Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Εκμάθηση γλωσσών προγραμματισμού μέσω μιας διαδικτυακής πλατφόρμας που ενσωματώνει ευφυείς τεχνικές πρόβλεψης των βαθμών των μαθητών.** **Learning computer programming through a web-based platform that incorporates intelligent techniques to predict students' grades**
Ονοματεπώνυμο Φοιτητή	**Βασιλική Νικητοπούλου**
Πατρώνυμο	**Ιωάννης**
Αριθμός Μητρώου	**ΜΠΠΛ/ 14061**
Επιβλέπων	**Μαρία Βίρβου, Καθηγήτρια**

Ημερομηνία Παράδοσης **Οκτώβριος 2019**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Βίρβου Μαρία Καθηγήτρια	Τσιχριντζής Γεώργιος Καθηγητής	Αλέπης Ευθύμιος Επίκουρος Καθηγητής

Περίληψη

Στην παρούσα διπλωματική εργασία χρησιμοποιούμε τη μηχανιστική μάθηση και συγκεκριμένα τον αλγόριθμο K-means με σκοπό να μπορέσουμε να βοηθήσουμε στη βελτίωση της επίδοσης των μαθητών στο τελικό διαγώνισμα μαθημάτων που διδάσκονται. Για το σκοπό αυτό, δημιουργήσαμε μια web εφαρμογή η οποία υλοποιήθηκε με χρήση Java, Hibernate, Postgresql και JSPs, η οποία λειτουργεί υπό το πρίσμα τριών διαφορετικών ρόλων. Αρχικά έχει τον διαχειριστή, ο οποίος αναλαμβάνει τη διαχείριση της εφαρμογής και των χρηστών της. Επίσης εξυπηρετεί το ρόλο του καθηγητή, ο οποίος εισάγει το υλικό για τα διάφορα μαθήματα. Τέλος, υπάρχει προφανώς ο ρόλος τους μαθητή ο οποίος δύναται να διδαχθεί τα μαθήματα και μέσα από μία σειρά από διαγωνίσματα να βελτιώσει την επίδοση του. Η δυνατότητα βελτίωσης δίνεται μέσα από μοντέλα πρόβλεψης της τελικής επίδοσης του μαθητή τα οποία, όπως αναφέραμε, στηρίζονται στον αλγόριθμό K-means.

Abstract

In this thesis we use machine learning, and in particular the K-means algorithm, in order to help improve students' performance in the final exams of the classes they are taught. To this end, we have created a web application using Java, Hibernate, Postgresql and JSPs, which includes three different roles. First, it has the administrator, who controls the application and its users. It also serves the role of the teacher, who introduces the material for the various lessons. Finally, there is the role of the student who can attend the lessons and through a series of tests manages to improve his/her performance. Improvement is provided through models of predicting the student's performance in the final tests which, as mentioned, are based on the K-means algorithm.

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε στο πλαίσιο ολοκλήρωσης του μεταπτυχιακού πληροφορικής του Πανεπιστημίου Πειραιώς κατά το έτος 2019. Αρχικά να ευχαριστήσω την καθηγήτριά μου, κα. Μαρία Βίρβου για την τιμή που μου έκανε να με αναλάβει και για την δυνατότητα που μου έδωσε τη δυνατότητα να μπορέσω να πραγματοποιήσω τη συγκεκριμένη πτυχιακή εργασία. Επίσης, θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντά μου κ. Σπύρο Παπαδημητρίου για τις πολύτιμες συμβουλές του, την τεχνογνωσία και την υποστήριξή του καθόλη την διάρκεια εκπόνησης της διπλωματικής μου εργασίας. Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου, οι οποίοι με στήριξαν όλα αυτά τα χρόνια και τον και τον σύζυγό μου που στέκεται πάντα δίπλα μου αρωγός μου σε κάθε μου προσπάθεια.

Table of Contents

1. Εισαγωγή

1.1. Περίληψη

Ο στόχος της παρούσας διπλωματικής εργασίας έχει ως κύριο γνώμονα την εκπαίδευση. Πιο συγκεκριμένα, τη χρήση των εργαλείων της μηχανιστικής μάθησης για τη διεξαγωγή χρήσιμων συμπερασμάτων που θα βελτιώσουν τις επιδόσεις των μαθητών προγραμματισμού. Δημιουργήσαμε μία ολοκληρωμένη e-learning πλατφόρμα εκμάθησης γλωσσών προγραμματισμού, στην οποία μπορεί ένας μαθητής μέσω αναζήτησης να βρίσκει όποιο μάθημα τον ενδιαφέρει και να το παρακολουθεί. Επίσης, προσφέρει και δυνατότητες administrator ώστε να προστίθενται καινούργια μαθήματα και η πλατφόρμα συνεχώς να εξελίσσεται. Σε αυτό το σημείο αξίζει να σημειωθεί, ότι η πλατφόρμα δεν είναι απλώς μία ακόμα εκπαιδευτική πλατφόρμα. Το κρίσιμο σημείο εν προκειμένω είναι ότι χρησιμοποιεί τον αλγόριθμο μηχανιστικής μάθησης K-Means με τον οποίο, δίνει την δυνατότητα στο μαθητή μέσα από αλλεπάλληλα τεστ να βελτιώσει την επίδοσή του. Δηλαδή, κάθε φορά που ολοκληρώνει το τεστ κάθε κεφαλαίου ο αλγόριθμος αποθηκεύει τις απαντήσεις (σωστό ή λάθος) του μαθητή, το χρόνο απάντησης καθώς και το αν ο μαθητής απάντησε περισσότερες από μία φορές την ίδια απάντηση. Ο μαθητής βλέπει το σκορ του στο συγκεκριμένο τεστ καθώς επίσης και πρόβλεψη της επίδοσης του στο τελικό τεστ του μαθήματος.

1.2. Κίνητρο και σκοπός για τη διεξαγωγή της εργασίας

Η εργασία αυτή εκπονήθηκε με σκοπό την αξιοποίηση αλγορίθμων μηχανιστικής μάθησης στη διαδικασία της πραγματικής μάθησης. Σκοπός είναι να βοηθάει τους μαθητές να μπορέσουν να βελτιώσουν στο μέγιστο τις επιδόσεις τους στα διαγωνίσματα και μέσα από αυτή τη διαδικασία να μπορέσουν να βελτιώσουν το πραγματικό επίπεδο γνώσεων τους. Βέβαια για να λειτουργήσει πλήρως αυτό, θα πρέπει να έχουμε ένα ολόκληρο περιβάλλον στο οποίο να μπορέσουμε να εισάγουμε τον χρήστη και να τον εκπαιδεύσουμε ώστε να φτάσει στο τέλος το επιθυμητό. Αυτό προσπαθήσαμε να πετύχουμε στην παρούσα διπλωματική εργασία.

1.3. Δομή της εργασίας

Η εργασία αυτή απαρτίζεται από 6 ξεχωριστά κεφάλαια. Το πρώτο είναι το εισαγωγικό κεφάλαιο που δίνει μια αφαιρετική άποψη όσον αφορά τους στόχους της εργασίας αυτής. Στο δεύτερο κεφάλαιο αναφέρονται εκτενώς όλα τα εργαλεία που χρησιμοποιήθηκαν για εκπόνηση της εργασίας. Το τρίτο κεφάλαιο αναφέρεται στις απαιτήσεις του συστήματος και στην ανάλυση της δομής του προγραμματιστικού μοντέλου. Στο τέταρτο κεφάλαιο παραθέτουμε τα σημαντικότερα κομμάτια κώδικα, καθώς και περιγραφή του. Στο πέμπτο κεφάλαιο αναφερόμαστε στην υλοποίηση του περιβάλλοντος από την πλευρά του τελικού χρήστη, ενώ στο έκτο κεφάλαιο περιλαμβάνουμε τα αποτελέσματα και κάποιες ιδέες για μελλοντικές επεκτάσεις.

2. Μεθοδολογία υλοποίησης

2.1. Μέθοδος ανάλυσης και ανάπτυξης της εφαρμογής

Η παρούσα εφαρμογή αναπτύχθηκε σε JAVA Spring Framework και στο IDE του Eclipse, χρησιμοποιώντας τεχνολογίες Hibernate για σύνδεση με τη βάση, JSP για παρουσίαση στο χρήστη, και PostrgeSQL για αποθήκευση των δεδομένων σε βάση. Η αρχιτεκτονική που ακολουθήθηκε είναι η αρχιτεκτονική του MVC. Όλα τα παραπάνω αναλύονται εκτενώς στη συνέχεια.

2.2. Η γλώσσα προγραμματισμού JAVA

Στις αρχές του 1991, η Sun αναζητούσε το κατάλληλο εργαλείο για να αποτελέσει την πλατφόρμα ανάπτυξης λογισμικού σε μικρο-συσκευές (έξυπνες οικιακές συσκευές έως πολύπλοκα συστήματα παραγωγής γραφικών). Τα εργαλεία της εποχής ήταν γλώσσες όπως η C++ και η C. Μετά από διάφορους πειραματισμούς προέκυψε το συμπέρασμα ότι οι υπάρχουσες γλώσσες δεν μπορούσαν να καλύψουν τις ανάγκες τους. Ο "πατέρας" της Java, James Gosling, που εργαζόταν εκείνη την εποχή για την Sun, έκανε ήδη πειραματισμούς πάνω στη C++ και είχε παρουσιάσει κατά καιρούς κάποιες πειραματικές γλώσσες (C++ ++,που μετέπειτα ονομάστηκε C#) ως πρότυπα για το νέο εργαλείο που αναζητούσαν στην Sun. Τελικά μετά από λίγο καιρό κατέληξαν με μια πρόταση για το επιτελείο της εταιρίας, η οποία ήταν η γλώσσα Oak. Το όνομά της το πήρε από το ομώνυμο δένδρο (βελανιδιά) το οποίο ο Gosling είχε έξω από το γραφείο του και έβλεπε κάθε μέρα.

Ένα από τα βασικά πλεονεκτήματα της Java έναντι των περισσότερων άλλων γλωσσών είναι η ανεξαρτησία του λειτουργικού συστήματος και πλατφόρμας. Τα προγράμματα που είναι γραμμένα σε Java τρέχουνε ακριβώς το ίδιο σε Windows, Linux, Unix και Macintosh (σύντομα θα τρέχουν και σε Playstation καθώς και σε άλλες κονσόλες παιχνιδιών) χωρίς να χρειαστεί να ξαναγίνει μεταγλώττιση (compiling) ή να αλλάξει ο πηγαίος κώδικας για κάθε διαφορετικό λειτουργικό σύστημα. Για να επιτευχθεί όμως αυτό χρειαζόταν κάποιος τρόπος έτσι ώστε τα προγράμματα γραμμένα σε Java να μπορούν να είναι «κατανοητά» από κάθε υπολογιστή ανεξάρτητα του είδους επεξεργαστή (Intel x86, IBM, Sun SPARC, Motorola) αλλά και λειτουργικού συστήματος (Windows, Unix, Linux, BSD, MacOS). Ο λόγος είναι ότι κάθε κεντρική μονάδα επεξεργασίας κατανοεί διαφορετικό κώδικα μηχανής. Ο συμβολικός κώδικας (assembly) που μεταφράζεται και εκτελείται σε Windows είναι διαφορετικός από αυτόν που μεταφράζεται και εκτελείται σε έναν υπολογιστή Macintosh. Η λύση δόθηκε με την ανάπτυξη της Εικονικής Μηχανής (Virtual Machine ή VM ή ΕΜ στα ελληνικά)

Αφού γραφεί κάποιο πρόγραμμα σε Java, στη συνέχεια μεταγλωτίζεται μέσω του μεταγλωττιστή javac, ο οποίος παράγει έναν αριθμό από αρχεία .class (κώδικας byte ή bytecode). Ο κώδικας byte είναι η μορφή που παίρνει ο πηγαίος κώδικας της Java όταν μεταγλωττιστεί. Όταν πρόκειται να εκτελεστεί η εφαρμογή σε ένα μηχάνημα, το Java Virtual Machine που πρέπει να είναι εγκατεστημένο σε αυτό θα αναλάβει να διαβάσει τα αρχεία .class. Στη συνέχεια τα μεταφράζει σε γλώσσα μηχανής που να υποστηρίζεται από το λειτουργικό σύστημα και τον επεξεργαστή, έτσι ώστε να εκτελεστεί. Αυτό συμβαίνει με την παραδοσιακή Εικονική Μηχανή (Virtual Machine). Πιο σύγχρονες εφαρμογές της εικονικής Μηχανής μπορούν και μεταγλωττίζουν εκ των προτέρων τμήματα bytecode απευθείας σε κώδικα μηχανής (εγγενή κώδικα ή native code) με αποτέλεσμα να βελτιώνεται η ταχύτητα. Χωρίς αυτό δε θα ήταν δυνατή η εκτέλεση λογισμικού

γραμμένου σε Java. Η JVM είναι λογισμικό που εξαρτάται από την πλατφόρμα, δηλαδή για κάθε είδος λειτουργικού συστήματος και αρχιτεκτονικής επεξεργαστή υπάρχει διαφορετική έκδοση του. Έτσι υπάρχουν διαφορετικές JVM για Windows, Linux, Unix, Macintosh, κινητά τηλέφωνα, παιχνιδομηχανές κλπ.

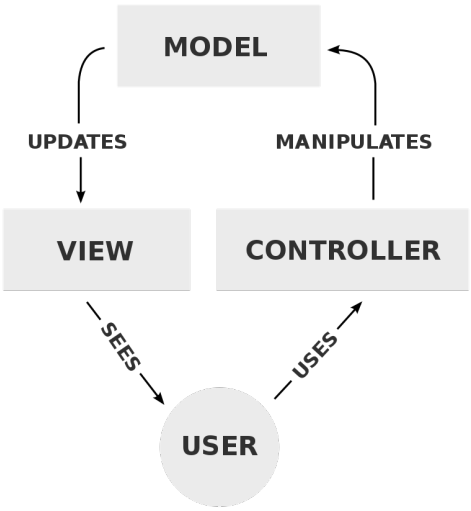
Οτιδήποτε θέλει να κάνει ο προγραμματιστής (ή ο χρήστης) γίνεται μέσω της εικονικής μηχανής. Αυτό βοηθάει στο να υπάρχει μεγαλύτερη ασφάλεια στο σύστημα γιατί η εικονική μηχανή είναι υπεύθυνη για την επικοινωνία χρήστη - υπολογιστή. Ο προγραμματιστής δεν μπορεί να γράψει κώδικα ο οποίος θα έχει καταστροφικά αποτελέσματα για τον υπολογιστή γιατί η εικονική μηχανή θα τον ανιχνεύσει και δε θα επιτρέψει να εκτελεστεί. Από την άλλη μεριά ούτε ο χρήστης μπορεί να κατεβάσει «κακό» κώδικα από το δίκτυο και να τον εκτελέσει. Αυτό είναι ιδιαίτερα χρήσιμο για μεγάλα κατανεμημένα συστήματα όπου πολλοί χρήστες χρησιμοποιούν το ίδιο πρόγραμμα συγχρόνως.

2.3. Η αρχιτεκτονική MVC

Το Model–view–controller (σε συντομογραφία αναφέρεται ως MVC) είναι ένα μοντέλο αρχιτεκτονικής λογισμικού το οποίο χρησιμοποιείται για την δημιουργία περιβαλλόντων αλληλεπίδρασης χρήστη. Στο μοντέλο αυτό η εφαρμογή διαιρείται σε τρία διασυνδεδεμένα μέρη ώστε να διαχωριστεί η παρουσίαση της πληροφορίας στον χρήση από την μορφή που έχει αποθηκευτεί στο σύστημα. Το κύριο μέρος του μοντέλου είναι το αντικείμενο Model το οποίο διαχειρίζεται την ανάκτηση/αποθήκευση των δεδομένων στο σύστημα. Το αντικείμενο View χρησιμοποιείται μόνο για να παρουσιάζεται η πληροφορία στον χρήστη (π.χ. με γραφικό τρόπο). Το τρίτο μέρος είναι ο Controller ο οποίος δέχεται την είσοδο και στέλνει εντολές στο αντικείμενο Model και στο View.

Εκτός από το να διαιρείται η εφαρμογή σε τρία μοντέλα, η σχεδίαση model–view–controller ορίζει και τις αλληλεπιδράσεις των μοντέλων

- Ο **controller** μπορεί να στέλνει εντολές στο μοντέλο και να ενημερώνει την κατάσταση του μοντέλου. Μπορεί επίσης να στέλνει εντολές ώστε να γίνει η αντίστοιχη αναπαράσταση των δεδομένων του μοντέλου μέσω του *View*.
- Το **model** ενημερώνει τις αντίστοιχες αναπαραστάσεις *views* και τους *controllers* όταν υπάρχει αλλαγή στα δεδομένα. Αυτή η ενημέρωση επιτρέπει στα *views* να ενημερώνουν την γραφική απεικόνιση.
- Το **view** αναπαριστά με γραφικό τρόπο την πληροφορία που περιέχει το *model* δημιουργώντας γραφική παρουσίαση στο χρήστη.



Εικόνα 1: Το μοντέλο υλοποίησης MVC

2.4. Spring framework

Το Spring Framework είναι λογισμικό ανοιχτού κώδικα (ελεύθερο λογισμικό) που σκοπό έχει να διευκολύνει την ανάπτυξη J2EE λογισμικού σε μεγάλη έκταση και βασίζεται στη γλώσσα προγραμματισμού Java.

Το Spring, προϊόν της εταιρείας Springsource, είναι ένα ελαφρύ Java/J2EE πλαίσιο εφαρμογών που βασίζεται σε κώδικα που δημοσιεύθηκε στο βιβλίο “Expert One-on-One J2EE Design and Development” από τον βασικό συντελεστή του framework, Rod Johnson. Το Spring ανάμεσα στα άλλα έχει τα εξής βασικά χαρακτηριστικά:

- Αποτελεί μία δυνατή λύση για τη διαχείριση ρυθμίσεων της εφαρμογής, που βασίζεται σε JavaBeans εφαρμόζοντας την αρχιτεκτονική αρχή (κατά άλλους design pattern) του Inversion of Control,
- Ένα γενικό abstraction layer για τη διαχείριση transactions που επίσης επιτρέπει pluggable transactions managers,
- Ένα JDBC abstraction layer,
- Ενσωμμάτωση με ένα πλήθος διαδεδομένων τεχνολογιών όπως το Hibernate, JDO, Apache OJB,
- Λειτουργικότητα Aspect Oriented Programming (AOP) προγραμματιστικό υπόδειγμα στο οποίο οι δευτερεύουσες λειτουργίες διαχωρίζονται από το business logic της εφαρμογής.
- Το καλύτερο για μένα είναι ότι έρχεται πακεταρισμένο με το δικό του, flexible MVC Web framework με πολλαπλές τεχνολογίες στο View κομμάτι.
- Διαδικασίες για authentication / authorisation που υποστηρίζουν διάφορα πρωτόκολλα (πχ. LDAP).

- Ένα Remote Access framework με λειτουργίες Remote Procedure Call (RPC) για εισαγωγή / εξαγωγή Java objects και το οποίο υποστηρίζει RMI, CORBA, SOAP και άλλα.

Το Spring Framework αποτελείται από έναν Container βασισμένο στο Inversion of Control (IoC) ή Dependency Injection κατά άλλους. Πρόκειται για μία τεχνική όπου υποδεικνυέται σε ένα κομμάτι εφαρμογής ποια αλλα κομμάτια μπορεί να χρησιμοποιεί. Για όλες τις εφαρμογές που χτίζονται πάνω στο Spring, ο Container αποτελεί την καρδιά του συστήματος και όλα τα τα Java Beans που περιέχει γίνονται instantiated, παραμετροποιούνται και συναρμολογούνται από τον Container.

Το interface org.springframework.context.ApplicationContext αντιπροσωπεύει τον Spring IoC container και υπάρχουν παρά πολλα implementations του ανάλογα με τον τύπο εφαρμογής (stand-alone, web κτλ.)

2.5. Hibernate

Το Hibernate Framework είναι λογισμικό ανοιχτού κώδικα (ελεύθερο λογισμικό) που σκοπό έχει να συνδέσει τα αντικείμενα που δημιουργούνται σε μια αντικειμενοστρεφή γλώσσα προγραμματισμού (Java) με τους πίνακες μιας σχεσιακής βάσης δεδομένων. Η σύνδεση αυτή επιτυγχάνεται με την χρήση επιπρόσθετης πληροφορίας (metadata) που τοποθετείται κατάλληλα (μαζί με τον κώδικα Java ή σε ξεχωριστά xml αρχεία) και περιγράφει την αντιστοιχία μεταξύ των αντικειμένων και της βάσης δεδομένων. Γενικά το Hibernate προσφέρει την αυτόματη μετατροπή της μιας μορφής (αντικείμενα) στην άλλη (σχεσιακή βάση δεδομένων).

Το Hibernate συγκαταλέγεται στην κατηγορία του λογισμικού ORM (object/relational mapping). Το λογισμικό ORM στοχεύει στη δημιουργία μιας διεπαφής (interface) μεταξύ των διαδεδομένων σχεσιακών βάσεων δεδομένων και του αντικειμενοστραφούς προγραμματισμού. Με απλά λόγια, προσφέρει τη χρησιμοποίηση μιας σχεσιακής βάσης δεδομένων ως να ήταν αντικειμενοστρεφής. Για να το επιτύχει αυτό δημιουργεί αντιστοιχίες μεταξύ των εννοιών του αντικειμενοστρεφούς προγραμματισμού (συσχετίσεις, κληρονομικότητα, πολυμορφισμός) - που δεν υπάρχουν σε μια σχεσιακή βάση δεδομένων - και των πινάκων και σχέσεων μεταξύ των πινάκων μιας σχεσιακής βάσης. Με αυτό τον τρόπο ο προγραμματιστής βλέπει τελικά μια αντικειμενοστρεφή βάση δεδομένων, παρ΄όλο που στην ουσία χρησιμοποιεί μια σχεσιακή. Έτσι ο προγραμματιστής χρησιμοποιεί τα αντικείμενα της συγκεκριμένης εφαρμογής, τα τροποποιεί σχετικά με τη λογική της εφαρμογής που αναπτύσσει και τα αποθηκεύει (τροποποιεί, διαγράφει και αναζητά) στη βάση ως αντικείμενα, σκεπτόμενος δηλαδή με αντικειμενοστρεφείς έννοιες και όχι με βάση το σχήμα της σχεσιακής βάσης δεδομένων. Σε αυτό το σημείο είναι το Hibernate που, γνωρίζοντας την αντιστοιχία μεταξύ βάσης και λογικής της εφαρμογής, αναλαμβάνει να κατασκευάσει την κατάλληλη εντολή της SQL, η οποία και στέλνεται τελικά στη βάση δεδομένων. Έπειτα, τα αποτελέσματα που επιστρέφει η βάση, το Hibernate τα επιστρέφει στον προγραμματιστή ως αντικείμενα της εφαρμογής. Είναι δηλαδή ένα ενδιάμεσο επίπεδο μεταξύ της εφαρμογής και της βάσης δεδομένων.

Το Hibernate προσφέρει τα παρακάτω στον προγραμματιστή:

- Παραγωγικότητα: Στην ανάπτυξη λογισμικού ένα μεγάλο μέρος της προγραμματιστικής προσπάθειας αφιερώνεται στη διεπαφή της εφαρμογής με τη βάση δεδομένων. Το Hibernate αυτοματοποιώντας τις βασικές λειτουργίες Δημιουργία/Ανάγνωση/Τροποποίηση/Διαγραφή (CRUD – Create Read Update Delete) επιτρέπει αρχικά στον προγραμματιστή να επικεντρώνει την προσπάθειά του στη λογική της εφαρμογής (business logic). Επίσης, υπάρχει η δυνατότητα να ακολουθηθούν δύο στρατηγικές ανάπτυξης λογισμικού: είτε αρχίζοντας από το

μοντέλο δεδομένων είτε από τη βάση δεδομένων. Αυτό μειώνει σε μεγάλο βαθμό το χρόνο ανάπτυξης.

- Συντηρησιμότητα: Με τη χρήση του Hibernate γράφονται σημαντικά λιγότερες γραμμές κώδικα και ο κώδικας είναι πιο κατανοητός και καλογραμμένος. Αυτό κάνει την συντήρηση της εφαρμογής ευκολότερη.
- Ανεξαρτησία από τη βάση δεδομένων: Με τη συμβατότητα του Hibernate με διαφορετικές βάσεις δεδομένων και τη δυνατότητα σύνδεσής του με τη βάση μέσω δηλώσεων οριζομένων σε ειδικό αρχείο η αναπτυσσόμενη εφαρμογή μπορεί με ελάχιστες τροποποιήσεις να χρησιμοποιηθεί με βάσεις δεδομένων διαφορετικών κατασκευαστών. Το γεγονός αυτό στερεί μεν από το Hibernate την εκμετάλλευση των ιδιαίτερων χαρακτηριστικών της χρησιμοποιούμενης βάσης, όμως, και σε αυτή την περίπτωση, δίνεται η δυνατότητα χρήσης πηγαίας SQL μέσα στο Hibernate που εκμεταλλεύεται τα ιδιαίτερα αυτά χαρακτηριστικά. Αυτό βέβαια μειώνει την ανεξαρτησία του Hibernate.

2.6. Postgresql

Η PostgreSQL είναι μια σχεσιακή βάση δεδομένων ανοικτού κώδικα με πολλές δυνατότητες. Η ανάπτυξη της διαρκεί ήδη πάνω από δύο δεκαετίες και βασίζεται σε μια αποδεδειγμένα καλή αρχιτεκτονική η οποία έχει δημιουργήσει μια ισχυρή αντίληψη των χρηστών της γύρω από την αξιοπιστία, την ακεραιότητα δεδομένων και την ορθή λειτουργία.

Η PostgreSQL τρέχει σε όλα τα βασικά λειτουργικά συστήματα, στα οποία περιλαμβάνονται το Linux, το UNIX (AIX, BSD, HP-UX, SGI, IRIX, MAC OS X, Solaris, Tru64) και τα Windows. Είναι συμβατή με ACID, και συμπεριλαμβάνει τους περισσότερους SQL92 και SQL99 τύπους δεδομένων συμπεριλαμβανομένων INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL και TIMESTAMP. Επίσης υποστηρίζει αποθήκευση μεγάλων δυαδικών αντικειμένων (binary), όπως εικόνες, ήχοι ή βίντεο. Διαθέτει επίσης περιβάλλοντα προγραμματισμού για τις γλώσσες προγραμματισμού C, C++, Java, Perl, Python, Ruby, Tcl, και υποστήριξη για την πλατφόρμα .NET και το πρότυπο ODBC, ενώ περιλαμβάνει και εξαιρετικό εγχειρίδιο χρήσης.

2.7. JSP

Με βάση μια απλουστευμένη ερμηνεία, τα Servlets είναι προγράμματα Java με ενσωματωμένο κώδικα HyperText Markup Language (HTML). Υπό αυτή την έννοια, οι JSP είναι ιστοσελίδες HTML με ενσωματωμένο κώδικα Java. Ο κώδικας των Servlets είναι δομημένος σε κλάσεις, που γράφονται, μεταγλωττίζονται και εκτελούνται όπως κάθε κλάση Java, ενώ οι JSP αποτελούνται από στατικό κώδικα HTML και δυναμικά παραγόμενο περιεχόμενο HTML, το οποίο είναι διαχωρισμένο με ειδικές ετικέτες (tags), της μορφής <%java κώδικας %>.

Τα Servlets και οι JSP θεωρούνται από πολλούς ταυτόσημα, επειδή οι JSP μεταφράζονται σε Servlets, δηλαδή βασίζονται στην τεχνολογία των Java Servlets. Τα Servlets είναι κατάλληλα για εργασίες προσανατολισμένες στην λογική και στην επεξεργασία, ενώ οι JSP είναι πιο κατάλληλες για εργασίες προσανατολισμένες στην παρουσίαση. Το κυριότερο μειονέκτημα της χρήσης των Servlets είναι ότι ο προγραμματιστής θα πρέπει να γράψει αρκετές εντολές «out.println» (εντολές εμφάνισης στην οθόνη), για να σχηματίσει την απόκριση (response) του Servlet στο χρήστη. Αυτό είναι επίπονο για τον προγραμματιστή, ιδίως όταν η απόκριση προκύπτει από τη χρήση στοιχείων HTML, τα οποία είναι πολύπλοκα. Άρα, η απόκριση είναι πιο εύκολο να γράφεται απευθείας σε

HTML και να χρησιμοποιείται κώδικας Java όπου χρειάζεται. Αυτήν ακριβώς την ανάγκη καλύπτουν οι JSP. Η τεχνολογία JSP μπορεί να διαχωρίζει τη διεπαφή (interface) του χρήστη από τη δημιουργία περιεχομένου, επιτρέποντας έτσι στους σχεδιαστές να αλλάζουν τη συνολική εμφάνιση της σελίδας (look and feel) χωρίς να μεταβάλλεται το βασικό δυναμικό περιεχόμενό της.

Τα αρχεία JSP είναι αποθηκευμένα σε έναν JSP (Servlet) Container. Ο ρόλος του JSP Container είναι να αναλύει (parsing) τις ετικέτες JSP , να παράγει το δυναμικό περιεχόμενο και να το στέλνει στον πελάτη ως σελίδα HTML.

Έτσι, συνοπτικά, οι JSP:

- είναι φυσική εξέλιξη της τεχνολογίας Servlets,
- αποτελούνται από αρχεία κειμένου με την κατάληξη .jsp, που περιέχουν HTML, τμήματα κώδικα (scriptlets) και ετικέτες τύπου XML,
- μεταγλωττίζονται μία φορά σε Servlets,
- διαχωρίζουν ευκολότερα την εμφάνιση της σελίδας (HTML) από την επιχειρησιακή λογική (business logic), που υλοποιεί η εφαρμογή,
- διαχωρίζουν τη σχεδίαση από την υλοποίηση της λογικής,
- ενσωματώνουν κώδικα Java μέσα σε ιστοσελίδες HTML, αντίθετα από τα Servlets, που ενσωματώνουν κώδικα HTML μέσα σε προγράμματα Java, και
- λαμβάνουν τα αποτελέσματα και τα εμφανίζουν, ενώ τα Servlets κάνουν επεξεργασία της αίτησης.

2.8. HTML

Η HTML (αρχικοποίηση του αγγλικού HyperText Markup Language, ελλ. Γλώσσα Σήμανσης Υπερκειμένου) είναι η κύρια γλώσσα σήμανσης για τις ιστοσελίδες, και τα στοιχεία της είναι τα βασικά δομικά στοιχεία των ιστοσελίδων.

Η HTML γράφεται υπό μορφή στοιχείων HTML τα οποία αποτελούνται από ετικέτες (tags), οι οποίες περικλείονται μέσα σε σύμβολα «μεγαλύτερο από» και «μικρότερο από» (για παράδειγμα <html>), μέσα στο περιεχόμενο της ιστοσελίδας. Οι ετικέτες HTML συνήθως λειτουργούν ανά ζεύγη (για παράδειγμα <h1> και </h1>), με την πρώτη να ονομάζεται ετικέτα έναρξης και τη δεύτερη ετικέτα λήξης (ή σε άλλες περιπτώσεις ετικέτα ανοίγματος και ετικέτα κλεισίματος αντίστοιχα). Ανάμεσα στις ετικέτες, οι σχεδιαστές ιστοσελίδων μπορούν να τοποθετήσουν κείμενο, πίνακες, εικόνες κλπ.

Ο σκοπός ενός web browser είναι να διαβάζει τα έγγραφα HTML και να τα συνθέσει σε σελίδες που μπορεί κανείς να διαβάσει ή να ακούσει. Ο browser δεν εμφανίζει τις ετικέτες HTML, αλλά τις χρησιμοποιεί για να παρουσιάσει το περιεχόμενο της σελίδας.

Τα στοιχεία της HTML χρησιμοποιούνται για να κτίσουν όλους του ιστότοπους. Η HTML επιτρέπει την ενσωμάτωση εικόνων και άλλων αντικειμένων μέσα στη σελίδα, και μπορεί να χρησιμοποιηθεί για να εμφανίσει διαδραστικές φόρμες. Παρέχει τις μεθόδους δημιουργίας δομημένων εγγράφων (δηλαδή εγγράφων που αποτελούνται από το περιεχόμενο που μεταφέρουν και από τον κώδικα μορφοποίησης του περιεχομένου) καθορίζοντας δομικά σημαντικά στοιχεία για το κείμενο, όπως κεφαλίδες, παραγράφους, λίστες, συνδέσμους, παραθέσεις και άλλα. Μπορούν επίσης να ενσωματώνονται σενάρια εντολών σε γλώσσες όπως η JavaScript, τα οποία επηρεάζουν τη συμπεριφορά των ιστοσελίδων HTML και από στατικές τις κάνουν διαδραστικές.

Οι Web browsers μπορούν επίσης να αναφέρονται σε στυλ μορφοποίησης CSS για να ορίζουν την εμφάνιση και τη διάταξη του κειμένου και του υπόλοιπου υλικού. Ο οργανισμός W3C, ο οποίος

δημιουργεί και συντηρεί τα πρότυπα για την HTML και τα CSS, ενθαρρύνει τη χρήση των CSS αντί διαφόρων στοιχείων της HTML για σκοπούς παρουσίασης του περιεχομένου.

2.9. CSS

Η CSS (Cascading Style Sheets – διαδοχικά φύλλα ύφους ή επάλληλα φύλλα ύφους) είναι μια γλώσσα υπολογιστή που ανήκει στην κατηγορία των γλωσσών φύλλων ύφους που χρησιμοποιείται για τον έλεγχο της εμφάνισης ενός εγγράφου που έχει γραφτεί με μια γλώσσα σήμανσης.

Χρησιμοποιείται δηλαδή για τον έλεγχο της εμφάνισης ενός εγγράφου που γράφτηκε στις γλώσσες HTML και XHTML, δηλαδή για τον έλεγχο της εμφάνισης μιας ιστοσελίδας και γενικότερα ενός ιστοτόπου. Η CSS είναι μια γλώσσα υπολογιστή προορισμένη να αναπτύσσει στυλιστικά μια ιστοσελίδα δηλαδή να διαμορφώνει περισσότερα χαρακτηριστικά, χρώματα, στοίχιση και δίνει περισσότερες δυνατότητες σε σχέση με την html. Για μια όμορφη και καλοσχεδιασμένη ιστοσελίδα η χρήση της CSS κρίνεται ως απαραίτητη.

2.10. JavaScript

Η JavaScript (JS) είναι διερμηνευμένη γλώσσα προγραμματισμού για ηλεκτρονικούς υπολογιστές. Αρχικά αποτέλεσε μέρος της υλοποίησης των φυλλομετρητών Ιστού, ώστε τα σενάρια από την πλευρά του πελάτη (client-side scripts) να μπορούν να επικοινωνούν με τον χρήστη, να ανταλλάσσουν δεδομένα ασύγχρονα και να αλλάζουν δυναμικά το περιεχόμενο του εγγράφου που εμφανίζεται.

Η JavaScript είναι μια γλώσσα σεναρίων που βασίζεται στα πρωτότυπα (prototype-based), είναι δυναμική, με ασθενείς τύπους και έχει συναρτήσεις ως αντικείμενα πρώτης τάξης. Η σύνταξή της είναι επηρεασμένη από τη C. Η JavaScript αντιγράφει πολλά ονόματα και συμβάσεις ονοματοδοσίας από τη Java, αλλά γενικά οι δύο αυτές γλώσσες δε σχετίζονται και έχουν πολύ διαφορετική σημασιολογία. Οι βασικές αρχές σχεδιασμού της JavaScript προέρχονται από τις γλώσσες προγραμματισμού Self και Scheme. Είναι γλώσσα βασισμένη σε διαφορετικά προγραμματιστικά παραδείγματα (multi-paradigm), υποστηρίζοντας αντικειμενοστρεφές, προστακτικό και συναρτησιακό στυλ προγραμματισμού.

Η JavaScript χρησιμοποιείται και σε εφαρμογές εκτός ιστοσελίδων — τέτοια παραδείγματα είναι τα έγγραφα PDF, οι εξειδικευμένοι φυλλομετρητές (site-specific browsers) και οι μικρές εφαρμογές της επιφάνειας εργασίας (desktop widgets). Οι νεότερες εικονικές μηχανές και πλαίσια ανάπτυξης για JavaScript (όπως το Node.js) έχουν επίσης κάνει τη JavaScript πιο δημοφιλή για την ανάπτυξη εφαρμογών Ιστού στην πλευρά του διακομιστή (server-side).

2.11. Αλγόριθμος μηχανιστικής μάθησης (machine learning)

Μηχανική μάθηση είναι υποπεδίο της επιστήμης των υπολογιστών που αναπτύχθηκε από τη μελέτη της αναγνώρισης προτύπων και της υπολογιστικής θεωρίας μάθησης στην τεχνητή νοημοσύνη. Το 1959, ο Άρθουρ Σάμουελ ορίζει τη μηχανική μάθηση ως "Πεδίο μελέτης που δίνει στους υπολογιστές την ικανότητα να μαθαίνουν, χωρίς να έχουν ρητά προγραμματιστεί". Η μηχανική μάθηση διερευνά τη μελέτη και την κατασκευή αλγορίθμων που μπορούν να μαθαίνουν από τα δεδομένα και να κάνουν προβλέψεις σχετικά με αυτά. Τέτοιοι αλγόριθμοι λειτουργούν κατασκευάζοντας μοντέλα από πειραματικά δεδομένα, προκειμένου να κάνουν προβλέψεις βασιζόμενες στα δεδομένα ή να εξάγουν αποφάσεις που εκφράζονται ως το αποτέλεσμα.

Η μηχανική μάθηση είναι στενά συνδεδεμένη και συχνά συγχέεται με υπολογιστική στατιστική, ένας κλάδος, που επίσης επικεντρώνεται στην πρόβλεψη μέσω της χρήσης των υπολογιστών. Έχει ισχυρούς δεσμούς με την μαθηματική βελτιστοποίηση, η οποία παρέχει μεθόδους, τη θεωρία και τομείς εφαρμογής. Η Μηχανική μάθηση εφαρμόζεται σε μια σειρά από υπολογιστικές εργασίες, όπου τόσο ο σχεδιασμός όσο και ο ρητός προγραμματισμός των αλγορίθμων είναι ανέφικτος. Παραδείγματα εφαρμογών αποτελούν τα φίλτρα spam (spam filtering), η οπτική αναγνώριση χαρακτήρων (OCR), οι μηχανές αναζήτησης και η υπολογιστική όραση. Η Μηχανική μάθηση μερικές φορές συγχέεται με την εξόρυξη δεδομένων, όπου η τελευταία επικεντρώνεται περισσότερο στην εξερευνητική ανάλυση των δεδομένων, γνωστή και ως μη επιτηρούμενη μάθηση.

Στο πεδίο της ανάλυσης δεδομένων, η μηχανική μάθηση είναι μια μέθοδος που χρησιμοποιείται για την επινόηση πολύπλοκων μοντέλων και αλγορίθμων που οδηγούν στην πρόβλεψη. Τα αναλυτικά μοντέλα επιτρέπουν στους ερευνητές, τους επιστήμονες δεδομένων, τους μηχανικούς και τους αναλυτές να παράγουν αξιόπιστες αποφάσεις και αποτελέσματα και να αναδείξουν αλληλοσυσχετίσεις μέσω της μάθησης από ιστορικές σχέσεις και τάσεις στα δεδομένα.

Οι εργασίες μηχανικής μάθησης συνήθως ταξινομούνται σε τρεις μεγάλες κατηγορίες, ανάλογα με τη φύση του εκπαιδευτικού «σήματος» ή την «ανατροφοδότηση» που είναι διαθέσιμα σε ένα σύστημα εκμάθησης. Αυτές είναι:

- Επιτηρούμενη μάθηση (αλλιώς επιβλεπόμενη μάθηση ή μάθηση με επίβλεψη) (supervised learning): Το υπολογιστικό πρόγραμμα δέχεται τις παραδειγματικές εισόδους καθώς και τα επιθυμητά αποτελέσματα από έναν «δάσκαλο», και ο στόχος είναι να μάθει έναν γενικό κανόνα προκειμένου να αντιστοιχίσει τις εισόδους με τα αποτελέσματα.
- Μη επιτηρούμενη μάθηση (αλλιώς μη επιβλεπόμενη μάθηση ή μάθηση χωρίς επίβλεψη (unsupervised learning): Χωρίς να παρέχεται κάποια εμπειρία στον αλγόριθμο μάθησης, πρέπει να βρεί την δομή των δεδομένων εισόδου. Η μη επιτηρούμενη μάθηση μπορεί να είναι αυτοσκοπός (ανακαλύπτοντας κρυμμένα μοτίβα σε δεδομένα) ή μέσο για ένα τέλος (χαρακτηριστικό της μάθησης).
- Ενισχυτική μάθηση: Ένα πρόγραμμα υπολογιστή αλληλεπιδρά με ένα δυναμικό περιβάλλον στο οποίο πρέπει να επιτευχθεί ένας συγκεκριμένος στόχος (όπως η οδήγηση ενός οχήματος), χωρίς κάποιος δάσκαλος να του λέει ρητά αν έχει φτάσει κοντά στο στόχο του. Ένα άλλο παράδειγμα είναι να μάθει να παίζει ένα παιχνίδι εναντίον κάποιου αντιπάλου.
- Μεταξύ της επιτηρούμενης και της μη επιτηρούμενης μάθησης είναι η ημι-επιτηρούμενη μάθηση, όπου ο δάσκαλος δίνει ένα ελλειπές εκπαιδευτικό σήμα: ένα σύνολο εκπαίδευσης με κάποια (συχνά πολλά) από τα αποτελέσματα στόχους να λείπουν. Η Μεταγωγή είναι μια ειδική περίπτωση της αρχής αυτής, όπου το σύνολο των καταστάσεων του προβλήματος είναι γνωστό κατά το χρόνο εκμάθησης, όμως ένα μέρος των στόχων λείπουν.

Μία μηχανή διανυσμάτων υποστήριξης, όπου τα δεδομένα ταξινομούνται σε δύο κλάσεις, που χωρίζονται από ένα γραμμικό σύνορο. Εδώ, έχει μάθει να διακρίνει τους μαύρους από τους άσπρους κύκλους.

Μεταξύ άλλων κατηγοριών μηχανικής μάθησης, υπάρχει ακόμα διαδικασία εκμάθησης (meta learning) που μαθαίνει στην μηχανή (να αναπτύσσει) τις δικές της επαγωγικές μεθόδους, βασιζόμενο στην προηγούμενη εμπειρία. Η Αναπτυξιακή μάθηση (Developmental robotics), η οποία έχει αναπτυχθεί για την εκμάθηση από ρομπότ, δημιουργεί τη δική της ακολουθία μαθησιακών καταστάσεων, ώστε το ρομπότ συσσωρευτικά αποκτά ποικιλία δεξιοτήτων μέσω

της αυτόνομης αυτοεξερεύνησης και της κοινωνικής αλληλεπίδρασης με ανθρώπους εκπαιδευτές και χρησιμοποιώντας μηχανισμούς καθοδήγησης, όπως η ενεργητική μάθηση, η ωρίμανση και η μίμηση.

Μια άλλη κατηγοριοποίηση των προβλημάτων μηχανικής μάθησης προκύπτει όταν κάποιος θεωρήσει το επιθυμητό αποτέλεσμα του σύστηματος μηχανικής μάθησης.

- Στην ταξινόμηση, τα δεδομένα εισόδου χωρίζονται σε δύο ή περισσότερες κλάσεις, και η μηχανή πρέπει να κατασκευάσει ένα μοντέλο, το οποίο θα αντιστοιχίζει τα δεδομένα σε μία ή περισσότερες (multi-label ταξινόμηση) κλάσεις. Αυτό συνήθως εμπίπτει στην επιτηρούμενη μάθηση. Τα φίλτρα Spam είναι ένα παράδειγμα ταξινόμησης, όπου οι είσοδοι είναι τα emails ή άλλα μηνύματα και οι κλάσεις είναι "spam" και "όχι spam".
- Στην παλινδρόμηση, επίσης πρόβλημα επιτηρούμενης μάθησης, τα αποτελέσματα είναι συνεχή και όχι διακριτά.
- Στην συσταδοποίηση, ένα σύνολο εισόδων πρόκειται να χωριστεί σε ομάδες. Σε αντίθεση με την ταξινόμηση, οι ομάδες δεν είναι γνωστές εκ των προτέρων, καθιστώντας αυτόν τον διαχωρισμό τυπική εργασία μη επιτηρούμενης μάθησης.
- Στην εκτίμηση πυκνότητας βρίσκει την κατανομή των δεδομένων εισόδου σε κάποιο χώρο.
- Σε προβλήματα μείωσης διαστασιμότητας (dimensionality reduction), τα δεδομένα απλοποιούνται και αντιστοιχίζονται σε ένα χώρο λιγότερων διαστάσεων. Το στατιστικό μοντέλο θεμάτων (Topic modeling) είναι ένα σχετικό πρόβλημα, όπου η μηχανή καλείται να βρει έγγραφα που καλύπτουν παρόμοια θέματα από ένα σύνολο εγγράφων γραμμένων σε φυσική γλώσσα.

2.11.1. K-means algorithm

Η ομαδοποίηση k-μέσων είναι μία μέθοδος διανυσματικής κβαντοποίησης η οποία είναι δημοφιλής στην ανάλυση συστάδων (κλάδος της εξόρυξης δεδομένων). Η ομαδοποίηση αυτή έχει ως στόχο να διαχωρίσει n παρατηρήσεις σε k ομάδες, έτσι ώστε κάθε παρατήρηση να ανήκει στη συστάδα με το κοντινότερο μέσο, το οποίο χρησιμεύει ως ένα χαρακτηριστικό δείγμα της συστάδας. Αυτό οδηγεί σε μια διαμέριση του χώρου δεδομένων σε κελιά Voronoi.

Ο αλγόριθμος k-means ξεκινάει με k τυχαία σημεία, τα οποία ονομάζονται κεντροειδή της συστάδας και δηλώνουν το κέντρο βάρους της συστάδας. Το k υποδηλώνει σε πόσες συστάδες θέλουμε ο αλγόριθμος να δημιουργήσει. Ο αλγόριθμος εκτελεί επαναληπτικά δύο βήματα. Το πρώτο βήμα αφορά την ανάθεση σε κάποια συστάδα, ενώ το δεύτερο βήμα αφορά τον επαναπροσδιορισμό και τη μετατόπιση του κεντροειδούς κάθε συστάδας.

Πιο αναλυτικά, όσον αφορά στο πρώτο βήμα, δηλαδή την ανάθεση σε κάποια συστάδα, ο αλγόριθμος εξετάζει κάθε δείγμα σε σχέση με τα κεντροειδή των συστάδων. Με χρήση κάποιου μέτρου απόστασης, αναθέτει το εξεταζόμενο δείγμα στη συστάδα, της οποίας το κεντροειδές είναι το πλησιέστερο ως προς το συγκεκριμένο δείγμα. Στο δεύτερο βήμα, παίρνοντας τον μέσο όρο των δειγμάτων κάθε συστάδας, επανυπολογίζονται τα κεντροειδή της κάθε συστάδας, ώστε το κεντροειδές να είναι πιο αντιπροσωπευτικό στην πρόσφατα διαμορφωμένη συστάδα.

Ο αλγόριθμος εκτελεί επαναληπτικά αυτά τα δύο βήματα, μέχρις ότου τα κεντροειδή των συστάδων να μετατοπίζονται ελάχιστα και σε απόσταση μικρότερη από κάποια δοθείσα τιμή κατωφλίου. Ως εναλλακτικό κριτήριο τερματισμού του αλγορίθμου μπορεί να χρησιμοποιηθεί και ο αριθμός επαναλήψεων του αλγορίθμου.

3. Ανάλυση στόχων εργασίας

Στόχος της παρούσας εργασίας είναι να μπορέσουμε να εκμεταλλευτούμε τα επιτεύγματα της τεχνολογίας και ειδικότερα της μηχανιστικής μάθησης, με σκοπό να μπορέσουμε να εξυπηρετήσουμε κάποιες ανάγκες της εκπαίδευσης. Πιο συγκεκριμένα, στόχος μας είναι να μπορέσουμε να παρέχουμε μια ολοκληρωμένη πλατφόρμα εκπαίδευσης για τον μαθητή που θα του δίνει συγκεκριμένα τη δυνατότητα επιτυχούς προετοιμασίας με σκοπό την επιτυχία του σε τελικές εξετάσεις.

3.1. Ανάλυση προβλήματος

Η μηχανιστική μάθηση έχει μπει τα τελευταία χρόνια στη ζωή μας και έχει επηρεάσει πολλές πτυχές της καθημερινότητας μας. Μέχρι τώρα όμως, δεν έχει εισχωρήσει σε ικανοποιητικό βαθμό στην πραγματική ανθρώπινη μάθηση. Τμήμα αυτού του κενού θέλουμε να προσπαθήσουμε καλύψουμε με την παρούσα εργασία, όπου θέτουμε τη μηχανιστική μάθηση στην υπηρεσία του μαθητή/φοιτητή/εκπαιδευόμενου ώστε να μπορέσει να προετοιμαστεί με τον καλύτερο δυνατό τρόπο για τις τελικές εξετάσεις σε κάποιο μάθημα. Για το σκοπό αυτό του δίνουμε τη δυνατότητα μέσα από το online περιβάλλον να παρακολουθήσει μια σειρά από μαθήματα και να εξεταστεί σε μια σειρά από τεστ για να μπορέσει να προετοιμαστεί επαρκώς για την τελική εξέταση. Μέσα από καταγραφή πληροφοριών από την πορεία της προετοιμασίας του και από ανάλογα αποτελέσματα άλλων μαθητών, επιτυγχάνουμε να υπολογίσουμε με σχετική ακρίβεια την επίδοση του μαθητή στην τελική εξέταση.

3.2. Απαιτήσεις συστήματος

Για να μπορέσουμε να επιτύχουμε το σωστό σχεδιασμό του περιβάλλοντος που θα μας δίνει τη δυνατότητα να επιτύχουμε τον σκοπό μας χρειαζόμαστε τα εξής δομικά υλικά:

- Ένα κομμάτι κώδικα το οποίο θα εκτελείται στο παρασκήνιο και θα εκτελεί όλη την υλοποίηση της λογικής και της διασύνδεσης με τη βάση δεδομένων (backend). Το κομμάτι αυτό έχει υλοποιηθεί σε JAVA με τη βοήθεια του Spring framework.
- Ένα κομμάτι κώδικα το οποίο θα εκτελείται στο προσκήνιο και θα λειτουργεί ως διεπαφή μεταξύ του τελικού χρήστη και του υπολογιστή (frontend). Το τμήμα αυτό έχει υλοποιηθεί σε με τη βοήθεια JSPs.
- Μία βάση δεδομένων η οποία θα αποθηκεύει όλη τη χρήσιμη πληροφορία όσον αφορά τα μαθήματα, τις εξετάσεις, του μαθητές, τους καθηγητές και τα αποτελέσματα των εξετάσεων. Το κομμάτι αυτό έχει υλοποιηθεί με τη βοήθεια της Postgresql, η οποία αποτελεί ένα σύγχρονο και αρκετά ευέλικτο Σύστημα Διαχείρισης Βάσεων Δεδομένων (DBMS).
- Ένα τμήμα κώδικα το οποίο αναλαμβάνει να συνδέσει τη λογική του προγράμματος με τη βάση δεδομένων. Το τμήμα αυτό έγινε και πάλι σε JAVA με τη βοήθεια του Hibernate Framework.

Αυτά αποτελούν τα βασικά δομικά εργαλεία που χρησιμοποιήσαμε για να καλύψουμε πλήρως τις απαιτήσεις του συστήματος που επιθυμούμε να δημιουργήσουμε.

3.3. Σχεδιασμός υλοποίησης

Η υλοποίηση της εργασίας στηρίχθηκε στα συστήματα και τις τεχνολογίες που περιγράφηκαν παραπάνω και εκτελέστηκε με βάση την περιγραφή του της υλοποίησης που ακολουθεί στο επόμενο κεφάλαιο. Όσον αφορά τη χρονική αλληλουχία που ακολουθήθηκε μετά τον αρχικό σχεδιασμό των απαιτήσεων, αυτή έχει ως εξής:

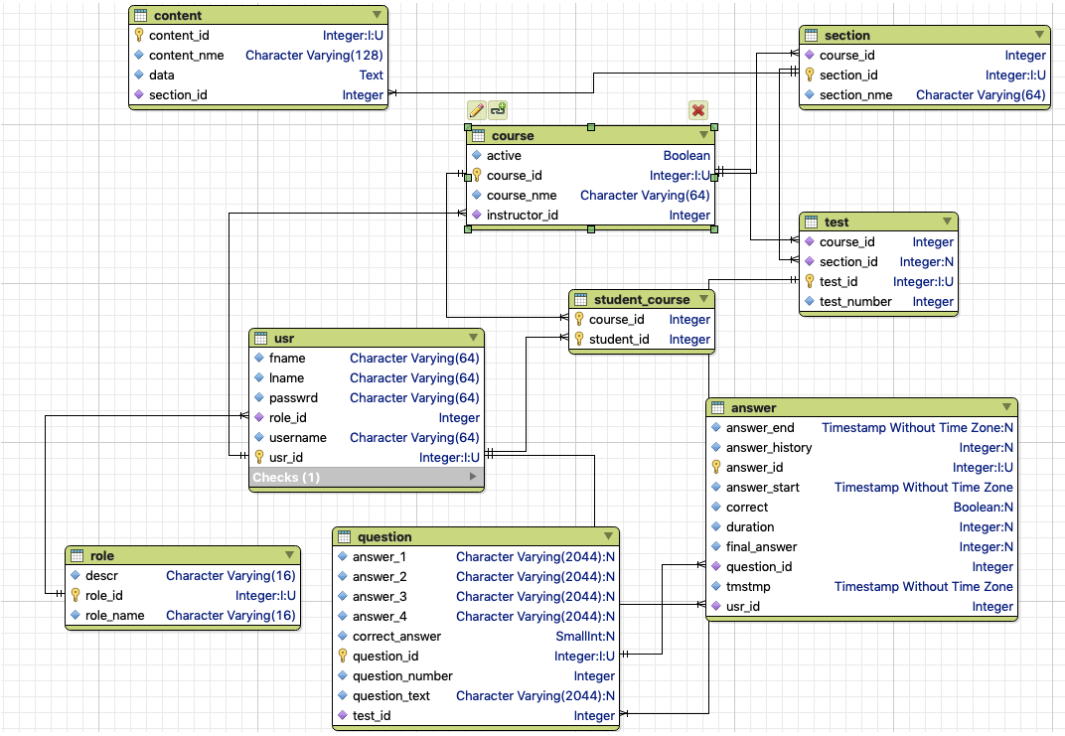
1. Αρχικά κατασκευάστηκε η βάση δεδομένων ώστε να μπορεί να φιλοξενήσει όλες τις λειτουργίες που επιθυμούσαμε.
2. Ακολούθως κατασκευάστηκε ένα μεγάλο κομμάτι της λογικής του backend

4. Υλοποίηση εργασίας

Σε αυτό το τμήμα της εργασίας περιγράφουμε λεπτομερώς τον κώδικα μας. Στο πρώτο μέρος αναλύουμε σε βάθος τη βάση δεδομένων που δημιουργήσαμε και ακολούθως την εφαρμογή μας, η οποία, όπως έχουμε ήδη αναφέρει στηρίζεται στο μοντέλο MVC και περιλαμβάνει την κυρίως εφαρμογή σε Java Spring, την διεπαφή με το χρήστη σε JSPs και την επικοινωνία με τη βάση σε Hibernate.

4.1. Βάση δεδομένων

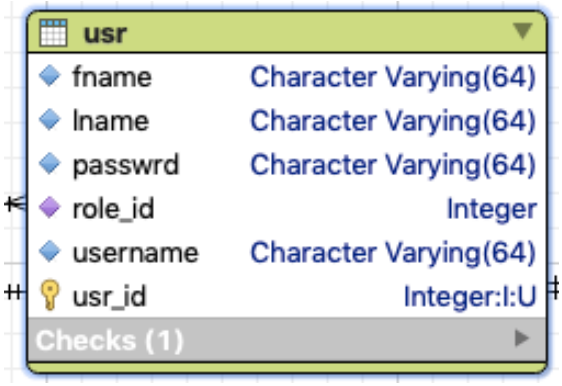
Το Σύστημα Διαχείρισης Βάσεων Δεδομένων που επιλέξαμε να χρησιμοποιήσουμε είναι η Postgresql, όπως έχει αναφερθεί και προηγουμένως. Παρακάτω περιγράφουμε όλες τις οντότητες και τις συσχετίσεις που έχουμε δημιουργήσει.



Εικόνα 2: Διάγραμμα Οντοτήτων-Συσχετίσεων της βάσης μας. (ER model)

4.1.1. User

Η πρώτη και πιο βασική οντότητα που έχουμε είναι αυτή που αποθηκεύει όλα τα στοιχεία των ατόμων που έρχονται σε επαφή με την εφαρμογή μας. Αυτοί μπορεί να είναι καθηγητές, μαθητές ή και διαχειριστές. Σε αυτό τον πίνακα αποθηκεύουμε το όνομα, το επίθετο, τον κωδικό και το username του πελάτη. Τέλος, μέσα από τη συσχέτιση με την οντότητα role, σημειώνουμε και τον ρόλο του κάθε χρήστη, δηλαδή αν είναι καθηγητής, μαθητής ή διαχειριστής.



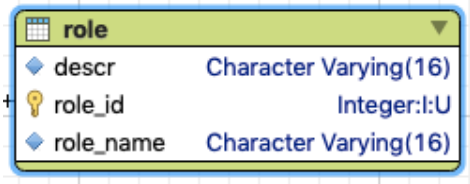
Εικόνα 3: Η οντότητα usr

4.1.2. Role

Μία άλλη βασική οντότητα είναι η οντότητα Role που μας δείχνει ποιος είναι ο ρόλος κάθε χρήστη. Όπως αναφέραμε και παραπάνω, η παρούσα υλοποίηση έχει τρεις ρόλους, του εξής:

- Καθηγητής
- Μαθητής
- Διαχειριστής

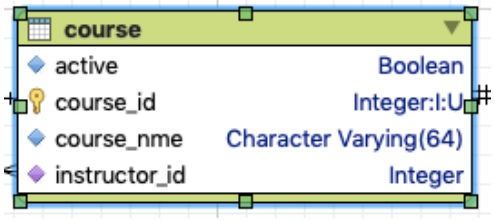
Οι δύο πρώτοι έχουν έναν εμφανή ρόλο στο σύστημα, ενώ ο τελευταίος επέχει ρόλο διαχειριστή του συστήματος, καθώς μπορεί να ελέγχει το σύνολο των μαθημάτων και το σύνολο των χρηστών και των στοιχείων τους.



Εικόνα 4: Η οντότητα role

4.1.3. Course

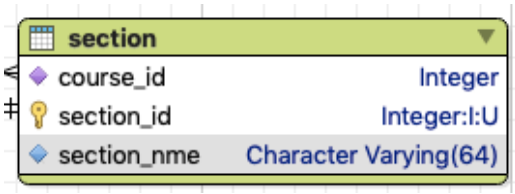
Μια ακόμη οντότητα που έχουμε είναι αυτή του μαθήματος, course. Η οντότητα αυτή φιλοξενεί όλα τα στοιχεία των μαθημάτων και επίσης έχει ευθεία συσχέτιση με τον πίνακα usr στον οποίο υπάρχει ο καθηγητής του κάθε μαθήματος.



Εικόνα 5: Η οντότητα course

4.1.4. Section

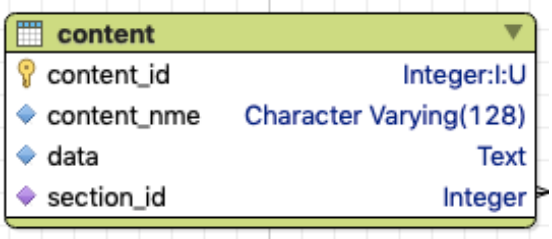
Η κάθε ενότητα του μαθήματος βρίσκεται στην οντότητα section και έχει ένα όνομα και τη συσχέτιση με τον πίνακα course.



Εικόνα 6: Η οντότητα section

4.1.5. Content

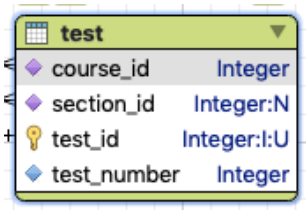
Τελειώνοντας αυτή τη σειρά συσχετίσεων, έχουμε τον πίνακα content, που φέρει το περιεχόμενο για κάθε τμήμα του μαθήματος. Εκτός από το όνομα και το πραγματικό περιεχόμενο, φέρει και συσχέτιση προς την οντότητα section.



Εικόνα 7: Η οντότητα content

4.1.6. Test

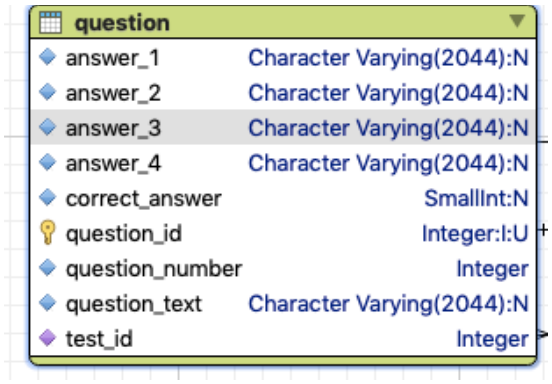
Μια άλλη χρήσιμη οντότητα, που υπάγεται στο κομμάτι των εξετάσεων, είναι αυτή του test. Περιλαμβάνει τα βασικά στοιχεία του κάθε τεστ, δηλαδή τον αριθμό του, το μάθημα με το οποί σχετίζεται καθώς και την ενότητα (section) με τα οποία σχετίζεται.



Εικόνα 8: Η οντότητα test

4.1.7. Question

Μία εξίσου σημαντική οντότητα για το κομμάτι των διαγωνισμάτων είναι η οντότητα question, που περιλαμβάνει πληροφορίες για τις ερωτήσεις των διαγωνισμάτων. Περιλαμβάνει την ερώτηση, τις πιθανές απαντήσεις, τη σωστή απάντηση και τη συσχέτισή της με την οντότητα των τεστ.



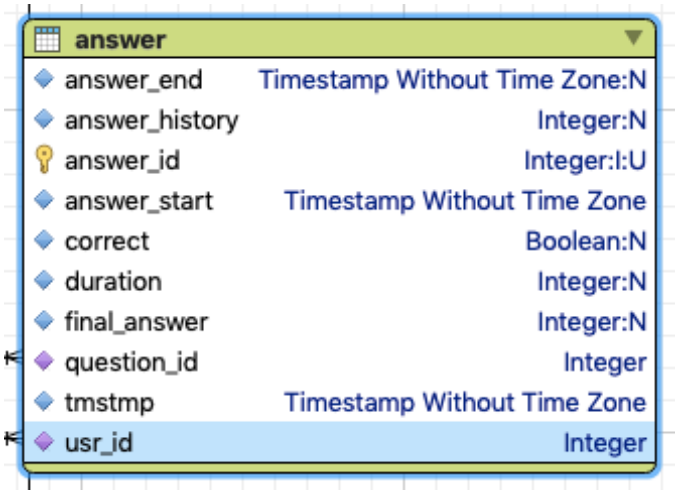
Εικόνα 9: Η οντότητα question

4.1.8. Answer

Η οντότητα answer είναι αυτή που περιλαμβάνει τις απαντήσεις των μαθητών στις διάφορες ερωτήσεις. Περιλαμβάνει επίσης μια σειρά από στοιχεία τα οποία τα χρησιμοποιούμε ακολούθως στον αλγόριθμο k-means για να μπορέσουμε να εξάγουμε τα επιθυμητά αποτελέσματα. Τα στοιχεία αυτά είναι τα ακόλουθα:

- Ώρα έναρξης της απάντησης
- Ώρα λήξης της απάντησης
- Πλήθος παλιών απαντήσεων
- Διάρκεια απάντησης
- Τελική απάντηση
- Ορθότητα απάντησης

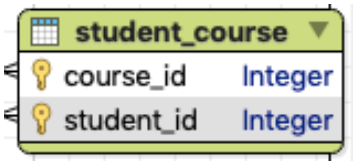
Επίσης περιλαμβάνει συσχετίσεις προς την οντότητα usr, για να υποδείξει τον μαθητή που απαντάει στην ερώτηση, καθώς και προς την οντότητα question για να γνωρίζουμε με ποια ερώτηση σχετίζεται.



Εικόνα 10: Η οντότητα answer

4.1.9. Student's course

Μια τελευταία συσχέτιση αυτή τη φορά που δημιουργήσαμε είναι η student_course, που μας υποδεικνύει ποια μαθήματα έχει πάρει ο κάθε μαθητής. Δημιουργήσαμε επιπλέον πίνακα, καθώς πρόκειται για μια συσχέτιση πολλά προς πολλά, καθώς ένας μαθητής μπορεί να εγγραφεί σε πολλά μαθήματα, αλλά και κάθε μάθημα μπορούν να το επιλέξουν πολλοί μαθητές.



Εικόνα 11: Η συσχέτιση student_course

4.2. Υλοποίηση αρχείων XML

Ένα πολύ βασικό στοιχείο της υλοποίησης της λύσης μας με χρήση Spring είναι τα διάφορα αρχεία XML τα οποία και αποτελούν βασικό στοιχείο του Spring Framework. Στην ενότητα αυτή αναφέρουμε τα 4 βασικά αρχεία που έχουμε δημιουργήσει και παραθέτουμε τον αντίστοιχο κώδικά τους.

4.2.1. Pom.xml

Ίσως το πιο βασικό αρχείο της υλοποίησής μας είναι το αρχείο pom.xml. Το αρχείο αυτό περιλαμβάνει το σύνολο των εξαρτήσεων του προγράμματός μας από εξωτερικές βιβλιοθήκες και επιπρόσθετα σημεία κώδικα. Στην περίπτωσή μας έχουμε χρησιμοποιήσει Spring, hibernate, postgresql και μια σειρά από άλλες μικρότερες βιβλιοθήκες που ολοκληρώνουν την εικόνα της εφαρμογής μας. Παραθέτουμε εδώ τον πλήρη κώδικα του αρχείου:

```
<project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://maven.apache.org/POM/4.0.0"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.smartstudent</groupId>
    <artifactId>account</artifactId>
    <packaging>war</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>Smartstudent Webapp</name>
    <url>http://maven.apache.org</url>
    <properties>
        <spring.version>4.2.0.RELEASE</spring.version>
        <spring-security.version>4.0.2.RELEASE</spring-security.version>
        <spring-data-jpa.version>1.8.2.RELEASE</spring-data-jpa.version>
        <hibernate.version>4.3.11.Final</hibernate.version>
        <hibernate-validator.version>5.2.1.Final</hibernate-validator.version>
        <postgresql-connector.version>42.2.6</postgresql-connector.version>
        <commons-dbcp.version>1.4</commons-dbcp.version>
        <jstl.version>1.2</jstl.version>
        <junit.version>3.8.1</junit.version>
        <logback.version>1.1.3</logback.version>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>${spring.version}</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${spring.version}</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-web</artifactId>
            <version>${spring-security.version}</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
            <version>${spring-security.version}</version>
        </dependency>

        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-validator</artifactId>
            <version>${hibernate-validator.version}</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.data</groupId>
            <artifactId>spring-data-jpa</artifactId>
```

```
            <version>${spring-data-jpa.version}</version>
        </dependency>

        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-entitymanager</artifactId>
            <version>${hibernate.version}</version>
        </dependency>

               <dependency>
                   <groupId>org.postgresql</groupId>
                   <artifactId>postgresql</artifactId>
                   <version>${postgresql-connector.version}</version>
               </dependency>

        <dependency>
            <groupId>commons-dbcp</groupId>
            <artifactId>commons-dbcp</artifactId>
            <version>${commons-dbcp.version}</version>
        </dependency>

        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
            <version>${jstl.version}</version>
        </dependency>

        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
            <version>${logback.version}</version>
        </dependency>

               <!-- https://mvnrepository.com/artifact/org.json/json -->
               <dependency>
                   <groupId>org.json</groupId>
                   <artifactId>json</artifactId>
                   <version>20190722</version>
               </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.eclipse.jetty</groupId>
                <artifactId>jetty-maven-plugin</artifactId>
                <version>9.2.11.v20150529</version>
                <configuration>
                    <scanIntervalSeconds>10</scanIntervalSeconds>
                    <webApp>
                        <contextPath>/</contextPath>
                    </webApp>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

```
</project>
```

4.2.2. Web.xml

Ένα άλλο αρχείο xml το οποίο μας δίνει πολλές πληροφορίες για τα views και κυρίως για τους controllers είναι το αρχείο web.xml. Εδώ θέτουμε βασικές επιλογές για τη θέση και τα βασικά χαρακτηριστικά των views και τον controllers (relative paths), καθώς και για βασικά χαρακτηριστικά του spring framework. Παραθέτουμε και πάλι τον πλήρη κώδικα.

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         version="2.5">

    <display-name>Smart Student Web Application</display-name>

    <!-- Location of Java @Configuration classes that configure the components that
makeup this application -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/appconfig-root.xml</param-value>
    </context-param>

    <filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-
class>
    </filter>
    <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value></param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
    </listener>
</web-app>
```

4.2.3. Appconfig-root.xml

Ένα ακόμα αρχείο που βοηθάει στη ρύθμιση των παραμέτρων για τα bean της εφαρμογής, καθώς και για την ασφάλεια και για το μοντέλο mvc είναι το appconfig-root, το οποίο περιλαμβάνει βασικές ρυθμίσεις και παρατίθεται πλήρως παρακάτω.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns="http://www.springframework.org/schema/beans"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd

              http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

    <import resource="appconfig-mvc.xml"/>

    <import resource="appconfig-data.xml"/>

    <import resource="appconfig-security.xml"/>

    <!-- Scans within the base package of the application for @Component classes to
configure as beans -->
    <context:component-scan base-package="com.smartstudent.*"/>

    <context:property-placeholder location="classpath:application.properties"/>

</beans>
```

4.2.4. Appconfig-data.xml

Τη σειρά των αρχείων που περιλαμβάνουν τις βασικές ρυθμίσεις τη εφαρμογή ολοκληρώνει το αρχείο appconfig-data το οποίο καθορίζει τους driver που θα χρησιμοποιηθούν για την επικοινωνία με τη βάση, το πακέτο στο οποίο περιλαμβάνεται το κομμάτι του μοντέλου της εφαρμογής, καθώς και τα αρχεία που θα περιλαμβάνουν τον sql κώδικα. Τέλος καθορίζει τις παραμέτρους σύνδεσης με τη βάση που εν προκειμένω τους λαμβάνουμε από κάποιο configuration file. Ο πλήρης κώδικας παρατίθεται παρακάτω.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jpa="http://www.springframework.org/schema/data/jpa"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns="http://www.springframework.org/schema/beans"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/data/jpa
    http://www.springframework.org/schema/data/jpa/spring-jpa.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx.xsd">

    <!-- Configure the data source bean -->
    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-
method="close">
        <property name="driverClassName" value="${jdbc.driverClassName}"/>
        <property name="url" value="${jdbc.url}"/>
        <property name="username" value="${jdbc.username}"/>
```

```
        <property name="password" value="${jdbc.password}"/>
    </bean>

    <!-- Configure the entity manager factory bean -->
    <bean id="entityManagerFactory"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
        <property name="dataSource" ref="dataSource"/>
        <property name="packagesToScan" value="com.smartstudent.model"/>
        <property name="jpaVendorAdapter">
            <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter"/>
        </property>
        <property name="jpaProperties">
            <props>
                <prop key="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</prop>
                <prop key="hibernate.show_sql">true</prop>
                <prop key="hibernate.default_schema">main</prop>
            </props>
        </property>
    </bean>

    <!-- Configure the transaction manager bean -->
    <bean id="transactionManager"
          class="org.springframework.orm.jpa.JpaTransactionManager">
        <property name="entityManagerFactory" ref="entityManagerFactory"/>
    </bean>

    <!-- Enable annotation driven transaction management -->
    <tx:annotation-driven/>

    <!--
      Configure Spring Data JPA and set the base package of the
      repository interfaces
    -->

    <jpa:repositories base-package="com.smartstudent.repository"/>

</beans>
```

4.3. Υλοποίηση μοντέλου (model) στην JAVA με χρήση Hibernate

Στο κεφάλαιο αυτό παραθέτουμε τις κλάσεις που με χρήση του Hibernate αντικατοπτρίζουν τη δομή και τα περιεχόμενα της βάσης δεδομένων στη γλώσσα JAVA. Ακολουθεί σύντομη περιγραφή και παράθεση των σημείων του κώδικα όπου αυτό έχει κριθεί απαραίτητο.

4.3.1. User

Η πρώτη κλάση που παραθέτουμε είναι η κλάση user, η οποία περιλαμβάνει όλα τα στοιχεία που χρειαζόμαστε να έχουμε από τη βάση οποιαδήποτε μορφή χρήστη, είτε αυτός είναι διαχειριστής, είτε είναι καθηγητής, είτε είναι σπουδαστής. Τα απαραίτητα πεδία είναι όνομα, επίθετο, κωδικός, συνθηματικό χρήστη, ρόλος και προφανώς το μοναδικό αναγνωριστικό της σχέσης.

Αξίζει βέβαια να σταθούμε σε τρία σημεία. Το ένα είναι το πεδίο επαλήθευσης του κωδικού, το οποίο στην πραγματικότητα δεν έχει κάποια αντιστοίχιση στη βάση και γι' αυτό καλείται transient. Το δεύτερο έχει να κάνει με το γεγονός ότι αν παρατηρήσουμε το μοναδικό αναγνωριστικό του πίνακα θα δούμε ότι έχει σημειωθεί με επιτυχία ότι αυτό παράγεται μόνο του αυτόματα.

Τέλος, αξίζει να παρατηρήσουμε τη σχέση πολλά προς ένα και το πώς αυτή σημειώνεται με τη βοήθεια του Spring Annotation στην περίπτωση του role, αλλά και τη σχέση πολλά προς πολλά στην περίπτωση της συσχέτισης μεταξύ των μαθημάτων και των μαθητών.

Για τον πίνακα user παραθέτουμε παρακάτω τον πλήρη κώδικα. Για τις υπόλοιπες κλάσεις παραθέτουμε μόνο τα βασικότερα σημεία.

```
@Entity
@Table(name = "usr")
public class User {
    private Long id;
    private String username;
    private String password;
    private String passwordConfirm;
    private Role role;
    private int roleId;
    private String fname;
    private String lname;
    private List<Course> courses = new ArrayList<Course>();

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name="usr_id")
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    @Column(name="username")
    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    @Column(name="passwrd")
    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
```

```
    }

    @Transient
    public String getPasswordConfirm() {
        return passwordConfirm;
    }

    public void setPasswordConfirm(String passwordConfirm) {
        this.passwordConfirm = passwordConfirm;
    }

    @ManyToOne
    @JoinColumn(name="role_id", nullable=false)
        public Role getRole() {
                return role;
        }

        public void setRole(Role role) {
                this.role = role;
        }

    @Transient
    public int getRoleId() {
        return roleId;
    }

    public void setRoleId(int roleId) {
        this.roleId = roleId;
    }

    @Column(name="fname")
        public String getFname() {
                return fname;
        }

        public void setFname(String fname) {
                this.fname = fname;
        }

    @Column(name="lname")
        public String getLname() {
                return lname;
        }

        public void setLname(String lname) {
                this.lname = lname;
        }
```

```
    @ManyToMany(
        fetch = FetchType.EAGER,
       cascade = CascadeType.ALL   )
        @JoinTable(
                name = "student_course",
                joinColumns = @JoinColumn(name = "student_id"),
                inverseJoinColumns = @JoinColumn(name = "course_id")
        )
        public List<Course> getCourses() {
                return courses;
        }

        public void setCourses(List<Course> courses) {
                this.courses = courses;
        }
}
```

4.3.2. Role

Ο επόμενος πίνακας που αναλύουμε είναι το πίνακας role ο οποίος αναφέρεται στο ρόλο που έχει ο κάθε χρήστης. Ο ρόλος αυτός μπορεί να είναι είτε καθηγητής, είτε μαθητής είτε διαχειριστής. Ανάλογα με το ρόλο που έχει ο κάθε χρήστης, προσαρμόζονται και οι σελίδες που αυτός βλέπει και οι δυνατότητες που έχει στα πλαίσια της εφαρμογής. Τα βασικά πεδία που περιλαμβάνονται εδώ είναι το αναγνωριστικό, το όνομα του ρόλου, η περιγραφή και η λίστα με τους χρήστες που έχουν τον ρόλο αυτό.

Το μόνο αξιοσημείωτο στην κλάση αυτή είναι και πάλι η σχέση ένα προς πολλά που εμφανίζεται μεταξύ ρόλου και χρηστών και είναι το μόνο τμήμα κώδικα που παραθέτουμε στο σημείο αυτό.

```
@Entity
@Table(name = "role")
public class Role {
    private Long id;
    private String rolename;
    private String descr;
    private List<User> users = new ArrayList<>();

    @OneToMany(
        cascade = CascadeType.ALL,
        orphanRemoval = true
    )
    public List<User> getUsers() {
        return users;
    }
```

```
    public void setUsers(List<User> users) {
        this.users = users;
    }
}
```

4.3.3. Course

Ακολούθως, εξετάζουμε τον πίνακα course, ο οποίος αποθηκεύει τα μαθήματα που έχουμε στην εφαρμογή. Τα βασικά στοιχεί του πίνακα αυτού που παρατίθενται και παρακάτω είναι το αναγνωριστικό του, το όνομα του αρχείου, ο εκπαιδευτής υπεύθυνος για το μάθημα και το αν το μάθημα έχει ενεργοποιηθεί ή όχι. Επίσης, περιλαμβάνονται οι λίστες για τα κεφάλαια που περιλαμβάνονται στο μάθημα, τα τεστ και τους μαθητές που έχουν πάρει το μάθημα. Τα δύο πρώτα περιλαμβάνουν μια σχέση ένα προς πολλά με τον πίνακα των μαθημάτων, ενώ το τελευταίο είναι μια σχέση πολλά προς πολλά, η μοναδική που παρουσιάζεται στην εφαρμογή μας, και καλύπτεται μέσα από τον ενδιάμεσο πίνακα συσχέτισης student_course.

Ακολούθως παρουσιάζονται τα πεδία του πίνακα, καθώς και οι βασικές συσχετίσεις του με άλλους πίνακες που αναφέραμε παραπάνω.

```
@Entity
@Table(name = "course")
public class Course implements Comparable<Object> {
    private Long id;
    private String courseName;
    private User instructor;
    private boolean active;
    private List<Section> sections = new ArrayList<Section>();
    private List<Test> tests = new ArrayList<Test>();
    private List<User> students = new ArrayList<User>();

    @OneToMany(
        mappedBy = "course",
        fetch = FetchType.EAGER,
        cascade = CascadeType.ALL,
        orphanRemoval = true
    )
    public List<Section> getSections() {
        return sections;
    }

    public void setSections(List<Section> sections) {
        this.sections = sections;
    }

    @OneToMany(
        mappedBy = "course",
        fetch = FetchType.EAGER,
```

```
        cascade = CascadeType.ALL,
        orphanRemoval = true
    )
    public List<Test> getTests() {
        return tests;
    }

    public void setTests(List<Test> tests) {
        this.tests = tests;
    }

    @ManyToMany(
        fetch = FetchType.EAGER,
        cascade = CascadeType.ALL
    )
    @JoinTable(
        name = "student_course",
        joinColumns = @JoinColumn(name = "course_id"),
        inverseJoinColumns = @JoinColumn(name = "student_id")
    )
    public List<User> getStudents() {
        return students;
    }

    public void setStudents(List<User> students) {
        this.students = students;
    }
```

4.3.4. Section

Επόμενος πίνακας που αναλύουμε είναι ο πίνακας section που περιλαμβάνει τα επιμέρους κεφάλαια του κάθε μαθήματος. Χαρακτηριστικά του είναι το αναγνωριστικό του, το όνομα του κεφαλαίου, το μάθημα στο οποίο ανήκει και η λίστα με τα περιεχόμενά του, που είναι μία συσχέτιση ένα προς πολλά προς των πίνακα content των περιεχομένων.

Παραθέτουμε τα πεδία του και τις βασικές συσχετίσεις του.

```
@Entity
@Table(name = "section")
public class Section {
    private Long id;
    private String sectionName;
    private Course course;
    private List<Content> contents = new ArrayList<>();

    @ManyToOne
    @JoinColumn(name="course_id", nullable=false)
```

```
    public Course getCourse() {
        return course;
    }

    public void setCourse(Course course) {
        this.course = course;
    }

    @OneToMany(
        mappedBy = "section",
        fetch = FetchType.EAGER,
        cascade = CascadeType.ALL,
        orphanRemoval = true
    )
    public List<Content> getContents() {
        return contents;
    }

    public void setContents(List<Content> contents) {
        this.contents = contents;
    }
}
```

4.3.5. Content

Ο πίνακας content περιλαμβάνει τα περιεχόμενα του κάθε μαθήματος που μπορεί να είναι βίντεο, κείμενο ή κάποια φωτογραφία. Απαρτίζεται από το αναγνωριστικό του πίνακα, το όνομα του περιεχομένου, το περιεχόμενο και το κεφάλαιο στο οποίο αυτό το περιεχόμενο υπάγεται.

Παραθέτουμε τα πεδία του και τη βασική συσχέτισή του που είναι αυτή με τον πίνακα section και είναι μια σχέση ένα προς πολλά.

```
@Entity
@Table(name = "content")
public class Content {
    private Long id;
    private String contentName;
    private String data;
    private Section section;

    @ManyToOne
    @JoinColumn(name="section_id", nullable=false)
    public Section getSection() {
        return section;
    }

    public void setSection(Section section) {
```

```
        this.section = section;
    }

    @Override
    public boolean equals(Object c) {
        return this.getId().equals(((Content)c).getId());
    }
}
```

4.3.6. Test

Ακόμα ένας πίνακας που έχουμε χρησιμοποιήσει είναι αυτός των test. Ο πίνακας αυτός περιλαμβάνει τις βασικές πληροφορίες για κάθε τεστ. Χαρακτηριστικά του είναι το αναγνωριστικό του, ο αριθμός του τεστ, το μάθημα στο οποίο ανήκει, το κεφάλαιο στο οποίο ανήκει (το οποίο είναι κενό όταν πρόκειται για τελικό τεστ) και η λίστα με τις ερωτήσεις που αυτό περιλαμβάνει. Παραθέτουμε εδώ τα πεδία του και τις βασικές συσχετίσεις του που είναι το course (ένα προς πολλά), το section (ένα προς πολλά) και οι ερωτήσεις (πολλά προς ένα).

```
@Entity
@Table(name = "test")
public class Test {
    private Long id;
    private int testNumber;
    private Course course;
    private Section section;
    private List<Question> questions;

    @ManyToOne
    @JoinColumn(name="course_id", nullable=false)
    public Course getCourse() {
        return course;
    }

    public void setCourse(Course course) {
        this.course = course;
    }

    @ManyToOne
    @JoinColumn(name="section_id", nullable=false)
    public Section getSection() {
        return section;
    }

    public void setSection(Section section) {
        this.section = section;
    }
```

```
    @OneToMany(
        mappedBy = "test",
        cascade = CascadeType.ALL,
        orphanRemoval = true
    )
    public List<Question> getQuestions() {
        return questions;
    }

    public void setQuestions(List<Question> questions) {
        this.questions = questions;
    }
}
```

4.3.7. Question

Ακολουθεί ο πίνακας question και το σύνολο των χαρακτηριστικών του. Πρόκειται για τον πίνακα που περιλαμβάνει τις ερωτήσει των τεστ. Βασικά χαρακτηριστικά του είναι το αναγνωριστικό του, η ερώτηση, οι τέσσερις πιθανές απαντήσεις, ο αριθμός της ερώτησης, η σωστή απάντηση και το τεστ στο οποίο ανήκει (συσχέτιση πολλά προς ένα). Παρακάτω παρατίθονται τα βασικά χαρακτηριστικά και η ανωτέρω συσχέτιση.

```
@Entity
@Table(name = "question")
public class Question implements Comparable<Object> {
    private Long id;
    private String questionText;
    private String answer1;
    private String answer2;
    private String answer3;
    private String answer4;
    private int questionNumber;
    private Integer correctAnswer;
    private Test test;

    @ManyToOne
    @JoinColumn(name="test_id", nullable=false)
    public Test getTest() {
        return test;
    }

    public void setTest(Test test) {
        this.test = test;
    }
}
```

4.3.8. Answer

Τέλος, έχουμε τον πίνακα των απαντήσεων, ο οποίος και είναι από τους πιο βασικούς πίνακες της υλοποίησής μας, καθώς συγκεντρώνει όλη την απαραίτητη πληροφορία για να μπορέσουμε ακολούθως να εκτελέσουμε τον αλγόριθμο K-means πάνω στα δεδομένα μας. Τα βασικά του χαρακτηριστικά είναι το αναγνωριστικό του, η τελική απάντηση του χρήστη, το κατά πόσο η απάντηση αυτή είναι σωστή, η ιστορικότητα των απαντήσεων του χρήστη, η διάρκεια απάντησης και η συσχετίσεις με τον πίνακα των ερωτήσεων (πολλά προς ένα) και με τον πίνακα των χρηστών/μαθητών (πολλά προς ένα). Αυτά τα βασικά πεδία και οι συσχετίσεις παρουσιάζονται παρακάτω.

```
@Entity
@Table(name = "answer")
public class Answer {
    private Long id;
    private int finalAnswer;
    private boolean correct;
    private int answerHistory[];
    private long duration;
    private Question question;
    private User user;

    @ManyToOne
    @JoinColumn(name="question_id", nullable=false)
    public Question getQuestion() {
        return question;
    }

    public void setQuestion(Question question) {
        this.question = question;
    }

    @ManyToOne
    @JoinColumn(name="usr_id", nullable=false)
    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }
}
```

4.4. Υλοποίηση αποθηκών (repository)

Σε αυτό το τμήμα της εργασίας παρουσιάζονται οι αλληλεπιδράσεις της εφαρμογής με τη βάση μέσα από τα σχετικά SQL queries που γίνονται μέσω του Hibernate. Σε κάθε repository περιλαμβάνονται τα σχετικά ερωτήματα που έχουν να κάνουν με τον αντίστοιχο πίνακα.

4.4.1. UserRepository

Το user repository είναι αυτό που περιλαμβάνει όλες τις βασικές λειτουργίες για δημιουργία λογαριασμού χρήστη, είσοδό του στο σύστημα, προσθήκη/διαγραφή μαθήματος από τα στοιχεία ενός μαθητή, ανανέωση των στοιχείων του μαθητή, εύρεση των μαθητών ενός μαθήματος, εύρεση των τεστ τα οποία έχει ολοκληρώσει μέχρι τώρα ο μαθητής. Όλα τα επιμέρους ερωτήματα είναι γραμμένα σε SQL και περιλαμβάνονται αμέσως μετά.

```
public interface UserRepository extends JpaRepository<User, Long> {
    User findByUsername(String username);

    @Transactional
    @Query(value = "select usr_id from usr where role_id = 0;",
      nativeQuery = true)
    Long[] findAllStudents();

    @Transactional
    @Modifying
    @Query(value = "insert into student_course (student_id, course_id) values (:user_id,
:course_id)",
      nativeQuery = true)
    void addCourse(@Param("user_id") Integer user_id, @Param("course_id") Integer
course_id);

    @Transactional
    @Modifying
    @Query(value = "delete from student_course where student_id = :user_id and course_id
= :course_id",
      nativeQuery = true)
    void removeCourse(@Param("user_id") Integer user_id, @Param("course_id") Integer
course_id);

    @Transactional
    @Query(value = "select count(distinct q.question_id) = 0 " +
           "from question q left join answer a on q.question_id = a.question_id and
a.final_answer is not null and a.answer_end < now() - interval '1 minute' and a.usr_id =
:user_id " +
           "where answer_id is null and test_id = :test_id ;",
      nativeQuery = true)
    boolean studentCompletedTest(@Param("user_id") long user_id, @Param("test_id") long
test_id);

    @Transactional
    @Query(value = "select count(distinct q.question_id) = 0 " +
                   "from question q inner join test using (test_id) " +
                   "                          left join answer a on q.question_id =
a.question_id and a.usr_id = :user_id " +
                   "where answer_id is null and test.section_id is not null and
course_id = :course_id ;",
          nativeQuery = true)
    boolean studentCompletedAllPrefinalTests(@Param("user_id") long user_id,
@Param("course_id") long course_id);
```

```
    @Transactional
    @Query(value = "select usr_id from usr where role_id = 0;",
      nativeQuery = true)
    Long [] getStudentsIds();

    @Transactional
    @Query(value = "select usr.* from student_course inner join usr on student_id =
usr_id and course_id = :course_id ;",
      nativeQuery = true)
       List<User> getCourseStudents(@Param("course_id") Long courseId);

    @Transactional
    @Modifying
    @Query(value = "update usr set username = :username, lname = :lname, fname = :fname,
role_id = :role_id where usr_id = :user_id ;",
      nativeQuery = true)
    void update(@Param("user_id") Long user_id, @Param("username") String username,
@Param("lname") String lname,
                @Param("fname") String fname, @Param("role_id") Long roleId);
}
```

4.4.2. RoleRepository

Η επόμενη αποθήκη που μελετάμε είναι αυτή του πίνακα role. Εδώ δε χρειαστήκαμε κάποια ιδιαίτερη υλοποίηση για πρόσβαση στη βάση, γι' αυτό δε θα ασχοληθούμε περαιτέρω.

```
public interface RoleRepository extends JpaRepository<Role, Long> {
    Role findById(long roleId);
}
```

4.4.3. CourseRepository

Ακολούθως μελετάμε τα ερωτήματα που σχετίζονται με τον πίνακα course των μαθημάτων. Εδώ έχουμε αρκετά ερωτήματα, τα οποία βοηθάνε στο να πάρουμε όλα τα ενεργά μαθήματα, τα ενεργά και τα ανενεργά μαθήματα ενός καθηγητή, τα ενεργά μαθήματα ενός μαθητή, τα ανενεργά μαθήματα, να ενεργοποιούμε ή να απενεργοποιούμε κατάλληλα κάποιο μάθημα και να σβήνουμε η να ενημερώνουμε τα στοιχεία ενός μαθήματος. Παραθέτουμε τον SQL κώδικα για τα ερωτήματα αυτά παρακάτω.

```
public interface CourseRepository extends JpaRepository<Course, Long> {
    Course findById(long courseId);

    List<Course> findAll();

    @Transactional
    @Query(value = "select course.* from course where active and course_id not in
(select distinct course_id from student_course where student_id = :user_id);",
      nativeQuery = true)
    List<Course> getActiveCourses(@Param("user_id") Long userId);

    @Transactional
    @Query(value = "select course.* from course where active and instructor_id =
:instructor_id ;",
      nativeQuery = true)
       List<Course> getInstructorActiveCourses(@Param("instructor_id") Long userId);
```

```
    @Transactional
    @Query(value = "select course.* from course where active is false and instructor_id
= :instructor_id ;",
      nativeQuery = true)
       List<Course> getInstructorInactiveCourses(@Param("instructor_id") Long userId);

    @Transactional
    @Query(value = "select course.* from course where active;",
      nativeQuery = true)
       List<Course> getActiveCourses();

    @Transactional
    @Query(value = "select course.* from course where active is false;",
      nativeQuery = true)
       List<Course> getInactiveCourses();

    @Transactional
    @Modifying
    @Query(value = "update course set active = true where course_id = :course_id ;",
      nativeQuery = true)
       void activateCourse(@Param("course_id") Long courseId);

    @Transactional
    @Modifying
    @Query(value = "update course set active = false where course_id = :course_id ;",
      nativeQuery = true)
       void deactivateCourse(@Param("course_id") Long courseId);

    @Transactional
    @Modifying
    @Query(value = "delete from course where course_id = :course_id ;",
      nativeQuery = true)
       void deleteCourse(@Param("course_id") Long courseId);

    @Transactional
    @Modifying
    @Query(value = "update course set course_nme = :course_name where course_id =
:course_id ;",
      nativeQuery = true)
       void update(@Param("course_id") Long courseId, @Param("course_name") String
courseName);
}
```

4.4.4. SectionRepository

Ακολούθως έχουμε τα ερωτήματα που αφορούν τον πίνακα section. Εδώ έχουμε μόνο δύο ερωτήματα που βοηθούν στη διαγραφή και την ανανέωση ενός κεφαλαίου. Παραθέτουμε τον κώδικα παρακάτω.

```
public interface SectionRepository extends JpaRepository<Section, Long> {
    Section findById(long sectionId);

    @Transactional
    @Modifying
    @Query(value = "delete from section where section_id = :section_id ;",
             nativeQuery = true)
    public void delete(@Param("section_id") long sectionId);
```

```
    @Transactional
    @Modifying
    @Query(value = "update section set section_nme = :section_name "
              + "where section_id = :section_id ;",
             nativeQuery = true)
    public void update(@Param("section_id") long sectionId,
              @Param("section_name")String sectionName);
}
```

4.4.5. ContentRepository

Και στην περίπτωση του content repository, η ανάγκη μας για πρόσβαση στη βάση και στον αντίστοιχο πίνακα περιορίζονται στη δυνατότητα διαγραφής και ανανέωσης των δεδομένων του πίνακα. Επίσης χρειάζεται να έχουμε τη δυνατότητα να βρούμε κάποιο συγκεκριμένο περιεχόμενο. Ο κώδικας για αυτά περιλαμβάνεται παρακάτω.

```
public interface ContentRepository extends JpaRepository<Content, Long> {
    Content findById(long contentId);

    @Transactional
    @Modifying
    @Query(value = "delete from content where content_id = :content_id ;",
             nativeQuery = true)
    public void delete(@Param("content_id") long contentId);

    @Transactional
    @Modifying
    @Query(value = "update content set content_nme = :content_name "
              + ", data = :content_data "
              + "where content_id = :content_id ;",
             nativeQuery = true)
    public void update(@Param("content_id") long contentId,
              @Param("content_name")String contentName,
              @Param("content_data")String contentData);
}
```

4.4.6. TestRepository

Μία ακόμα μεγάλη κλάση είναι το η αποθήκη για τον πίνακα test. Εδώ έχουμε αρκετά ερωτήματα, και πιο συγκεκριμένα ερωτήματα που βοηθάνε στη διαγραφή, εύρεση όλων των τεστ ενός μαθήματος, εύρεση του επόμενου αύξοντα αριθμού για ένα τεστ, εύρεση του τελικού διαγωνίσματος για ένα συγκεκριμένο μάθημα και την ανάκτηση όλων των αποτελεσμάτων των τεστ ενός μαθητή. Παραθέτουμε στη συνέχεια τον κώδικα από τα ερωτήματα αυτά.

```
public interface TestRepository extends JpaRepository<Test, Long> {
    Test findById(long testId);

    @Transactional
    @Modifying
    @Query(value = "delete from test where test_id = :test_id ;",
      nativeQuery = true)
       void delete(@Param("test_id") Long testId);

    @Transactional
```

```
    @Query(value = "select test.* from test where course_id = :course_id order by
course_id, section_id, test_number ;",
      nativeQuery = true)
    List<Test>findAllCourseTests(@Param("course_id") long courseId);

    @Transactional
    @Query(value = "select distinct on (course_id, section_id) test_number + 1 from test
where course_id = :course_id and section_id = :section_id order by course_id,
section_id, test_number desc ;",
      nativeQuery = true)
    int getNextTestNumber(@Param("course_id") long courseId, @Param("section_id") long
sectionId);

    @Transactional
    @Query(value = "select test.* from test where course_id = :course_id and section_id
is null limit 1;",
      nativeQuery = true)
    Test getCourseFinalTest(@Param("course_id") long courseId);

    @Transactional
    @Query(value = "select to_char(tmstmp, 'dd/mm/yyyy') || ';' || test_number || ';' ||
round(100.0*count(distinct answer.answer_id) filter (where correct)/count(distinct
answer.answer_id)) as result " +
                                "from main.answer inner join main.question using
(question_id) inner join main.test using (test_id) " +
                                "where usr_id = :student_id " +
                                "group by to_char(tmstmp, 'dd/mm/yyyy'),
test.test_number;",
      nativeQuery = true)
    List<String>getTestStudentResults(@Param("student_id") long studentId);
}
```

4.4.7. QuestionRepository

Η επόμενη αποθήκη που μελετάμε είναι αυτή του πίνακα question. Και εδώ οι ανάγκες μας περιορίζονται στο να βρίσκουμε το τεστ με βάση το αναγνωριστικό του, να κάνουμε ανανέωση του τεστ και να παίρνουμε τον επόμενο αριθμό ερώτησης. Ο κώδικάς τους παρατίθεται στη συνέχεια.

```
public interface QuestionRepository extends JpaRepository<Question, Long> {
    Question findById(long questionId);

    @Transactional
    @Query(value = "select question.* from question where test_id = :test_id order by
question_number ;",
       nativeQuery = true)
    List<Question> findByTestId(@Param("test_id") long testId);

    @Transactional
    @Modifying
    @Query(value = "update question set question_text = :question_text, answer_1 =
:answer_1,"
              + "answer_2 = :answer_2, answer_3 = :answer_3, answer_4 = :answer_4,
correct_answer = :correct_answer "
              + "where question_id = :question_id ;",
      nativeQuery = true)
    void update(@Param("question_id") long questionId, @Param("question_text") String
question, @Param("answer_1") String answer1,
```

```
                @Param("answer_2") String answer2, @Param("answer_3") String answer3,
@Param("answer_4") String answer4,
                @Param("correct_answer") int correctAnswer);

    @Transactional
    @Query(value = "select distinct on (test_id) question_number + 1 from question where
test_id = :test_id order by test_id, question_number desc ;",
        nativeQuery = true)
    int getNextQuestionNumber(@Param("test_id") long testId);
}
```

4.4.8. AnswerRepository

Τέλος, έχουμε και πάλι πιθανότατα την πιο σημαντική αποθήκη, που είναι αυτή που σχετίζεται με τον πίνακα answer. Εδώ έχουμε μια σειρά από ερωτήσεις που υποστηρίζουνε όλες τις ενέργειες που θέλουμε να εκτελούμε πάνω στον πίνακα answer, αλλά και μια σειρά από ερωτήματα για να λαμβάνουμε τα απαραίτητα δεδομένα για την υλοποίηση του αλγορίθμου K-means. Έτσι έχουμε ερωτήματα που εξυπηρετούν την αποθήκευση μιας ερώτησης, την εύρεση με βάση το αναγνωριστικό, την εύρεση όλων των αποτελεσμάτων, την εύρεση της απάντησης ενός μαθητή σε συγκεκριμένη ερώτηση, την εύρεση της απάντησης ενός μαθητή σε μια ερώτηση μέσα στο τελευταίο ημίωρο, την ανανέωση των στοιχείων μιας απάντησης, την εύρεση των απαντήσεων για ένα συγκεκριμένο μάθημα, την εύρεση του πλήθους των απαντήσεων για ένα συγκεκριμένο μάθημα, την εύρεση των σωστών απαντήσεων για ένα μάθημα και την εύρεση όλων των απαντήσεων ενός μαθητή για κάποιο μάθημα. Ο κώδικας περιλαμβάνεται στη συνέχεια και κλείνει αυτό το κεφάλαιο που περιλαμβάνει τον SQL κώδικα.

```
public interface AnswerRepository extends JpaRepository<Answer, Long> {
        @Transactional
    @Modifying
    @Query(value = "insert into answer (usr_id, question_id) values ( :user_id,
:question_id );",
      nativeQuery = true)
    void save(@Param("user_id") long userId, @Param("question_id") long questionid);

        Answer findById(long answerId);

        List<Answer> findAll();

    @Transactional
    @Query(value = "select distinct on (question_id, student_id) answer.* from answer "
                + "where question_id = :question_id and usr_id = :student_id order by
question_id, usr_id, tmstmp desc ;",
        nativeQuery = true)
    Answer findByQuestionStudentId(@Param("question_id") long questionId,
@Param("student_id") long studentid);

    @Transactional
    @Query(value = "select answer_id is null from answer "
                + "where question_id = :question_id and usr_id = :student_id and tmstmp >
now() - interval '30 minutes' "
                + "order by question_id, usr_id, tmstmp desc ;",
        nativeQuery = true)
    Boolean existsQuestionStudentIdHalfHour(@Param("question_id") long questionId,
@Param("student_id") long studentid);

    @Transactional
    @Query(value = "select final_answer from answer "
```

```
                + "where question_id = :question_id and usr_id = :student_id and tmstmp >
now() - interval '30 minutes' "
                + "order by question_id, usr_id, tmstmp desc ;",
        nativeQuery = true)
    Short getAnswerQuestionStudentIdHalfHour(@Param("question_id") long questionId,
@Param("student_id") long studentid);

    @Transactional
    @Modifying
    @Query(value = "update answer set answer_history = answer_history ||
array[:answer_number], final_answer = :answer_number, "
                + "correct = :answer_number = question.correct_answer "
                + "from question "
                + "where answer.question_id = question.question_id and answer.question_id
= :question_id "
                + "and usr_id = :student_id and tmstmp > now() - interval '30 minutes'
;",
        nativeQuery = true)
    void update(@Param("student_id") long studentId, @Param("question_id") long
questionId, @Param("answer_number") int answerNumber);

    @Transactional
    @Modifying
    @Query(value = "update answer set answer_end = now(), duration = EXTRACT(epoch FROM
(now() - answer_start)) "
                + "where question_id = :question_id and usr_id = :student_id and tmstmp >
now() - interval '30 minutes' ;",
        nativeQuery = true)
    void update(@Param("student_id") long studentId, @Param("question_id") long
questionId);

    @Transactional
    @Query(value = "select distinct on (answer.usr_id, question_id) usr_id || ';' ||
question_id || ';' || extract(epoch from (answer_end - answer_start)) || ';' ||
array_length(answer_history, 1) || ';' || case when correct then 1 else 0 end " +
                "from answer inner join question using (question_id) inner join test
using (test_id) " +
                "where course_id = :course_id " +
                "  and usr_id in (select usr_id from answer inner join question using
(question_id) inner join test using (test_id) " +
                "                                where course_id = :course_id group by
usr_id " +
                "                                having count(distinct question_id) =
(select count(distinct question_id) from question inner join test using (test_id) where
course_id = :course_id)) " +
                "and section_id is not null " +
                "order by answer.usr_id, question_id, tmstmp desc;",
        nativeQuery = true)
    List<String> getAnswersForCourse(@Param("course_id") long courseId);

    @Transactional
    @Query(value = "select 3 * count(distinct question_id) from question inner join test
using (test_id) where course_id = :course_id ;",
        nativeQuery = true)
    int getNumberOfAnswersForCourse(@Param("course_id") long courseId);

    @Transactional
    @Query(value = "select count(*) from answer inner join question using (question_id)
where correct and test_id = :test_id and usr_id = :usr_id ;",
        nativeQuery = true)
    int getCorrectAnswers(@Param("usr_id") long studentId, @Param("test_id") long
testId);
```

```
    @Transactional
    @Query(value = "select answer.* from answer inner join question using (question_id)
"
                + "inner join test using (test_id) where course_id = :course_id and
usr_id = :student_id order by question_id, answer_id;",
        nativeQuery = true)
    List<Answer> getStudentAnswersForCourse(@Param("course_id") long courseId,
@Param("student_id") long studentId);

    @Transactional
    @Query(value = "select round(100*count(*) filter (where correct) / count(*)) " +
                "from " +
                "(select distinct on (usr_id, question_id) usr_id, question_id, correct "
+
                "from answer inner join question using (question_id) inner join test
using (test_id) " +
                "where course_id = :course_id and section_id is null and usr_id in
:student_ids " +
                "order by usr_id, question_id, tmstmp desc) as q;",
        nativeQuery = true)
    int getFinalResult(@Param("course_id") Long courseId, @Param("student_ids")
List<Long> cusIds);
}
```

4.5. Υλοποίηση υπηρεσιών (services)

Αντίστοιχα προς τους παραπάνω πίνακες και τα repositories που αναφέρουμε έχουμε και τα σχετικά services (abstract) και τα σχετικά services implementation. Ο ρόλος αυτών είναι η επικοινωνία μεταξύ των controllers και των repositories και της βάσης δεδομένων, καθώς λειτουργούν ως spring beans. Κάθε πίνακας έχει και το αντίστοιχο service του. Παρακάτω παραθέτουμε ονομαστικά τα υπάρχοντα στην υλοποίησή μας services:

- UserService
- RoleService
- CourseService
- SectionService
- ContentService
- TestService
- QuestionService
- AnswerService

4.6. Υλοποίηση ελεγκτών (controllers)

Σε αυτό το τμήμα της εργασίας περιλαμβάνονται οι ελεγκτές (controllers) του μοντέλου που περικλείουν όλη τη λογική λειτουργία της εφαρμογής.

4.6.1. UserController

Ο πρώτος ελεγκτής που εξετάζουμε είναι αυτός του χρήστη (UserController). Ο ελεγκτής αυτός ελέγχει την εγγραφή, την είσοδο και την έξοδο του χρήστη. Εκτός αυτών ελέγχει και την αρχική οθόνη του κάθε χρήστη ανάλογα με τον ρόλο του (διαχειριστής, καθηγητής ή μαθητής) και οδηγεί στην κατάλληλη ιστοσελίδα. Τέλος εμφανίζει το σύνολο των χρηστών στην περίπτωση που ο

διαχειριστής θέλει να μπορεί να τους δει ή να τους τροποποιήσει, και φροντίζει να γίνεται η διαγραφή τους εφόσον αυτό κρίνεται αναγκαίο. Ο κώδικας της κλάσης περιλαμβάνεται ακολούθως.

```
@Controller
public class UserController {
        @Autowired
    private UserService userService;

        @Autowired
    private RoleService roleService;

    @Autowired
    private SecurityService securityService;

    @Autowired
    private CourseService courseService;

    @Autowired
    private UserValidator userValidator;

    @RequestMapping(value = "/registration", method = RequestMethod.GET)
    public String registration(Model model) {
        Map<Integer,String> roles = new LinkedHashMap<Integer,String>();
                roles.put(0, "Student");
                roles.put(1, "Instructor");
                roles.put(10, "Administrator");

        model.addAttribute("userForm", new User());
        model.addAttribute("roleList", roles);

         return "registration";
    }

    @RequestMapping(value = "/registration", method = RequestMethod.POST)
    public String registration(@ModelAttribute("userForm") User userForm, BindingResult
bindingResult, Model model) {
         userValidator.validate(userForm, bindingResult);

         if (bindingResult.hasErrors()) {
                Map<Integer,String> roles = new LinkedHashMap<Integer,String>();
                roles.put(0, "Student");
                roles.put(1, "Instructor");
                roles.put(10, "Administrator");
                model.addAttribute("roleList", roles);

                return "registration";
         }

         userForm.setRole(roleService.findById(userForm.getRoleId()));
         userService.save(userForm);

         securityService.autologin(userForm.getUsername(),
userForm.getPasswordConfirm());

        return "redirect:/welcome";
    }

    @RequestMapping(value = "/registrationAdmin", method = RequestMethod.POST)
```

```
    public String registrationAdmin(@ModelAttribute("userFormAdmin") User userForm,
BindingResult bindingResult, Model model) {
        userValidator.validate(userForm, bindingResult);

        if (userForm.getId() != null) {
            userForm.setRole(roleService.findById(userForm.getRoleId()));
            userService.update(userForm.getId(), userForm.getUsername(),
userForm.getLname(), userForm.getFname(), userForm.getRole().getId());

            Map<Integer,String> roles = new LinkedHashMap<Integer,String>();
            roles.put(0, "Student");
            roles.put(1, "Instructor");
            roles.put(10, "Administrator");
            model.addAttribute("roleList", roles);

            return "editStudents";
        }
        else if (bindingResult.hasErrors()) {
            Map<Integer,String> roles = new LinkedHashMap<Integer,String>();
            roles.put(0, "Student");
            roles.put(1, "Instructor");
            roles.put(10, "Administrator");
            model.addAttribute("roleList", roles);

            return "editStudents";
        }

        userForm.setRole(roleService.findById(userForm.getRoleId()));
        userService.save(userForm);

        List<User> users = userService.findAll();
        model.addAttribute("users", users);

        Map<Integer,String> roles = new LinkedHashMap<Integer,String>();
            roles.put(0, "Student");
            roles.put(1, "Instructor");
            roles.put(10, "Administrator");

        model.addAttribute("userFormAdmin", new User());
        model.addAttribute("roleList", roles);

        return "editStudents";
    }

    @RequestMapping(value = "/login", method = RequestMethod.GET)
    public String login(Model model, String error, String logout) {
        if (error != null)
            model.addAttribute("error", "Your username and password is invalid.");

        if (logout != null)
            model.addAttribute("message", "You have been logged out successfully.");

        return "login";
    }

    @RequestMapping(value = {"/", "/welcome"}, method = RequestMethod.GET)
    public String welcome(Model model) {
        Object principal =
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        String username = null;
        if (principal instanceof UserDetails) {
          username = ((UserDetails)principal).getUsername();
```

```
        } else {
          username = principal.toString();
        }

        User u = userService.findByUsername(username);
        if(u.getRole().getDescr().equals("instructor")) {
                System.out.println();
                List<Course> activeCourses =
courseService.getInstructorActiveCourses(u.getId());
                List<Course> inactiveCourses =
courseService.getInstructorInactiveCourses(u.getId());

                model.addAttribute("user", u);
                model.addAttribute("activeCourses", activeCourses);
                model.addAttribute("inactiveCourses", inactiveCourses);
                Collections.sort(activeCourses);
                Collections.sort(inactiveCourses);

                return "welcomeInstructor";
        }
        else if(u.getRole().getDescr().equals("admin")) {
                System.out.println();
                List<Course> activeCourses = courseService.getActiveCourses();
                List<Course> inactiveCourses = courseService.getInactiveCourses();

                model.addAttribute("user", u);
                model.addAttribute("activeCourses", activeCourses);
                model.addAttribute("inactiveCourses", inactiveCourses);
                Collections.sort(activeCourses);
                Collections.sort(inactiveCourses);

                return "welcomeAdmin";
        }
        else {
                List<Course> courses = u.getCourses();
                List<Course> allCourses = courseService.getActiveCourses(u.getId());
                Collections.sort(courses);
                Collections.sort(allCourses);

                model.addAttribute("user", u);
                model.addAttribute("courses", courses);
                model.addAttribute("allCourses", allCourses);

                return "welcomeStudent";
        }
    }

    @RequestMapping(value = {"/viewUsers"}, method = RequestMethod.POST)
    public String addCourse(Model model) {
        List<User> users = userService.findAll();
        model.addAttribute("users", users);

        Map<Integer,String> roles = new LinkedHashMap<Integer,String>();
                roles.put(0, "Student");
                roles.put(1, "Instructor");
                roles.put(10, "Administrator");

        model.addAttribute("userFormAdmin", new User());
        model.addAttribute("roleList", roles);

        return "editStudents";
    }
```

```
    @RequestMapping(value = {"/deleteUser"}, params = {"userId"}, method =
RequestMethod.POST)
    public String deleteUser(Model model, @RequestParam(value = "userId") int userId) {
        userService.delete(userId);
        List<User> users = userService.findAll();
        model.addAttribute("users", users);

        Map<Integer,String> roles = new LinkedHashMap<Integer,String>();
                roles.put(0, "Student");
                roles.put(1, "Instructor");
                roles.put(10, "Administrator");

        model.addAttribute("userFormAdmin", new User());
        model.addAttribute("roleList", roles);

        return "editStudents";
    }

}
```

4.6.2. CourseController

Ο επόμενος ελεγκτής που εξετάζουμε είναι αυτός του μαθήματος. Εδώ προφανώς εξυπηρετούνται πολύ περισσότερες ανάγκες. Πιο συγκεκριμένα έχουμε:

- Την προετοιμασία της σελίδας που εμφανίζει το μάθημα στον μαθητή.
- Την προετοιμασία της σελίδας που εμφανίζει τα παλιά αποτελέσματα.
- Την προσθήκη και την αφαίρεση μαθήματος από μαθητή
- Την ενεργοποίηση/απενεργοποίηση ενός μαθήματος
- Τη διαγραφή ενός μαθήματος.
- Τον καθορισμό των περιεχομένων ενός μαθήματος
- Την προετοιμασία της σελίδας για την εύρεση των μαθημάτων ενός μαθητή
- Την προετοιμασία της σελίδας για εμφάνιση των αποτελεσμάτων ενός μαθητή στα τεστ ενός μαθήματος
- Την προσθήκη και τη διαγραφή ενός μαθήματος
- Την προσθήκη και τη διαγραφή ενός κεφαλαίου από ένα μάθημα
- Την προσθήκη, τη διαγραφή και την ανανέωση του περιεχομένου ενός κεφαλαίου ενός μαθήματος

Όλες αυτές οι διαδικασίες περιλαμβάνονται στον ελεγκτή αυτό και παρατίθενται ακριβώς παρακάτω.

```
@Controller
public class CourseController {
        @Autowired
    private CourseService courseService;

        @Autowired
    private UserService userService;

        @Autowired
    private SectionService sectionService;
```

```
        @Autowired
    private ContentService contentService;

        @Autowired
        private TestService testService;

        @Autowired
        private AnswerService answerService;

    @RequestMapping(value = {"/course"}, params = {"courseId"}, method =
RequestMethod.POST)
    public String course(Model model, @RequestParam(value = "courseId") int courseId) {
            Object principal =
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        String username = null;
        if (principal instanceof UserDetails) {
          username = ((UserDetails)principal).getUsername();
        } else {
          username = principal.toString();
        }
        User u = userService.findByUsername(username);

        Course c = courseService.findCourseById(courseId);
        List<Test> tests = testService.findAllCourseTests(courseId);
        model.addAttribute("course", c);
        model.addAttribute("tests", tests);
        model.addAttribute("finalTestReady",
userService.studentCompletedAllPrefinalTests(u.getId(), c.getId()));
        model.addAttribute("errorMessage", "");
        try {
            KMeans kmeans = new KMeans(answerService, courseId);
            model.addAttribute("finalResults", kmeans.getResults(answerService,
u.getId(), courseId));
        } catch (Exception e) {
            model.addAttribute("finalResults", 0);
        }
        return "course";
    }

    @RequestMapping(value = {"/pastResults"}, params = {"courseId"}, method =
RequestMethod.POST)
    public String pastResults(Model model, @RequestParam(value = "courseId") int
courseId) {
            Object principal =
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        String username = null;
        if (principal instanceof UserDetails) {
          username = ((UserDetails)principal).getUsername();
        } else {
          username = principal.toString();
        }
        User u = userService.findByUsername(username);

        Course c = courseService.findCourseById(courseId);
        List<Test> tests = testService.findAllCourseTests(courseId);

        List<String> stringResults = testService.getTestStudentResults(u.getId());
        List<TestResultFromDB> results = new ArrayList<TestResultFromDB>();
        for (String stringResult : stringResults) {
            String [] data = stringResult.split(";");
            results.add(new TestResultFromDB(data[0], data[1], data[2]));
```

```
        }

        model.addAttribute("course", c);
        model.addAttribute("tests", tests);
        model.addAttribute("finalTestReady",
userService.studentCompletedAllPrefinalTests(u.getId(), c.getId()));
        model.addAttribute("errorMessage", "");
        model.addAttribute("answers",
answerService.getStudentAnswersForCourse(c.getId(), u.getId()));
                model.addAttribute("results", results);
                try {
                KMeans kmeans = new KMeans(answerService, courseId);
                model.addAttribute("finalResults", kmeans.getResults(answerService,
u.getId(), courseId));
        } catch (Exception e) {
                model.addAttribute("finalResults", 0);
        }
                return "testResults";
    }

    @RequestMapping(value = {"/addStudentCourse"}, params = {"courseId", "userId"},
method = RequestMethod.POST)
    public String addCourse(Model model, @RequestParam(value = "courseId") int courseId,
                @RequestParam(value = "userId") int userId) {
        userService.addCourse(userId, courseId);
        return "redirect:/welcome";
    }

    @RequestMapping(value = {"/removeCourse"}, params = {"courseId", "userId"}, method =
RequestMethod.POST)
    public String removeCourse(Model model, @RequestParam(value = "courseId") int
courseId,
                @RequestParam(value = "userId") int userId) {
        userService.removeCourse(userId, courseId);
        return "redirect:/welcome";
    }

    @RequestMapping(value = {"/activateCourse"}, params = {"courseId"}, method =
RequestMethod.POST)
    public String activateCourse(Model model, @RequestParam(value = "courseId") int
courseId) {
        courseService.activateCourse(courseId);
        return "redirect:/welcome";
    }

    @RequestMapping(value = {"/deactivateCourse"}, params = {"courseId"}, method =
RequestMethod.POST)
    public String deactivateCourse(Model model, @RequestParam(value = "courseId") int
courseId) {
        courseService.deactivateCourse(courseId);
        return "redirect:/welcome";
    }

    @RequestMapping(value = {"/deleteCourse"}, params = {"courseId"}, method =
RequestMethod.POST)
    public String deleteCourse(Model model, @RequestParam(value = "courseId") int
courseId) {
        courseService.deleteCourse(courseId);
        return "redirect:/welcome";
    }
```

```
    @RequestMapping(value = {"/setCourseContent"}, params = {"courseId"}, method =
RequestMethod.POST)
    public String setCourseContent(Model model, @RequestParam(value = "courseId") int
courseId) {
        Course course = courseService.findCourseById(courseId);
        model.addAttribute("course", course);

                return "courseContent";
    }

    @RequestMapping(value = {"/viewStudentsCourse"}, params = {"courseId"}, method =
RequestMethod.POST)
    public String viewStudentsCourseForm(Model model, @RequestParam(value = "courseId")
int courseId) {
        Course course = courseService.findCourseById(courseId);
        List<User> students = userService.getCourseStudents(courseId);
        model.addAttribute("course", course);
        model.addAttribute("students", students);
        model.addAttribute("answers", null);
        model.addAttribute("student", null);

                return "courseStudents";
    }

    @RequestMapping(value = {"/studentCourseAnswers"}, params = {"courseId"}, method =
RequestMethod.POST)
    public String studentCourseAnswers(Model model, @RequestParam(value = "courseId")
int courseId,
                @RequestParam(value = "studentId") int studentId) {
        Course course = courseService.findCourseById(courseId);
        List<User> students = userService.getCourseStudents(courseId);
        User student = userService.findById(studentId);
        List<Answer> answers = answerService.getStudentAnswersForCourse(courseId,
studentId);
        model.addAttribute("course", course);
        model.addAttribute("students", students);
        model.addAttribute("answers", answers);
        model.addAttribute("student", student);

                return "courseStudents";
    }

    @RequestMapping(value = {"/addNewCourse"}, params = {"addCourseInstructor",
"addCourseName"}, method = RequestMethod.POST)
    public String addNewCourse(Model model, @RequestParam(value = "addCourseInstructor")
int instructorId,
                @RequestParam(value = "addCourseName") String courseName) {
        Course course = new Course(courseName, userService.findById(instructorId),
false, null, null, null);
        courseService.save(course);
        int testNumber = 1;
        testService.addCourseTest(new Test(course, testNumber));

        return "redirect:/welcome";
    }

    @RequestMapping(value = {"/addCourseSection"}, params = {"addSectionCourseId"},
method = RequestMethod.POST)
    public String addCourseSection(Model model, @RequestParam(value =
"addSectionCourseId") int courseId) {
        Course course = courseService.findCourseById(courseId);
        sectionService.save(new Section("New Section", course));
```

```
        course = courseService.findCourseById(courseId);
        model.addAttribute("course", course);

                return "courseContent";
    }

    @RequestMapping(value = {"/removeCourseSection"}, params = {"removeSectionCourseId",
"removeSectionId"},
                method = RequestMethod.POST)
    public String removeCourseSection(Model model, @RequestParam(value =
"removeSectionCourseId") int courseId,
                @RequestParam(value = "removeSectionId") long sectionId) {
        sectionService.delete(sectionId);
        Course course = courseService.findCourseById(courseId);
        model.addAttribute("course", course);

                return "courseContent";
    }

    @RequestMapping(value = {"/addCourseContent"}, params = {"addContentCourseId",
"addContentSectionId"}, method = RequestMethod.POST)
    public String addCourseContent(Model model, @RequestParam(value =
"addContentCourseId") int courseId,
                @RequestParam(value = "addContentSectionId") int sectionId) {
        Course course = courseService.findCourseById(courseId);
        for (Section section : course.getSections()) {
                if(section.getId() == sectionId)
                        section.getContents().add(new Content());
        }

        model.addAttribute("course", course);

                return "courseContent";
    }

    @RequestMapping(value = {"/removeCourseContent"}, params = {"removeContentCourseId",
"removeContentId"},
                method = RequestMethod.POST)
    public String removeCourseContent(Model model, @RequestParam(value =
"removeContentCourseId") int courseId,
                @RequestParam(value = "removeContentId") long contentId) {
        contentService.delete(contentId);
        Course course = courseService.findCourseById(courseId);
        model.addAttribute("course", course);

                return "courseContent";
    }

    @RequestMapping(value = {"/saveCourseContent"},
                                params = {"courseId", "courseName", "sectionId",
"sectionName", "contentId", "contentName",
                                                        "content"},
                                method = RequestMethod.POST)
    public String saveCourseContent(Model model,
                @RequestParam(value = "courseId") long courseId,
                @RequestParam(value = "courseName") String courseName,
                @RequestParam(value = "sectionId") Long sectionIds [],
                @RequestParam(value = "sectionName") String sectionNames [],
                @RequestParam(value = "contentId") Long contentIds [],
                @RequestParam(value = "contentSectionId") Long contentSectionIds [],
                @RequestParam(value = "contentName") String contentNames [],
```

```
                @RequestParam(value = "content") String contents []) {

        int contentIndex = 0;
        int sectionIndex = 0;
                for (long contentSectionId : contentSectionIds) {
                 try {
                        if(contentIds[contentIndex] != null)
                                contentService.update(contentIds[contentIndex],
contentNames[contentIndex],
                                                        contents[contentIndex]);
                        else
                                contentService.save(new
Content(contentNames[contentIndex],
                                                        contents[contentIndex],
sectionService.findById(contentSectionId)));
                        } catch (ArrayIndexOutOfBoundsException e) {
                                contentService.save(new
Content(contentNames[contentIndex],
                                                contents[contentIndex],
sectionService.findById(contentSectionId)));
                        }

                if (sectionIds.length > sectionIndex && contentSectionId !=
sectionIds[sectionIndex])
                        sectionService.update(sectionIds[sectionIndex],
sectionNames[sectionIndex]);
                        sectionIndex++;
                contentIndex++;
                }
        if (sectionIds.length > sectionIndex)
                sectionService.update(sectionIds[sectionIndex],
sectionNames[sectionIndex]);

        courseService.update(courseId, courseName);

                Course course = courseService.findCourseById(courseId);
        model.addAttribute("course", course);

                return "courseContent";
    }
}
```

4.6.3. TestController

Ο τελευταίο ελεγκτής που έχουμε χρησιμοποιήσει είναι αυτός που σχετίζεται με την εγγραφή και τη διενέργεια των διαγωνισμάτων. Εδώ περιλαμβάνονται επίσης μια σειρά από διεργασίες που εξυπηρετούν τη λειτουργία των αντίστοιχων JSP σελίδων και οι οποίες είναι οι εξής:

- Προσθήκη/Επεξεργασία διαγωνίσματος σε κάποιο μάθημα
- Προσθήκη/Επεξεργασία του τελικού διαγωνίσματος σε κάποιο μάθημα
- Διαγραφή διαγωνίσματος
- Αποθήκευση περιεχομένου διαγωνίσματος
- Διενέργεια διαγωνίσματος

Τα παραπάνω αποτελούν το σύνολο των διεργασιών που είναι απαραίτητες για την προετοιμασία των ιστοσελίδων JSP οι οποίες σχετίζονται με τα διαγωνίσματα. Ο αντίστοιχος κώδικας του ελεγκτή περιλαμβάνεται στη συνέχεια.

```
@Controller
public class TestController {
	@Autowired
    private TestService testService;

	@Autowired
    private CourseService courseService;

	@Autowired
    private SectionService sectionService;

	@Autowired
    private UserService userService;

	@Autowired
    private QuestionService questionService;

	@Autowired
    private AnswerService answerService;

    @RequestMapping(value = {"/addCourseTest"}, params = {"addTestCourseId",
"addTestSectionId"}, method = RequestMethod.POST)
    public String addCourseTest(Model model, @RequestParam(value = "addTestCourseId")
long courseId,
		@RequestParam(value = "addTestSectionId") long sectionId) {
        Course course = courseService.findCourseById(courseId);
        Section section = sectionService.findById(sectionId);
        int testNumber = 1;
        try {
		testNumber = testService.getNextTestNumber(courseId, sectionId);
        } catch (Exception e) {
        }
        Test test = testService.addCourseTest(new Test(course, section, testNumber));
		List<Question> questions = questionService.findByTestId(test.getId());
        if (questions != null)
		Collections.sort(questions);
        model.addAttribute("test", test);
        model.addAttribute("course", course);
        model.addAttribute("section", section);
        model.addAttribute("questions", questions);
        return "testContent";
    }

    @RequestMapping(value = {"/editMainCourseTest"}, params = {"editMainTestCourseId"},
method = RequestMethod.POST)
    public String editMainCourseTest(Model model, @RequestParam(value =
"editMainTestCourseId") long courseId) {
        Course course = courseService.findCourseById(courseId);
        int testNumber = 1;
        Test test = testService.getCourseFinalTest(courseId);
        if (test == null)
		test = testService.addCourseTest(new Test(course, testNumber));
		List<Question> questions = questionService.findByTestId(test.getId());
        if (questions != null)
		Collections.sort(questions);
        model.addAttribute("test", test);
        model.addAttribute("course", course);
        model.addAttribute("section", null);
        model.addAttribute("questions", questions);
        return "testContent";
```

```
    }

    @RequestMapping(value = {"/editCourseTest"}, params = {"editTestId"}, method =
RequestMethod.POST)
    public String editCourseTest(Model model, @RequestParam(value = "editTestId") long
testId) {
        Test test = testService.findById(testId);
        Course course = test.getCourse();
        Section section = test.getSection();
                List<Question> questions = questionService.findByTestId(test.getId());
        if (questions != null)
                Collections.sort(questions);
        model.addAttribute("test", test);
        model.addAttribute("course", course);
        model.addAttribute("section", section);
        model.addAttribute("questions", questions);
        return "testContent";
    }

    @RequestMapping(value = {"/saveTestContent"}, params = {"courseId", "sectionId",
"testId", "questionId",
                                "question", "answer1", "answer2", "answer3", "answer4",
"correctAnswer", "newQuestion"}, method = RequestMethod.POST)
    public String saveTestContent(Model model, @RequestParam(value = "courseId") long
courseId,
                @RequestParam(value = "sectionId") Long sectionId,
                @RequestParam(value = "testId") long testId,
                @RequestParam(value = "questionId") int questionIds[],
                @RequestParam(value = "question") String questionTexts[],
                @RequestParam(value = "answer1") String answers1[],
                @RequestParam(value = "answer2") String answers2[],
                @RequestParam(value = "answer3") String answers3[],
                @RequestParam(value = "answer4") String answers4[],
                @RequestParam(value = "correctAnswer") int correctAnswer[],
                @RequestParam(value = "newQuestion") int newQuestion) {

        for (int id = 0; id < questionIds.length; id++) {
                questionService.update(questionIds[id], questionTexts[id], answers1[id],
answers2[id], answers3[id], answers4[id], correctAnswer[id]);
        }
        if (newQuestion == 1) {
                int questionNumber = 1;
                try {
                        questionNumber = questionService.getNextQuestionNumber(testId);
                } catch (Exception e) {
                }
                questionService.save(new Question(questionNumber,
testService.findById(testId)));
        }

        Course course = courseService.findCourseById(courseId);
        Section section = null;
        if (sectionId != null)
                section = sectionService.findById(sectionId);
        Test test = testService.findById(testId);
                List<Question> questions = questionService.findByTestId(testId);
        if (questions != null)
                Collections.sort(questions);
        model.addAttribute("test", test);
        model.addAttribute("course", course);
        model.addAttribute("section", section);
        model.addAttribute("questions", questions);
```

```
            return "testContent";
        }

        @RequestMapping(value = {"/saveEmptyTestContent"}, params = {"courseId",
    "sectionId", "testId", "newQuestion"},
                    method = RequestMethod.POST)
            public String saveEmptyTestContent(Model model, @RequestParam(value =
    "courseId") long courseId,
                    @RequestParam(value = "sectionId") Long sectionId,
                    @RequestParam(value = "testId") long testId,
                    @RequestParam(value = "newQuestion") int newQuestion) {

                    if (newQuestion == 1) {
                            int questionNumber = 1;
                            try {
                                    questionNumber =
    questionService.getNextQuestionNumber(testId);
                            } catch (Exception e) {
                            }
                            questionService.save(new Question(questionNumber,
    testService.findById(testId)));
                    }

                    Course course = courseService.findCourseById(courseId);
                    Section section = null;
                    if (sectionId != null)
                            section = sectionService.findById(sectionId);
                    Test test = testService.findById(testId);
                    List<Question> questions = questionService.findByTestId(testId);
            if (questions != null)
                    Collections.sort(questions);
                    model.addAttribute("test", test);
                    model.addAttribute("course", course);
                    model.addAttribute("section", section);
            model.addAttribute("questions", questions);
                    return "testContent";
            }

        // Methods for tests
        @RequestMapping(value = {"/doTest"}, params = {"courseId", "testId", "prevQuestion",
    "question", "timeRemaining"}, method = RequestMethod.POST)
        public String doTest(Model model, @RequestParam(value = "courseId") long courseId,
                    @RequestParam(value = "testId") long testId,
                    @RequestParam(value = "prevQuestion") int prevQuestion,
                    @RequestParam(value = "question") int question,
                    @RequestParam(value = "timeRemaining") double timeRemaining) {
            Course c = courseService.findCourseById(courseId);
            List<Test> tests = testService.findAllCourseTests(courseId);
            Test test = testService.findById(testId);
                    List<Question> questions = questionService.findByTestId(testId);

                    Object principal =
    SecurityContextHolder.getContext().getAuthentication().getPrincipal();
            String username = null;
            if (principal instanceof UserDetails) {
              username = ((UserDetails)principal).getUsername();
            } else {
              username = principal.toString();
            }
            User u = userService.findByUsername(username);
                    // Create answer for this new question
```

```
                if (question != -1 && question != -2 &&
!answerService.existsQuestionStudentIdHalfHour(questions.get(question).getId(),
u.getId())) {
                answerService.save(u.getId(), questions.get(question).getId());
                }
                if (prevQuestion != -1) {
                        answerService.update(u.getId(),
questions.get(prevQuestion).getId());
                }

        model.addAttribute("course", c);
        model.addAttribute("tests", tests);

                if (question >= 0 &&
answerService.existsQuestionStudentIdHalfHour(questions.get(question).getId(),
u.getId())) {
                        model.addAttribute("answer",
answerService.getAnswerQuestionStudentIdHalfHour(questions.get(question).getId(),
u.getId()));
                }

        if (question == -2) {
                model.addAttribute("finalTestReady",
userService.studentCompletedAllPrefinalTests(u.getId(), c.getId()));
                model.addAttribute("errorMessage", "");
                try {
                        KMeans kmeans = new KMeans(answerService, courseId);
                        model.addAttribute("finalResults",
kmeans.getResults(answerService, u.getId(), courseId));
                } catch (Exception e) {
                        model.addAttribute("finalResults", 0);
                }
                return "course";
        }
        model.addAttribute("test", test);
        model.addAttribute("question", question);
        model.addAttribute("questions", questions);
        model.addAttribute("timeRemaining", timeRemaining);
        model.addAttribute("correctAnswer", answerService.getCorrectAnswers(u.getId(),
testId));
        return "test";
    }

    @RequestMapping(value = {"/updateAnswer"}, params = {"question", "answer"}, method =
RequestMethod.GET)
    public String updateAnswer(Model model, @RequestParam(value = "question") long
questionId,
                @RequestParam(value = "answer") short answerNumber) {
                Object principal =
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        String username = null;
        if (principal instanceof UserDetails) {
          username = ((UserDetails)principal).getUsername();
        } else {
          username = principal.toString();
        }
        answerService.update(userService.findByUsername(username).getId(), questionId,
answerNumber);

        return "ok";
    }
```

```
    @RequestMapping(value = {"/removeCourseTest"}, params = {"removeTestCourseId",
"removeTestId"},
                method = RequestMethod.POST)
    public String removeCourseTest(Model model, @RequestParam(value =
"removeTestCourseId") long courseId,
                @RequestParam(value = "removeTestId") long testId) {
        testService.delete(testId);
        Course course = courseService.findCourseById(courseId);
        model.addAttribute("course", course);

                return "courseContent";
    }
}
```

4.7. Υλοποίηση αλγορίθμων μηχανιστικής μάθησης (machine learning)

Σε αυτό το τμήμα της εργασίας ασχολούμαστε με την καρδιά του προβλήματος που καλούμαστε να επιλύσουμε και αφορά τη χρήση των αλγορίθμων μηχανιστικής μάθησης με σκοπό την πρόβλεψη και βελτίωση της επίδοσης του μαθητή σε κάποιο τελικό διαγώνισμα κάποιου μαθήματος. Στα πλαίσια της παρούσας εργασίας έχει χρησιμοποιηθεί μόνο το αλγόριθμός K-means τον οποίο περιγράφουμε ενδελεχώς στο σημείο αυτό και τα αποτελέσματα του στο επόμενο κεφάλαιο.

4.7.1. K-means algorithm

Όπως έχουμε περιγράψει και παραπάνω, ο αλγόριθμος K-means στηρίζεται στον εντοπισμό μιας σειράς από παραμέτρους τις οποίες αναθέτουμε σε έναν ν-διάστατο χώρο και τις οποίες βαθμονομούμε καταλλήλως ώστε να είναι συγκρίσιμες μεταξύ τους. Ακολούθως, με βάση κάποια κεντρικά σημεία, τα οποία αρχικά είναι αυθαίρετα αλλά μετά υπολογίζονται επακριβώς, υπολογίζουμε τις αποστάσεις κάθε σημείου από αυτά. Κάθε σημείο ανατίθεται στο κοντινότερό του κεντρικό σημείο.

Στην περίπτωσή μας έχουμε επιλέξει μια σειρά από μεταβλητές οι οποίες σχετίζονται με την απάντηση ρου μαθητή σε κάθε ερώτημα. Έτσι για κάθε μαθητή που έχει συμπληρώσει πλήρως όλα τα τεστ (συμπεριλαμβανομένου του τελικού) για κάποιο μάθημα παίρνουμε όλες τις τελευταίες ερωτήσεις που έχει δώσει αυτός. Για κάθε συγκεκριμένη απάντηση λαμβάνουμε τις εξής πληροφορίες:

- Αν έχει κάνει σωστή επιλογή απάντησης
- Πόσες απαντήσεις έχει δώσει σε αυτή την ερώτηση προτού να καταλήξει στην τελική
- Πόσο χρόνο χρειάστηκε για να απαντήσει σε μια ερώτηση

Λαμβάνοντας τις απαντήσεις στα παραπάνω για όλες τις απαντήσεις δημιουργούμε για κάθε σημείο του αλγορίθμου 3p (τριπλάσια των σημείων που έχουμε) χαρακτηριστικά, τα οποία και τοποθετούνται στον αντίστοιχο ν-διάστατο χώρο. Εφόσον τοποθετήσουμε και τα κεντρικά συστήματα θα μπορούμε να υπολογίσουμε από ποιο κεντρικό σημείο απέχει λιγότερο το κάθε σημείο και να το αναθέσουμε σε αυτό. Ακολούθως, και εφόσον ολοκληρωθεί η διαδικασία για έναν πλήρη γύρο, επανυπολογίζεται η θέση των κεντρικών σημείων, υπολογίζοντας τη μέση τιμή για κάθε παράμετρο για όλα τα σημεία τα οποία ανήκουν σε κάποιο segment. Αφού το υλοποιήσουμε και αυτό ξεκινάμε πάλι από το προηγούμενο βήμα για να μπορέσουμε να επαναλάβουμε τη διαδικασία όσες φορές χρειαστεί ώστε να μην έχουμε καμία μετάπτωση σημείου σε άλλο κέντρο.

Για αποφυγή προβλημάτων στο σημείο αυτό έχουμε βάλει έναν έλεγχο ώστε το πλήθος των επαναλήψεων να μην υπερβαίνει ποτέ τις 1,000. Έτσι πετυχαίνουμε να υπολογίσουμε με αρκετά μεγάλη ακρίβεια το αποτέλεσμα που θα φέρει ο κάθε μαθητής στο τελικό διαγώνισμα και να τον βοηθούμε πιθανώς να βελτιωθεί από πριν σε αυτό.

Στη συνέχεια ακολουθεί ο κώδικας για το σημείο αυτό.

```
public class Point {
	long cusId;
	int cluster;
	HashMap<String, Double> values;
}
public class KMeans {

	final static int clusters = 5;
	final static int epochs = 1000;

	private Centroid[] centroids;
	private List<Point> points;

	public KMeans(AnswerService answerService, long courseId) {
		points = new ArrayList<Point>();
		int totalColumns = answerService.getNumberOfAnswersForCourse(courseId);
		List<String> answersString = answerService.getAnswersForCourse(courseId);

		for (int i = 0; i < answersString.size(); i += totalColumns/3) {
			String[] answers = answersString.get(i).split(";");
			long key = Long.parseLong(answers[0]);
			HashMap<String, Double> values = new HashMap<String, Double>();
			for (int j = 0; j < totalColumns/3; j++) {
				String question = answers[1];
				values.put(question + "a",
Double.parseDouble(answers[2]));
				values.put(question + "b",
Double.parseDouble(answers[3]));
				values.put(question + "c",
Double.parseDouble(answers[4]));
			}
			points.add(new Point(key, values));
		}

		centroids = new Centroid[clusters];
		Random r = new Random();
		for (int i = 0; i < centroids.length; i++) {
			HashMap<String, Double> values = new HashMap<String, Double>();
			Point p = points.get(r.nextInt() % points.size());
			for (String key : p.getValues().keySet())
				values.put(key, p.getValues().get(key));
			centroids[i] = new Centroid(i, values);
		}

		for (int i = 0; i < epochs; i++) {
			for (Point p : points) {
				double min = p.calculateDistance(centroids[0]);
				p.setCluster(0);
				for (int j = 1; j < centroids.length; j++) {
					if (p.calculateDistance(centroids[j]) < min) {
						min = p.calculateDistance(centroids[j]);
						p.setCluster(j);
					}
```

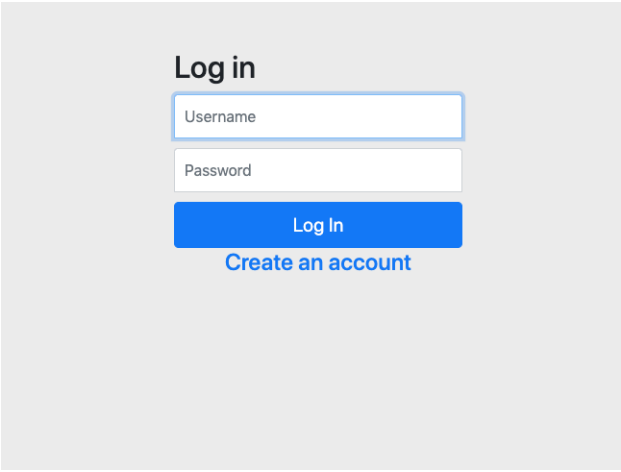
```
                                }
                        }

                        for (int j = 0; j < centroids.length; j++) {
                                int pointsNumber = 0;
                                for (String key : centroids[j].getValues().keySet())
                                        centroids[j].getValues().put(key, 0.0);
                                for (Point p : points) {
                                        if (p.getCluster() == j) {
                                                for (String key :
centroids[j].getValues().keySet())
                                                        centroids[j].getValues().put(key,
centroids[j].getValues().get(key) + p.getValues().get(key));
                                                pointsNumber++;
                                        }
                                }
                                for (String key : centroids[j].getValues().keySet())
                                        if (pointsNumber != 0)
                                                centroids[j].getValues().put(key,
centroids[j].getValues().get(key) / pointsNumber);
                        }
                }
        }

        public int getResults(AnswerService answerService, long cusId, long courseId) {
                Point point = null;
                for (Point p : points)
                        if (p.getCusId() == cusId) {
                                point = p;
                                break;
                        }

                List<Long> checkPoints = new ArrayList<Long>();
                for (Point p : points)
                        if (p.getCluster() == point.getCluster()) {
                                checkPoints.add(p.getCusId());
                        }

                return answerService.getFinalResult(courseId, checkPoints);
        }
}
```

5. Περιήγηση στην εφαρμογή

Σε αυτό το τμήμα κάνουμε μια γρήγορη περιήγηση της εφαρμογής από την πλευρά του χρήστη, για όλους τους πιθανούς ρόλους που υπάρχουν με βάση τον ορισμό της εφαρμογής (διαχειριστής, καθηγητής, μαθητής).

5.1. Είσοδος

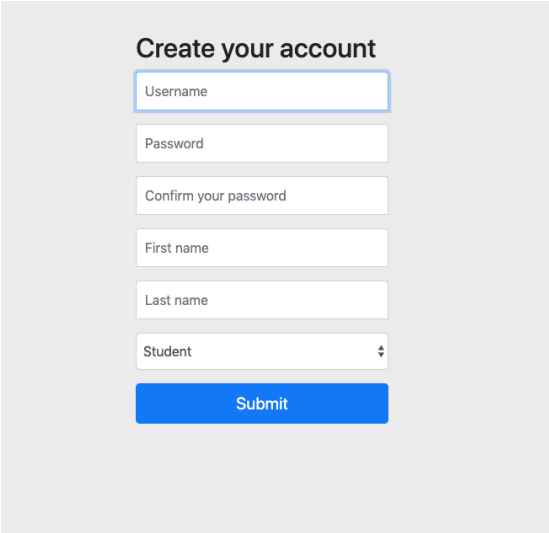
Η πρώτη επαφή οποιουδήποτε χρήστη με την εφαρμογή έχει να κάνει με τη διαδικασία εγγραφής και εισόδου στην εφαρμογή. Η είσοδος γίνεται και για τις τρεις περιπτώσεις από ένα απλό περιβάλλον όπως αυτό φαίνεται στην Εικόνα 12.



Εικόνα 12: Σελίδα εισόδου

5.2. Δημιουργία λογαριασμού

Μία ακόμα σελίδα η οποία είναι κοινή για όλους του χρήστες της εφαρμογής είναι η σελίδα της εγγραφής. Εδώ, εφόσον ο πελάτης βάλει τα προσωπικά του στοιχεία επιλέγει ποιον από τους τρεις προαναφερθέντες ρόλους έχει και επιλέγει αντίστοιχα. Η σελίδα που εμφανίζεται φαίνεται στην Εικόνα 13.



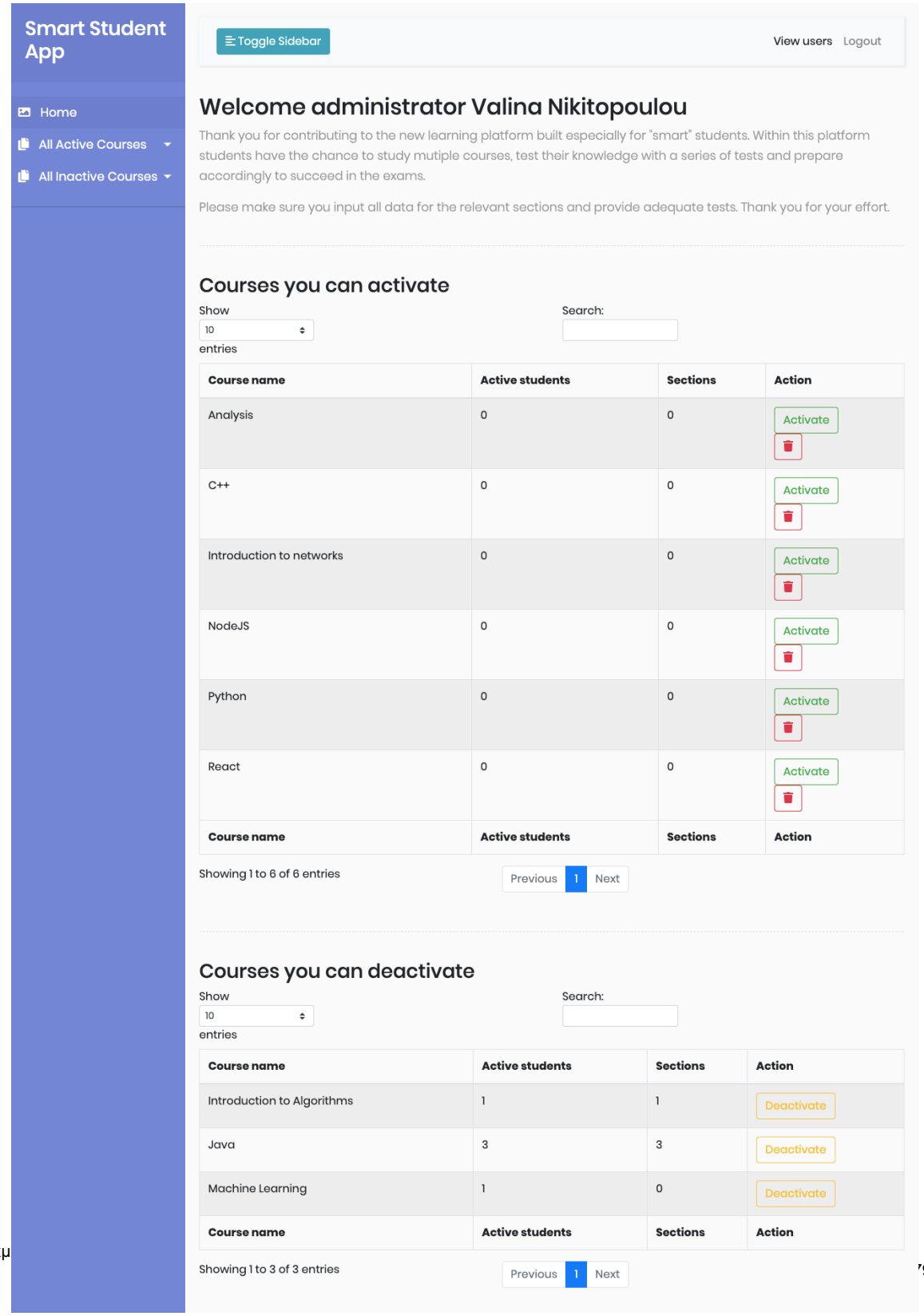
Εικόνα 13: Σελίδα εγγραφής χρήστη

5.3. Ρόλος διαχειριστή (administrator)

Ο πρώτος από τους ρόλους που αναλύουμε είναι αυτός του διαχειριστή. Ο διαχειριστής έχει τη δυνατότητα να βλέπει όλα τα μαθήματα, τα περιεχόμενα τους και τα διαγωνίσματά τους και να κάνει αλλαγές πάνω σε αυτά και, επίσης, να βλέπει όλους τους χρήστες και επίσης να κάνει αλλαγές στα στοιχεία τους.

5.3.1. Αρχική σελίδα

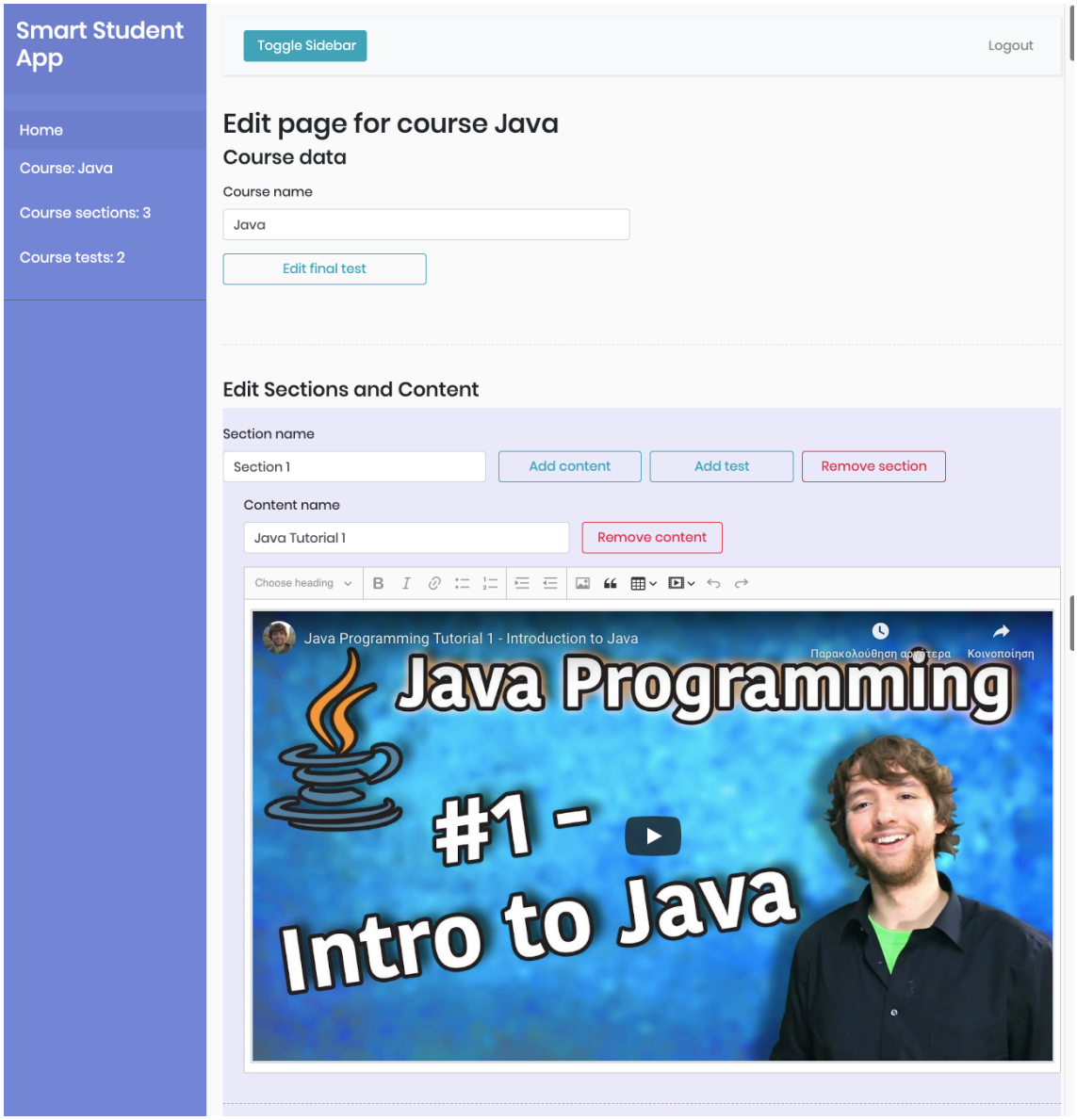
Η πρώτη σελίδα που έχει πρόσβαση ο διαχειριστής μετά την είσοδό του στην εφαρμογή είναι η σελίδα διαχείρισης των μαθημάτων. Εδώ μπορεί να ενεργοποιήσει ή να απενεργοποιήσει οποιοδήποτε μάθημα επιθυμεί. Επίσης μπορεί να επιλέξει να δει λεπτομέρειες για οποιοδήποτε μάθημα και να τις τροποποιήσει αναλόγως. Το περιβάλλον φαίνεται στην Εικόνα 14.


Smart Student App
Home
All Active Courses
All Inactive Courses
Toggle Sidebar
View users
Logout
Welcome administrator Valina Nikitopoulou
Thank you for contributing to the new learning platform built especially for "smart" students. Within this platform students have the chance to study mutiple courses, test their knowledge with a series of tests and prepare accordingly to succeed in the exams.
Please make sure you input all data for the relevant sections and provide adequate tests. Thank you for your effort.
Courses you can activate
Show 10 entries
Search:
Course name | Active students | Sections | Action
Analysis | 0 | 0 | Activate
C++ | 0 | 0 | Activate
Introduction to networks | 0 | 0 | Activate
NodeJS | 0 | 0 | Activate
Python | 0 | 0 | Activate
React | 0 | 0 | Activate
Showing 1 to 6 of 6 entries
Previous 1 Next
Courses you can deactivate
Show 10 entries
Search:
Course name | Active students | Sections | Action
Introduction to Algorithms | 1 | 1 | Deactivate
Java | 3 | 3 | Deactivate
Machine Learning | 1 | 0 | Deactivate
Showing 1 to 3 of 3 entries
Previous 1 Next

Εικόνα 14: Σελίδα εισόδου διαχειριστή

5.3.2. Σελίδα διαχείρισης περιεχομένου μαθήματος

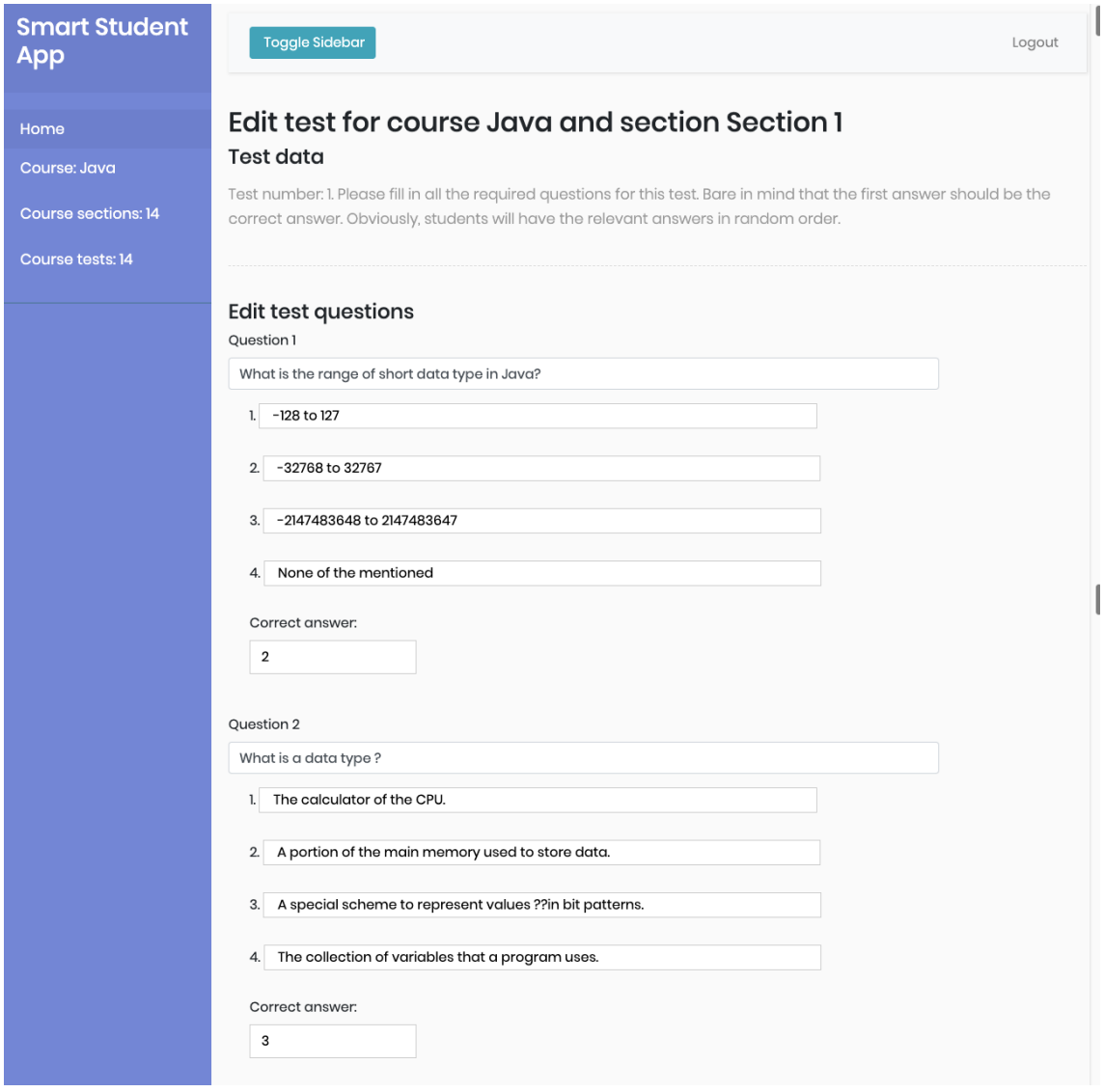
Όπως ήδη αναφέραμε, ο διαχειριστής μπορεί να αλλάξει το περιεχόμενο ενός μαθήματος. Αυτό μπορεί να το κάνει μέσα από την σελίδα όπως αυτή παρουσιάζεται στην Εικόνα 15.



Εικόνα 15: Σελίδα διαχείρισης περιεχομένου

5.3.3. Σελίδα διαχείρισης διαγωνισμάτων μαθήματος

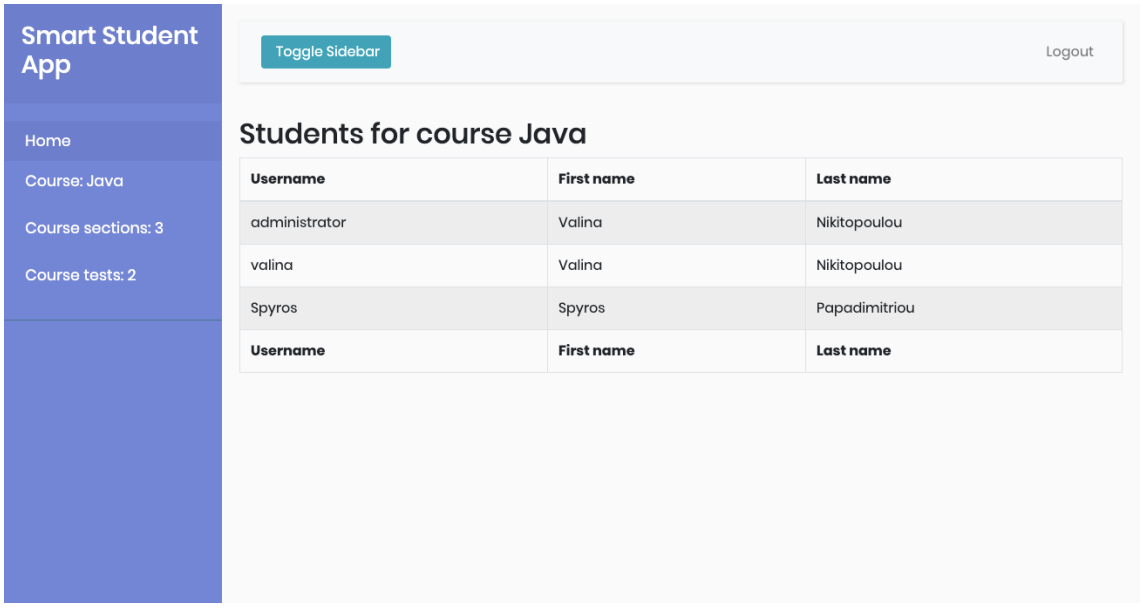
Το ίδιο συμβαίνει και στην περίπτωση των διαγωνισμάτων οποιουδήποτε μαθήματος, όπως φαίνεται στην Εικόνα 16.



Εικόνα 16: Σελίδα διαχείρισης διαγωνισμάτων

5.3.4. Σελίδα διαχείρισης χρηστών

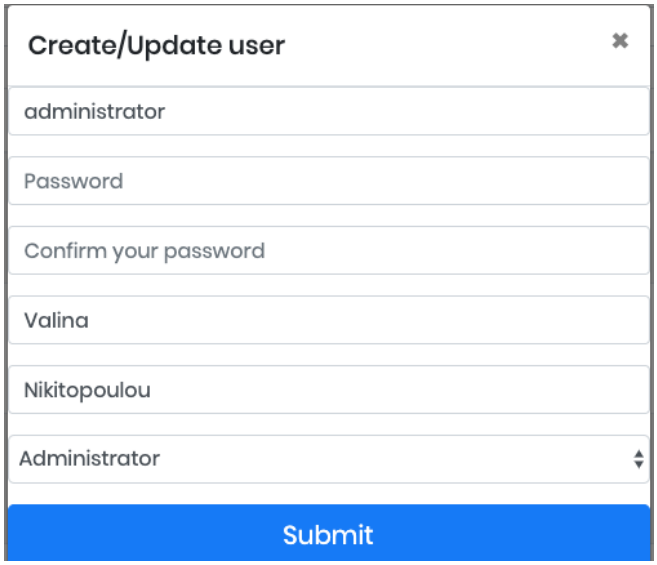
Και οι χρήστες μπορούν να διαχειριστούν από τον διαχειριστή της εφαρμογής. Η σχετική σελίδα φαίνεται στην Εικόνα 17.



Εικόνα 17: Εμφάνιση χρηστών

5.3.5. Δημιουργία/Αλλαγή χρηστών

Επίσης ο διαχειριστής έχει τη δυνατότητα είτε να προσθέσει καινούριους χρήστες, είτε να αλλάξει τα δεδομένα τους, όπως φαίνεται στην Εικόνα 18

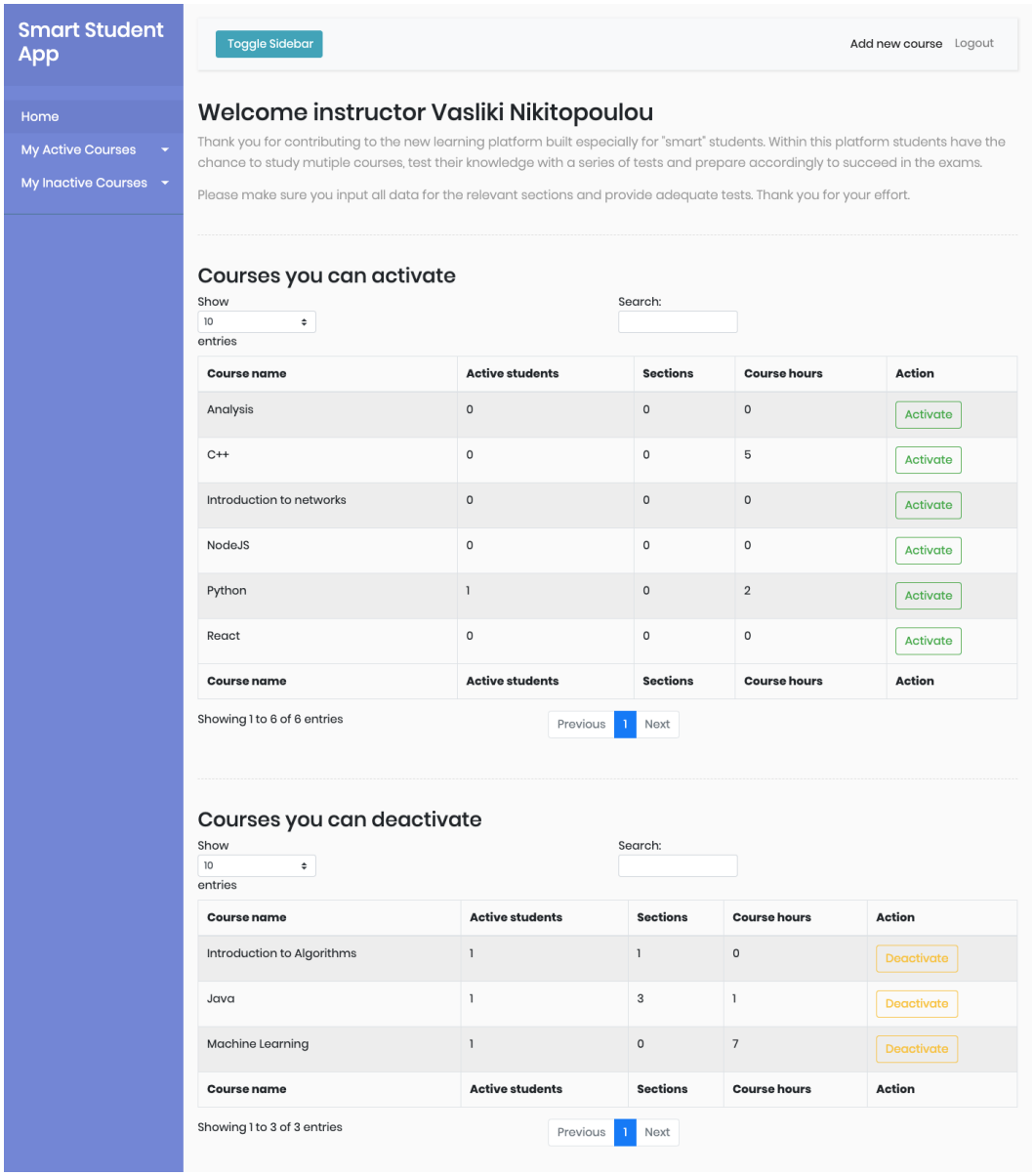


Εικόνα 18: Δημιουργία/Διαχείριση χρηστών

5.4. Ρόλος καθηγητή

Έχοντας ολοκληρώσει το ρόλο του διαχειριστή, περνάμε στο ρόλο του καθηγητή, ο οποίος έχει τις ίδιες δυνατότητες με τον διαχειριστή στα μαθήματα του μόνο, αλλά δεν έχει δικαιοδοσία στους λογαριασμούς των υπόλοιπων χρηστών.

5.4.1. Αρχική σελίδα

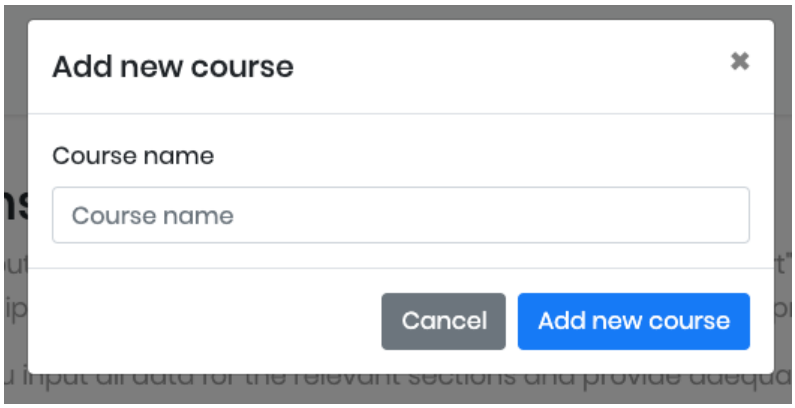


Εικόνα 19: Αρχική σελίδα καθηγητή

Και στην περίπτωση του καθηγητή, η πρώτη σελίδα που βλέπει με την είσοδό του στην εφαρμογή είναι η σελίδα διαχείρισης των μαθημάτων του, όπου μπορεί να ενεργοποιήσει, απενεργοποιήσει ή προσθέσει κάποιο μάθημα. Με κατάλληλους συνδέσμους μπορεί να δει τους μαθητές κάποιου μαθήματος, την πρόοδο τους και τα υπόλοιπα στοιχεία του μαθήματος. Η σχετική σελίδα φαίνεται στην Εικόνα 19.

5.4.2. Προσθαφαίρεση μαθημάτων

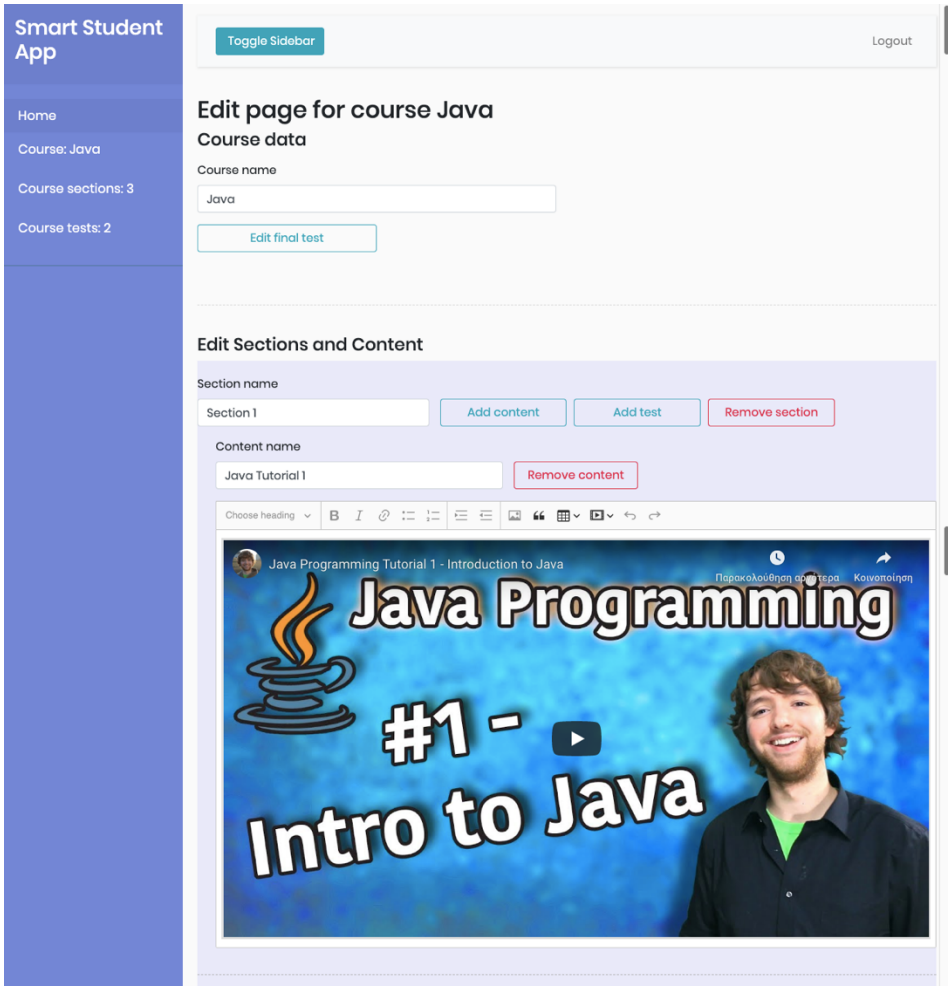
Εδώ βλέπουμε τη σελίδα μέσω της οποίας ο καθηγητής μπορεί να προσθέσει κάποιο μάθημα στην Εικόνα 20.



Εικόνα 20: Προσθήκη μαθήματος

5.4.3. Προσθαφαίρεση υλικού μαθημάτων

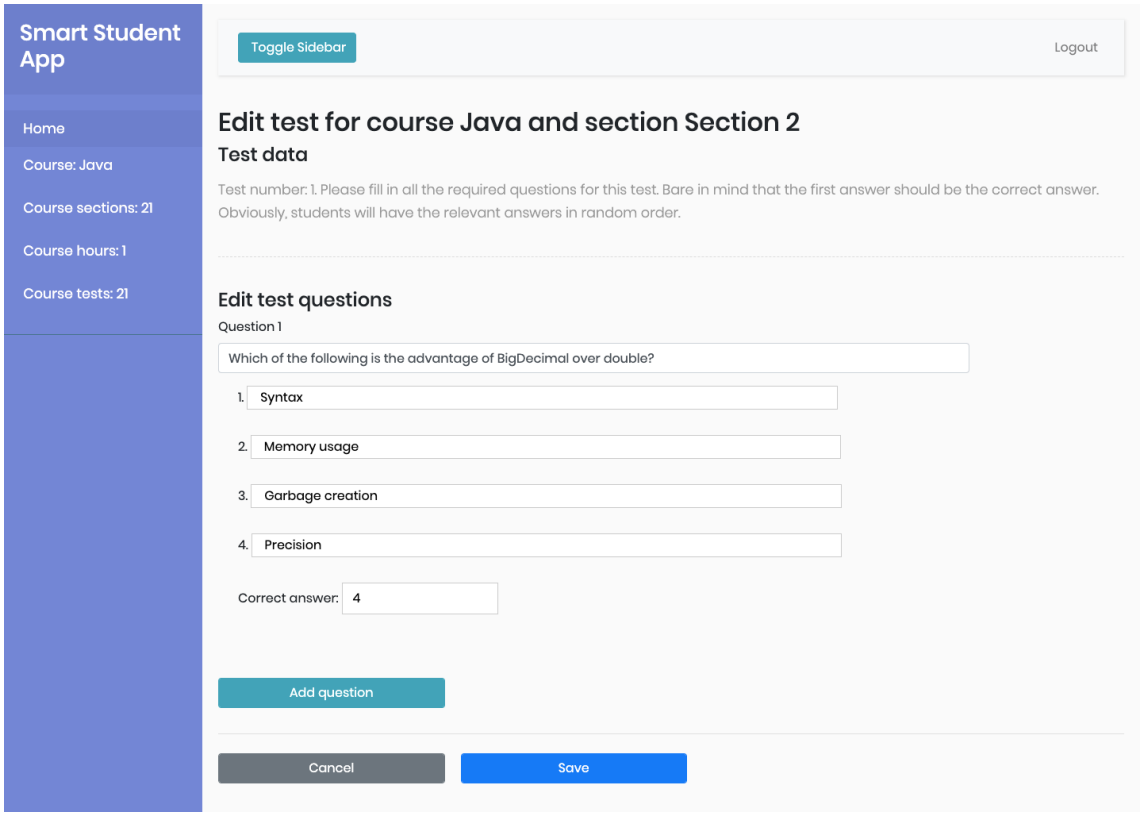
Ο καθηγητής έχει επίσης τη δυνατότητα προσθαφαίρεσης υλικού από τα μαθήματά του, όπως φαίνεται στην Εικόνα 21.



Εικόνα 21: Προσθαφαίρεση υλικού μαθημάτων

5.4.4. Προσθαφαίρεση διαγωνισμάτων

Τέλος, ο καθηγητής μπορεί να προσθέτει, να αφαιρεί και να αλλάζει τα περιεχόμενα των διαγωνισμάτων των δικών του μαθημάτων. Σχετική σελίδα φαίνεται στην Εικόνα 22.



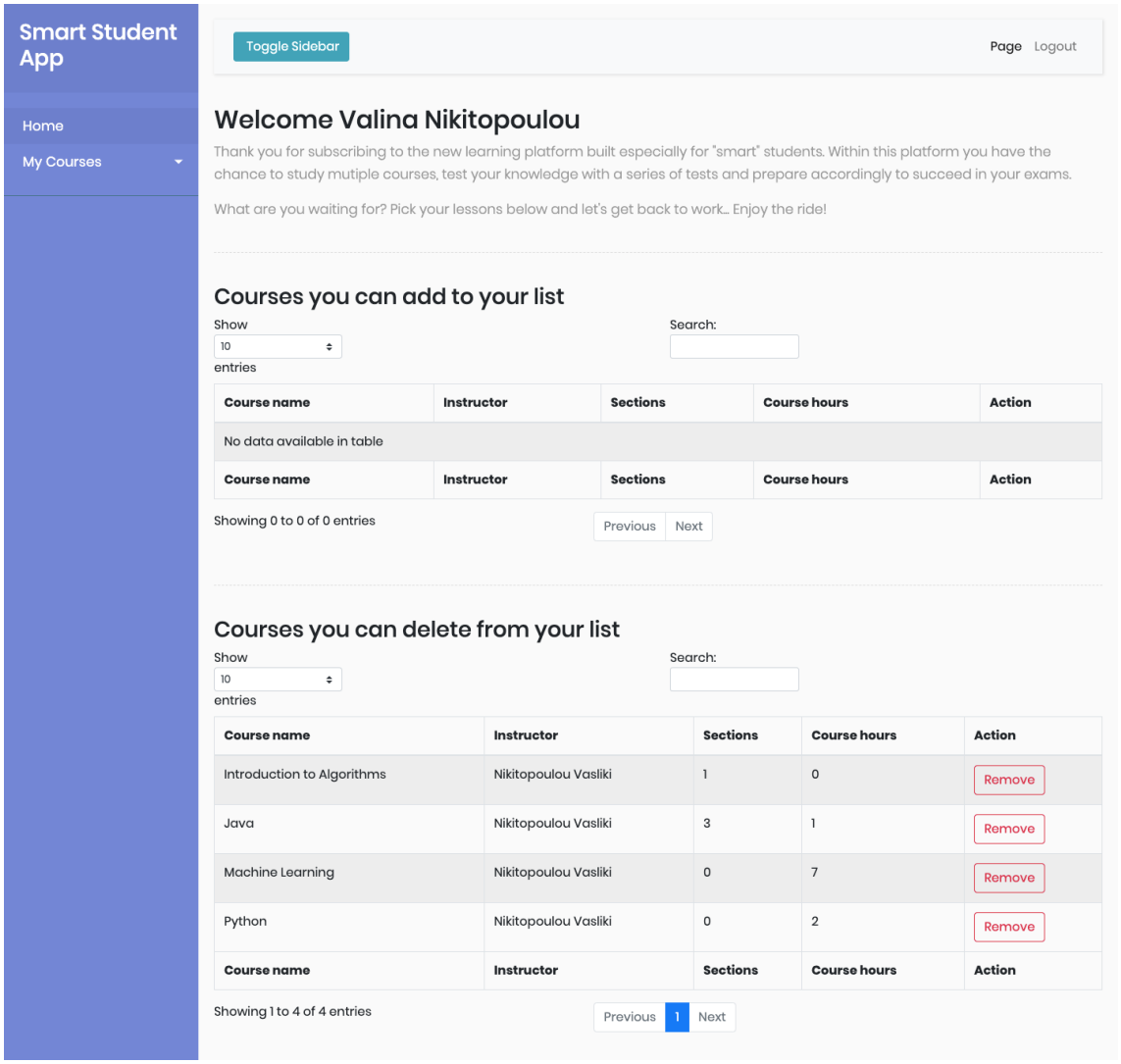
Εικόνα 22: Προσθαφαίρεση διαγωνισμάτων

5.5. Ρόλος μαθητή

Ο τελευταίος ρόλος που υπάρχει στην εφαρμογή μας είναι ο ρόλος του μαθητή. Ο μαθητής έχει τη δυνατότητα προσθαφαίρεσης μαθημάτων από το λογαριασμό του, παρακολούθησης του υλικού των μαθημάτων, ολοκλήρωσης διαγωνισμάτων και παρακολούθησης της επίδοσής του.

5.5.1. Αρχική σελίδα (Προσθαφαίρεση μαθημάτων)

Η πρώτη σελίδα του μαθητή του δίνει τη δυνατότητα να προσθέτει και να αφαιρεί μαθήματα από το λογαριασμό του για να μπορεί να τα παρακολουθήσει, όπως φαίνεται στην Εικόνα 23.



Εικόνα 23: Σελίδα εισόδου μαθητή

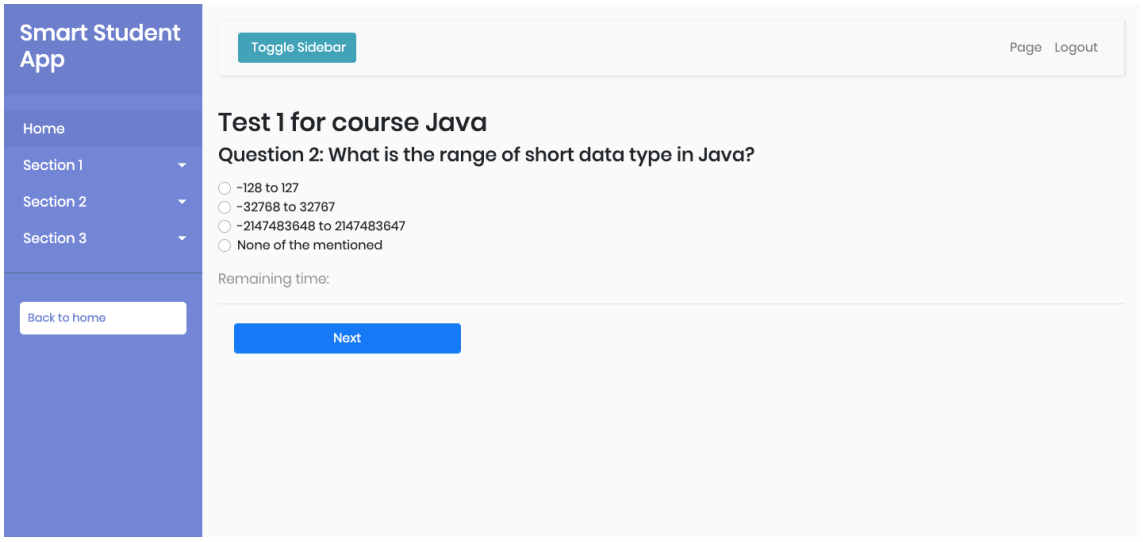
5.5.2. Παρακολούθηση υλικού

Όπως αναφέραμε και προηγουμένως, ο μαθητής μπορεί προφανώς να παρακολουθήσει τα περιεχόμενα του μαθήματος, όπως φαίνεται στην Εικόνα 24.

Εικόνα 24: Σελίδα μαθήματος μαθητή

5.5.3. Διενέργεια διαγωνισμάτων

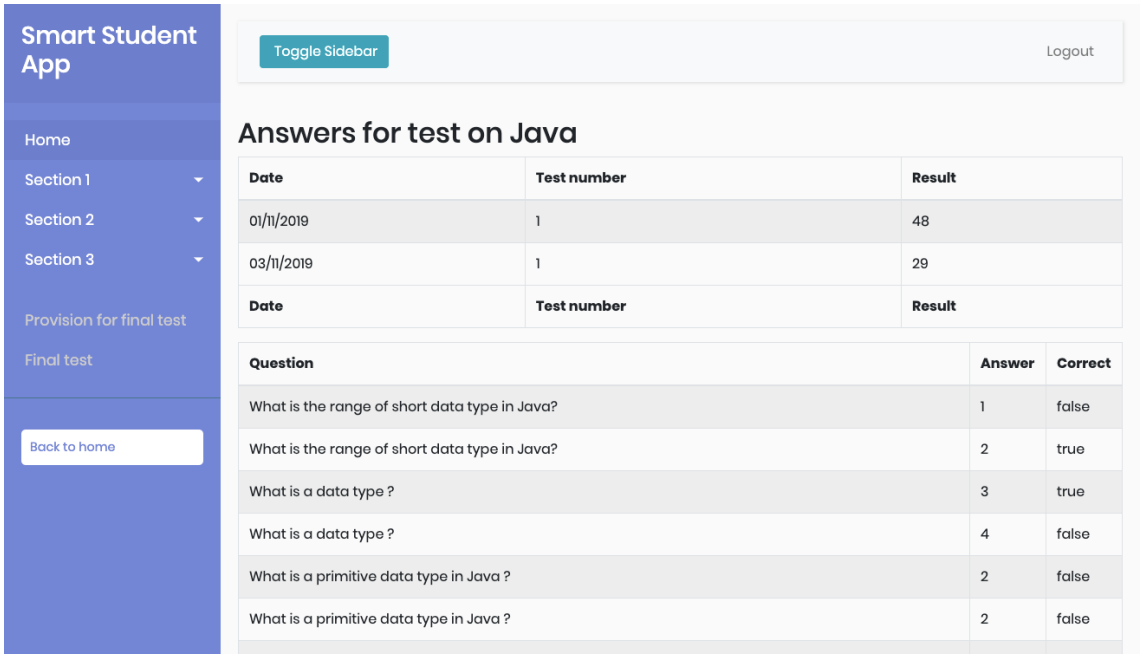
Επίσης ο μαθητής μπορεί να εκτελεί τα διαγωνίσματα των μαθημάτων που έχει επιλέξει, όπως φαίνεται στην Εικόνα 25.



Εικόνα 25: Σελίδα διαγωνίσματος μαθητή

5.5.4. Ολοκλήρωση διαγωνισμάτων & αποτελέσματα

Τέλος, ο μαθητής έχει τη δυνατότητα να δει τα αποτελέσματα των διαγωνισμάτων που έχει δώσει διαχρονικά. Σχετική σελίδα φαίνεται στην Εικόνα 26.



Εικόνα 26: Σελίδα αποτελεσμάτων επίδοσης μαθητή

6. Αποτελέσματα

6.1. Συμπεράσματα

Μέσα από την παρούσα εργασία καθίσταται σαφές ότι η μηχανιστική μάθηση μπορεί να τεθεί στην υπηρεσία της πραγματικής μάθησης και να βοηθήσει τους μαθητές να εντρυφήσουν περισσότερο και με μεγαλύτερη ακρίβεια στα σημεία εκείνα που έχουν μεγαλύτερα προβλήματα σε κάποιο μάθημα.

Αυτό βεβαίως αποτελεί μόνο μία από τις πολλές εφαρμογές που μπορεί να έχει η μηχανιστική μάθηση στο συγκεκριμένο πεδίο. Προφανώς, η μηχανιστική μάθηση θα μπορούσε να εφαρμοστεί περισσότερο και στα πλαίσια της καθαρής διδασκαλίας αρκετών μαθημάτων για να βοηθήσει τόσο τον εκπαιδευτή όσο και τον εκπαιδευόμενο να βελτιστοποιήσουν την απόδοση και την επίδοσή τους. Στην παρούσα εφαρμογή, μέσα από την πρόβλεψη των αποτελεσμάτων που μπορεί να φέρει ο κάθε εκπαιδευόμενος και τις προτάσεις που κάνει για περαιτέρω εκμάθηση, δείχνει ότι μπορεί να βοηθήσει και να κατευθύνει τον εκπαιδευόμενο, ώστε να καλύψει όσο το δυνατόν καλύτερα τις εκπαιδευτικές του αδυναμίες.

6.2. Μελλοντική εργασία και επεκτάσεις

Μια προφανής επέκταση της παρούσας εφαρμογής είναι να χρησιμοποιηθούν και επιπλέον αλγόριθμοι για την πρόβλεψη της επίδοσης των μαθητών, όπως ο αλγόριθμος Gaussian Mixture Model και ο αλγόριθμος που ακολουθεί το Agglomerative Hierarchical Clustering. Κάποιοι από αυτούς τους αλγορίθμους μάλιστα μπορεί να έχουν ακόμα καλύτερα αποτελέσματα σε σχέση με τον χρησιμοποιούμενο K-means αλγόριθμο. Επίσης μια άλλη δυνατότητα επέκτασης είναι να εφαρμοστεί η μηχανιστική μάθηση σε άλλες εκφάνσεις και διαδικασίες της μαθησιακής διαδικασίας. Σε κάθε περίπτωση, ο τομέας της παιδείας είναι ένας πρόσφορος τομέας στον οποίο θα μπορούσε να εφαρμοστεί εκτενώς η ιδέα της μηχανιστικής μάθησης με πολύ καλά αποτελέσματα.

Πίνακας εικόνων

Βιβλιογραφία

1. https://hellokoding.com/registration-and-login-example-with-spring-xml-configuration-maven-jsp-and-mysql/
2. https://www.sanfoundry.com/java-questions-answers-data-type-date-timezone/
3. https://bootstrapious.com/p/bootstrap-sidebar
4. https://getbootstrap.com/docs/4.0/components/buttons/
5. https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68
6. https://apothesis.lib.teicrete.gr/bitstream/handle/11713/8194/MavroforakisDimitris2017.pdf?sequence=1&isAllowed=y
7. https://el.wikipedia.org/wiki/Java
8. https://el.wikipedia.org/wiki/Model-view-controller
9. https://el.wikipedia.org/wiki/Hibernate_Framework
10. https://el.wikipedia.org/wiki/HTML
11. https://el.wikipedia.org/wiki/CSS
12. https://el.wikipedia.org/wiki/JavaScript
13. https://ckeditor.com/
14. https://start.spring.io/
15. https://www.jee.gr/spring-framework-series-%CE%BC%CE%AD%CF%81%CE%BF%CF%82-1%CE%BF/
16. https://repository.kallipos.gr/bitstream/11419/3977/1/01_chapter_8.pdf
17. https://repository.kallipos.gr/bitstream/11419/2972/1/02_chapter_06.pdf