ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή Εργασία

Τίτλος Πτυχιακής Εργασίας	Μελέτη και ανάπτυξη εφαρμογής για δυναμική διαχείριση διαδικασιών σε Java με χρήση Drools. Web app for dynamic process management using Spring Boot and Drools
Ονοματεπώνυμο Φοιτητή	Ευθύμιος Κοτούμπας
Πατρώνυμο	Ιωάννης
Αριθμός Μητρώου	Π/ 16055
Επιβλέπων	Ευάγγελος Σακκόπουλος, Αναπληρωτής Καθηγητής

Ημερομηνία Παράδοσης Σεπτέμβριος 2024

Copyright ©

Επιτελική Σύνοψη

Η παρούσα πτυχιακή εργασία επικεντρώνεται στην ανάπτυξη μιας διαδικτυακής εφαρμογής για τη δυναμική διαχείριση αιτήσεων μεταπτυχιακών προγραμμάτων, χρησιμοποιώντας τις τεχνολογίες Spring Boot και Drools. Το αντικείμενο της μελέτης αφορά τη δημιουργία ενός συστήματος που θα επιτρέπει στους υποψήφιους φοιτητές να εγγράφονται, να επιλέγουν το πρόγραμμα που επιθυμούν και να υποβάλλουν αίτηση. Η καινοτομία της εφαρμογής έγκειται στη χρήση των κανόνων Drools, οι οποίοι επιτρέπουν την άμεση αξιολόγηση της αίτησης του υποψηφίου βάσει συγκεκριμένων ακαδημαϊκών και προσωπικών κριτηρίων. Το πρόβλημα που επιδιώκεται να αντιμετωπιστεί αφορά τη συχνά χρονοβόρα και περίπλοκη διαδικασία αξιολόγησης αιτήσεων που παρατηρείται στα εκπαιδευτικά ιδρύματα. Στόχος της εφαρμογής είναι να προσφέρει μια αυτοματοποιημένη λύση που να επιταχύνει την αξιολόγηση των αιτήσεων, μειώνοντας τον χρόνο αναμονής για τους φοιτητές και τον φόρτο εργασίας για τους διαχειριστές των προγραμμάτων. Η ενσωμάτωση του Drools ως μηχανισμού κανόνων επιτρέπει την εφαρμογή σύνθετων κριτηρίων επιλογής και την άμεση απόκριση στον υποψήφιο για το εάν η αίτησή του έχει γίνει αποδεκτή ή απορριφθεί.

Για να επιτευχθεί αυτό, αρχικά ένας φοιτητής εγγράφεται και συνδέεται. Στη συνέχεια, εμφανίζεται μια λίστα με τα διαθέσιμα μεταπτυχιακά προγράμματα, και ο φοιτητής επιλέγει ένα από αυτά για να υποβάλει την αίτησή του για το μεταπτυχιακό πρόγραμμα της επιλογής του. Μετά τη συμπλήρωση των στοιχείων του, με βάση τους κανόνες του Drools που χρησιμοποιούνται, εμφανίζεται ένα μήνυμα που δείχνει εάν η αίτηση έχει εγκριθεί ή απορριφθεί. Τέλος, ο διαχειριστής μπορεί να δει όλες τις υποβληθείσες αιτήσεις και να προσθέσει νέα μεταπτυχιακά προγράμματα.

Ευχαριστίες

Ξεκινώντας, θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στον κ. Σακκόπουλο Ευάγγελο καθηγητή του Τμήματος Πληροφορικής στο Πανεπιστήμιο Πειραιά για την ανάθεση της πτυχιακής εργασίας και την πολύτιμη καθοδήγησή του. Τέλος, ένα μεγάλο ευχαριστώ σε όλους τους ανθρώπους που βρίσκονται δίπλα μου όλο αυτό το διάστημα, για την πολύτιμη στήριξή τους.

Σεπτέμβριος 2024

Ευθύμιος Κοτούμπας

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

Κεφάλαιο 1ο

1 Εισαγωγή

1.1 Περιγραφή του υπό μελέτη προβλήματος

Η διαχείριση επιχειρηματικών διαδικασιών αποτελεί μια κρίσιμη πρόκληση για τις σύγχρονες επιχειρήσεις, καθώς απαιτεί τη δυνατότητα διαρκούς προσαρμογής και βελτιστοποίησης των διαδικασιών τους, ανάλογα με τις αλλαγές στο επιχειρηματικό περιβάλλον. Παραδοσιακές λύσεις λογισμικού για τη διαχείριση διαδικασιών είναι συχνά στατικές και δύσκολα παραμετροποιήσιμες, καθιστώντας την ευελιξία και τη δυναμικότητα ζωτικής σημασίας ζητούμενα. Το υπό μελέτη πρόβλημα επικεντρώνεται στην ανάπτυξη μιας διαδικτυακής εφαρμογής που θα επιτρέπει τη δυναμική διαχείριση διαδικασιών, χρησιμοποιώντας σύγχρονες τεχνολογίες όπως το Spring Boot και το Drools. Η συγκεκριμένη προσέγγιση παρέχει τη δυνατότητα δημιουργίας, τροποποίησης και εκτέλεσης κανόνων σε πραγματικό χρόνο, βελτιώνοντας την απόδοση και την αποδοτικότητα των επιχειρηματικών διαδικασιών. Επιπλέον, δίνεται έμφαση στη διαλειτουργικότητα της εφαρμογής, επιτρέποντας την εύκολη ενσωμάτωση με άλλες επιχειρηματικές εφαρμογές και συστήματα. Η ανάγκη για ένα τέτοιο σύστημα προκύπτει από την αδυναμία των παραδοσιακών συστημάτων να προσαρμόζονται γρήγορα σε νέες επιχειρηματικές απαιτήσεις και στην πολυπλοκότητα που δημιουργείται από την ανάγκη για συνεχή εξέλιξη των κανόνων και των διαδικασιών.

1.2 Σκοπός και στόχοι της εργασίας

Ο κύριος σκοπός της παρούσας εργασίας είναι η ανάπτυξη μιας διαδικτυακής εφαρμογής που θα επιτρέπει την αποτελεσματική και αυτοματοποιημένη διαχείριση αιτήσεων για μεταπτυχιακά προγράμματα, χρησιμοποιώντας τις σύγχρονες τεχνολογίες Spring Boot και Drools. Η εφαρμογή αυτή αποσκοπεί στη βελτίωση της διαδικασίας επιλογής των υποψήφιων φοιτητών, παρέχοντας άμεση ενημέρωση σχετικά με την αποδοχή ή απόρριψη της αίτησής τους, μειώνοντας την ανάγκη για ανθρώπινη παρέμβαση και εξασφαλίζοντας μια ακριβή και αμερόληπτη αξιολόγηση.

Οι βασικοί στόχοι της εργασίας περιλαμβάνουν:

- **Ανάπτυξη Συστήματος Εγγραφής και Εισόδου Χρηστών:** Δημιουργία ενός ασφαλούς και αξιόπιστου συστήματος εγγραφής και εισόδου, μέσω του οποίου οι φοιτητές θα μπορούν να δημιουργούν λογαριασμό, να συνδέονται στην πλατφόρμα και να διαχειρίζονται τις αιτήσεις τους.
- **Επιλογή και Προβολή Μεταπτυχιακών Προγραμμάτων:** Παροχή δυνατότητας στους φοιτητές να εξερευνούν τα διαθέσιμα μεταπτυχιακά προγράμματα, με αναλυτικές

πληροφορίες για κάθε πρόγραμμα, ώστε να μπορούν να επιλέξουν το καταλληλότερο για αυτούς.

- **Αυτόματη Υποβολή και Αξιολόγηση Αιτήσεων:** Σχεδιασμός μιας διαδικασίας υποβολής αιτήσεων, όπου ο φοιτητής μπορεί να υποβάλει αίτηση για το πρόγραμμα που επιθυμεί. Η εφαρμογή θα αξιολογεί άμεσα την αίτηση με βάση προκαθορισμένους κανόνες, παρέχοντας άμεση ενημέρωση στον χρήστη για το αποτέλεσμα.
- **Ενσωμάτωση Κανόνων Drools:** Εφαρμογή σύνθετων κανόνων και λογικών ελέγχων μέσω του Drools, για την αυτοματοποιημένη και εξατομικευμένη αξιολόγηση των αιτήσεων. Οι κανόνες αυτοί μπορούν να περιλαμβάνουν κριτήρια όπως ο βαθμός πτυχίου, η εργασιακή εμπειρία, και άλλα ακαδημαϊκά ή προσωπικά κριτήρια.
- **Αυτοματισμός και Διαφάνεια στη Λήψη Αποφάσεων:** Εξασφάλιση ότι η διαδικασία επιλογής φοιτητών γίνεται με διαφανή και αμερόληπτο τρόπο, ελαχιστοποιώντας την ανάγκη για χειροκίνητη παρέμβαση και μειώνοντας τον χρόνο αναμονής των υποψηφίων.
- **Βελτίωση της Χρήστης Εμπειρίας:** Δημιουργία μιας εύχρηστης διεπαφής χρήστη (UI) που θα καθιστά τη διαδικασία επιλογής και αξιολόγησης αιτήσεων απλή, προσβάσιμη και κατανοητή από όλους τους φοιτητές, ανεξάρτητα από τις τεχνικές τους γνώσεις.

Με την ολοκλήρωση της εργασίας, αναμένεται να προσφέρει ένα λειτουργικό σύστημα που θα βελτιώσει την εμπειρία των φοιτητών και την αποτελεσματικότητα της διαδικασίας επιλογής, ενώ ταυτόχρονα θα μειώσει το λειτουργικό κόστος και τον φόρτο εργασίας των διαχειριστών των μεταπτυχιακών προγραμμάτων.

1.3 Βασικοί ορισμοί

Προτού γίνει η παρουσίαση της εργασίας θα αναλυθούν συνοπτικά μερικοί από τους βασικούς ορισμούς που χρησιμοποιούνται στην εργασία. Θα δοθεί μια σύντομη περιγραφή της γλώσσας προγραμματισμού που χρησιμοποιήθηκε, του framework το οποίο επιλέχθηκε καθώς και των εργαλείων που την πλαισίωσαν

1.3.1 Java

Η Java είναι μια από τις πιο δημοφιλείς γλώσσες προγραμματισμού στον κόσμο, γνωστή για την ευελιξία, τη σταθερότητα και την πλατφόρμα-ανεξαρτησία της (write once, run anywhere). Δημιουργήθηκε από την Sun Microsystems (τώρα ιδιοκτησία της Oracle) το 1995 και χρησιμοποιείται ευρέως για την ανάπτυξη εφαρμογών σε διάφορους τομείς, όπως επιτραπέζιες εφαρμογές, εφαρμογές για Android, διακομιστές και υπηρεσίες ιστού. Η Java υποστηρίζει αντικειμενοστραφή προγραμματισμό και είναι γνωστή για το μοντέλο της βασισμένο στη διαχείριση μνήμης μέσω του "Garbage Collector".

1.3.2 Springboot

Το Spring Boot είναι ένα πλαίσιο (framework) για τη γλώσσα Java, το οποίο δημιουργήθηκε για να απλοποιήσει την ανάπτυξη εφαρμογών επιχειρηματικής κλίμακας. Είναι μέρος του μεγαλύτερου οικοσυστήματος του Spring Framework και επιτρέπει τη γρήγορη και εύκολη δημιουργία εφαρμογών χωρίς την ανάγκη για πολλή διαμόρφωση (configuration). Το Spring Boot διευκολύνει την ανάπτυξη εφαρμογών που βασίζονται σε μικροϋπηρεσίες

(microservices) και παρέχει εργαλεία για τη διαχείριση εξαρτήσεων, την ασφάλεια, και την αλληλεπίδραση με βάσεις δεδομένων. Ενσωματώνει επίσης έναν ενσωματωμένο διακομιστή (π.χ., Tomcat), επιτρέποντας την εκτέλεση εφαρμογών Java ως αυτόνομες εφαρμογές.

1.3.3 Drools

Το Drools είναι ένα σύστημα διαχείρισης επιχειρηματικών κανόνων (Business Rules Management System - BRMS) που επιτρέπει την εύκολη δημιουργία και διαχείριση σύνθετων κανόνων λήψης αποφάσεων σε εφαρμογές. Είναι γραμμένο σε Java και χρησιμοποιείται για την αυτοματοποίηση επιχειρηματικών αποφάσεων και τη διαχείριση σύνθετων λογικών διαδικασιών. Με το Drools, οι κανόνες μπορούν να γραφτούν με απλή γλώσσα και να συντηρούνται ανεξάρτητα από την κύρια λογική της εφαρμογής, κάνοντας το σύστημα ευέλικτο και επεκτάσιμο. Στο πλαίσιο της εφαρμογής σου, το Drools χρησιμοποιείται για την άμεση αξιολόγηση των αιτήσεων φοιτητών σε μεταπτυχιακά προγράμματα.

1.3.4 HTML

Η HTML (Hypertext Markup Language) είναι η τυπική γλώσσα σήμανσης που χρησιμοποιείται για τη δημιουργία ιστοσελίδων και εφαρμογών Ιστού. Παρέχει έναν δομημένο και σημασιολογικό τρόπο ορισμού του περιεχομένου και της δομής μιας ιστοσελίδας. Η HTML, χρησιμοποιεί ετικέτες για τη σήμανση στοιχείων όπως επικεφαλίδες, παραγράφους, εικόνες, συνδέσμους, φόρμες και άλλα. Αυτές οι ετικέτες καθορίζουν τον σκοπό και τη μορφοποίηση κάθε στοιχείου, επιτρέποντας στα προγράμματα περιήγησης ιστούνα αποδώσουν σωστά τη σελίδα. Με την HTML, οι προγραμματιστές μπορούν να δημιουργήσουν διαδραστικό και προσβάσιμο περιεχόμενο ιστού ενσωματώνοντας πολυμέσα, στυλ και διαδραστικότητα μέσω CSS (Cascading Style Sheets) και JavaScript. Η HTML είναι μια θεμελιώδης γλώσσα για την ανάπτυξη Ιστού και χρησιμεύει ως η ραχοκοκαλιά του Παγκόσμιου Ιστού.

1.3.5 JavaScript

Η JavaScript είναι μια γλώσσα προγραμματισμού υψηλού επιπέδου που επιτρέπει τη δυναμική και διαδραστική συμπεριφορά σε ιστότοπους. Χρησιμοποιείται κυρίως, για την ανάπτυξη ιστού από την πλευρά του client, επιτρέποντας στους προγραμματιστές να δημιουργήσουν ανταποκρίσιμες και ελκυστικές διεπαφές χρήστη. Η JavaScript συχνά ενσωματώνεται απευθείας σε σελίδες HTML και εκτελείται από προγράμματα περιήγησης Ιστού. Παρέχει ένα ευρύ φάσμα λειτουργιών, όπως χειρισμό και τροποποίηση περιεχομένου ιστοσελίδας, χειρισμό συμβάντων, υποβολή ασύγχρονων αιτημάτων σε διακομιστές (AJAX), επικύρωση φορμών και δημιουργία διαδραστικών οπτικών εφέ. Ένα από τα βασικά χαρακτηριστικά της JavaScript είναι η ικανότητά της να αλληλοεπιδρά με το Document Object Model (DOM), το οποίο αντιπροσωπεύει τη δομή μιας σελίδας HTML. Η JavaScript μπορεί να έχει δυναμική πρόσβαση και να τροποποιεί στοιχεία του DOM, επιτρέποντας στους προγραμματιστές να δημιουργούν δυναμικές και διαδραστικές εφαρμογές Ιστού. Συνολικά, η JavaScript είναι μια ευέλικτη και ισχυρή γλώσσα για την ανάπτυξη ιστού. Η

ικανότητά του να προσθέτει διαδραστικότητα και δυναμική συμπεριφορά σε ιστοσελίδες την καθιστά απαραίτητο εργαλείο για τη δημιουργία σύγχρονων και ελκυστικών εμπειριών χρηστών στον Ιστό.

1.3.6 CSS

Το CSS (Cascading Style Sheets) είναι μια γλώσσα επικαλυπτόμενων φύλλων στυλ που χρησιμοποιείται για την περιγραφή της παρουσίασης και της οπτικής διάταξης των ιστοσελίδων. Παρέχει έναν τρόπο διαχωρισμού του περιεχομένου και της δομής ενός εγγράφου HTML από την οπτική του εμφάνιση. Με το CSS, οι προγραμματιστές μπορούν να ελέγχουν τα χρώματα, τις γραμματοσειρές, τα κενά, την τοποθέτηση και άλλες οπτικές πτυχές των στοιχείων σε μια ιστοσελίδα. Το CSS το επιτυγχάνει αυτό ορίζοντας κανόνες που στοχεύουν συγκεκριμένα στοιχεία HTML ή ομάδες στοιχείων και εφαρμόζοντας στυλ σε αυτά. Ακόμα, προσφέρει ένα ευρύ φάσμα δυνατοτήτων στυλ, συμπεριλαμβανομένης της ρύθμισης χρωμάτων φόντου, καθορισμού στυλ και μεγεθών γραμματοσειράς, προσαρμογή περιθωρίων και paddings, έλεγχο ιδιοτήτων διάταξης όπως float και flexbox και κινούμενα στοιχεία. Υποστηρίζει αποκριτικό σχεδιασμό, επιτρέποντας στους προγραμματιστές να προσαρμόσουν τη διάταξη και το στυλ με βάση διαφορετικά μεγέθη οθόνης και συσκευές. Διαδραματίζει καθοριστικό ρόλο στη διαμόρφωση της εμπειρίας χρήστη και της παρουσίασης των ιστοσελίδων.

1.3.7 MySql Workbench

Το MySQL Workbench είναι ένα ολοκληρωμένο εργαλείο διαχείρισης και σχεδίασης βάσεων δεδομένων για τον διακομιστή MySQL. Παρέχει ένα γραφικό περιβάλλον χρήστη (GUI) που επιτρέπει στους διαχειριστές βάσεων δεδομένων και τους προγραμματιστές να δημιουργούν, να διαχειρίζονται και να εκτελούν ερωτήματα σε βάσεις δεδομένων MySQL. Το Workbench υποστηρίζει διάφορες λειτουργίες, όπως τη μοντελοποίηση δεδομένων, τη διαχείριση χρηστών, την εκτέλεση SQL ερωτημάτων και την παρακολούθηση της απόδοσης του διακομιστή. Είναι ένα εργαλείο χρήσιμο για τη διαχείριση και την ανάπτυξη βάσεων δεδομένων σε μικρά και μεγάλα συστήματα.

1.4 Παραδοτέα της εργασίας

Παραδοτέα της εργασίας αποτελούν, η παρούσα αναλυτική παρουσίαση της πτυχιακής, ένα αρχείο zip με όλο το project καθώς και ένα link για το GitHub repository.

Κεφάλαιο 2ο

2 Σχεδίαση της εφαρμογής

Σε αυτό το κεφάλαιο θα αναλυθεί η χρηστικότητα της εφαρμογής αλλά και οι περιπτώσεις χρήσεις που μπορούν να προκύψουν από τους δυο χρήστες με την βοήθεια των διαγραμμάτων UML. Τα διαγράμματα UML (Unified Modeling Language) αποτελούν ένα πρότυπο γλώσσας μοντελοποίησης που χρησιμοποιείται για τον σχεδιασμό, την ανάλυση και την κατανόηση συστημάτων λογισμικού.

Τα διαγράμματα UML παρέχουν διάφορους τύπους διαγραμμάτων, όπως:

- Διαγράμματα Κλάσεων (Class Diagrams): Παρουσιάζουν τις κλάσεις, τα αντικείμενα και τις σχέσεις μεταξύ τους, αποτελώντας ένα συνολικό σχήμα της δομής του συστήματος.
- Διαγράμματα Ακολουθίας (Sequence Diagrams): Περιγράφουν την αλληλεπίδραση μεταξύ αντικειμένων και την αλληλουχία εκτέλεσης των μεθόδων.
- Διαγράμματα Δραστηριότητας (Activity Diagrams): Αναπαριστούν τη ροή εργασιών και τις διαδικασίες ενός συστήματος, με διακλαδώσεις, συγχρονισμούς και συνδέσεις.
- Διαγράμματα Πακέτων (Package Diagrams): Οργανώνουν τα στοιχεία του συστήματος σε πακέτα και απεικονίζουν τις σχέσεις μεταξύ τους.
- Διαγράμματα περιπτώσεων χρήσης (Use Case Diagrams): Παρουσιάζει τους διάφορους ρόλους (actors) που αλληλεπιδρούν με το σύστημα και τις διάφορες λειτουργίες (use cases) που παρέχονται από το σύστημα.

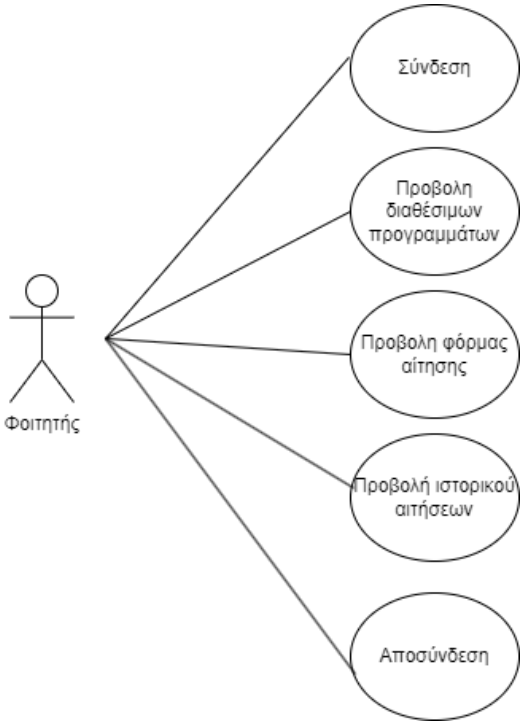
Με τη βοήθεια των διαγραμμάτων UML, οι προγραμματιστές και οι αναλυτές μπορούν να κατανοήσουν και να επικοινωνήσουν μεταξύ τους για τον σχεδιασμό του συστήματος, την ανίχνευση προβλημάτων, την πρόταση βελτιώσεων και την ανάπτυξη λογισμικού με μεγαλύτερη ακρίβεια και αποτελεσματικότητα.

Στην πτυχιακή αυτή εργασία θα χρησιμοποιηθούν τα διαγράμματα Use Case και Activity Diagrams.

2.1 Χρήση διαγραμμάτων UML

2.1.1 Διαγράμματα περιπτώσεων χρήσης (Use Case Diagrams)

Με την χρήση των διαγραμμάτων περιπτώσεων χρήσης θα δημιουργηθεί μια εικόνα για το ποιες περιπτώσεις χρήσης υπάρχουν ανάλογα με τις επιλογές του χρήστη. Ξεκινώντας με τα διαγράμματα για την πλευρά του φοιτητή χρήστη.

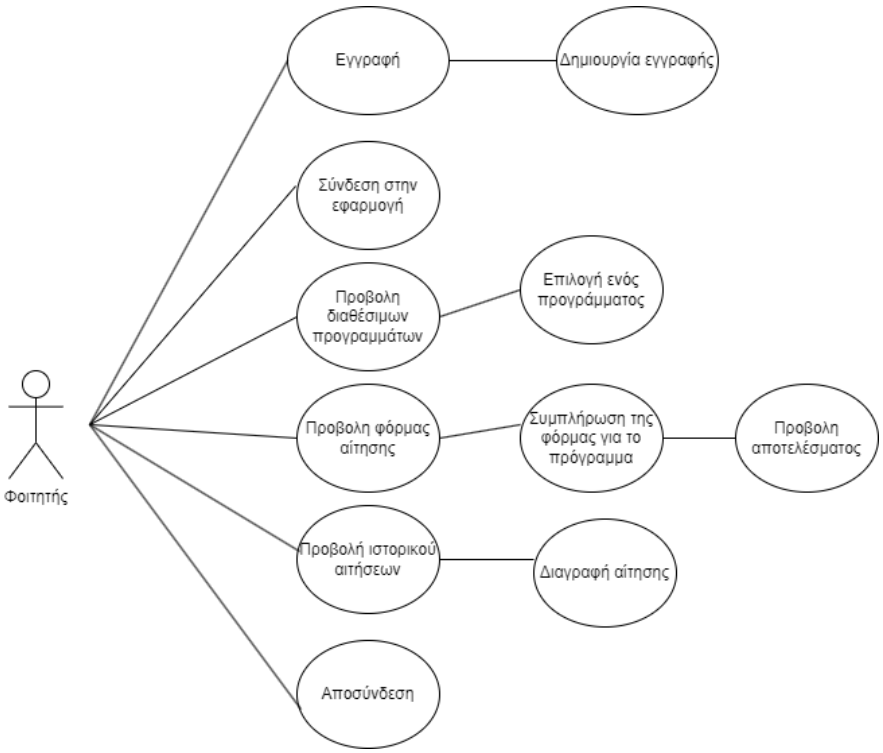


Εικόνα 1:1ο use case diagram φοιτητή

Οι πιο απλές ενέργειες που μπορεί να κάνει ο φοιτητής είναι:

1. Να συνδεθεί με τον λογαριασμό του στην εφαρμογή.
2. Να δει τα διαθέσιμα μεταπτυχιακά προγράμματα που υπάρχουν στην εφαρμογή.
3. Να ανοίξει την φόρμα για να δημιουργήσει ένα νέο αίτημα για ένα μεταπτυχιακό πρόγραμμα.
4. Να προβάλλει το πλήρες ιστορικό του.
5. Και τέλος, να αποσυνδεθεί από την εφαρμογή.

Παρακάτω θα αναλυθεί η πιο σύνθετη περίπτωση χρήσης.



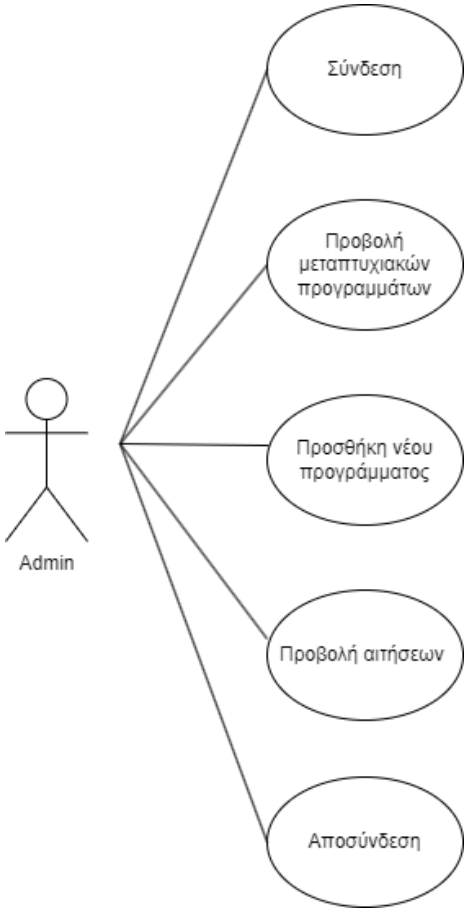
Εικόνα 2: 2ο use case diagram φοιτητή

Η περίπτωση αυτή αναπαριστά όλες τις περιπτώσεις χρήσης που μπορεί να προκύψουν από την χρήση της εφαρμογής από τον φοιτητή - χρήστη. Αναλυτικότερα, έχουμε τις εξής περιπτώσεις χρήσης:

1. **Εγγραφή:** Περίπτωση κατά την οποία ο φοιτητής στην σελίδα υποδοχής επιλεγεί να δημιουργήσει τον δικό του λογαριασμό.
2. **Δημιουργία εγγραφής:** Εδώ ο φοιτητής καλείται να συμπληρώσει την φόρμα εγγραφής και να επιλέξει την ιδιότητά του ως φοιτητής για να συνεχίσει.
3. **Σύνδεση στην εφαρμογή:** Περίπτωση κατά την οποία ο φοιτητής εισέρχεται με την χρήση των στοιχείων του στον λογαριασμό.
4. **Προβολή διαθέσιμων μεταπτυχιακών προγραμματων :** Κατα την είσοδο του στην εφαρμογή γίνεται η προβολή των διαθέσιμων προγραμμάτων , τα οποια ειναι εντασσώμενα σε έναν πίνακα , παρέχοντας στον φοιτητή τον τίτλο και μια περιγραφή για το κάθε πρόγραμμα .
5. **Επιλογή μεταπτυχιακού προγράμματος :** Ο φοιτητής αφού δει τα προγράμματα και αφού εχει διαβάσει τις περιγραφές για κάθε ενα απο αυτά του δίνεται η δυνατότητα να επιλέξει ένα απο αυτό .
6. **Προβολή φόρμα αίτησης :** Περίπτωση κατα την οποία ο φοιτητής επιλέξει ένα πρόγραμμα για να στείλει αίτηση για αυτό.
7. **Συμπλήρωση φόρμας αίτησης :** Ο φοιτητής εισάγει τα προωπικά του στοιχεία που του ζητείται για να συμπληρώσει την αίτηση του .

8. **Προβολή αποτελέσματος :** Έχοντας συμπληρώσει τα στοιχεία του και στέιλει την αίτηση , του εμφανίζεται ένα μήνυμα για το αν η αίτηση του έχει γίνει αποδεκτή ή έχει απορριθφεί.
9. **Προβολή ιστορικό αιτήσεων :** Περίπτωση κατα την οποία ο φοιτητής θέλει να δει τις αιτήσεις που έχει στείλει για τα προγράμματα.
10. **Διαγραφή αίτησης :** Ο φοιτητής έχει την δυνατότητα να διαγράψει αιτήσεις που επιθυμεί.
11. **Αποσύνδεση:** Περίπτωση χρήσης κατά την οποία ο φοιτητής αποσυνδέεται από την εφαρμογή.

Στην συνέχεια θα αναλυθούν οι περιπτώσεις χρήσης για τον διαχειριστή .Αρχικά, θα παρουσιαστεί η πιο απλή χρήση που μπορεί να προκύψει κατά την χρήση της εφαρμογής από την πλευρά του.



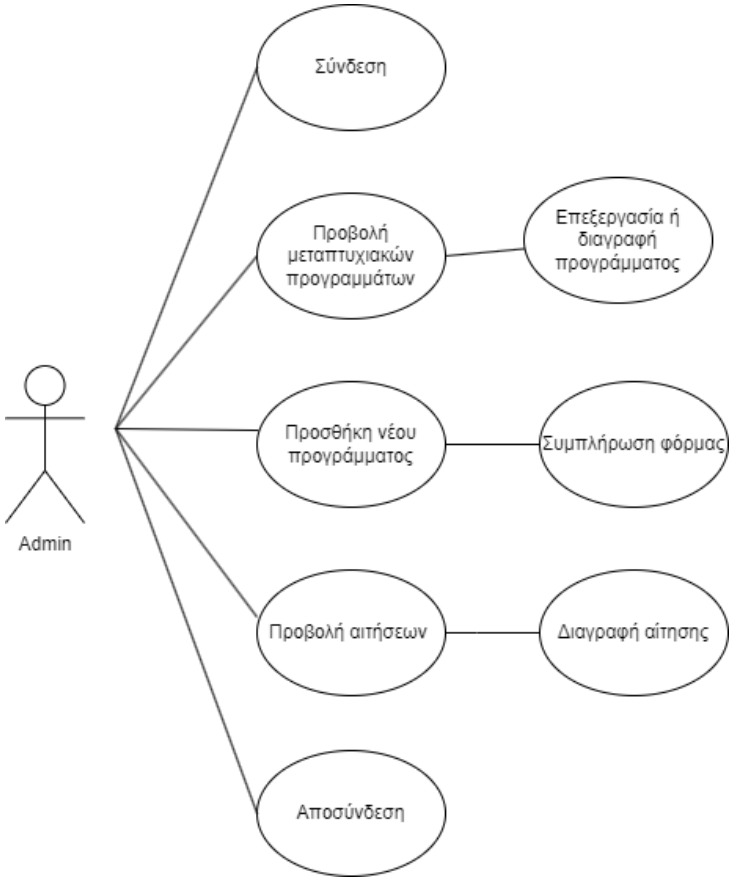
Εικόνα 3: 1ο Use case diagram διαχειρηστή

Οι ενέργειες λοιπόν που παρατηρούνται σε αυτή την περίπτωση χρήσης είναι:

1. Να συνδεθεί στην εφαρμογή.

2. Να δει όλα τα διαθέσιμα μεταπτυχιακά προγράμματα.
3. Να ανοίξει την φόρμα για την προσθήκη ενός καινούργιου μεταπτυχιακού προγράμματος.
4. Να δει όλες τις αιτήσεις που έχουν γίνει από τους φοιτητές.
5. Να αποσυνδεθεί από την εφαρμογή.

Παρακάτω είναι ένα σύνθετο διάγραμμα χρήσης οπού και θα αναλυθούν όλες οι περιπτώσεις χρήσης.



Εικόνα 4: 2ο use case diagram διαχειριστή

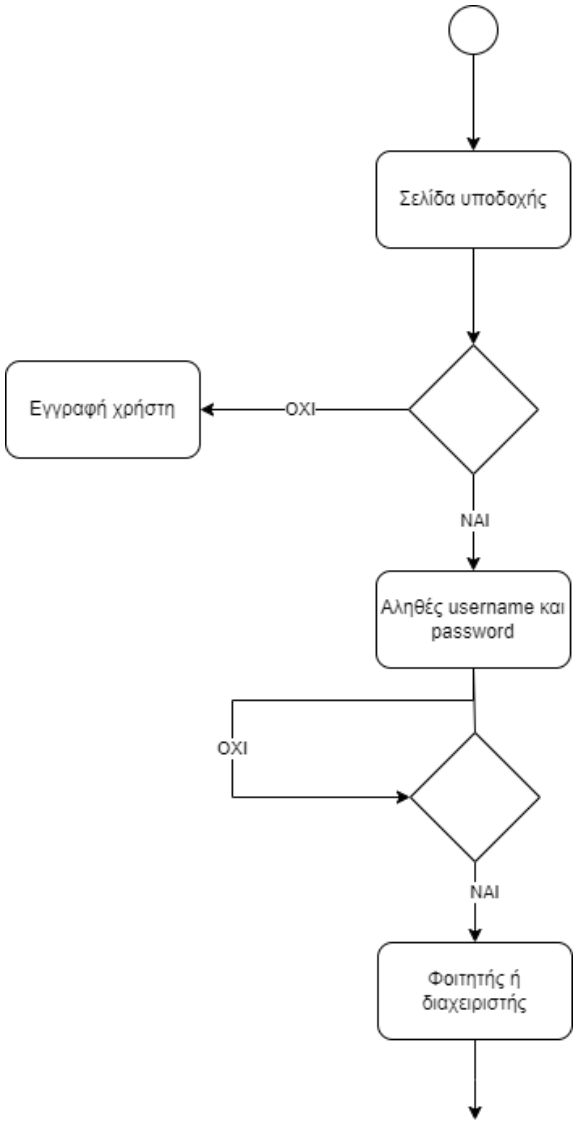
Στο διάγραμμα αυτό αναπαρίστανται όλες οι περιτπώσεις χρήσης που μπορεί να προκύψουν από την χρήση του διαχειριστή. Αναλυτικότερα λοιπόν, οι περιπτώσεις αυτές είναι:

1. **Σύνδεση στην εφαρμογή:** Περίπτωση κατά την οποία ο διαχειριστής εισέρχεται με την χρήση των στοιχείων του στον λογαριασμό.
2. **Προβολή μεταπτυχιακών προγραμμάτων :** Του εμφανίζεται ένας πίνακας με τα διαθέσιμα μεταπτυχιακά προγράμματα .
3. **Επεξεργασία ή διαγραφή προγράμματος :** Περίπρωση κατα την οποία ο διαχειριστής θέλει να επεξεγραστέι κάποιο πρόγραμμα ή ακόμα και να το διαγράψει.
4. **Προσθήκη νέου προγράμματος :** Περίπρωση κατα την οποία ο διαχειριστής θέλει να προσθέσει ένα νέο μεταπτυχιακό πρόγραμμα.
5. **Συμπλήρωση φόρμας :** Συμπληρώνει τα στοιχεία του καινούργιου μεταπτυχιακού προγράμματος .
6. **Προβολή αιτήσεων :** Περίπτωση κατα την οποία μπορεί να δει όλες τις αιτήσεις που έχουν γίνει από τους φοιτητές.
7. **Διαγραφή αίτησης :** Περίπτωση κατα την οποία ο διαχειριστής θέλει να διαγράψει κάποια αίτηση .
8. **Αποσύνδεση:** Περίπτωση χρήσης κατά την οποία ο διαχειριστής αποσυνδέεται από την εφαρμογή.

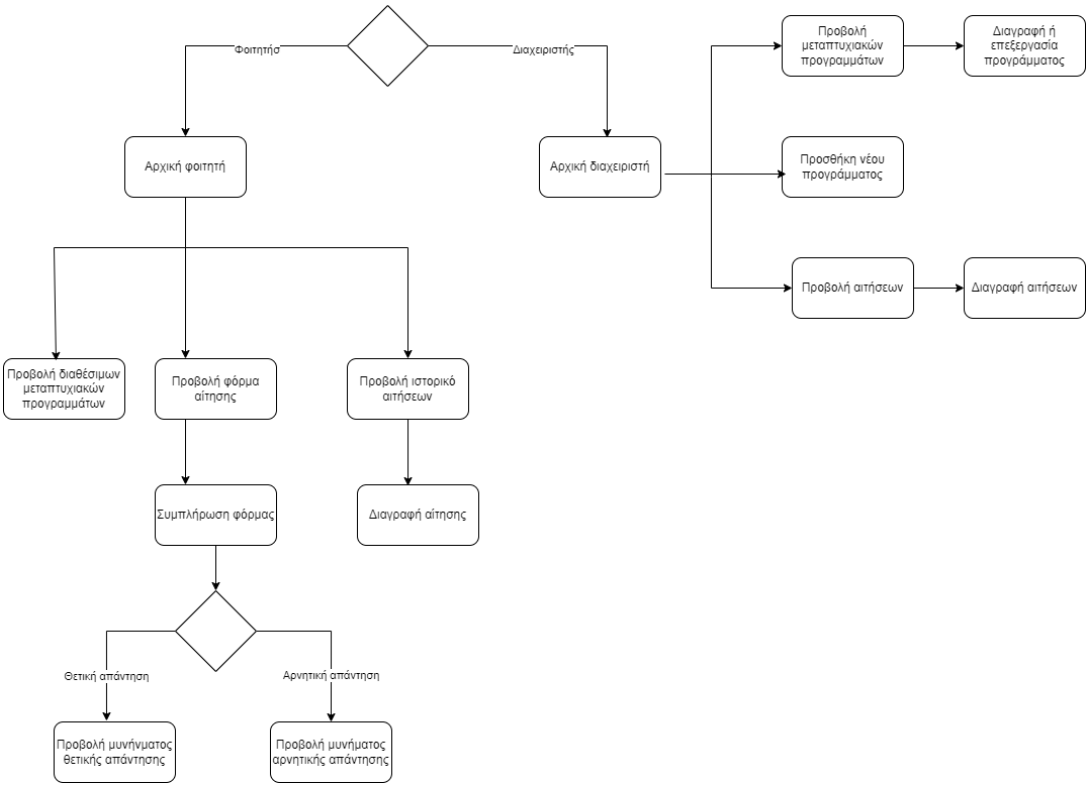
2.1.2 Διαγράμματα ροής (Activity Diagrams)

Για την καλύτερη κατανόηση της ροής της εφαρμογής που περιεγράφηκε παραπάνω με την βοήθεια των user case diagrams θα χρησιμοποιηθούν και τα διαγράμματα ροής (activity diagrams), ώστε να γίνει κατανοητή η ακριβής ροή των ενεργειών από τους δυο χρήστες.

Στο διάγραμμα ροής παρακάτω λοιπόν, παρατηρείται η ροή της εφαρμογής βάσει των αποφάσεων του χρήστη και αποτυπώνονται ξεκάθαρα τα μονοπάτια εκτέλεσης που μπορεί να παρουσιαστούν.



Εικόνα 5: 1ο διάγραμμα ροής



Εικόνα 6 : 2ο Διάγραμμα ροής swimlane

Μερικές από τις ροές εκτέλεσης που μπορούν να παρατηρηθούν είναι οι εξής:

<u>Για τον φοιτητή:</u>

1η ροή : Περίπτωση κατά την οποία γίνεται η εγγραφή του φοιτητή .

- Ο χρήστης ανοίγει την οθόνη εγγραφής.
- Ο χρήστης εισάγει τα προσωπικά στοιχεία του (όνομα,επώνυμο, email, κωδικός πρόσβασης).
- Ο χρήστης υποβάλλει τη φόρμα εγγραφής.
- Το σύστημα αποθηκεύει τα στοιχεία του χρήστη στη βάση δεδομένων.
- Το σύστημα επιβεβαιώνει την εγγραφή και ενημερώνει τον χρήστη.

2η ροή : Περίπτωση κατά την οποία ο φοιτητής θέλει να συνδεθεί στην εφαρμογή.

- Ο χρήστης ανοίγει την οθόνη σύνδεσης.
- Ο χρήστης εισάγει το email και τον κωδικό πρόσβασής του.
- Ο χρήστης υποβάλλει τη φόρμα σύνδεσης.
- Το σύστημα επαληθεύει τα στοιχεία του χρήστη.

- Το σύστημα επιβεβαιώνει τη σύνδεση και ανακατευθύνει τον χρήστη στην αρχική σελίδα.

3η ροή : Περίπτωση κατά την οποία ο φοιτητής θέλει να δει τα διαθέσιμα μεταπτυχιακά προγράμματα.

- Το σύστημα ανακτά τα διαθέσιμα μεταπτυχιακά προγράμματα από τη βάση δεδομένων.
- Το σύστημα εμφανίζει τη λίστα με τα μεταπτυχιακά προγράμματα στον χρήστη.
- Ο χρήστης επιλέγει το μεταπτυχιακό πρόγραμμα που επίθυμει να υποβάλει αίτηση.

4η ροή : Περίπτωση κατά την οποία ο φοιτητής θέλει να κάνει αίτηση για το μεταπτυχιακό πρόγραμμα.

- Ο χρήστης επιλέγει το μεταπτυχιακό πρόγραμμα για το οποίο θέλει να υποβάλει αίτηση.
- Ο χρήστης συμπληρώνει τη φόρμα αίτησης με τα απαραίτητα στοιχεία (π.χ. προσωπικά στοιχεία, βαθμολογία πτυχίου,εισόδημα,πόλη).
- Ο χρήστης υποβάλλει τη φόρμα αίτησης.
- Το σύστημα αποθηκεύει την αίτηση στη βάση δεδομένων.
- Το σύστημα επιβεβαιώνει την υποβολή της αίτησης και ενημερώνει τον χρήστη.

5η ροή : Περίπτωση κατά την οποία ο φοιτητής θέλει να δεί το ιστορικό των αιτήσεων και να διαγράψει κάποια απο αυτές.

- Ο χρήστης επιλέγει την επιλογή "Προβολή Αιτήσεων".
- Το σύστημα ανακτά τα αποτελέσματα των αιτήσεων του χρήστη από τη βάση δεδομένων.
- Το σύστημα ανακτά τα αποτελέσματα των αιτήσεων (δεκτός/απορριφθείς) στον χρήστη.
- Ο χρήστης πατώντας το κουμπί διαγραφή που υπάρχει δίπλα απο κάθε αίτηση.

<u>Για τον διαχειριστή :</u>

1η ροή : Περίπτωση κατα την οποία ο διαχειριστής θέλει να δει τα διαθέσιμα μεταπτυχιακά προγράμματα και να επεξεργαστεί ή να διαγράψει κάποιο απο αυτά.

- Είσοδο στην εφαρμογή μέσω της φόρμας εισόδου και με χρήση των στοιχείων που του έχουν δοθεί.
- Άνοιγμα του πίνακα με τα υπάρχοντα μεταπτυχιακά προγράμματα.
- Επιλογή του προγράμματος προς επεξεργασία.
- Επεξεργασία ή διαγραφή αυτού μέσω της φόρμας που θα δει ο χρήστης.
- Αποσύνδεση από την πλατφόρμα.

2η ροή : Περίπτωση κατά την οποία ο διαχειριστής θέλει να προσθέσει ένα νέο μεταπτυχιακό πρόγραμμα.

- Είσοδο στην εφαρμογή μέσω της φόρμας εισόδου και με χρήση των στοιχείων που του έχουν δοθεί.
- Χρήση της φόρμας για προσθήκη νέου πόρου.
- Συμπλήρωση αυτής καταλλήλως.
- Αποσύνδεση από την εφαρμογή.

3η ροή : Περίπτωση κατά την οποία ο διαχειριστής θέλει να δεί το ιστορικό αιτήσεων και να διαγράψει κάποια από αυτές.

- Είσοδο στην εφαρμογή μέσω της φόρμας εισόδου και με χρήση των στοιχείων που του έχουν δοθεί.
- Άνοιγμα του πίνακα με τις αιτήσεις.
- Επιλογή της αίτησης που θέλει να διαγράψει.
- Αποσύνδεση από την πλατφόρμα

2.2 Ανάλυση βάσης δεδομένων.

Προτού γίνει η ανάλυση της βάσης δεδομένων, θα ήταν χρήσιμο να γίνει αναφορά στον ορισμό της βάσης δεδομένων και στα σχεσιακά μοντέλα. Ως βάση δεδομένων λοιπόν, ορίζεται μια οργανωμένη συλλογή δεδομένων που αποθηκεύονται και οργανώνονται με τρόπο που επιτρέπει την αποτελεσματική ανάκτηση, ενημέρωση και διαχείριση των δεδομένων αυτών.

Ενώ, το σχεσιακό μοντέλο, είναι μια προσέγγιση για την οπτικοποίηση του σχεδιασμού της βάσης δεδομένων αλλά και των σχέσεων που υπάρχουν αναμεσά στους πίνακες της. Βασίζεται στη θεωρία των συσχετίσεων και χρησιμοποιεί τον πίνακα ως το κύριο δομικό στοιχείο. Οι σχέσεις μεταξύ των πινάκων καθορίζουν τον τρόπο με τον οποίο συνδέονται τα δεδομένα και παρέχουν τη δυνατότητα ανάκτησης δεδομένων από πολλαπλούς πίνακες μέσω συνδέσμων.

Παρακάτω λοιπόν, παρουσιάζεται το ER διάγραμμα της βάσης δεδομένων, ώστε με την χρήση αυτού να αναλύσουμε την σχεδίασή της και των σχέσεων μεταξύ των πινάκων που έχουν δημιουργηθεί.

Εικόνα 7: Βάση δεδομένων.

Παρατηρώντας το διάγραμμα της βάσης φαίνεται πως υπάρχουν 3 μοντέλα : **PROGRAMS** , **APPLICATIONS**,**USERS**.

Αναλυτικά τα χαρακτηριστικά των τριών αυτών μοντέλων είναι τα εξής:

Πίνακας USERS :

1. Id : Αποθηκεύει το ID του κάθε χρήστη το οποίο είναι και μοναδικό.
2. Email : Αποθηκεύει το email του χρήστη το οποίο είναι και αυτό μοναδικό για κάθε χρήστη, το οποίο είναι αυτό που θα χρησιμοποιήσει κατά την σύνδεση του.
3. Firstname : Αποθηκεύει το μικρό όνομα του χρήστη.
4. Lastname : Αποθηκεύει το επίθετο του χρήστη.
5. Password : Αποθηκεύει κρυπτογραφημένα τον κωδικό πρόσβασης που επιλέγει ο χρήστης.
6. Role : Είναι τύπου Boolean και αποθηκεύει αν ο χρήστης είναι φοιτητής η όχι.

Πίνακας PROGRAMS:

1. Id : Αποθηκεύει το ID του κάθε πραγράμματος το οποίο είναι και μοναδικό.
2. City : Αποθηκεύει την πόλη που θα βρίσκεται το κάθε πρόγραμμα .
3. Degree_score: Αποθηκεύει τον ελάχιστο βαθμό πτυχίου που πρέπει να έχει ο φοιτητής για να γίνει αποδεκτός.
4. Description : Αποθηκεύει μία μικρή περιγραφή για το κάθε πρόγραμμα.
5. Name : Αποθηκεύει το όνομα του προγράμματος.
6. Remote : Είναι τύπου Boolean και αποθηκεύει αν το πρόγραμμα μπορεί να γίνει εξ αποστάσεως ή όχι.

7. Required_income : Αποθηκεύει το απαιτούμενο μηνιαίο εισόδημα που πρέπει να έχει ο φοιτητής για να γίνει αποδεκτός.

Πίνακας APPLICATIONS :

1. Id : Αποθηκεύει το ID της κάθε αίτησης το οποίο είναι και μοναδικό.
2. User_id : Το χαρακτηριστικό user_id είναι το foreign key με το οποίο συνδέεται ο πίνακας αυτός με τον πίνακα USERS.
3. Program_id: Το χαρακτηριστικό program_id είναι το foreign key με το οποίο συνδέεται ο πίνακας αυτός με τον πίνακα PROGRAMS.
4. City : Αποθηκεύει την πόλη που θα βρίσκεται ο κάθε φοιτητής .
5. Accepted : Είναι τύπου Boolean και αποθηκεύει αν η αίτηση έχει γίνει αποδεκτή ή όχι.
6. Degree_score: Αποθηκεύει τον βαθμό πτυχίου που εισάγει ο φοιτητής.
7. Income : Αποθηκεύει το μηνιαίο εισόδημα που έχει ο φοιτητής.
8. Remote : Είναι τύπου Boolean και αποθηκεύει αν ο φοιτητής έχει επιλέξει να το κάνει εξ αποστάσεως ή όχι.
9. Application_date_time : Αποθηκεύει την ώρα και την ημερομηνία που έστειλε την αίτηση ο φοιτητής.

Κεφάλαιο 3ο

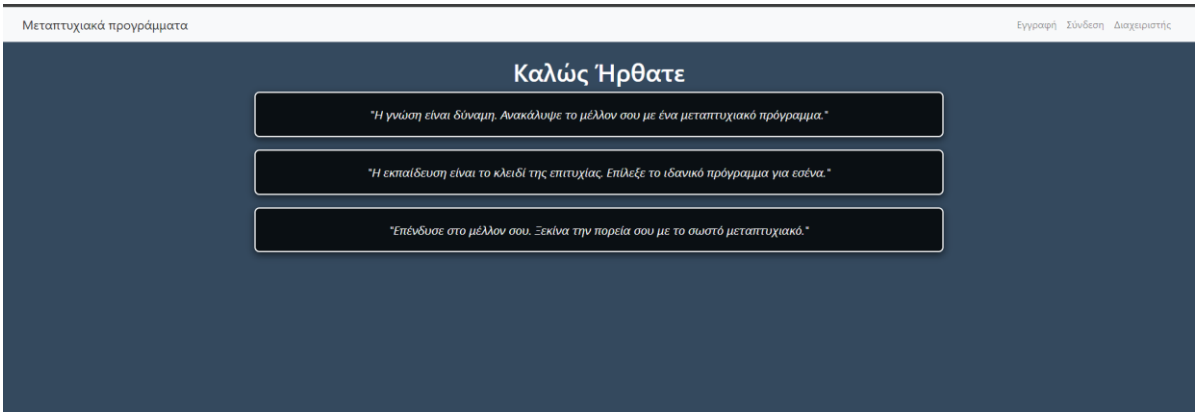
3 Αναλυτική περιγραφή της υλοποίησης της εφαρμογής

Σε αυτό το κεφάλαιο θα γίνει η παρουσίαση της υλοποίησης της εφαρμογής με αποσπάσματα

τόσο από το display, ώστε να γίνει κατανοητός ο τρόπος χρήσης της εφαρμογής σε κάθε πιθανό σενάριο όσο και από τον κώδικα που θα εξηγηθεί το πως διαχειρίζεται κάθε ενέργεια.

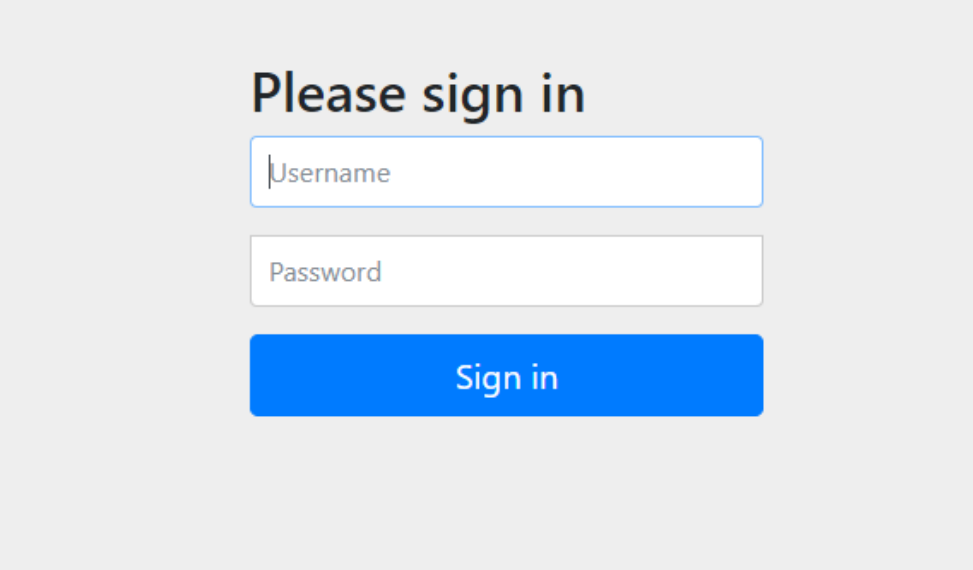
3.1 Γενικές περιοχές εφαρμογής

Αρχικά θα αναλυθούν οι περιοχές που είναι κοινές για τους δυο χρήστες της εφαρμογής τόσο για τον φοιτητή όσο και για τον διαχειριστή. Οι περιοχές αυτές είναι η σελίδα υποδοχής όπου ο χρήστης επιλέγει αν θέλει να κάνει εγγραφή, αν δεν έχει ήδη κάνει ή σύνδεση αν διαθέτει λογαριασμό.



Εικόνα 8: Σελίδα υποδοχής

Στην παραπάνω εικόνα φαίνεται λοιπόν η σελίδα υποδοχής. Στην περίπτωση που ο χρήστης επιλέξει να κάνει σύνδεση ανακατευθύνεται στην παρακάτω περιοχή, όπου με χρήση των στοιχείων του συνδέεται στην εφαρμογή.



Εικόνα 9: Σελίδα σύνδεσης

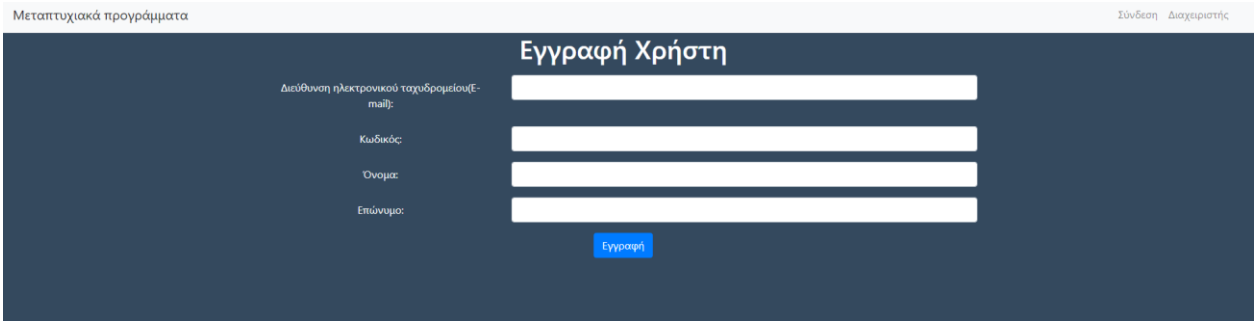
Στην παραπάνω εικόνα, φαίνεται η περιοχή σύνδεσης για τους χρήστες.

Εικόνα 10: Μηνύματα λάθους

Για κάθε σφάλμα που συμβαίνει κατά την διάρκεια της σύνδεσης του χρήστη υπάρχουν τα κατάλληλα μηνύματα προς τον χρήστη στα αντίστοιχα πλαίσια, όπως αυτά παρατηρούνται στις εικόνες παραπάνω.

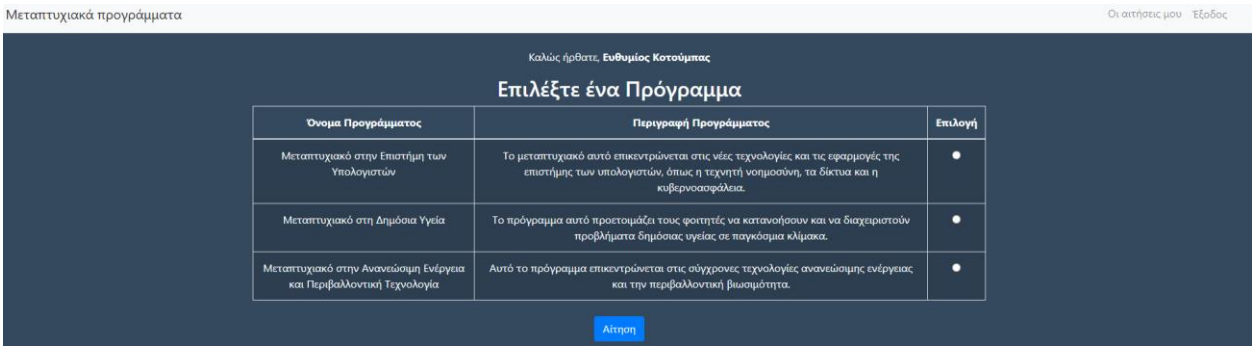
3.2 Παρουσίαση περιοχών φοιτητή

Αφού παρουσιάστηκαν οι γενικές περιοχές, θα γίνει παρουσίαση των περιοχών του κάθε χρήστη ξεκινώντας με τον φοιτητή.



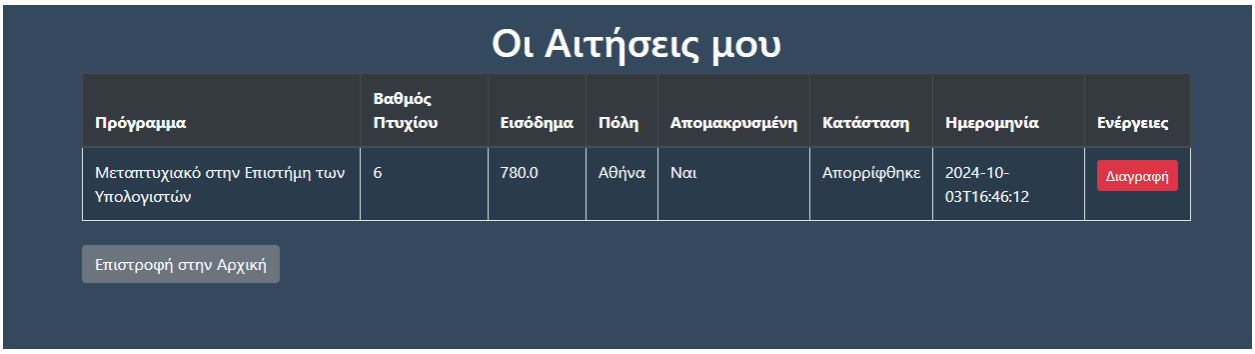
Εικόνα 11: Εγγραφή Χρήστη

Στην φόρμα αυτή, η εφαρμογή ζητάει από τον χρήστη να ορίσει ένα το email του, το ονοματεπώνυμο του καθώς και την δημιουργία κωδικού πρόσβασης Χρησιμοποιήθηκε αυτή η φόρμα εγγραφής έτσι ώστε να οπτικοποιηθεί η διαδικασία, θα μπορούσε εύκολα να παρακαμφθεί η διαδικασία με χρήση των στοιχείων του πανεπιστημίου για τους δυο χρήστες. Σε αυτή την περίπτωση δεν θα υπήρχε ανάγκη φόρμας εγγραφής και η ιδιότητα των δυο χρηστών θα ήταν διαχειρίσιμη από το σύστημα μέσω των μητρώων τους.



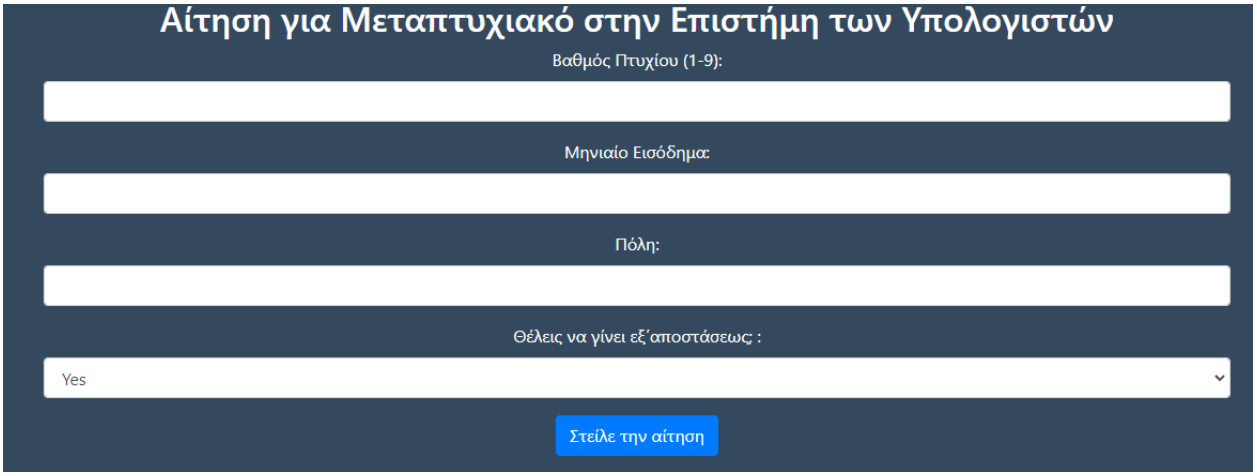
Εικόνα 12: Αρχική σελίδα φοιτητή

Στην παραπάνω είκονα φαίνεται η αρχική σελίδα του φοιτητή,η οποία περιέχει έναν πίνακα με τα διαθέσιμα μεταπτυχιακά προγράμματα , που έχει την δυνατότητα να επιλέξει ένα από αυτά για να στείλει αίτηση. Καθώς κάνοντας κλικ στο κουμπί «Οι αιτήσεις μου» , μπορεί να δει το ιστορικό των αιτήσεών του.



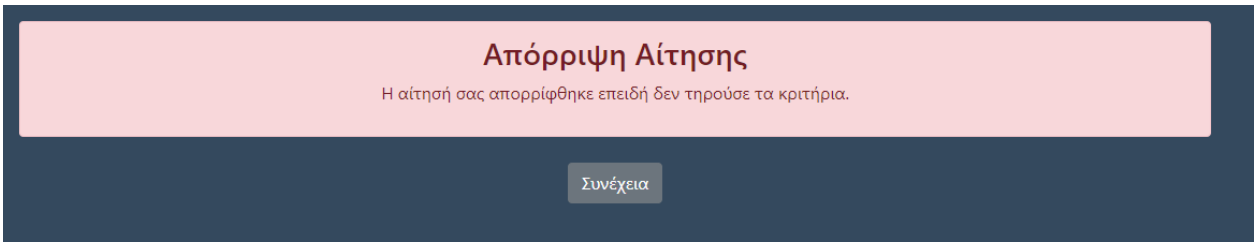
Εικόνα 13: Ιστορικό αιτήσεων

Στην παραπάνω εικόνα μπορεί ο φοιτητής να δεί και να διαγράψει εάν επιθυμεί τις αιτήσεις του .



Εικόνα 14: Φόρμα αίτησης

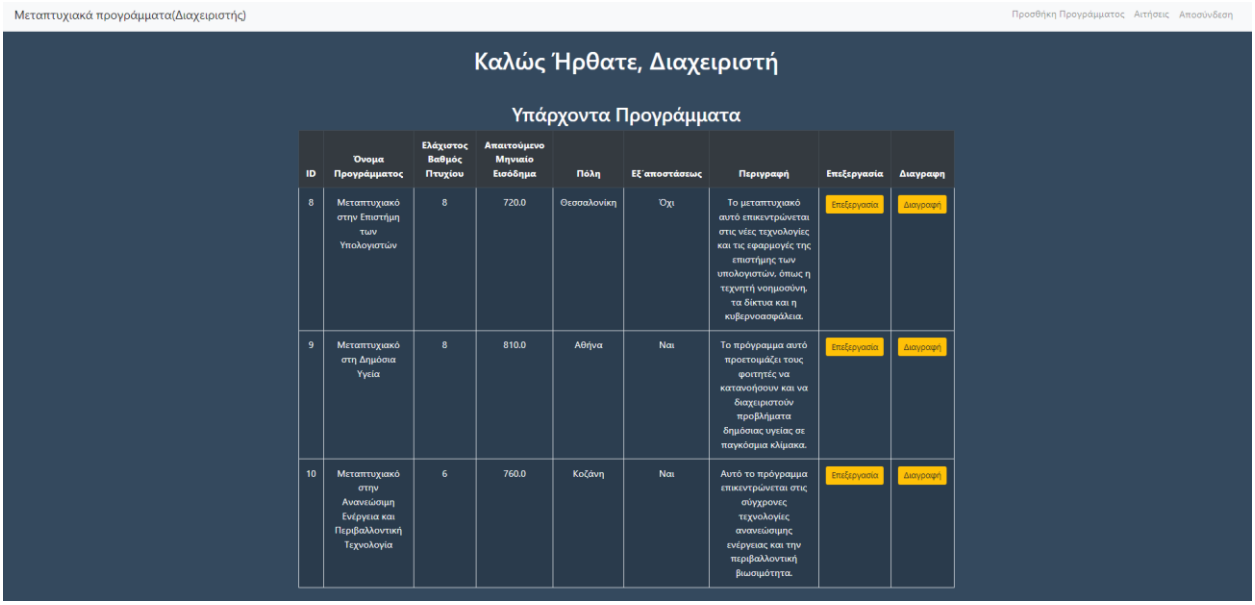
Στην παραπάνω εικόνα ο φοιτητής συμπληρώνει την φόρμα αίτησης με τα στοιχεία που του ζητούνται για το μεταπτυχιακό προόγραμμα που έχει επιλέξει , αφού την συμπληρώσει πατώντας το κουμπί «Στείλε την αίτηση» , ανακατευθύνεται στην παρακάτω σελίδα οπου του γίνεται γνωστό αν η αίτηση του εγκρίθηκε ή απορρίφθηκε.



Εικόνα 15: Μήνυμα απόρριψης ή αποδοχής αίτησης

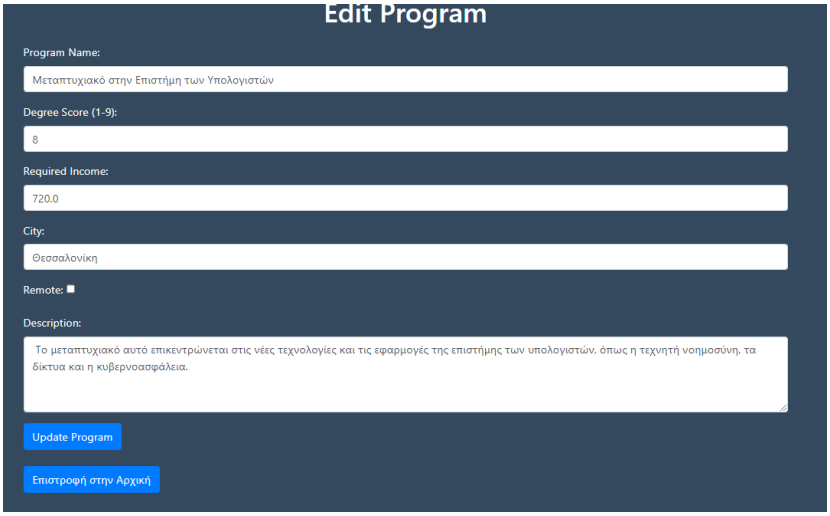
3.3 Παρουσίαση περιοχών διαχειριστή

Στην εφαρμογή υπάρχει ένας διαχειριστής όπου τα στοιχεία του για να συνδεθεί είναι(email : admin@example.com ,password : admin123). Κλείνοντας τις παρουσιάσεις των περιοχών υπάρχει ο διαχειριστής, ο οποίος αφού συνδεθεί ανακατευθύνεται στην βασική του περιοχή όπως φαίνεται παρακάτω.

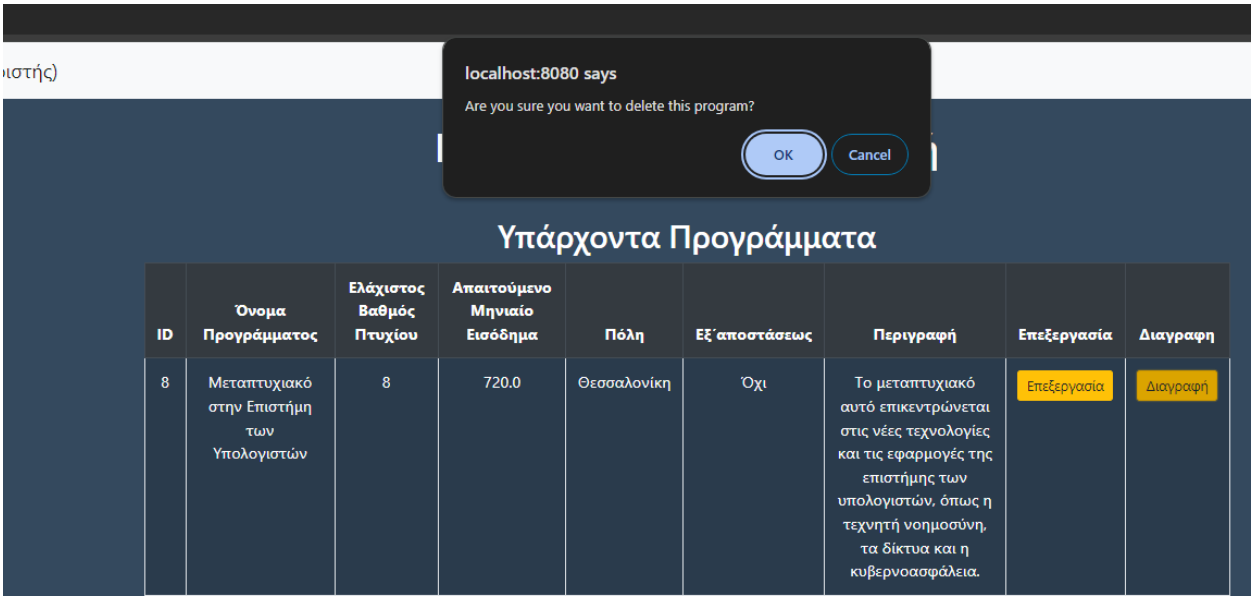


Εικόνα 16: Αρχική σελίδα διαχείριστη

Στην παραπάνω εικόνα είναι η αρχική του διαχειριστή , η οποία περιέχει έναν πίνακα με τα διαθέσιμα μεταπτυχιακά προγράμματα τα οποία μπορεί να επεξεργαστεί ή να διαγράψει. Καθώς υπάρχει και η επιλογή να δεί τις αιτήσεις που έχουν γίνει από τους φοιτητές και να προσθέσει ένα νεο μεταπτυχιακό πρόγραμμα.

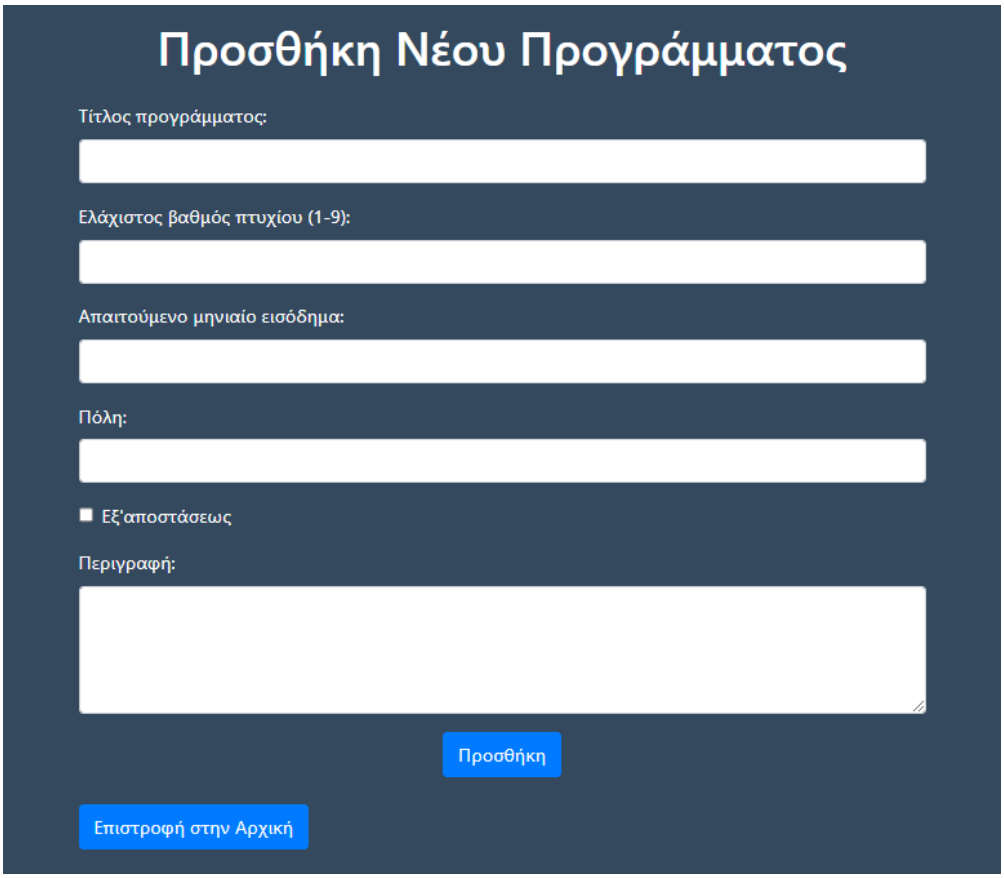


Εικόνα 17: Φόρμα επεξεργασίας προγράμματος

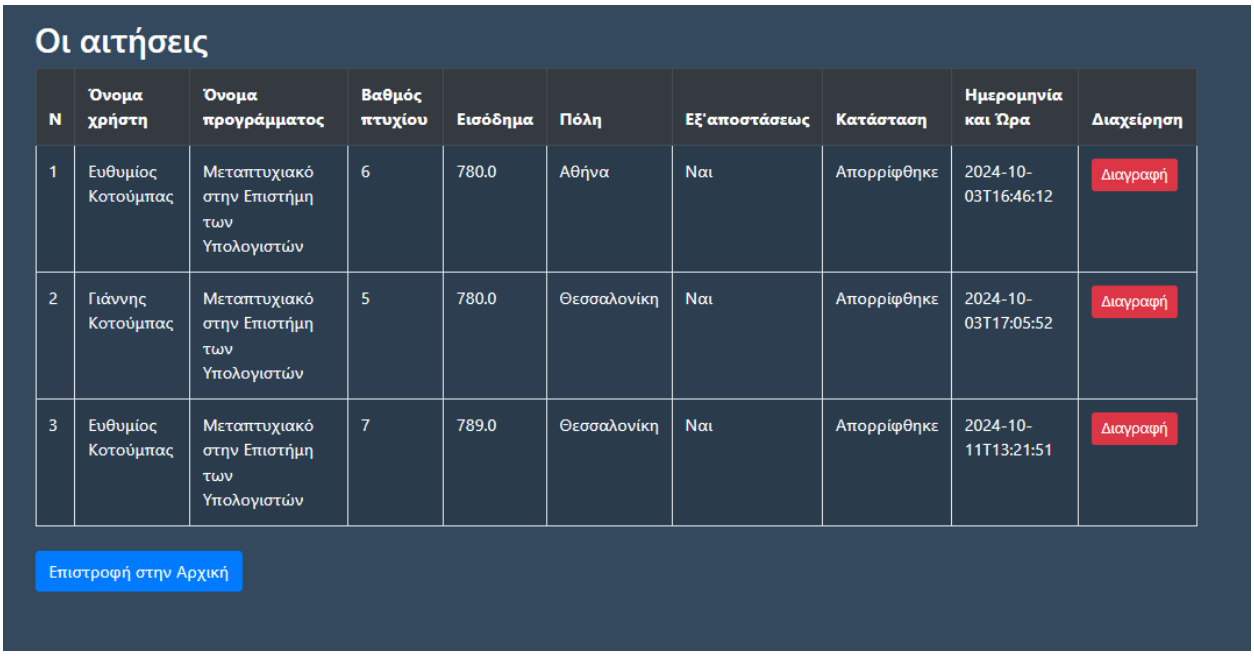


Εικόνα 18: Μήνυμα διαγραφής προγράμματος

Στην παρακάτω εικόνα βρίσκεται η φόρμα την οποία πρέπει να συμπληρώσει ο διαχειριστής για να προσθέσει ένα καινούργιο μεταπτυχιακό πρόγραμμα.

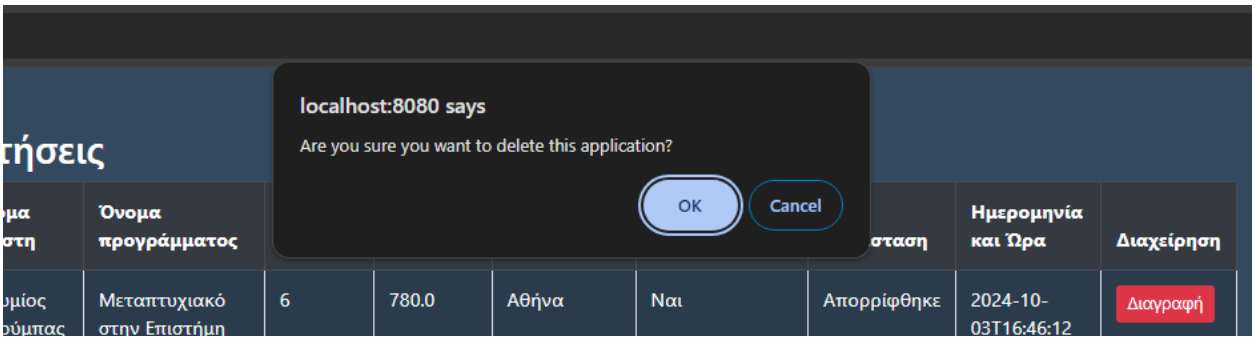


Εικόνα 19: Προσθήκη νέου προγράμματος



Εικόνα 20: Οι αιτήσεις των φοιτητών

Στην παραπάνω εικόνα , ο διαχειριστής βλέπει όλες τις αιτήσεις που έχουν γίνει απο τους φοιτητές , καθώς μπορεί να τις διαγράψει.



Εικόνα 21: Μήνυμα διαγραφής αίτησης

3.4 Παρουσίαση κώδικα

Στην 4η και τελευταία υπο-ενότητα της παρουσίασης τη εφαρμογής θα γίνει ανάλυση των σημαντικότερων σημείων του κώδικα, ώστε να γίνει κατανοητό πως λειτουργούν οι σημαντικότερες ενέργειες της εφαρμογής.

3.4.1 Σχεδιασμός και υλοποίηση των κλάσεων Users,Programs και Applications

```
@Entity
@Table(name= "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique=true, length=45)
    private String email;

    @Column(nullable = false, length=60)
    private String password;

    @Column(nullable = false, length=20)
    private String firstname;

    @Column(nullable = false, length=20)
    private String lastname;

    @Column(nullable = false, length=20)
    private String role = "ROLE_USER"; // Νέα πεδία για το ρόλο του χρήστη

     @Column(name = "degree_score", nullable = true)
        private Integer degreeScore; // Accepts values from 1 to 9

        @Column(name = "income", nullable = true)
        private Double income; // Accepts numerical values for income

        @Column(name = "city", nullable = true, length = 100)
        private String city; // City where the user is located

        @Column(name = "remote", nullable = true)
        private Boolean remote; // Indicates if the user wants remote programs


      @ManyToMany
     @JoinTable(
            name = "user_programs",
            joinColumns = @JoinColumn(name = "user_id"),
            inverseJoinColumns = @JoinColumn(name = "program_id")
        )

        private List<Program> programs;

      @OneToMany(mappedBy = "user")
        private List<Application> applications;
```

Εικόνα 22: Κλάση User

Η κλάση User αναπαριστά έναν χρήστη της εφαρμογής και είναι συνδεδεμένη με τον πίνακα users στη βάση δεδομένων μέσω του Hibernate. Το πεδίο id είναι το πρωτεύον κλειδί και παράγεται αυτόματα. Το email είναι μοναδικό και υποχρεωτικό, ενώ το password, το firstname, και το lastname είναι επίσης υποχρεωτικά, με συγκεκριμένο μήκος για κάθε πεδίο. Το πεδίο role καθορίζει τον ρόλο του χρήστη, με προεπιλογή "ROLE_USER".

Επιπλέον, η κλάση περιλαμβάνει προαιρετικά πεδία όπως το degreeScore, που αντιπροσωπεύει τη βαθμολογία του χρήστη, το income, που αποθηκεύει το εισόδημά του, το

city, που δηλώνει την πόλη του, και το remote, που δείχνει αν ο χρήστης ενδιαφέρεται για εξ αποστάσεως προγράμματα. Υπάρχει σχέση ManyToMany με την κλάση Program, που σημαίνει ότι ένας χρήστης μπορεί να εγγραφεί σε πολλά προγράμματα, ενώ ταυτόχρονα ένα πρόγραμμα μπορεί να έχει πολλούς χρήστες. Επίσης, υπάρχει σχέση OneToMany με την κλάση Application, η οποία αντιπροσωπεύει τις αιτήσεις του χρήστη για τα προγράμματα. Η κλάση χρησιμοποιεί annotations του Hibernate για να διευκολύνει τη χαρτογράφηση των πεδίων σε στήλες της βάσης δεδομένων και τη διαχείριση των σχέσεων μεταξύ των οντοτήτων.Καθώς στο τέλος υπάρχουν και τα getters και setters.

```java
package com.example;

import jakarta.persistence.*;

@Entity
@Table(name = "programs")
public class Program {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true, length = 100)
    private String name;

    @Column(name = "degree_score", nullable = false)
    private Integer degreeScore; // Accepts values from 1 to 9

    @Column(name = "required_income", nullable = false)
    private Double requiredIncome; // Accepts numerical values for the income

    @Column(nullable = false, length = 100)
    private String city; // City where the program is located

    @Column(name = "remote", nullable = false)
    private Boolean remote = false; // Indicates if the program can be done remotely

    @Column(length = 255)
    private String description; // Short description of the program

    @OneToMany(mappedBy = "program", cascade = CascadeType.REMOVE, orphanRemoval = true)
    private List<Application> applications;

    @ManyToMany(mappedBy = "programs")
    private List<User> users;



    // Getters and setters
```

Εικόνα 23: Κλάση Programs

Η κλάση Program αναπαριστά ένα πρόγραμμα σπουδών και χαρτογραφείται στον πίνακα programs της βάσης δεδομένων μέσω του Hibernate. Το πεδίο id είναι το πρωτεύον κλειδί και δημιουργείται αυτόματα από τη βάση δεδομένων. Το πεδίο name αποθηκεύει το όνομα του προγράμματος, είναι μοναδικό και υποχρεωτικό, με μέγιστο μήκος 100 χαρακτήρες. Το πεδίο degreeScore απαιτεί βαθμολογία από 1 έως 9 και το requiredIncome καθορίζει το απαραίτητο εισόδημα για την ένταξη στο πρόγραμμα, και τα δύο είναι υποχρεωτικά. Το πεδίο city αποθηκεύει την πόλη στην οποία προσφέρεται το πρόγραμμα και είναι επίσης υποχρεωτικό, με μέγιστο μήκος 100 χαρακτήρες. Το πεδίο remote είναι λογικό και δηλώνει αν το πρόγραμμα μπορεί να γίνει εξ αποστάσεως, με προεπιλεγμένη τιμή false. Η περιγραφή του προγράμματος αποθηκεύεται στο πεδίο description με μέγιστο μήκος 255 χαρακτήρες. Η σχέση OneToMany με την κλάση Application δηλώνει ότι ένα πρόγραμμα μπορεί να έχει πολλές αιτήσεις, και όταν διαγράφεται ένα πρόγραμμα, οι αντίστοιχες αιτήσεις διαγράφονται αυτόματα λόγω του cascade = CascadeType.REMOVE και orphanRemoval = true. Επίσης, υπάρχει μια σχέση ManyToMany με την κλάση User, η οποία επιτρέπει

πολλούς χρήστες να συνδέονται με πολλά προγράμματα. Η κλάση χρησιμοποιεί τα annotations του Hibernate για να διευκολύνει τη χαρτογράφηση των πεδίων της σε στήλες της βάσης δεδομένων και τη διαχείριση των σχέσεών της με τις οντότητες Application και User.

```java
package com.example;

import java.time.LocalDateTime;

@Entity
@Table(name = "applications")
public class Application {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @ManyToOne
    @JoinColumn(name = "program_id", nullable = true)
    private Program program;

    @Column(name = "degree_score", nullable = false)
    private Integer degreeScore;

    @Column(name = "income", nullable = false)
    private Double income;

    @Column(nullable = false, length = 100)
    private String city;

    @Column(name = "remote", nullable = false)
    private Boolean remote;

    @Column(name = "accepted",nullable = false)
    private Boolean accepted = false;

    @Column(name = "application_date_time", nullable = false)
    private LocalDateTime applicationDateTime;
```

Εικόνα 24: Κλάση Application

Η κλάση Application αναπαριστά μια αίτηση που υποβάλλεται από έναν χρήστη για ένα πρόγραμμα και αντιστοιχίζεται στον πίνακα applications της βάσης δεδομένων μέσω του Hibernate. Το πεδίο id είναι το πρωτεύον κλειδί και παράγεται αυτόματα από τη βάση δεδομένων. Η σχέση ManyToOne με την κλάση User καθορίζει ότι κάθε αίτηση συνδέεται με έναν συγκεκριμένο χρήστη, με την αντίστοιχη ξένη στήλη user_id να είναι υποχρεωτική. Υπάρχει επίσης σχέση ManyToOne με την κλάση Program, που σημαίνει ότι μια αίτηση μπορεί να συνδέεται με ένα πρόγραμμα μέσω του πεδίου program_id, το οποίο είναι προαιρετικό. Το πεδίο degreeScore αποθηκεύει την βαθμολογία του αιτούντος και είναι υποχρεωτικό, όπως και το πεδίο income, το οποίο αποθηκεύει το εισόδημα του αιτούντος. Το πεδίο city αποθηκεύει την πόλη στην οποία βρίσκεται ο χρήστης και έχει μέγιστο μήκος 100 χαρακτήρες, ενώ το πεδίο remote δηλώνει αν η αίτηση αφορά ένα εξ αποστάσεως πρόγραμμα. Το πεδίο accepted είναι λογικό (Boolean) και δείχνει αν η αίτηση έχει γίνει αποδεκτή, με προεπιλεγμένη τιμή false. Το πεδίο applicationDateTime αποθηκεύει την ημερομηνία και την ώρα υποβολής της αίτησης και είναι υποχρεωτικό. Η κλάση χρησιμοποιεί τα annotations του Hibernate για να χαρτογραφήσει τα πεδία της σε στήλες της βάσης δεδομένων και να διαχειριστεί τις σχέσεις της με τις οντότητες User και Program.

3.4.2 Σχεδιασμός και υλοποίηση του WebSecurityConfig

Η κλάση WebSecurityConfig είναι μια ρύθμιση ασφαλείας για μια εφαρμογή που χρησιμοποιεί το Spring Security. Σκοπός της είναι να διαμορφώσει την ασφάλεια της εφαρμογής, καθορίζοντας πώς θα γίνεται ο έλεγχος ταυτότητας και εξουσιοδότησης των χρηστών.

```java
@Configuration
public class WebSecurityConfig   {

    @Bean
    UserDetailsService userDetailsService() {
        return new CustomUserDetailsService();
    }

    @Bean
    PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public DaoAuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService());
        authProvider.setPasswordEncoder(passwordEncoder());

        return authProvider;
    }

    @Bean
    SecurityFilterChain configure(HttpSecurity http) throws Exception {

        http.authenticationProvider(authenticationProvider());

        http.authorizeHttpRequests(auth ->
            auth
                .requestMatchers("/admin").hasRole("ADMIN")
                .requestMatchers("/users").authenticated()
                .anyRequest().permitAll()
            )
            .formLogin(login ->
                login.usernameParameter("email")
                .defaultSuccessUrl("/users")
                .permitAll()
            )
            .logout(logout -> logout.logoutSuccessUrl("/").permitAll()
        );

        return http.build();
    }
}
```

Εικόνα 25: Κλάση WebsecurityConfig

Η κλάση φέρει το annotation @Configuration, που δηλώνει ότι πρόκειται για μια κλάση ρυθμίσεων (configuration class).Ο ορισμός userDetailsService() δημιουργεί ένα bean που επιστρέφει ένα αντικείμενο CustomUserDetailsService, το οποίο χρησιμοποιείται για τη φόρτωση των στοιχείων του χρήστη από την βάση δεδομένων κατά τη διαδικασία του authentication.Η μέθοδος passwordEncoder() ορίζει έναν κωδικοποιητή κωδικών πρόσβασης (password encoder), εδώ χρησιμοποιείται το BCryptPasswordEncoder, που είναι ένας ισχυρός αλγόριθμος κρυπτογράφησης των κωδικών.

Η μέθοδος authenticationProvider() δημιουργεί ένα DaoAuthenticationProvider, το οποίο συνδέεται με το UserDetailsService και τον κωδικοποιητή κωδικών πρόσβασης για τον έλεγχο ταυτότητας χρηστών.Η κύρια μέθοδος configure(HttpSecurity http) είναι αυτή που ρυθμίζει τις πολιτικές ασφαλείας της εφαρμογής. Αρχικά, προσθέτει τον πάροχο ελέγχου ταυτότητας μέσω της κλήσης authenticationProvider(). Στη συνέχεια, με την μέθοδο authorizeHttpRequests(), καθορίζει ποιοι χρήστες έχουν πρόσβαση σε ποιες διευθύνσεις

(URLs): μόνο χρήστες με ρόλο "ADMIN" έχουν πρόσβαση στο /admin, οι αυθεντικοποιημένοι χρήστες έχουν πρόσβαση στο /users, και όλες οι υπόλοιπες διευθύνσεις είναι διαθέσιμες σε όλους (permitAll()).Η διαμόρφωση για τη φόρμα σύνδεσης (formLogin()) καθορίζει ότι η εφαρμογή θα χρησιμοποιεί το πεδίο "email" για το όνομα χρήστη και ότι μετά την επιτυχημένη σύνδεση, ο χρήστης θα ανακατευθύνεται στη διεύθυνση /users. Τέλος, η διαμόρφωση του logout() καθορίζει ότι μετά την αποσύνδεση, ο χρήστης θα επιστρέφει στην αρχική σελίδα.Η κλάση αυτή διασφαλίζει ότι η εφαρμογή ακολουθεί κατάλληλα πρότυπα ασφαλείας για την προστασία των δεδομένων και των χρηστών της.

3.4.3 Σχεδιασμός και υλοποίηση του Controller

Ο Controller αυτός παίζει κεντρικό ρόλο στην αλληλεπίδραση της διεπαφής χρήστη με την επιχειρησιακή λογική της εφαρμογής, επιτρέποντας στους χρήστες να εγγράφονται, να υποβάλλουν αιτήσεις και να λαμβάνουν αποφάσεις βάσει των κανόνων του συστήματος.

```
@GetMapping("")
public String viewHopePage() {
    return "index";
}

@GetMapping("/register")
public String showSignUpForm(Model model) {
    model.addAttribute("user", new User());

    return"signup_form";
}

@GetMapping("/admin")
public String showAdminPage(Model model) {
     List<Program> programs = programRepo.findAll();
        model.addAttribute("programs", programs);

    return"admin_page";
}
@PostMapping("/process_register")
public String processRegistration(User user) {
    BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();
    String encodedPassword = encoder.encode(user.getPassword());
    user.setPassword(encodedPassword);
     user.setRole("ROLE_USER");

     // Ενημέρωση με τα νέα πεδία
    Integer degreeScore = user.getDegreeScore();
    Double income = user.getIncome();
    String city = user.getCity();
    Boolean remote = user.getRemote();


    // Έλεγχος επιλόγων για βαθμό πτυχίου
    if (degreeScore != null && (degreeScore < 1 || degreeScore > 9)) {
        return "error"; // ή δείξε μήνυμα σφάλματος στον χρήστη
    }


    repo.save(user);

    return"register_sucess";
}
```

Εικόνα 26: 1η εικόνα από τον controller

Η μέθοδος viewHopePage() με το @GetMapping("") επιστρέφει την αρχική σελίδα της εφαρμογής, εμφανίζοντας τη σελίδα "index". Αποτελεί την κεντρική πύλη εισόδου της εφαρμογής.Η μέθοδος showSignUpForm() διαχειρίζεται το αίτημα για την εμφάνιση της φόρμας εγγραφής νέων χρηστών. Μέσω του @GetMapping("/register"), δημιουργείται ένα νέο αντικείμενο User, το οποίο προστίθεται στο μοντέλο για να χρησιμοποιηθεί στη φόρμα εγγραφής. Η φόρμα αυτή εμφανίζεται στη σελίδα "signup_form".Η μέθοδος

showAdminPage() με το @GetMapping("/admin") είναι υπεύθυνη για την εμφάνιση της σελίδας διαχειριστή. Ανακτά όλα τα διαθέσιμα προγράμματα από το programRepo, τα προσθέτει στο μοντέλο και τα εμφανίζει στη σελίδα "admin_page".Η μέθοδος processRegistration() με το @PostMapping("/process_register") επεξεργάζεται τα δεδομένα εγγραφής του χρήστη. Αρχικά, ο κωδικός πρόσβασης του χρήστη κρυπτογραφείται χρησιμοποιώντας το BCryptPasswordEncoder. Ο χρήστης ορίζεται με τον προεπιλεγμένο ρόλο "ROLE_USER". Επιπλέον, ενημερώνονται τα πεδία που αφορούν τον βαθμό πτυχίου (degreeScore), το εισόδημα (income), την πόλη (city) και αν ο χρήστης προτιμά εξ αποστάσεως προγράμματα (remote). Υπάρχει επίσης ένας βασικός έλεγχος για την τιμή του βαθμού πτυχίου, και αν είναι εκτός των αποδεκτών ορίων (1-9), επιστρέφει σελίδα σφάλματος. Τέλος, ο νέος χρήστης αποθηκεύεται στο αποθετήριο χρηστών και η μέθοδος επιστρέφει τη σελίδα επιτυχίας εγγραφής "register_success".

Ο Controller περιλαμβάνει μεθόδους που χειρίζονται αιτήματα για την υποβολή αιτήσεων σε προγράμματα και την προβολή της λίστας αιτήσεων των χρηστών.

```
@GetMapping("/apply_program")
    public String showApplyProgramForm(@RequestParam("programId") Long programId, Model model) {
        Program program = programRepo.findById(programId).orElseThrow(() -> new RuntimeException("Program not found"));
        model.addAttribute("program", program);
        return "apply_program";
    }

@PostMapping("/process_application")
    public String processApplication(@RequestParam("programId") Long programId,
                                     @RequestParam("degreeScore") Integer degreeScore,
                                     @RequestParam("income") Double income,
                                     @RequestParam("city") String city,
                                     @RequestParam("remote") Boolean remote) {
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        String username = authentication.getName();
        User user = repo.findByEmail(username);

        Program program = programRepo.findById(programId).orElseThrow(() -> new RuntimeException("Program not found"));

        Application application = new Application();
        application.setUser(user);
        application.setProgram(program);
        application.setDegreeScore(degreeScore);
        application.setIncome(income);
        application.setCity(city);
        application.setRemote(remote);
        application.setApplicationDateTime(LocalDateTime.now()); // Set the current date and time

     // Εκτέλεση των κανόνων Drools για να αξιολογηθεί η αίτηση
        droolsService.evaluateApplication(application,program);



        applicationRepo.save(application);

        return application.getAccepted() ? "application_success" : "application_rejected";

    }


   @GetMapping("/applications_list")
    public String viewApplicationsPage(Model model) {
        List<Application> listApplications = applicationRepo.findAll();
        model.addAttribute("listApplications", listApplications);
        return "applications_list"; //
    }
```

Εικόνα 27: 2η εικόνα από τον Controller

Η μέθοδος showApplyProgramForm() με το @GetMapping("/apply_program") είναι υπεύθυνη για την εμφάνιση της φόρμας υποβολής αίτησης για συγκεκριμένο πρόγραμμα. Ορίζεται ένα παράμετρος programId, που προσδιορίζει το πρόγραμμα για το οποίο ο χρήστης θα υποβάλει αίτηση. Το πρόγραμμα αντλείται από το αποθετήριο προγραμμάτων (programRepo) και προστίθεται στο μοντέλο για εμφάνιση στη φόρμα "apply_program".Η μέθοδος processApplication() με το @PostMapping("/process_application") επεξεργάζεται την αίτηση του χρήστη. Αρχικά, αντλεί τον συνδεδεμένο χρήστη από το Spring Security

χρησιμοποιώντας το SecurityContextHolder και τον βρίσκει στη βάση δεδομένων μέσω του UserRepository. Στη συνέχεια, ανακτά το αντίστοιχο πρόγραμμα από το αποθετήριο προγραμμάτων (programRepo). Δημιουργείται μια νέα οντότητα Application και τα δεδομένα της αίτησης, όπως η βαθμολογία πτυχίου, το εισόδημα, η πόλη, και αν η αίτηση αφορά εξ αποστάσεως πρόγραμμα, αποθηκεύονται σε αυτήν. Η ημερομηνία και η ώρα υποβολής της αίτησης καθορίζονται στη στιγμή υποβολής με την LocalDateTime.now().Κατά την επεξεργασία της αίτησης, εκτελείται το Drools rule engine μέσω του DroolsService, το οποίο αξιολογεί αν η αίτηση γίνεται αποδεκτή ή απορρίπτεται με βάση συγκεκριμένους κανόνες. Αν η αίτηση εγκριθεί, αποθηκεύεται στο αποθετήριο αιτήσεων (applicationRepo) και επιστρέφει τη σελίδα επιτυχίας αίτησης "application_success", αλλιώς επιστρέφει τη σελίδα απόρριψης "application_rejected".

Η μέθοδος viewApplicationsPage() με το @GetMapping("/applications_list") ανακτά όλες τις αιτήσεις από το αποθετήριο αιτήσεων και τις προσθέτει στο μοντέλο, ώστε να εμφανιστούν στη σελίδα "applications_list".

```
@GetMapping("/my_applications")
public String viewMyApplications(Model model) {
    // Πάρε το Authentication αντικείμενο
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    String username = authentication.getName(); // Αυτό θα πάρει το όνομα χρήστη

    // Βρες τον χρήστη με βάση το email
    User user = repo.findByEmail(username); // Προσάρμοσε αυτή τη μέθοδο στο UserRepository

    if (user != null) {
        // Πάρε τις αιτήσεις του χρήστη
        List<Application> applications = user.getApplications();
        model.addAttribute("listApplications", applications);
    } else {
        model.addAttribute("error", "User not found");
    }

    return "my_applications"; // Επιστροφή στην αντίστοιχη σελίδα
}

@GetMapping("/delete_application/{id}")
 public String deleteApplication(@PathVariable("id") Long id, RedirectAttributes redirectAttributes) {
 // Πάρε το Authentication αντικείμενο
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    String username = authentication.getName(); // Αυτό θα πάρει το όνομα χρήστη

    // Βρες τον χρήστη με βάση το email
    User user = repo.findByEmail(username); // Προσάρμοσε αυτή τη μέθοδο στο UserRepository

    // Βρες την αίτηση με το συγκεκριμένο id
     Application applications = applicationRepo.findById(id)
         .orElseThrow(() -> new RuntimeException("Application not found"));

     // Διαγραφή της αίτησης
     applicationRepo.delete(applications);

     // Προσθήκη μηνύματος επιτυχίας στο RedirectAttributes
     redirectAttributes.addFlashAttribute("message", "Application deleted successfully");

  // Έλεγχος ρόλου χρήστη
     if (user.getRole().equals("ROLE_ADMIN")) {
         return "redirect:/applications_list"; // Ανακατεύθυνση για διαχειριστή
     } else {
         return "redirect:/my_applications"; // Ανακατεύθυνση για απλό χρήστη
     }
 }
```

Εικόνα 28: 3η εικόνα από τον Controller

Έχουμε δύο νέες λειτουργίες: την προβολή των αιτήσεων ενός χρήστη και τη δυνατότητα διαγραφής μίας αίτησης, ενώ διαφοροποιεί τη συμπεριφορά ανάλογα με τον ρόλο του χρήστη (χρήστης ή διαχειριστής).

Η μέθοδος viewMyApplications() με το @GetMapping("/my_applications") επιτρέπει σε έναν χρήστη να δει όλες τις αιτήσεις που έχει υποβάλει. Αρχικά, ανακτά το όνομα χρήστη από το σύστημα αυθεντικοποίησης (SecurityContextHolder), το οποίο αντιστοιχεί στο email του χρήστη. Στη συνέχεια, χρησιμοποιεί το UserRepository για να βρει τον χρήστη μέσω του email του. Αν ο χρήστης βρεθεί, ανακτά τις αιτήσεις του και τις προσθέτει στο μοντέλο,

ώστε να εμφανιστούν στη σελίδα "my_applications". Αν ο χρήστης δεν βρεθεί, προστίθεται στο μοντέλο μήνυμα σφάλματος.

Η μέθοδος deleteApplication() με το @GetMapping("/delete_application/{id}") επιτρέπει στον χρήστη να διαγράψει μια συγκεκριμένη αίτηση. Αρχικά, ανακτά το όνομα χρήστη και βρίσκει τον αντίστοιχο χρήστη από τη βάση δεδομένων. Στη συνέχεια, ανακτά την αίτηση από το ApplicationRepository βάσει του id που παρέχεται στη διαδρομή. Αν η αίτηση βρεθεί, διαγράφεται από τη βάση δεδομένων και προστίθεται μήνυμα επιτυχίας στα RedirectAttributes. Στο τέλος, η μέθοδος ελέγχει τον ρόλο του χρήστη. Αν ο χρήστης είναι διαχειριστής ("ROLE_ADMIN"), ανακατευθύνεται στη λίστα όλων των αιτήσεων (/applications_list), ενώ αν είναι απλός χρήστης, ανακατευθύνεται στη δική του λίστα αιτήσεων (/my_applications).

Τέλος , Ο Controller αυτός περιλαμβάνει μεθόδους για τη διαχείριση προγραμμάτων, επιτρέποντας τη διαγραφή και την επεξεργασία τους από έναν διαχειριστή.

```
@GetMapping("/delete_program/{id}")
public String deleteProgram(@PathVariable("id") Long id, RedirectAttributes redirectAttributes) {
    Program program = programRepo.findById(id)
        .orElseThrow(() -> new RuntimeException("Program not found"));

    programRepo.delete(program);

    redirectAttributes.addFlashAttribute("message", "Program deleted successfully");
    return "redirect:/admin?continue";
}

// Εμφάνιση της φόρμας επεξεργασίας
@GetMapping("/edit_program/{id}")
public String showEditProgramForm(@PathVariable("id") Long id, Model model) {
    Program program = programRepo.findById(id)
        .orElseThrow(() -> new RuntimeException("Program not found"));
    model.addAttribute("program", program);
    return "edit_program";
}

// Επεξεργασία του προγράμματος
@PostMapping("/update_program")
public String updateProgram(@ModelAttribute("program") Program program, RedirectAttributes redirectAttributes) {
    programRepo.save(program);
    redirectAttributes.addFlashAttribute("message", "Program updated successfully");
    return "redirect:/admin?continue";
}
```

Εικόνα 29:4η εικόνα από τον Controller

Η μέθοδος deleteProgram() με το @GetMapping("/delete_program/{id}") είναι υπεύθυνη για τη διαγραφή ενός προγράμματος. Ανακτά το πρόγραμμα από το programRepo βάσει του id που παρέχεται στο URL. Εάν το πρόγραμμα δεν βρεθεί, ρίχνει μια εξαίρεση. Αν το πρόγραμμα εντοπιστεί, διαγράφεται από τη βάση δεδομένων, και ένα μήνυμα επιτυχίας προστίθεται στα RedirectAttributes, ώστε να εμφανιστεί στην επόμενη σελίδα. Μετά τη διαγραφή, ο χρήστης ανακατευθύνεται στη σελίδα διαχειριστή με την προσθήκη παραμέτρου ?continue για να επιβεβαιωθεί η επιτυχής ενέργεια.

Η μέθοδος showEditProgramForm() με το @GetMapping("/edit_program/{id}") εμφανίζει τη φόρμα επεξεργασίας για ένα συγκεκριμένο πρόγραμμα. Ανάλογα με το id που παρέχεται, ανακτά το πρόγραμμα από το αποθετήριο και το προσθέτει στο μοντέλο για να μπορέσει να εμφανιστεί στη φόρμα "edit_program". Εάν το πρόγραμμα δεν βρεθεί, ρίχνει μια εξαίρεση.

Η μέθοδος updateProgram() με το @PostMapping("/update_program") επεξεργάζεται και αποθηκεύει τις αλλαγές στο πρόγραμμα. Οι αλλαγές προέρχονται από τη φόρμα επεξεργασίας και αποθηκεύονται στο programRepo. Μετά την επιτυχή αποθήκευση,

προστίθεται ένα μήνυμα επιτυχίας στα RedirectAttributes και ο χρήστης ανακατευθύνεται ξανά στη σελίδα διαχειριστή, επιβεβαιώνοντας την επιτυχία της ενημέρωσης.

3.4.4 Σχεδιασμός και υλοποίηση των κανόνων drools

Στην εφαρμογή χρησιμοποιούνται και κανόνες drools για την αυτόματη αξιολόγηση των αιτήσεων που γίνονται από τους φοιτητές.Παρακάτω θα δούμε το πώς ενσωματώθηκαν αυτοί οι κανονές στην εφαρμογή μας .

```java
package com.example;

import org.kie.api.KieServices;

@Configuration
public class DroolsConfig {

    @Bean
    public KieServices kieServices() {
        return KieServices.Factory.get();
    }

    @Bean
    public KieFileSystem kieFileSystem() {
        KieFileSystem kieFileSystem = kieServices().newKieFileSystem();
        kieFileSystem.write(ResourceFactory.newClassPathResource("rules.drl"));
        return kieFileSystem;
    }

    @Bean
    public KieContainer kieContainer() {
        final KieRepository kieRepository = kieServices().getRepository();
        kieRepository.addKieModule(() -> kieServices().getRepository().getDefaultReleaseId());
        kieServices().newKieBuilder(kieFileSystem()).buildAll();
        return kieServices().newKieContainer(kieRepository.getDefaultReleaseId());
    }

    @Bean
    public KieSession kieSession() {
        return kieContainer().newKieSession();
    }
}
```

Εικόνα 30: Κλάση DroolsConfig

Ο παραπάνω κώδικας περιγράφει την κλάση DroolsConfig, η οποία είναι υπεύθυνη για τη ρύθμιση και την αρχικοποίηση των υπηρεσιών του Drools, ενός rule engine που χρησιμοποιείται για την εκτέλεση κανόνων. Η κλάση αυτή έχει την ανατομία ενός Spring Configuration class και περιλαμβάνει διάφορους beans που χρησιμοποιούνται για την ενσωμάτωσή του Drools στην εφαρμογή. Συνολικά, η κλάση DroolsConfig ενορχηστρώνει τη ρύθμιση του Drools μέσα στην εφαρμογή, εξασφαλίζοντας ότι οι κανόνες μπορούν να φορτωθούν και να εκτελούνται κατάλληλα.

```java
package com.example;

import org.kie.api.runtime.KieSession;

@Service
public class DroolsService {

    private final KieSession kieSession;

    @Autowired
    public DroolsService(KieSession kieSession) {
        this.kieSession = kieSession;
    }

    public void evaluateApplication(Application application ,Program program) {
        kieSession.setGlobal("program", program);
        kieSession.insert(application);
        kieSession.fireAllRules();
    }
}
```

Εικόνα 31: Κλάση DroolsService

Η παραπάνω κλάση DroolsService είναι υπεύθυνη για την εκτέλεση κανόνων Drools στην εφαρμογή. Χρησιμοποιεί την KieSession, η οποία έχει ρυθμιστεί προηγουμένως, για να εκτελέσει τους κανόνες που έχουν οριστεί σχετικά με τις αιτήσεις προγραμμάτων.

Στην αρχή, η κλάση δηλώνει μια ιδιωτική μεταβλητή kieSession, η οποία είναι τύπου KieSession. Αυτή η μεταβλητή αρχικοποιείται μέσω του constructor της κλάσης, χρησιμοποιώντας την annotation @Autowired για να ενσωματωθεί η κατάλληλη KieSession που έχει ρυθμιστεί προηγουμένως από το Spring container.

Η μέθοδος evaluateApplication(Application application, Program program) αναλαμβάνει την εκτίμηση μιας συγκεκριμένης αίτησης προγράμματος. Στην αρχή της μεθόδου, ορίζεται μια παγκόσμια μεταβλητή program στην KieSession, η οποία μπορεί να χρησιμοποιηθεί στους κανόνες. Στη συνέχεια, η αίτηση (application) εισάγεται στην KieSession, επιτρέποντας στους κανόνες να έχουν πρόσβαση σε όλα τα δεδομένα της αίτησης.

Αφού εισαχθεί η αίτηση, καλείται η μέθοδος fireAllRules(), η οποία ενεργοποιεί όλους τους κανόνες που είναι ενεργοί στη συγκεκριμένη συνεδρία. Αυτή η διαδικασία θα αξιολογήσει την αίτηση βάσει των κανόνων που έχουν οριστεί στο αρχείο "rules.drl".

Συνολικά, η κλάση DroolsService λειτουργεί ως ενδιάμεσος ανάμεσα στην επιχειρηματική λογική της εφαρμογής και στη λογική των κανόνων του Drools, επιτρέποντας την ευέλικτη εκτίμηση αιτήσεων βάσει προκαθορισμένων κανόνων.

Και τέλος πάμε να δούμε το αρχείο rules.drl.

```
package com.example.rules

import com.example.Application
import com.example.Program

global Program program;

rule "Accept application if criteria met considering remote option"
when
    $application : Application(
        (program.getRemote() == true &&
         remote == true &&
         degreeScore >= program.getDegreeScore() &&
         income >= program.getRequiredIncome()) ||
        (program.getRemote() == true &&
         remote == false &&
         degreeScore >= program.getDegreeScore() &&
         income >= program.getRequiredIncome() &&
         city == program.getCity) ||
        (program.getRemote() == false &&
         remote == false &&
         degreeScore >= program.getDegreeScore() &&
         income >= program.getRequiredIncome() &&
         city == program.getCity)
    )
then
    $application.setAccepted(true);
end

rule "Reject application if criteria not met or remote mismatch or wrong city"
when
    $application : Application(
        (program.getRemote() == true &&
         remote == false &&
         city != program.getCity) ||
        (degreeScore < program.getDegreeScore() ||
         income < program.getRequiredIncome()) ||
        (program.getRemote() == false &&
         (remote == true ||
          city != program.getCity))
    )
then
    $application.setAccepted(false);
end
```

Εικόνα 32: Αρχείο rules

Ο παραπάνω κώδικας περιγράφει δύο κανόνες για την εκτίμηση αιτήσεων προγραμμάτων, γραμμένους σε γλώσσα κανόνων Drools. Οι κανόνες αυτοί ελέγχουν αν μια αίτηση θα γίνει αποδεκτή ή θα απορριφθεί, βασιζόμενοι σε διάφορα κριτήρια.

Κανόνας 1: Accept application if criteria met considering remote option

Ο πρώτος κανόνας ελέγχει αν μια αίτηση μπορεί να γίνει αποδεκτή βάσει των κριτηρίων που σχετίζονται με τη δυνατότητα απομακρυσμένης εκπαίδευσης. Ο κανόνας ενεργοποιείται εάν πληρούνται οι εξής προϋποθέσεις:

- **Remote Program and Remote Application**: Εάν το πρόγραμμα προσφέρει απομακρυσμένη εκπαίδευση και η αίτηση είναι για απομακρυσμένο πρόγραμμα, τότε ελέγχεται αν ο βαθμός πτυχίου (degreeScore) της αίτησης είναι μεγαλύτερος ή ίσος με τον απαιτούμενο βαθμό του προγράμματος και αν το εισόδημα είναι μεγαλύτερο ή ίσο με το απαιτούμενο εισόδημα.
- **Remote Program and In-Person Application**: Εάν το πρόγραμμα προσφέρει απομακρυσμένη εκπαίδευση αλλά η αίτηση δεν είναι για απομακρυσμένο πρόγραμμα, ελέγχεται επίσης αν ο βαθμός πτυχίου και το εισόδημα πληρούν τις απαιτήσεις, καθώς και αν η πόλη του υποψήφιου είναι η ίδια με την πόλη του προγράμματος.
- **Non-Remote Program**: Εάν το πρόγραμμα δεν προσφέρει απομακρυσμένη εκπαίδευση, η αίτηση θα γίνει αποδεκτή αν πληρούνται οι απαιτήσεις του βαθμού πτυχίου και του εισοδήματος και η πόλη είναι η ίδια.

Εάν κάποια από αυτές τις συνθήκες πληρείται, η αίτηση γίνεται αποδεκτή και το πεδίο accepted της αίτησης ορίζεται σε true.

Κανόνας 2: Reject application if criteria not met or remote mismatch or wrong city

Ο δεύτερος κανόνας ελέγχει αν η αίτηση θα πρέπει να απορριφθεί. Οι προϋποθέσεις που οδηγούν στην απόρριψη της αίτησης είναι:

- **Remote Mismatch**: Εάν το πρόγραμμα προσφέρει απομακρυσμένη εκπαίδευση αλλά η αίτηση δεν είναι για απομακρυσμένο πρόγραμμα και η πόλη δεν αντιστοιχεί, η αίτηση απορρίπτεται.
- **Score or Income Criteria Not Met**: Εάν ο βαθμός πτυχίου είναι χαμηλότερος από τον απαιτούμενο ή το εισόδημα είναι χαμηλότερο από το απαιτούμενο, η αίτηση απορρίπτεται.
- **Non-Remote Program with Remote Application**: Εάν το πρόγραμμα δεν προσφέρει απομακρυσμένη εκπαίδευση αλλά η αίτηση ζητά απομακρυσμένη εκπαίδευση ή η πόλη δεν αντιστοιχεί, η αίτηση απορρίπτεται.

Εάν κάποια από αυτές τις συνθήκες πληρείται, η αίτηση απορρίπτεται και το πεδίο accepted ορίζεται σε false.

Οι κανόνες αυτοί επιτρέπουν την αυτόματη εκτίμηση των αιτήσεων βάσει προκαθορισμένων κριτηρίων, διευκολύνοντας τη διαδικασία αξιολόγησης και εξασφαλίζοντας ότι οι αποφάσεις βασίζονται σε αντικειμενικά δεδομένα.

Κεφάλαιο 4ο

4 Μελλοντικές Επεκτάσεις

Η διαδικτυακή εφαρμογή για τη διαχείριση αιτήσεων μεταπτυχιακών προγραμμάτων έχει τη δυνατότητα να επεκταθεί σε πολλές κατευθύνσεις, προκειμένου να καλύψει τις μεταβαλλόμενες ανάγκες των χρηστών και των εκπαιδευτικών ιδρυμάτων. Μία από τις σημαντικότερες επεκτάσεις θα μπορούσε να είναι η ενσωμάτωση ενός συστήματος παρακολούθησης και ανάλυσης των επιδόσεων των φοιτητών, επιτρέποντας στους διαχειριστές των προγραμμάτων να παρακολουθούν την πρόοδο των μαθητών και να παρέχουν εξατομικευμένες προτάσεις για επιπλέον μαθήματα ή υποστήριξη. Επίσης, η εφαρμογή θα μπορούσε να επεκταθεί ώστε να υποστηρίζει διεθνείς αιτήσεις, διευκολύνοντας φοιτητές από διάφορες χώρες να υποβάλουν αιτήσεις και να αποκτούν πρόσβαση σε πληροφορίες σχετικά με τις απαιτήσεις εισαγωγής και τα διαθέσιμα προγράμματα σε πραγματικό χρόνο. Ακόμη, η ενσωμάτωση τεχνολογιών τεχνητής νοημοσύνης και μηχανικής μάθησης θα μπορούσε να επιτρέψει την προσαρμογή των διαδικασιών αξιολόγησης, παρέχοντας προγνωστικές αναλύσεις για την πιθανότητα αποδοχής των αιτήσεων, βασισμένες σε ιστορικά δεδομένα. Η δημιουργία ενός forum ή μιας κοινότητας μέσα στην εφαρμογή θα μπορούσε να διευκολύνει τη συνεργασία και την αλληλεπίδραση μεταξύ των φοιτητών, προσφέροντας τους τη δυνατότητα να μοιράζονται εμπειρίες και συμβουλές σχετικά με τις διαδικασίες αίτησης και τα προγράμματα σπουδών. Τέλος, η ανάπτυξη ενός mobile app θα μπορούσε να διευκολύνει τους φοιτητές να διαχειρίζονται τις αιτήσεις τους από το κινητό τους τηλέφωνο, προσφέροντας μια πιο βολική και φιλική προς το χρήστη εμπειρία. Αξιοποιώντας τις δυνατότητες των APIs, η εφαρμογή θα μπορούσε να ενσωματωθεί με άλλα εκπαιδευτικά εργαλεία και πλατφόρμες, επιτρέποντας στους φοιτητές να έχουν πρόσβαση σε εκπαιδευτικό υλικό, διαδικτυακά μαθήματα ή ακόμα και ευκαιρίες πρακτικής άσκησης. Μέσω αυτών των επεκτάσεων, η εφαρμογή μπορεί να ενισχύσει τη λειτουργικότητά της και να προσφέρει μεγαλύτερη αξία στους χρήστες της.

Κεφάλαιο 5ο

5 Βιβλιογραφικές Πηγές

- Java documentation : https://docs.oracle.com/en/java/
- Springboot documentation : https://docs.spring.io/spring-boot/index.html
- Drools documentation : https://docs.drools.org/7.6.0.Final/drools-docs/html_single/
- HTML documentation : https://developer.mozilla.org/en-US/docs/Web/HTML
- Javascript documentation: https://devdocs.io/javascript/
- Css documentation: https://devdocs.io/css/
- MySql Workbench documentation : https://dev.mysql.com/doc/workbench/en/
- Git: https://github.com/
- Stackoverflow : https://stackoverflow.com/
- Uml Documentation: https://tallyfy.com/uml-diagram/
- Swimlane diagram : https://miro.com/diagramming/what-is-a-swimlane-diagram/
- Draw io : https://app.diagrams.net/