ΣΧΟΛΗ ΟΙΚΟΝΟΜΙΚΩΝ ΕΠΙΣΤΗΜΩΝ ΚΑΙ ΔΙΟΙΚΗΣΗΣ ΕΠΙΧΕΙΡΗΣΕΩΝ

ΤΜΗΜΑ ΔΙΟΙΚΗΤΙΚΗΣ ΕΠΙΣΤΗΜΗΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΑΥΤΟΜΑΤΟΠΟΙΗΜΕΝΗ ΑΝΑΘΕΣΗ ΠΤΥΧΙΑΚΩΝ ΕΡΓΑΣΙΩΝ

Σπουδαστής: ΑΛΙΚΑΙ ΑΛΜΠΙ

Επιβλέπων Καθηγητής: ΣΤΑΜΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

ΠΑΤΡΑ, 2021

Ευχαριστίες

Μέσω του παρόντος θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή κ. Στάμο Κωνσταντίνο για την πολύτιμη βοήθειά, την υπομονή του αλλά κυρίως για την ευκαιρία που μου έδωσε να αναπτύξω μια εφαρμογή που στάθηκε αφορμή να μελετήσω και να εμβαθύνω σε ένα ενδιαφέρον γνωστικό αντικείμενο στον τομέα της πληροφορικής. Επίσης θα ήθελα να ευχαρηστήσω και την οικογένεια μου για την υποστήριξη που μου παρείχε όλα αυτά τα χρόνια, απαραίτητη για να ολοκληρώσω τις σπουδές μου.

Πρόλογος

Στόχος της πτυχιακής εργασίας είναι η ανάπτυξη μιας διαδικτυακής (web) εφαρμογής μέσω της οποίας θα πραγματοποιείται η ανάθεση των πτυχιακών εργασιών. Θεωρήσαμε ενδιαφέρον το θέμα της εργασίας μιας και θα μπορούσε να αποτελεί ενα χρήσιμο εργαλείο και να έχει άμεση εφαρμογή στην διαδικασία ανάθεσης των εργασιών του τμήματος μας.

Μέσω της πλατφόρμας αυτής οι σπουδαστές θα εισέρχονται, θα διαμορφώνουν ομάδες, θα επιλέγουν τα θέματα που τους ενδιαφέρουν και στη συνέχεια σε κάθε ομάδα θα ανατίθεται ένα θέμα. Η ανάθεση θα πραγματοποιείται είτε απευθείας απο καθηγητές, είτε αυτόματα ύστερα από κλήρωση. Τέλος μέσω της εφαρμογής δίνεται η δυνατότητα στις ομάδες να προτείνουν κάποιο θέμα, και τον καθηγητή που θα ήθελαν να την επιβλέψει. Οι καθηγητές απο την μεριά τους θα ανεβάζουν τους τίτλους με τα θέματα των πτυχιακών εργασιών και να διαχειρίζονται αιτήματα που πιθανό να έχουν προκύψει απο τις ομάδες. Τέλος η εφαρμογή θα χρησιμοποιείται και απο εναν χρήστη με δικαιώματα διαχειριστή, οπου εκτός απο κάποιες ρυθμίσεις για την συμπεριφορά της εφαρμογής, είναι και εκείνος που θα εκκινήσει την κλήρωση της ανάθεσης πτυχιακών εργασιών.

Ουσιαστικός σκοπός της συγκεκριμένης εργασίας είναι η κατανόηση της διδαχθείσας θεωρίας, και η εξοικείωση με διαδικασία ανάπτυξης μιας εφαρμογής. Ενέργειες όπως η ανάλυση των απαιτήσεων, η σχεδίαση, η επιλογή των τεχνολογιών και εργαλείων, είναι σημαντικές πριν προχωρήσουμε στην συγγραφή κώδικα και την υλοποίηση της εφαρμογής.

Περίληψη – Abstract

Σκοπός της εργασίας είναι η δημιουργία μιας διαδικτυακής εφαρμογής μέσω της οποίας θα πραγματοποιείται η ανάθεση των πτυχιακών εργασιών. Πρόκειται για μία εφαρμογή που απευθύνεται τόσο στους σπουδαστές όσο και τους καθηγητές του τμήματος Διοίκησης Επιχειρήσεων του Τεχνολογικού Εκπαιδευτικού Ιδρύματος Δυτικής Ελλάδας.

Οι σπουδαστές αφού κάνουν εγγραφή θα μπορούν να συνδεθούν, να διαμορφώσουν ομάδες και να επιλέξουν από τα διαθέσιμα θέματα εκείνα που τους ενδιαφέρουν. Επίσης δίνεται η δυνατότητα στους σπουδαστές να προτείνουν οι ίδοι κάποιο θέμα πτυχιακής εργασίας καθώς και τον επιβλέπων καθηγητή της επιλογής τους, ο οποίος θα είναι αρμόδιος να διαχειριστεί το αίτημα αυτό, να αποφασίσει δηλαδή εαν θέλει να αναλάβει η να απορίψει την συγκεκριμένη πρόταση. Οι καθηγητές θα μπορούν να συνδεονται και να ανεβάζουν θέματα πτυχιακών εργασιών, να βλέουν πόσες και ποιες ομάδες έχουν επιλέξει τα θέματα τους, και κατα βούληση να αναθέσουν απ ευθείας κάποιο θέμα. Τέλος οι καθηγητές θα πρέπει να διαχειρίζονται τα αιτήματα των ομάδων για επιβλεψη κάποιας πτυχιακής εργασίας. Η διαδικασία της κλήρωσης και αυτόματης ανάθεσης των πτυχιακών εργασιών, θα ξεκινήσει ύστερα από εντολή του διαχειριστή της εφαρμογής. Στην κλήρωση θα λάβουν μέρος μόνο ομάδες που δεν έχουν αναλάβει κάποιο θέμα, και τα θέματα που συμμετέχουν δεν έχουν ανατεθεί σε κάποια άλλη ομάδα. Ο αλγόριθμος εξετάζοντας κάποιες παραμέτρους θα αναθέτει ενα θέμα σε μια μόνο ομάδα.

The purpose of this dissertation is to develop a web application tasked with assigning students their senior thesis.

The application users have to register and log in in order to use the application. Also users are assigned to different roles with specific access rights. After signing up, students will be able to team up with their classmates, choose a research topic from a pre- existing list as well as propose their own and select a supervisor who will in turn accept or deny said proposal. Supervisors will have the ability to upload research topics, oversee how many of their students have finalized their topics, as well as allocate subjects at will. The administrator of the application will have the ability to initiate an automatic/ automated draw procedure, in cases where students or teams have not yet finalized their research topic. After examining an array of parameters, the algorithm will be appointing a thesis subject to a specific team.

Πίνακας περιεχομένων

Ευχαριστίες 1

Πρόλογος 2

Περίληψη – Abstract 3

Συμβολισμοί & Συντομογραφίες 6

1. Εισαγωγή 8

1.1 Σκοπός 8

1.2 Περιήγηση 8

1.3 Περίγραμμα πτυχιακής 9

1.4 Τεχνολογίες που χρησιμοποιήθηκαν 10

1.4.1 Git 10

1.4.2 IDE Visual Studio 12

1.4.3 SQL Server 12

1.4.4 SQL 12

1.4.5 C# 13

1.4.6 .NET Framework 13

1.4.7 ASP.Net Framework 14

1.4.8 MVC Pattern 15

1.4.9 HTML 16

1.4.10 CSS 16

1.4.11 JavaScript 16

1.4.12 Bootstrap Framework 16

2. Υλοποίηση της εφαρμογής 18

2.1 Ανάλυση απαιτήσεων και σχεδιασμός 18

2.2 Σχεδίαση βάσης δεδομένων 19

2.3 Entity Framework 19

2.4 Data Annotations 19

2.5 Δημιουργία κλάσεων 20

2.6 Κατασκευή της βάσης δεδομένων 25

2.7 Repository pattern 29

2.8 Dependency Injection 30

2.9 Εφαρμόζουμε το Dependency Injection 31

2.10 Controllers 32

2.10.1 Home Controller 32

2.10.2 Student Controller 32
2.10.3 Professor Controller 33
2.10.4 Team Controller 34
2.10.5 Project Controller 36
2.10.6 Assignment Controller 39
2.11 Views 40
3. Παρουσίαση λειτουργίας 46
3.1 Εισαγωγή 46
3.2 Περιήγησή ως Σπουδαστής 46
3.2.1 Εγγραφή & σύνδεση 46
3.2.2 Διαμόρφωση ομάδας 48
3.2.3 Επιλογή θεμάτων 50
3.3 Περιήγηση ως Διδάσκων 51
3.3.1 Σύνδεση 51
3.3.2 Πρόταση πτυχιακής εργασίας 52
3.3.3 Διαχείριση αιτήματος επίβλεψης πτυχιακής εργασίας 54
3.4 Περιήγηση ως Διαχειριστής 56
3.4.1 Σύνδεση και επισκόπηση 56
3.4.2 Πρόσκληση καθηγητή 57
3.4.3 Παραμετροποίηση εφαρμογής 58
3.4.4 Έναρξη διαδικασίας ανάθεσης 59
4. Συμπεράσματα – Μελλοντικές εργασίες 61
5. Παράρτημα 62
5.1 Πίνακας εικόνων 62
5.2 Βιβλιογραφία 63
5.3 Πηγές εικόνων 63
5.4 Download links 63

Συμβολισμοί & Συντομογραφίες

Ακρωνύμια	**Περιγραφή**
MVC	Model View Controller
API	Application Programming Interface
CSS	Cascading Style Sheet
DOM	Document Object Model
HTML	Hypertext Mark-up Language
JS	JavaScript
OOP	Object Oriented Programming
SQL	Structured Query Language
EF	Entity Framework
ASP	Active Server Pages
DB	Database

1. Εισαγωγή

1.1 Σκοπός

Σκοπός της παρούσας πτυχιακής εργασίας είναι η ανάπτυξη μιας διαδικτυακής εφαρμογής που έχει ως στόχο την βελτίωση της διαδικασίας για την ανάθεση των πτυχιακών εργασιών. Πρόκειται για μία εφαρμογή που απευθύνεται τόσο στους σπουδαστές αλλά και τους καθηγητές του τμήματος Διοίκησης Επιχειρήσεων του Τεχνολογικού Εκπαιδευτικού Ιδρύματος Δυτικής Ελλάδας. Πρόσβαση στην εφαρμογή θα έχει και ένας χρήστης με δικαιώματα διαχειριστή ο οποίος θα μπορεί να κάνει κάποιες παραμετροποιήσεις, ενώ θα πυροδοτήσει την διαδικασία της αυτόματης ανάθεσης των πτυχιακών εργασιών. Η εφαρμογή θα είναι ικανή να παρέχει με ασφάλεια την προβολή δεδομένων όπως τους σπουδαστές που έχουν εγγραφεί, τις ομάδες που έχουν διαμορφωθεί, τα διαθέσιμα προτεινόμενα από τους διδάσκοντες θέματα πτυχιακών εργασιών καθώς και τις αναθέσεις που έχουν πραγματοποιηθεί. Παρέχονται επίσης λειτουργικότητες απαραίτητες για μια web εφαρμογή όπως μεταξύ άλλων εγγραφής χρήστη, σύνδεσης και αποσύνδεσης χρήστη, αλλαγής κωδικού πρόσβασης. Τέλος υλοποιήσαμε μια διαδικασία καταγραφής σημαντικών ενεργειών σε ξεχωριστό αρχείο, όπου μπορούμε να ανατρέξουμε και να δούμε με χρονολογική σειρά σημαντικές ενέργειες που έγιναν στην εφαρμογή μας (logs).

1.2 Περιήγηση

Μια διαδικτυακή (web) εφαρμογή είναι διαθέσιμη στους χρήστες μέσω του Διαδικτύου (internet) ή κατά περίπτωση ενδοδικτύου (intranet), συνήθως όταν πρόκειται για εταιρικές εφαρμογές. Το στοιχείο που διακρίνει μια Διαδικτυακή (web) εφαρμογή έναντι άλλου είδους εφαρμογών είναι ότι ο χρήστης χρειάζεται μόνο έναν πρόγραμμα περιήγησης ή αλλιώς φυλλομετρητή (browser) για να την χρησιμοποιήσει. Μια web εφαρμογή φιλοξενείται και εκτελείται σε κάποιο ισχυρό υπολογιστικό μηχάνημα, που ονομάζεται σταθμός εξυπηρέτησης (server) ο οποίος διαφέρει ανάλογα με τις απαιτήσεις της εφαρμογής και τις τεχνολογίες που χρησιμοποιούνται. Στην περίπτωσή μας, αν και μιλάμε για μια web εφαρμογή, θα αναπτυχθεί και θα λειτουργήσει μόνο τοπικά, στον υπολογιστή μας (local host), όπου με τη βοήθεια εργαλείων θα κάνουμε το μηχάνημά μας να συμπεριφέρεται σαν ένας server.

Η εφαρμογή μας είναι διαθέσιμη στους χρήστες τριών διαφορετικών ρόλων, του σπουδαστή, του καθηγητή και του διαχειριστή. Σε όλους τους χρήστες παρέχονται κάποιες βασικές δυνατότητες ωστόσο ανά περίπτωση, και βάσει του ρόλου που έχει ο χρήστης θα έχει διαφορετικα ορισμένα δικαιώματα πρόσβασης, όπως για παράδειγμα στην ίδια σελίδα να βλέπει διαφορετικό περιεχόμενο. Οι ρόλοι της εφαρμογής θα είναι ‘*σπουδαστής*’ (student), ‘*καθηγητής*’ (professor) και ‘*διαχειριστής*’ (admin).

Συγκεκριμένα:

- Ο χρήστης με τον ρόλο του *σπουδαστή* αρχικά θα πρέπει να κάνει εγγραφή συμπληρώνοντας σε μια φόρμα στοιχεία όπως το ονοματεπώνυμό του, τον αριθμό μητρώου, το τμήμα προέλευσης ενώ απαραίτητα θα είναι η διεύθυνση ηλεκτρονικού

ταχυδρομείου (e-mail) και ένας κωδικός πρόσβασης (password) με τα οποία θα μπορεί να συνδέεται και να χρησιμοποιεί την εφαρμογή. Αφού συνδεθεί θα μπορέσει να δημιουργήσει μια ομάδα, και στη συνέχεια προεραιτικά να προσθέσει κάποιον άλλον σπουδαστή στην ομάδα που δημιούργησε. Στη συνέχεια κάθε μέλος της ομάδας έχει την δυνατότητα να επιλέξει από τα προτεινόμενα διαθέσιμα θέματα πτυχιακών εργασιών εκείνα που τον ενδιαφέρουν. Ωστόσο δίνεται η δυνατότητα στις ομάδες, εάν το επιθυμούν να προτείνουν οι ίδιες κάποιο θέμα πτυχιακής εργασίας, και τον επιβλέπων καθηγητή της επιλογής τους. Γίνεται ξεκάθαρο πως στην περίπτωση που κάποια ομάδα προτείνει θέμα πτυχιακής εργασίας, το συγκεκριμένο θέμα δεν θα είναι ορατό και διαθέσιμο σε άλλες ομάδες να το επιλέξουν. Το συγκεκριμένο θέμα θα είναι ορατό μόνο στους σπουδαστές μέλη της ομάδας που το πρότεινε καθώς και στον καθηγητή στον οποίο έχει γίνει η πρόταση.

- Ο χρήστης με τον ρόλο του καθηγητή δεν έχει την δυνατότητα εγγραφής, καθώς η εγγραφή του στην εφαρμογή θα γίνεται από τον διαχειριστή. Όταν η εγγραφή πραγματοποιηθεί απο τον διαχειριστή, ο καθηγητής θα λάβει εάν e-mail με εναν υπερσύνδεσμο που οδηγεί στην εφαρμογή καθώς και τα στοιχεία εισόδου του σε αυτήν. Κατά την πρώτη σύνδεση στην εφαρμογή, για λόγους ασφαλείας ο καθηγητής καλείται να αλλάξει τον κωδικό πρόσβασης. Ύστερα από την επιτυχή σύνδεση στην εφαρμογή ο χρήστης με τον ρόλο του καθηγητή, θα μπορεί να δημιουργεί, επεξεργάζεται, διαγράφει θέματα πτυχιακών εργασιών, και προαιρετικά να αναθέτει απ' ευθείας κάποιο από τα θέματα που έχει ο ίδιος προτείνει σε συγκεκριμένη ομάδα. Τέλος καλείται να διαχειρίζεται αιτήματα για επίβλεψη πτυχιακής εργασίας, όπως έχουν σταλθεί από τις ομάδες, δηλαδή με βάση τον τίτλο, την περιγραφή κάποιου θέματος να επιλέξει εάν δέχεται να επιβλέψει την συγκεκριμένη ομάδα ή να απορρίψει την πρόταση.
- Ο χρήστης με τον ρόλο του διαχειριστή, δεν θα έχει δικαιώματα που έχουν οι χρήστες που προαναφέραμε, όμως είναι εκείνος που μπορεί να διαχειρίζεται τους χρήστες, καθώς και να αλλάξει κάποιες ρυθμίσεις της εφαρμογής. Επίσης είναι ο μοναδικός χρήστης που μπορεί να εκκινήσει την διαδικασία αυτόματης ανάθεσης πτυχιακών εργασιών.

1.3 Περίγραμμα πτυχιακής

Στο πρώτο κεφάλαιο θα δούμε τα εργαλεία, τις γλώσσες και τεχνολογίες που θα χρησιμοποιήσουμε. Δεν θα αναφερθούμε σε ότι έχει να κάνει με υλικό αλλά μόνο σε επιπεδο λογισμικού.

Στο δεύτερο κεφάλαιο, που ακολουθεί θα δούμε πως αξιοποιήσαμε τα εργαλεία που προαναφέραμε. Αρχικά θα μιλήσουμε για την αρχιτεκτονική της εφαρμογής μας και τον τρόπο που οργανώσαμε τα αρχεία και τον κώδικά. Θα σας παρουσιάσουμε κάποιες πρακτικές που ακολουθήσαμε καθώς και την σημασία τους με γνώμονα την εύκολη συντήρηση της εφαρμογής αλλά και την δυνατότητα το έργο να συνεχιστεί, και να μεγαλώσει μελλοντικά. Στο τέλος του δευτέρου κεφαλαίου θα δούμε αποσπάσματα από τον πηγαίο κώδικα της εφαρμογής,

εμβαθύνοντας και επεξηγώντας ορισμένα αποσπάσματα που θεωρούμε ότι έχουν κάποιο μεγαλύτερο ενδιαφέρον.

Στο τρίτο κεφάλαιο θα δούμε μαζί την εφαρμογή σε λειτουργία, χρησιμοποιώντας τρεις χρήστες με διαφορετικό ρόλο για τον καθε έναν.

Τέλος στο τέταρτο κεφάλαιο θα αναφέρουμε κάποια προβλήματα που αντιμετωπίσαμε, λάθη που θα μπορούσαμε να αποφύγουμε ενώ θα μοιραστούμε σκέψεις για το πως η συγκεκριμένη εργασία μπορεί να εξελιχθεί και να βελτιωθεί.

1.4 Τεχνολογίες που χρησιμοποιήθηκαν

Μία web εφαρμογή όσον αφορά την υλοποίηση της, διακρίνεται σε δυο βασικές κατηγορίες, το front-end και το back-end.

Το front-end αφορά το τμήμα εκείνο το οποίο ο χρήστης μπορεί να δει και να αλληλεπιδράσεις. Με τη σειρά του χωρίζεται σε σχεδιασμός και υλοποίηση. Εμείς θα σταθούμε κυρίως στο κομμάτι της υλοποίησης όσον αφορά το front-end, ενώ θα αναφέρουμε και εργαλεία που μας βοήθησαν να έχουμε ένα καλό αισθητικό αποτέλεσμα χωρίς να έχουμε κάνει κάποια αναλυτική σχεδίαση. Οι πιο συνηθισμένες τεχνολογίες που συναντάμε στο front-end και θα αναλύσουμε σε επόμενη ενότητα είναι η γλώσσα HTML η οποία είναι η κύρια γλώσσα σήμανσης για τις ιστοσελίδες και τα στοιχεία της είναι βασικά δομικά στοιχεία των ιστοσελίδων, η γλώσσα CSS με την οποία μορφοποιούμε μια HTML σελίδα και η JavaScript η οποία μας επιτρέπει να κάνουμε την σελίδα πιο διαδραστική.

Το back-end αφορά τόσο το λογισμικό όσο και το υλικό για την λειτουργία μιας εφαρμογής. Μπορεί με τη σειρά του να διακριθεί σε τρία μέρη. Το πρώτο μέρος αφορά τον διακομιστή (server) δηλαδή το φυσικό υπολογιστικό μηχάνημα που τρέχει η εφαρμογή. Το δέυτερο μέρος που διακρίνουμε στο back-end αποτελεί η ίδια η εφαρμογή (application), όπου είναι και το βασικό που αναλύουμε στην παρούσα εργασία, και τέλος η βάση δεδομένων (database) για την αποθήκευση πληροφοριών αλλά και την παραγωγή δεδομένων.

1.4.1 Git

Κατά την διάρκεια υλοποίησης της εργασίας, πολλές φορές κάναμε κάποιο λάθος ή αλλαγές που δημιουργούσαν προβλήματα στην λειτουργία της εφαρμογής. Ήταν επιτακτική η ανάγκη να εντοπίσουμε και να αναιρέσουμε τις συγκεκριμένες αλλαγές που προκάλεσαν το πρόβλημα. Στην αρχή αυτό δεν ήταν δυνατό με αποτέλεσμα ορισμένες φορές να χάνουμε μέρος της δουλειάς που είχαμε κάνει. Για να αντιμετοπίσουμε λοιπόν το πρόβλημα αυτό, ξεκινήσαμε να χρησιμοποιούμε την τεχνολογία GIT.

Το GIT είναι ένα σύστημα ελέγχου εκδόσεων (version controlling system ή αλλιώς source code manager) που παρακολουθεί και διατηρεί ιστορικό με τις αλλαγές που κάνουμε σε ένα αρχείο, ή σε κάποιο φάκελο. Υπάρχει επίσης η δυνατότητα, ξεκινώντας απο κάποιο σημείο αναφοράς να δημιουργήσω ενα νέο μονοπάτι (branch) όπου σε κάθε branch μπορώ να δουλεύω συγκεκριμένα

πράγματα, και στο τέλος όλα αυτά να συγκροτηθούν. Έτσι μπορούμε σχετικά εύκολα να επιστρέψουμε σε μια προηγούμενη κατάσταση.

Κάθε κατάλογος εργασίας του Git είναι ένα ολοκληρωμένο αποθετήριο λογισμικού (repository) με πλήρες ιστορικό και δυνατότητες πλήρους παρακολούθησης της έκδοσης, ανεξάρτητα από την πρόσβαση δικτύου ή ενός κεντρικού διακομιστή. Το GIT λειτουργεί με έναν αρκετά γρήγορο και έξυπνο τρόπο καθώς κάθε φορά που εμείς θέλουμε, κάνοντας commit, κρατάει ένα στιγμιότυπο από το αρχείο, θα συγκρίνει τις δυο εκδόσεις (την παλιά με την νέα) και θα ενημερώσει μόνο τα σημεία που έχουν υποστεί αλλαγές. Κάθε φορά που κάνουμε το commit διατηρείται ενα νέο στιγμιότυπο με τις αλλαγές που κάναμε, ενώ μπορούμε να ορίσουμε και κάποιο μύνημα για να μας είναι πιο έυκολο. Κάθε commit έχει εναν μοναδικό κωδικό με κρυπτογράφηση SHA-1 και μήκος έως και 40 ψηφία ο οποίος αποτελεί την ταυτότητα του commit.

Ένα πολύ βασικό πλεονέκτημα που έχει το GIT εμφανίζεται όταν πρόκειται πολλά άτομα να εργαστούν στο ίδιο project. Σε αυτή την περίπτωση θα μπορούσαμε να έχουμε τον κώδικα μας ανεβασμένο στο διαδίκτυο, και στη συνέχεια κάθε μέλος της ομάδας θα δούλευε πάνω σε κάποιο αντίγραφο. Στη συνέχεια κάθε μέλος μπορεί να ανεβάζει τις αλλαγές που κάνει, ο κώδικας συγκροτείται με τη σειρά και έτσι αναπτύσσεται πιο γρήγορα και με μεγαλύτερη ασφάλεια.

Ο τρόπος που εμείς εργαστήκαμε είναι για κάθε νέα ενέργαια ή αλλαγή, να δημιουργούμε ενα ξεχωριστό branch, να το ελέγχουμε σχολαστικά με διάφορα τέστ, και εάν όλα λειτουργούν όπως αναμέναμε αυτό να συγχωνέυεται με το βασικό branch που είναι το master, διατηρώντας έτσι το master branch χωρίς μισοτελειωμένες λειτουργικότητες και χωρίς σφάλματα στον κώδικα.

Κάποιες βασικές εντολές που χρησιμοποιήσαμε κατα την διάρκεια υλοποίησης της εφαρμογής είναι τα παρακάτω :

git init	Δημιουργία ενος νέου repository
git add <filename>	Προσθήκη ενός νέου ή ενος τροποποιημένου αρχείο στον κατάλογο εργασίας
git add .	Προσθήκη όλων των αρχείων στον κατάλογο εργασίας
git remote add origin <link of remote repository>	Συγχρονισμός του repository που έχω στο μηχάνημά μου, με κάποιο απομακρυσμένο πχ στο Github.
git commit -m"<commit message>"	Αποθήκευση των αλλαγών στο repository που βρίσκεται στο μηχάνημά μας, και ορισμός σχετικού μυνήματος
git push -u origin <branch>	Ενημέρωση του απομακρυσμένου repository, αποστέλοντας τις αλλαγές που έχουμε τοπικά, σε συγκεκριμένο branch

git-pull	Ενημέρωση του τοπικού repository, λαμβάνωντας τις αλλαγές που πιθανό να υπάρχουν στο απομακρυσμένο, για το branch στο οποίο δουλευω
git branch	Βλέπω σε ποιο branch εργάζομαι
git branch <new branch name>	Δημιουργώ νέο branch
git checkout <branch name>	Αλλάζω branch
git merge <branch name>	Συγχωνέυω το branch της παραμέτρου με το branch που εργάζομαι
git log	Βλέπω τα commits που έχουν γίνει, απο ποιον έγινε, πότε έγινε καθώς το σχετικό commit message
git reset HEAD~1	Σβήνω το τελευταίο commit στο git

1.4.2 IDE Visual Studio

Βασικό λογισμικό που χρησιμοποιήσαμε για την ανάπτυξη της εφαρμογής ήταν το Visual Studio, και συγκεκριμένα το Visual Studio Community η οποία είναι μια δωρεάν έκδοση. Πρόκειται για ένα IDE (Integrated Development Environment) της Microsoft και δίνει την δυνατότητα επεξεργασίας, μεταγλώττισης εκτέλεσης και απασφαλμάτωσης (debugging) του κώδικα ενός προγράμματος μέσα από το ίδιο περιβάλλον. Ο λόγος που χρησιμοποίησα το συγκεκριμένο IDE είναι διότι για τις τεχνολογίες που χρησιμοποιώ με βοηθάει να γράφω πιο γρήγορα, απο το να δίνω εντολές απο κάποιο terminal console, να έχω καλύτερο έλεγχο καθώς εντοπίζω πιο εύκολα λάθη ή παραλείψεις, ενώ κάνει αρκετές απαραίτητες επαναλαμβανόμενες διαδικασίες αυτόματα.

1.4.3 SQL Server

Έπειτα από έρευνα αποφασίσαμε η πτυχιακή εργασία να αναπτυχθεί με την χρήση του ASP.NET MVC Framework. Στη συνέχεια θα σας παρουσιάσω περισσότερες λεπτομέρειες σχετικά με το ASP.NET Framework και ωστόσο αυτό που αξίζει να αναφέρουμε στο σημείο αυτό είναι πως κατάλληλο περιβάλλον για την φιλοξενία και λειτουργία μιας τέτοιας εφαρμογής είναι ο SQL Web Server. Όπως προαναφέραμε η εφαρμογή θα υλοποιηθεί και θα λειτουργήσει τοπικά, στον υπολογιστή μας, και όχι σε κάποιον διακομιστή (server). Αυτό θα γίνει με την χρήση του IIS (Internet Information Services), ο οποίος είναι ένας διακομιστής ιστού γενικής χρήσης που λειτουργεί στο λειτουργικό σύστημα Windows. Κατεβάσαμε λοιπόν την εφαρμογή SQL Server 2019 Express όπου είναι μια δωρεάν έκδοση του κύριου συστήματος διαχείρισης σχεσιακών βάσεων δεδομένων της Microsoft. Για την διαχείριση της βάσης δεδομένων κατεβάσαμε και χρησιμοποιήσαμε την εφαρμογή SQL Server Mmanagement Studio (SSMS).

1.4.4 SQL

Η γλώσσα SQL (Structured Query Language) είναι μια γλώσσα που χρησιμοποιείται στον προγραμματισμό για τον σχεδιασμό και την διαχείριση δεδομένων (domain specific

language), σε μια σχεσιακή βάση δεδομένων (Relational Database Management System ή εν συντομία RDBMS). Το σχεσιακό μοντέλο βάσεων δεδομένων είναι μια απλή και κατανοητή δομή των βάσεων, το οποίο μεταξύ άλλων μας εξυπηρετεί καθώς:

1. αποφεύγονται οι διπλό-εγγραφές στην βάση και έτσι διατηρούμε χαμηλά επίπεδα κατανάλωσης πόρων
2. διατηρείται σωστή λογική στις σχέσεις μεταξύ των πινάκων
3. τα δεδομένα δεν εξαρτώνται από τις εφαρμογές που χρησιμοποιούν την συγκεκριμένη βάση.

1.4.5 C#

Η C# (C Sharp) είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού που αναπτύχθηκε από την Microsoft το 2000 και τρέχει σε .NET Framework. Η C# έχει τις ρίζε της στην οικογένεια C και μοιάζει αρκετά με την C++ και την Java. Σκοπό έχει να κάνει τα απλά πράγματα εύκολα και τα δύσκολα εφικτά. Πρόκειται για μια από τις δημοφιλέστερες γλώσσες προγραμματισμού στον κόσμο, και ένας λόγος που την κάνει τόσο δημοφιλή είναι ότι με την συγκεκριμένη γλώσσα μπορούμε αναπτύξουμε Mobile εφαρμογές, Desktop εφαρμογές, Web εφαρμογές, WEB services, παιχνίδια, VR.

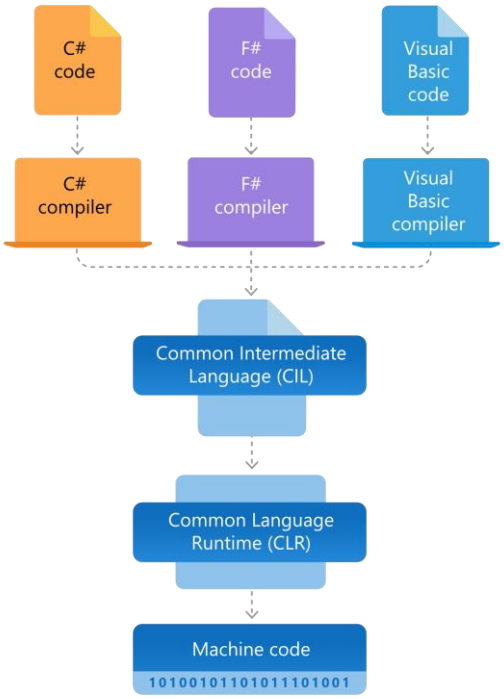
Αν και πρόκειται για μια αντικειμενοστραφή γλώσσα, που υποστηρίζει τις βασικές αρχές του αντικειμενοστραφούς προγραμματισμού όπως κληρονομικότητα, πολυμορφισμό, ενθυλάκωση και γενίκευση, ενσωματώνει και μερικά χαρακτηριστικά διαφόρων προγραμματιστικών πρωτοτύπων. Είναι μία γλώσσα που συνεχώς ενημερώνεται κάθε φορά με νέα χαρακτηριστικά και συντακτικό.

Ο βασικός λόγος που επέλεξα την συγκεκριμένη γλώσσα είναι καθώς είναι μια απλή γλώσσα που μπορεί να χρησιμοποιηθεί χωρίς εντατική εκμάθηση ενώ οι βασικές αρχές της μπορούν να κατανοηθούν εύκολα και γρήγορα, με αποτέλεσμα να είμαι παραγωγικός σε σύντομο χρονικό διάστημα.

1.4.6 .NET Framework

Το .NET Framework είναι ένα πλαίσιο λογισμικού, το οποίο αναπτύχθηκε από την Microsoft και προορίζεται για την πλατφόρμα των Windows. Αποτελείται από μια αρκετά μεγάλη βιβλιοθήκη το Framework Class Library (FCL) το οποίο παρέχει την δυνατότητα δυο διαφορετικών γλωσσών προγραμματισμού να αλληλοεπιδράσουν ως μέλη του ίδιου συστήματος.

Τα προγράμματα που είναι γραμμένα για το .NET Framework εκτελούνται αρχικά σε περιβάλλον λογισμικού και όχι υλικού, το οποίο ονομάζεται Common Language Runtime (CLR). Το CLR περιέχει μια εικονική μηχανή (virtual machine) που διαχειρίζεται την εκτέλεση ενός προγράμματος και παρέχει μια σειρά σημαντικών υπηρεσιών, χαρακτηριστικά για το .NET Framework, όπως ασφάλεια, διαχείριση μνήμης και διαχείριση σφαλμάτων. Το FCL και CLR απαρτίζουν το .NET Framework.



Εικόνα 1 - .NET Framework

1.4.7 ASP.Net Framework

Το ASP.Net είναι ένα ανοικτού κώδικα server-side Framework σχεδιασμένο για την ανάπτυξη web εφαρμογών και ιστοσελίδων. Αναπτύχθηκε από την Microsoft και σκοπό έχει να βοηθήσει τους προγραμματιστές να χτίσουν μια πλατφόρμα ή μια εφαρμογή χωρίς να ξεκινούν κάθε φορά από το μηδέν, παρέχοντας κάποιες απλές συναρτήσεις ή έτοιμα configurations. Το ASP.Net είναι γραμμένο σε .Net γλώσσες και διαθέτει άδεια Apache. Λειτουργεί σε Windows, και Linux ή MacOS (ASP.Net Core). Χρησιμοποιεί εντολές HTTP και λειτουργεί με πρωτόκολλο HTTP για να έχει σωστή επικοινωνία σε επίπεδο προγράμματος περιήγησης (browser) και διακομιστή (server). Όπως είπαμε απλοποιεί τη λειτουργία ιστοσελίδων και εφαρμογών καθώς αναπτύσει μια βιβλιοθήκη με κώδικα και εργαλεία μερικά απο τα οποία αναφέρονται επιγραμματικά παρακάτω.

1. Standard Compliance
    a. Browser capability detection
    b. Style properties on the control
    c. Control Adapters
    d. Page and control callbacks
2. Reliability and Performance
    a. Long-Running Requests (>110 seconds)
    b. SqlMembershipProvider
    c. Asynchronous Page Events with Web Forms

d. Request Entity Body
e. Fire-and-Forget Work
f. EnableViewState and ViewStateMode
g. Redirect and Response.End

3. Security
    a. AppSettings
    b. EnableViewStateMac
    c. Medium Trust
    d. Request Validation
    e. Cookieless Forms Authentication and Session

Μπορούμε να χρησιμοποιήσουμε το ASP.Net για front-end και back-end ανάπτυξη ενώ πρόκειται για ένα Framework που ακολουθεί την λογική Model View Controller.

1.4.8 MVC Pattern

Έκανε την εμφάνιση του το 1978 στην Smrttalk, μια αντικειμενοστραφή γλώσσα προγραμματισμού, και είναι μια τεχνική που χρησιμοποιείται μέχρι και σήμερα από αρκετές πλατφόρμες και Frameworks όπως είναι το ASP.NET MVC Framework. Το μοντέλο MVC είναι μια αρχιτεκτονική που διακρίνει την εφαρμογή σε τρια βασικά στοιχεία. Τα στοιχεία αυτά, όπως μαρτυρούν τα ακρωνύμια MVC είναι το Model, το View και ο Controller, αν και αρκετοί θεωρούν πως θα πρέπει να μετονομαστεί σε MVCR προσθέτωντας δηλαδή και το Routing.

Model

Το μοντέλο αποτελεί όλη τη λογική που σχετιζεται με τα δεδομένα με τα οποία θα εργαστεί ο χρήστης. Αυτό μπορεί να αντιπροσωπεύει είτε τα δεδομένα που μεταφέρονται μεταξύ των στοιχείων View και Controller, είτε οποιαδήποτε άλλα δεδομένα που σχετίζονται με τη λογική της εφαρμογής μας ή όπως ίθισται να αποκαλείται business logic. Το Model είναι το σκαρίφημα που θα δημιουργηθεί κάποιο αντικείμενο. Περιέχει διάφορες συναρτήσεις με τις οποίες μπορούμε να λάβουμε δεδομένα από μια πηγή δεδομένων όπως για παράδειγμα μια βάση και να τα καταχωρήσουμε σε διάφορα αντικείμενα. Για παράδειγμα ενα αντικείμενο τύπου ‘Σπουδαστής’ θα ανακτήσει τις πληροφορίες των σπουδαστών απο την βάση δεδομένων, θα γίνουν ενδεχομένως διάφορες ενέργειες και στη συνέχεια θα ενημερώσει την βάση δεδομένων.

View

Το View αποτελεί το user interface και είναι ο χώρος που βλέπει ο χρήστης, όπου στην περίπτωση μας αφού μιλάμε για web εφαρμογή, κάποιες HTML σελίδες. Είναι το σημείο που αναπαριστά τα δεδομένα και το περιβάλλον αλληλεπίδρασης με τον τελικό χρήστη.

Controller

Ο Controller λειτουργεί ως ο συνδετικός κρίκος ανάμεσα στο View που προαναφέραμε και το Model, και είναι υπευθυνος για την διαχείρηση των αιτημάτων, την διαχείρηση των δεδομένων

με την βοήθεια του Model φυσικά, και την αληλλεπίδραση που υπάρχει στο View ωστε να εξάγει το τελικό αποτέλεσμα. Για παράδειγμα ο Controller θα λάβει κάποιο request από το View, θα το προωθήσει στο Model για να λάβει τα δεδομένα. Αφού ο Controller λάβει απάντηση από το Model, θα προωθήσει τα δεδομένα στο View. Φυσικά σε ενδεχόμενο λάθους, τα MVC Framework είναι πλήρως εξοπλισμένα με συναρτήσεις για σφάλματα που μας δείχνουν τι πήγε στραβά.

1.4.9 HTML

Η HTML (Hypertext Markup Language), Γλώσσα Σήμανσης Υπερκειμένου όπως μεταφράζεται στην ελληνική, είναι η κύρια γλώσσα σήμανσης για τις ιστοσελίδες, και τα στοιχεία της είναι τα βασικά δομικά στοιχεία των ιστοσελίδων. Εμείς χρησιμοποιούμε την 5η έκδοση της HTML, η οποία κυκλοφόρισε το 2011 από την W3C (World Wide Web Consortium). Η εκδοση αυτή υποστηρίζεται από τους περισσότερους φυλλομετρητές (browsers) ενώ παρέχει πλουσιότερη γκάμα ψηφιακών μέσων.

1.4.10 CSS

Η CSS (Cascade Style Sheet) είναι μια γλώσσα που ορίζει την εμφάνιση ενός δομημένου εγγράφου. Όπως είπαμε η HTML είναι υπεύθυνη για την δομή μιας σελίδας ενώ η CSS είναι υπεύθυνη για το τρόπο που θα εμφανίζεται ένα έγγραφο, το εικαστικό αποτέλεσμα δηλαδή. Αυτό επιτυγχάνεται μέσω εντολών που επηρεάζουν μεταξύ άλλων την γραμματοσειρά, χρώματα και φόντο, εφέ, πολλαπλές διατάξεις στηλών, κινούμενη απεικόνιση και άλλα. Επίσης μέσω της CSS μπορούμε να πετύχουμε διαφορετική εμφάνιση του εγγράφου ανάλογα την συσκευή, το πρόγραμμα περιήγησης του χρήστη ή την ανάλυση της οθόνης του (responsive design).

1.4.11 JavaScript

Η JavaScript είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού, αρκετά δημοφιλής και ευρέως γνωστή. Αναπτύχθηκε από την Netscape Communication Corporation το 1996, και αρχικά η χρήση της περιοριζόταν κυρίως για δυναμική αλληλεπίδραση μιας σελίδας με τους χρήστες. Λειτουργεί στα περισσότερα προγράμματα περιήγησης, ενώ συνεργάζεται άψογα με την HTML. Πλέον αξιοποιώντας διάφορα JavaScript Frameworks χρησιμοποιείται και από προγραμματιστές για διαδικασίες που αφορούν και το back-end.

1.4.12 Bootstrap Framework

Για περισσότερη ευκολία, οικονομία χρόνου και καλύτερο αισθητικό αποτέλεσμα χρησιμοποιήσαμε το Bootstrap Framework το οποίο μας δίνει κάποια έτοιμα σχεδιασμένα στοιχεία όπως κουμπιά, γραμματοσειρές, πλαίσια και άλλα. Καλώντας κάποιες συγκεκριμένες CSS κλάσεις, διαμορφώνουμε πιο εύκολα, γρήγορα και με συνοχή την εμφάνιση της σελίδας μας. Μαζί με το Bootstrap Framework συμπεριλάβαμε επίσης και τη βιβλιοθήκη jQuery, με την

οποία επιτυγχάνουμε μεταξύ άλλων μερικά animations και εφέ για πιο ευχάριστη εμπειρία του χρήστη.

2. Υλοποίηση της εφαρμογής

Κατά την υλοποίηση της εφαρμογής και όσον αφορά τη συγγραφή κώδικα, αν και δεν ήταν εξ' αρχής εύκολο προσπαθήσαμε να ακολουθήσουμε κάποιες βασικές αρχές για την ανάπτυξη λογισμικού. Οι αρχές αυτές γνωστές και ως SOLID principles, αναφέρουν:

S - Single Responsibility Principle

O - Open/Closed Principle

L - Liskov's Substitution Principle

I - Interface Segregation Principle

D - Dependency Inversion Principle

Στόχος ηταν να γραφουμε οσο γίνεται κώδικα που μπορούμε να επαναχρησιμοποιούμε, να μην επαναλαμβανόμαστε και να είμαστε όσο γίνεται ευέλικτοι στις αλλαγές που ενδεχομένως προκύψουν στο μέλλον.

2.1 Ανάλυση απαιτήσεων και σχεδιασμός

Η επιτυχία ενός έργου εξαρτάται σε μεγάλο βαθμό από την κατανόηση του προβλήματος, την ανάλυση των απαιτήσεων, τον εντοπισμό και ανάδειξη υπό-προβλημάτων που μπορεί να προκύψουν. Με τον τρόπο που δουλέψαμε και όπως θα δούμε παρακάτω, η πρώτη ενέργεια ήταν ο εντοπισμός των βασικών οντοτήτων της εφαρμογής και η δημιουργία των σχετικών κλάσεων. Αρχικά δημιουργήσαμε ενα νέο Solution μέσω του Visual Studio. Το Solution είναι ένας χώρος – ένας εικονικός φάκελος, που θα περιέχει όλα τα project που θα δημιουργήσουμε στη συνέχεια. Για καλύτερη οργάνωση των αρχείων και του κώδικά μας, θα χρησιμοποιήσουμε μερικά διαφορετικά projects, κάθε ένα εκ των οποίων θα έχει συγκεκριμένο σκοπό και περιεχόμενο. Τα project επικοινωνούν όμως είναι ανεξάρτητα μεταξύ τους, το οποίο βοηθάει στην συντήρηση ή αλλαγές που πιθανό να προκύψουν στο μέλλον, ακολουθώντας μια σχεδιαστική πρακτική που ονομάζεται separation of concerns (SoC).

Ξεκινάμε λοιπόν με την δημιουργία των projects δίνοντας μια σύντομη περιγραφή για τον τύπο και περιεχόμενο του καθενός, και είναι τα εξής:

Thesis.Entities	Class Library	περιλαμβάνει τις κλάσεις που αναπαριστούν τις οντότητες της εφαρμογής μας
Thesis.Database	Class Library	περιλαμβάνει τις κλάσεις που θα είνα υπευθυνες για την επικοινωνία της εφαρμογής μας με την βάση δεδομένων
Thesis.Services	Class Library	περιλαμβάνει τις κλάσεις με ρόλο διαμεσολαβητή ανάμεσα στην οποιαδήποτε πηγή δεδομένων και την λογική της εφαρμογής μας
Thesis.WebApp	ASP.Net MVC	αποτελεί τον πυρήνα της εφαρμογής μας, περιλαμβάνει

		models, Controllers και τα Views
Thesis.UnitTests	Unit Test	περιλαμβάνει κλάσεις και μεθόδους που σκοπό έχουν να ελέγξουν και να επιβεβαιώσουν την ορθή λειτουργεία της εφαρμογής μας και των επιμέρους μεθοδων απο την οποια αυτη αποτελείται

Ο τρόπος αυτός που οργανώσαμε τα αρχεία της εφαρμογής μας, δεν είναι μοναδικός ούτε ο καλύτερος, αλλά βοηθάει το project μας είναι πιο οργανωμένο και ο κώδικάς μας πιο καθαρός.

2.2 Σχεδίαση βάσης δεδομένων

Η βάση δεδομένων που θα κατασκευάσουμε ακολουθεί τα πρότυπα Σχεσιακού Μοντέλου, με πίνακες οι οποίοι συνδυάζουν δεδομένα άλλων πινάκων. Για την δημιουργία τόσο της βάσης όσο και των πινάκων, δεν θα χρησιμοποιήσουμε άμεσα την SQL γλώσσα όπως συνηθίζεται, αλλά θα αξιοποιήσουμε ένα εργαλείο, το Entity Framework.

2.3 Entity Framework

Το Entity Framework είναι ένα open-source Framework που βασίζεται στο ADO.Net, και χρησιμοποιεί μια τεχνική που ονομάζεται object-relational mapping. Μέσω του Entity Framework οπως θα δούμε παρακάτω εμείς απλώς θα δημιουργήσουμε τις κλάσεις για τις βασικές οντότητες, και η λογική αυτή θα μετατραπεί σε μια SQL βάση δεδομένων. Είναι πολύ σημαντικό να ορίσουμε και να συσχετίσουμε τις κλασεις σωστά, να περιγράψουμε καλύτερα ένα χαρακτηριστικό μιας Classς, προκειμένου να δημιουργηθεί σωστά η βάση δεδομένων. Για να περιγράψουμε καλύτερα ή να δώσουμε οδηγίες στο Entity Framework για το πως να χειριστεί κάποια Class ή κάποιο χαρακτηρηστικό, επιτυγχάνεται με την χρήση της βιβλιοθήκης Data Annotations.

2.4 Data Annotations

Τα Data Annotations είναι μια βιβλιοθήκη που μας δίνει την δυνατότητα να περιγράψουμε καλύτερα τα χαρακτηριστικά (properties) ή πεδία (fields) μιας Class. Για παράδειγμα όταν αφορά το Entity Framework υπάρχουν κάποια attributes με τα οποία ορίζουμε εμείς ποιο θα είναι το πρωτεύων κλειδί, εάν υπάρχει ξένο κλειδί ή σε περίπτωση που θέλουμε κάποιο property να μην δημιουργηθεί η αντίστοιχη στήλη στον πίνακα. Επίσης με την χρήση των Data Annotations μπορούμε να κάνουμε εύκολα κάποια Data Validations, διασφαλίζοντας πως η βάση θα λάβει έγκυρες τιμές για κάθε πεδίο του πίνακα, ενώ μπορούμε να ορίσουμε και χαρακτηριστικά που αφορούν τον τρόπο που θα εμφανίζονται τα δεδομένα μας.

Για να χρησιμοποιήσουμε το Entity Framework, το εγκαθιστούμε μέσω εντολών στην κονσόλα ή μέσω του NuGet package manager από το Visual Studio. Συνεχίζουμε με την δημιουργία των κλάσεων, βάσει των οποίων θα κατασκευαστεί η βάση δεδομένων.

2.5 Δημιουργία κλάσεων

Σε αυτό το σημείο να αναφέρουμε πως μερικές οντότητες της εφαρμογής μας έχουν οριστεί και παρέχονται από το ASP.NET Framework, συνεπώς δεν θα κάνουμε εκτενή αναφορά σε αυτές. Παρακάτω θα δημιουργήσουμε τις κλάσεις για τις βασικές οντότητες της εφαρμογής μας, οι οποίες είναι ο σπουδαστής, ο καθηγητής, η ομάδα, το θέμα και η ανάθεση. Ξεκινάμε λοιπόν με την δημιουργία των κλάσεων στο class library project που δημιουργήσαμε, με όνομα Thesis.Entites.

Τα στοιχεία που θέλουμε να αποθηκεύουμε για κάθε έναν σπουδαστή είναι :

- Ενας μοναδικός κωδικός, ο οποίος αν και θα μπορούσε να είναι ο Αριθμός Μητρώου του σπουδαστή, επιλέξαμε να είναι ένας αύξων ακέραιος αριθμός που θα αποτελεί και το κλειδί του πίνακα στην βάση. Ο λόγος που επιλέξαμε το πρωτεύον κλειδί να διαφέρει από κάποιο χαρακτηριστικό που ήδη έχει ο σπουδαστής είναι για να διαχειριστούμε καλύτερα στο μέλλον πιθανές αλλαγές, όπως απενεργοποίηση λογαριασμού, διαγραφή, επανεγγραφή κάποιου σπουδαστή κλπ.
- Το ονομα και το επώνυμό, δυο χαρακτηρηστικά που θα είναι τύπου string.
- Ο αριθμός μητρώου, όπου αν και πρόκειται για αριθμό δεν χρειαζόμαστε να γίνουν μαθηματικές πράξεις οπότε ο τύπος της πληροφορίας αυτής θέλουμε να είναι string.
- Το τμήμα προέλευσης του, μιας και ορισμένοι σπουδαστές προέρχονται απο άλλα τμήματα λόγω συγχώνευσης. Η πληροφορία για το τμήμα προέλευσης θα είναι τύπου department, ενας τύπος που εμείς έχουμε δημιουργήσει και θα αναφέρουμε παρακάτω.
- Την ομάδα που ανήκει ο σπουδαστής. Ενδεχομένως κάποιος σπουδαστής να μήν ανήκει ακόμη σε κάποια ομάδα, οπότε το πεδίο αυτό μπορεί να είναι κενό, δηλαδή θα είναι τύπου nullable int.
- Τον κωδικό χρήστη (user ID). Κάθε σπουδαστής που κάνει εγγραφή και κάνει χρήση της εφαρμογής συσχετίζεται με εναν Application User.

Συνοψίζοντας η Class για τον σπουδαστή είναι:

```
public Class Student
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string AM { get; set; }
    public Department Department { get; set; }
    public int? TeamId { get; set; }
    public virtual Team Team { get; set; }
    public string UserId { get; set; }
    public ApplicationUser User { get; set; }
}
```

Το τμήμα που ανήκει ενας σπουδαστής είναι τύπου 'Department' και είναι ενας τύπος που εμείς δημιουργήσαμε. Συγκεκριμένα είναι ένα Enum με διαθέσιμες επιλογές τμήματα που έχουν συγχωνευτεί. Τα ακρωνύμια εξυπηρετούν τον κώδικα ωστε να είναι πιο ευαναγνωστος ωστόσο η τιμή που θα αποθηκευτεί στην βάση για κάθε επιλογή ορίζεται απο την σειρά που έχει το κάθε επιλογή, ξεκινοντας απο το μηδέν και είναι 0 για DE και 1 για EPDO. Το Display attribute, αφορά την πληροφορία που θα δει ο χρήστης, είναι μια περιγραφή της εκάστοτε επιλογής.

Οι επιλογές προς το παρόν είναι δυο και είναι :

Option	**Value**	**Display name**
DE	0	Διοίκησης επιχειρήσεων
EPDO	1	Εφαρμογών πληροφορικής στη διοικηση & οικονομία

```
public enum Department
{
    [Display(Name = "Διοίκησης επιχειρήσεων")]
    DE,
    [Display(Name = "Εφαρμογών πληροφορικής στη διοίκηση & οικονομία")]
    EPDO
}
```

Για να διασφαλίσουμε πως η πληροφορία θα μεταφερθεί σωστά μέσω του Entity Framework και θα δημιουργηθεί σωστά ο πίνακας στην βάση, θα κάνουμε χρήση της βιβλιοθήκης Data Annotations που είδαμε σε προηγούμενο κεφάλαιο. Το Entity Framework είναι ικανό να αναγνωρίσει πως το property 'Id' της Classς Student θα αποτελεί την στήλη κλειδί του πίνακα που θα δημιουργηθεί, από το όνομα του property, διαφορετικά μπορούμε να ορίσουμε ποιο είναι το πρωτεύων κλειδί με τη χρήστη κάποιου Data Annotation attribute. Εμείς θα χρησιμοποιήσουμε κάποιο attribute που αφορά τον τρόπο εμφάνισης της πληροφορίας στο UI, ενώ για παράδειγμα στο property 'FullName' το οποίο δεν θέλουμε να μεταφερθεί στην βάση σαν νέα στήλη καθώς η πληροφορία αυτή υπάρχει ήδη, χρησιμοποιούμε το attribute [NotMapped]. Πλέον η Class 'Student' διαμορφώνεται ως εξής:

```
public Class Student
{
    public int Id { get; set; }

    [Display(Name = "Όνομα")]
    public string FirstName { get; set; }

    [Display(Name = "Επώνυμο")]
    public string LastName { get; set; }

    [Display(Name = "Αριθμός μητρώου")]
    public string AM { get; set; }
```

```
    [Display(Name = "Τμήμα")]
    public Department Department { get; set; }

    [Display(Name = "Ομάδα")]
    public int? TeamId { get; set; }
    public virtual Team Team { get; set; }

    public string UserId { get; set; }
    public ApplicationUser User { get; set; }

    [NotMapped]
    public string FullName => this.FirstName + " " + this.LastName;
}
```

Συνεχίζουμε με την δημιουργία του μοντέλου της ομάδας. Για κάθε ομάδα θέλουμε να αποθηκεύουμε :

- Έναν μοναδικό αύξοντα ακέραιο αριθμό ID.
- Την ημερομηνία που δημιουργήθηκε η συγκεκριμένη ομάδα.
- Τα μέλη της ομάδας δηλαδή μια λίστα με σπουδαστές.
- Τις προτιμίσεις της ομάδας δηλαδή μια λίστα με τα θέματα πτυχιακών εργασιών που έχει επιλέξει.
- Τις προτάσεις που ενδεχομένως να έχει κάνει η ομάδα.
- Εάν έχει αναλάβει θέμα, δηλαδή ενα πεδίο τύπου bool, το οποίο μας υποδεικνύει εάν στην ομάδα έχει ανατεθεί θέμα ή οχι.
- Εάν η ομάδα έχει οριστικοποιηθεί ή μπορεί να τροποποιηθεί.

Συνεπώς η Class που προκύπτει είναι η παρακάτω :

```
public Class Team
{
    public int Id { get; set; }
    [Display(Name = "Ημερομηνία δημιουργίας")]
    public DateTime Created { get; set; } = DateTime.Now;
    public virtual ICollection<Student> Students { get; set; }
    public virtual ICollection<Project> Projects { get; set; }
    public virtual ICollection<ProjectRequest> ProjectRequests { get; set; }
    [Display(Name = "Έχει ανατεθεί θέμα")]
    public bool HasAssignment { get; set; }
    [Display(Name = "Είναι υπο διαμόρφωση")]
    public bool IsEditable { get; set; } = true;
}
```

Ο λόγος που δημιουργήσαμε το τελευταίο property είναι καθώς θέλαμε να επιτρέψουμε τις αλλαγές στην ομάδα και στα μέλη της εάν το επιθυμούν, μέχρι να διαμορφωθεί και όσο δεν της έχει ανατεθεί κάποιο θέμα. Όταν ανατεθεί θέμα στην ομάδα δεν θα πρέπει να δίνεται η δυνατότητα να γίνει κάποια αλλαγή, οπότε μετά την ανάθεση το πεδίο 'IsEditable' θα ισούται με 'False'. Όπως θα προσέξετε χρησιμοποιούμε το attribute [Display(Name = "")] όπου ορίζουμε το όνομα της συγκεκριμένης στήλης αργότερα στην σελίδα μας.

Την ίδια διαδικασία ακολουθούμε και για να ορίσουμε το μοντέλο για το θέμα της πτυχιακής εργασίας. Κάθε πρόταση θέματος πτυχιακής εργασίας αποτελείται από :

- Έναν μοναδικό αύξοντα ακέραιο αριθμό ID που θα είναι και το πεδίο κλειδί του πίνακα.
- Έναν τίτλο.
- Μια σύντομη περιγραφή.
- Εαν είναι διαθέσιμο ή οχι.
- Τον επιβλέπων καθηγητή.
- Τις ομάδες έχουν επιλέξει το συγκεκριμένο θέμα.
- Εαν είναι επεξεργάσιμο η όχι.

Ο λόγος που υπάρχει το τελευταίο property είναι διότι δεν θέλουμε να επιτρέψουμε σε κανέναν χρήστη της εφαρμογής να επεξεργαστεί κάποιο θέμα της εφαρμογής όταν αυτό ανατεθεί σε κάποια ομάδα. Η Class 'Project' διαμορφώνεται ως εξής :

```
public Class Project
{
    public Project()
    {
        this.Teams = new HashSet<Team>();
    }
    public int Id { get; set; }
    [MinLength(1, ErrorMessage = "Ο {0} θα πρέπει να αποτελείται από τουλάχιστον
{1} χαρακτήρες")]
    [MaxLength(80, ErrorMessage = "Ο {0} δεν πρέπει να υπερβαίνει τους {1}
χαρακτήρες")]
    [Display(Name = "Τίτλος")]
    public string Title { get; set; }
    [Display(Name = "Περιγραφή")]
    public string Description { get; set; }
    [Display(Name = "Δημοσίευση")]
    public bool Available { get; set; } = false;
    public int? ProfessorId { get; set; }
    public Professor Professor { get; set; }
    public virtual ICollection<Team> Teams { get; set; }
    public bool IsEditable { get; set; } = true;
}
```

Αυτό που έχει αξία να σταθούμε στο συγκεκριμένο σημείο, και είναι κάτι που αναφέραμε και σε προηγούμενο κεφάλαιο, είναι στο πώς με την χρήση attributes μπορούμε να κάνουμε data validation. Συγκεκριμένα, για το property 'Title' ελέγχουμε εάν ο τίτλος αποτελείται από τουλάχιστον 1 και το πολύ 80 χαρακτήρες, ενώ ορίζουμε και τα αντίστοιχα μνήματα σε περίπτωση που το κριτήριο δεν ικανοποιείται.

Με την ίδια μεθοδολογία δημιουργούμε την Class για τον διδάσκων, η οποία είναι παρόμοια με του σπουδαστή, με ελάχιστες διαφορές αφού ο καθηγητής δεν θα ανήκει σε κάποια ομάδα. Συγκεκριμένα ο Καθηγητής αποτελείται από :

- Εναν μοναδικό αριθμό.
- Το όνομα και το επώνυμό του.
- Τον αριθμό μητρώου.
- Τα θέματα που έχει προτείνει.
- Τον μοναδικό αριθμό χρήστη (user Id).

```
public Class Professor
 {
     public int Id { get; set; }
     [Display(Name = "Όνομα")]
     public string FirstName { get; set; }
     [Display(Name = "Επώνυμο")]
     public string LastName { get; set; }
     [Display(Name = "Αριθμός μητρώου")]
     public string AM { get; set; }
     public virtual ICollection<Project> Projects { get; set; }
     public string UserId { get; set; }
     public ApplicationUser User { get; set; }
     [NotMapped]
     public string FullName => this.FirstName + " " + this.LastName;
 }
```

Όπως προαναφέραμε, μια ομάδα έχει το δικαίωμα να προτείνει εκείνη κάποιο θέμα, αποστέλλοντας ένα αίτημα στον καθηγητή που έχει επιλέξει, μέσω της εφαρμογής. Αναλύοντας την συγκεκριμένη ανάγκη, αυτό που χρειαζόμαστε είναι ένας ενδιάμεσος πίνακας στον οποίο θα αποθηκεύουμε πληροφορίες για :

- Το θέμα που δημιουργήθηκε.
- Από ποια ομάδα δημιουργήθηκε.
- Τον καθηγητή στον οποίο αναφαίρεται η πρότασηε ποιον καθηγητή αναφέρεται.
- Εάν έχει γίνει αποδεκτό από τον εκάστοτε καθηγητή η όχι.

Με βάση τα παραπάνω η Class που προκύπτει είναι η εξής :

```
public Class ProjectRequest
```

```
{
    public int Id { get; set; }
    public int? ProjectId { get; set; }
    public virtual Project Project { get; set; }
    public int? TeamId { get; set; }
    public virtual Team Team { get; set; }
    public int? ProfessorId { get; set; }
    public virtual Professor Professor { get; set; }
    public bool Accepted { get; set; } = false;
}
```

Τέλος, για κάθε ανάθεση που πραγματοποιείται αυτό που μας ενδιαφέρει να κρατήσουμε είναι :

- Το θέμα που ανατέθηκε.
- Την ομάδα στο οποίο γίνεται ανάθεση.
- Τον τύπο της ανάθεσης (απ'ευθείας, πρόταση της ομάδας ή κλήρωση).
- Την ημερομηνία που έγινε η ανάθεση.
- ανατέθηκε σε ποια ομάδα.

Απόρροια των παραπάνω αναγκών είναι η δημιουργία της παρακάτω Class.

```
public Class Assignment
{
    [Key]
    [Column(Order = 1)]
    public int TeamId { get; set; }
    [Key]
    [Column(Order = 2)]
    public int ProjectId { get; set; }
    public Team Team { get; set; }
    public Project Project { get; set; }
    [Display(Name = "Τύπος ανάθεσης")]
    public AssignmentType AssignmentType { get; set; }
    [Display(Name = "Ημερομηνία ανάθεσης")]
    public DateTime DateOfAssignment { get; set; } = DateTime.Now;
}
```

Οι κλάσεις που προαναφέραμε αποτελούν τις βασικές οντότητες της εφαρμογής μας, και σχετίζονται με τα δεδομένα που θέλουμε να κρατήσουμε. Είναι οι κλάσεις, βάσει των οποίων θα κατασκευαστούν οι πίνακες στην βάση δεδομένων.

2.6 Κατασκευή της βάσης δεδομένων

Στο στην προηγούμενη ενότητα δημιουργήσουμε τις κλάσεις για τις βασικές οντότητες της εφαρμογής μας. Οι κλάσεις αυτές θα υποδείξουν στο Entity Framework τον τρόπο που θα κατασκευαστούν οι πίνακες αλλά και οι μεταξύ τους σχέσεις στην βάση δεδομένων. Το project στο οποίο θα εργαστούμε προς το παρόν είναι το Thesis.Database όπου θα δημιουργήσουμε μια

Class μέσω της οποίας η εφαρμογή μας θα επικοινωνεί με την βάση δεδομένων. Μέσω της συγκεκριμένης Class θα ορίσουμε για ποιά οντότητα, ποια Class δηλαδή θέλουμε να δημιουργηθεί ο αντιστοιχος πίνακας στην βάση δεδομένων.

Ξακινάμε με την δημιουργία της Class ApplicationDbContect η οποία κληρονομεί την IdentityDbContext<ApplicationUser> και στον κατασκευαστή της θα ορίσουμε τον τρόπο που θα κατασκευαστεί η base Class δηλαδή η μητέρα κλάση την οποία κληρωνομούμε, που στην προκειμένη περίπτωση είναι η IdentityDbContext. Παραμετρικά ορίζουμε το όνομα που που αντιστοιχεί με το connection string της βάσης μας.

```
public Class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public ApplicationDbContext()
        : base("localDB", throwIfV1Schema: false)
    {
    }

    public static ApplicationDbContext Create()
    {
        return new ApplicationDbContext();
    }

    public DbSet<Student> Students { get; set; }
    public DbSet<Assignment> Assignments { get; set; }
    public DbSet<Project> Projects { get; set; }
    public DbSet<Team> Teams { get; set; }
    public DbSet<Professor> Professors { get; set; }
    public DbSet<ProjectRequest> ProjectRequests { get; set; }
}
```

Στο project Thesis.Database υπάρχει το xml αρχείο App.config όπου υπάρχουν πληροφορίες και ρυθμίσεις για το project. Μεταξύ άλλων υπάρχει και η ρύθμιση που περιέχει πληροφορίες για την σύνδεση με την βάση δεδομένων, και συγκεκριμένα τα Connection Strings. Κάθε Connection String διακρίνεται από ένα όνομα, στην περίπτωση μας “localDB” το οποίο connection string περιέχει πληροφορίες απαραίτητες για την επικοινωνία με την βάση.

```
  <connectionStrings>
    <add
name="localDb"
connectionString="
   Data Source=DESKTOP-F5QJKFQ\SQLEXPRESS;Initial Catalog=Thesis;
   Integrated
   Security=True;
   Connect Timeout=30;
   Encrypt=False;
   TrustServerCertificate=False;
```

```
   ApplicationIntent=ReadWrite;
   MultiSubnetFailover=False"
providerName="System.Data.SqlClient" />
  </connectionStrings>
```

Στο Visual Studio αφού ορίσουμε σαν default project το Thesis.Database από το παράθυρο Package Manager Console δίνουμε διαδοχικά τις εντολές.

enable-migrations

Μέσω της συγκεκριμένης εντολής δημιουργείται η Configuration Class

```
internal sealed Class Configuration :
DbMigrationsConfiguration<ApplicationDbContext>
{
    public Configuration()
    {
        AutomaticMigrationsEnabled = false;
    }
}
```

Στην Configuration Class και μέσω της μεθόδου Seed() μπορούμε να καταχωρήσουμε κάποιες εγγραφές στην βάση, όταν αυτή κατασκευαστεί. Παρακάτω βλέπουμε την δημιουργία του ρόλου ‘Admin’ ενώ αντοίστιχα θα δημιουργήσουμε και τους ρόλους ‘Professor’ και ‘Student.

```
if (!context.Roles.Any(x => x.Name == "Admin"))
{
    var _store = new RoleStore<IdentityRole>(context);
    var _manager = new RoleManager<IdentityRole>(_store);
    var _role = new IdentityRole { Name = "Admin" };
    _manager.Create(_role);
}
```

Στη συνέχεια δημιουργούμε έναν χρήστη και του αναθέτουμε τον ρόλο ‘Admin’ που δημιουργήσαμε μόλις, δηλαδή του διαχειριστή της εφαρμογής.

```
if (!context.Users.Any(x => x.UserName == "admin@mail.com"))
{
    var user = new ApplicationUser()
    {
        UserName = "admin@mail.com",
        Email = "admin@mail.com",
        PasswordHash = PasswordHash.HashPassword("Admin1!")
    };
    manager.Create(user);
    manager.AddToRole(user.Id, "Admin");
}
```

Επεξηγώντας τον κώδικα που βλέπουμε παραπάνω, αυτό που γίνεται στην μέθοδο Seed() στην προκειμένη περίπτωση είναι να ελέγξουμε τους χρήστες και εαν δεν υπάρχει χρήστης όπου το e-mail του είναι 'admin@mail.com' να δημιουγήσουμε εναν ApplicationUser, να ορίσουμε το UserName του το E-mail (ίδιο με αυτό που κάναμε την αναζήτηση) καθώς και εναν κωδικό ο οποίος θα κρυπτογραφηθεί. Στη συνέχεια αποθηκεύουμε τον χρήστη που μόλις δημιουργήσαμε ενώ του αναθέτουμε και τον ρόλο 'Admin'.

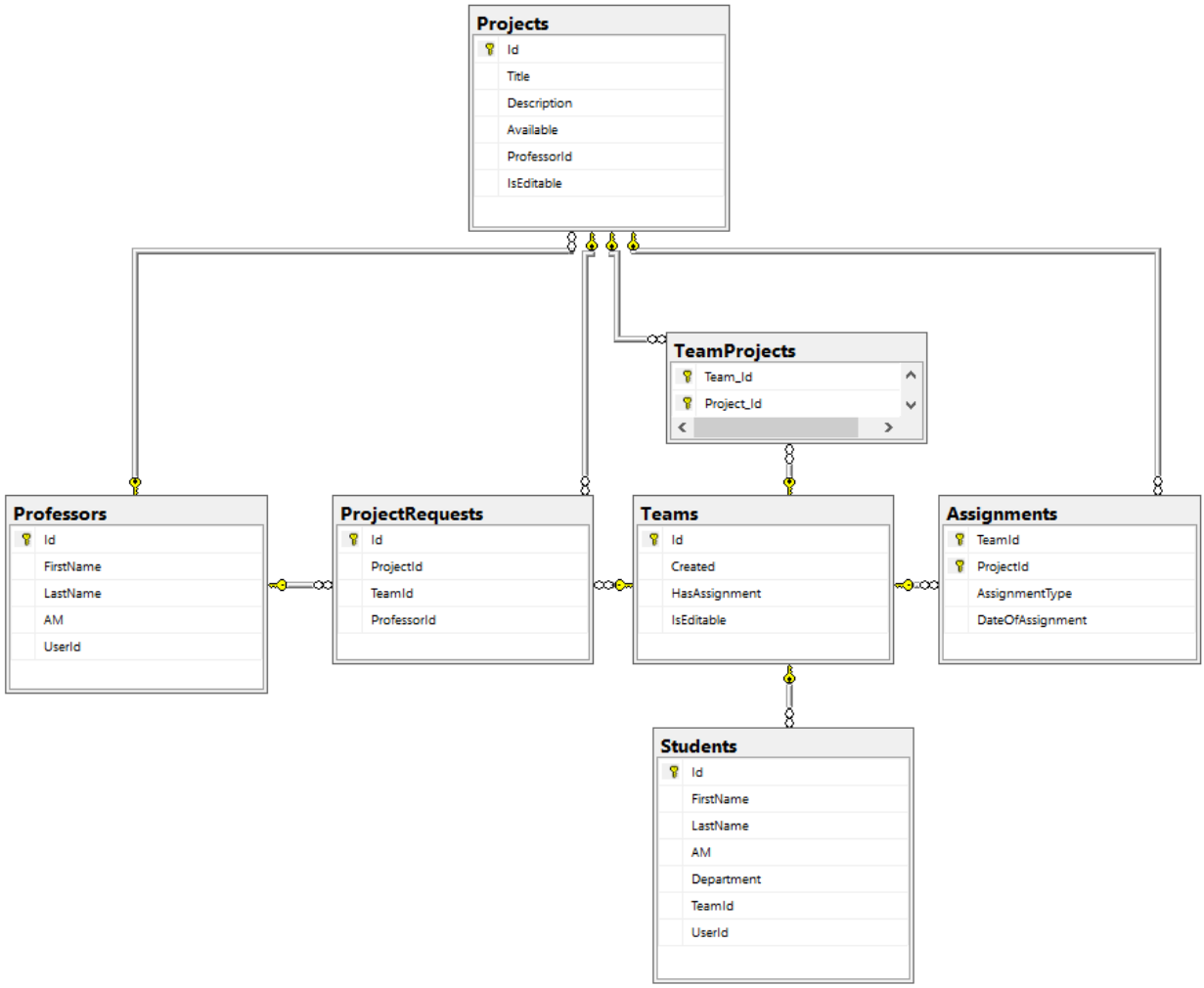
add-migration

Μόλις δώσουμε την εντολή 'add-migration' θα ζητηθεί να δώσουμε ενα όνομα που αποτελεί μια περιγραφή για να μας είναι πιο έυκολο να γνωρίζουμε τι περιείχε το συγκεκριμένο migration. πχ Initial-migration. Όταν η εντολή ολοκληρωθεί θα δημιουργηθεί μια νέα Class όπου περιέχει δύο μεθόδους την Up() και την Down(). Σε κάθε μια απο αυτες τις μεθόδους υπάρχει κώδικας C# που παράχθηκε αυτόματα, με βάση τις κλάσεις που εμείς δημιουργήσαμε και δηλώσαμε στην ApplicationDbContext. Στην μέθοδο Up() υπάρχουν οι εντολές που θα πρέπει να τρέξουν στην βάση δεδομένων (το όνομα της οποίας έχει οριστεί στο connection string) και να δημιουργηθούν οι αντίστιχοι πίνακες με τα σχετικά πεδία τους. Στην μέθοδο Down() υπάρχει κώδικας που θα τρέξει σε περίπτωση που δεν μπορέσει να τρέξει ή δεν ολοκληρωθούν σωστά οι εντολές που υπάρχουν στην Up().

update-database

Είναι η εντολή που θα εκτελέσει όσα περιγράφονται στίς migration Classes, και θα ενημερώσει την βάση δεδομένων.

Αποτέλεσμα των παραπάνω ενεργειών είναι η δημιουργία της βάσης δεδομένων, το διάγραμμα της οποίας βλέπουμε παρακάτω.



Εικόνα 2 – Διάγραμμα συσχετίσεων πινάκων της βάσης δεδομένων

Στο διάγραμμα κατ' επιλογή, βλέπουμε μόνο τους πίνακες για τους οποίους εμείς δημιουργήσαμε τις σχετικές κλάσεις. Ταυτόχρονα δημιουργήθηκαν και άλλοι πίνακες που αποθηκεύουν πληροφορίες για τους χρήστες, τους ρόλους κ.λ.π. τα οποία προσφέρονται απο το ASP.Net Framework γι αυτό και δεν κάνουμε εκτενή αναφορά.

2.7 Repository pattern

Το Repository Pattern είναι ένα design pattern, μια μεθοδολογία δηλαδή που γράφουμε και οργανώνουμε τον κώδικά μας. Πρόκειται για ένα αποθετήριο που μεσολαβεί ανάμεσα στη λογική της εφαρμογής μας ή αλλιώς το business logic, και των αντικειμένων που αναπαριστούν δεδομένα της βάσης μας. Μέσω του Entity Framework μπορούμε να ανακτήσουμε από την βάση μια εγγραφή ή ένα σύνολο με εγγραφές, τα οποία θα συμπεριφερθούν ως αντικείμενα στην μνήμη μας. Έτσι μπορούμε να τροποποιήσουμε, να δημιουργήσουμε ή διαγράψουμε αντικείμενα και στη συνέχεια οι αλλαγές αυτές να μεταφερθούν και στην βάση δεδομένων. Μέσω του Repository Pattern πετυχαίνουμε ενθυλάκωση, μιας και η λογική της εφαρμογής, οι μέθοδοι

δηλαδή που διαχειριζόμαστε τα δεδομένα, δεν επικοινωνεί άμεσα με την βάση αλλά χρησιμοποιώντας ένα ενδιάμεσο συνδετικό επίπεδο. Τα βήματα που ακολουθήσαμε για την υλοποίηση του Repository Pattern περιγράφονται παρακάτω.

Για κάθε Class που αντίστοιχος πίνακας υπάρχει και στην βάση μας, δημιουργήσαμε μεθόδους για να διαβάζουμε, να καταχωρούμε, να επεξεργαζόμαστε ή να διαγράφουμε δεδομένα. Η διαδικασία θα ήταν ίδια για κάθε Class με μόνη διαφορά την ίδια την Class κάθε φορά. Με τον τρόπο αυτό όμως έχουμε καταπατήσει μια αρχή του αντικειμενοστραφούς προγραμματισμού, αφού θα επαναλάβουμε κοινό κώδικα, αλλάζοντας μόνο τον τύπο. Συνεπώς μια καλύτερη επιλογή θα ήταν να είχαμε μια μόνο Class οι μέθοδοι της οποίας θα λειτουργούσαν για όλους τους τύπους αντικειμένων που χρειάζεται να αλληλοεπιδράσουν με την βάση, και η λύση αυτή περιγράφει το Repository Pattern ή Unit Of Work.

Όπως θα δούμε στους Controllers παρακάτω, θα χρειαστεί να ανακτούμε δεδομένα από την βάση μας. Η διαδικασία αυτή θα γίνεται πιο εύκολα και γρήγορα χάρη στο Unit Of Work object αφού μέσω ενός μόνο αντικειμένου, θα έχουμε πρόσβαση σε όλους σχεδόν τους πίνακες της βάσης. Πρακτικά , σε κάθε Controller θα δημιουργούμε κάθε φορά ένα αντικείμενο τύπου UnitOfWork.

Εδώ προκύπτει ένα νέο πρόβλημα και αφορά τη σχέση εξάρτησης που έχει ο εκάστοτε Controller με το αντικείμενο UnitOfWork. Σκεφτείτε εάν θέλουμε να αντικαταστήσουμε το αντικείμενο τύπου Unit Of Work με κάποιο άλλο, θα έπρεπε να βρούμε και να κάνουμε την αλλαγή σε κάθε Controller ξεχωριστά. Το πρόβλημα αυτό έρχεται να επιλύσει μια τεχνική που ονομάζεται Dependency Injection.

2.8 Dependency Injection

Το Dependency Injection είναι μια τεχνική στην οποία ένα αντικείμενο λαμβάνει κατα την κατασκευή του, άλλα αντικείμενα από τα οποία εξαρτάται. Αυτά τα άλλα αντικείμενα ονομάζονται εξαρτήσεις (dependencies), το αντικείμενο λήψης ονομάζεται πελάτης (client) και το αντικείμενο που έχει περάσει, δηλαδή, "εγχέεται" ονομάζεται υπηρεσία (service). Ο όρος injection που στα ελληνικά σημαίνει "ένεση" αναφέρεται στη μετάβαση μιας εξάρτησης, μιας υπηρεσίας στο αντικείμενο που θα το χρησιμοποιούσε.Ο σκοπός του Dependency Injection ή όπως μεταφράζεται στα ελληνικά ‘έγχυσης εξάρτησης’ είναι να επιτευχθεί ο διαχωρισμός της κατασκευής και της χρήσης αντικειμένων. Αυτό μπορεί να αυξήσει την αναγνωσιμότητα και την επαναχρησιμοποίηση κώδικα.

Το Dependency Injection είναι μια μορφή της ευρύτερης τεχνικής αντιστροφής του ελέγχου. Ένας client που θέλει να καλέσει ορισμένα services δεν πρέπει να γνωρίζει πώς να κατασκευάσει αυτά τα services. Αντί αυτού, ο client μεταβιβάζει την ευθύνη παροχής των υπηρεσιών του σε εξωτερικό κώδικα. Δεν επιτρέπεται στον client να καλέσει τον κωδικό του εγχυτήρα. Ο εγχυτήρας στη συνέχεια εγχέει (περνά) τις υπηρεσίες (services) στον πελάτη (client) που μπορεί να υπάρχουν ήδη ή μπορεί επίσης να κατασκευαστούν από τον εγχυτήρα. Ο πελάτης (client) στη συνέχεια χρησιμοποιεί τις υπηρεσίες (services). Αυτό σημαίνει ότι ο

πελάτης δεν χρειάζεται να γνωρίζει για τον εγχυτήρα, πώς να κατασκευάζει τις υπηρεσίες ή ακόμη και ποιες πραγματικές υπηρεσίες χρησιμοποιεί. Ο πελάτης πρέπει να γνωρίζει μόνο για τις εγγενείς διεπαφές των υπηρεσιών, επειδή αυτές καθορίζουν τον τρόπο με τον οποίο ο πελάτης μπορεί να χρησιμοποιεί τις υπηρεσίες. Αυτό διαχωρίζει την ευθύνη της "χρήσης" από την ευθύνη της "κατασκευής".

2.9 Εφαρμόζουμε το Dependency Injection

Για να γίνει ευκολότερα κατανοητό, θα δούμε ένα παράδειγμα και συγκεκριμένα πως εμείς υλοποιήσαμε το Dependency Injection.

Για την υλοποίηση της εργασίας εφαρμόσαμε το Dependency Injection με τη χρήση ενός εργαλείου, του Autofac. Αρχικά εγκαταστήσαμε το σχετικό package μέσω του Visual Studio και NuGet package manager, στο project μας και στη συνέχεια δημιουργήσαμε την Class ContainerConfig όπου ουσιαστικά δηλώνουμε τον τύπου αντικειμένου που θέλουμε να δημιουργείται όταν χρησιμοποιείται συγκεκριμένο interface. Όπως βλέπουμε και στον κώδικα παρακάτω δηλώνουμε ή πιο σωστά κάναμε Register τον τύπο UnitOfWork ως IUnitOfWork.

```
private static void SetAutofacContainer()
{
    var builder = new ContainerBuilder();
    builder.RegisterControllers(Assembly.GetExecutingAssembly());

    builder.RegisterType<ApplicationDbContext>();
    builder.RegisterType<UnitOfWork>().As<IUnitOfWork>();

    IContainer container = builder.Build();
    DependencyResolver.SetResolver(new AutofacDependencyResolver(container));
}
```

Πλέον όπως θα δούμε και στους Controllers, δηλώνουμε μια μεταβλητή τύπου IUnitOfWork, και στον κατασκευαστή της Class, παραμετρικά περνάμε το αντικείμενο από το οποίο εξαρτάται η Class μάς. Αυτό ακριβώς συμβαίνει και παρακάτω, στον κατασκευαστή της Classς η παράμετρος είναι IUnitOfWork το οποίο μας επιστρέφει ένα αντικείμενο UnitOfWork.

```
public Class ProfessorController : Controller
{
    private IUnitOfWork _repository;

    public ProfessorController(IUnitOfWork unitOfWork)
    {
        _repository = unitOfWork;
    }
}
```

2.10 Controllers

Σε συνέχεια της προηγούμενης ενότητας που μιλήσαμε για το Dependency Injection, σε κάθε Controller που χρειάζεται να επικοινωνεί με την βάση χρειάζεται να δημιουργούμε ένα αντικείμενο τύπου IUnitOfWork. Όπως βλέπουμε και στο παρακάτω απόσπασμα κώδικα, δηλώνουμε ένα private property τύπου IUnitOfWork. Αυτό θα μπορούσε να είναι οποιοδήποτε αντικείμενο υλοποιεί το συγκεκριμένο Interface. Τοποθετήσαμε λοιπόν τον παρακάτω κώδικα σε κάθε Controller.

```
private IUnitOfWork _repository;

public DemoController(IUnitOfWork unitOfWork)
{
    _repository = unitOfWork;
}
```

2.10.1 Home Controller

Μετά από επιτυχή σύνδεση στην εφαρμογή ο χρήστης οδηγείται στον HomeController και στην μέθοδο Index. Στο σημείο αυτό γίνεται απλώς ένας έλεγχος στον ρόλο του συγκεκριμένου χρήστη και ανακατευθύνεται στον αντίστοιχο αρμόδιο Controller. Εδώ δεν χρειάζεται να κάνουμε κάποιον έλεγχο σε περίπτωση που το αντικείμενο 'User' είναι null διότι το attribute 'Authorize' μας εξασφαλίζει πως ο χρήστης θα φτάσει εδώ μόνο εάν έχει πραγματοποιήσει σύνδεση, και όταν έχει πραγματοποιήσει σύνδεση το αντικείμενο User, δεν είναι null.

```
public Class HomeController : Controller
{
    [Authorize]
    public ActionResult Index()
    {
        if (User.IsInRole("Professor"))
            return RedirectToAction("Index", "Professor");

        if (User.IsInRole("Admin"))
            return RedirectToAction("Index", "Admin");

        return RedirectToAction("Index", "Student");
    }
}
```

2.10.2 Student Controller

Έστω λοιπόν πως ο χρήστης μας έχει τον ρόλο 'Student', τότε θα μεταφερθούμε στον StudentController και την μέθοδο Index. Όπως βλέπουμε και στο παρακάτω απόσπασμα του πηγαίου κώδικα, στην πρώτη σειρά , με το attribute 'Authorize' διασφαλίζουμε πως ο χρήστης όχι μόνο είναι συνδεδεμένος, αλλά έχει τον ρόλο Student. Με βάση τον αριθμό χρήστη (user Id)

θα αναζητήσουμε στην βάση και τον πίνακα με τους σπουδαστές, τον σπουδαστή εκείνο που συσχετίζεται με τον συγκεκριμένο χρήστη.

```
[Authorize(Roles = "Student")]
public ActionResult Index()
{
    string userId = User.Identity.GetUserId();
    Student student = _repository.Students
                .GetAll()
                .Where(x => x.UserId == userId)
                .First();
    return View(student);
}
```

Με την χρήση Linq syntax και Lambda Expression πραγματοποιούμε ενα query ή πιο σωστά φιλτράρισμα, και από το αντικείμενο '_repository', φορτώνουμε στην μνήμη όλους τους σπουδαστές μέσω της μεθόδου 'GetAll()'. Στη συνέχεια με την μέθοδο 'Where()' φιλτράρουμε ώστε να κρατήσουμε τα δεδομένα που μας ενδιαφέρουν. Με την μέθοδο 'First()' θα πάρουμε το πρώτο αντικείμενο το οποίο ικανοποιεί τα κριτήρια, μιας και η μέθοδος 'GetAll()' μας επιστρέφει λίστα με αρκετά αποτελέσματα. Τέλος η μέθοδος Index επιστρέφει ενα 'View' με παράμετρο το αντικείμενο 'student'. Όμως με αυτό τον τρόπο, έχουμε φορτώσει όλα τα δεδομένα του πίνακα 'Students' από την βάση στην μνήμη γεγονός που, ειδικά στην περίπτωση που είχαμε πολλά δεδομένα αυτό θα καθυστερούσε αρκετά. Μια λύση στο πρόβλημά μας θα δούμε στο αμέσως επόμενο.

Στο παρακάτω απόσπασμα του πηγαίου κώδικα βλέπουμε την μέθοδο που μας επιστρέφει όλους τους διαθέσιμους σπουδαστές, εκείνους δηλαδή που δεν ανήκουν σε κάποια ομάδα.Ο τρόπος αυτός είναι λίγο διαφορετικός καθώς ορίζουμε ένα κριτήριο το οποίο θα είναι παράμετρος κατά την ανάκτηση δεδομένων από την βάση. Στην προκειμένη περίπτωση δεν θα λάβουμε όλα τα δεδομένα από την βάση στην μνήμη, αλλά το ερώτημα - κριτήριο θα τρέξει στην βάση δεδομένων, θα γίνει το φιλτράρισμα στην βάση και θα επιστραφούν μόνο όσα δεδομένα πληρούν τα κριτήρια. Η μέθοδος αυτή είναι καλύτερη όσον αφορά την ταχύτητα της εφαρμογής μας καθώς δεσμεύουμε λιγότερη μνήμη.

```
public ActionResult Available()
{
    Expression<Func<Student, bool>> criteria = student => student.TeamId == null;
    var students = _repository.Students.FindByEntity(criteria);
    return View(students);
}
```

2.10.3 Professor Controller

Στις μεθόδους που υπάρχουν στην Class ProfessorController, θέλουμε να έχουν πρόσβαση αυστηρά χρήστες που έχουν συνδεθεί στην εφαρμογή και μάλιστα με τον ρόλο

‘Professor’. Την περίπτωση αυτή καλύπτουμε μέσω του attribute ‘Authorize’ που παρέχεται από την βιβλιοθήκη System.Web.Mvc, όπου όχι μόνο εξασφαλίζουμε πως ο χρήστης έχει συνδεθεί αλλά συσχετίζεται και με συγκεκριμένο ρόλο.

```
[Authorize(Roles = "Professor")]
public Class ProfessorController : Controller
```

Η μέθοδος Index, τον κώδικα της οποίας βλέπουμε παρακάτω, αυτό που κάνει είναι να ανακτήσει τα στοιχεία του καθηγητή, με βάση τον χρήστη που είναι συνδεδεμένος, ενώ στην μεταβλητή ‘ViewBag.Requests’ αποθηκεύουμε τον αριθμό των αιτημάτων που έχει ο συγκεκριμένος καθηγητής.

```
public ActionResult Index()
{
    var professor = _repository.Professors
                .GetAll()
                .Where(s => s.UserId == User.Identity.GetUserId())
                .First();

    ViewBag.Requests = _repository.ProjectRequests
                .GetAll()
                .Where(p => p.ProfessorId == professor.Id)
                .Count();
    return View(professor);
}
```

2.10.4 Team Controller

Όσον αφορά την ομάδα, θα δούμε αρχικά τον Controller που είναι υπεύθυνος για την δημιουργία της ομάδας από κάποιο σπουδαστή. Όπως βλέπουμε παρακάτω, αρχικά ανακτούμε από την βάση τα στοιχεία του σπουδαστή. Γίνεται ένας έλεγχος για να δούμε εαν ο σπουδαστής ανήκει ήδη σε κάποια ομάδα. Εαν ναι, τότε δεν μπορεί να δημιουργήσει νέα οπότε επιστρέφει στην αρχική σελίδα όπου θα εμφανιστεί σχετικό μήνυμα. Στην περίπτωση που ο σπουδαστής δεν ανήκει σε κάποια ομάδα, μια νέα ομάδα θα δημιουργηθεί και ο σπουδαστής αυτόματα θα προστεθεί ως μέλος.

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create([Bind(Include = "Id,Created")] Team team)
{
    var student = _repository.Students.GetAll().Where(s => s.UserId ==
User.Identity.GetUserId()).FirstOrDefault();

    if (student.TeamId.HasValue)
        return RedirectToAction("Index", "Student", new
ResultMessage(AlertStatus.danger, "Ανήκεις ήδη σε μια ομάδα!"));
```

```
    if (ModelState.IsValid)
    {
        student.Team = team;
        _repository.Students.Update(student);
        _repository.Students.Save();
    }

    return RedirectToAction("Index", "Student");
}
```

Μια ακόμα μέθοδος που αξίζει να δούμε είναι αυτή με την οποία γίνεται προσθήκη ενός μέλους στην ομάδα. Σαν παράμετρος στην συγκεκριμένη μέθοδος είναι μόνο το studentId, δηλαδή ο κωδικός του χρηστη που θέλουμε να προσθέσουμε, ωστόσο κάνουμε αρκετούς ελέγχους μέχρι το νέο μέλος να προστεθεί στην ομάδα.

Αρχικά ανακτούμε από την βάση τα στοιχεία του σπουδαστή – χρήστη της εφαρμογής. Απο εκεί θα πάρω την πληροφορία για την ομάδα, το ID δηλαδή της ομάδας που πρόκειται να προστεθεί το νέο μέλος. Στη συνέχεια με βάση το ID του σπουδαστή αναζητώ στην βάση τον σπουδαστή που θέλω να προσθέσω στην ομάδα ως νέο μέλος. Εαν το αποτέλεσμα είναι null ή ο σπουδαστής ανήκει ήδη σε άλλη ομάδα, δεν μπορούμε να προχωρήσουμε οπότε επιστρέφουμε στην αρχική οθόνη οπού ο χρήστης θα δει το σχετικό μήνυμα.

Προχωράμε με τον έλεγχο του συνδεδεμένου στην εφαρμογή σπουδαστή, εάν ανήκει σε ομάδα. Εαν δεν ανήκει, καμία ενέργεια δεν θα γίνει και σχετικό μύνημα θα εμφανιστεί στον χρήστη. Ο λόγος που κάνουμε αυτό τον έλεγχο είναι για να εξασφαλίσουμε την ορθή λειτουργία της εφαρμογής μας καθώς ενδεχομένως κάποιος σπουδαστής που δεν ανήκει σε κάποια ομάδα, πιθανόν καταλλάθος και πληκτρολογώντας συγκεκριμένη διεύθυνση στον browser του (πχ μέσω ιστορικού, ή απο κάποιο σελειδοδείκτη κλπ) να βρεθεί σε αυτόν τον Controller.

Αφού έχουν ολοκληρωθεί οι παραπάνω έλεγχοι ανακτούμε τα στοιχεία της ομάδας από την βάση, και ελέγχουμε εάν μπορεί η ομάδα να τροποποιηθεί και εφ' όσον ο αριθμός μελών δεν ξεπερνάει το επιτρεπτό όριο, τότε το νέο μέλος προστίθεται στην ομάδα.

```
public ActionResult Add(int id)
{
    var student = _repository.Students.GetAll().Where(s => s.UserId ==
User.Identity.GetUserId()).FirstOrDefault();
    var newMember = _repository.Students.GetById(id);

    if (newMember == null || newMember.TeamId.HasValue)
        return RedirectToAction("Index", "Student", new
ResultMessage(AlertStatus.danger, "Δεν βρέθηκε διαθέσιμος σπουδαστής."));

    if (!student.TeamId.HasValue)
        return RedirectToAction("Index", "Student", new
ResultMessage(AlertStatus.warning, $"Θα πρέπει να ανήκετε σε κάποια ομάδα για να
```

```
προσθέσετε ενα νέο μέλος."));

    var currentTeam = _repository.Teams.GetById(student.TeamId);

    if (currentTeam.IsEditable)
    {
        currentTeam.Students.Add(newMember);
        _repository.Teams.Save();
        return RedirectToAction("Index", "Student", new
ResultMessage(AlertStatus.success, $"Ο {newMember.FullName} προστέθηκε στην
ομάδα!"));
    }

    return RedirectToAction("Index", "Student", new
ResultMessage(AlertStatus.danger, $"Η ομάδα έχει οριστικοποιηθεί!"));
}
```

2.10.5 Project Controller

Για την οντότητα 'Project' δηλαδή το θέμα πτυχιακής εργασίας, δημιουργήσαμε τις μεθόδους που είναι απαραίτητες για δημιουργία, επεξεργασία, ενημέρωση και διαγραφή. Αυτό που αξίζει να δούμε αναλυτικά σε αυτό το σημείο είναι η μέθοδος που είναι υπεύθυνη για την πρόταση ενός project από τους σπουδαστές καθώς και τις ενέργειες που κάνει ο καθηγητής. Όπως βλέπουμε παρακάτω, στον συγκεκριμένο Controller η πρόσβαση επιτρέπεται μόνο σε συνδεδεμένους χρήστες, και μάλιστα με τον ρόλο του σπουδαστή. Στη συνέχεια με το [AllowTeamsToSuggest] attribute, ελέγχουμε εάν επιτρέπεται ή όχι στις ομάδες να προτείνουν θέμα.

Το AllowTeamsToSuggest είναι ένα custom attribute που δημιουργήσαμε. Η επιλογή αυτή ορίζεται από τον διαχειριστή της σελίδας και ανάλογα τις τιμές που υπάρχουν στην βάση επιτρέπεται ή όχι η πρόσβαση στον χρήστη.

```
[Authorize(Roles = "Student")]
[AllowTeamsToSuggest]
public ActionResult Suggest()
{
    ViewBag.ProfessorId = new SelectList(db.Professors, "Id", "FullName");
    return View();
}
```

Όπως βλέπουμε, το ViewBag.ProfessorId είναι μια λίστα η οποία περιέχει δεδομένα από την βάση για τους καθηγητές, και συγκεκριμένα τα πεδία Id και FullName. Η λίστα αυτή θα χρειαστεί στο αντίστοιχο View οπού ο χρήστης καλείται να επιλέξει ποιον καθηγητή προτιμά για να επιβλέψει την εργασία που προτείνει. Το 'FullName' είναι αυτό που θα εμφανίζεται στο στην οθόνη του χρήστη, σε ενα select input, αλλά το ID θα είναι η τιμή της επιλογής του, και υποδυκνύει τον επιβλέπων καθηγητή που αφορά.

Παρακάτω βλέπουμε τον Controller που καλείται όταν ο χρήστης έχει υποβάλει το αίτημα του. Αρχικά ανακτούμε από την βάση τα στοιχεία του καθηγητή που έχει επιλεχθεί να αναλάβει την συγκεκριμένη εργασία. Στη συνέχεια ανακτούμε τα στοιχεία του σπουδαστή, με βάση τα στοιχεία χρήστη που είναι συνδεδεμένος αυτή τη στιγμή στην εφαρμογή. Αφού γίνει έλεγχος οτι ο χρήστης ανήκει σε ομάδα, στη λίστα με τα θέματα που έχει επιλέξει η ομάδα προστίθεται και το θέμα που έχει μόλις προτείνει. Στη συνέχεια δημιουργείται ένα νέο αίτημα που περιέχει πληροφορίες για το θέμα, τον καθηγητή και την ομάδα.

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Suggest([Bind(Include = "Id,Title,Description,ProfessorId")]
Project project)
{
    if (ModelState.IsValid)
    {
        var _professorId = (int)project.ProfessorId;
        var student = _repository.Students.GetAll().Where(s => s.UserId ==
User.Identity.GetUserId()).FirstOrDefault();

        if (student.TeamId.HasValue)
        {
            var team = _repository.Teams.GetById(student.TeamId);
            team.Projects.Add(project);
            project.ProfessorId = null;
            _repository.Projects.Add(project);
            _repository.Projects.Save();

            ProjectRequest projectRequest = new ProjectRequest(_professorId,
project.Id, team.Id);
            _repository.ProjectRequests.Add(projectRequest);
            _repository.ProjectRequests.Save();

            return RedirectToAction("Index", "Student", new
ResultMessage(AlertStatus.success, $"Η πρότασή σας απεστάλη!"));
        }
        else
        {
            return RedirectToAction("Index", "Student", new
ResultMessage(AlertStatus.danger, $"Για να προτείνεται κάποιο θέμα πτυχιακής
εργασίας θα πρέπει να δημιουργήσετε η να ενταχθείτε σε κάποια ομάδα."));
        }


    }
    ViewBag.ProfessorId = new SelectList(db.Professors, "Id", "FullName",
project.ProfessorId);
```

```
    return View(project);
}
```

Τέλος αξίζει να δούμε και τον κώδικα του Controller οπού κάποιος καθηγητής επιλέγει να αποδεχτεί το αίτημα. Πέρα από κάποιους απαραίτητους ελέγχους που εξασφαλίζουν την ορθή λειτουργία της εφαρμογής, όπως για το εάν κάποιο αντικείμενο είναι null, αυτό που μας ενδιαφέρει κυρίως είναι η ομάδα που έκανε το αίτημα να μην έχει αναλάβει κάποιο θέμα στο χρόνο που μεσολάβησε από την δημιουργία του αιτήματος δηλαδή μέχρι την στιγμή απόκρισης του καθηγητή. Εφόσον λοιπόν η ομάδα δεν έχει αναλάβει κάποιο θέμα μέχρι στιγμής, κατά την αποδοχή του αιτήματος από τον καθηγητή το συγκεκριμένο θέμα της εργασίας ανατίθεται στην ομάδα.

```
[Authorize(Roles = "Professor")]
public ActionResult Accepted(int? id)
{
    if (id == null)
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

    var request = _repository.ProjectRequests.GetById(id);
    if (request == null)
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

    var professor = _repository.Professors.GetAll().Where(s => s.UserId ==
User.Identity.GetUserId()).FirstOrDefault();
    if (request.ProfessorId != professor.Id)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }

    var team = _repository.Teams.GetById(request.TeamId);

    if (team.HasAssignment || (team.IsEditable == false))
    {
        return Content("Εχει ήδη ανατεθεί θέμα στην ομάδα. <a
href='/Home/Index/'>Επιστροφή</a>");
    }

    team.HasAssignment = true;
    team.IsEditable = false;
    _repository.Teams.Update(team);
    _repository.Teams.Save();

    var project = _repository.Projects.GetById(request.ProjectId);
    project.ProfessorId = professor.Id;
    project.IsEditable = false;
    request.Accepted = true;
```

```
    _repository.Projects.Update(project);
    _repository.Projects.Save();

    Assignment assignment = new Assignment((int)request.TeamId,
(int)request.ProjectId, AssignmentType.Request);

    _repository.Assignments.Add(assignment);
    _repository.Assignments.Save();

    int noOfRowDeleted = db.Database.ExecuteSqlCommand($"DELETE
FROM[Thesis].[dbo].[TeamProjects] WHERE[Team_Id] = {request.TeamId}");
    _repository.ProjectRequests.Update(request);
    _repository.ProjectRequests.Save();

    return RedirectToAction("MyRequests", "Project");
}
```

2.10.6 Assignment Controller

Επειδή μια ανάθεση μπορεί να γίνει με διάφορους τρόπους, απευθείας, από τον καθηγητή ή ύστερα από κλήρωση, είχαμε την ανάγκη να δημιουργήσουμε μια μοναδική μέθοδο που θα επαναχρησιμοποιείται. Αυτή στην ουσία είναι και η φιλοσοφία του αντικειμενοστραφούς προγραμματισμού. Η μέθοδος αυτή, ο κώδικας της οποίας φαίνεται παρακάτω, λαμβάνει παραμετρικά έναν ακέραιο που αποτελεί το Id της ομάδας, το Id του θέματος πτυχιακής εργασίας, και μια τιμή τύπου AssignmentType που περιγράφει τον τρόπο ανάθεσης. Η ανάθεση θα γίνει μόνο εάν στη συγκεκριμένη ομάδα δεν έχει ήδη ανατεθεί κάποιο θέμα.

```
public void DoAssignment(int teamId, int projectId, AssignmentType assignmentType)
{

    var team = _db.Teams.Find(teamId);
    var project = _db.Projects.Find(projectId);

    if (team.HasAssignment == false)
    {
        team.HasAssignment = true;
        team.IsEditable = false;
        team.Projects = null;
        _db.Entry(team).State = EntityState.Modified;

        project.Available = false;
        project.IsEditable = false;
        _db.Entry(project).State = EntityState.Modified;

        Assignment assignment = new Assignment(team.Id, project.Id,
assignmentType);
        CleanUpTeamsList(assignment.TeamId);
```

```
        _db.Assignments.Add(assignment);

        _db.SaveChanges();

        Log.Info("Το θέμα { project.Id } ανατέθηκε επιτυχώς στην ομάδα { team.Id
}!");
    }
}
```

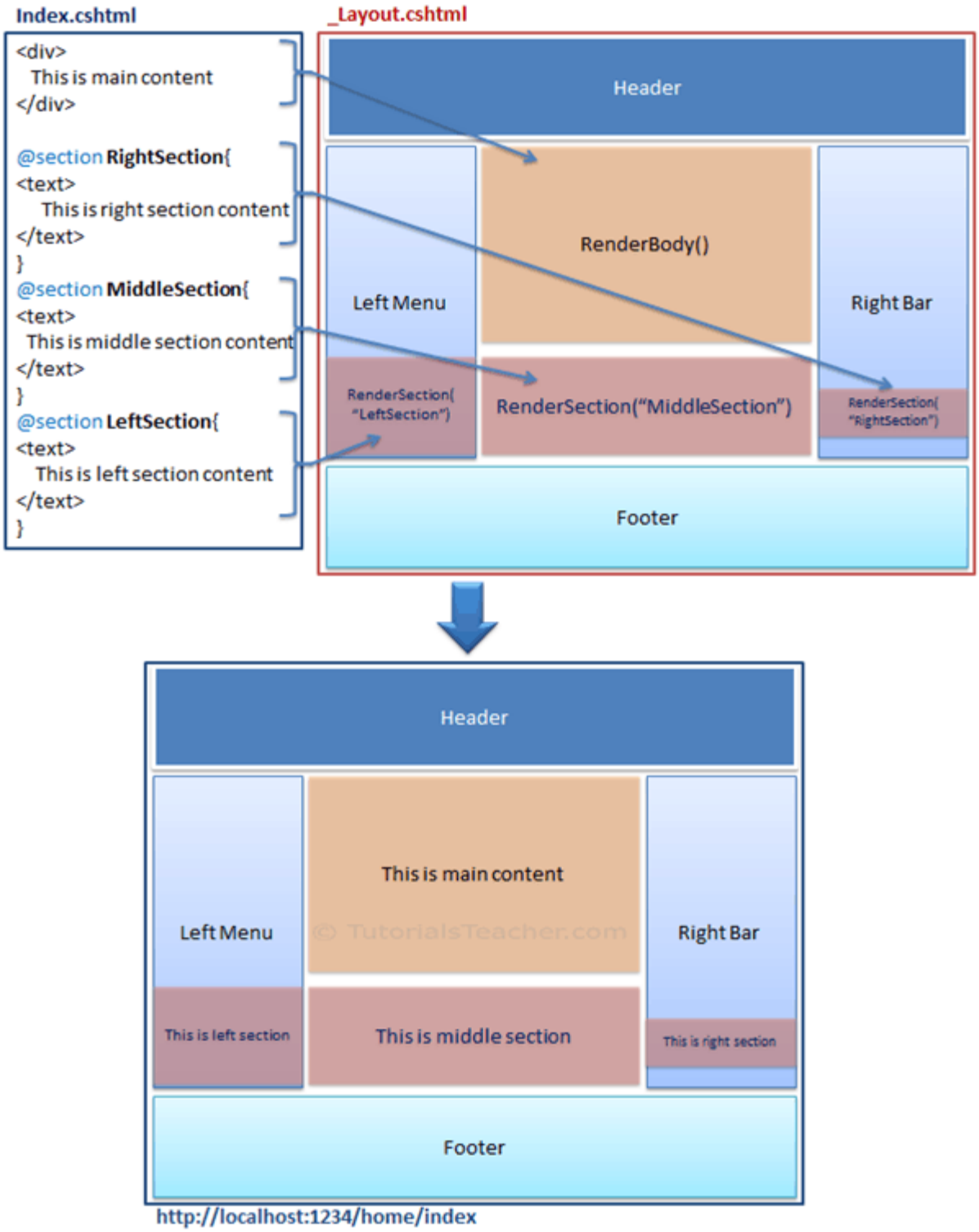
2.11 Views

Τα Views αφορούν το γραφικό περιβάλλον της εφαρμογής μας, τα σημεία που θα αλληλοεπιδράσει ο χρήστης, θα αναπαρασταθούν τα δεδομένα και θα σταλούν νέα αιτήματα στον Controller για να λάβει σαν απάντηση άλλα - νέα δεδομένα. Για την δημιουργία των Views χρειάζεται να κατασκευάσουμε την δομή της σελίδας, με HTML και στη συνέχεια με κάποιους κανόνες CSS να ορίσουμε την εμφάνισή της για καλύτερο αισθητικό αποτέλεσμα. Στην περίπτωση μας θα χρησιμοποιήσουμε έτοιμες CSS κλάσεις της βιβλιοθήκης Bootstrap ακολουθώντας τις οδηγίες που υπάρχουν στο επίσημο Documentation της.

Για την κατασκευή των σελίδων, θα χρησιμοποιήσουμε ένα συντακτικό που ονομάζεται Razor. Ως εκ τουτου τα αρχεία μας θα πρέπει να αποθηκευτούν ως .cshtml και όχι απλως .html. Το Razor δεν είναι μια γλώσσα προγραμματισμού, αλλά μια γλώσσα σύνταξης που χρησιμοποιείται για τη δημιουργία δυναμικών ιστοσελίδων με τις γλώσσες προγραμματισμού C # ή VB.NET.

Οι σελίδες περιέχουν μερικά στοιχεία τα οποία είναι κοινά, όπως για παράδειγμα το head, το footer, ενώ κάποια άλλα παραμένουν ίδια όπως για παράδειγμα τα εξωτερικά CSS ή Javascript αρχεία που καλούν, και ένα μέρος της σελίδας είναι δυναμικό. Για τον λόγο αυτό θα δημιουργήσουμε ένα Layout, το οποίο θα περιέχει όλα εκείνα τα στατικά στοιχεία ενώ θα υπάρχει χώρος που θα ενταχθεί το δυναμικό περιεχόμενο.

Το Layout είναι κοινόχρηστο καθώς θα μπορεί να χρησιμοποιηθεί από όλα τα Views. Προσθέσαμε το head και το footer tag, όπου καλούμε τα απαραίτητα css και javascript αρχεία καθώς και κάποια επιπλέον metadata. Στη συνέχεια καλούμε μια μέθοδο, την RenderBody() η οποία θα καλέσει το εκάστοτε View (child Views), και θα εμφανίσει το περιεχόμενό της.



Εικόνα 3 – Δομικά στοιχεία μιας σελίδας με την χρήση layout.

Στο παρακάτω απόσπασμα βλέπουμε ένα μέρος του layout που δημιουργήσαμε και διαπιστώνουμε πως κάθε σελίδα έχει ένα HTML tag h1 που αποτελεί τον τίτλο της σελίδας,

όπως αυτός ορίζεται κάθε φορά. Στη συνέχεια βλέπουμε με την μέθοδο Html.Partial() καλεί ένα Partial View και συγκεκριμένα το _alert.cshtml.

```
<div Class=" container-fluid">
    <h1 Class="h3 mb-4 text-gray-800">@ViewBag.Title</h1>
    @Html.Partial("~/Views/Shared/_alert.cshtml")
    @RenderBody()
</div>
```

Το PartialView αποτελεί μέρος ενος View, το οποίο μπορεί να επαναχρησιμοποιηθεί απο διάφορες σελίδες, και στην προκειμένη περίπτωση αφορά τον χώρο που εμφανίζεται κάποιο μήνυμα στον χρήστη. Συγκεκριμένα υπάρχει ένα div με id alertPartial και την Class d-none χάρη στην οποία το συγκεκριμένο στοιχείο δεν είναι εμφανές εξ αρχής καθώς θα ορίσουμε εμείς πότε θέλουμε να εμφανίζεται. Υπάρχει επίσης ένα κουμπί ο ρόλος του οποίου είναι να κλείνει το συγκεκριμένο μήνυμα όταν ο χρήστης το επιλέξει και τέλος έναν τίτλο h6 με id alertPartialMessage το οποίο tag είναι κενό.

```
<div Class="alert alert-dismissible d-none shadow" id="alertPartial">
    <button type="button" Class="close" data-dismiss="alert" aria-label="Close">
        <span aria-hidden="true">&times;</span>
    </button>
    <h6 id="alertPartialMessage"></h6>
</div>
```

Το συγκεκριμένο πλαίσιο διαμορφώνεται ανάλογα, με την χρήση JavaScript και μιας τεχνικής που καλλείται DOM manipulation. Οι τιμές των μεταβλητών message και status ορίζονται από το HTTP request οπως βλέπουμε παρακάτω. Εάν η μεταβλητή message δεν είναι κενή, τότε από το div με id alertPartial αφαιρούμε την Class d-none με αποτέλεσμα το συγκεκριμένο στοιχείο πλέον να εμφανιστεί. Στη συνέχεια ανάλογα την τιμή της μεταβλητής status, η οποία μπορεί να είναι ‘success’,’warning’,’danger’ προστίθεται η Class alert-success, alert-warning ή alert-danger και σκοπό έχει απλώς στο να αλλάξει τα χρώματα του συγκεκριμένου πλαισιου. Τέλος στο στοιχείο με id alertPartialMessage πρόσθεσε το κείμενο που περιέχει η μεταβλητή message και αποτελεί το μήνυμα που θέλουμε να διαβάσει ο χρήστης.

```
var status = getUrlParameter('Status');
var message = getUrlParameter('Message');

if (message.length > 0) {
    $("#alertPartial").removeClass("d-none");
    $("#alertPartial").addClass("alert-" + status);
    $("#alertPartialMessage").append(message);
}
$('textarea').val($('textarea').val().trim());
```

Στο παρακάτω απόσπασμα του πηγαίου κώδικα θα δούμε το View για τον διαχειριστή.

Είναι ένα τύπου .cshtml αρχείο το οποίο είναι html ωστόσο έχουμε την δυνατότητα να γράψουμε και κώδικα C#. Το @model είναι το αποτέλεσμα που επιστρέφει η μέθοδος Index() του Controller Admin, και είναι μια λίστα από UserViewModel. Ορίζουμε τον τίτλο ο οποίος όμως θα εμφανιστεί όπως έχει οριστεί στο layout. Στη συνέχεια υπάρχουν κάποιοι υπερ-συνδεσμοι και τέλος ένας πίνακας που εμφανίζει στοιχεία για τους χρήστες. Συγκεκριμένα για κάθε στοιχείο που υπάρχει στο Model, την λίστα δηλαδή με τους χρήστες, δημιούργησε μια νέα γραμμή στον πίνακα, και στην πρώτη στήλη εμφάνισε το username, στην δεύτερη στήλη το email και στην τρίτη στήλη τον ρόλο του χρήστη.

```
@model IEnumerable<Thesis.WebApp.Models.UserViewModel>
@{
    ViewBag.Title = "Αρχική σελίδα διαχειριστή";
}
<div Class="row d-flex justify-content-between">
    <div Class="col">
        <a  href="/Account/RegisterPro"  Class="btn  btn-primary"><i  Class="fa  fa-
plus"></i> Πρόσκληση καθηγητή</a>
        <a   href="/Admin/LockTeams"   Class="btn   btn-primary"><i   Class="fa   fa-
lock"></i> Κλείδωμα ομάδων</a>
    </div>
    <div>
        <a href="/Admin/Run" Class="btn btn-danger"><i Class="fa fa-terminal"></i>
Έναρξη κλήρωσης</a>
    </div>
</div>
<hr />
<div Class="row">
    <div Class="col">
        <table Class="display" id="datatable">
            <thead>
                <tr>
                    <th>Username</th>
                    <th>Email</th>
                    <th>Role</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var user in Model)
                {
                    <tr>
                        <td>@user.Username</td>
                        <td>@user.Email</td>
                        <td>@user.Role</td>
                    </tr>
                }
            </tbody>
        </table>
```

```
    </div>
</div>
```

Χρησιμοποιώντας Razor syntax, μπορούμε να κάνουμε λογικούς ελέγχους στα Views, όπως το παρακάτω παράδειγμα στο View του σπουδαστή. Εδώ ελέγχουμε εάν το πεδίο μας που περιέχει την τιμή για το id της ομάδας του σπουδαστή δεν είναι κενό. Αυτό μας υποδυκνύει πως ο σπουδαστής ανήκει σε κάποια ομάδα, τότε για κάθε σπουδαστή στην συγκεκριμένη ομάδα θέλουμε να εμφανίσουμε τον αριθμό μητρώου και το ονοματεπώνυμό του.

```
if (Model.TeamId.HasValue)
{
    foreach (var student in Model.Team.Students)
    {
        <div>(@student.AM)  @student.FullName</div>
    }
}
```

Στα Views που το μοντέλο είναι λίστα εφαρμόζουμε τον παρακάτω έλεγχο για να επιβεβαιώσουμε πως έχουμε λάβει αποτελέσματα. Σε διαφορετική περίπτωση στην οθόνη του χρήστη εμφάνισε το σχετικό μήνυμα.

```
@if (Model.Any() == false)
{
    <div Class="col-lg-6"><p>Δεν βρέθηκαν δεδομένα.</p></div>
}
```

Ενώ το View είναι κοινό σε όλους τους χρήστες, κάποιες πληροφορίες είναι διαθέσιμες μόνο σε χρήστες με συγκεκριμένο ρόλο, όπως για παράδειγμα στο View που εμφανίζονται όλα τα διαθέσιμα θέματα. Εάν ο χρήστης είναι καθηγητής, μπορεί να δει πόσες ομάδες επέλεξαν το συγκεκριμένο θέμα. Συγκεκριμένα εάν ο χρήστης ανήκει στον ρόλο ‘Professor’ τότε εμφάνισε το σύνολο των ομάδων που έχουν επιλέξει το συγκεκριμένο θέμα.

```
@if (User.IsInRole("Professor"))
{
    <small> @project.Teams.Count() ομάδες το επέλεξαν</small>
}
```

Τέλος στην σελίδα που βλέπουμε όλες τις αναθέσεις που έχουν πραγματοποιηθεί, υπάρχουν οι επιλογές για εκτύπωση ή αποθήκευση ως pdf αρχείο. Για την καλύτερη αποτύπωση των πληροφοριών στο χαρτί επιλέγουμε να μην εμφανιστεί η εικόνα που υπάρχει στον browser μας, αλλά θέλουμε μια πιο απλή εμφάνιση. Αφού ορίσουμε ενα id στον σχετικό πίνακα που περιέχει τα αποτελέσματα των αναθέσεων, θα προσθέσουμε στο CSS κάποια χαρακτηρηστικά. Συγκεκριμένα θέλουμε μόνο οταν πρόκειται για εκτύπωση, το χρώμα της γραμματοσειράς να είναι μαυρο, να μην υπάρχει κάποιο φόντο με χρώμα και ενα διακριτικό πλαίσιο, για κάθε κελί του πίνακα. Παρακάτω βλέπουμε τον σχετικό CSS κανόνα.

```
@media print {
    #printTable{
        border:solid 1px;
        color:black;
        background:none;
    }
}
```

3. Παρουσίαση λειτουργίας

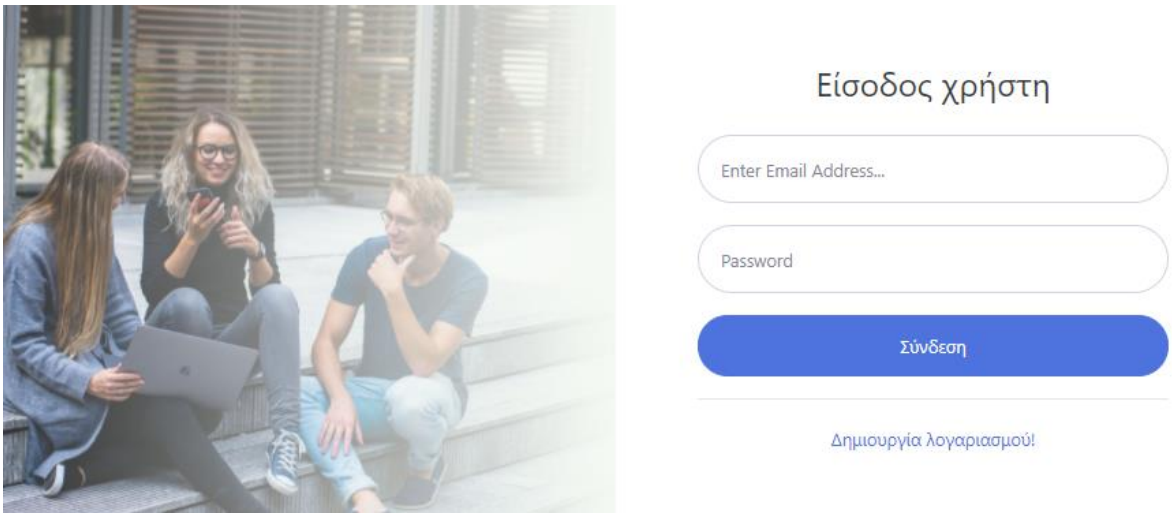
3.1 Εισαγωγή

Στην ενότητα αυτή θα δούμε την εφαρμογή σε λειτουργία, από την σκοπιά και των τριών, διαφορετικού ρόλου, χρηστών. Έχουμε καταχωρήσει μερικά δεδομένα στην βάση, για να δούμε όλες τις ενέργειες μέχρι να ολοκληρωθεί η ανάθεση των πτυχιακών εργασιών. Θα εξηγήσουμε λίγο καλύτερα την λογική που ακολουθεί η εφαρμογή και όταν κρίνεται απαραίτητο θα αναφέρουμε κάποιες ακόμη τεχνικές ή εργαλεία που χρησιμοποιούμε.

Σαν αρχική σελίδα της εφαρμογής μας είναι η σελίδα εισόδου. Ένας εγγεγραμμένος χρήστης πληκτρολογώντας το e-mail και το password, ανάλογα τον ρόλο θα δρομολογηθεί σε διαφορετική σελίδα. Ας ξεκινήσουμε κάνοντας εγγραφή για έναν σπουδαστή.

3.2 Περιήγησή ως Σπουδαστής

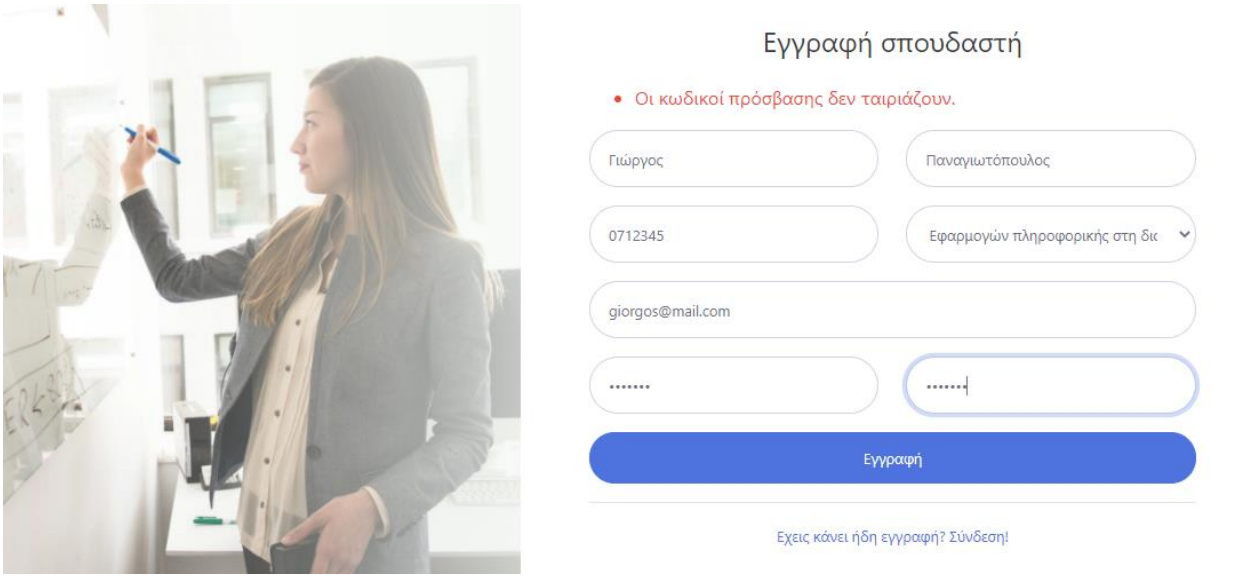
3.2.1 Εγγραφή & σύνδεση



Εικόνα 4 - Φόρμα σύνδεσης χρήστη στην εφαρμογή

Βρισκόμαστε στην αρχική σελίδα της εφαρμογής μας, η οποία είναι η σελίδα όπου ο χρήστης μπορεί να κάνει σύνδεση. Έστω οτι δεν έχουμε κάνει εγγραφή, οπότε επιλέγουμε τον υπέρ-σύνδεσμο 'Δημιουργία λογαριασμού' για να μεταφερθούμε στην σχετική σελίδα.

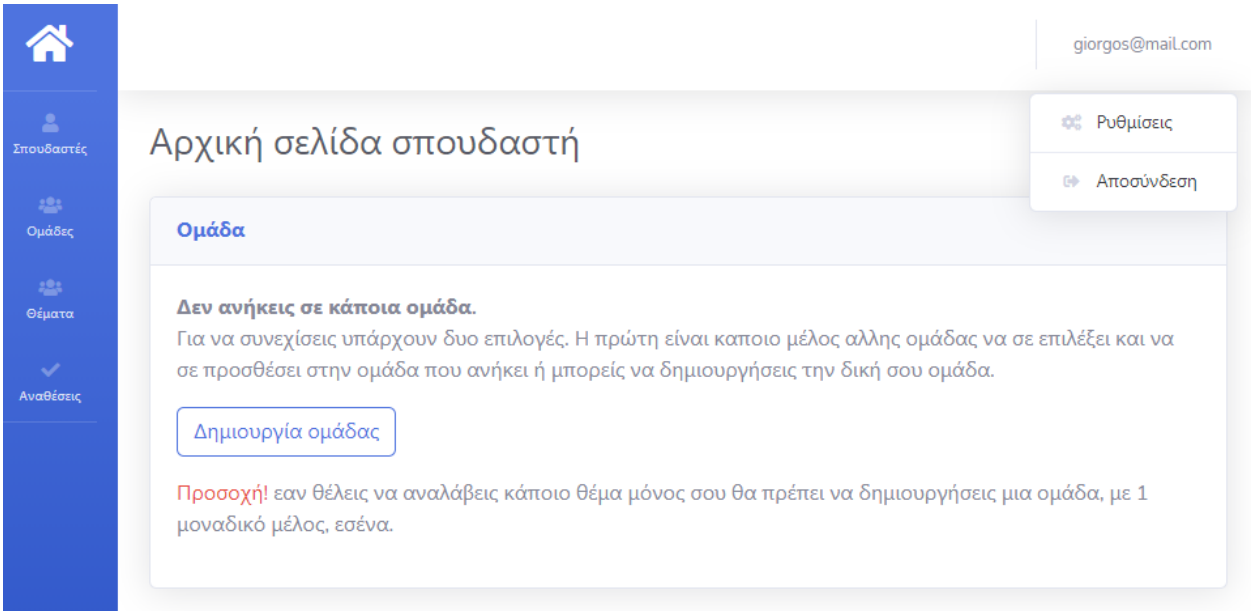
Στην σελίδα για την εγγραφή του σπουδαστή, όπως βλέπουμε υπάρχει μια φόρμα με τα σχετικά πεδία για τις πληροφορίες του χρήστη. Πληκτρολογήσαμε τα στοιχεία για τον σπουδαστή όπως όνομα επώνυμο, αριθμό μητρώου και το τμήμα, ενώ απαραίτητα είναι και το e-mail καθώς και κάποιος κωδικός πρόσβασης. Στα τελευταία δυο πεδία ορίζουμε τον κωδικό πρόσβασης που θα χρησιμοποιήσουμε για να συνδεθούμε. Σκόπιμα για χάρη του παραδείγματος πληκτρολογήσαμε δυο διαφορετικούς κωδικούς, η επαλήθευση δεν έγινε και το σχετικό μήνυμα σφάλματος εμφανίστηκε.



Εικόνα 5 - Φόρμα εγγραφής σπουδαστή

Προσπαθούμε ξανά, πληκτρολογώντας έγκυρα δεδομένα και ένας νέος χρήστης δημιουργήθηκε. Κατά την δημιουργία του χρήστη, πρακτικά δημιουργείτε μια εγγραφή στην βάση μας στον πίνακα Students, και μια ακόμη εγγραφή στον πίνακα Users, ενώ αυτές οι δύο εγγραφές συσχετίζονται βάσει ρόλου. Στη συνέχεια, πραγματοποιείται αυτόματα σύνδεση του χρήστη και βάσει του ρόλου που έχει ο χρήστης ανακατευθυνόμαστε στην αντίστοιχη αρχική σελίδα.

Στην αρχική σελίδα του σπουδαστή στο δεξί επάνω μέρος εμφανίζεται το όνομα χρήστη, ενώ επιλέγοντας το εμφανίζεται μια λίστα με επιλογές. Η μία μας οδηγεί μια σελίδα για αλλαγή κωδικού πρόσβασης ενώ η άλλη επιλογή για αποσύνδεση από την εφαρμογή.



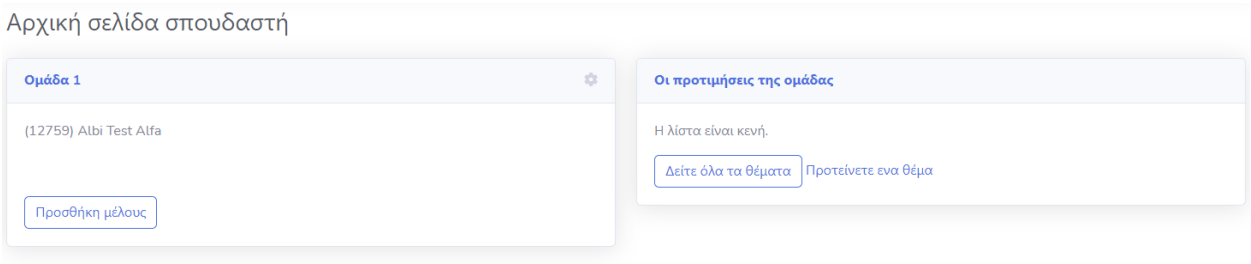
Εικόνα 6 - Αρχική σελίδα χρήστη με τον ρόλο του σπουδαστή

Στο αριστερό μέρος βλέπουμε το βασικό μενού με επιλογές για να δούμε τους σπουδαστές, τις ομάδες, τα διαθέσιμα θέματα καθώς και τις αναθέσεις των εργασιών που έχουν γίνει. Στο κύριο μέρος της σελίδας εμφανίζονται πληροφορίες για την συνολική εικόνα της ομάδας που ανήκει ο σπουδαστής. Στην περίπτωσή μας όμως ο σπουδαστής δεν ανήκει ακόμη σε κάποια ομάδα, αφού μόλις εγγράφηκε οπότε επιλέγουμε το κουμπί 'Δημιουργία ομάδας'.

3.2.2 Διαμόρφωση ομάδας

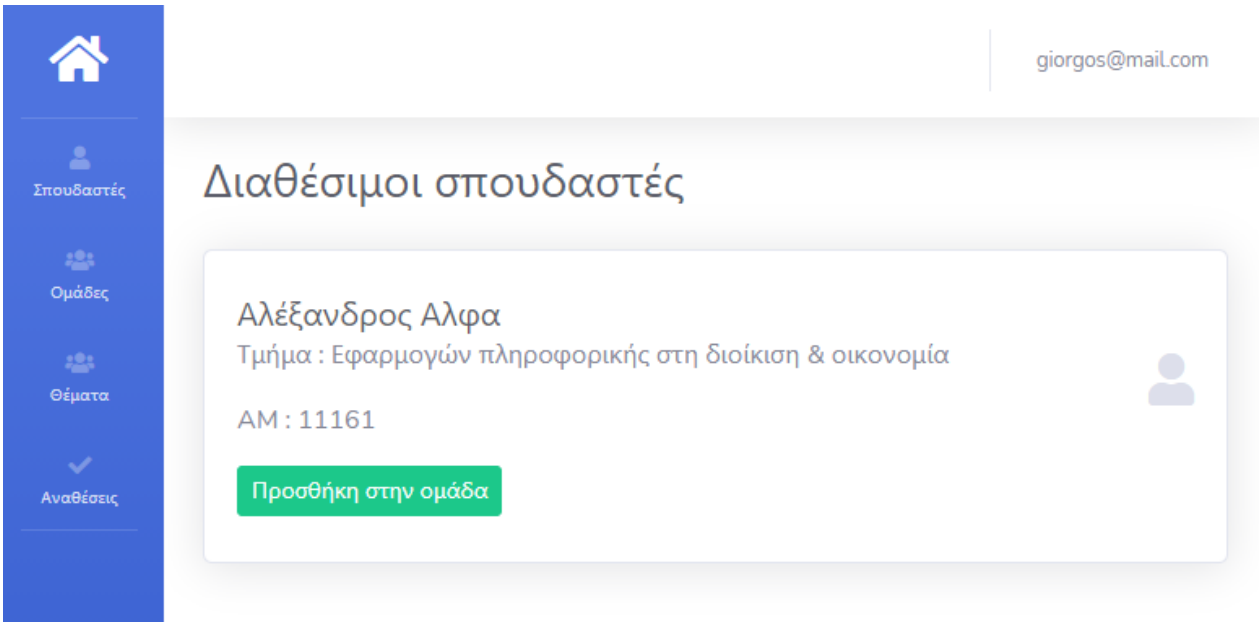
Αφού επιλέξουμε να κάνουμε δημιουργία ομάδας, θα ζητηθεί μια επιβεβαίωση και εφόσον επιλέξουμε να συνεχίσουμε, μια ομάδα θα δημιουργηθεί με μοναδικό μέλος τον σπουδαστή που την δημιούργησε. Ο λόγος που υπάρχει το βήμα της επιβεβαίωσης είναι διότι ενα μέλος που ανήκει σε κάποια ομάδα δεν μπορεί να προστεθεί ως μέλος σε κάποια άλλη.

Πλέον το περιεχόμενο της αρχικής σελίδας έχει αλλάξει, και στην οθόνη μας βλέπουμε δυο καρτέλες. Αριστερά βλέπουμε μια ενότητα με πληροφορίες για την ομάδα και την επιλογή να προσθέσουμε κάποιο μέλος, ενώ στο δεξί μερος της οθόνης έχει εμφανιστεί μια ενότητα που περιέχει μια λίστα με τις προτιμήσεις της ομάδας. Η λίστα με τις προτιμήσεις της ομάδας είναι κενή, μιας και η ομάδα μόλις δημιουργήθηκε.



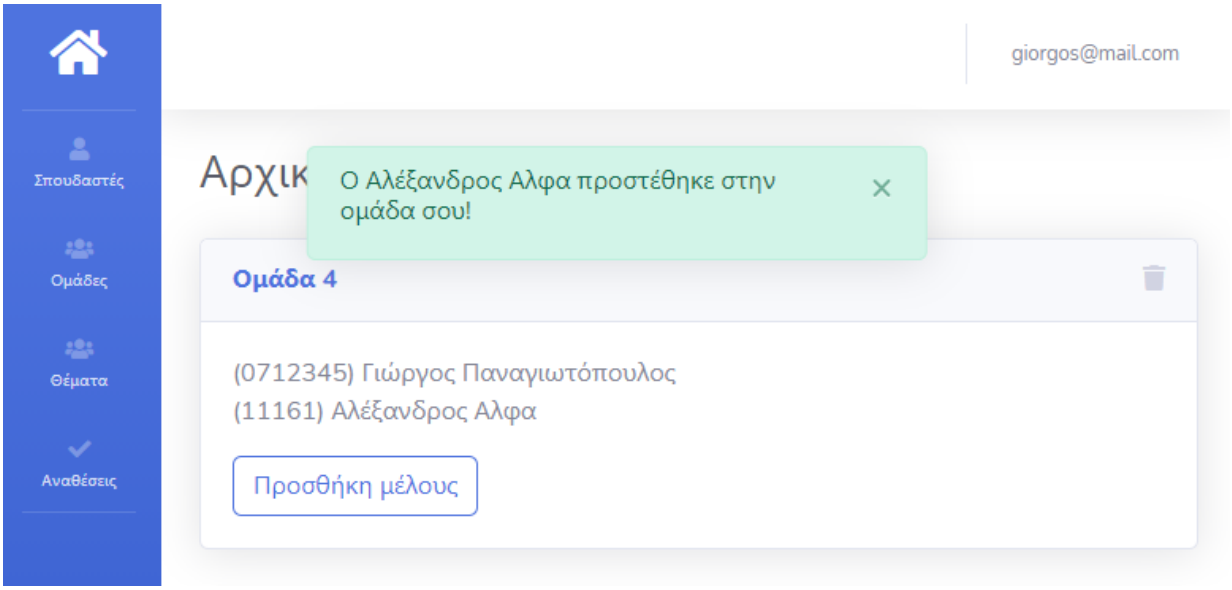
Εικόνα 7 – Αρχική σελίδα χρήστη που ανήκει σε κάποια ομάδα

Επιλέγουμε λοιπόν να προσθέσουμε κάποιο μέλος στην ομάδα. Κάνοντας κλικ στην επιλογή 'Προσθήκη μέλους', μεταφερόμαστε σε μια σελίδα που εμφανίζονται όλοι οι διαθέσιμοι σπουδαστές, εκείνοι δηλαδή που δεν ανήκουν σε άλλη ομάδα. Για κάθε σπουδαστή βλέπουμε το ονοματεπώνυμο του, το τμήμα και τον αριθμό μητρώου του. Επιλέγουμε λοιπόν το αντίστοιχο κουμπί για προσθήκη στην ομάδα.



Εικόνα 8 - Λίστα με τους διαθέσιμους προς προσθήκη στην ομάδα σπουδαστών

Αφού προσθέσουμε τον συγκεκριμένο σπουδαστή στην ομάδα μας, θα επιστρέψουμε αυτόματα στην αρχική σελίδα οπού θα δούμε το σχετικό μήνυμα επιβεβαίωσης ενώ το όνομα του συγκεκριμένου σπουδαστή να εμφανίζεται πλέον ως μέλος της ομάδας.

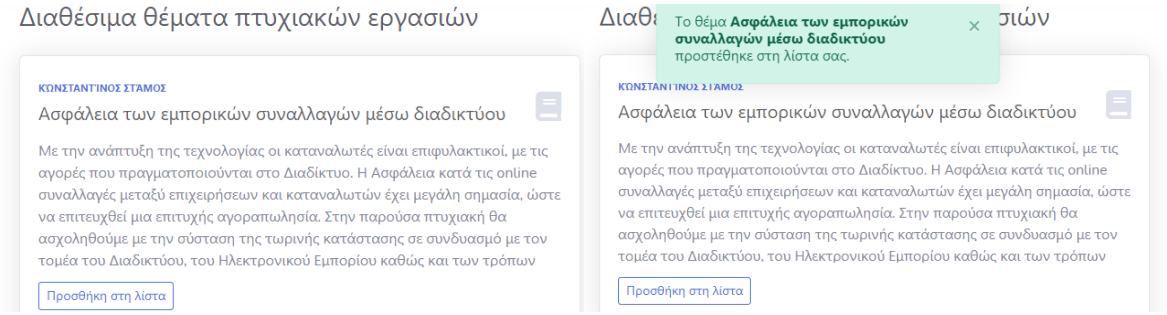


Εικόνα 9 - Μυνημα επιβεβαίωσης προσθήκης μέλους στην ομάδα

Σειρά έχει να επιλέξουμε από τα προτεινόμενα θέματα πτυχιακών εργασιών εκείνα που μας ενδιαφέρουν.

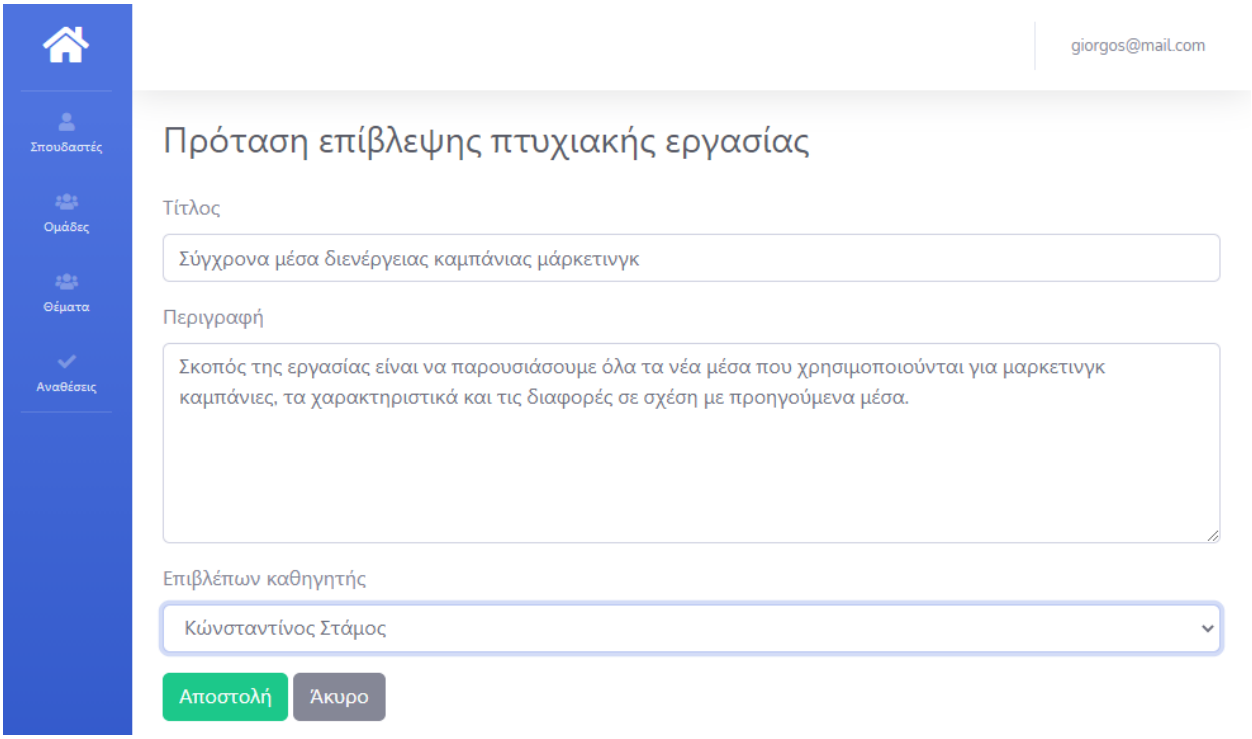
3.2.3 Επιλογή θεμάτων

Στην αρχική σελίδα, στην ενότητα ‘Οι προτιμήσεις της ομάδας’ υπάρχει η επιλογή ‘Δείτε όλα τα θέματα’, ένας υπερσύνδεσμος που μας οδηγεί σε μια σελίδα με όλα τα διαθέσιμα προς επιλογή προτεινόμενα θέματα πτυχιακών εργασιών. Για κάθε θέμα βλέπουμε ορισμένες πληροφορίες όπως το ονοματεπώνυμο του καθηγητή, την τίτλο της εργασίας καθώς και μια σύντομη περιγραφή, και φυσικά την επιλογή να προσθέσουμε το συγκεκριμένο θέμα στην λίστα με τις προτιμήσεις μας. Υπενθυμίζεται ότι μπορούμε να επιλέξουμε περισσότερα από ένα θέματα.



Εικόνα 10 – Προσθήκη πτυχιακής εργασίας στην λίστα προτιμήσεων

Η ομάδα έχει την δυνατότητα να προτείνει κάποιο θέμα που την ενδιαφέρει και να αποστείλει αίτημα επίβλεψης σε κάποιον καθηγητή. Στην αρχική σελίδα στην ενότητα που εμφανίζονται τα θέματα που έχει επιλέξει η ομάδα υπάρχει η επιλογή ‘Προτείνετε ένα θέμα’, όπου μας οδηγεί σε μία σελίδα για να υποβάλουμε το αίτημά μας. Όπως βλέπουμε και στην παρακάτω εικόνα, είναι απαραίτητος ο τίτλος του θέματος, και προαιρετικά μια σύντομη περιγραφή, καθώς και τον καθηγητή που θέλουμε να επιβλέψει την εργασία. Κατά την επιλογή του καθηγητή, εμφανίζεται μια λίστα με όλους τους εγγεγραμμένους στην εφαρμογή μας καθηγητές.



Εικόνα 11 - Φόρμα δημιουργίας αιτήματος για επίβλεψη πτυχιακής εργασίας

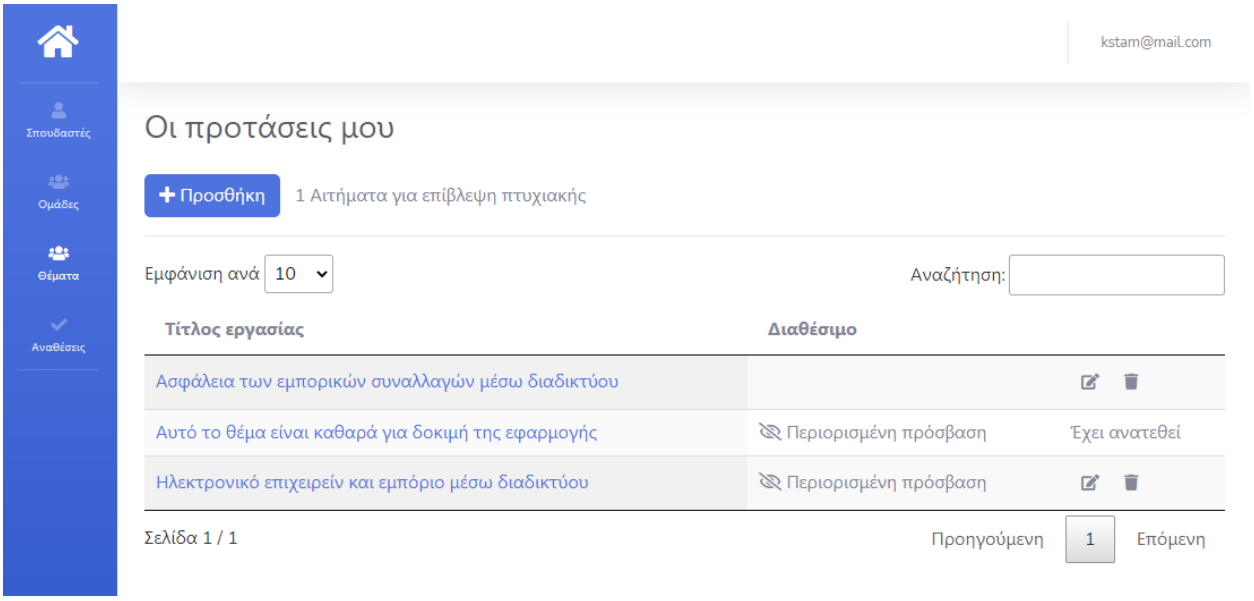
Αφού συμπληρώσουμε τα απαραίτητα στοιχεία, πατάμε το κουμπί 'Αποστολή' και στη συνέχεια μεταφερόμαστε στην αρχική σελίδα, οπού βλέπουμε το σχετικό μήνυμα επιβεβαίωσης. Την εξέλιξη του αιτήματος θα την δούμε στην παρακάτω ενότητα, από την σκοπιά του καθηγητή.

3.3 Περιήγηση ως Διδάσκων

Ας δούμε λοιπόν τώρα την εφαρμογή από την πλευρά ενός άλλου χρήστη, που έχει τον ρόλο του καθηγητή. Ένας καθηγητής θα λάβει μέσω email μια πρόσκληση για να χρησιμοποιήσει την εφαρμογή. Την διαδικασία αυτή θα την δούμε στην παρακάτω ενότητα, καθώς γίνεται από τον διαχειριστή της εφαρμογής.

3.3.1 Σύνδεση

Η σελίδα σύνδεσης της εφαρμογής όπως προαναφέραμε είναι κοινή, συνεπώς και ο καθηγητής θα χρησιμοποιήσει την ίδια φόρμα με τις ίδιες ακριβώς δυνατότητες, λειτουργίες και ελέγχους. Στη συνέχεια αφού συνδεθεί θα μεταφερθεί στην αρχική σελίδα όπου κάποια στοιχεία είναι κοινά με τον σπουδαστή όμως λόγω του ρόλου του βλέπει διαφορετικό περιεχόμενο. Βασικό στοιχείο στην συγκεκριμένη σελίδα, είναι ο πίνακας που περιέχει τα θέματα που έχει προτείνει ο ίδιος. Στην πρώτη στήλη εμφανίζεται ο τίτλος του θέματος, στη συνέχεια εάν είναι διαθέσιμο και στην τρίτη στήλη οι επιλογές για επεξεργασία ή διαγραφή της συγκεκριμένης εγγραφής.



Εικόνα 12 - Αρχική σελίδα χρήστη με τον ρόλο του καθηγητή

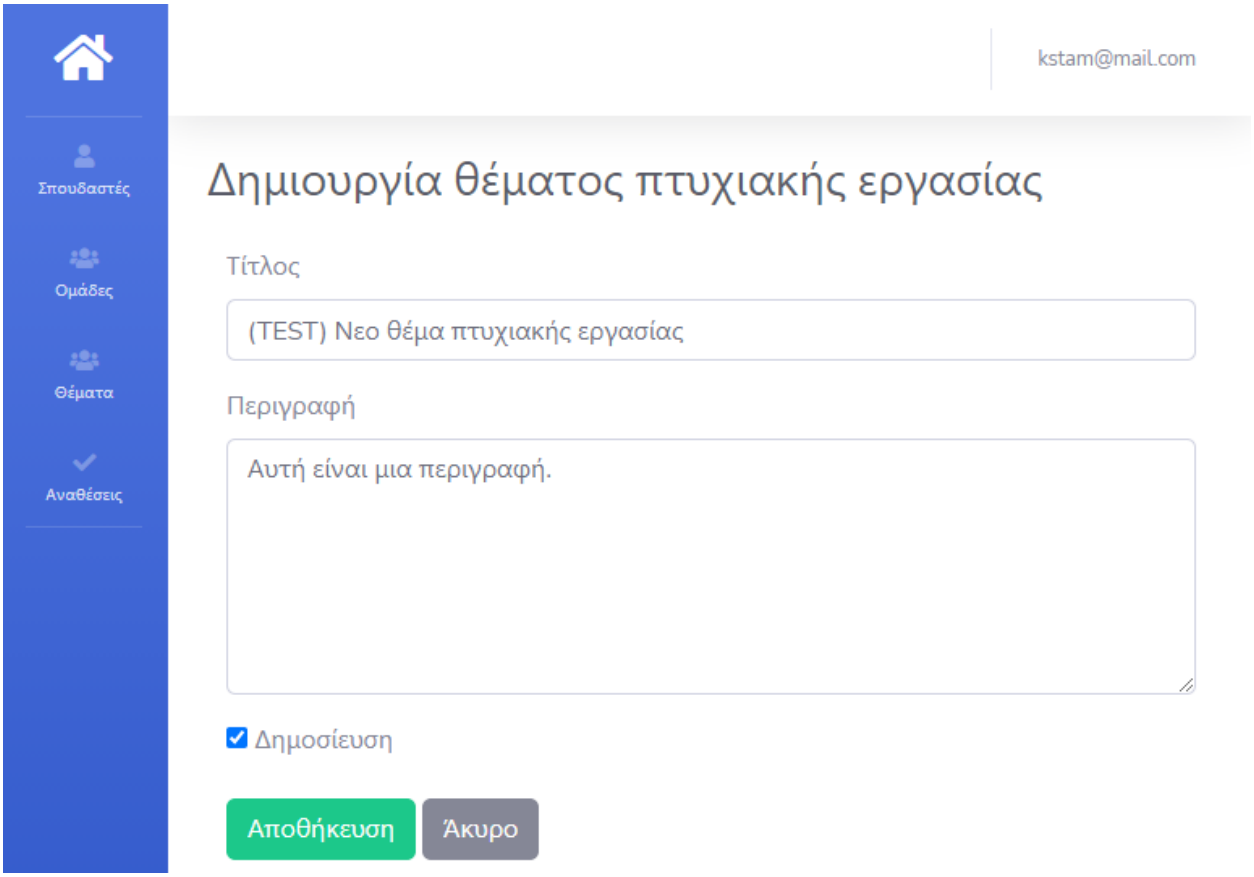
Όπως μπορείτε να παρατηρήσετε στην εικόνα, στην δεύτερη και τρίτη σειρά του πίνακα, στην δεύτερη στήλη βλέπουμε το μήνυμα 'Περιορισμένη πρόσβαση'. Το μήνυμα αυτό εμφανίζεται όταν το συγκεκριμένο θέμα έχει ήδη ανατεθεί, ή οταν δεν είναι διαθέσιμο προς τις ομάδες να το επιλέξουν καθώς ο χρήστης ακόμη δεν θέλει να εμφανιστεί.

Στην δεύτερη σειρά του πίνακα, όπως βλέπουμε δεν υπάρχουν τα σχετικά εικονίδια για επεξεργασία η διαγραφή, αντί αυτού εμφανίζεται το μήνυμα 'Έχει ανατεθεί. Ο λόγος είναι διότι δεν θέλουμε όταν κάποιο θέμα ανατεθεί να γίνει κάποια τροποποίηση.

Ας επιλέξουμε τώρα να προτείνουμε κάποιο ακόμη θέμα, κάνοντας κλικ στην επιλογή 'Προσθήκη'.

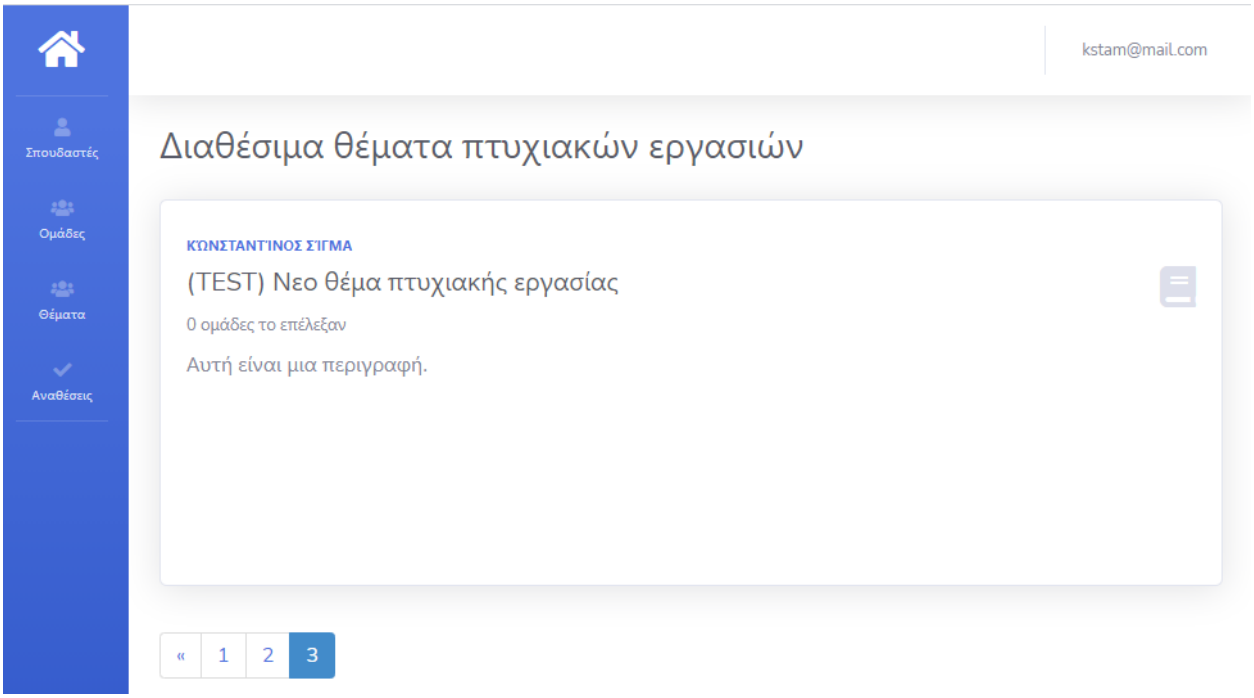
3.3.2 Πρόταση πτυχιακής εργασίας

Οι πληροφορίες που χρειάζονται είναι ο τίτλος του θέματος και προαιρετικά μια σύντομη περιγραφή, ενώ στο τέλος βλέπουμε την επιλογή 'Δημοσίευση'. Εάν δεν επιλέξουμε την συγκεκριμένη επιλογή τότε το θέμα που προτείναμε δεν θα είναι ορατό στις ομάδες και κατ' επέκταση δεν θα έχουν την δυνατότητα να το επιλέξουν. Επιλέγουμε λοιπόν 'Δημοσίευση' και αποθήκευση.



Εικόνα 13 - Φόρμα δημιουργίας θέματος πτυχιακής εργασίας

Αφού δημιουργήσουμε λοιπόν το θέμα, θα πάμε στην σελίδα που εμφανίζονται όλα τα διαθέσιμα θέματα για να επιβεβαιώσουμε πως έχει εμφανιστεί και εκεί. Μια λεπτομέρεια που παρατηρούμε και αναφέραμε σε προηγούμενο κεφάλαιο, είναι πως πλέον βλέπουμε και πόσες ομάδες έχουν επιλέξει το συγκεκριμένο θέμα. Αυτό οφείλεται στον ρόλο που έχει ο συγκεκριμένος χρήστης και είναι μια πληροφορία που οι σπουδαστές δεν μπορούν να έχουν. Ο λόγος που δίνουμε αυτή την πληροφορία είναι απλώς για ανατροφοδότηση του καθηγητή.

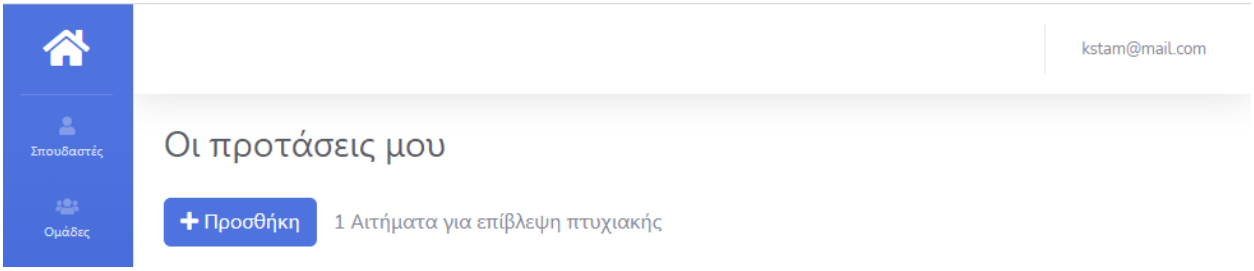


Εικόνα 14 - Λίστα διαθέσιμων θεμάτων πτυχιακής εργασίας

Ολοκληρώσαμε λοιπόν την διαδικασία δημιουργίας μιας πρότασης πτυχιακής εργασία και θα πάμε να διαχειριστούμε, εάν υπάρχει, κάποιο αίτημα για επίβλεψη εργασίας από τις ομάδες.

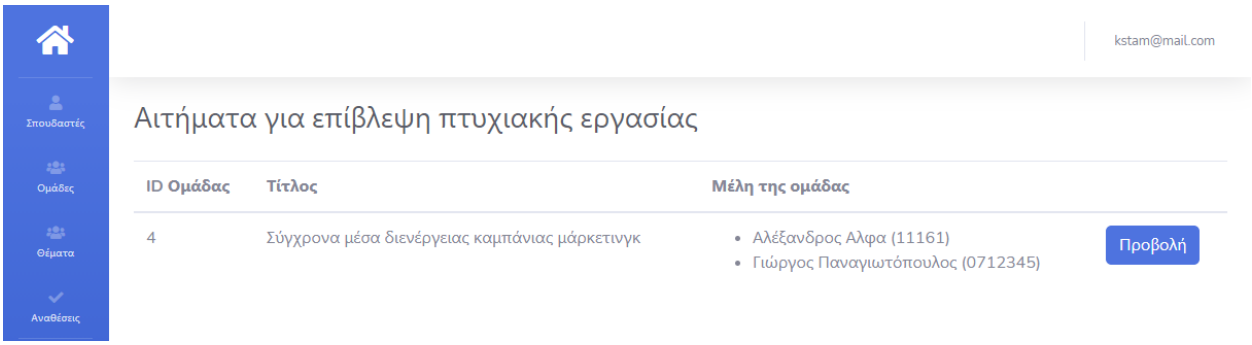
3.3.3 Διαχείριση αιτήματος επίβλεψης πτυχιακής εργασίας

Στην αρχική σελίδα βλέπουμε πως υπάρχει ένα αίτημα για επίβλεψη πτυχιακής εργασίας. Πρόκειται για την πρόταση που κάναμε στην προηγούμενη ενότητα, το επιλέγουμε και πάμε να διαχειριστούμε το αίτημα.



Εικόνα 15 – Ειδοποίηση για αίτημα επίβλεψης πτυχιακής εργασίας

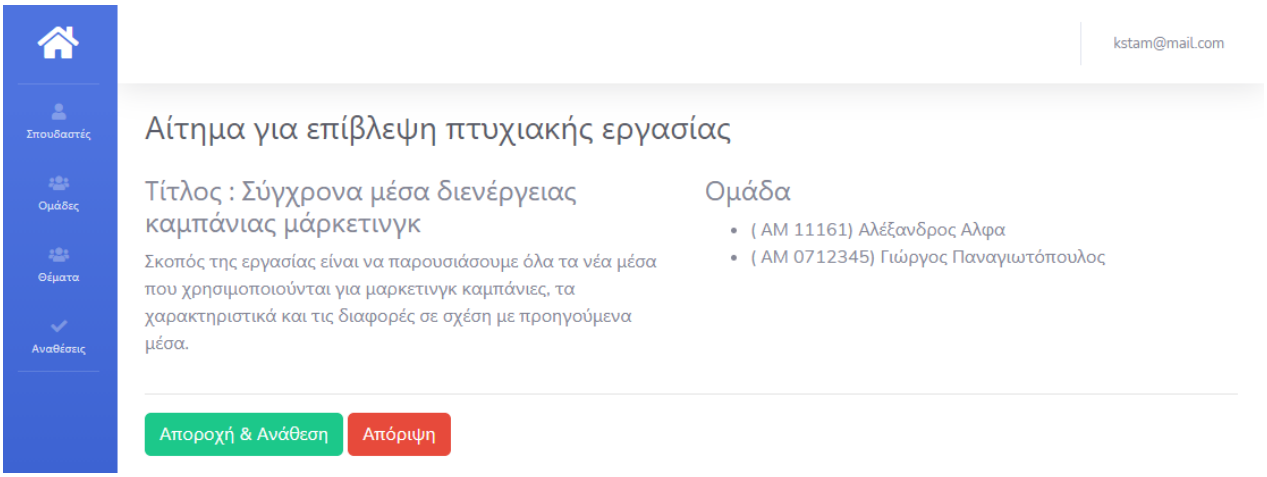
Στην συγκεκριμένη σελίδα βλέπουμε τα αιτήματα που έχουν υποβάλει οι ομάδες, για επίβλεψη πτυχιακής εργασίας και αφορούν τον συγκεκριμένο χρήστη.



Εικόνα 16 - Λίστα με αιτήματα επίβλεψης πτυχιακής εργασίας

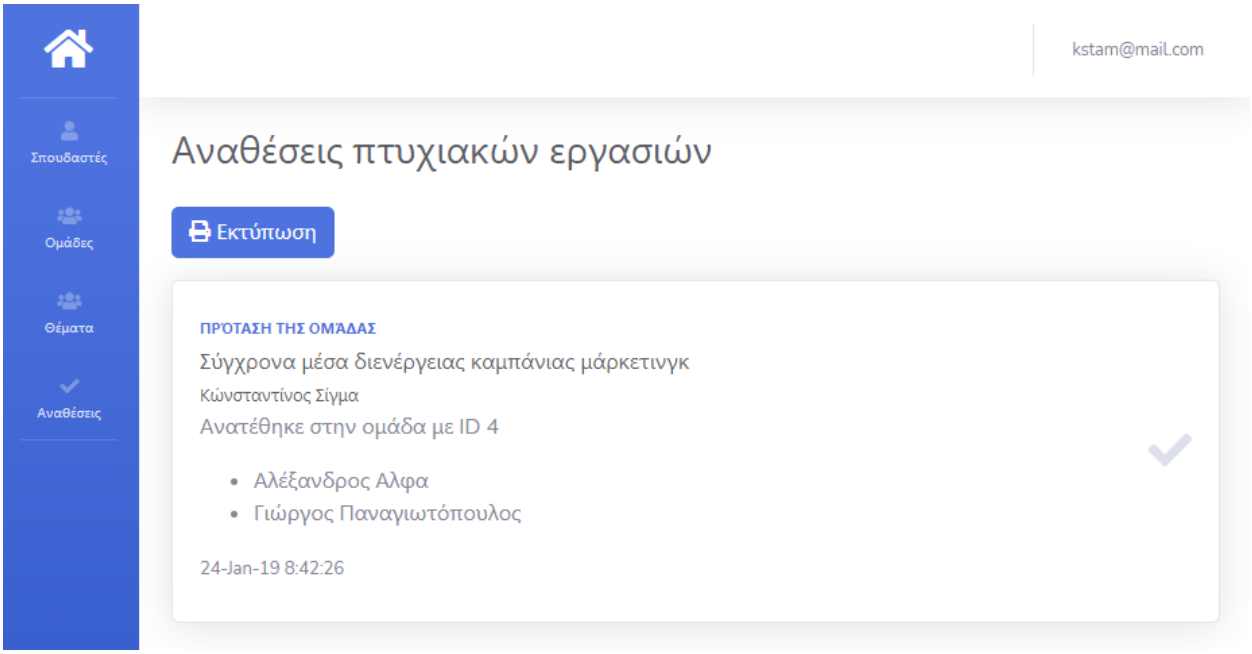
Στον πίνακα ο καθηγητής βλέπει το ID της ομάδας, τον τίτλο του θέματος καθώς και τα μέλη της ομάδας. Επιλέγουμε προβολή, για να δούμε περισσότερες λεπτομέρειες.

Στο αριστερό μέρος βλέπουμε τον τίτλο του θέματος και μια περιγραφή, που βοηθάει τον καθηγητή να αξιολογήσει εάν πρέπει ή όχι να αποδεχτεί το αίτημα, ενώ στα αριστερά βλέπουμε ποια ομάδα το πρότεινε.



Εικόνα 17 - Πληροφορίες αιτήματος επίβλεψης πτυχιακής εργασίας

Στο κάτω μέρος υπάρχουν οι επιλογές ‘Αποδοχή & Ανάθεση’ ή ‘Απόρριψη’, όπου το θέμα σε αυτή την περίπτωση θα διαγραφεί. Επιλέγουμε λοιπόν ‘Αποδοχή & Ανάθεση’, και όπως βλέπουμε στην εικόνα, στην σελίδα που εμφανίζονται όλες οι αναθέσεις προστέθηκε μια ακόμη ανάθεση. Παρατηρούμε ότι πέρα από τον τίτλο, τα στοιχεία της ομάδας και το ονοματεπώνυμο του επιβλέπων καθηγητή, βλέπουμε την διαδικασία που έγινε η ανάθεση καθώς και την ημερομηνία.



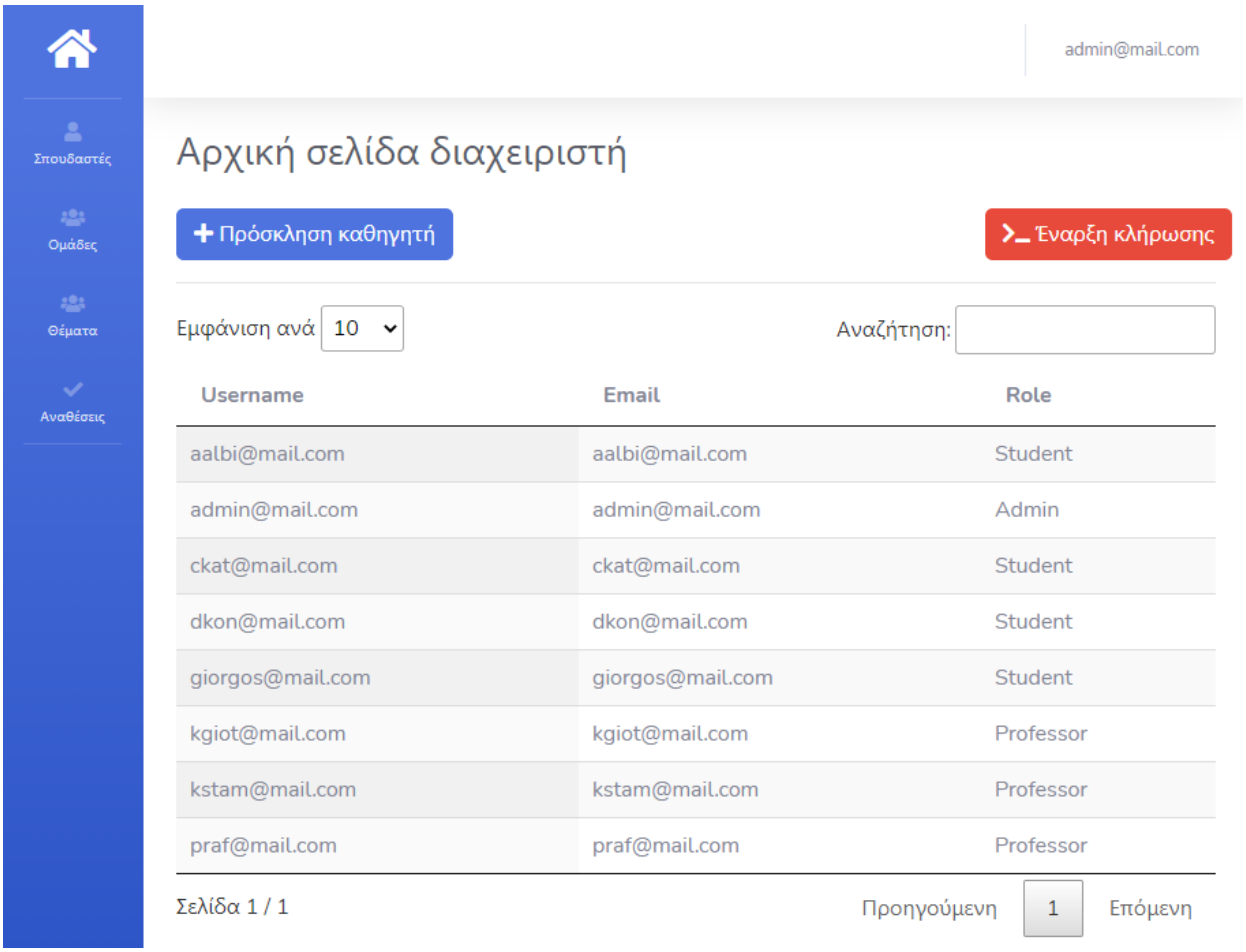
Εικόνα 18 - Ολοκληρωμένες αναθέσεις πτυχιακών εργασιών

3.4 Περιήγηση ως Διαχειριστής

3.4.1 Σύνδεση και επισκόπηση

Υστερα από επιτυχή σύνδεση στην εφαρμογή ο χρήστης μεταφέρεται σε μία σελίδα που έχει τον ρόλο ενός πίνακα ελέγχου. Για τον διαχειριστή επιλέξαμε στην αρχική σελίδα να υπάρχουν οι επιλογές 'Προσθήκης ενός καθηγητή', καθώς και ενας πίνακας με όλους τους εγγεγραμένους χρήστες και τον ρόλο που έχουν στην εφαρμογή.

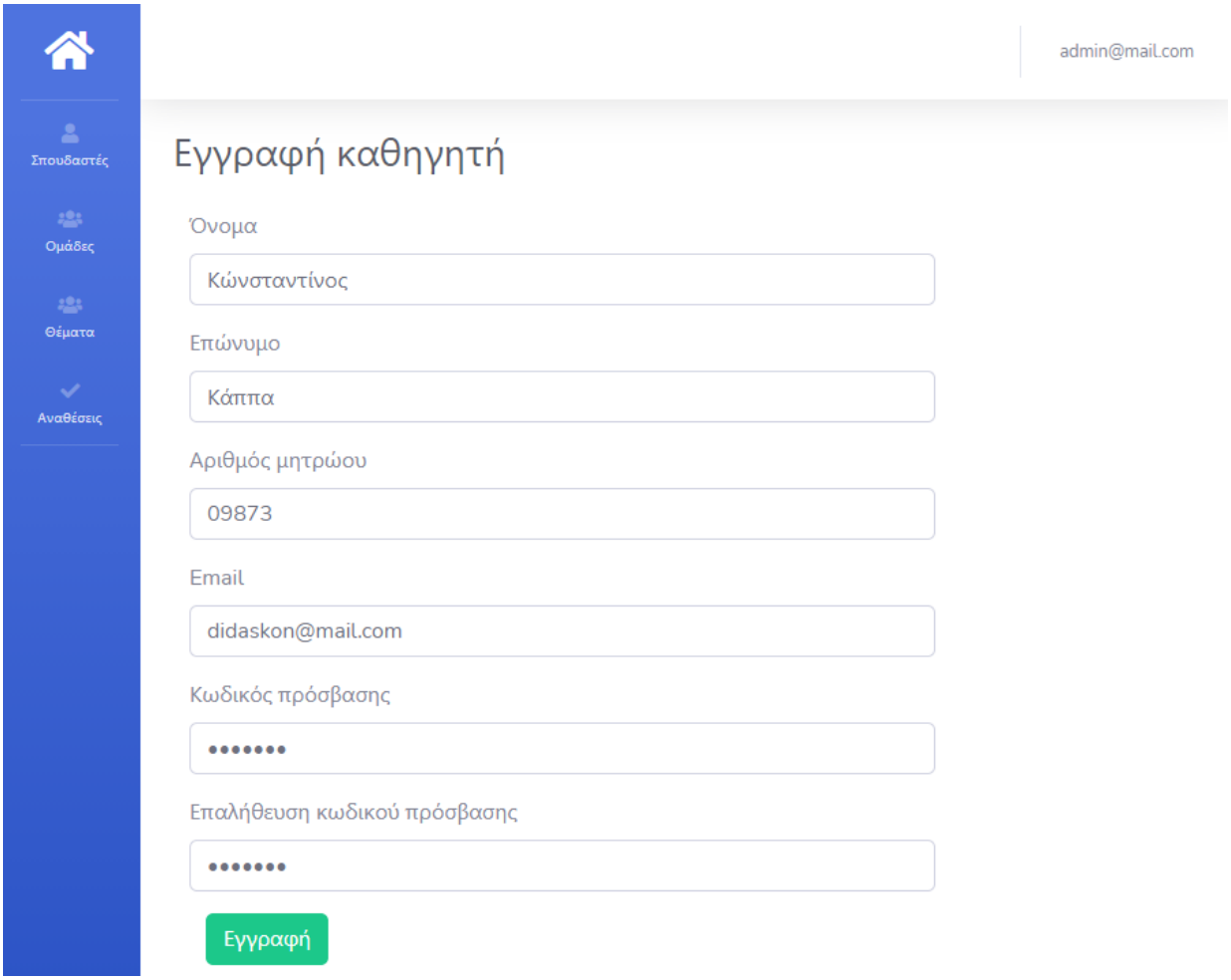
Όπως προαναφέραμε ο καθηγητής της εφαρμογής δεν θα μπορεί να κάνει εγγραφεί, αλλά η εγγραφή του γίνεται εκ μέρους του διαχειριστή. Στην επόμενη ενότητα θα δούμε τα βήματα που είναι απαραίτητα για να εγγράψουμε, ως διαχειριστής πλέον, εναν καθηγητή στην εφαρμογή μας.



Εικόνα 19 - Αρχική σελίδα χρήστη με τον ρόλο του διαχειριστή

3.4.2 Πρόσκληση καθηγητή

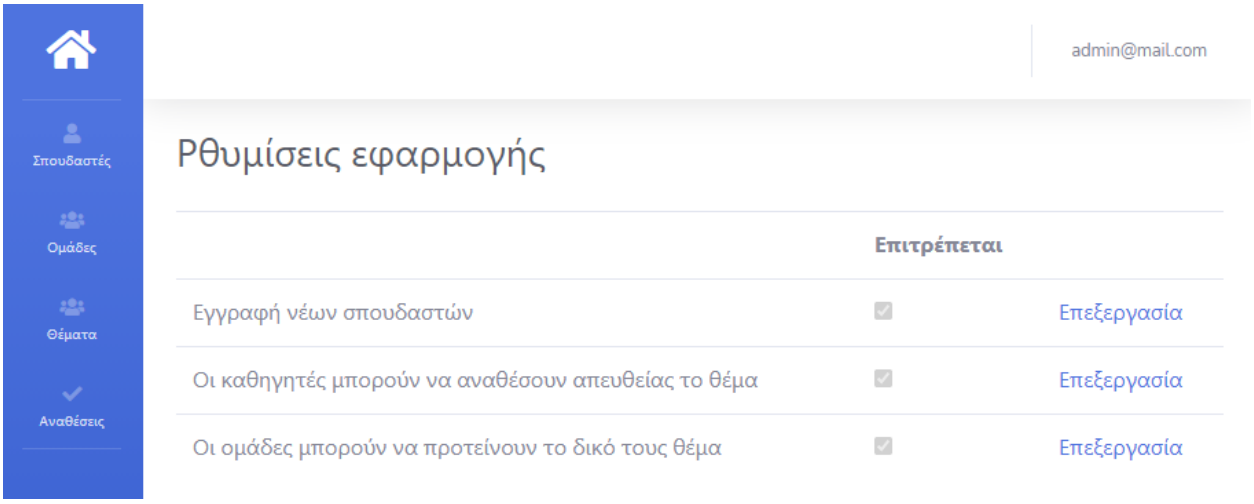
Συμπληρώνουμε τα πεδία που απαιτούνται όπως το όνομα το επώνυμο και τον αριθμό μητρώου του καθηγητή, την διεύθυνση ηλεκτρονικού ταχυδρομείου καθώς και έναν κωδικό πρόσβασης. Επιλέγοντας εγγραφή, εάν δεν υπάρχει ήδη χρήστης με τον ρόλο του καθηγητή και τα συγκεκριμένα στοιχεία, η εγγραφή θα πραγματοποιηθεί και θα σταλεί ένα μύνημα μέσω ηλεκτρονικού ταχυδρομείου στον συγκεκριμένο καθηγητή, όπου θα περιλαμβάνει τα στοιχεία εισόδου. Στη συνέχεια ο καθηγητής καλείται να αλλάξει τον κωδικό πρόσβασης του, για λόγους ασφαλείας.



Εικόνα 20 - Φόρμα εγγραφής και πρόσκλησης ενος καθηγητή στην εφαρμογή

3.4.3 Παραμετροποίηση εφαρμογής

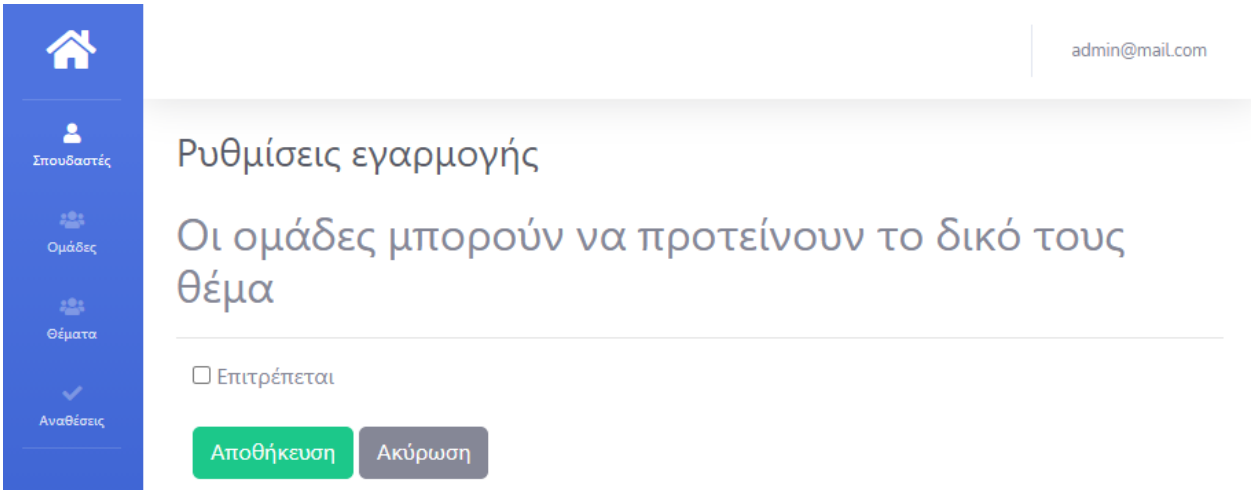
Ο διαχειριστής της εφαρμογής έχει την δυνατότητα να αλλάξει την συμπεριφορά της εφαρμογής, μέσω κάποιων ρυθμίσεων. Στην παρακάτω εικόνα βλέπουμε πως ο διαχειριστής μπορεί να διακόψει την δυνατότητα εγγραφής νέων σπουδαστών ή για παράδειγμα να μην επιτρέπει στις ομάδες να προτείνουν το δικό τους θέμα.



Εικόνα 21 - Επιλογές ρυθμίσεων εφαρμογής

Κάθε τέτοια ρύθμιση μπορεί να αλλάξει, κάνοντας κλικ στην επιλογή 'Επεξεργασία' στην αντίστοιχη σειρά.

Τροποποιούμε την τιμή της επιλογής 'Επιτρέπεται' ωστε να είναι κενή, συνεπώς κατά την αποθήκευση οι ομάδες δεν θα μπορούν πλέον να προτείνουν κάποιο θέμα. Διευκρινίζεται πως η ρύθμιση αυτή ισχύει από την στιγμή της αποθήκευσης και έπειτα, ενώ ομάδες που έχουν ήδη κάνει κάποια πρόταση δεν επηρεάζονται.



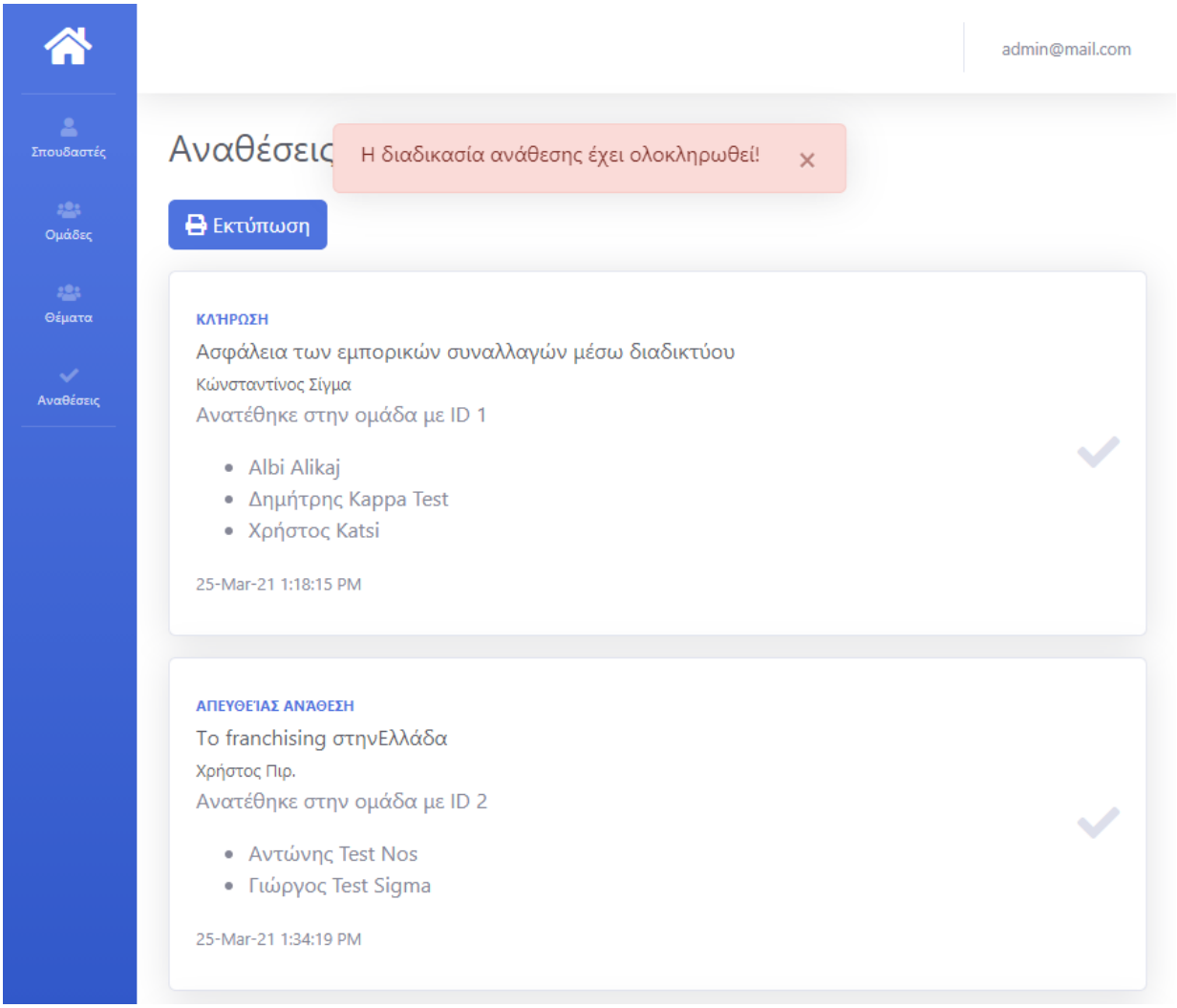
Εικόνα 22 - Αλλαγή ρύθμισης απο τον διαχειριστή

3.4.4 Έναρξη διαδικασίας ανάθεσης

Τέλος το σημαντικότερο μέρος της εργασίας, είναι η δυνατότητα αυτόματης ανάθεσης των πτυχιακών εργασιών. Η δυνατότητα αυτή παρέχεται μόνο στον διαχειριστή της εφαρμογής, μέσω της σχετικής επιλογής στην αρχική σελίδα.

Η λογική που κρύβεται πίσω απο την αυτόματη ανάθεση των πτυχιακών εργασιών, είναι πως για κάθε θέμα ελέγχουμε τις ομάδες που έχουν επιλέξει, και αξιολογώντας κάποια κριτήρια όπως το

πλήθος των μελών που αποτελείται μια ομάδα ή τον αριθμό των θεμάτων που έχει επιλέξει, επιλέγεται η ομάδα που θα το αναλάβει. Όταν η διαδικασία ολοκληρωθεί, ο χρήστης της εφαρμογής ανακατευθύνεται στην αρχική σελίδα όπου θα εμφανιστεί το σχετικό μήνυμα για την εξέλιξη της διαδικασίας.



Εικόνα 23 - Αποτελέσματα διαδικασίας αυτόματης ανάθεσης πτυχιακών εργασιών

4. Συμπεράσματα – Μελλοντικές εργασίες

Κάνοντας μια μικρή επιγραμματική ανασκόπηση στα κεφάλαια ένα, δύο και τρία κάποιος πιθανό να συμπεράνει πως η εργασία ηταν αρκετά απλή, καθώς τα βήματα είναι ξεκάθαρα και συγκεκριμένα. Ωστόσο η πραγματικότητα είναι αρκετά διαφορετική, καθώς δεν υπήρξε καμία αναφορά σε ότι δεν δύλεψε ή σε διάφορα σφάλματα.
Αφιερώθηκαν αμέτρητες ώρες αναζήτησης, μελέτη και προετοιμασίας, πρίν ακόμη ξεκινήσω να γράψω την πρώτη σειρά κώδικα. Το αποτέλεσμα που σας παρουσίασα είναι ενα πολύ μικρό ποσοστό σε σχέση με τον όγκο της εργασίας, αφού αρκετές φορές χρειάστηκε να διαγράψω πράγματα, ακόμη και να ξεκινήσω απο την αρχή. Αμέτρητες ώρες προσπαθώντας να καταλάβω κάποιο σφάλμα ή για ποιο λόγο κάτι δεν δούλεψε, πολλές φορές κολλημένος σε προβλήματα που εκείνη την στιγμή φάνταζαν απροσπέραστα.

Αναμφισβήτητα η εργασία μπορεί να βελτιωθεί, οργανώνοτας και ξαναγράφωντας ενδεχομένως λίγο καλύτερα τον κώδικα (refactoring) ακολουθόντας πιο αυστηρά κάποια αρχιτεκτονική, και να αναπτυχθεί ακόμη περισσότερο.

Για παράδειγμα αρχικά θα μπορούσε να δημιουργηθούν διάφορες εργασίες που θα τρέχουν αυτόματα (jobs) για συγκεκριμένες διαδικασίες. Οι δυνατότητες και οι λειτουργικότητες (features) που παρέχει η εφαρμογή θα μπορύσαν να αξιολογηθούν, να τροποποιηθούν και φυσικά να προστεθούν και νέες. Για παράδειγμα να πραγματοποιηθεί επικοινωνία με άλλα πληροφοριακά συστήματα του ΤΕΙ Δυτικής Ελλάδας, ωστε να γίνεται επαλήθευση των στοιχείων του σπουδαστή πρίν την εγγραφή. Ενα ακόμη χρήσιμο feature που μπορεί να υλοποιηθεί με την βοήθεια της τεχνητής νοημοσύνης είναι μια διαδικασία επιβεβαίωσης του χρήστη. Αυτό μπορεί να επιτευχθεί για παράδειγμα ανοίγοντας την κάμερα του υπολογιστή και η εικόνα να συγκριθεί με την φωτογραφία φωτογραφία του σπουδαστή που κατέχει η Γραμματεία της σχολής.

Όπως επιτάσσει η εποχή θα μπορούσε η εφαρμογή να αναπτυχθεί και σε άλλες συσκευές. Μάλιστα, όπως αναφέραμαι και στο πρώτο κιολας κεφάλαιο, με την χρήση της C# μπορούμε να δημιουργούμε εφαρμογές για διάφορες συσκευές. Εκμεταλευόμενοι λοιπόν αυτή την δυνατότητα μπορούμε να επαναχρησιμοποιήσουμε κώδικα που έχουμε ήδη, όπως παράδειγμα το project Thesis.Services και με Xamarin Forms να δημιουργήσουμε μια εφαρμογή για συσκευές κινητής τηλεφωνίας.

5. Παράρτημα

5.1 Πίνακας εικόνων

Εικόνα 1 - .NET Framework 14
Εικόνα 2 – Διάγραμμα συσχετίσεων πινάκων της βάσης δεδομένων 29
Εικόνα 3 – Δομικά στοιχεία μιας σελίδας με την χρήση layout. 41
Εικόνα 4 - Φόρμα σύνδεσης χρήστη στην εφαρμογή 46
Εικόνα 5 - Φόρμα εγγραφής σπουδαστή 47
Εικόνα 6 - Αρχική σελίδα χρήστη με τον ρόλο του σπουδαστή 47
Εικόνα 7 – Αρχική σελίδα χρήστη που ανήκει σε κάποια ομάδα 48
Εικόνα 8 - Λίστα με τους διαθέσιμους προς προσθήκη στην ομάδα σπουδαστών 49
Εικόνα 9 - Μυνημα επιβεβαίωσης προσθήκης μέλους στην ομάδα 49
Εικόνα 10 – Προσθήκη πτυχιακής εργασίας στην λίστα προτιμήσεων 50
Εικόνα 11 - Φόρμα δημιουργίας αιτήματος για επίβλεψη πτυχιακής εργασίας 51
Εικόνα 12 - Αρχική σελίδα χρήστη με τον ρόλο του καθηγητή 52
Εικόνα 13 - Φόρμα δημιουργίας θέματος πτυχιακής εργασίας 53
Εικόνα 14 - Λίστα διαθέσιμων θεμάτων πτυχιακής εργασίας 54
Εικόνα 15 – Ειδοποίηση για αίτημα επίβλεψης πτυχιακής εργασίας 54
Εικόνα 16 - Λίστα με αιτήματα επίβλεψης πτυχιακής εργασίας 55
Εικόνα 17 - Πληροφορίες αιτήματος επίβλεψης πτυχιακής εργασίας 55
Εικόνα 18 - Ολοκληρωμένες αναθέσεις πτυχιακών εργασιών 56
Εικόνα 19 - Αρχική σελίδα χρήστη με τον ρόλο του διαχειριστή 57
Εικόνα 20 - Φόρμα εγγραφής και πρόσκλησης ενος καθηγητή στην εφαρμογή 58
Εικόνα 21 - Επιλογές ρυθμίσεων εφαρμογής 59
Εικόνα 22 - Αλλαγή ρύθμισης απο τον διαχειριστή 59
Εικόνα 23 - Αποτελέσματα διαδικασίας αυτόματης ανάθεσης πτυχιακών εργασιών 60

5.2 Βιβλιογραφία

Git

- https://git-scm.com/
- https://el.wikipedia.org/wiki/Git_(%CE%BB%CE%BF%CE%B3%CE%B9%CF%83%CE%BC%CE%B9%CE%BA%CF%8C)

HTML

- https://en.wikipedia.org/wiki/HTML
- https://www.w3schools.com/

CSS

- https://en.wikipedia.org/wiki/CSS
- https://www.w3schools.com/

JavaScript,

- https://www.w3schools.com/

Bootstrap

- https://getbootstrap.com/

jQuery

- https://api.jquery.com/

C#, .Net Framework, ASP.Net, Entity Framework

- https://docs.microsoft.com/en-us/documentation/

5.3 Πηγές εικόνων

Εικόνα 1, https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-Framework

Εικόνα 3, https://www.tutorialsteacher.com/

Εικόνα 2, 4 έως 23, Στιγμιότυπο της εφαρμογής

5.4 Download links

Visual Studio Community,

- https://visualstudio.microsoft.com/vs/community/

SQL Express,

- https://www.microsoft.com/en-us/Download/details.aspx?id=101064

SQL Server Management Studio (SSMS),

- https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15