίνεται στον παραπανω κώδικα δηλωνεται ο τύπος των χαρακτηριστικών που εισάγουμε “Float_input” για τον τένσορα, στη συνέχεια χρησιμοποιούμε την μέθοδο convert_sklearn με παράμετρο τον ταξινομητή, στη συνέχεια ανοίγουμε μια ροη δεδομενων (stream) για να αποθηκεύσουμε το μοντέλο τοπικά.

Ο παρακάτω κώδικας χρησιμοποιεί το onnxruntime για να φορτώσει το μοντέλο και να κάνουμε μια δοκιμή της λειτουργίας του.

Αρχικά πρέπει να δεδομένα που προορίζονται για έλεγχο από το αρχικό dataset να υποστούν την ίδια μετατροπή One-hot encoding για να μπορεί να γίνει inference, καθώς το μοντέλο ζητάει 22 feature.

```
import onnxruntime as rt
import numpy as np

xiloc = X.iloc[-1]
yiloc = y.iloc[-1]

test =
xiloc#[63.0,1.0,1.0,145.0,233.0,1.0,2.0,150.0,0.0,2.3,3.0,0.0,6.0]
input_data = pd.DataFrame(test)
input_data = input_data.T
input_data.columns = X.columns
input_data

categorical_features = ["cp", "restecg", "thal", "slope"]
selected_input_data = input_data[categorical_features]

# Convert numerical categories to strings
selected_input_data["cp"] = selected_input_data["cp"].astype(str)
selected_input_data["thal"] = selected_input_data["thal"].astype(str)
selected_input_data["restecg"] =
selected_input_data["restecg"].astype(str)
selected_input_data["slope"] = selected_input_data["slope"].astype(str)


# Load the saved encoder from the file
with open('encoder.pkl', 'rb') as file:
    encoder2 = pickle.load(file)

#encoded_data =
encoder2.fit_transform(input_data.replace(mapping_dict))
encoded_data = encoder2.transform(selected_input_data)

# Repalce the values in original input
encoded_df = pd.DataFrame(encoded_data,
columns=encoder.get_feature_names_out(categorical_features))
```

```
data_encoded =
pd.concat([input_data.drop(columns=categorical_features), encoded_df],
axis=1)

input_data = data_encoded.loc[0, :].values.flatten().tolist()

sess = rt.InferenceSession("rf_heart.onnx",
providers=["CPUExecutionProvider"])
# Ensure the input data has the expected shape and data type
input_name = sess.get_inputs()[0].name
# Perform inference
outputs = sess.run(None, {input_name: [input_data]})

# Retrieve the output
output_name = sess.get_outputs()[0].name
predictions = outputs[0]

# Print the predictions
print(predictions)
```

Αφότου έχει γίνει η δοκιμή και είναι επιτυχής μπορούμε να συνεχίσουμε στην μεταφόρτωση του μοντέλου στην διαδικτυακή υπηρεσία πρόβλεψης που θα δημιουργήσουμε. Η διαδικτυακή υπηρεσία θα αναλυθεί στο επόμενο κεφάλαιο.

Η προσέγγισή μας όσον αφορά την την χρήση μοντέλων μηχανικής μάθησης σε διαφορετικές πλατφόρμες είναι δύο, είτε θα πρέπει να βρούμε έναν native compiler για να χρησιμοποιήσει το μοντέλο άμεσα στην γλώσσα προγραμματισμού που χρησιμοποιούμε στην υπηρεσία, είτε θα επιλέξουμε να δημιουργήσουμε μια υπηρεσία στην γλώσσα που υλοποιήθηκε το μοντέλο και να εκθέσουμε την πρόβλεψη μέσω HTTP. Η naitive προσέγγιση αν και παρέχει πολλά πλεονεκτήματα είναι αρκετά χρονοβόρα καθώς πρέπει να αναπτυχθεί το περιβάλλον για την χρήση του μοντέλου. Στα πλαίσια της πτυχιακής για ευκολία και καλύτερη κατανόηση υλοποιήθηκε υπηρεσία που παρέχει μια διεπαφή για την πρόβλεψη.

Με τη χρήση της βιβλιοθήκης open neural network exchange μας δίνεται η δυνατότητα μετατροπής του μοντέλου ωστε να είναι συμβατό με πολλές τεχνολογίες, έτσι μπορούμε μελλοντικά να δοκιμάσουμε και το μοντέλο σε java καθώς παρέχεται αντίστοιχη υλοποίηση.

Στον παρακάτω κώδικα δίνεται ένα παράδειγμα χρήσης του onnx runtime

```
var env = OrtEnvironment.getEnvironment();
var session = env.createSession("model.onnx",new
OrtSession.SessionOptions());
```

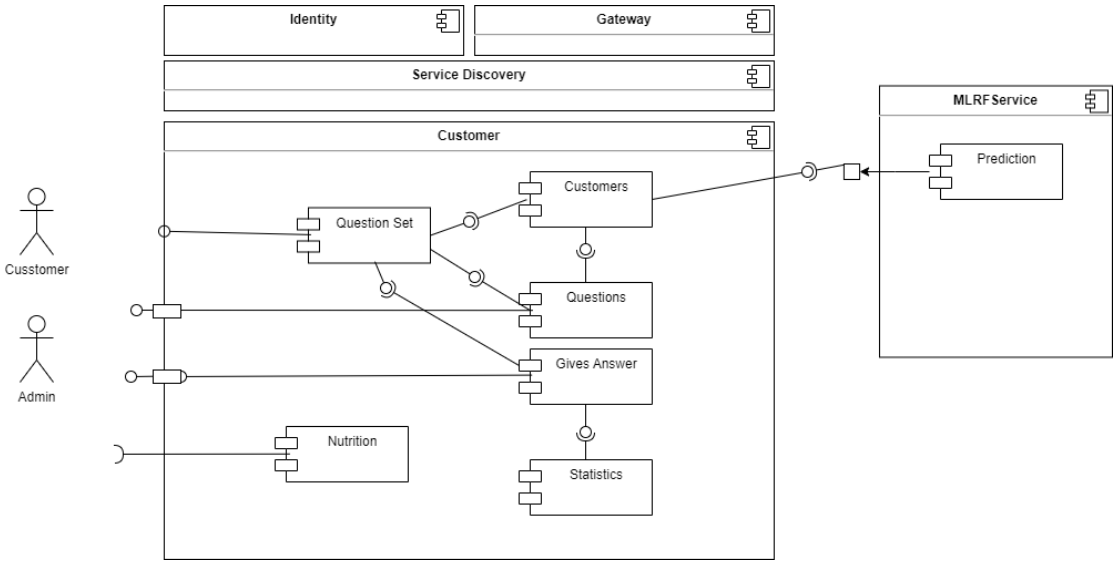
3. Σχεδίαση - Υλοποίηση της Εφαρμογής

3.1 Διαδικτυακή Υπηρεσία Συλλογής δεδομένων

3.1.1 Αρχιτεκτονική και δομή έργου

Η εργασία αποτελείται από τρία βασικά μέρη, τις διαδικτυακές υπηρεσίες που εξυπηρετούν τις υπάρχουσες διεπαφές, την εφαρμογή διάδρασης και την υπηρεσία προβλεψης, καθώς η περισσότερη λογική περιλαμβάνεται στις διαδικτυακές υπηρεσίες θα δώσουμε περισσότερη έμφαση σε αυτές.

Παρουσιάζεται ένα Component Diagram των διαδικτυακών υπηρεσιών.



Εικόνα 7. *Το Component Diagram των διαδικτυακών υπηρεσιών.*

3.1.2 Maven Bom (Bill of materials)

Ως bill of materials δηλώνουμε το project object model (pom) όπου μπορούν να κληρονομήσουν μια διαμόρφωση (configuration) ένα σύνολο από maven projects.

Στο dependencyManagement δηλώνουμε τις βιβλιοθήκες που θα είναι κοινές στις υπομονάδες - applications που περιλαμβάνονται στο Project.

```
<dependencyManagement>
    <dependencies>
```

```
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>${spring.boot.dependencies.version}</version>
        <scope>import</scope>
        <type>pom</type>
      </dependency>
      <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-parent</artifactId>
        <version>${spring.cloud-version}</version>
        <scope>import</scope>
        <type>pom</type>
    </dependency>
    </dependencies>
  </dependencyManagement>
```

Στο παρακάτω κομμάτι κώδικα δηλώνουμε τα microservices με αυτόν τον τρόπο, έτσι ώστε το maven να αναγνωρίσει ότι είναι φωλιασμένα (nested) project και να τα επεξεργαστεί ανάλογα.

```
  <modules>
    <module>customer</module>
    <module>eureka-server</module>
    <module>Gateway</module>
  </modules>
```

Στη συνέχεια δηλώνεται το το build της εφαρμογής που περιλαμβάνει το Spring boot maven plugin και στον annotation processor προσθέτουμε το mapstruct processor για να αναγνωρίζεται το mapper annotation.

```
<build>
  <pluginManagement>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <version>${spring.boot.maven.plugin.version}</version>
        <configuration>
          <mainClass>${start.class}</mainClass>
          <layout>ZIP</layout>
        </configuration>
        <executions>
          <execution>
            <goals>
              <goal>java</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.5.1</version>
        <configuration>
            <source>1.8</source>
            <target>1.8</target>
            <annotationProcessorPaths>
                <path>
                    <groupId>org.mapstruct</groupId>
                    <artifactId>mapstruct-processor</artifactId>
                    <version>1.5.3.Final</version>
                </path>
            </annotationProcessorPaths>
        </configuration>
    </plugin>
    </plugins>
  </pluginManagement>
```

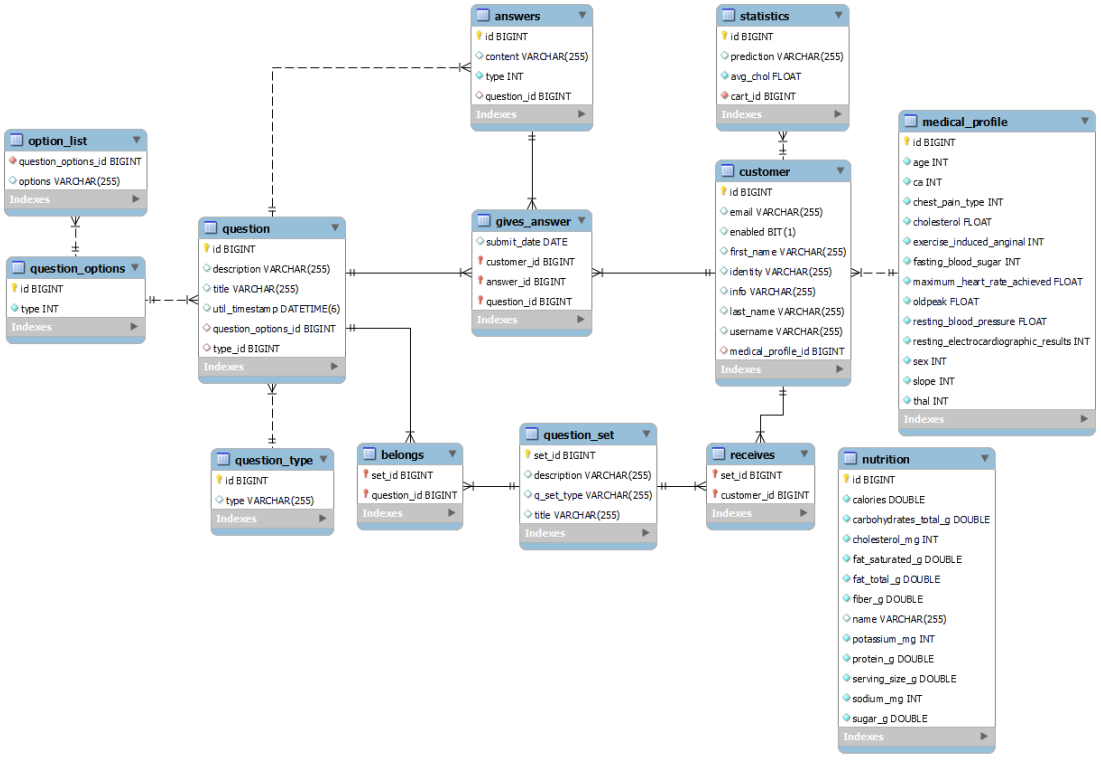
```
</build>
```

3.2 Microservices

3.2.1 Customer Service Application

Ξεκινώντας από την δημιουργία των οντοτήτων έχει χρησιμοποιηθεί η τεχνολογία JPA για object relational mapping για να επιταχυνθεί η διαδικασία ανάπτυξης του λογισμικού.

Entity Relation Diagram



Εικόνα 8. *ER διάγραμμα του Customer Service.*

Η λειτουργικότητα στο επίπεδο εξυπηρετητή (Service) περιγράφεται παρακάτω με την παρουσίαση των διεπαφών (Interfaces).

Θα παρουσιαστεί ως παράδειγμα ένα από τα σημαντικά μέρη της εφαρμογής που περιγράφει την διαδικασία ανάπτυξης των υπηρεσιών. Ως πρωταρχικό συστατικό θα χαρακτηρίζεται η οντότητα των ερωτηματολογίων, καθώς αυτό περιλαμβάνει τις ερωτήσεις και τους χρήστες και τα συνδέει σε αυτόνομα σύνολα.

```
@Entity(name = "QuestionSet")
@Table(name = "QuestionSet")
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class QuestionSet {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "set_id")
    private Long id;
    private @NonNull String title;
    private String description;
    private String qSetType;

    @ManyToMany
    @JoinTable(name = "belongs", joinColumns = @JoinColumn(name =
"set_id"), inverseJoinColumns = @JoinColumn(name = "question_id"))
    private @NonNull Set<QuestionEntity> questions;
    @ManyToMany
    @JoinTable(name = "receives", joinColumns = @JoinColumn(name =
"set_id"), inverseJoinColumns = @JoinColumn(name = "customer_id"))
    private Set<CustomerEntity> customers;
}
```

Όπως βλέπουμε παραπάνω μπορούμε να αναγνωρίσουμε τον τρόπο που η JPA συνδέει τις οντότητες με τύπου συσχετίσεων. Για παράδειγμα το QuestionSet σχετίζεται με ένα σύνολο ερωτήσεων και αυτό φαίνεται με το annotation ManyToMany και τη δήλωση ενός Java Set για το QuestionEntity.

Η οντότητα Question σχετίζεται με το QuestionSet και QuestionOptions

```
@Entity(name = "question")
@Table(name = "question")
@Getter
@Setter
@ToString
@RequiredArgsConstructor
@NoArgsConstructor
public class QuestionEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private @NonNull String title;
    private String description;

    @OneToOne(mappedBy = "question")
    private Answer answer;

    @Basic
    @Temporal(TemporalType.TIMESTAMP)
    private @NonNull Date date;

    @ManyToMany(mappedBy = "questions", cascade = CascadeType.REFRESH)
    private Set<QuestionSet> questionSet;

    @ManyToOne(cascade = CascadeType.DETACH, optional = true)
    @JoinColumn(name = "question_options_id", referencedColumnName =
"id")
    private QuestionOptions questionOptions = new QuestionOptions();

    @ManyToOne
    @JoinColumn(name = "type_id")
    private QuestionType type;
```

```
}
```

Η JPA παρέχει την πρόσβαση σε διεπαφές που επεκτείνουν τις μεθόδους DML και αυτό μπορεί να γίνει με τον παρακάτω κώδικα.

```
public interface QuestionSetRepo extends JpaRepository<QuestionSet,
Long> {}
```

Μια καλή πρακτική είναι να γράψουμε μερικά custom exception ωστε να δίνουμε μια καλύτερη διευκρίνιση το συμβαίνει στο σύστημα μας σε περίπτωση σφάλματος.

```
public class QuestionNotFoundException extends RuntimeException {
    public QuestionNotFoundException(String errorMessage, Throwable
err) {super(errorMessage, err);}
    public QuestionNotFoundException(String errorMessage) {
        super(errorMessage);
    }}
```

Παρακάτω περιγράφεται η διεπαφή που χαρακτηρίζει το QuestionSet σε επίπεδο υπηρεσιών.

```
public interface QuestionSetService {
QuestionSet createDietQuestionSetForUsers(List<Long> userIds,
List<Long> questionIds, String name, String description);
List<QuestionSet> getQuestionSetsByIdentityCode(String identityCode);
List<QuestionEntity> getQuestionsBySetId(String identityCode, Long id);
Boolean addQuestionsToQuestionSet(String identityCode, Long setId,
List<QuestionEntity> questions);

Boolean removeQuestionsFromQuestionSet(String identityCode, Long setId,
List<Long> questions);

QuestionSetDTO updateQuestionSet(Long Id, QuestionSetDTO
questionSetDTO);
Boolean deleteQuestionSetById(Long id);
```

```
List<QuestionSetDTO> getQuestionSets();
QuestionSetDTO getQuestionSetById(Long id);
}
```

Έτσι μπορούμε να αναγνωρίσουμε τις μεθόδους που θα υλοποιηθούν, σκοπός των διεπαφών είναι να ορίσουν μια κοινή λειτουργία για να μπορεί το λογισμικό να είναι ανθεκτικό σε μελλοντικές αλλαγές καθώς μπορούμε να χρησιμοποιήσουμε άλλο implementation.

Παρακάτω περιγράφεται το implementation που περιέχει την λογική.

```
@Slf4j
@Service
@Transactional
@RequiredArgsConstructor
public class QuestionSetServiceImpl implements QuestionSetService {

    private final CustomerService customerService;
    private final QuestionService questionService;
    private final QuestionSetRepo questionSetRepo;
    private final CustomerMapper customerMapper;
    private final QuestionSetMapper questionSetMapper;
```

Γίνεται η δηλωση των Service και των Repositories που θα χρησιμοποιηθούν για να υλοποιήσουν το QuestionSet Service. Παρατηρούμε ότι επειδή δεν υπάρχει λόγος τροποποίησης τους δηλώνονται με την δεσμευμένη μεταβλητη final.

```
    @Override
    public QuestionSet createDietQuestionSetForUsers(List<Long>
userIds, List<Long> questionIds, String name,
            String description) {

        Set<CustomerEntity> customerEntities = new HashSet<>();
        Set<QuestionEntity> questionEntities = new HashSet<>();
```

```
        userIds.forEach(userId -> {

customerEntities.add(customerService.findCustomerById(userId));
        });

        questionIds.forEach(questionId -> {

questionEntities.add(questionService.findQuestionById(questionId));
        });

        QuestionSet questionSet = QuestionSet.builder()
                .title(name)
                .description(description)
                .customers(customerEntities)
                .questions(questionEntities)
                .build();

        return questionSetRepo.save(questionSet);
    }
```

Για να υλοποιήσουμε ένα καινούριο ερωτηματολόγιο θα πρέπει να έχουμε τα σύνολα των πελατών και των ερωτήσεων που αντιστοιχούνται. Όπως φαίνεται στον παραπάνω κώδικα ορίζεται η συνάρτηση **createDietQuestionSetForUsers που** παίρνει ως παράμετρο τα id των χρηστών, τα id των ερωτήσεων, το όνομα που δίνουμε στο ερωτηματολόγιο και την περιγραφή. Για την υλοποίηση της συνάρτησης απαιτούνται οι διεπαφές **CustomerService** και **QuestionService** που χρησιμοποιούνται για την εύρεση των οντοτήτων, στη συνέχεια χρησιμοποιούμε το **QuestionSetRepo** για την αποθήκευση του ερωτηματολογίου. Σε περίπτωση που δεν βρεθεί κάποιος πελάτης ή ερώτηση, οι υπηρεσίες θα επιστρέψουν μια εξαίρεση.

Για την ευκολία εύρεσης ενος ερωτηματολογίου ορίσαμε την συνάρτηση **getQuestionSetsByIdentityCode,** μπορούμε να βρούμε ένα ερωτηματολόγιο με τον κωδικό χρήστη που έχει δοθεί από τον Identity provider.

```
    @Override
    public List<QuestionSet> getQuestionSetsByIdentityCode(String
identity) {
        Optional<CustomerEntity> customer = customerService
                .findCustomerByIdentity(identity);
```

```
        return
customer.get().getQuestionSets().stream().collect(Collectors.toList());
    }
```

Για ευκολία εσωτερικά των υπηρεσιών ορίσαμε την συνάρτηση **getQuestionsBySetId.**

```
    @Override
    public List<QuestionEntity> getQuestionsBySetId(String
identityCode, Long id) {
        Optional<QuestionSet> qset = questionSetRepo.findById(id);
        if (!qset.isPresent()) {
            throw new QuestionNotFoundException("Question Set not
found");
        }
        return
qset.get().getQuestions().stream().collect(Collectors.toList());
    }
```

Για την τροποποίηση του ορίζουμε την συνάρτηση **addQuestionsToQuestionSet**.

```
    @Override
    public QuestionSetDTO updateQuestionSet(Long Id, QuestionSetDTO
questionSetDTO) {
        Optional<QuestionSet> questionset =
questionSetRepo.findById(Id);
        if (!questionset.isPresent()) {
            throw new QuestionSetNotFoundException("Question Set not
found, id:" + Id, null);
        }
        QuestionSet questionSet = questionset.get();
        questionSet.setTitle(questionSetDTO.getTitle());
        questionSet.setDescription(questionSetDTO.getDescription());

        questionSet.setQuestions(questionService.findAllByids(
```

```
                questionSetDTO.getQuestions().stream().map(question ->
question.getId()).collect(Collectors.toList()))
                .stream().collect(Collectors.toSet()));


questionSet.setCustomers(customerMapper.customersDtoToEntities(customer
Service.findCustomersByIds(
                questionSetDTO.getCustomers().stream().map(customer ->
customer.getId()).collect(Collectors.toList()))

.stream().collect(Collectors.toList())).stream().collect(Collectors.toS
et()));
        return
questionSetMapper.questionSetToDto(questionSetRepo.save(questionSet));
    }
```

Για την διαγραφή ενος ερωτηματολογίου ορίζεται η συνάρτηση **deleteQuestionSetById**

```
    @Override
    public Boolean deleteQuestionSetById(Long id) {
        Optional<QuestionSet> questionset =
questionSetRepo.findById(id);
        questionSetRepo.delete(questionset.get());
        return true;
    }
```

Για την εύρεση των ερωτηματολογίων ορίζουμε την συνάρτηση **getQuestionSets,** όπου επιστρέφει το συνολο των ερωτηματολογίων. Είναι σημαντικό να αναφέρουμε ότι στις περιπτώσεις λίστας μπορούμε να ορίσουμε μια σελίδα με αρχή και τέλος έτσι ώστε να επιλέγουμε ποια δεδομένα θέλουμε να παρουσιάσουμε χωρίς να καταγγείλουμε τον Client με πολλά δεδομένα, επίσης μπορούμε να ορίσουμε μια σειρά για τα δεδομένα όπου αυτό γίνεται στους Controller στην προκειμένη περίπτωση.

```
    @Override
    public List<QuestionSetDTO> getQuestionSets() {
        return questionSetRepo.findAll().stream().map(questionSet -> {
```

```
            return questionSetMapper.questionSetToDto(questionSet);
        }).collect(Collectors.toList());
    }

    @Override
    public QuestionSetDTO getQuestionSetById(Long id) {
        return
questionSetMapper.questionSetToDto(questionSetRepo.findById(id).get());
    }
}
```

Τα Data Transfer Objects χρησιμοποιείται για να ενοποιήσει δεδομένα με σκοπό την μείωση των κλήσεων σε μεθόδους. όπως φαίνεται παρακάτω ένα Question Set περιέχει την πληροφορία των χρηστών και τών ερωτήσεων που περιλαμβάνει.

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class QuestionSetDTO {
    private Long id;
    private String title;
    private String description;
    private List<CustomerDTO> customers;
    private List<QuestionDTO> questions;
}
```

Ο παρακάτω κώδικας περιγράφει την διαδικασία μετατροπής από το DTO σε Entity , γίνεται με το annotation Mapper της βιβλιοθήκης Mapstruct.

```
@Mapper(componentModel = "spring")
public interface QuestionSetMapper {

    QuestionSet questionSetDtoToEntity(QuestionSetDTO questionSetDto);
```

```
    QuestionSetDTO questionSetToDto(QuestionSet questionOptionsDTO);
}
```

ΠΑΡΆΓΡΑΦΟΣ REST CONTROLLERS

```
@RestController
@RequestMapping("/questionSet")
@AllArgsConstructor
public class QuestionSetsController {
        private final QuestionSetService questionSetService;
        private final QuestionMapper questionMapper;

 @PostMapping("/create")
 public ResponseEntity<Boolean> createQuestionSet(Principal principal,
                        @RequestBody CreateQuestionSetDTO
questionSetDTO) {
                questionSetService.createDietQuestionSetForUsers(

questionSetDTO.getCustomers().stream().map(customer ->
customer.getId())

.collect(Collectors.toList()),

questionSetDTO.getQuestions().stream().map(question ->
question.getId())

.collect(Collectors.toList()),
                                questionSetDTO.getTitle(),
                                questionSetDTO.getDescription());
                return new ResponseEntity<Boolean>(true,
HttpStatus.OK);
        }

        @GetMapping()
```

```
        public ResponseEntity<List<QuestionSetDTO>>
getQuestionSet(Principal principal) {
                return new
ResponseEntity<List<QuestionSetDTO>>(questionSetService.getQuestionSets
(), HttpStatus.OK);
        }

        @PatchMapping("/edit/{id}")
        public ResponseEntity<QuestionSetDTO>
updateQuestionSet(Principal principal, @PathVariable Long id,
                        @RequestBody QuestionSetDTO questionSetDTO) {
                return new ResponseEntity<QuestionSetDTO>(

questionSetService.updateQuestionSet(id, questionSetDTO),
                                HttpStatus.OK);
        }

        @GetMapping("/show/{id}")
        public ResponseEntity<QuestionSetDTO>
getQuestionsBySetId(Principal principal, @PathVariable Long id) {
                return new ResponseEntity<QuestionSetDTO>(

questionSetService.getQuestionSetById(id),
                                HttpStatus.OK);
        }

        @PostMapping("/add/{id}")
        public ResponseEntity<Boolean> addQuestionToSet(
                        Principal principal,
                        @PathVariable Long id,
                        @RequestBody List<QuestionDTO> questionDTOs) {
                return new ResponseEntity<Boolean>(

questionSetService.addQuestionsToQuestionSet(principal.getName(), id,

questionDTOs.stream().map(questionDTO -> {
                                                        return
questionMapper.questionDtoToQuestionEntity(questionDTO);
```

```
}).sorted(Comparator.comparingLong(QuestionEntity::getId))

.collect(Collectors.toList())),
                                HttpStatus.OK);
        }

        @PostMapping("/remove/{id}")
        public ResponseEntity<Boolean> removeQuestionFromSet(
                        Principal principal,
                        @PathVariable Long id,
                        @RequestBody List<Long> questionIds) {
                return new ResponseEntity<Boolean>(

questionSetService.removeQuestionsFromQuestionSet(principal.getName(),
id, questionIds),
                                HttpStatus.OK);
        }

        @DeleteMapping("/{id}")
        public ResponseEntity<Boolean>
deleteQuestionSet(@PathVariable("id") Long id) {
                return new
ResponseEntity<Boolean>(questionSetService.deleteQuestionSetById(id),
HttpStatus.OK);
        }
}
```

Με την υλοποίηση των Controller κάνουμε χρήση τις μεθόδους που υλοποιήσαμε στο Service Layer. Σκοπός του επιπέδουν αυτού είναι να διαχειριστούμε τα ερωτήματα και την μορφή που επιστρέφουμε την πληροφορία.

3.2.2 Gateway Service Application

Το Spring Cloud Gateway λειτουργεί ως reverse-proxy για τις εφαρμογές web που υποστηρίζουν τα microservices που δημιουργούμε. Είναι μια πύλη που βρίσκεται μεταξύ των χρηστών και των διαφορετικών τμημάτων μιας διαδικτυακής εφαρμογής,

ανακατευθείνει τα αιτήματα προς τον σωστό εξυπηρετητη. Διαχειρίζεται τα αιτήματα και ελέγχει την πρόσβαση, καθως προσθέτει ακόμη και κάποια επιπλέον μέτρα ασφαλείας.

Ενσωματώνει δυνατότητες για καλύτερη διαχείριση των πόρων του συστήματος κατανέμοντας τον φόρτο τον εισερχόμενων αιτημάτων σε διαφορετικά αντίγραφα της ίδιας υπηρεσίας.

Επίσης δίνει την δυνατότητα για μια κεντρική διαχείριση παραμετροποιόντας λειτουργίες όπως η δρομολόγηση, ασφαλεια και τροποποίηση κεφαλίδων και γενική διαχείριση της κυκλοφορίας, καθώς κάνει την ενημέρωση του συστήματος πολυ πιο απλή. Είναι κατάλληλο για αρχιτεκτονική microservices καθώς ενθυλακώνει λύσεις για την δρομολόγηση και εξισορρόπηση φόρτου. Οι λειτουργίες ασφάλειας που περιλαμβάνει αφορούν την ταυτοποίηση χρηστών και εξουσιοδότηση, υποστηρίζει εύκολη ενσωμάτωση τρίτων υπηρεσιών ταυτότητας. Τέλως παρέχει ανοχή στην διακοπή λειτουργίας των υπηρεσιών, επαναδρομολογόντας ή δίνοντας την κατάλληλη ενημέρωση στον χρήστη.

Ο παρακάτω κώδικας περιέχει το Configuration του Gateway σε YAML.

```
server:
  port: 8084

logging:
  level:
    root: INFO
    org:
      springframework: ERROR
    "[com.dhcservices.gateway]": DEBUG

spring:
  application:
    name: gateway
  cloud:
    gateway:
```

```
      default-filters:
        - TokenRelay
        - RemoveRequestHeader=Cookie
        - DedupeResponseHeader=Access-Control-Allow-Credentials
Access-Control-Allow-Origin
      routes:
        - id: customer
          uri: lb://CUSTOMER
          predicates:
            - Path=/customers/**
        - id: question
          uri: lb://CUSTOMER
          predicates:
            - Path=/questions/**
        - id: questionSet
          uri: lb://CUSTOMER
          predicates:
            - Path=/questionSet/**
        - id: medicalprofile
          uri: lb://CUSTOMER
          predicates:
            - Path=/medicalprofile/**
        - id: submitAnswers
          uri: lb://CUSTOMER
          predicates:
            - Path=/submitAnswers/**
        - id: statistics
          uri: lb://CUSTOMER
          predicates:
            - Path=/statistics/**
        - id: settings
          uri: lb://CUSTOMER
          predicates:
            - Path=/settings/**
        - id: QuestionOptions
          uri: lb://CUSTOMER
          predicates:
            - Path=/QuestionOptions/**
```

```
      globalcors:
        cors-configurations:
          "[/**]":
            allowedOrigins: "http://localhost:3000"
            allowedHeaders: "*"
            allowedMethods: [GET, POST, PUT, PATCH, DELETE]

eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka
    fetch-registry: true
    register-with-eureka: true

management:
  endpoints:
    web:
      exposure:
        include: "*"
```

3.2.3 Eureka Service Application

Με την χρήση ενός Service Discovery όπως το Apache-Eureka μπορούμε να αντιστοιχίσουμε τις διευθύνσεις IP με τα ονόματα των Service για μεγαλύτερη διασάφηση όπως λειτουργεί ένα DNS service.

Eureka Server Configuration YAML

```
spring:
    application:
        name: eureka-server
server:
    port: 8761
```

```
eureka:
    client:
        fetch-registry: false
        register-with-eureka: false
```

Eureka Client Configuration YAML

```
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka
    fetch-registry: true
    register-with-eureka: true

spring:
  application:
    name: customer
```

Είναι σημαντικό να δηλωθεί το ονομα της υπηρεσίας στο Spring application όπως φαίνεται στον παραπάνω κώδικα.

3.2 Διαδικτυακη Υπηρεσία πρόβλεψης

Προκειμένου να διαχωρίσουμε την λειτουργικότητα και να ακολουθήσουμε την πρακτική του Single responsibility θα βάλουμε την λειτουργία πρόβλεψης σε μια αυτόνομη υπηρεσία. Σκοπός μας είναι να απομονώσουμε τις λειτουργίες πρόβλεψης και στατιστικών που μπορούν να προκύψουν μελλοντικά απο το codebase της Python, καθώς οι βιβλιοθήκες που υποστηρίζονται είναι εξειδικευμένες γύρω από την ανάλυση δεδομένων.

3.2.1 Flask application

Ξεκινάμε με την δημιουργια του περιβάλλοντος, εγκαθιστόντας τις κατάλληλες βιβλιοθήκες.

HTTP Request με την χρήση Flask

```
app = Flask(__name__)

port = 5000
# The flowing code will register your server to eureka server and also
start to send heartbeat every 30 seconds
eureka_client.init(eureka_server="http://localhost:8761",
                   app_name="MLRFService",
                   instance_port=port)

# Load the saved encoder from the file
with open('encoder.pkl', 'rb') as file:
    encoder = pickle.load(file)


@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"


if __name__ == "__main__":
    app.run(debug=False)


@app.route("/pred",  methods=['POST'])
def predict():
    requestList = request.json
```

Εισάγουμε το dataframe που αποθηκεύσατε σε δομή pickle για να μπορέσουμε να κάνουμε one-hot encoding.

```
    input_data = pd.DataFrame(requestList)
    input_data = input_data.T
```

```
    input_data.columns = ["age",
"sex","cp","trestbps","chol","fbs","restecg","thalach",
"exang","oldpeak","slope","ca","thal"]

    categorical_features = ["cp", "restecg", "thal", "slope"]
    selected_input_data = input_data[categorical_features]

    # Convert numerical categories to strings
selected_input_data["cp"] = selected_input_data["cp"].astype(str)
selected_input_data["thal"] = selected_input_data["thal"].astype(str)
selected_input_data["restecg"] =
selected_input_data["restecg"].astype(str)
selected_input_data["slope"] = selected_input_data["slope"].astype(str)

    encoded_data = encoder.transform(selected_input_data)

# Repalce the values in original input
encoded_df =
pd.DataFrame(encoded_data,columns=encoder.get_feature_names_out(
categorical_features))

data_encoded =
pd.concat([input_data.drop(columns=categorical_features), encoded_df],
axis=1)

input_data = data_encoded.loc[0, :].values.flatten().tolist()
sess = rt.InferenceSession("rf_heart.onnx", providers=[
                               "CPUExecutionProvider"])

    # Ensure the input data has the expected shape and data type
    input_name = sess.get_inputs()[0].name

    # Perform inference
    outputs = sess.run(None, {input_name: [input_data]})

    # Retrieve the output
    output_name = sess.get_outputs()[0].name
    predictions = outputs[0]
```

```
    value = predictions[0]
    responseData = {
        "prediction": str(value)
    }

    return jsonify(responseData)
```

Επιστρέφουμε την πρόβλεψη σε μορφή JSON. Οταν ο πελάτης δεν έχει καρδιακό πρόβλημα η τιμη του prediction ειναι 0, ένω όταν έχει πιθανότητα καρδιοπάθειας η τιμή prediction είναι 1.

```
# Στην περίπτωση που είναι αρνητική
{
   "prediction": 0
}
# Στην περίπτωση που είναι θετική
{
   "prediction": 1
}
```

3.3 Πλατφόρμα διάδρασης

3.3.1 React UI

Για την δημιουργία της διεπαφής χρηστών χρησιμοποιήθηκε το πλαίσιο React και πιο συγκεκριμένα το σύνολο βιβλιοθηκών του Refine.

Θα γίνει συνοπτική περιγραφή της χρήσης της βιβλιοθήκης.

Για την χρήση του refine απαραίτητες να είναι εγκατεστημένο το npm και node.js
Με την ακόλουθη εντολή δημιουργούμε ένα καινούριο project.

```
npm create refine-app@latest -- -o refine-headless tutorial
```

Στη συνέχεια χρησιμοποιούμε το CLI που προσφέρει το refine για να δημιουργήσουμε τα resources.

```
npm run refine create-resource questions
```

```
src/questions/
└── categories
    ├── create.tsx
    ├── edit.tsx
    ├── index.ts
    ├── list.tsx
    └── show.tsx
```

Μια από τις σημαντικότερες δυνατότητες που προσφέρει το refine είναι ότι μπορούμε να χρησιμοποιήσουμε ένα component που ονομάζεται Inferencer και μας παράγει μια βασική δομή για την διεπαφή σύμφωνα με το θέμα που διαλέξαμε.

Χρησιμοποιούμε την παρακάτω εντολή να εισάγουμε και να χρησιμοποιήσουμε έαν inference.

```
npm i @refinedev/inferencer
```

Οι βιβλιοθήκες γραφικών που υποστηρίζονται περιλαμβάνει τις εξής, Ant Design, Material UI, Mantine, Chakra UI, για τους σκοπούς αυτής της πτυχιακής χρησιμοποιούμε το Ant UI. Με το παρακάτω παράδειγμα μπορούμε να δημιουργήσουμε μια σελίδα.

```
import { AntdInferencer } from "@refinedev/inferencer/antd";
<Refine
                routerProvider={routerProvider}
                resources={[
                    {
                        name: "questions",
                        list: "/questions",
                    },
                ]}
            >
                <Routes>
                    <Route path="/samples" element={<AntdInferencer />}
/>
                </Routes>
            </Refine>
```

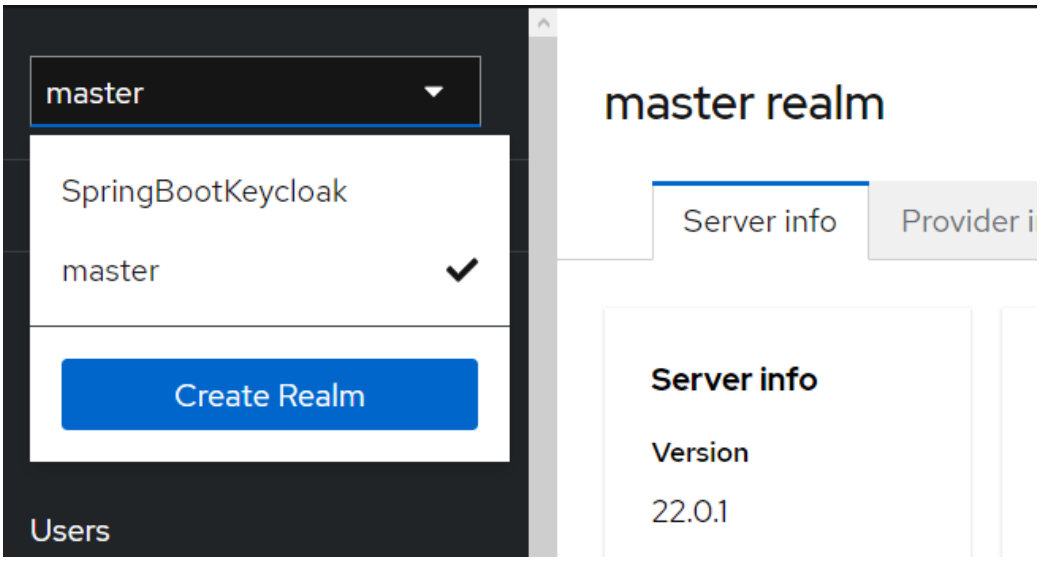
Όπως φαίνεται στον παραπάνω κώδικα το περιεχόμενο της σελίδας λαμβάνεται απο τους πόρους (resources) που δίνονται μέσα στο component Refine.

Στα πλαίσια μιας εφαρμογής CRUD, ένας πόρος αναφέρεται συνήθως σε μια οντότητα δεδομένων που μπορεί να δημιουργηθεί, να διαβαστεί, να ενημερωθεί ή να διαγραφεί. Έτσι το refine αναγνωρίζει τους πόρους που ορίζουμε και παράγει κομμάτι της διεπαφής, η οποία είναι πλήρως παραμετροποιήσιμη.

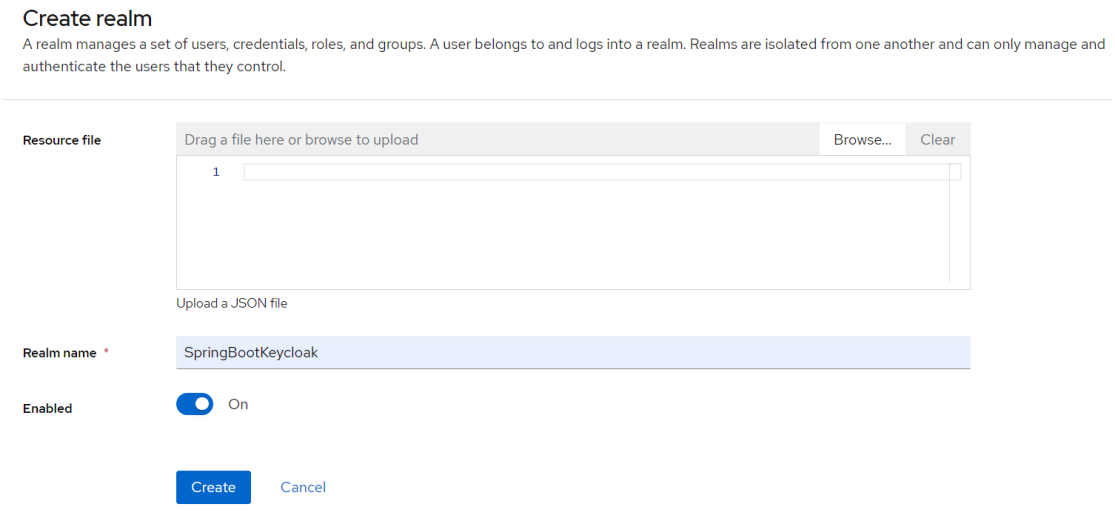
3.4 Single Sign on Server

3.4.1 Keycloak

Για την διανομή τον token θα χρησιμοποιήσουμε ένα open source Identity provider, η δημιουργία των JWT είναι μια διαδικασία που θα είχε ενδιαφέρον να περιγραφεί αναλυτικά στα πλαίσια μιας πτυχιακής, αλλά δεν περιλαμβάνεται στον στόχο της συγκεκριμενεις, έτσι θα δείξουμε βασικές διαδικασίες για δημιουργία Realm, Clients, User και Roles.

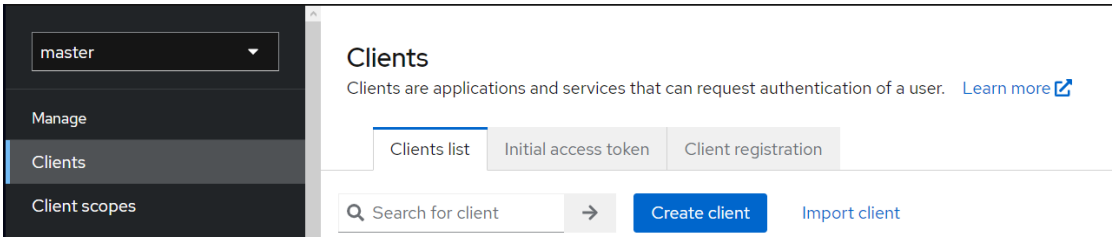


Εικόνα 9. *Keycloak Realms*

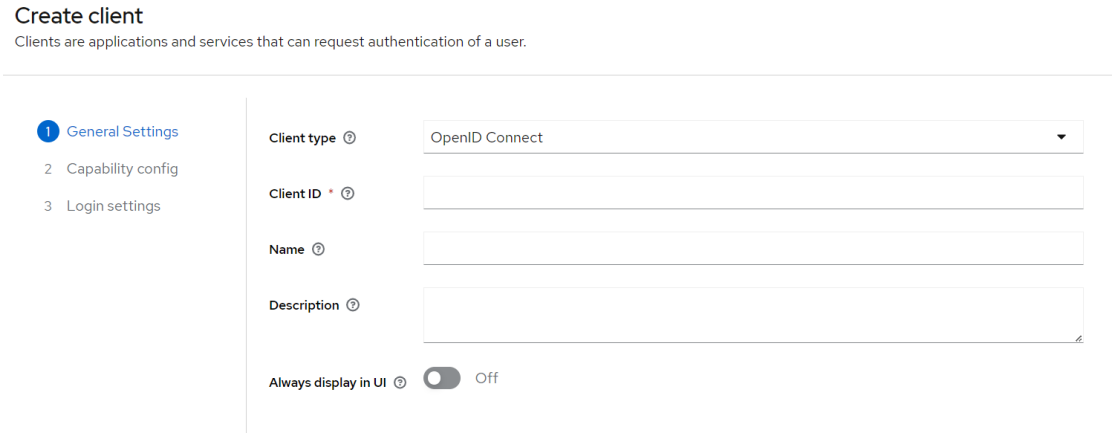


Εικόνα 10. *Keycloak create Realm*

Το Realm αναφέρεται σε ένα τομέα ασφάλειας και διαχείρισης όπου περιλαμβάνονται οι χρήστες τα προγράμματα και οι ρόλοι.

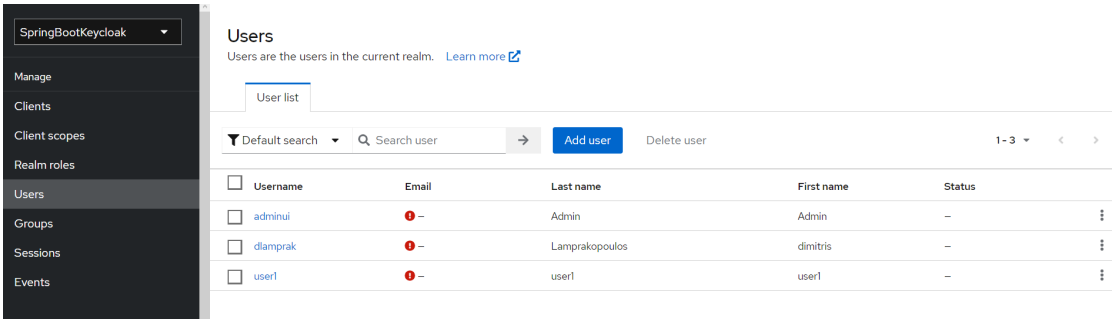


Εικόνα 11. *Keycloak Clients*

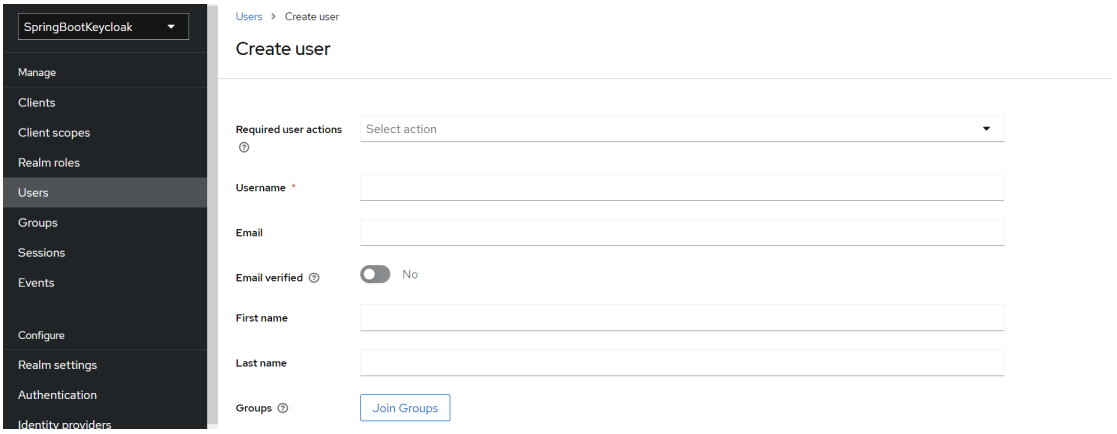


Εικόνα 12. *Keycloak create Client*

Clients είναι οι οντότητες που ταυτίζονται με το πρόγραμμα και μπορούν να αιτηθούν πιστοποίηση χρηστών.

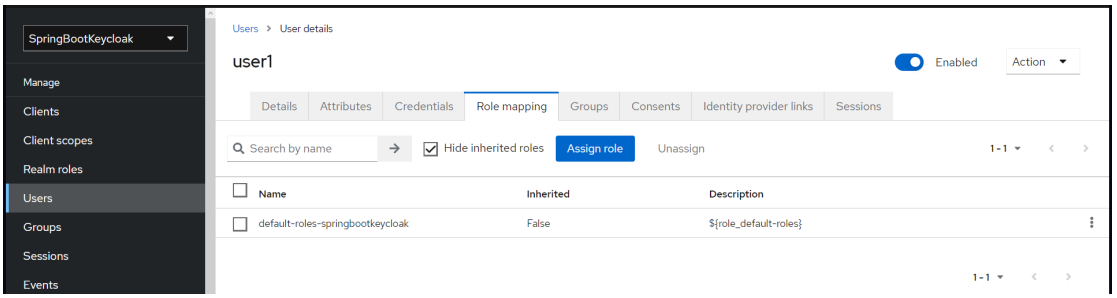


Εικόνα 13. *Keycloak User*



Εικόνα 14. *Keycloak Create User*

Μπορούμε δημιουργήσουμε χρήστες και να αναθέσουμε ρόλους.



Εικόνα 15. *Keycloak Assign Role.*

4. Συμπεράσματα

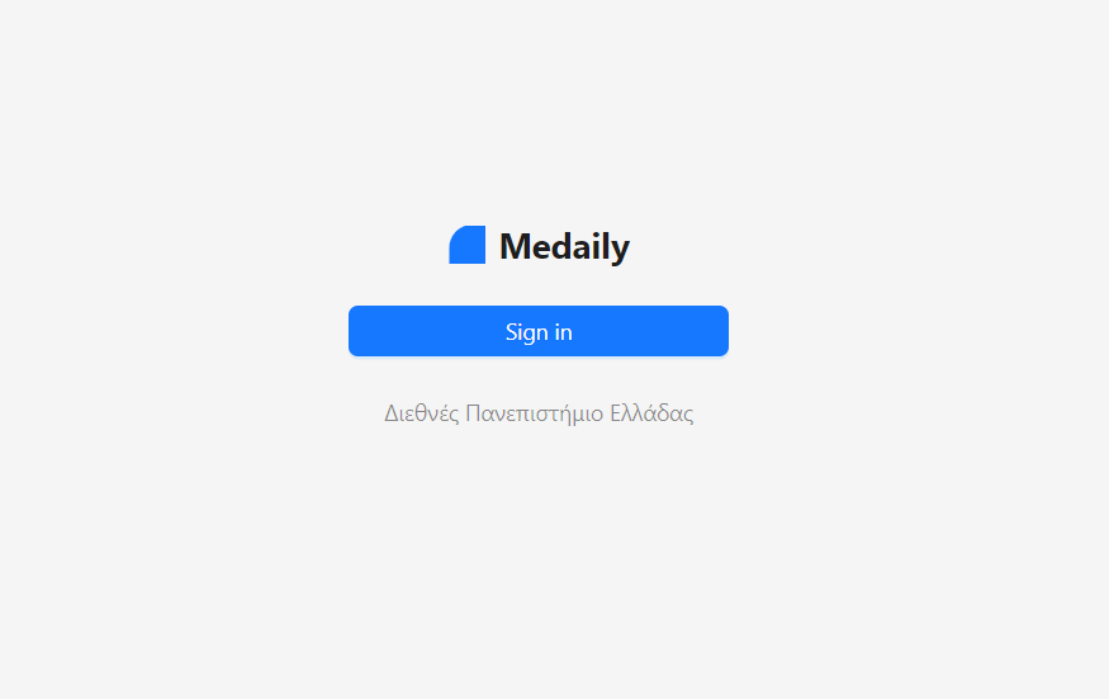
Σε αυτή την πτυχιακή ερευνήθηκε η δυνατότητα ανάπτυξης για μια πλατφόρμα δεδομένων και η χρήση των πληροφοριών για την πρόβλεψη καρδιακής ασθένειας.

Η απόδοση του μοντέλου με 88% μπορεί να βοηθήσει τους ειδικούς να βγάλουν πιο ακριβή συμπεράσματα και η συλλογή των στατιστικών να μας δώσει την δυνατότητα για την δημιουργία ενός μοντέλου πιο αποδοτικού αλλά πιο απαιτητικού σε δεδομένα για την εκπαίδευση. Η σωστή περιγραφή των στατιστικών που αφορούν την καθημερινότητα μας μπορεί να μας βοηθήσει να βελτιώσουμε την καθημερινή ζωή μας, πολλές από τις επιλογές που κάνουμε καθημερινά επηρεάζουν άμεσα την υγεία μας, έχοντας μια πλατφόρμα για να παρέχει με εύκολο τρόπο πληροφορίες για να βελτιώσουμε την ποιότητα ζωής.

5. Διαχειριση Εφαρμογής

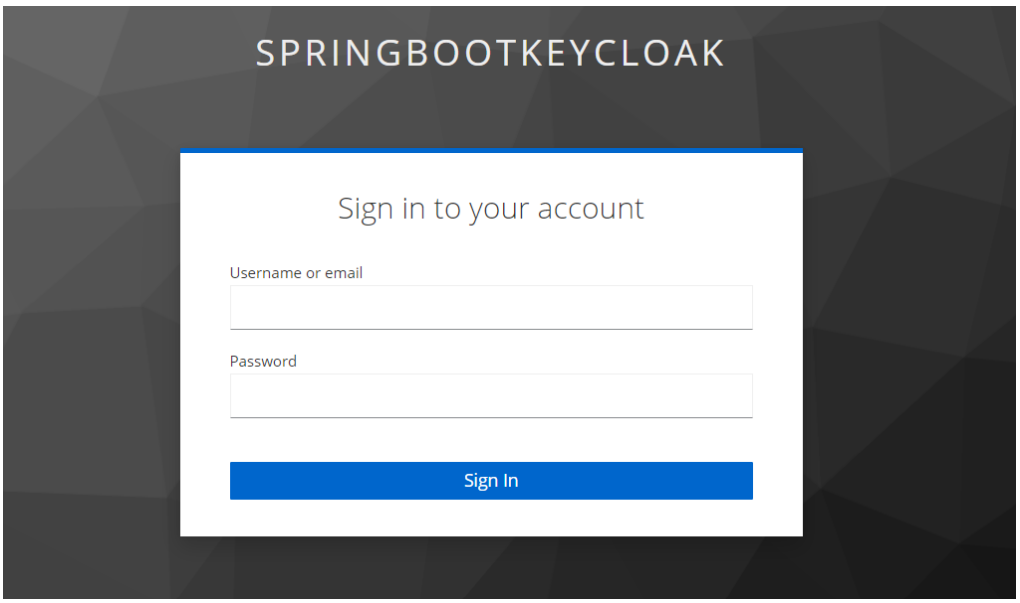
Σε αυτό το κεφάλαιο περιγράφονται αναλυτικά τα βήματα για την διαχείριση της εφαρμογής.

Αρχικά παρουσιάζεται η πρώτη σελίδα με το logo της εφαρμογής, το όνομα και την ανακατεύθυνση για login.



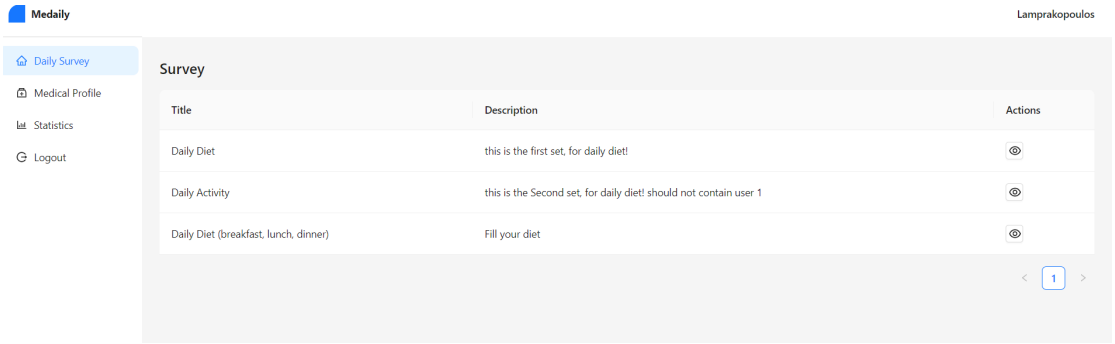
Εικόνα 16. *Η σελίδα που παρουσιάζεται με την είσοδο στην εφαρμογή.*

Μετά την ανακατεύθυνση στο Keycloak μας δίνει την δυνατότητα για είσοδο στην εφαρμογή. Θα παρουσιάσουμε πρώτα τις περιπτώσεις χρήστη.



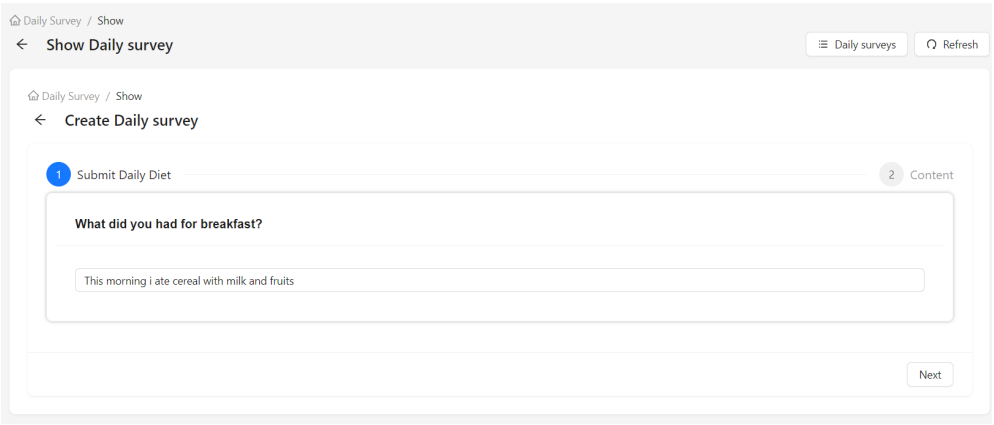
Εικόνα 17. *Login μεσω Keycloak.*

Στο Daily Survey παρουσιάζονται τα ερωτηματολόγια που έχουν δημιουργηθεί από τον διαχειριστή ή γιατρό για ένα σύνολο πελατών.



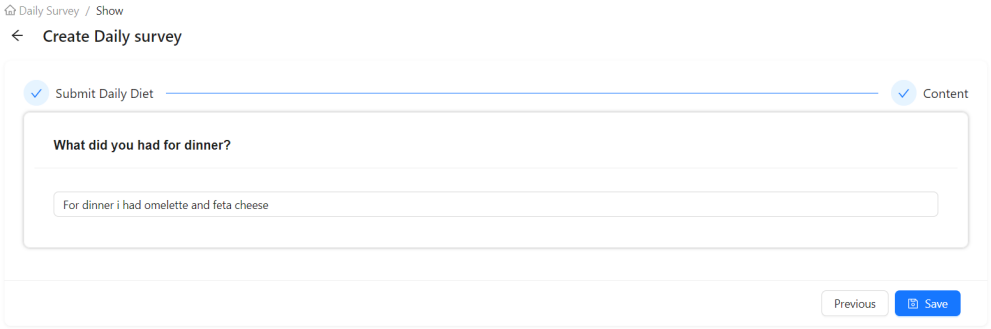
Εικόνα 18. *Daily Survey.*

Με την επιλογή ενος ερωτηματολογίου μπορούμε να απαντήσουμε σε ένα σύνολο ερωτήσεων που έχουν προσδιοριστεί και με τη συλλογή τους θα προκύψουν τα στατιστικά.



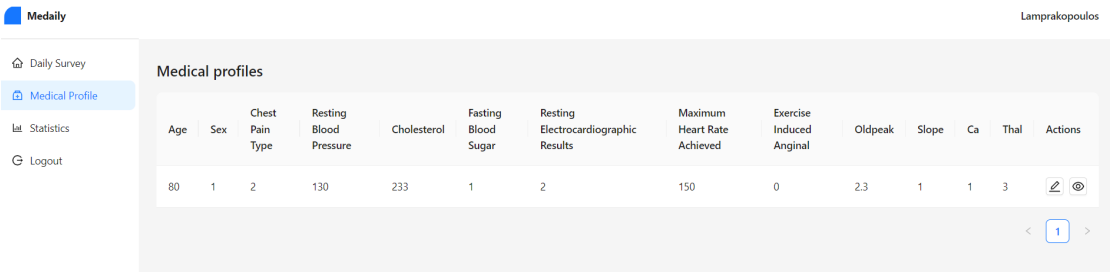
Εικόνα 19. *Απαντήσεις για ένα Question Set.*

Με την ολοκλήρωση των απαντήσεων μπορούμε να αποθηκεύσουμε το ερωτηματολόγιο.



Εικόνα 20. *Αποθήκευση ερωτηματολογίου.*

Η εφαρμογή κρατάει ένα ιστορικό με τιμές που είναι χρήσιμες για την πρόβλεψη καρδιοπάθειας και αυτό μπορούμε να το δούμε στο Medical Profile.



Εικόνα 21. *Medical Profile.*

Η ενημέρωση του Medical Profile φαίνεται στην παρακάτω εικόνα.

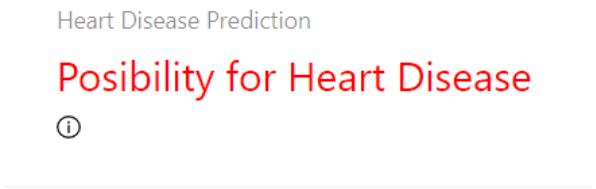
Εικόνα 22. *Επεξεργασία τιμών του Medical Profile.*

Παρακάτω θα δείξουμε τα περιεχόμενα της σελίδας στατιστικών.

Με την συλλογή των δεδομένων από τα ερωτηματολόγια μπορούμε να δούμε τις αξίες των θρεπτικών ουσιών για το διάστημα που προσδιορίζουμε.

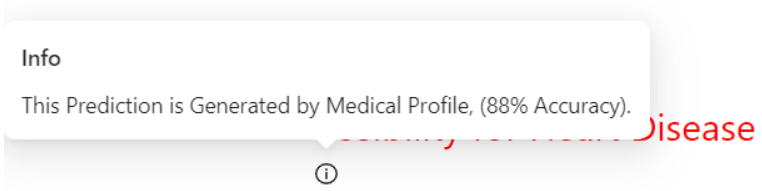
Εικόνα 23. *Προβολή των μέσων τιμών από τα θρεπτικά συστατικά στο διάστημα που δώθηκε.*

Ανανεωνοντας τα στοιχεία του Medical Profile μπορούμε να δούμε την πρόβλεψη για καρδιοπάθεια να ενημερώνεται.



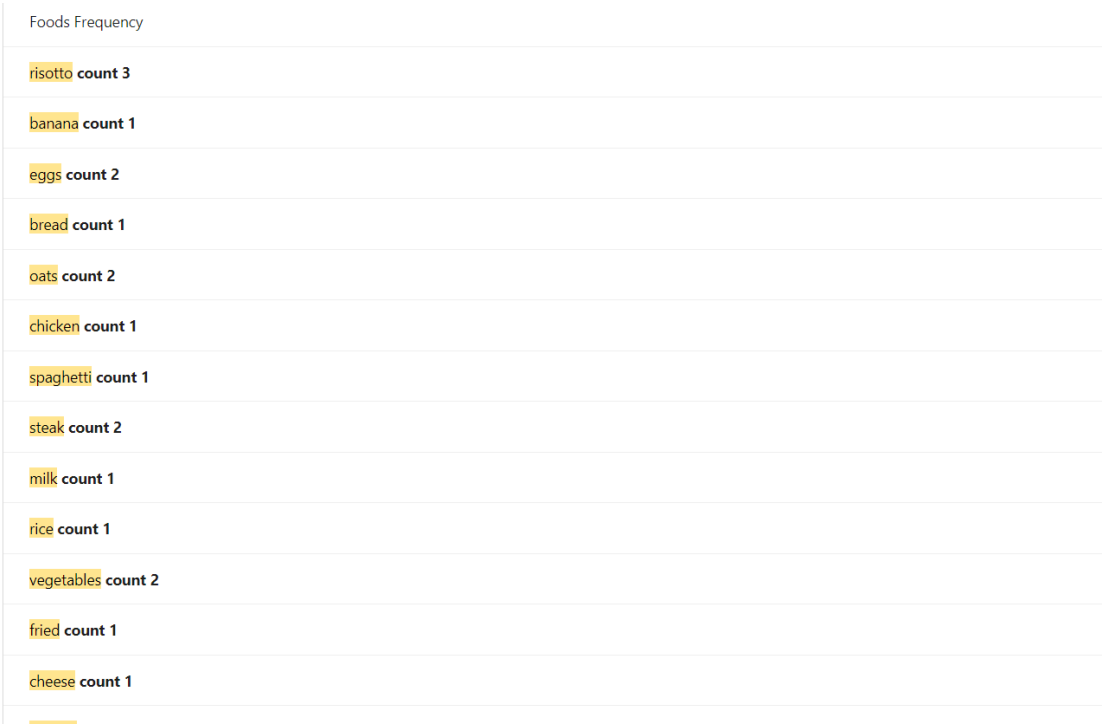
Εικόνα 24. *Πρόβλεψη καρδιοπάθειας.*

Ο χρήστης πληροφορείται ότι η ανανέωση γίνεται απο το Medical Profile.



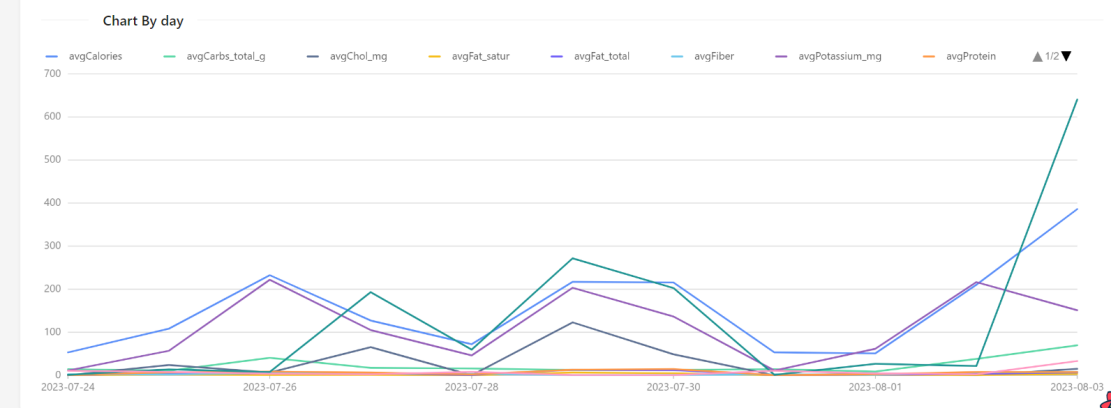
Εικόνα 25. *Ενημέρωση χρήστη.*

Μπορούμε επίσης να δούμε την συχνότητα τον τροφών που λαμβάνουμε.



Εικόνα 26. *Ποσότητες φαγητών.*

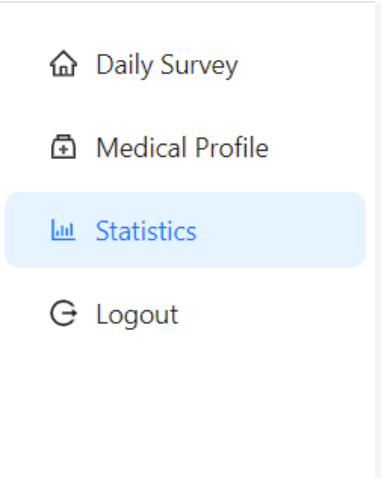
Τέλος για συμπεράσματα μπορούμε να δούμε το διάγραμμα που δίνει της ημερήσιες ποσότητες θρεπτικών ουσιών.



Εικόνα 27. *Διάγραμμα στατιστικών ανά ημέρα.*

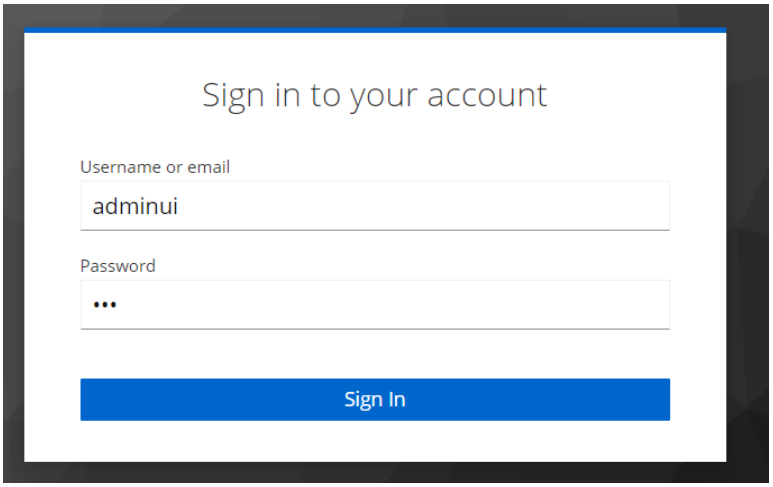
Στη συνέχεια θα δούμε τις λειτουργίες που παρέχονται στον διαχειριστή για επεξεργάζεται τα ερωτηματολόγια.

Κάνοντας logout θα βρεθούμε στην αρχική σελίδα εισόδου χρηστών του keycloak.



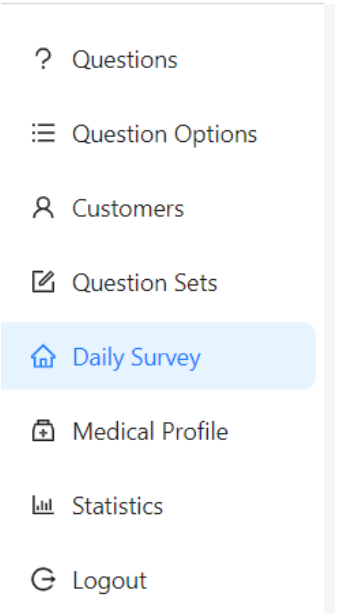
Εικόνα 28. *Αποσύνδεση από την εφαρμογή.*

θα κάνουμε είσοδο ως διαχειριστής



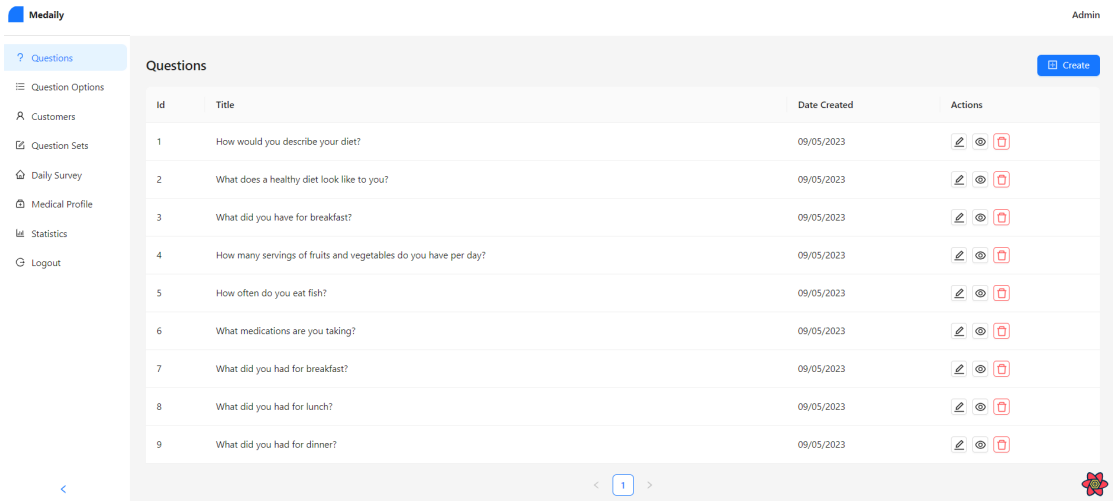
Εικόνα 29. *Σύνδεση ως διαχειριστής.*

Μπορούμε να δούμε ότι στο μενού του διαχειριστή φαίνονται πρόσθετες λειτουργίες για την διαχείριση ερωτηματολογίων.



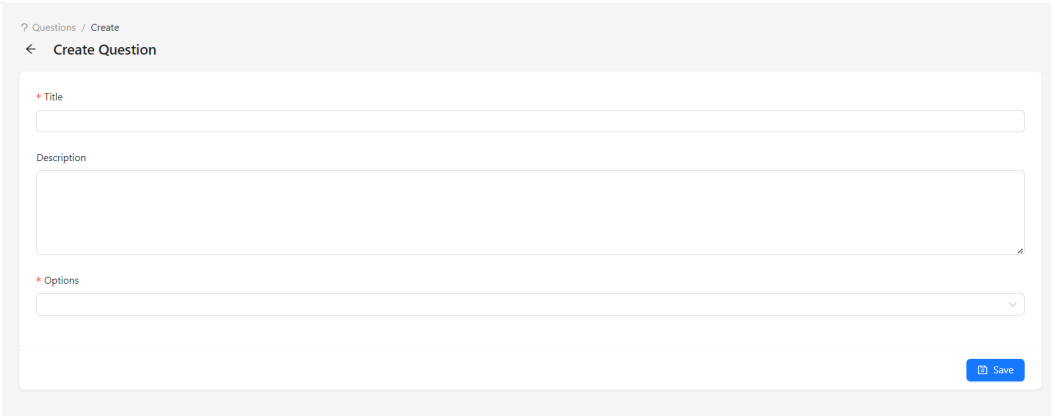
Εικόνα 29. *Πρόσθετες λειτουργίες στο μενού του διαχειριστή.*

Παρουσιάζεται η λίστα των ερωτήσεων με τον τίτλο και την ημερομηνία που δημιουργήθηκε, στις δυνατότητες μπορούμε να επιλέξουμε επεξεργασία με την εικόνα του μολυβιού, προβολή με τη εικόνα του ματικού ή διαγραφή με την εικόνα κάδου.



Εικόνα 30. *Λίστα ερωτήσεων.*

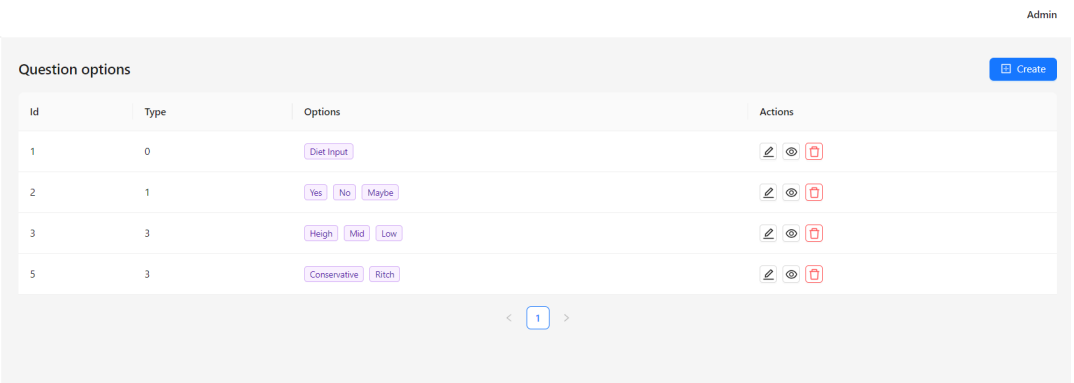
Με το Create μπορούμε να δημιουργήσουμε μια καινούρια ερώτηση.



Εικόνα 31. *Δημιουργία ερώτησης.*

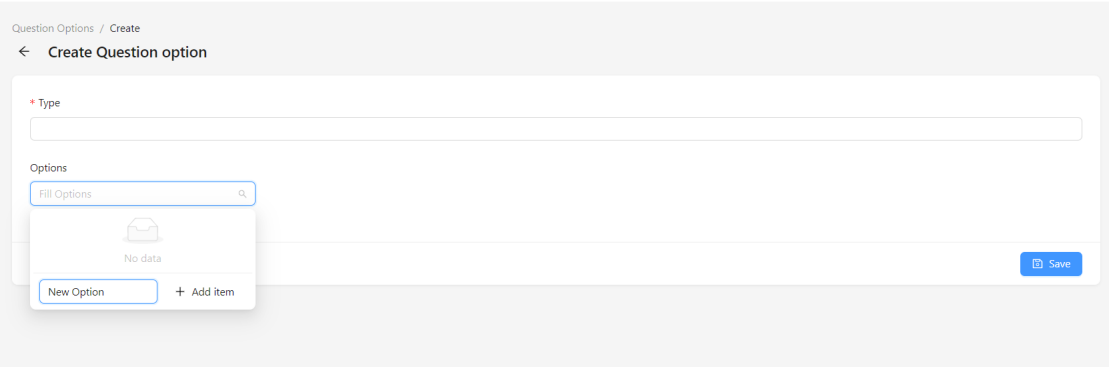
Στα Options φαίνεται η λίστα των επιλογών που έχουμε δημιουργήσει.

Παρακάτω φαίνεται η λίστα των επιλογών που έχουμε ορίσει και ακολουθεί το ίδιο πρότυπο για επιλογές (Actions).



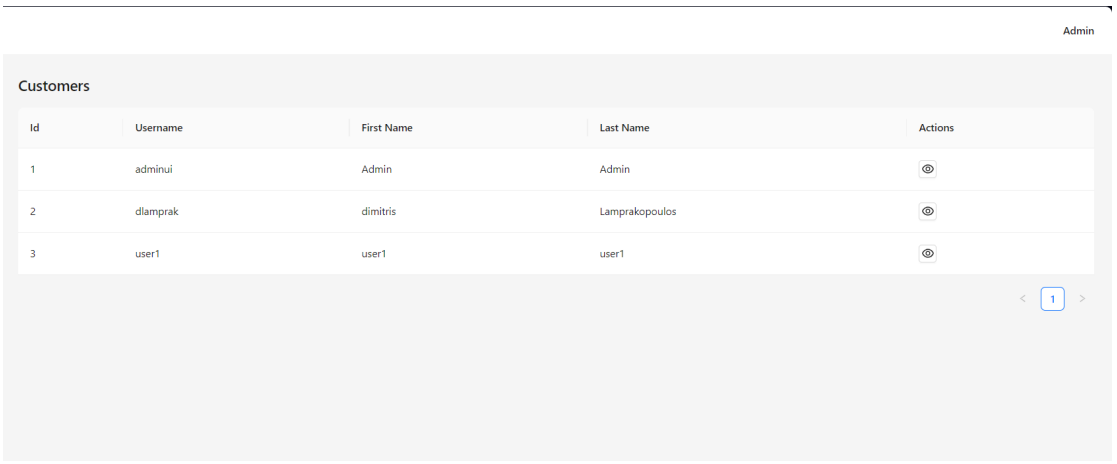
Εικόνα 32. *Λίστα επιλογών ερωτήσεις.*

Με το Create Question Option μπορούμε να δημιουργήσουμε νέα επιλογή για τις ερωτήσεις.

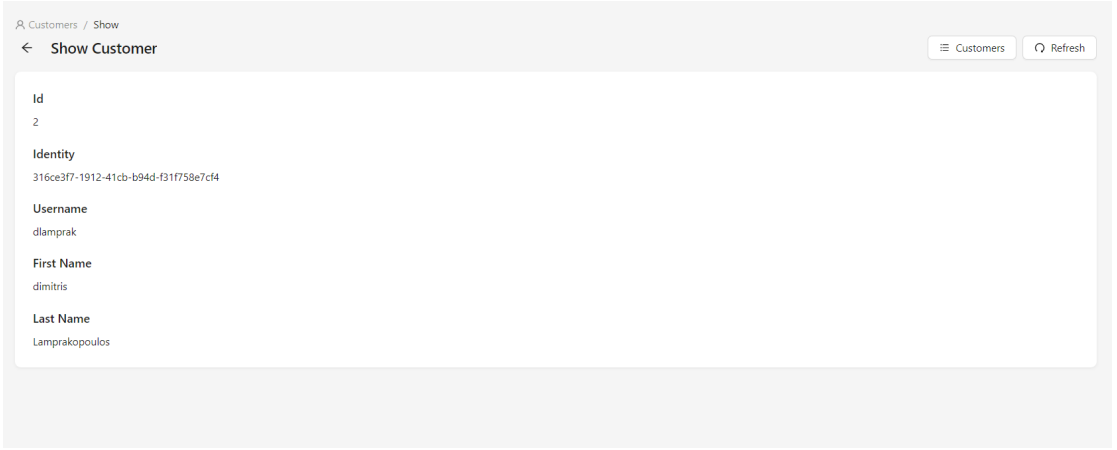


Εικόνα 33. *Προσθήκη επιλογών ερωτήσεις.*

Οι χρήστες επειδή διχειρίζονται από το Keycloak έχουμε μονο δυνατότητα προβολής από την εφαρμογή.

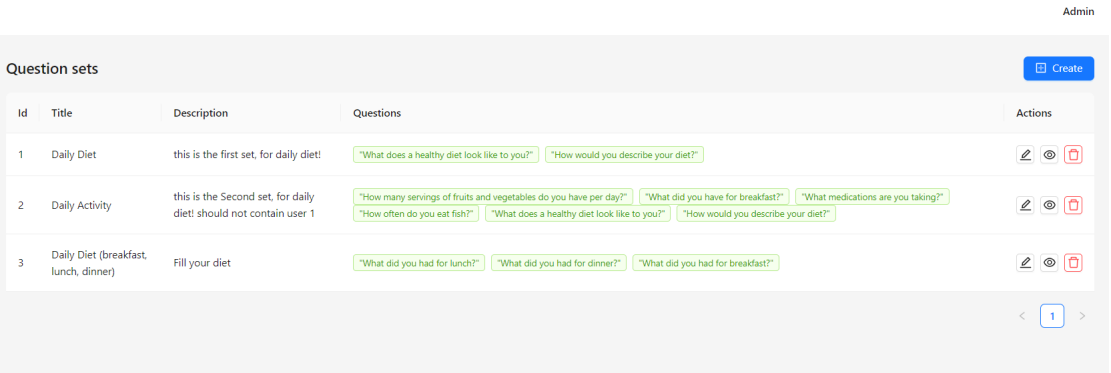


Εικόνα 34. *Λίστα χρηστών.*



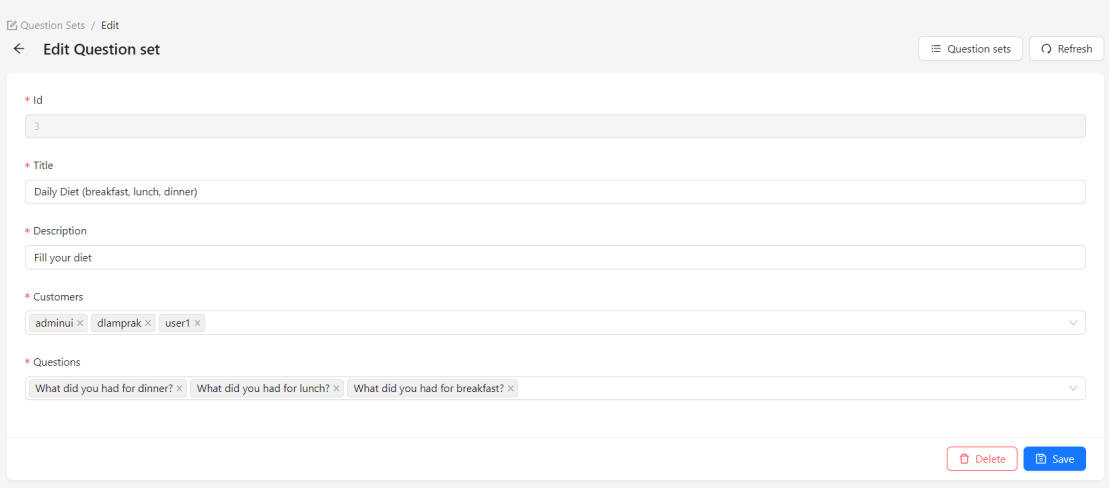
Εικόνα 35. *Προβολή χρήστη.*

Τέλος τα ερωτηματολόγια συντάσσονται από τις ερωτήσεις που έχουμε δημιουργήσει και τους υπάρχοντες χρήστες, ακολουθούν το ίδιο πρότυπο επιλογών (Actions). Φαίνεται ο τίτλος, η περιγραφή και μια σύνοψη των ερωτήσεων που περιλαμβάνει.



Εικόνα 36. *Λίστα ερωτηματολογίων.*

Στην επεξεργασία (Edit) των ερωτηματολογίων μπορούμε να τροποποιήσουμε τον τιτλο και την περιγραφή, καθώς και να προσθέσουμε ή να αφαιρέσουμε χρήστες ή ερωτήσεις.



Εικόνα 37. *Επεξεργασία ερωτηματολογίου.*

Βιβλιογραφία

[1] UCI Machine Learning Repository. (n.d.). Heart Disease Dataset.

[2] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825–2830.

[3] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D.liphant, T. E. (2020). Array Programming with NumPy. Nature, 585(7825), 357–362

[4] McKinney, W. (2010). Data Structures for Statistical Computing in Python. In Proceedings of the 9th Python in Science Conference (pp. 56-61)

[5] Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering, 9(3), 90–95

[6] skl2onnx. (n.d.)

[7] ONNX Runtime. (n.d.)

[8] World Health Organization. (n.d.). Title of the Document.

[9] European Society of Cardiology. (n.d.). Title of the Document.