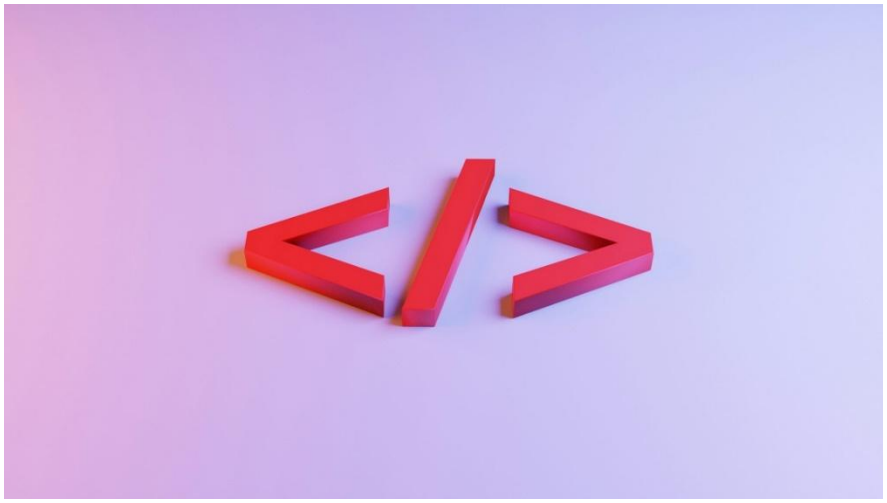
ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Ανάπτυξη διαδικτυακής εφαρμογής διασύνδεσης φοιτητών για φροντιστηριακά μαθήματα»



Καριώτης Πολυχρόνης Αλέξανδρος
175102

Επιβλέπων
Δρ. Κυριάκος Τσιακμάκης

Ιούνιος 2025

Ανάπτυξη διαδικτυακής εφαρμογής διασύνδεσης φοιτητών για φροντιστηριακά μαθήματα
Κωδικός: 24173
Φοιτητής: Καριώτης Πολυχρόνης Αλέξανδρος
Εισηγητής: Δρ. Κυριάκος Τσιακμάκης
Ημερομηνία ανάληψης Π.Ε. 27-03-2024
Ημερομηνία περάτωσης Π.Ε. 30-Μαΐου-2025

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

Περίληψη

Η εργασία αυτή αφορά την έρευνα, τον σχεδιασμό και την ανάπτυξη ενός διαδικτυακού ιστότοπου, όπου οι φοιτητές θα μπορούν να δημιουργούν και να συμμετέχουν σε πανεπιστημιακά φροντιστηριακά μαθήματα. Σκοπός των μαθημάτων αυτών είναι να βοηθήσουν τους φοιτητές στην επιτυχή ολοκλήρωση μαθημάτων που τους δυσκολεύουν, παρέχοντας τη δυνατότητα συγκέντρωσης όλων των φροντιστηρίων σε έναν ενιαίο ιστότοπο. Ο φοιτητής μπορεί οποιαδήποτε στιγμή να βλέπει τα διαθέσιμα μαθήματα και να συμμετέχει σε αυτά. Η εφαρμογή υλοποιήθηκε με Java και JavaScript/TypeScript, χρησιμοποιώντας το Spring Framework και το Angular, ενώ ως βάση δεδομένων χρησιμοποιήθηκε το PostgreSQL.

Developing a web application to provide private/group courses among university students

Kariotis Polychronis Alexandros

Abstract

This project involves the research, design, and development of a web-based platform where university students can create and participate in academic tutorial sessions. These sessions aim to support students in successfully completing courses they find challenging, by centralizing all tutorials in a single website. At any time, students can browse the available sessions and enroll in them. The application was developed using Java and JavaScript/TypeScript, with the Spring Framework and Angular, and PostgreSQL was used as the database.

Ευχαριστίες

Θέλω να ευχαριστήσω τους γονείς μου και τον επιβλέπων κ Τσιακμάκη Κυριάκο για την αμέριστη βοήθειά του στην εκπόνηση αυτής της εργασίας.

Περιεχόμενα

Κατάλογος Πινάκων

Κεφάλαιο 1ο: Εισαγωγή

1.1 Εισαγωγή

Η τριτοβάθμια εκπαίδευση αποτελεί ένα καθοριστικό στάδιο στην ακαδημαϊκή πορεία ενός ανθρώπου, διαφορετικό από αυτό που έχει συνηθίσει μέχρι το τέλος της δευτεροβάθμιας εκπαίδευσης. Προερχόμενος από ένα εκπαιδευτικό περιβάλλον όπου κυριαρχεί η στείρα γνώση και η αποστήθιση, ο φοιτητής καλείται πλέον να αναπτύξει την κριτική του σκέψη, να συνεργαστεί σε ομάδες και να συνδιαμορφώσει ιδέες με άλλους που μοιράζονται παρόμοια ενδιαφέροντα και στόχους. Στο πανεπιστήμιο, η μάθηση αποκτά πιο ουσιαστικό και βιωματικό χαρακτήρα, καθώς ο φοιτητής δεν περιορίζεται απλώς στη λήψη γνώσης, αλλά καλείται να τη μεταδώσει, να εμβαθύνει σε αντικείμενα που τον ενδιαφέρουν και να συνεισφέρει ενεργά στην ακαδημαϊκή κοινότητα.

Μέσα σε αυτό το πλαίσιο, δεν είναι σπάνιο ένας φοιτητής να επιθυμεί επιπλέον υποστήριξη σε ένα μάθημα ή τομέα που δυσκολεύεται. Ωστόσο, συχνά προκύπτει η εξής πρόκληση: δεν γνωρίζει κάποιον στον άμεσο φιλικό ή ακαδημαϊκό του κύκλο που να μπορεί να τον βοηθήσει, είτε επειδή δεν υπάρχουν κοινές επαφές, είτε επειδή οι φοιτητές που έχουν τη γνώση δεν έχουν προβληθεί ή προτείνει τις υπηρεσίες τους. Από την άλλη πλευρά, υπάρχουν φοιτητές που διαθέτουν τη γνώση και τη διάθεση να βοηθήσουν, αλλά δεν έχουν κάποιο μέσο να επικοινωνήσουν αυτή την πρόθεση ή να προσεγγίσουν όσους χρειάζονται βοήθεια.

Η διαδικτυακή εφαρμογή που αναπτύχθηκε στα πλαίσια της παρούσας πτυχιακής εργασίας, φιλοδοξεί να καλύψει ακριβώς αυτό το κενό. Μέσω του συστήματος διασύνδεσης, δίνεται η δυνατότητα στους φοιτητές να έρχονται σε επαφή με συναδέλφους που προσφέρουν υποστήριξη σε συγκεκριμένα μαθήματα, αξιοποιώντας ένα οργανωμένο και εύχρηστο περιβάλλον. Οι χρήστες μπορούν να επιλέξουν από μια λίστα διαθέσιμων φοιτητών, να διαβάσουν τις περιγραφές και το αντικείμενο ειδίκευσής τους και να επικοινωνήσουν άμεσα μαζί τους. Με αυτόν τον τρόπο, μειώνεται η σπατάλη χρόνου και η αβεβαιότητα της αναζήτησης βοήθειας, ενώ παράλληλα ενισχύεται η αλληλοϋποστήριξη και η συνεργασία μεταξύ φοιτητών.

Η εφαρμογή αυτή δεν απευθύνεται μόνο σε φοιτητές που αναζητούν βοήθεια, αλλά και σε εκείνους που επιθυμούν να μοιραστούν τις γνώσεις τους, να εξασκήσουν τις διδακτικές τους ικανότητες ή ακόμη και να χτίσουν ένα προφίλ συνεργασίας και αλληλεπίδρασης που θα τους φανεί χρήσιμο στην μελλοντική επαγγελματική τους πορεία. Ενισχύεται, έτσι, η λογική της αυτοοργάνωσης και της συμμετοχικής μάθησης, στοιχεία που συνάδουν με τις σύγχρονες παιδαγωγικές αντιλήψεις στην ανώτατη εκπαίδευση.

Η πτυχιακή αυτή εργασία περιλαμβάνει το σύνολο των φάσεων σχεδίασης και ανάπτυξης της εφαρμογής: από την αρχική ανάλυση απαιτήσεων και τον καθορισμό των βασικών λειτουργιών, μέχρι την υλοποίηση του συστήματος με τη χρήση τεχνολογιών διαδικτύου αιχμής. Η αρχιτεκτονική της

εφαρμογής βασίζεται στη μοντέρνα προσέγγιση client-server, με τον διαχωρισμό frontend και backend, εξασφαλίζοντας έτσι ευελιξία, επεκτασιμότητα και ευχρηστία.

Η επιλογή των τεχνολογιών έγινε με γνώμονα τη σταθερότητα, την ευκολία συντήρησης και την ευρεία υποστήριξη από την κοινότητα. Ο χρήστης έχει στη διάθεσή του ένα απλό και λειτουργικό περιβάλλον διεπαφής, μέσω του οποίου μπορεί να δημιουργήσει προφίλ, να αναζητήσει διαθέσιμους φοιτητές-διδάσκοντες ανά μάθημα ή θεματική ενότητα, να αξιολογήσει τις υπηρεσίες τους και να πραγματοποιήσει άμεση επικοινωνία.

Σκοπός της εργασίας είναι η ανάδειξη ενός πρακτικού ψηφιακού εργαλείου που να ανταποκρίνεται σε πραγματικές ανάγκες της φοιτητικής κοινότητας, ενισχύοντας τη συνεργασία, την αλληλεγγύη και την ανταλλαγή γνώσεων. Πέρα από την τεχνική της διάσταση, η εφαρμογή αυτή αποτελεί μια πρόταση που προάγει τη μαθησιακή κουλτούρα μέσα στα ιδρύματα τριτοβάθμιας εκπαίδευσης, προσφέροντας στους φοιτητές έναν καινοτόμο και αποδοτικό τρόπο να διαχειρίζονται τις μαθησιακές τους προκλήσεις.

Η ανάπτυξη της εφαρμογής βασίστηκε σε σύγχρονες τεχνολογίες που εξασφαλίζουν αξιοπιστία, ευχρηστία και επεκτασιμότητα. Στο επίπεδο του backend χρησιμοποιήθηκε το Spring Boot, ένα σύγχρονο πλαίσιο ανάπτυξης web εφαρμογών που βασίζεται στη γλώσσα προγραμματισμού Java. Η Java, με τη μακροχρόνια παρουσία και τη σταθερότητά της στον χώρο της πληροφορικής, αποτέλεσε ιδανική επιλογή για την κατασκευή ενός συστήματος που απαιτεί σωστή διαχείριση των δεδομένων και ξεκάθαρη επιχειρησιακή λογική. Το Spring Boot, αξιοποιώντας τα πλεονεκτήματα του οικοσυστήματος της Java, επέτρεψε την απλοποίηση της ανάπτυξης και την ενσωμάτωση βασικών λειτουργιών όπως η ασφάλεια, η διασύνδεση με τη βάση δεδομένων και η επικοινωνία μέσω REST υπηρεσιών.

Παράλληλα, για την πλευρά του χρήστη επιλέχθηκε το Angular, ένα σύγχρονο framework για την κατασκευή δυναμικών και διαδραστικών διεπαφών. Μέσω αυτής της τεχνολογίας, σχεδιάστηκε ένα περιβάλλον φιλικό και ευχάριστο στην πλοήγηση, που επιτρέπει στους φοιτητές να χρησιμοποιούν την εφαρμογή εύκολα και αποτελεσματικά, αλλά επίσης την καθιστά και ευπαρουσίαστη.

Για την αποθήκευση και διαχείριση των δεδομένων επιλέχθηκε το σύστημα βάσης δεδομένων PostgreSQL, το οποίο παρέχει σταθερότητα, καλή απόδοση και ευκολία στη διαχείριση πολύπλοκων δεδομένων. Αποτελεί μια δημοφιλή και αξιόπιστη επιλογή για σύγχρονες διαδικτυακές εφαρμογές, ενώ η ευελιξία και η ευρεία υποστήριξή της από την κοινότητα επιτρέπουν τη διατήρηση και μελλοντική επέκταση της εφαρμογής με ομαλό τρόπο.

1.2 Σκοπός και Δομή της Εργασίας

Σκοπός της παρούσας πτυχιακής εργασίας είναι η σχεδίαση και ανάπτυξη μιας διαδικτυακής εφαρμογής που απευθύνεται σε φοιτητές πανεπιστημίου, προσφέροντας τη δυνατότητα να δημιουργούν και να συμμετέχουν σε φροντιστηριακά μαθήματα, αξιοποιώντας τις γνώσεις και την εμπειρία συμφοιτητών

τους. Η εφαρμογή στοχεύει να καλύψει ένα υπαρκτό κενό που παρατηρείται στον πανεπιστημιακό χώρο: την έλλειψη επίσημου μηχανισμού μέσω του οποίου οι φοιτητές μπορούν να οργανώνουν και να παρακολουθούν υποστηρικτικά μαθήματα σε συνεργασία μεταξύ τους, πέρα από το τυπικό ωρολόγιο πρόγραμμα.

Η υλοποίηση εστιάζει στην ενδοπανεπιστημιακή χρήση και φιλοξενία της εφαρμογής σε πανεπιστημιακό εξυπηρετητή (server), διασφαλίζοντας ότι η πλατφόρμα λειτουργεί εντός των ορίων του ακαδημαϊκού περιβάλλοντος, με έλεγχο ταυτότητας και διατήρηση της ιδιωτικότητας των χρηστών. Η εφαρμογή σχεδιάστηκε με γνώμονα τη λειτουργικότητα, την απλότητα και την επεκτασιμότητα, ώστε να μπορεί να ενταχθεί εύκολα στις ανάγκες διαφόρων τμημάτων και σχολών.

Η εργασία δομείται σε επτά κεφάλαια:

- Κεφάλαιο 1: Παρουσιάζει την εισαγωγή στο αντικείμενο, τη θεματική προσέγγιση και τον σκοπό της εργασίας.
- Κεφάλαιο 2: Αναλύει την ιδέα της εφαρμογής και παραθέτει υπάρχουσες σχετικές πλατφόρμες που αποτέλεσαν σημείο αναφοράς.
- Κεφάλαιο 3: Περιγράφει το τεχνολογικό υπόβαθρο, τις γλώσσες, τα εργαλεία και τα πλαίσια που χρησιμοποιήθηκαν στην υλοποίηση.
- Κεφάλαιο 4: Εστιάζει στην ανάπτυξη της εφαρμογής, αναλύοντας τα user stories, τις λειτουργικές απαιτήσεις, τις οθόνες και τα σενάρια χρήσης.
- Κεφάλαιο 5: Παρουσιάζει τα γενικά συμπεράσματα και ενδεικτικές προτάσεις για μελλοντικές βελτιώσεις ή επεκτάσεις της εφαρμογής.
- Κεφάλαιο 6 (Παράρτημα): Περιλαμβάνει τεχνικά παραρτήματα, όπως αποσπάσματα κώδικα, screenshots, και άλλα υποστηρικτικά στοιχεία.
- Βιβλιογραφία: Παραθέτει τις πηγές και αναφορές που αξιοποιήθηκαν καθ’ όλη τη διάρκεια της συγγραφής και της ανάπτυξης της εργασίας.

Με τον τρόπο αυτό, η εργασία συνδυάζει την ανάλυση αναγκών, την τεχνολογική υλοποίηση και την τεκμηρίωση μιας εφαρμογής που βασίζεται στις αρχές της συμμετοχικής μάθησης και της αλληλοϋποστήριξης, αποτελώντας ένα πρακτικό παράδειγμα αξιοποίησης της τεχνολογίας για την ενίσχυση της φοιτητικής εμπειρίας.

Κεφάλαιο 2ο: Ανάλυση Προβλήματος και Υφιστάμενες Λύσεις

2.1 Εισαγωγή

Η ιδέα για την ανάπτυξη της παρούσας εφαρμογής προέκυψε από μια κοινή εκπαιδευτική ανάγκη που εντοπίζεται συχνά στα πανεπιστήμια: την ανάγκη για αλληλοϋποστήριξη μεταξύ φοιτητών μέσω της ανταλλαγής γνώσεων και εμπειρίας σε μαθήματα αυξημένης δυσκολίας. Πολλοί φοιτητές δυσκολεύονται να ανταπεξέλθουν σε μαθήματα όπως τα Μαθηματικά ή ο Προγραμματισμός, ενώ την ίδια στιγμή άλλοι συμφοιτητές τους έχουν αποκτήσει ισχυρό υπόβαθρο στα ίδια αντικείμενα. Η αξιοποίηση αυτής της δυνητικής σχέσης μέσα από μια οργανωμένη ψηφιακή πλατφόρμα ήταν το έναυσμα για τη συγκεκριμένη εργασία.

Η εφαρμογή που προτείνεται λειτουργεί ως ενδοπανεπιστημιακή υπηρεσία, μέσω της οποίας ένας φοιτητής μπορεί είτε να προσφέρει είτε να λάβει μαθήματα σε ατομικό ή ομαδικό επίπεδο. Με άλλα λόγια, κάθε χρήστης μπορεί να δημιουργήσει ένα “μάθημα” στο οποίο δηλώνει τη διαθεσιμότητά του και την εξειδίκευσή του σε συγκεκριμένο γνωστικό αντικείμενο (π.χ. Java, Ανάλυση Ι, Δομές Δεδομένων), ενώ παράλληλα μπορεί να δηλώσει συμμετοχή σε άλλα διαθέσιμα μαθήματα ως “μαθητής”.

Η πλατφόρμα έχει σχεδιαστεί με σκοπό να λειτουργεί σε περιβάλλον εντός του πανεπιστημιακού δικτύου (self-hosted σε πανεπιστημιακό server), προσφέροντας ένα ασφαλές και πλήρως ελεγχόμενο οικοσύστημα. Αυτό επιτρέπει τη διασφάλιση της ιδιωτικότητας των χρηστών, τον έλεγχο της ποιότητας του περιεχομένου, και την απουσία εξωτερικών παρεμβολών (όπως διαφημίσεις ή εμπορικές χρεώσεις).

Η εφαρμογή ενσωματώνει βασικές λειτουργίες όπως:

- Καταχώριση και περιγραφή μαθημάτων από χρήστες με ρόλο “διδάσκοντα”.
- Αναζήτηση και συμμετοχή σε διαθέσιμα μαθήματα από χρήστες με ρόλο “μαθητή”.
- Υποστήριξη για μαθήματα τύπου 1-προς-1 ή σε μικρές ομάδες.
- Προγραμματισμός συναντήσεων, είτε δια ζώσης εντός του πανεπιστημίου είτε μέσω online μέσων (όπως Zoom ή Google Meet).
- Αξιολόγηση μαθημάτων και χρηστών για ενίσχυση της ποιότητας.
- Επαλήθευση ταυτότητας μέσω πανεπιστημιακού λογαριασμού.

Η υλοποίηση της εφαρμογής θα βασιστεί σε σύγχρονες τεχνολογίες web, ενώ ιδιαίτερη βαρύτητα δίνεται στη διασφάλιση της ευχρηστίας και της ταχύτητας απόκρισης, ώστε η πλατφόρμα να μπορεί να χρησιμοποιηθεί εύκολα από οποιονδήποτε φοιτητή ανεξαρτήτως τεχνικού υποβάθρου.

Η συγκεκριμένη εργασία επιδιώκει όχι μόνο να καλύψει ένα τεχνολογικό κενό εντός της πανεπιστημιακής κοινότητας, αλλά και να προτείνει ένα πρακτικό εργαλείο ενίσχυσης της συνεργατικής μάθησης και της φοιτητικής αυτοοργάνωσης.

2.2 Καταγραφή υπαρχουσών πλατφορμών

Η ραγδαία ανάπτυξη της τεχνολογίας και η ανάγκη εξατομικευμένης υποστήριξης στη μάθηση έχουν οδηγήσει στη δημιουργία εξειδικευμένων ψηφιακών πλατφορμών peer tutoring, δηλαδή εφαρμογών που επιτρέπουν τη διδασκαλία μεταξύ συνομηλίκων. Στο πλαίσιο της παρούσας εργασίας, κρίνεται απαραίτητη η μελέτη επιλεγμένων παραδειγμάτων τέτοιων εφαρμογών, ώστε να εντοπιστούν λειτουργίες και τεχνολογικές προσεγγίσεις που μπορούν να αξιοποιηθούν ή να προσαρμοστούν στο υπό ανάπτυξη σύστημα.

2.2.1 Knack

Το Knack αποτελεί μια εμπορική πλατφόρμα που έχει υιοθετηθεί από αρκετά πανεπιστήμια των ΗΠΑ για την παροχή υπηρεσιών peer tutoring. Οι φοιτητές μπορούν να προσφέρουν αμειβόμενες ή εθελοντικές συνεδρίες διδασκαλίας σε συμφοιτητές τους, μέσω ενός συστήματος αξιολόγησης και αντιστοίχισης βασισμένου στην επίδοσή τους σε συγκεκριμένα μαθήματα. Η πλατφόρμα λειτουργεί σε συνεργασία με τα ακαδημαϊκά ιδρύματα και προσφέρει οργανωμένη και ασφαλή διεπαφή τόσο για φοιτητές όσο και για διαχειριστές [35].

2.2.2 GoPeer

Το GoPeer είναι μια διαδικτυακή πλατφόρμα που συνδέει μαθητές πρωτοβάθμιας και δευτεροβάθμιας εκπαίδευσης (ηλικίες 5–18) με φοιτητές από πανεπιστήμια υψηλού κύρους, οι οποίοι προσφέρουν ιδιωτικά μαθήματα εξ αποστάσεως. Αν και η κύρια του στόχευση δεν αφορά πανεπιστήμια, η τεχνική του υλοποίηση και η λειτουργική του δομή αποτελούν αξιόλογο σημείο αναφοράς. Η πλατφόρμα παρέχει διαδραστικά εργαλεία, όπως εικονικό πίνακα, δυνατότητα κοινής χρήσης εγγράφων και ιστορικό προηγούμενων συνεδριών. Το GoPeer υιοθετεί εμπορικό μοντέλο, με τις υπηρεσίες να προσφέρονται επί πληρωμή, αλλά με υψηλό βαθμό ασφάλειας και επαγγελματισμού [36].

2.2.3 Schoolhouse.world

Η Schoolhouse.world αποτελεί μια εθελοντική, μη κερδοσκοπική πλατφόρμα που ιδρύθηκε από τον Sal Khan, δημιουργό της Khan Academy. Προσφέρει δωρεάν tutoring σε μαθητές και φοιτητές παγκοσμίως, κυρίως σε μαθήματα μαθηματικών και προετοιμασίας εξετάσεων SAT. Οι διδάσκοντες είναι εθελοντές που έχουν αξιολογηθεί ως προς την επάρκειά τους και λειτουργούν βάσει προγράμματος μέσω Zoom. Η πλατφόρμα παρέχει επίσης σύστημα διακρίσεων (badges) για την επιβράβευση των tutors, καθώς και στατιστικά στοιχεία για τη βελτίωση της διδακτικής ποιότητας. Η φιλοσοφία της πλατφόρμας βασίζεται στην ελεύθερη πρόσβαση και στην ενίσχυση της μαθησιακής κοινότητας σε διεθνές επίπεδο [37].

2.2.4 YTeach

Το YTeach είναι μια peer tutoring εφαρμογή που απευθύνεται κυρίως σε μαθητές λυκείου, προσφέροντας ένα διαδραστικό περιβάλλον μέσω του οποίου μπορούν να δηλώσουν είτε την ανάγκη τους για βοήθεια είτε τη διάθεσή τους να προσφέρουν μαθήματα. Η πλατφόρμα έχει σχεδιαστεί με έμφαση στη φιλικότητα προς κινητές συσκευές και δίνει έμφαση στην εμπειρία χρήσης μέσω ενός μηχανισμού κρατήσεων, αντιστοίχισης, υπενθυμίσεων και συγκέντρωσης πόντων. Το στοιχείο της παιχνιδοποίησης (gamification) είναι εμφανές, καθώς η συμμετοχή των μαθητών επιβραβεύεται με εικονικά credits. Παράλληλα, παρέχονται στατιστικά συμμετοχής, τα οποία μπορούν να αξιοποιηθούν και ως στοιχεία κοινωνικής προσφοράς [38].

2.2.5 Tutorpeers

Το Tutorpeers είναι μια σύγχρονη πλατφόρμα peer tutoring που δημιουργήθηκε από φοιτητές για φοιτητές. Επιτρέπει σε κάθε χρήστη να δημιουργήσει προφίλ, να ορίσει τη διαθεσιμότητά του, να καθορίσει γνωστικά πεδία στα οποία έχει εξειδίκευση, και να προσφέρει ή να λάβει εκπαιδευτική υποστήριξη. Η πλατφόρμα ενσωματώνει απλό σύστημα μηνυμάτων και αξιολογήσεων, ενώ οι συνεδρίες μπορούν να πραγματοποιούνται τόσο online όσο και δια ζώσης. Παρότι δεν ενσωματώνεται σε πανεπιστημιακά πληροφοριακά συστήματα, υποστηρίζει ένα μοντέλο που προωθεί την αυτοοργάνωση και τη συνεργασία εντός της φοιτητικής κοινότητας [39].

2.2.6 OATutor

Το OATutor διαφέρει από τις προηγούμενες πλατφόρμες, καθώς δεν βασίζεται στην ανθρώπινη διάδραση αλλά στην προσαρμοστική τεχνητή νοημοσύνη (adaptive tutoring). Πρόκειται για ένα έργο

ανοικτού κώδικα που έχει σχεδιαστεί για να ενσωματώνεται σε Learning Management Systems (LMS) και να προσφέρει εξατομικευμένο περιεχόμενο με βάση τις απαντήσεις του χρήστη. Η τεχνολογία του βασίζεται σε τεχνικές machine learning, επιτρέποντας στο σύστημα να εντοπίζει τα αδύναμα σημεία του μαθητή και να προσαρμόζει τη ροή των ερωτήσεων. Η χρήση του OATutor είναι εντελώς δωρεάν και η φιλοσοφία του το καθιστά κατάλληλο για ερευνητική αξιοποίηση ή προσαρμογή σε εκπαιδευτικά ιδρύματα [40].

Κεφάλαιο 3ο: Τεχνολογικό υπόβαθρο

3.1 Εισαγωγή

Η ανάπτυξη της εφαρμογής βασίστηκε σε ένα σύνολο σύγχρονων τεχνολογιών, οι οποίες επιλέχθηκαν με γνώμονα τη σταθερότητα, την επεκτασιμότητα και την ευρεία υποστήριξη από την παγκόσμια κοινότητα προγραμματιστών. Το τεχνολογικό υπόβαθρο διακρίνεται σε τρία βασικά υποσυστήματα: το backend, το οποίο αναλαμβάνει την επιχειρησιακή λογική και τη διαχείριση των δεδομένων· το frontend, που είναι υπεύθυνο για την αλληλεπίδραση με τον χρήστη· και τη βάση δεδομένων, στην οποία αποθηκεύεται και οργανώνεται η πληροφορία που διακινείται από και προς την εφαρμογή.

Κάθε επιμέρους υποσύστημα υλοποιήθηκε με τη χρήση καθιερωμένων τεχνολογιών ανοιχτού κώδικα, αξιοποιώντας δοκιμασμένες μεθόδους και πρότυπα ανάπτυξης. Στις ενότητες που ακολουθούν παρουσιάζονται αναλυτικά οι τεχνολογίες που χρησιμοποιήθηκαν σε κάθε τμήμα του συστήματος, καθώς και οι λόγοι που οδήγησαν στην επιλογή τους.

Η υλοποίηση του backend της εφαρμογής βασίστηκε στη γλώσσα προγραμματισμού Java και στο πλαίσιο Spring Boot. Οι τεχνολογίες αυτές επελέγησαν για τη σταθερότητά τους, την ευρεία χρήση τους στη βιομηχανία και την ευκολία που προσφέρουν στην ανάπτυξη RESTful διαδικτυακών υπηρεσιών. Παρακάτω περιγράφονται αναλυτικά τα βασικά χαρακτηριστικά και ο ρόλος της κάθε τεχνολογίας στην εφαρμογή.

3.2 Backend

Το backend της εφαρμογής είναι υπεύθυνο για την επιχειρησιακή λογική, τη διαχείριση των δεδομένων και την εξυπηρέτηση των αιτημάτων που προέρχονται από τον χρήστη μέσω της διεπαφής. Η υλοποίησή του βασίστηκε σε ένα σύνολο σύγχρονων τεχνολογιών, με στόχο τη δημιουργία ενός σταθερού, επεκτάσιμου και ασφαλούς συστήματος. Ο βασικός κορμός αποτελείται από τη γλώσσα προγραμματισμού Java και το πλαίσιο Spring Boot, ενώ αξιοποιήθηκαν επιπλέον εργαλεία για την επικοινωνία με τη βάση δεδομένων, τη διαχείριση ασφάλειας, την έγκυρη μεταφορά δεδομένων και τη συντήρηση της βάσης.

Στις υποενότητες που ακολουθούν παρουσιάζονται αναλυτικά οι τεχνολογίες που χρησιμοποιήθηκαν στο backend της εφαρμογής και ο ρόλος που διαδραματίζει η καθεμία στη συνολική αρχιτεκτονική του συστήματος.

3.2.1 Java

Η υλοποίηση του backend της εφαρμογής βασίστηκε στη γλώσσα προγραμματισμού Java, η οποία αποτελεί μία από τις πλέον διαδεδομένες, ώριμες και σταθερές τεχνολογίες στον χώρο της ανάπτυξης λογισμικού. Αναπτύχθηκε αρχικά από τη Sun Microsystems το 1995 και, από το 2010, συντηρείται από

την Oracle Corporation [1]. Η Java εισήγαγε θεμελιώδεις έννοιες όπως η αντικειμενοστραφής προσέγγιση, η στατική τυποποίηση και η φορητότητα του κώδικα, με σκοπό τη δημιουργία ασφαλών, επαναχρησιμοποιήσιμων και συντηρήσιμων εφαρμογών [2].

Ένα από τα πιο ισχυρά χαρακτηριστικά της Java είναι η Java Virtual Machine (JVM), η οποία επιτρέπει την εκτέλεση bytecode σε διαφορετικά λειτουργικά συστήματα και αρχιτεκτονικές, χωρίς την ανάγκη επαναμεταγλώττισης. Αυτό το χαρακτηριστικό συνοψίζεται εύστοχα στη φράση *«Write Once, Run Anywhere»* [3]. Η JVM παρέχει επίσης υποσυστήματα όπως garbage collection, memory management και απομόνωση νημάτων (threading), τα οποία συνεισφέρουν στην αξιοπιστία και την ασφάλεια των εφαρμογών.

Η Java Platform περιλαμβάνει, εκτός από τη γλώσσα, και ένα πλούσιο σύνολο APIs και εργαλείων, όπως το Java Standard Edition (SE) για γενική ανάπτυξη εφαρμογών, το Java Enterprise Edition (EE) για εφαρμογές μεγάλης κλίμακας και το Java Development Kit (JDK) που περιλαμβάνει compiler, debugger και άλλα βασικά εργαλεία. Η υποστήριξη από ευρέως διαδεδομένα IDEs όπως το IntelliJ IDEA και το Eclipse, καθώς και η χρήση εργαλείων αυτοματισμού όπως Maven και Gradle, καθιστούν την Java ιδιαίτερα παραγωγική για ομάδες ανάπτυξης.

Η σταθερότητα, η προβλεψιμότητα της συμπεριφοράς, η ευρεία κοινότητα υποστήριξης και η συνεχής εξέλιξή της, με σύγχρονες δυνατότητες όπως `records`, `pattern matching`, `var` και `stream API`, την έχουν διατηρήσει στις κορυφαίες θέσεις των πιο δημοφιλών γλωσσών προγραμματισμού παγκοσμίως.

Στην παρούσα εφαρμογή, η Java χρησιμοποιήθηκε ως η κύρια γλώσσα υλοποίησης του backend, καλύπτοντας την επιχειρησιακή λογική, την επεξεργασία των αιτημάτων των χρηστών, την επικοινωνία με τη βάση δεδομένων και την οργάνωση της εφαρμογής σε αρθρωτή αρχιτεκτονική. Ο κώδικας δομήθηκε σε σαφώς διαχωρισμένα επίπεδα (Controller, Service, Repository), ακολουθώντας τις αρχές της καθαρής αρχιτεκτονικής (Clean Architecture), διευκολύνοντας έτσι τη συντηρησιμότητα και τη δυνατότητα δοκιμών.

Η επιλογή της Java κρίθηκε ιδιαίτερα κατάλληλη για το συγκεκριμένο έργο, καθώς προσέφερε σταθερό περιβάλλον εκτέλεσης, πληθώρα διαθέσιμων βιβλιοθηκών και φυσική συνεργασία με το Spring οικοσύστημα. Ο συνδυασμός της με το Spring Boot, που περιγράφεται στην επόμενη ενότητα, συνέβαλε στην επιτάχυνση της ανάπτυξης, την ελαχιστοποίηση του boilerplate κώδικα και την ενίσχυση της επεκτασιμότητας της εφαρμογής.

3.2.2 Spring Boot

Το Spring Boot είναι ένα πλαίσιο ανάπτυξης που βασίζεται στο οικοσύστημα του Spring και έχει σχεδιαστεί για να απλοποιεί τη διαδικασία δημιουργίας και διαχείρισης εφαρμογών Java. Παρέχει ένα σύνολο εργαλείων, προκαθορισ μένων ρυθμίσεων (opinionated defaults) και μηχανισμών αυτόματης

διαμόρφωσης, που επιτρέπουν την ταχεία υλοποίηση λειτουργικών εφαρμογών με ελάχιστη ανάγκη για χειροκίνητη παραμετροποίηση.

Ένα από τα κύρια χαρακτηριστικά του Spring Boot είναι η αυτόματη διαμόρφωση (auto-configuration), μέσω της οποίας το πλαίσιο μπορεί να ενεργοποιεί ή να παραμετροποιεί αυτόματα beans και ρυθμίσεις, βασισμένο στις εξαρτήσεις που έχουν δηλωθεί στο έργο και στις τιμές των αρχείων ρυθμίσεων [4]. Για παράδειγμα, αν εντοπιστεί σύνδεση με βάση δεδομένων και παρουσία του Spring Data JPA, το Spring Boot δημιουργεί αυτόματα τα απαραίτητα components για πρόσβαση και επικοινωνία με τη βάση. Αυτή η συμπεριφορά μειώνει δραστικά τον απαιτούμενο boilerplate κώδικα και διευκολύνει τους προγραμματιστές στο να επικεντρωθούν στη λογική της εφαρμογής.

Επιπλέον, το Spring Boot προσφέρει ενσωματωμένους διακομιστές εφαρμογών, όπως τον Tomcat, επιτρέποντας την εκτέλεση της εφαρμογής ως αυτόνομης μονάδας, χωρίς ανάγκη για εξωτερικό application server [5]. Αυτή η δυνατότητα καθιστά το Spring Boot ιδανικό εργαλείο για την ανάπτυξη μικροϋπηρεσιών, αφού κάθε υπηρεσία μπορεί να εκτελείται αυτόνομα, με δικό της web server, endpoints και ρυθμίσεις.

Σημαντικό πλεονέκτημα του Spring Boot είναι επίσης το γεγονός ότι ενσωματώνει πλήρως όλα τα βασικά υποσυστήματα του Spring (όπως Spring Security, Spring Data, Spring MVC κ.ά.), και προσφέρει “starters” — πακέτα έτοιμων εξαρτήσεων που επιταχύνουν τη ρύθμιση νέων modules. Η ύπαρξη του Spring Boot CLI επιτρέπει την εκτέλεση απλών εφαρμογών από τη γραμμή εντολών, ενισχύοντας την παραγωγικότητα και μειώνοντας τον χρόνο αρχικοποίησης [32].

Στην παρούσα εφαρμογή, το Spring Boot χρησιμοποιήθηκε ως θεμέλιο για την υλοποίηση RESTful υπηρεσιών, την ενσωμάτωση με τη βάση δεδομένων μέσω Spring Data JPA, και τη διαχείριση της ασφάλειας με χρήση Spring Security. Η δομή της εφαρμογής βασίστηκε σε αρθρωτή αρχιτεκτονική (Controller, Service, Repository), με σαφή διαχωρισμό ευθυνών. Η δυνατότητα ενσωμάτωσης εργαλείων όπως το Flyway για τη διαχείριση μεταναστεύσεων και του Actuator για παρακολούθηση της εφαρμογής σε πραγματικό χρόνο, καθιστά το Spring Boot ένα πλήρες εργαλείο για σύγχρονες web εφαρμογές.

Η επιλογή του Spring Boot απέδειξε τη χρησιμότητά της τόσο σε επίπεδο ευκολίας όσο και σε επίπεδο επεκτασιμότητας, επιτρέποντας την ταχύτατη ανάπτυξη, την ευκολία συντήρησης και την απρόσκοπτη συνεργασία μεταξύ επιμέρους τεχνολογικών επιπέδων της εφαρμογής.

3.2.2.1 Spring Boot Mail Starter

Για την αποστολή email εντός της εφαρμογής, αξιοποιήθηκε το Spring Boot Mail Starter, μια εξάρτηση που προσφέρει άμεση διασύνδεση με SMTP servers και απλό API για την αποστολή HTML ή plain text

μηνυμάτων. Χρησιμοποιήθηκε κυρίως για λειτουργίες που σχετίζονται με την επαναφορά κωδικού πρόσβασης (password reset) και την επικοινωνία με τον χρήστη μέσω email.

Η ενσωμάτωσή του έγινε με βασικές ρυθμίσεις στον application.yml, ενώ η χρήση του API του Spring επέτρεψε την εύκολη δημιουργία και αποστολή προσωποποιημένων email, ενισχύοντας τη χρηστικότητα και τη λειτουργικότητα της εφαρμογής. Η χρήση του συνέβαλε στην υλοποίηση μιας πιο ολοκληρωμένης εμπειρίας χρήστη, ειδικά σε περιπτώσεις όπου απαιτείται επιβεβαίωση ή ενημέρωση μέσω ηλεκτρονικής αλληλογραφίας [33].

3.2.3 Spring Data JPA

Το Spring Data JPA είναι ένα υποσύνολο του οικοσυστήματος Spring Data που στοχεύει στην απλοποίηση της πρόσβασης σε σχεσιακές βάσεις δεδομένων μέσω του Java Persistence API (JPA). Αποτελεί ένα επίπεδο αφαίρεσης πάνω από τον JPA provider (όπως το Hibernate), επιτρέποντας την υλοποίηση αποθετηρίων (repositories) με ελάχιστο ή καθόλου boilerplate κώδικα. Αυτό επιτυγχάνεται μέσω της χρήσης προκαθορισμένων μεθόδων και της αυτόματης δημιουργίας ερωτημάτων βάσει της ονοματολογίας των μεθόδων [6].

Ένα από τα βασικά χαρακτηριστικά του Spring Data JPA είναι η δυνατότητα δημιουργίας ερωτημάτων με βάση τα ονόματα των μεθόδων. Για παράδειγμα, μια μέθοδος με όνομα *findByUsername* θα μετατραπεί αυτόματα σε ένα κατάλληλο SQL ερώτημα που αναζητά εγγραφές με βάση το πεδίο *username*. Αυτό μειώνει σημαντικά την ανάγκη για γραφή χειροκίνητων SQL ερωτημάτων και ενισχύει την παραγωγικότητα του προγραμματιστή [7].

Επιπλέον, το Spring Data JPA υποστηρίζει τη χρήση της ανατοποθέτησης ερωτημάτων μέσω του annotation *@Query*, επιτρέποντας την εκτέλεση πιο σύνθετων ή ειδικών ερωτημάτων όταν η ονοματολογία των μεθόδων δεν επαρκεί. Αυτό παρέχει ευελιξία στην υλοποίηση της πρόσβασης στα δεδομένα, διατηρώντας παράλληλα την καθαρότητα και την αναγνωσιμότητα του κώδικα [8].

Στην παρούσα εφαρμογή, το Spring Data JPA χρησιμοποιήθηκε για την υλοποίηση του επιπέδου πρόσβασης στα δεδομένα, επιτρέποντας την εύκολη διαχείριση των οντοτήτων και των σχέσεών τους, καθώς και την εκτέλεση σύνθετων ερωτημάτων με απλό και κατανοητό τρόπο.

3.2.4 Spring Security και JWT

Η ασφάλεια της εφαρμογής υλοποιήθηκε μέσω της ενσωμάτωσης του Spring Security με την τεχνολογία JSON Web Tokens (JWT), προσφέροντας ένα σύγχρονο και αποδοτικό μοντέλο αυθεντικοποίησης και εξουσιοδότησης για RESTful APIs.

Το Spring Security αποτελεί μία από τις πιο διαδεδομένες και καθιερωμένες λύσεις για την ασφάλεια εφαρμογών Java, παρέχοντας μηχανισμούς αυθεντικοποίησης, εξουσιοδότησης και προστασίας από κοινές επιθέσεις. Ενσωματώνεται άψογα με το Spring Boot και υποστηρίζει διάφορες στρατηγικές ασφάλειας, συμπεριλαμβανομένης της χρήσης JWT για stateless authentication [9].

Τα JWT είναι αυτοτελή tokens που περιέχουν πληροφορίες χρήστη και υπογράφονται ψηφιακά, επιτρέποντας την επαλήθευση της αυθεντικότητας και της ακεραιότητας των δεδομένων χωρίς την ανάγκη αποθήκευσης κατάστασης στον διακομιστή. Αποτελούνται από τρία μέρη (header, payload και signature) και χρησιμοποιούνται ευρέως για την υλοποίηση αυθεντικοποίησης και εξουσιοδότησης σε σύγχρονες web εφαρμογές [10].

Στην παρούσα εφαρμογή, η διαδικασία αυθεντικοποίησης λειτουργεί ως εξής: ο χρήστης υποβάλλει τα διαπιστευτήριά του, και αν η είσοδος είναι επιτυχής, δημιουργείται ένα JWT που επιστρέφεται στον client. Στη συνέχεια, ο χρήστης περιλαμβάνει το token σε κάθε αίτημα μέσω του HTTP header "Authorization". Το Spring Security επαληθεύει την εγκυρότητα του token και αποδίδει τα κατάλληλα δικαιώματα πρόσβασης στον χρήστη [11].

Η χρήση του Spring Security σε συνδυασμό με JWT προσφέρει ένα ασφαλές, επεκτάσιμο και ανεξάρτητο από συνεδρίες μοντέλο αυθεντικοποίησης, ιδανικό για σύγχρονες RESTful εφαρμογές.

3.2.5 Προγραμματιστικά μοτίβα και Vavr

Η εφαρμογή αξιοποιεί σε ορισμένα σημεία τεχνικές του **λειτουργικού προγραμματισμού** (functional programming), προκειμένου να βελτιωθεί η σαφήνεια, η εκφραστικότητα και η διαχείριση των ροών του backend. Ο λειτουργικός προγραμματισμός αποτελεί ένα παραδειγματικό προγραμματιστικό μοντέλο που βασίζεται στη χρήση καθαρών συναρτήσεων, στην απουσία παρενεργειών (side effects) και στη χρήση αμετάβλητων δομών δεδομένων [12]. Αν και η Java είναι κατά βάση αντικειμενοστραφής γλώσσα, από την έκδοση 8 και μετά υποστηρίζει βασικά στοιχεία λειτουργικού προγραμματισμού, όπως lambda expressions και streams.

Για να ενισχυθεί περαιτέρω αυτή η προσέγγιση, στην εφαρμογή χρησιμοποιήθηκε η βιβλιοθήκη **Vavr**. Η Vavr προσφέρει μια σειρά από **λειτουργικές δομές δεδομένων και συναρτήσεις υψηλότερης τάξης**, οι οποίες επεκτείνουν τη γλώσσα Java με χαρακτηριστικά που συναντώνται σε γλώσσες όπως Scala ή Haskell. Ενδεικτικά, η βιβλιοθήκη περιλαμβάνει τύπους όπως Option (για ασφαλή διαχείριση null τιμών), Try (για διαχείριση εξαιρέσεων χωρίς χρήση try-catch), Either (για επιστροφή πολλαπλών πιθανών αποτελεσμάτων) και Tuple (για ομαδοποίηση δεδομένων) [13].

Η Vavr χρησιμοποιήθηκε σε περιπτώσεις όπου κρίθηκε σκόπιμο να εφαρμοστούν λειτουργικά μοτίβα: όπως στην ασφαλή επιστροφή αποτελεσμάτων, στην καθαρή διαχείριση σφαλμάτων και στην αποφυγή πλεονασμού λογικής ελέγχου. Η ενσωμάτωσή της προσέφερε ευανάγνωστο και πιο "προβλέψιμο"

κώδικα, χωρίς να διαταράσσει τον αντικειμενοστραφή χαρακτήρα της εφαρμογής, αλλά λειτουργώντας συμπληρωματικά [14].

Η επιλογή αυτής της προσέγγισης αντικατοπτρίζει τη σύγχρονη τάση για σύνθεση διαφορετικών προγραμματιστικών παραδειγμάτων μέσα στην ίδια εφαρμογή, με σκοπό την καλύτερη προσαρμογή των εργαλείων στη φύση του εκάστοτε προβλήματος.

3.2.6 Project Lombok

Η βιβλιοθήκη Lombok χρησιμοποιήθηκε για την αυτόματη παραγωγή επαναλαμβανόμενου κώδικα σε κλάσεις του backend, όπως getters, setters, constructors, equals, hashCode και toString μεθόδους. Μέσω της χρήσης annotations, όπως @Getter, @Setter, @NoArgsConstructor, @AllArgsConstructor και @Builder, επιτεύχθηκε μείωση του boilerplate κώδικα και ενίσχυση της καθαρότητας και αναγνωσιμότητας των οντοτήτων [15].

Η Lombok ενσωματώνεται στη φάση της μεταγλώττισης και δεν επηρεάζει τον παραγόμενο bytecode, γεγονός που την καθιστά διαφανή προς το περιβάλλον εκτέλεσης. Η χρήση της συνέβαλε ιδιαίτερα στην ευκολότερη συντήρηση του κώδικα και στην αποφυγή σφαλμάτων που μπορεί να προκύψουν από χειροκίνητη υλοποίηση επαναλαμβανόμενων μεθόδων.

3.2.7 Hypersistence Utils

Το Hypersistence Utils είναι μια βιβλιοθήκη ανοιχτού κώδικα που επεκτείνει τη λειτουργικότητα του Hibernate, προσφέροντας εργαλεία για advanced τεχνικές όπως DTO projections, dynamic fetching, pagination, soft deletes και attribute converters [16]. Αν και δεν αποτέλεσε κεντρικό στοιχείο της αρχιτεκτονικής της εφαρμογής, χρησιμοποιήθηκε επιλεκτικά σε περιπτώσεις όπου οι δυνατότητες της JPA ήταν περιοριστικές.

Η ενσωμάτωσή του παρείχε λύσεις χωρίς την ανάγκη για χειροκίνητη εγγραφή πολύπλοκων SQL queries, βελτιώνοντας έτσι τη συντηρησιμότητα του persistence layer. Χρησιμοποιήθηκε, για παράδειγμα, για pagination σε custom queries που δεν μπορούσαν να καλυφθούν μόνο από το Spring Data JPA.

3.2.8 Flyway

Για τη διαχείριση των μεταβολών στο σχήμα της βάσης δεδομένων χρησιμοποιήθηκε το εργαλείο Flyway, ένα ελαφρύ και ευέλικτο σύστημα διαχείρισης μεταναστεύσεων βάσεων (database migrations). Το Flyway επιτρέπει την έλεγχομενη εξέλιξη της βάσης δεδομένων μέσα από versioned SQL scripts ή Java classes, τα οποία εκτελούνται με προβλέψιμο τρόπο κατά την εκκίνηση της εφαρμογής [34].

Η λειτουργία του βασίζεται στην παρακολούθηση της κατάστασης της βάσης μέσω ενός εσωτερικού πίνακα (flyway_schema_history), όπου καταγράφονται τα εκτελεσμένα scripts. Έτσι διασφαλίζεται ότι κάθε migration εφαρμόζεται μία και μόνο φορά και με σωστή σειρά. Στην παρούσα εφαρμογή, το Flyway χρησιμοποιήθηκε για τη δημιουργία και σταδιακή τροποποίηση πινάκων όπως user, course, enrollment, διατηρώντας ταυτόχρονα τη συνέπεια μεταξύ διαφορετικών περιβαλλόντων ανάπτυξης και παραγωγής.

Η ενσωμάτωσή του έγινε μέσω της εξάρτησης flyway-core στο αρχείο pom.xml και ρυθμίστηκε μέσω του αρχείου application.yml, με ορισμό των credentials και του path των migration scripts. Το Flyway προσφέρει επίσης δυνατότητες για rollback, repeatable migrations και καθαρισμό της βάσης, αν και στην παρούσα εργασία έγινε χρήση μόνο των βασικών versioned scripts.

Η χρήση του Flyway συνέβαλε στην αυτοματοποίηση των αλλαγών της βάσης δεδομένων και ενίσχυσε τη συνεργατική ανάπτυξη, εξασφαλίζοντας ότι όλοι οι προγραμματιστές εργάζονται πάνω στην ίδια έκδοση του σχήματος της βάσης, μειώνοντας σημαντικά τα λάθη και τις ασυνέπειες.

3.3 Frontend

3.3.1 Εισαγωγή

Το frontend της εφαρμογής αναπτύχθηκε με στόχο να προσφέρει μια σύγχρονη, γρήγορη και φιλική προς τον χρήστη εμπειρία. Η διεπαφή χρήστη αποτελεί τον κύριο δίαυλο επικοινωνίας ανάμεσα στον τελικό χρήστη και το σύστημα, και για τον λόγο αυτόν δόθηκε ιδιαίτερη έμφαση στην εργονομία, την ανταπόκριση και τη σαφή παρουσίαση της πληροφορίας.

Για την υλοποίηση του frontend επιλέχθηκε το Angular, ένα πλήρες framework που αναπτύσσεται και υποστηρίζεται από τη Google. Το Angular προσφέρει μια ισχυρή δομή για την κατασκευή δυναμικών, modular και επεκτάσιμων web εφαρμογών μονής σελίδας (Single Page Applications – SPAs). Συνδυάζει τη χρήση της γλώσσας TypeScript, υποστήριξη για reactive προγραμματισμό, δισδιάστατη επικοινωνία δεδομένων (two-way data binding), καθώς και ένα ισχυρό σύστημα δρομολόγησης (routing) και διαχείρισης κατάστασης.

Η επιλογή του Angular βασίστηκε στην πληρότητά του ως framework, στην ευρεία τεκμηρίωση και στην υποστήριξη που προσφέρει για αρχιτεκτονική component-based, η οποία ενισχύει την επαναχρησιμοποίηση και τη συντηρησιμότητα του κώδικα. Οι βασικές διεπαφές της εφαρμογής υλοποιήθηκαν ως επαναχρησιμοποιήσιμα components, ενώ η διαχείριση της κατάστασης, των forms και των routes οργανώθηκε με τρόπο που εξυπηρετεί τη λογική και τη ροή της εφαρμογής.

3.3.2 JavaScript/TypeScript

Το frontend της εφαρμογής υλοποιήθηκε με χρήση της JavaScript και της TypeScript, δύο στενά συνδεδεμένων τεχνολογιών που αποτελούν τον πυρήνα της σύγχρονης ανάπτυξης web εφαρμογών. Η JavaScript είναι η κύρια γλώσσα προγραμματισμού του ιστού, ενώ η TypeScript είναι μια υπερσύνολό της που εισάγει επιπλέον χαρακτηριστικά όπως στατική τυποποίηση, καθαρές κλάσεις και interfaces.

Η JavaScript είναι μια δυναμική και ευέλικτη γλώσσα, η οποία υποστηρίζει πολλαπλά προγραμματιστικά παραδείγματα – αντικειμενοστραφή, λειτουργικά και επαναχρησιμοποιήσιμα. Χρησιμοποιείται εκτενώς για την ανάπτυξη διαδραστικών διεπαφών και προσφέρει τη δυνατότητα δημιουργίας εφαρμογών που λειτουργούν απευθείας στον φυλλομετρητή του χρήστη, χωρίς την ανάγκη επικοινωνίας με τον διακομιστή για κάθε ενέργεια [17].

Η TypeScript δημιουργήθηκε από τη Microsoft και επεκτείνει τη JavaScript με δυνατότητες όπως στατικά καθορισμένοι τύποι μεταβλητών, ασφαλέστερη διαχείριση σφαλμάτων και καλύτερη υποστήριξη εργαλείων (IDEs, editors). Ενισχύει τον προγραμματιστή στην κατανόηση και την πρόληψη σφαλμάτων σε μεγαλύτερης κλίμακας εφαρμογές, διευκολύνοντας τη συντήρηση και την αναβάθμιση του κώδικα [18].

Η χρήση της TypeScript ήταν ιδιαίτερα επωφελής στην παρούσα εφαρμογή, καθώς ενσωματώνεται πλήρως με το Angular και επιτρέπει την ανάπτυξη modular, επεκτάσιμων και τυποποιημένων components. Παράλληλα, επιτρέπει τον σαφή διαχωρισμό μεταξύ τύπων, μοντέλων δεδομένων και λειτουργιών του UI.

Ο συνδυασμός των δύο τεχνολογιών οδήγησε σε ένα ισχυρό και αξιόπιστο περιβάλλον ανάπτυξης, όπου η JavaScript παρέχει τη βάση, ενώ η TypeScript προσφέρει το απαιτούμενο επίπεδο δομής και σταθερότητας για την υλοποίηση μιας σύγχρονης, διαδραστικής εμπειρίας χρήστη.

3.3.3 Angular

Το Angular είναι ένα σύγχρονο, πλήρως εξοπλισμένο πλαίσιο ανάπτυξης διαδικτυακών εφαρμογών μονής σελίδας (*Single Page Applications – SPAs*), το οποίο αναπτύσσεται και υποστηρίζεται από τη Google. Είναι γραμμένο σε TypeScript και προσφέρει ένα ολοκληρωμένο σύνολο εργαλείων και λειτουργιών για τη δημιουργία δομημένων, επεκτάσιμων και υψηλής απόδοσης εφαρμογών [19].

Ένα από τα βασικά χαρακτηριστικά του Angular είναι η αρχιτεκτονική βασισμένη σε components, σύμφωνα με την οποία η εφαρμογή δομείται ως σύνολο επαναχρησιμοποιήσιμων, αυτόνομων μονάδων. Κάθε component περιλαμβάνει τον δικό του κώδικα, HTML πρότυπο και στυλ, ενισχύοντας τη σαφήνεια, την επεκτασιμότητα και τη συντηρησιμότητα του frontend [22].

Το σύστημα δρομολόγησης (Angular Router) επιτρέπει τη μετάβαση μεταξύ διαφορετικών views χωρίς ανανέωση της σελίδας. Υποστηρίζει δυναμικά routes με παραμέτρους, δρομολόγηση σε modules μέσω

lazy loading, καθώς και φρουρούς πρόσβασης (*Route Guards*) για τον περιορισμό πρόσβασης σε ευαίσθητες σελίδες. Στην παρούσα εφαρμογή εφαρμόστηκαν guards ώστε μη αυθεντικοποιημένοι χρήστες να μην έχουν πρόσβαση σε προστατευμένα routes [20].

Η επικοινωνία μεταξύ components υλοποιήθηκε μέσω των μηχανισμών *@Input()* και *@Output()* για parent-child σχέσεις, ενώ για ανταλλαγή δεδομένων μεταξύ ανεξάρτητων components χρησιμοποιήθηκαν κοινές υπηρεσίες. Η προσέγγιση αυτή προσφέρει σαφή διαχωρισμό ρόλων και βοηθά στη διατήρηση καθαρής αρχιτεκτονικής.

Για τη διαχείριση φορμών, αξιοποιήθηκε το σύστημα Reactive Forms, το οποίο βασίζεται στη χρήση της *FormBuilder* και των *FormGroup* / *FormControl*. Με αυτό τον τρόπο, υλοποιήθηκαν δυναμικές φόρμες με ενσωματωμένο validation, υψηλό βαθμό ελέγχου και δυνατότητα testability, καθιστώντας τις φόρμες πιο ευέλικτες και σταθερές [21].

Τέλος, η ανάπτυξη υποστηρίχθηκε σε μεγάλο βαθμό από το Angular CLI, ένα εργαλείο γραμμής εντολών που διευκολύνει τη δημιουργία αρχείων, την κατασκευή και τη δοκιμή της εφαρμογής. Το CLI επιτρέπει την αυτόματη δημιουργία modules, components, services και άλλων στοιχείων, ακολουθώντας τις βέλτιστες πρακτικές του framework [23].

Η επιλογή του Angular έγινε με βάση τη σταθερότητά του, τη στενή του σύνδεση με τη TypeScript, τη μεγάλη κοινότητα υποστήριξης και την πληρότητα των εργαλείων που παρέχει, καθιστώντας το κατάλληλο για εφαρμογές με απαιτήσεις σε δομή και επεκτασιμότητα.

3.3.4 RxJS και Αντιδραστικός Προγραμματισμός

Το RxJS (Reactive Extensions for JavaScript) είναι μια βιβλιοθήκη που υποστηρίζει τον αντιδραστικό προγραμματισμό μέσω της χρήσης Observables, επιτρέποντας τη σύνθεση ασύγχρονων και βασισμένων σε γεγονότα προγραμμάτων με έναν δηλωτικό τρόπο . Στο πλαίσιο της εφαρμογής μας, το RxJS χρησιμοποιήθηκε εκτενώς για τη διαχείριση ασύγχρονων ροών δεδομένων, όπως αιτήματα HTTP, αλληλεπιδράσεις χρηστών και επικοινωνία μεταξύ components.

Βασική έννοια του RxJS είναι το Observable, το οποίο αντιπροσωπεύει μια ροή δεδομένων που μπορεί να εκπέμπει πολλαπλές τιμές με την πάροδο του χρόνου. Οι Observers εγγράφονται σε αυτά τα Observables για να λαμβάνουν ειδοποιήσεις όταν υπάρχουν νέες τιμές, σφάλματα ή όταν ολοκληρώνεται η ροή. Η σύνδεση μεταξύ Observable και Observer επιτυγχάνεται μέσω του Subscription, το οποίο μπορεί να ακυρωθεί για να σταματήσει η παρακολούθηση της ροής .RxJS

Το RxJS παρέχει ένα πλούσιο σύνολο Operators, όπως map, filter, merge, switchMap, debounceTime, που επιτρέπουν τον μετασχηματισμό, το φιλτράρισμα και τον συνδυασμό ροών δεδομένων με έναν καθαρό και δηλωτικό τρόπο. Αυτοί οι operators είναι κρίσιμοι για την επεξεργασία σύνθετων ασύγχρονων ροών δεδομένων και την αποφυγή της λεγόμενης "callback hell" .RxJS

Στην εφαρμογή μας, το RxJS χρησιμοποιήθηκε για:

Διαχείριση αιτημάτων HTTP με χρήση του HttpClient, όπου τα αποτελέσματα επεξεργάζονται μέσω operators όπως map και catchError.

Αντιμετώπιση αλληλεπιδράσεων χρηστών, όπως clicks και πληκτρολογήσεις, με χρήση του fromEvent και εφαρμογή operators όπως debounceTime για την αποφυγή περιττών αιτημάτων.

Επικοινωνία μεταξύ components μέσω Subjects, όπως το BehaviorSubject, που επιτρέπει την αποστολή και λήψη δεδομένων σε πραγματικό χρόνο μεταξύ διαφορετικών τμημάτων της εφαρμογής.

Η χρήση του RxJS ενίσχυσε την επεκτασιμότητα και τη συντηρησιμότητα της εφαρμογής, επιτρέποντας μια πιο καθαρή και οργανωμένη διαχείριση των ασύγχρονων ροών δεδομένων.

3.4 Βάσεις Δεδομένων

Η αποθήκευση, η αναζήτηση και η συσχέτιση των δεδομένων της εφαρμογής πραγματοποιούνται μέσω της χρήσης σχεσιακής βάσης δεδομένων. Για την υλοποίηση του συστήματος επιλέχθηκε η PostgreSQL, ένα ισχυρό και ευέλικτο σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων ανοιχτού κώδικα. Η επιλογή βασίστηκε στην αξιοπιστία της, την πληρότητα δυνατοτήτων της και τη συμβατότητά της με το οικοσύστημα του Spring Boot.

Η βάση δεδομένων αποτελεί το θεμέλιο της εφαρμογής, καθώς σε αυτήν αποθηκεύονται χρήστες, μαθήματα, σχέσεις εγγραφών, καθώς και στοιχεία σύνδεσης και ρόλων. Η διασύνδεση του backend με τη βάση υλοποιείται μέσω του Spring Data JPA, ενώ η δομή της βάσης οργανώθηκε με γνώμονα τη βελτιστοποίηση της απόδοσης και της συντηρησιμότητας.

3.4.1 PostgreSQL

Η PostgreSQL είναι ένα εξελιγμένο σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων (RDBMS) ανοικτού κώδικα, το οποίο αναπτύσσεται ενεργά από τη δεκαετία του 1990. Υποστηρίζει το πρότυπο SQL, ενώ διαθέτει πλήρη συμμόρφωση με τις αρχές του ACID και χρησιμοποιεί το μοντέλο Multi-Version Concurrency Control (MVCC) για αποτελεσματική διαχείριση ταυτόχρονων αιτημάτων και αποφυγή αποκλεισμών (deadlocks) [29].

Η PostgreSQL προσφέρει μια πληθώρα προηγμένων χαρακτηριστικών που την καθιστούν κατάλληλη για εφαρμογές με αυξημένες απαιτήσεις. Υποστηρίζει σύνθετους τύπους δεδομένων, όπως `ARRAY`, `JSON`, `UUID`, λειτουργίες για JSONB indexing, καθώς και stored procedures, triggers και προηγμένα indexes (π.χ. GIN, GiST). Στην εφαρμογή χρησιμοποιήθηκαν UUIDs ως primary keys για μεγαλύτερη ασφάλεια και ανεξαρτησία από την εσωτερική αύξηση ταυτοτήτων, καθώς και μοναδικά indexes για την αποτροπή διπλότυπων εγγραφών σε πίνακες όπως `user` και `enrollment` [30].

Η επιλογή της PostgreSQL βασίστηκε στην αξιοπιστία της, την ισχυρή κοινότητα υποστήριξης και την άριστη ενσωμάτωσή της με τεχνολογίες του οικοσυστήματος Spring Boot, όπως το Spring Data JPA. Η διασύνδεση με την εφαρμογή έγινε μέσω του PostgreSQL JDBC driver, ενώ οι αλλαγές στη δομή της βάσης (π.χ. δημιουργία πινάκων, προσθήκη πεδίων) διαχειρίστηκαν με το εργαλείο Flyway, το οποίο επιτρέπει ελεγχόμενες μεταναστεύσεις και διασφαλίζει συνέπεια μεταξύ περιβάλλοντος ανάπτυξης και παραγωγής [31].

Η PostgreSQL συνέβαλε καθοριστικά στην εύρυθμη λειτουργία του backend, καθώς προσέφερε σταθερότητα, επεκτασιμότητα και άμεση υποστήριξη των απαιτήσεων της εφαρμογής σε ό,τι αφορά την αποθήκευση, την αναζήτηση και τη συσχέτιση δεδομένων.

Κεφάλαιο 4ο: Ανάπτυξη της Εφαρμογής

4.1 Εισαγωγή

Αφού αναλύθηκαν το θεωρητικό υπόβαθρο, οι υφιστάμενες λύσεις και οι τεχνολογίες που αξιοποιήθηκαν, στο παρόν κεφάλαιο παρουσιάζεται η πρακτική υλοποίηση της εφαρμογής. Ειδικότερα, γίνεται πλήρης περιγραφή του τρόπου με τον οποίο ενσωματώθηκαν οι λειτουργικές απαιτήσεις που προέκυψαν από την ανάλυση χρηστών (user stories), καθώς και των βασικών διεπαφών και σεναρίων χρήσης που διαμορφώνουν την εμπειρία του τελικού χρήστη.

Η εφαρμογή έχει σχεδιαστεί ώστε να είναι απλή, λειτουργική και αποτελεσματική. Δεν στοχεύει να υποκαταστήσει την εκπαιδευτική διαδικασία, αλλά να τη συμπληρώσει, λειτουργώντας ως μέσο διασύνδεσης φοιτητών με σκοπό την ανταλλαγή γνώσης. Η πλατφόρμα επιτρέπει τη δημιουργία και δημοσίευση μαθημάτων, την εγγραφή/απεγγραφή συμμετεχόντων, καθώς και την αποστολή ειδοποιήσεων για σημαντικές ενέργειες. Όλες οι λειτουργίες υλοποιήθηκαν με σεβασμό στην ευχρηστία, την ασφάλεια και την επεκτασιμότητα του συστήματος.

Στις επόμενες ενότητες, παρουσιάζονται οι βασικές λειτουργίες του συστήματος μέσα από User Stories, τεκμηριώνονται οι απαιτήσεις, περιγράφονται τα σενάρια χρήσης και αναλύονται βασικές πτυχές της υλοποίησης, τόσο από την πλευρά του χρήστη όσο και από την πλευρά του προγραμματιστή. Ο στόχος είναι η συνολική αποτύπωση του τρόπου με τον οποίο το σύστημα καλύπτει τις ανάγκες για τις οποίες σχεδιάστηκε.

4.2 User Stories και Λειτουργικές Απαιτήσεις

Η ορθή καταγραφή και ανάλυση των αναγκών του τελικού χρήστη αποτελεί κρίσιμο στάδιο στον σχεδιασμό κάθε πληροφοριακού συστήματος. Στην περίπτωση της παρούσας εφαρμογής, οι χρήστες είναι φοιτητές που επιθυμούν είτε να παρέχουν είτε να λάβουν εκπαιδευτική υποστήριξη σε πανεπιστημιακά μαθήματα. Δεδομένου ότι πρόκειται για έναν ενιαίο ρόλο χρήστη με δύο βασικές ιδιότητες (δημιουργός ή συμμετέχων σε μάθημα), η προσέγγιση των **User Stories** επέτρεψε την καταγραφή των απαιτήσεων σε φυσική γλώσσα, εστιάζοντας στις πραγματικές ανάγκες και ενέργειες που επιθυμούν να εκτελέσουν οι φοιτητές μέσα στην εφαρμογή.

Η χρήση User Stories είναι ευρέως διαδεδομένη στη μεθοδολογία Agile και επιτρέπει τη γρήγορη αποτύπωση λειτουργικών αναγκών, πριν ακόμα ξεκινήσει η φάση της υλοποίησης. Στην παρούσα εργασία, τα σενάρια που ακολουθούν αποτέλεσαν τον οδηγό για τη δημιουργία των διεπαφών χρήστη, των backend endpoints, αλλά και της συνολικής ροής του συστήματος.

4.2.1 User Stories

Κάθε User Story διατυπώνεται με τη μορφή: «Ως [τύπος χρήστη], θέλω να [ενέργεια] ώστε να [σκοπός].»
Η προσέγγιση αυτή επιτρέπει στον αναλυτή να κατανοήσει όχι μόνο τι ζητά ο χρήστης, αλλά και γιατί το ζητά.

Α. Γενικές ενέργειες χρήστη

- Ως χρήστης, θέλω να μπορώ να δημιουργήσω λογαριασμό ώστε να αποκτήσω πρόσβαση στην πλατφόρμα.
- Ως χρήστης, θέλω να μπορώ να συνδεθώ και να αποσυνδεθώ από την εφαρμογή με ασφάλεια.
- Ως χρήστης, θέλω να επεξεργάζομαι τα προσωπικά μου στοιχεία (όνομα, email) ώστε να διατηρώ επικαιροποιημένο προφίλ.
- Ως χρήστης, θέλω να μπορώ να επαναφέρω τον κωδικό μου σε περίπτωση που τον ξεχάσω, χωρίς να χρειάζεται εξωτερική υποστήριξη.
- Ως χρήστης, θέλω να μπορώ να διαγράψω οριστικά τον λογαριασμό μου, διασφαλίζοντας τη διαγραφή των δεδομένων μου.
- Ως χρήστης, θέλω να λαμβάνω email ειδοποιήσεις όταν εγγράφομαι ή διαγράφομαι από την εφαρμογή, ώστε να ενημερώνομαι για σημαντικές αλλαγές.

Β. Διαχείριση μαθημάτων (ως δημιουργός)

- Ως χρήστης, θέλω να μπορώ να δημιουργώ νέα μαθήματα εισάγοντας βασικά στοιχεία (τίτλο, περιγραφή, μέγιστο πλήθος συμμετεχόντων), ώστε να προσφέρω ενισχυτική βοήθεια.
- Ως χρήστης, θέλω να μπορώ να επεξεργάζομαι τις πληροφορίες ενός μαθήματος που έχω δημιουργήσει.
- Ως χρήστης, θέλω να μπορώ να διαγράψω μαθήματα που δεν ισχύουν πλέον ή δεν παρουσιάζουν ενδιαφέρον.
- Ως χρήστης, θέλω να μπορώ να δω ποια μαθήματα έχω δημιουργήσει, με πληροφορίες συμμετεχόντων.
- Ως χρήστης, θέλω να μπορώ να αφαιρέσω κάποιον συμμετέχοντα από το μάθημά μου εφόσον παραβιάζει κανόνες ή δεν ανταποκρίνεται.
- Ως χρήστης, θέλω να λαμβάνω email ειδοποιήσεις όταν ένας φοιτητής εγγράφεται ή αποχωρεί από μάθημά μου, ή όταν διαγράψω ή τροποποιήσω το μάθημα.

C. Συμμετοχή σε μαθήματα

- Ως χρήστης, θέλω να μπορώ να περιηγούμαι στα διαθέσιμα μαθήματα.
- Ως χρήστης, θέλω να φιλτράρω μαθήματα με βάση τη θεματική ή λέξεις-κλειδιά, ώστε να βρίσκω εύκολα ό,τι με ενδιαφέρει.
- Ως χρήστης, θέλω να μπορώ να εγγραφώ σε ένα μάθημα, εφόσον υπάρχουν διαθέσιμες θέσεις.
- Ως χρήστης, θέλω να μπορώ να απεγγραφώ από μάθημα χωρίς επιπτώσεις.
- Ως χρήστης, θέλω να ενημερώνομαι μέσω email αν ακυρωθεί ή τροποποιηθεί κάποιο μάθημα στο οποίο είμαι εγγεγραμμένος.

Η παραπάνω καταγραφή συνέβαλε στον καθορισμό της ροής των οθονών, της οργάνωσης της βάσης δεδομένων και της διάκρισης των API endpoints μεταξύ «δημιουργών» και «συμμετεχόντων», χωρίς να απαιτείται η εισαγωγή ξεχωριστών ρόλων.

4.2.2 Λειτουργικές Απαιτήσεις

Οι λειτουργικές απαιτήσεις της εφαρμογής προέκυψαν άμεσα από τα παραπάνω User Stories και καταγράφονται ως εξής:

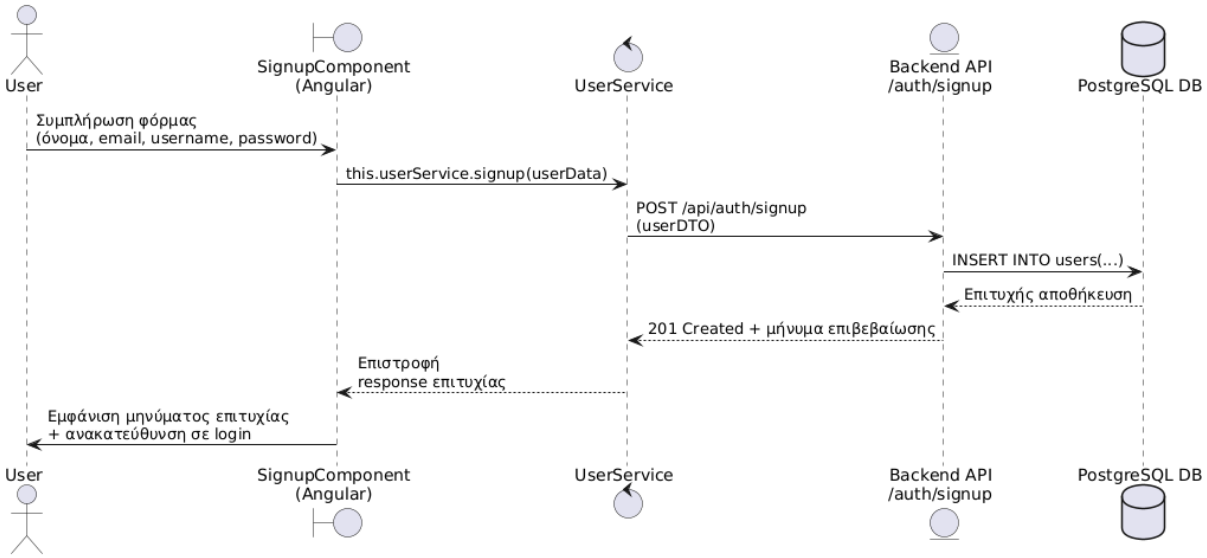
- Εγγραφή, είσοδος και έξοδος χρήστη με έλεγχο ταυτότητας μέσω email και κωδικού.
- Δυνατότητα τροποποίησης προφίλ, με απλό μηχανισμό αλλαγής στοιχείων.
- Επαναφορά κωδικού πρόσβασης μέσω email με ασφαλές token.
- Διαγραφή λογαριασμού, με πλήρη απομάκρυνση σχετιζόμενων δεδομένων.
- Δημιουργία, επεξεργασία και διαγραφή μαθημάτων από οποιονδήποτε εγγεγραμμένο χρήστη.
- Περιήγηση και αναζήτηση μαθημάτων ανά θεματική ενότητα ή λέξη-κλειδί.
- Δήλωση και ακύρωση συμμετοχής από μαθήματα.
- Προβολή συμμετοχών σε μάθημα από τον δημιουργό του.
- Αφαίρεση φοιτητών από μάθημα, με σχετική ειδοποίηση.
- Αποστολή ειδοποιήσεων μέσω email για σημαντικά γεγονότα (εγγραφή, διαγραφή, τροποποίηση μαθημάτων κ.λπ.).
- Έλεγχος διαθεσιμότητας θέσεων, ώστε να μην ξεπερνιέται το προκαθορισμένο όριο συμμετοχών.

Οι παραπάνω λειτουργίες αποτελούν τον πυρήνα της εφαρμογής και σχεδιάστηκαν με τρόπο που να εξασφαλίζεται τόσο η ευχρηστία όσο και η ασφάλεια του συστήματος.

4.3 Περιγραφή Κύριων Λειτουργιών και Οθονών

Η παρούσα ενότητα περιγράφει τις βασικές λειτουργικές οθόνες της εφαρμογής, μέσα από τις οποίες ο χρήστης αλληλεπιδρά με το σύστημα. Κάθε οθόνη καλύπτει μία ή περισσότερες λειτουργίες που ανταποκρίνονται στα user stories που αναλύθηκαν στην προηγούμενη ενότητα. Η προσέγγιση είναι προσανατολισμένη στη χρηστικότητα, τη σαφή παρουσίαση της πληροφορίας και την ελαχιστοποίηση των βημάτων που απαιτούνται για την ολοκλήρωση μιας ενέργειας.

4.3.1 Δημιουργία Λογαριασμού Χρήστη



Εικόνα 4.1 Διάγραμμα ροής για την δημιουργία λογαριασμού χρήστη

Η σελίδα εγγραφής αποτελεί την αφετηρία για την αλληλεπίδραση του φοιτητή με την πλατφόρμα. Η φόρμα που εμφανίζεται περιλαμβάνει έξι βασικά πεδία εισόδου:

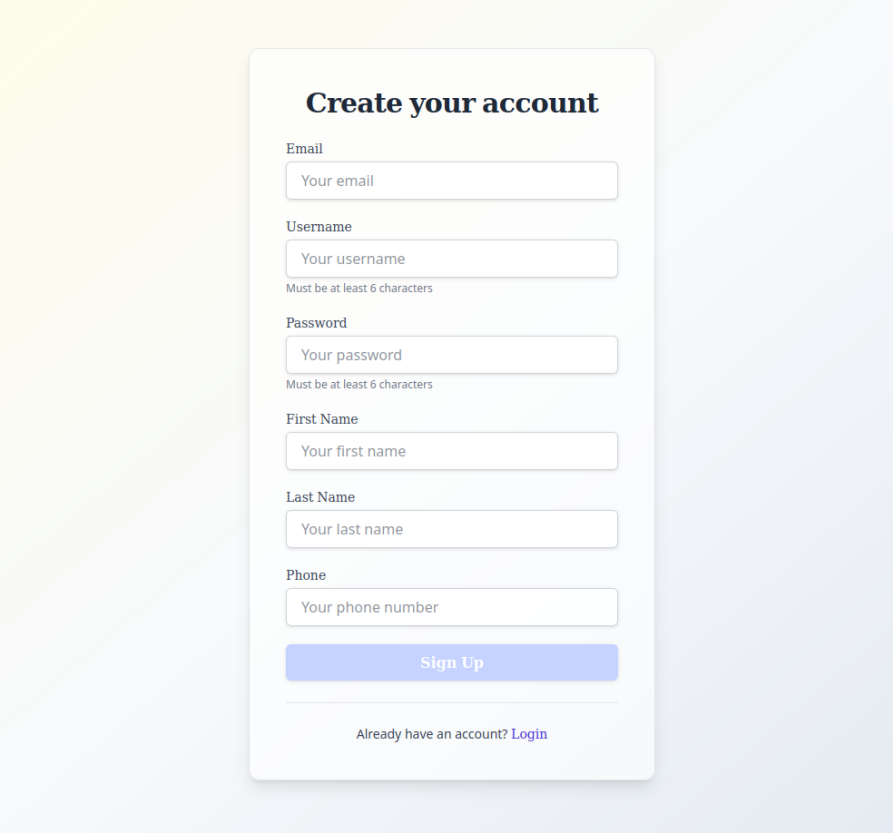
- Email: Χρησιμοποιείται για την ταυτοποίηση του χρήστη και την αποστολή ειδοποιήσεων.
- Username: Το δημόσια ορατό όνομα του χρήστη εντός της πλατφόρμας.
- Password: Κωδικός ασφαλείας (ελάχιστο μήκος 6 χαρακτήρες), αποθηκευμένος με κρυπτογράφηση.

- First Name / Last Name: Προσωπικά στοιχεία του χρήστη, απαραίτητα για την επικοινωνία και την εμφάνιση σε λίστες συμμετεχόντων.
- Phone: Χρησιμοποιείται εσωτερικά ή για μελλοντική επέκταση (δεν αποστέλλονται SMS).

Η φόρμα είναι σχεδιασμένη με καθαρό και μοντέρνο τρόπο, με άμεση οπτική ανατροφοδότηση για την εγκυρότητα των πεδίων. Αφού συμπληρωθούν όλα τα απαραίτητα στοιχεία και πατηθεί το κουμπί "Sign Up", ο λογαριασμός δημιουργείται.

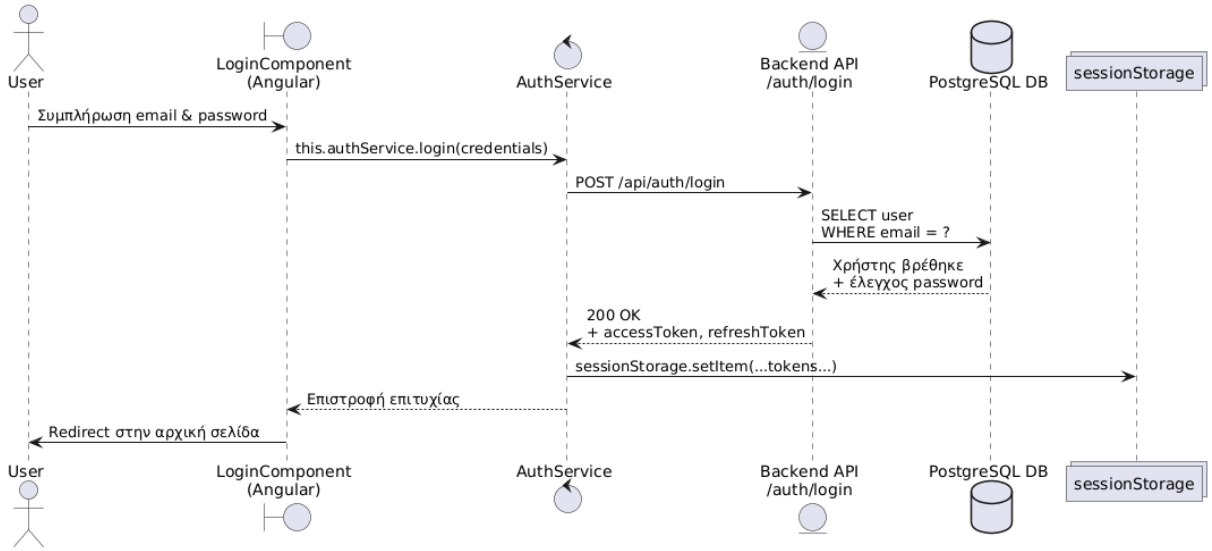
Με την επιτυχή εγγραφή, η εφαρμογή αποστέλλει αυτόματα email καλωσορίσματος στον νέο χρήστη, το οποίο επιβεβαιώνει την εγγραφή του και περιέχει πληροφορίες για το πώς μπορεί να ξεκινήσει να χρησιμοποιεί την πλατφόρμα. Το email αυτό είναι το πρώτο βήμα στην εξατομικευμένη επικοινωνία με τον χρήστη και συμβάλλει στην ενίσχυση της αξιοπιστίας του συστήματος.

Η διαδικασία εγγραφής είναι πλήρως εναρμονισμένη με τις αρχές της φιλικότητας προς τον χρήστη, της ασφάλειας και της διαφάνειας.



Εικόνα 4.2: Φόρμα δημιουργίας λογαριασμού χρήστη (Signup)

4.3.2 Σύνδεση Χρήστη



Εικόνα 4.3 Διάγραμμα ροής για την σύνδεση χρήστη

Η σελίδα σύνδεσης αποτελεί την κύρια πύλη πρόσβασης στην πλατφόρμα για κάθε χρήστη που έχει ήδη εγγραφεί. Ο σχεδιασμός της είναι καθαρός και λειτουργικός, επιτρέποντας στον φοιτητή να εισέλθει γρήγορα στο σύστημα.

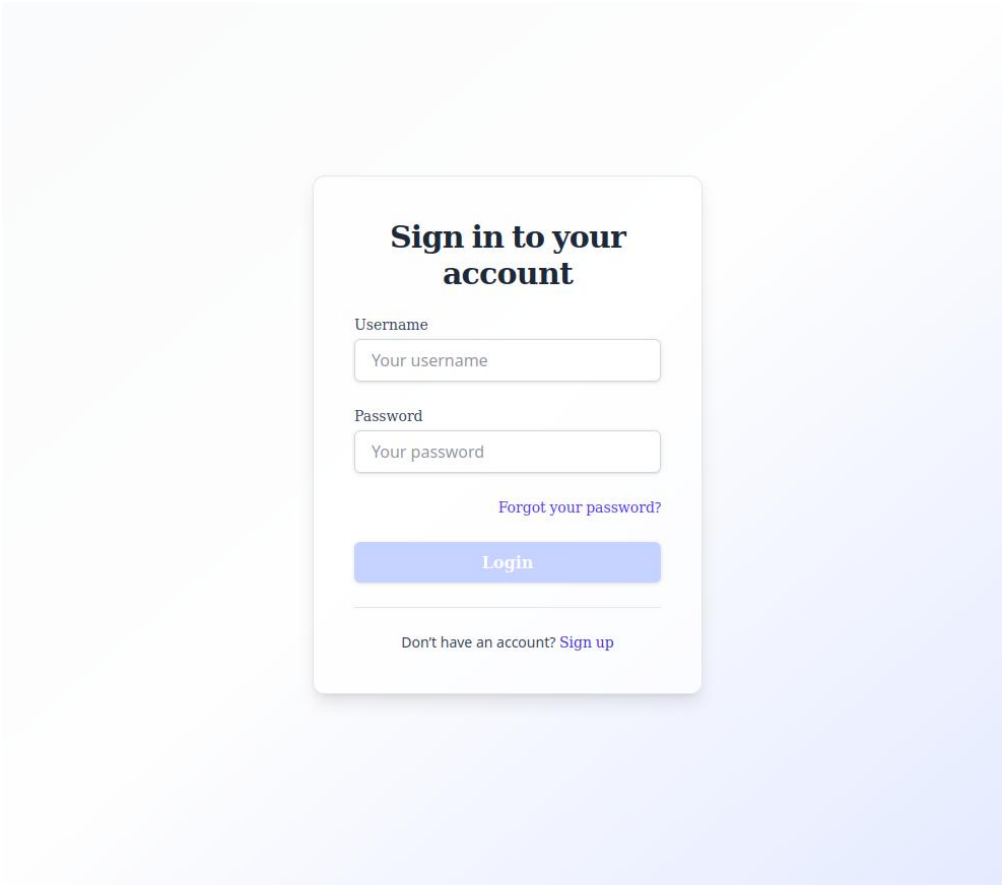
Η φόρμα περιλαμβάνει δύο βασικά πεδία:

- Username: Το μοναδικό όνομα χρήστη που ορίστηκε κατά την εγγραφή. Το σύστημα διασφαλίζει ότι δεν υπάρχουν διπλότυπα usernames, ενώ η σύνδεση γίνεται αποκλειστικά με αυτό.
- Password: Ο προσωπικός κωδικός πρόσβασης του χρήστη. Το πεδίο αποκρύπτει τους χαρακτήρες για λόγους ασφαλείας και συνοδεύεται από βασικό έλεγχο εγκυρότητας.

Κάτω από τη φόρμα υπάρχουν δύο χρήσιμοι σύνδεσμοι:

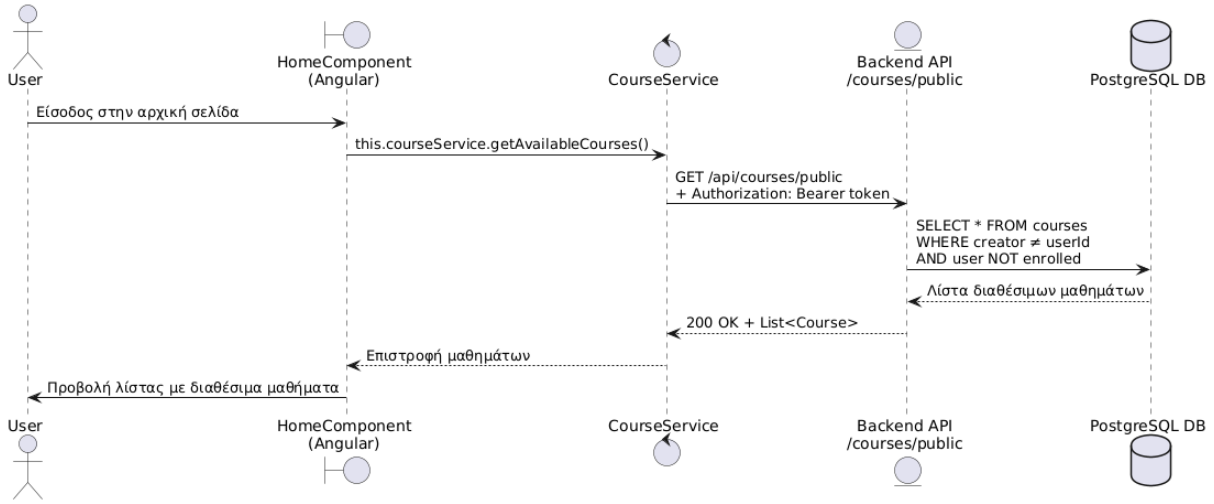
- **“**Forgot your password?”: Επιτρέπει στον χρήστη να ξεκινήσει διαδικασία επαναφοράς του κωδικού σε περίπτωση που τον έχει ξεχάσει. Η διαδικασία αυτή θα παρουσιαστεί σε επόμενη ενότητα.
- “Sign up”: Παρέχει άμεση μετάβαση στη φόρμα εγγραφής για νέους χρήστες που δεν διαθέτουν ακόμα λογαριασμό.

Κατά την επιτυχή είσοδο, ο χρήστης μεταφέρεται απευθείας στην αρχική σελίδα της εφαρμογής. Η τεχνική λειτουργία της σύνδεσης, καθώς και η ροή ταυτοποίησης, περιγράφονται αναλυτικά στην ενότητα 4.4 – Τεχνική Υλοποίηση.



Εικόνα 4.4: Φόρμα σύνδεσης χρήστη (Login)

4.3.3 Αρχική Σελίδα – Λίστα Διαθέσιμων Μαθημάτων



Εικόνα 4.5 Διάγραμμα ροής λίστας διαθέσιμων μαθημάτων

Μετά τη σύνδεση, ο χρήστης μεταφέρεται στην αρχική σελίδα της εφαρμογής, η οποία αποτελεί τον κεντρικό κόμβο πλοήγησης και ενημέρωσης. Εδώ προβάλλονται **όλα τα διαθέσιμα μαθήματα** που έχουν δημιουργηθεί από άλλους φοιτητές, μαζί με βασικές πληροφορίες που επιτρέπουν την ταχεία αξιολόγηση και επιλογή.

Η διάταξη της οθόνης είναι χωρισμένη σε δύο βασικά τμήματα:

- Στα αριστερά βρίσκεται το **πλευρικό μενού πλοήγησης**, το οποίο περιλαμβάνει:
- **Home**: Επιστροφή στην αρχική λίστα μαθημάτων.
- **My Profile**: Πρόσβαση στο προσωπικό προφίλ του χρήστη.
- **Add Course**: Δημιουργία νέου μαθήματος.

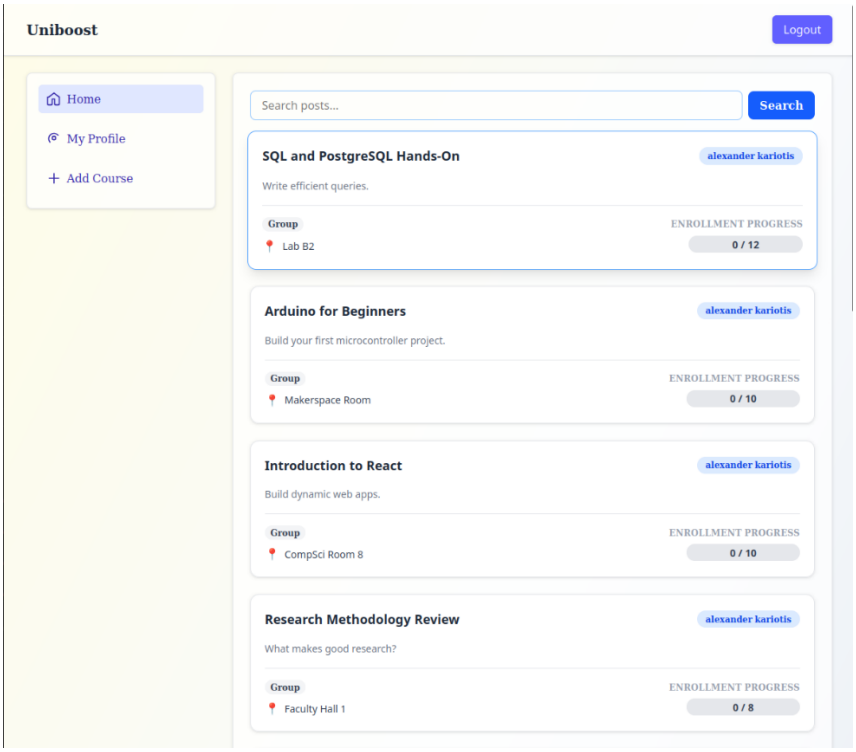
Στο κύριο περιεχόμενο (δεξιά), προβάλλονται σε μορφή καρτών τα διαθέσιμα μαθήματα. Κάθε κάρτα περιλαμβάνει:

- **Τίτλο** του μαθήματος
- Σύντομη περιγραφή (preview)
- **Τύπο** (π.χ. Group)
- **Τοποθεσία** του μαθήματος
- **Δημιουργό** (username του διδάσκοντα)
- **Ενδείκτη συμμετοχών**, με εμφανή το μέγιστο και τις ήδη εγγεγραμμένες συμμετοχές

Στην κορυφή της σελίδας υπάρχει **πεδίο αναζήτησης με κουμπί Search**, που επιτρέπει στον χρήστη να φιλτράρει τα μαθήματα πληκτρολογώντας λέξεις-κλειδιά. Η ανανέωση των αποτελεσμάτων είναι άμεση και η λειτουργικότητα απλή και κατανοητή.

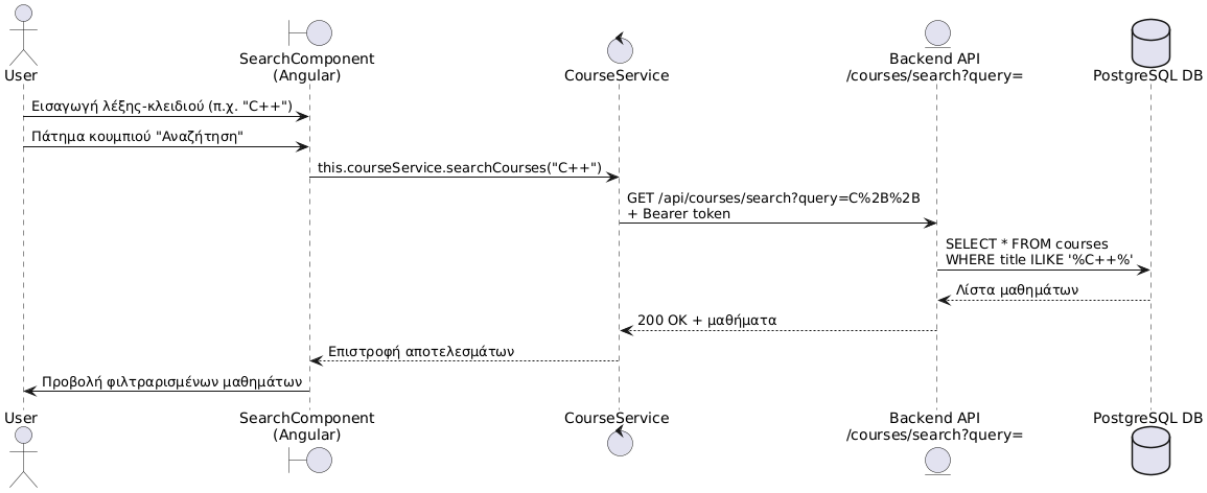
Ο σχεδιασμός της αρχικής σελίδας εξυπηρετεί την **εύκολη περιήγηση και επιλογή μαθήματος**. Η προεπισκόπηση των μαθημάτων παρέχει όλα τα απαραίτητα στοιχεία χωρίς την ανάγκη πλοήγησης σε ξεχωριστή σελίδα. Σε περίπτωση που ο χρήστης ενδιαφέρεται για ένα μάθημα, μπορεί να πατήσει στην κάρτα για να δει περισσότερες λεπτομέρειες ή να εγγραφεί άμεσα.

Η διεπαφή είναι πλήρως responsive, σχεδιασμένη να λειτουργεί εξίσου άνετα σε υπολογιστή και φορητές συσκευές.



Εικόνα 4.6: Αρχική σελίδα εφαρμογής με προβολή διαθέσιμων μαθημάτων

4.3.4 Αναζήτηση Μαθημάτων



Εικόνα 4.7 Διάγραμμα ροής για την αναζήτηση μαθημάτων

Η εφαρμογή παρέχει στον χρήστη τη δυνατότητα αναζήτησης μαθημάτων με λέξεις-κλειδιά, μέσα από ένα εμφανές και προσβάσιμο πεδίο αναζήτησης τοποθετημένο στο επάνω μέρος της αρχικής σελίδας.

Καθώς ο χρήστης πληκτρολογεί λέξεις-κλειδιά (π.χ. "ja"), το σύστημα εκτελεί δυναμική αναζήτηση και εμφανίζει μόνο τα μαθήματα που αντιστοιχούν στην καταχωρημένη φράση. Η αναζήτηση πραγματοποιείται σε πεδία όπως:

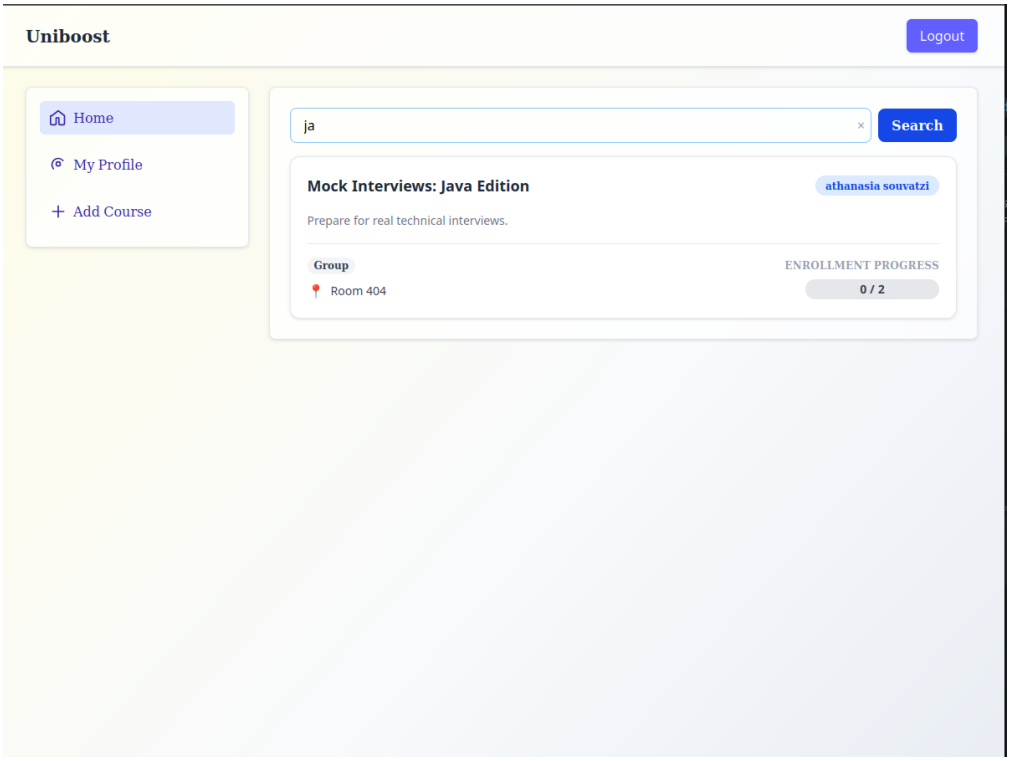
- τίτλος μαθήματος,
- σύντομη περιγραφή (preview),
- ή πιθανώς και τοποθεσία ή όνομα δημιουργού (ανάλογα με την τεχνική υλοποίηση που περιγράφεται στο 4.4).

Στο παράδειγμα της εικόνας, ο χρήστης πληκτρολογεί “ja” και το σύστημα εμφανίζει μόνο το μάθημα “Mock Interviews: Java Edition”, με όλες τις σχετικές πληροφορίες (τίτλος, τύπος, αίθουσα, δημιουργός και διαθεσιμότητα θέσεων).

Η λειτουργία αναζήτησης:

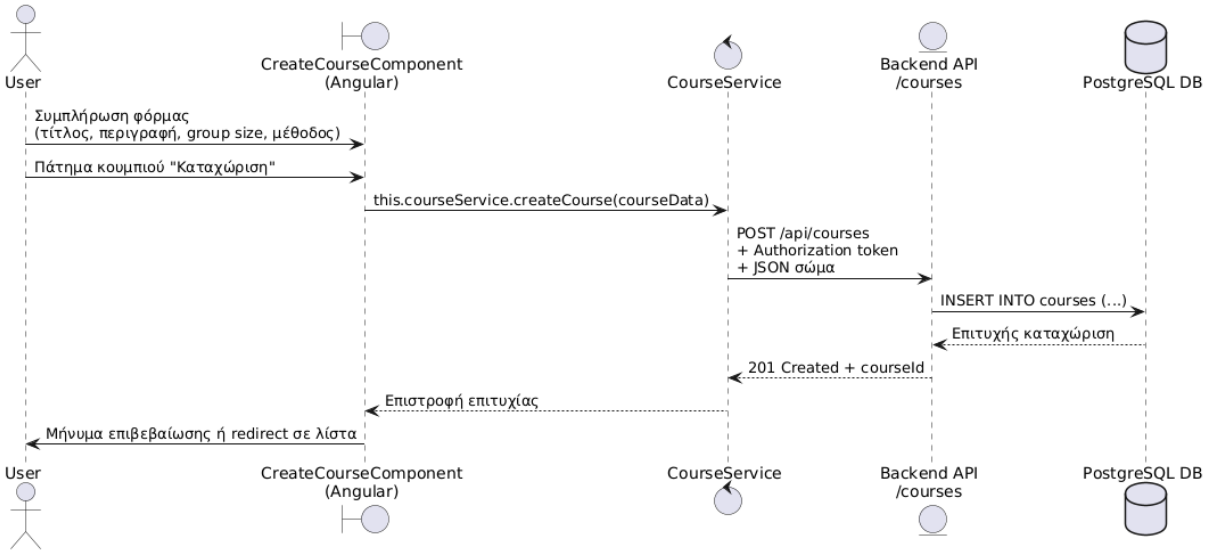
- ενισχύει σημαντικά τη χρηστικότητα της εφαρμογής,
- μειώνει τον χρόνο εντοπισμού του κατάλληλου μαθήματος,
- και προσφέρει βελτιωμένη εμπειρία πλοήγησης, ιδιαίτερα σε περιπτώσεις όπου τα διαθέσιμα μαθήματα είναι πολλά.

Το πεδίο αναζήτησης υποστηρίζει άμεση ανανέωση των αποτελεσμάτων με το πάτημα του πλήκτρου “Search” ή κατά την πληκτρολόγηση, καθιστώντας τη λειτουργία φιλική και ταυτόχρονα αποδοτική.



Εικόνα 4.8: Αναζήτηση μαθημάτων με λέξεις-κλειδιά

4.3.5 Δημιουργία Νέου Μαθήματος



Εικόνα 4.9 Διάγραμμα ροής για την δημιουργία νέου μαθήματος

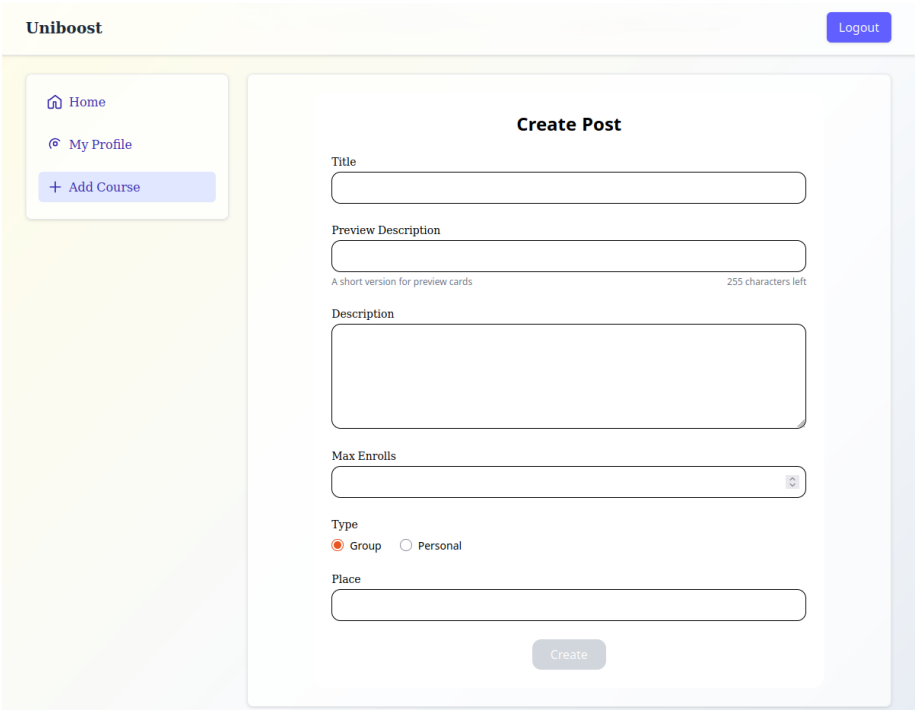
Η δημιουργία μαθήματος αποτελεί μία από τις βασικότερες λειτουργίες της εφαρμογής, καθώς επιτρέπει σε κάθε φοιτητή να λειτουργήσει ως διδάσκων και να προτείνει μάθημα στους συμφοιτητές του. Η οθόνη δημιουργίας προσφέρεται μέσω του κουμπιού **"Add Course"** στο πλευρικό μενού πλοήγησης και οδηγεί τον χρήστη σε ειδικά διαμορφωμένη φόρμα.

Η φόρμα περιλαμβάνει τα παρακάτω πεδία:

- Title: Ο τίτλος του μαθήματος, ο οποίος αποτελεί και το κύριο στοιχείο αναγνώρισης στη λίστα μαθημάτων. Είναι υποχρεωτικό πεδίο.
- Preview Description: Μία σύντομη περιγραφή (μέχρι 255 χαρακτήρες) που προβάλλεται συνοπτικά στις κάρτες της αρχικής σελίδας.
- Description: Πλήρης περιγραφή του περιεχομένου, των στόχων ή της δομής του μαθήματος. Επιτρέπει στον δημιουργό να εξηγήσει με περισσότερες λεπτομέρειες τη φύση του μαθήματος.
- Max Enrolls: Ο μέγιστος αριθμός φοιτητών που μπορούν να εγγραφούν στο μάθημα. Χρησιμοποιείται για τον έλεγχο διαθεσιμότητας.
- Type: Επιλογή μεταξύ Group ή Personal, προσδιορίζοντας αν πρόκειται για ομαδικό μάθημα ή ατομική συνεδρία.
- Place: Τοποθεσία ή αίθουσα διεξαγωγής του μαθήματος (π.χ. "Lab B2", "Room 404").

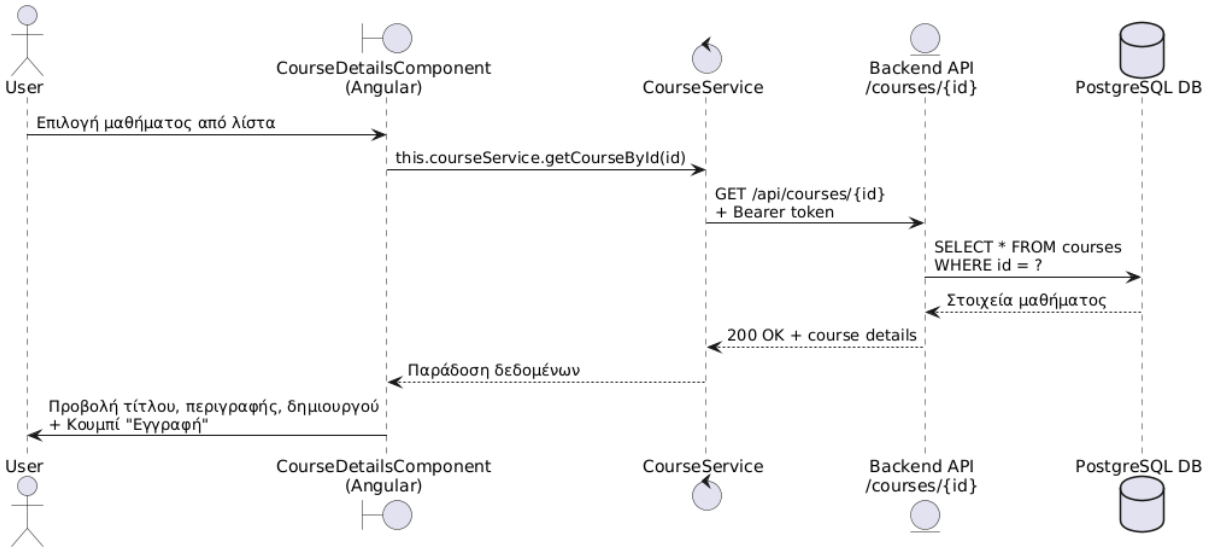
Το κουμπί “Create” είναι απενεργοποιημένο (grayed out) μέχρι να συμπληρωθούν σωστά τα απαιτούμενα πεδία, εξασφαλίζοντας έτσι τη βασική εγκυρότητα των δεδομένων.

Η διεπαφή είναι λιτή, ξεκάθαρη και δίνει στον φοιτητή-δημιουργό τον απόλυτο έλεγχο στην καταχώριση του περιεχομένου. Όταν ολοκληρωθεί επιτυχώς η διαδικασία, το μάθημα προστίθεται στη βάση δεδομένων και εμφανίζεται άμεσα στη λίστα των διαθέσιμων μαθημάτων για όλους τους χρήστες. Επιπλέον, ο δημιουργός του μαθήματος λαμβάνει σχετική επιβεβαίωση μέσω email για την επιτυχή καταχώριση, όπως θα παρουσιαστεί αναλυτικά σε επόμενη ενότητα.



Εικόνα 4.10: Φόρμα δημιουργίας νέου μαθήματος

4.3.6 Προβολή Λεπτομερειών Μαθήματος (Μη Εγγεγραμμένος Χρήστης)



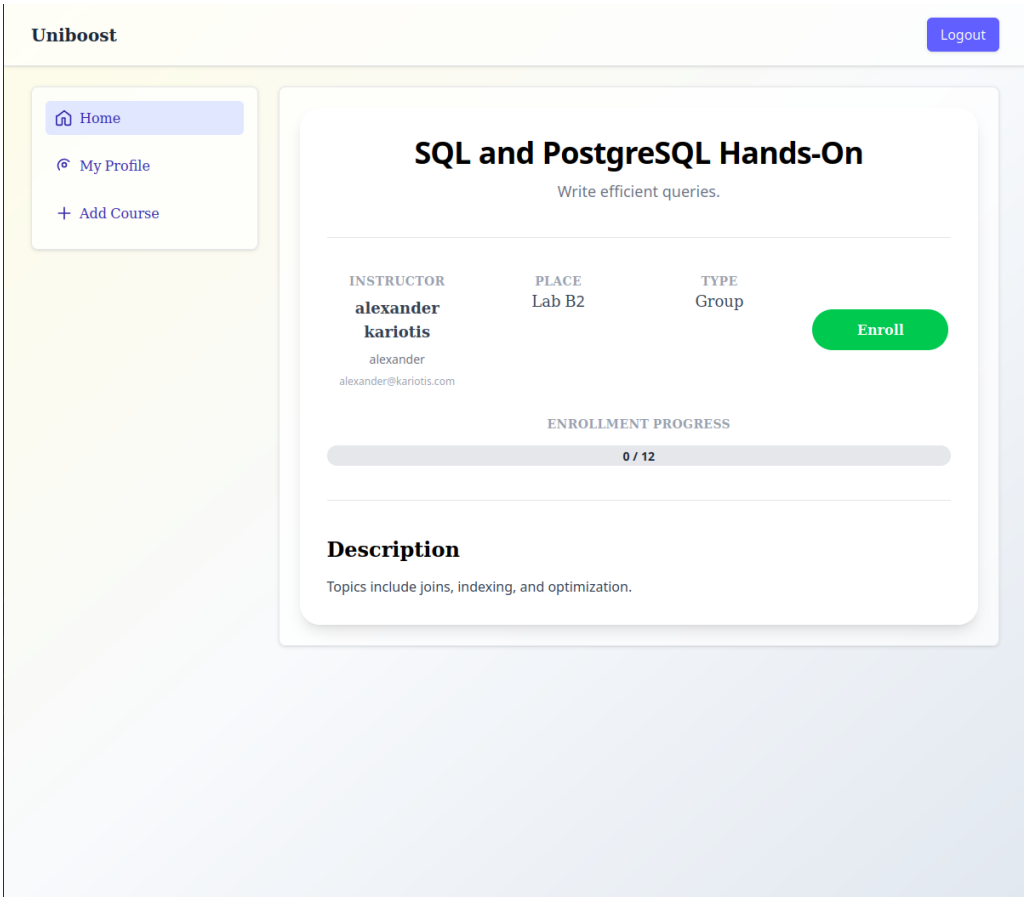
Εικόνα 4.11 Διάγραμμα ροής για την προβολή λεπτομερειών μαθήματος

Όταν ο χρήστης επιλέγει ένα μάθημα από την αρχική σελίδα και **δεν έχει εγγραφεί** σε αυτό, εμφανίζεται μια πλήρης προβολή των λεπτομερειών του μαθήματος. Ο σκοπός αυτής της οθόνης είναι να προσφέρει όλες τις απαραίτητες πληροφορίες, ώστε ο φοιτητής να μπορεί να αποφασίσει αν θέλει να εγγραφεί.

Η σελίδα περιλαμβάνει:

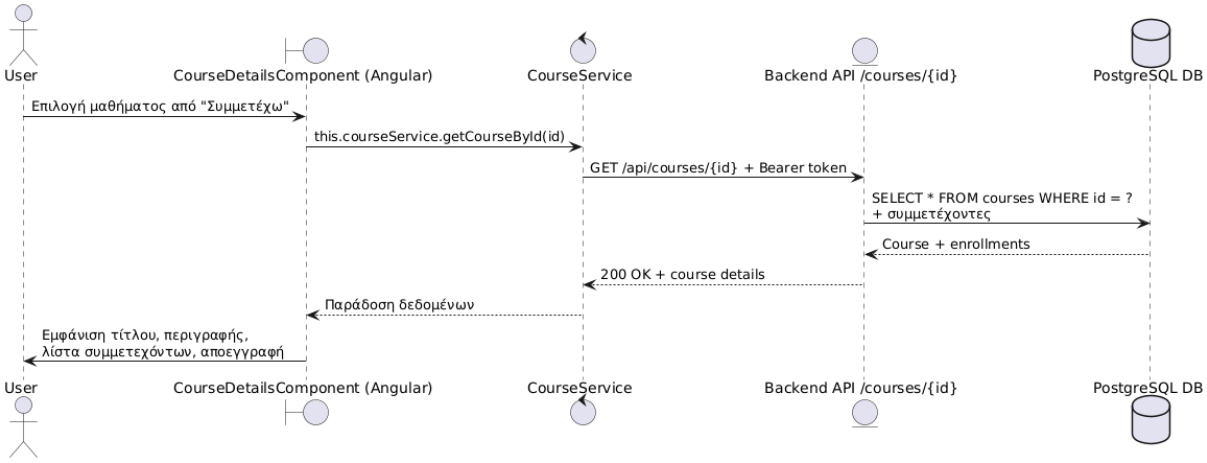
- Τον τίτλο και την εισαγωγική περιγραφή του μαθήματος
- Πληροφορίες για:
    - Instructor (όνομα, username, email δημιουργού)
    - Place διεξαγωγής
    - Type μαθήματος (Group/Personal)
- Ένα έντονα χρωματισμένο κουμπί "Enroll", διαθέσιμο μόνο σε μη εγγεγραμμένους χρήστες
- Μπάρα προόδου εγγραφών, που δείχνει αριθμό συμμετοχών έναντι του μέγιστου επιτρεπτού
- Την πλήρη περιγραφή του μαθήματος

Η παρουσίαση είναι καθαρή, ευανάγνωστη και ενισχύει τη διαφάνεια πριν από την τελική απόφαση του χρήστη να εγγραφεί. Η ενέργεια εγγραφής ενεργοποιεί τις κατάλληλες ροές (τεχνικά στο 4.4) και αποστέλλει σχετικές ειδοποιήσεις μέσω email, όπως θα αναλυθεί σε επόμενη ενότητα.



Εικόνα 4.12: Προβολή λεπτομερειών μαθήματος από μη εγγεγραμμένο χρήστη (μη εγγεγραμμένος χρήστης)

4.3.7 Λεπτομέρειες Μαθήματος (μετά την εγγραφή)



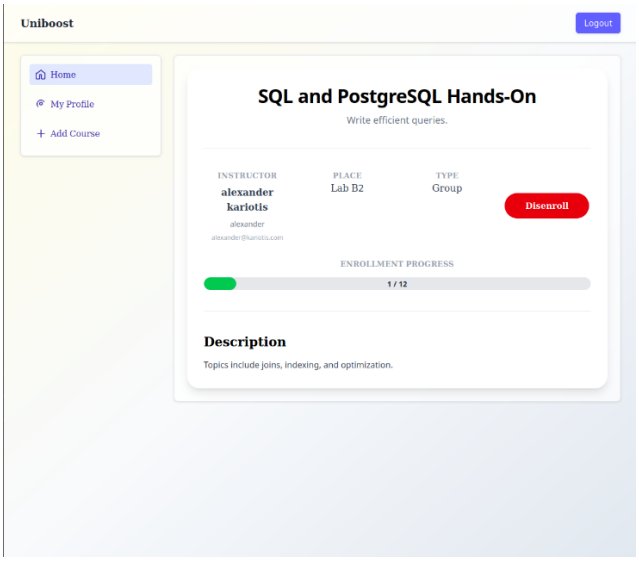
Εικόνα 4.13 Διάγραμμα ροής για τις λεπτομέρειες μαθήματος (μετά την εγγραφή)

Αφού ο χρήστης εγγραφεί σε ένα μάθημα, η ίδια σελίδα λεπτομερειών προσαρμόζεται και εμφανίζει πλέον τη νέα κατάσταση συμμετοχής. Η οθόνη διαφοροποιείται ως εξής:

- Το κουμπί "Enroll" αντικαθίσταται με κουμπί "Disenroll" σε κόκκινο χρώμα, δίνοντας τη δυνατότητα αποχώρησης από το μάθημα.
- Η μπάρα προόδου πλέον περιλαμβάνει την τρέχουσα εγγραφή του χρήστη (π.χ. 1/12)
- Όλες οι υπόλοιπες πληροφορίες παραμένουν ως έχουν, ενισχύοντας την αίσθηση συνέχειας στην εμπειρία χρήστη.

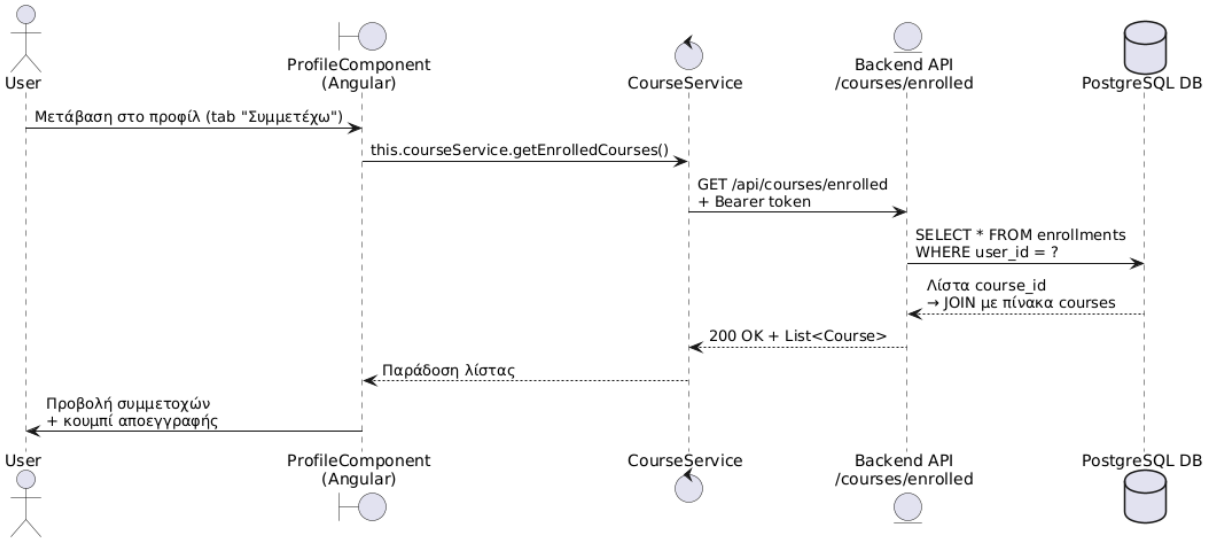
Η δυνατότητα απεγγραφής παρέχει ευελιξία και αυτονομία στον φοιτητή, χωρίς την ανάγκη διαχειριστικής παρέμβασης. Παράλληλα, η ενέργεια απεγγραφής ενεργοποιεί αντίστοιχο email ενημέρωσης προς τον ίδιο και τον δημιουργό του μαθήματος, όπως προβλέπεται από τον μηχανισμό ειδοποιήσεων (θα αναλυθεί στο 4.7).

Η δυναμική εναλλαγή της σελίδας βάσει της κατάστασης του χρήστη συμβάλλει στη βελτιστοποίηση της εμπειρίας και ενισχύει τη διαδραστικότητα της εφαρμογής.



Εικόνα 4.14: Προβολή λεπτομερειών μαθήματος μετά την εγγραφή του χρήστη

4.3.8 Προφίλ Χρήστη – Μαθήματα στα Οποία Συμμετέχω



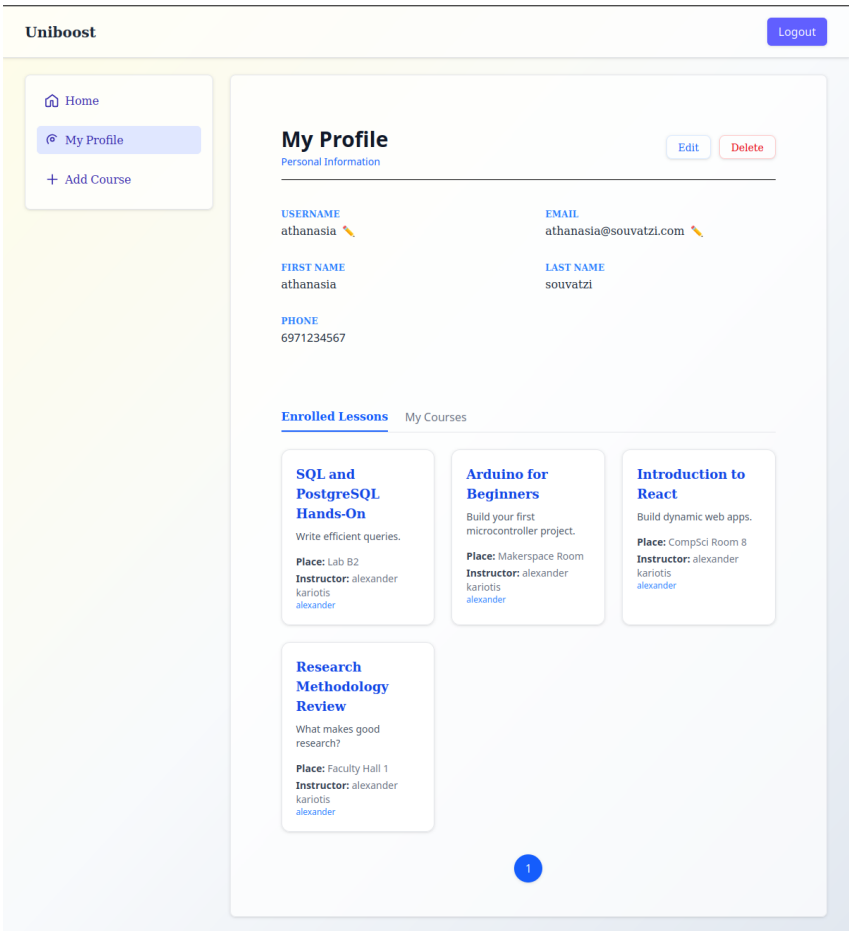
Εικόνα 4.15 Διάγραμμα ροής για το Προφίλ Χρήστη - Μαθήματα στα οποία συμμετέχω

Στην καρτέλα Enrolled Lessons της σελίδας προφίλ, παρουσιάζονται όλα τα μαθήματα στα οποία ο χρήστης έχει εγγραφεί ως φοιτητής. Κάθε εγγραφή προβάλλεται με τη μορφή κάρτας, η οποία περιλαμβάνει:

- τον τίτλο του μαθήματος,
- μια σύντομη περιγραφή,
- την τοποθεσία διεξαγωγής,
- και τον διδάσκοντα (username & όνομα).

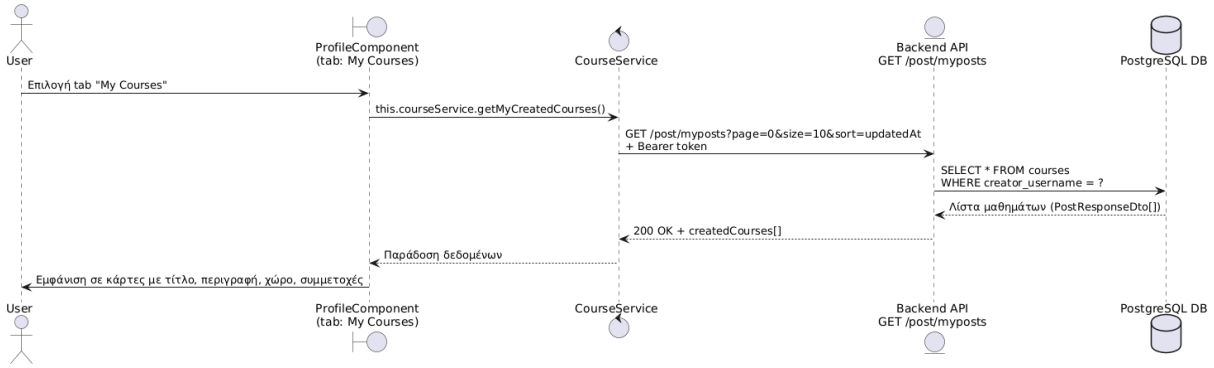
Η προβολή αυτή προσφέρει στον χρήστη εύκολη επισκόπηση των ενεργών συμμετοχών του, με στόχο να εντοπίζει άμεσα τα μαθήματα που παρακολουθεί. Η σχεδίαση είναι καθαρή και με επαρκή πληροφορία ώστε ο φοιτητής να μπορεί να μεταβεί εύκολα σε κάθε επιμέρους μάθημα ή να διαχειριστεί τη συμμετοχή του.

Η σελίδα υποστηρίζει και σελιδοποίηση, ώστε να μπορεί να διαχειριστεί μεγάλος αριθμός συμμετοχών χωρίς να θυσιάζεται η αναγνωσιμότητα.



Εικόνα 4.16: Προβολή μαθημάτων στα οποία συμμετέχει ο χρήστης

4.3.9 Προφίλ Χρήστη – Μαθήματα που Έχω Δημιουργήσει



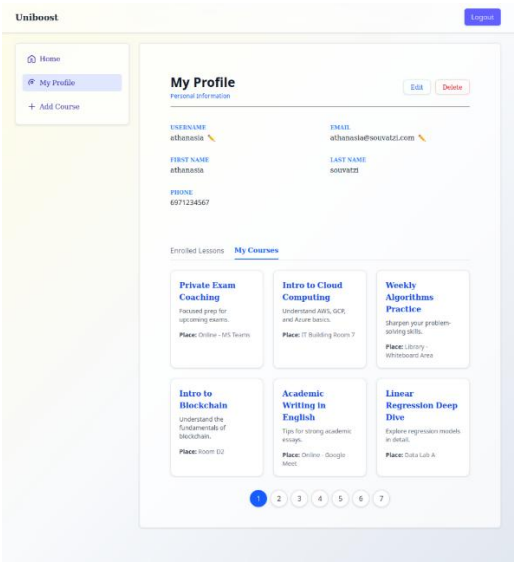
Εικόνα 4.17 Διάγραμμα ροής για το Προφίλ Χρήστη – Μαθήματα που Έχω Δημιουργήσει

Στην καρτέλα My Courses, παρουσιάζονται συγκεντρωτικά όλα τα μαθήματα που έχει δημιουργήσει ο χρήστης. Όπως και στα Enrolled Lessons, κάθε μάθημα εμφανίζεται σε κάρτα, στην οποία περιλαμβάνονται:

- ο τίτλος και η σύντομη περιγραφή του μαθήματος,
- ο χώρος διεξαγωγής (π.χ. αίθουσα, online),
- και βασικές πληροφορίες για την παρακολούθηση.

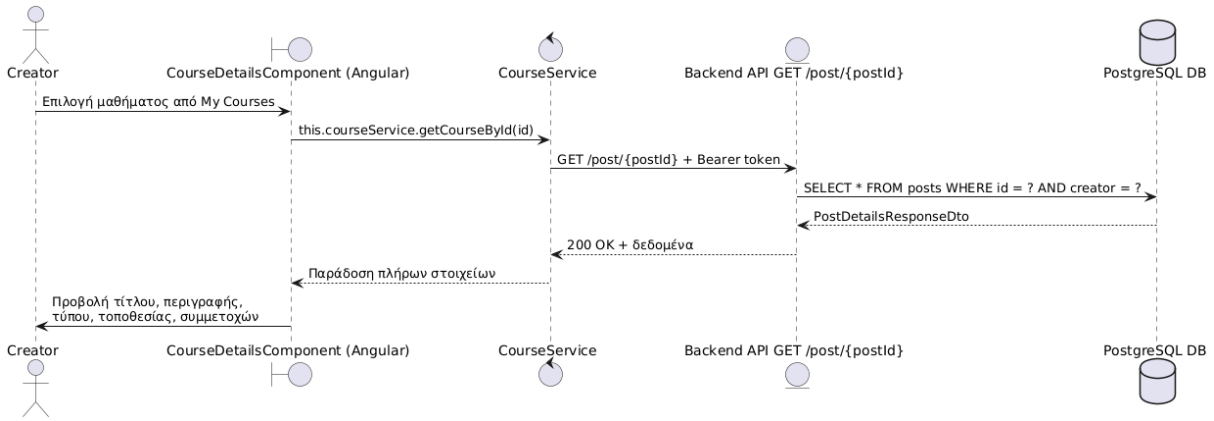
Η προβολή αυτή επιτρέπει στον διδάσκοντα να δει με μια ματιά το σύνολο των μαθημάτων που έχει δημιουργήσει, να αξιολογήσει τη θεματολογία τους και να προχωρήσει σε διαχείριση. Μέσω επιλογών που παρέχονται σε επόμενες σελίδες, μπορεί να προβεί σε επεξεργασία ή διαγραφή κάθε μαθήματος.

Η παρουσία σελιδοποίησης στο κάτω μέρος επιτρέπει τη βολική πλοήγηση ανάμεσα σε πολλές δημιουργίες, διατηρώντας την οθόνη καθαρή και ευανάγνωστη.



Εικόνα 4.18: Προβολή μαθημάτων που έχει δημιουργήσει ο χρήστης

4.3.10 Προβολή Μαθήματος από τον Δημιουργό



Εικόνα 4.19 Διάγραμμα ροής για την προβολή μαθήματος από τον δημιουργό

Όταν ο χρήστης είναι ο δημιουργός ενός μαθήματος, η σελίδα προβολής διαφοροποιείται ώστε να προσφέρει επιπλέον λειτουργίες διαχείρισης. Ο σκοπός αυτής της προβολής είναι να παρέχει πλήρη εικόνα του μαθήματος και τις απαραίτητες επιλογές για τη διαχείρισή του.

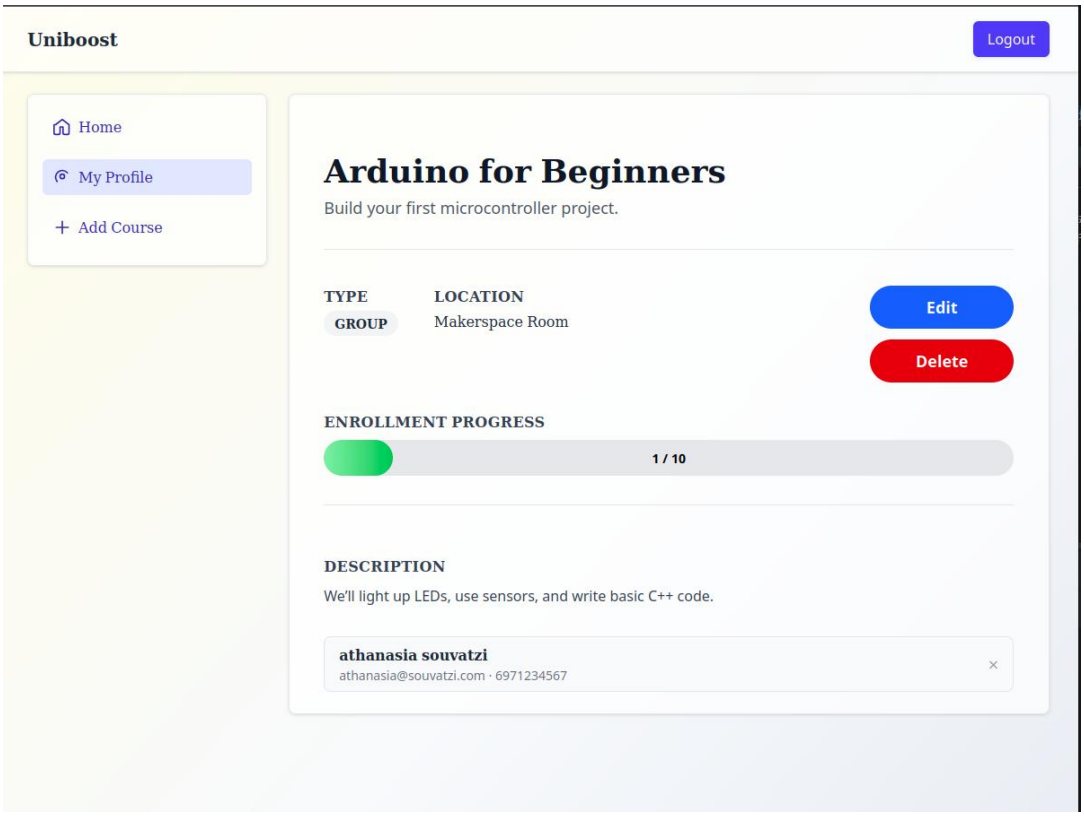
Η οθόνη περιλαμβάνει όλα τα βασικά στοιχεία του μαθήματος: τίτλο, σύντομη περιγραφή, τύπο (group ή personal), τοποθεσία διεξαγωγής, περιγραφή και αριθμό συμμετεχόντων μέσω της μπάρας προόδου.

Επιπλέον, εμφανίζονται δύο επιλογές διαχείρισης:

- edit: οδηγεί σε ξεχωριστή οθόνη όπου ο δημιουργός μπορεί να τροποποιήσει όλα τα πεδία εκτός από τον τίτλο του μαθήματος, ο οποίος παραμένει σταθερός
- delete: επιτρέπει τη διαγραφή του μαθήματος

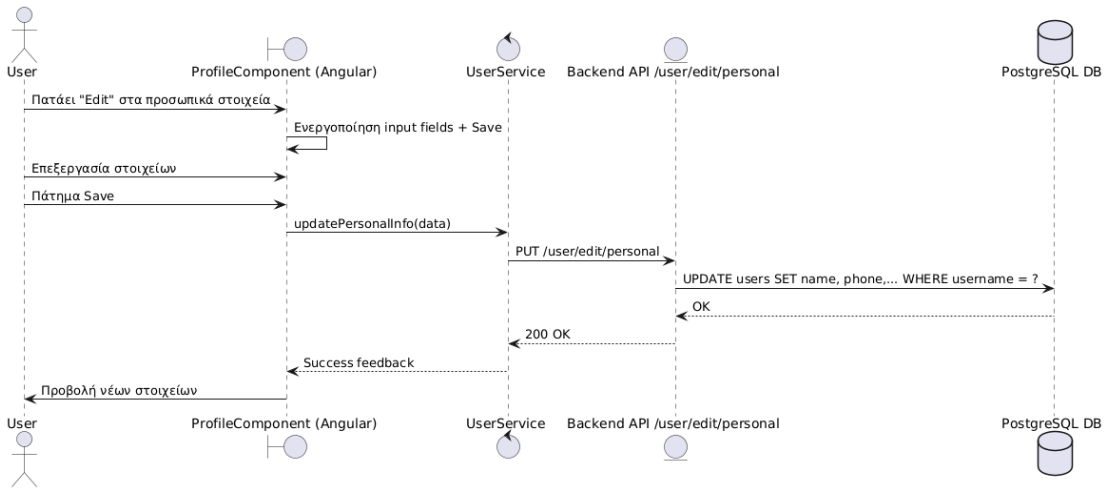
Στο κάτω μέρος της σελίδας εμφανίζεται η λίστα των εγγεγραμμένων φοιτητών, με στοιχεία όπως ονοματεπώνυμο, email και τηλέφωνο. Για κάθε φοιτητή υπάρχει διαθέσιμη επιλογή διαγραφής από το μάθημα.

Η προβολή αυτή επιτρέπει στον διδάσκοντα να διαχειριστεί το μάθημά του με σαφήνεια και ευκολία, προσφέροντάς του πρόσβαση σε όλες τις λειτουργίες που σχετίζονται με το περιεχόμενο και τους συμμετέχοντες.

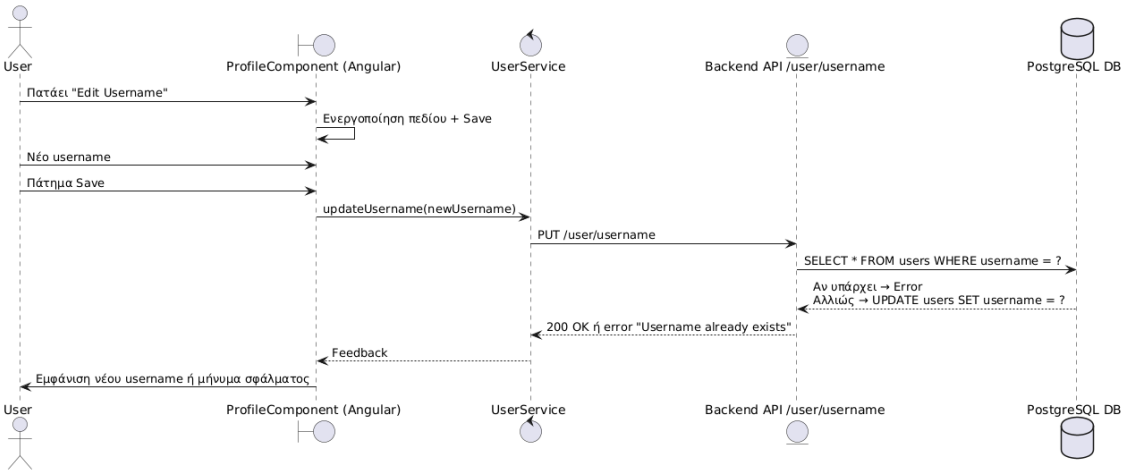


Εικόνα 4.20: Προβολή μαθήματος από την πλευρά του δημιουργού με δυνατότητα επεξεργασίας και διαχείρισης συμμετοχών

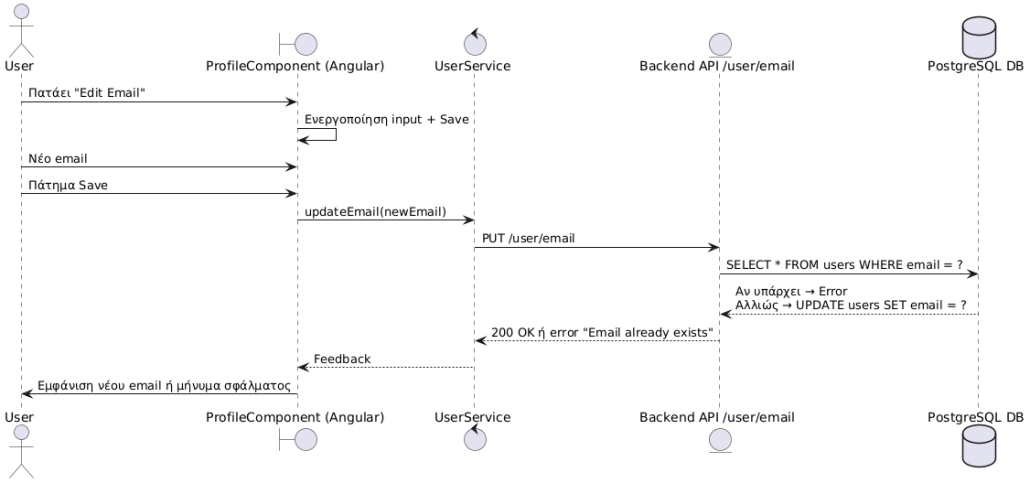
4.3.11 Επεξεργασία Προφίλ Χρήστη (Edit Mode)



Εικόνα 4.21 Διάγραμμα ροής για την επεξεργασία των προσωπικών πληροφοριών



Εικόνα 4.22 Διάγραμμα ροής για την επεξεργασία του username



Εικόνα 4.23 Διάγραμμα ροής για την επεξεργασία του email

Η εφαρμογή επιτρέπει στον χρήστη να επεξεργαστεί τα προσωπικά του στοιχεία μέσω ενός ευέλικτου και διακριτού μηχανισμού. Η επεξεργασία γίνεται σε τρία ανεξάρτητα μέρη, προσφέροντας ακριβή έλεγχο και εστιασμένη λειτουργικότητα.

Οι τρεις λειτουργίες επεξεργασίας είναι οι εξής:

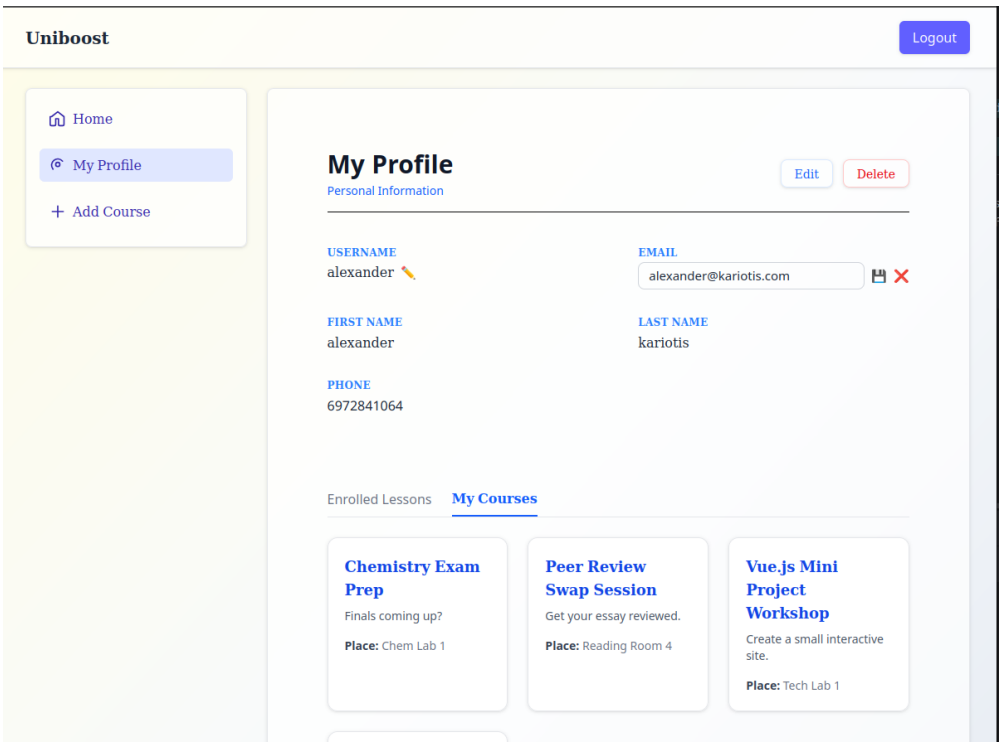
- Επεξεργασία προσωπικών στοιχείων
  Ο χρήστης μπορεί να τροποποιήσει τα πεδία first name, last name και phone. Μόλις ενεργοποιηθεί η λειτουργία, τα πεδία μετατρέπονται σε πεδία εισαγωγής και εμφανίζονται κουμπιά για αποθήκευση ή ακύρωση της αλλαγής. Η διαδικασία είναι άμεση και λειτουργεί χωρίς επαναφόρτωση της σελίδας.

- Επεξεργασία username

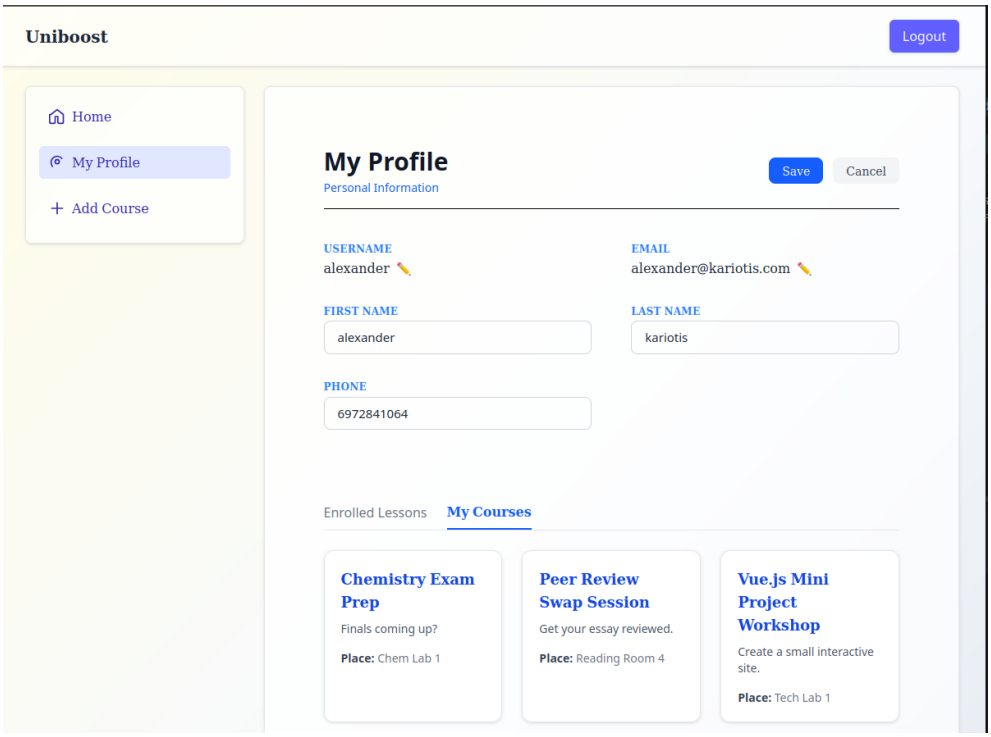
  Το username είναι μοναδικό στη βάση δεδομένων και οποιαδήποτε αλλαγή ελέγχεται για ενδεχόμενα διπλότυπα. Το πεδίο γίνεται προσωρινά επεξεργάσιμο και συνοδεύεται από κουμπί αποθήκευσης και ακύρωσης. Σε περίπτωση που το νέο username χρησιμοποιείται ήδη, εμφανίζεται σχετικό μήνυμα σφάλματος.
- Επεξεργασία email

  Η τροποποίηση του email λειτουργεί με παρόμοιο τρόπο. Ο χρήστης εισάγει νέο email σε ειδικό πεδίο και στη συνέχεια μπορεί να το αποθηκεύσει ή να ακυρώσει την αλλαγή. Το email είναι επίσης μοναδικό και επαληθεύεται ως προς τη μορφή και την ύπαρξή του.

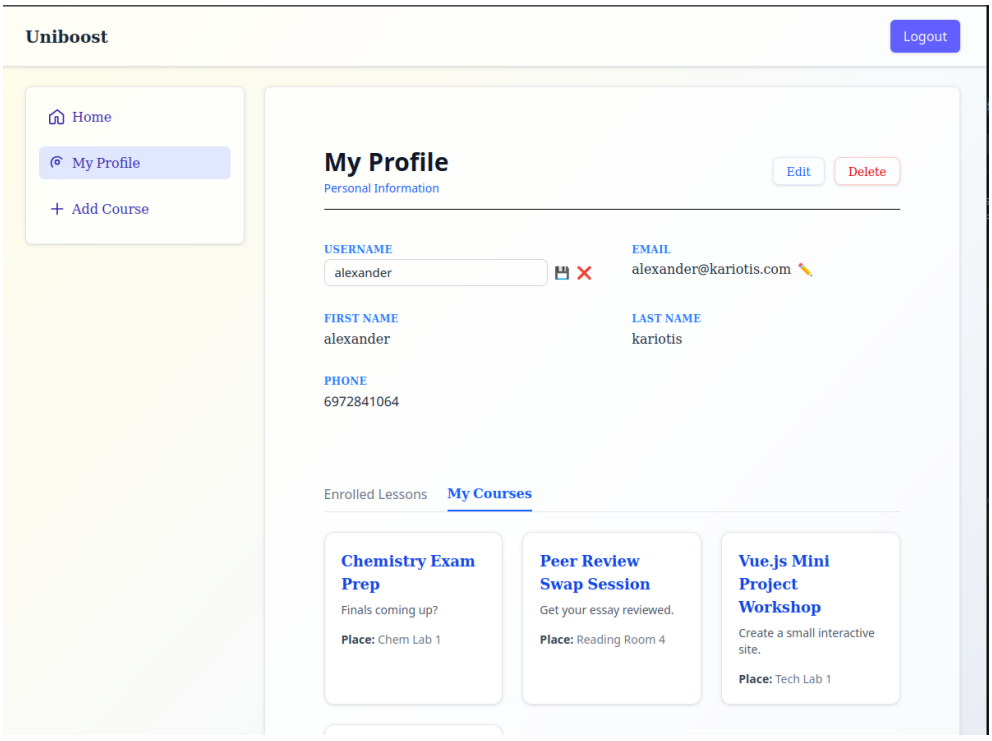
Η λειτουργία edit mode εφαρμόζει έναν απλό και αποτελεσματικό σχεδιασμό, παρέχοντας οπτικό διαχωρισμό των πεδίων και περιορίζοντας τις ενέργειες του χρήστη μόνο στο συγκεκριμένο στοιχείο που επιθυμεί να αλλάξει. Αυτό εξασφαλίζει ακρίβεια, καλύτερη εμπειρία χρήσης και προστασία των κρίσιμων προσωπικών δεδομένων.



Εικόνα 4.24: Επεξεργασία μοναδικού email χρήστη

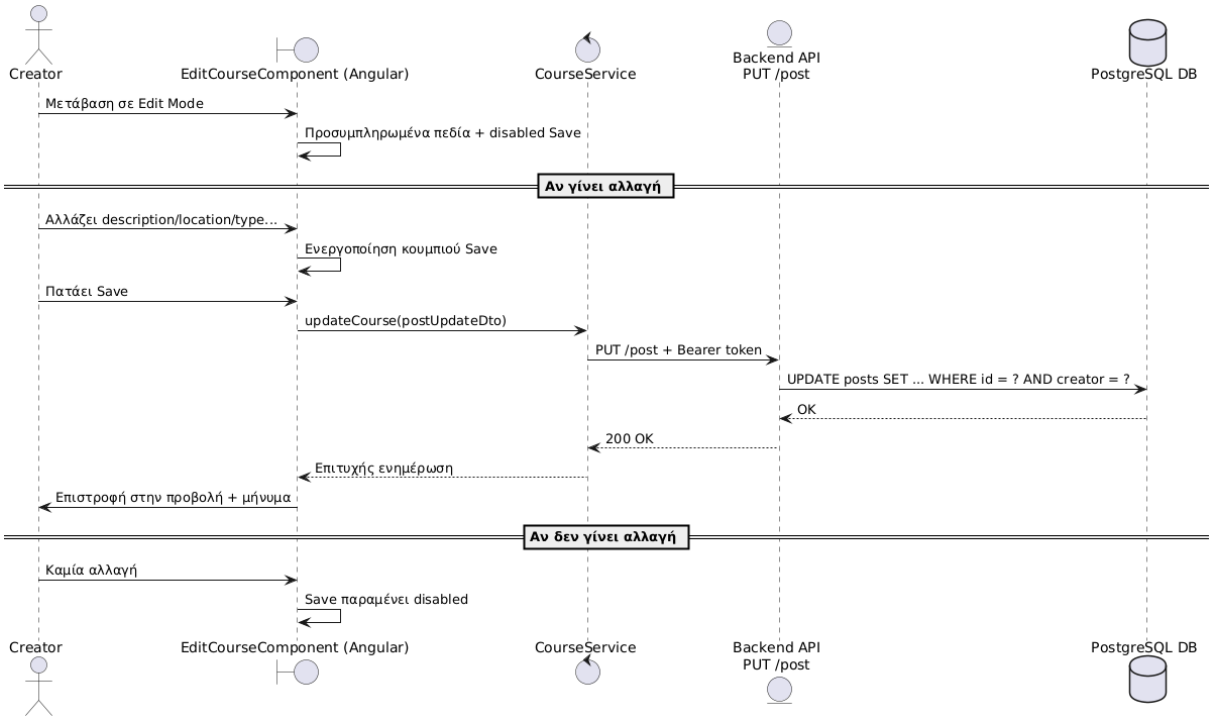


Εικόνα 4.25: Επεξεργασία προσωπικών στοιχείων χρήστη (όνομα, επώνυμο, τηλέφωνο)



Εικόνα 4.26: Επεξεργασία μοναδικού username χρήστη

4.3.12 Επεξεργασία Μαθήματος (Edit Mode)



Εικόνα 4.27 Διάγραμμα ροής για την επεξεργασία μαθήματος (Edit Mode)

Η εφαρμογή παρέχει στον δημιουργό κάθε μαθήματος τη δυνατότητα να επεξεργαστεί τις πληροφορίες του, μέσα από ειδικά διαμορφωμένη οθόνη edit mode. Η λειτουργία αυτή ενεργοποιείται από την προβολή μαθήματος και επιτρέπει την τροποποίηση των εξής πεδίων:

- σύντομη περιγραφή (preview description)
- περιγραφή (description)
- τοποθεσία (location)
- τύπος μαθήματος (group ή personal)
- μέγιστος αριθμός εγγραφών (max enrolls)

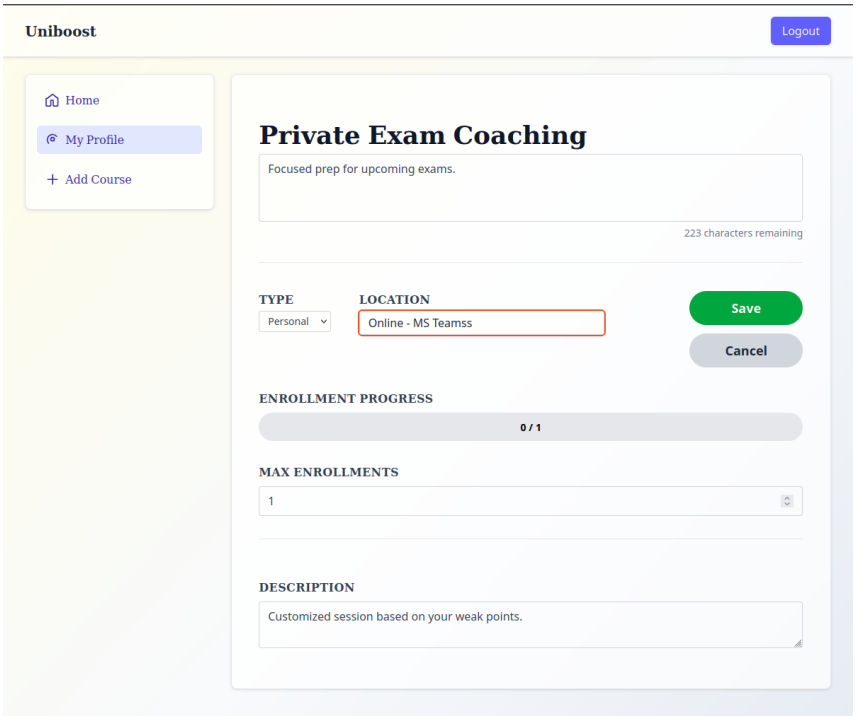
Ο τίτλος του μαθήματος παραμένει σταθερός και δεν είναι επεξεργάσιμος.

Αρχικά, όλα τα πεδία εμφανίζονται προσυμπληρωμένα με τις τρέχουσες τιμές. Ο χρήστης μπορεί να τα τροποποιήσει κατά βούληση. Αν όμως δεν αλλάξει τίποτα, τότε το κουμπί "Save" παραμένει ανενεργό, αποτρέποντας την αποστολή περιττού αιτήματος προς τον διακομιστή.

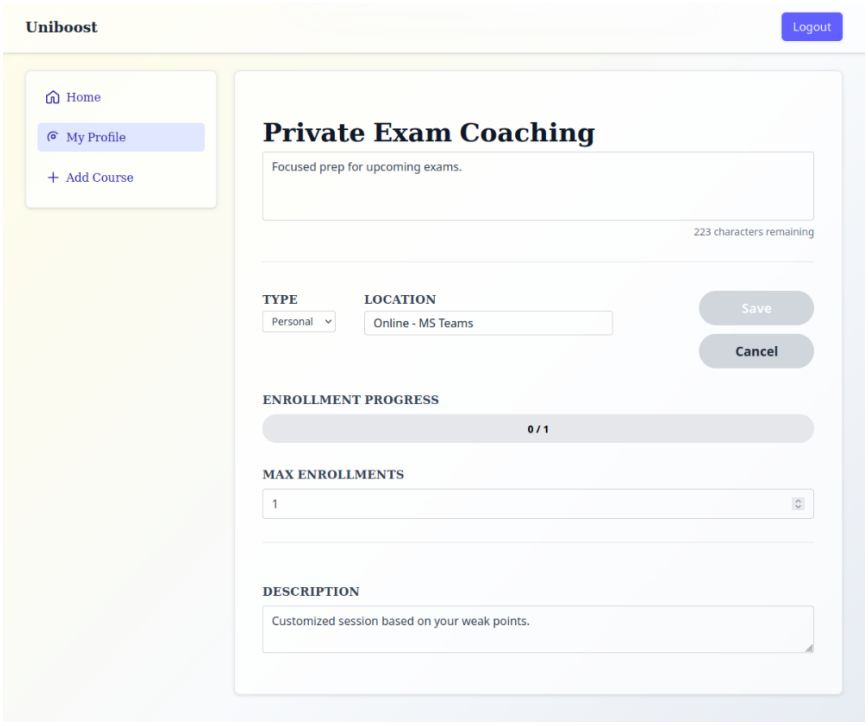
Σε περίπτωση που πραγματοποιηθεί έστω και μία αλλαγή σε οποιοδήποτε πεδίο, το κουμπί αποθήκευσης ενεργοποιείται και η νέα πληροφορία αποστέλλεται στη βάση δεδομένων, ενημερώνοντας τα δεδομένα του μαθήματος.

Η παρουσία του κουμπιού "Cancel" επιτρέπει την άμεση επιστροφή στην προβολή χωρίς να χαθούν τα προηγούμενα δεδομένα, προσφέροντας έτσι μεγαλύτερη ευελιξία στον χρήστη.

Η συμπεριφορά της φόρμας ακολουθεί τις αρχές της ευχρηστίας και της αποτελεσματικότητας, αποτρέποντας περιττές ενέργειες και ενισχύοντας τη σταθερότητα της εφαρμογής.

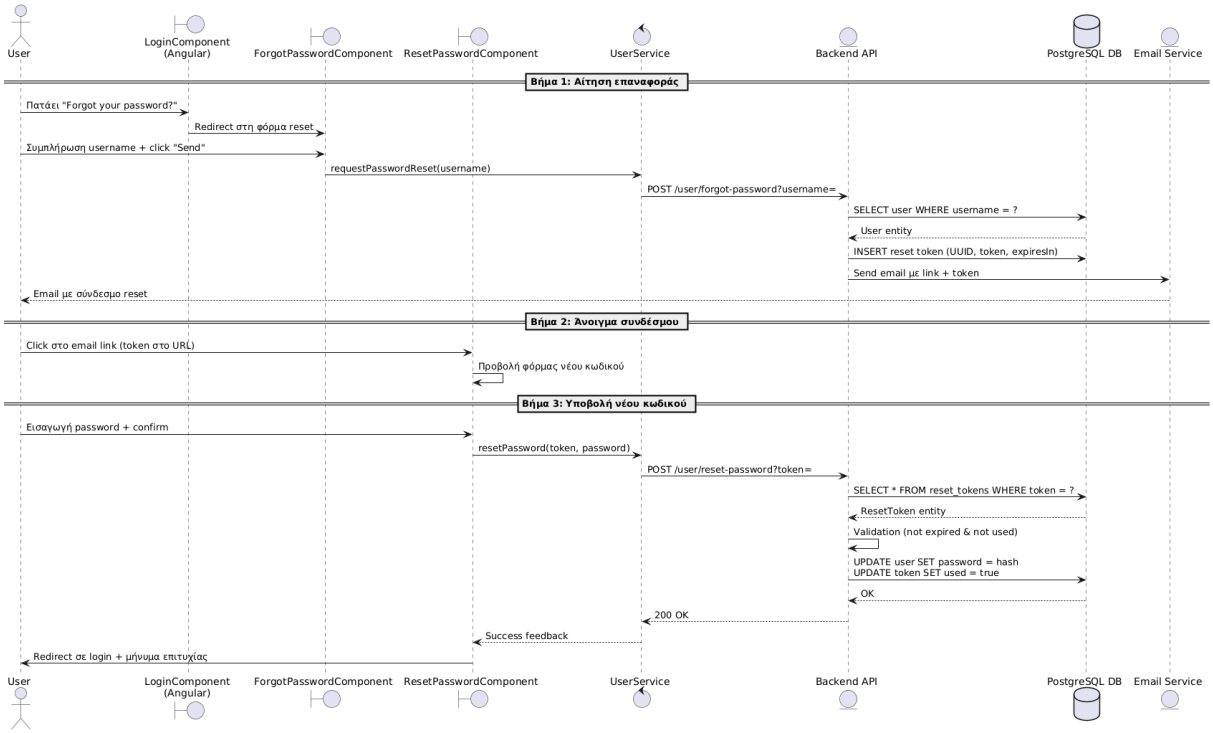


Εικόνα 4.28: Επεξεργασία μαθήματος με αλλαγή πεδίου και ενεργοποίηση του κουμπιού αποθήκευσης



Εικόνα 4.29: Επεξεργασία μαθήματος χωρίς αλλαγές – η επιλογή αποθήκευσης παραμένει απενεργοποιημένη

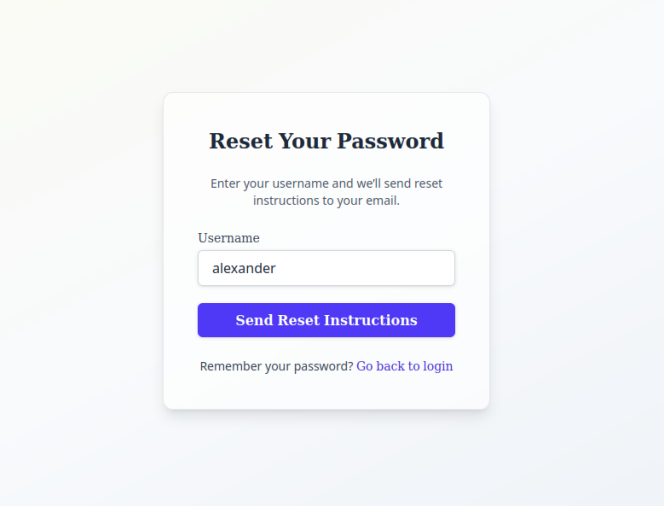
4.3.13 Επαναφορά Κωδικού Πρόσβασης



Εικόνα 4.30 Διάγραμμα ροής για την επαναφορά του κωδικού πρόσβασης

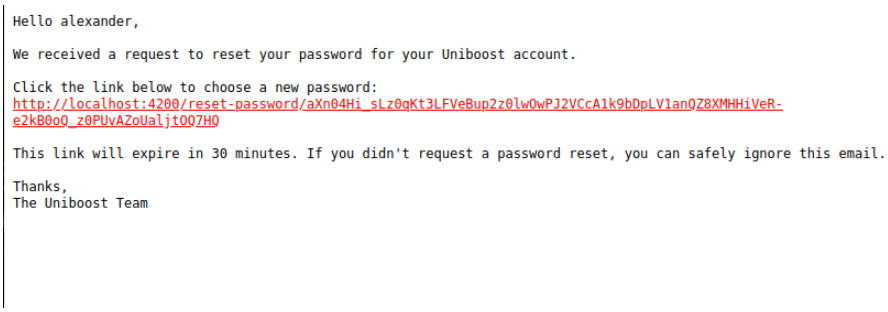
Η εφαρμογή παρέχει λειτουργία επαναφοράς κωδικού πρόσβασης σε περίπτωση που ο χρήστης ξεχάσει τον λογαριασμό του ή δεν έχει πρόσβαση στον τρέχοντα κωδικό. Η διαδικασία είναι χωρισμένη σε τρία στάδια, με σαφή και καθαρό διαχωρισμό ροής.

Στο πρώτο βήμα, ο χρήστης μεταφέρεται στη σελίδα «Reset Your Password» μέσω σχετικού συνδέσμου που βρίσκεται στη φόρμα σύνδεσης. Εκεί καλείται να συμπληρώσει το username του και να πατήσει την επιλογή «Send Reset Instructions».



Εικόνα 4.31: Αίτηση επαναφοράς κωδικού από τον χρήστη

Αν το username αντιστοιχεί σε χρήστη που υπάρχει στη βάση δεδομένων, το σύστημα αποστέλλει αυτοματοποιημένο email στη διεύθυνση του χρήστη, με προσωρινό σύνδεσμο επαναφοράς κωδικού. Το email περιλαμβάνει σύντομες οδηγίες και σύνδεσμο που παραμένει ενεργός για χρονικό διάστημα 30 λεπτών. Αν ο χρήστης δεν αιτήθηκε ο ίδιος την αλλαγή, μπορεί να αγνοήσει το email χωρίς επιπτώσεις.


Hello alexander,

We received a request to reset your password for your Uniboost account.

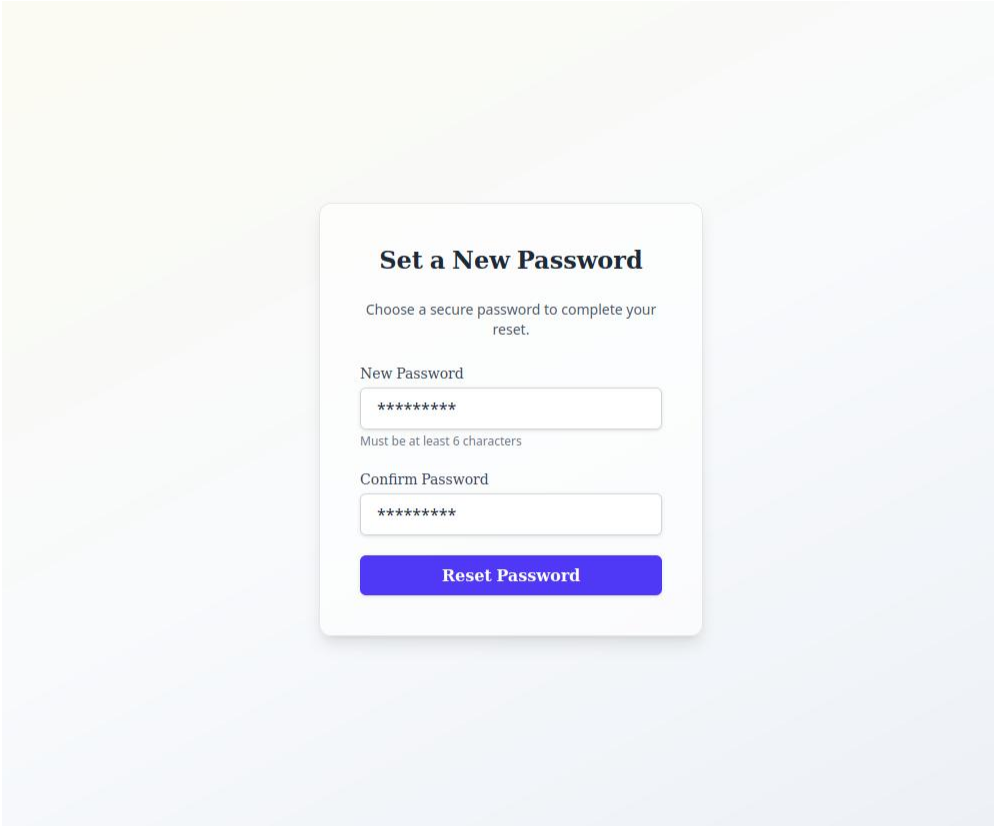
Click the link below to choose a new password:
http://localhost:4200/reset-password/aXn04Hi_sLz0qKt3LFVeBup2z0lwOwPJ2VCcA1k9bDpLV1anQZ8XMHHiVeR-e2kB0oQ_z0PUvAZoUaljtOQ7HQ

This link will expire in 30 minutes. If you didn't request a password reset, you can safely ignore this email.

Thanks,
The Uniboost Team

Εικόνα 4.32: Email με σύνδεσμο επαναφοράς κωδικού

Μόλις ο χρήστης πατήσει στον σύνδεσμο επαναφοράς, μεταφέρεται στη σελίδα «Set a New Password», όπου καλείται να εισάγει νέο κωδικό και επιβεβαίωση. Ο κωδικός πρέπει να είναι τουλάχιστον έξι χαρακτήρες. Αν οι δύο τιμές συμφωνούν, η αλλαγή αποθηκεύεται, και ο χρήστης μπορεί πλέον να χρησιμοποιήσει τον νέο κωδικό για να συνδεθεί.



Εικόνα 4.33: Φόρμα εισαγωγής νέου κωδικού πρόσβασης από τον χρήστη

Η συνολική εμπειρία επαναφοράς είναι σχεδιασμένη έτσι ώστε να είναι ασφαλής, κατανοητή και άμεση, με έλεγχο εγκυρότητας του συνδέσμου, περιορισμό χρονικής ισχύος, και αποφυγή έκθεσης προσωπικών δεδομένων.

4.4 Τεχνική Υλοποίηση

Αφού παρουσιάστηκαν οι λειτουργικές δυνατότητες και η εμπειρία χρήσης της εφαρμογής, στο παρόν υποκεφάλαιο περιγράφεται η τεχνική της υλοποίηση. Εδώ αναλύεται η εσωτερική λειτουργία των βασικών ροών της εφαρμογής, τόσο από την πλευρά του backend όσο και του frontend, με έμφαση στη δομή των API endpoints, τη ροή δεδομένων, την αυθεντικοποίηση, τις επικοινωνίες μεταξύ επιπέδων, καθώς και τις επικυρώσεις και ελέγχους που εφαρμόζονται σε κάθε ενέργεια.

Η εφαρμογή βασίζεται σε αρχιτεκτονική τύπου client-server. Το backend υλοποιήθηκε με χρήση του Spring Boot και εκθέτει RESTful endpoints για την εξυπηρέτηση των λειτουργιών. Η αποθήκευση των δεδομένων γίνεται σε PostgreSQL βάση, ενώ το frontend αναπτύχθηκε με Angular και επικοινωνεί με το backend μέσω HTTP αιτήσεων.

Η ανάλυση που ακολουθεί είναι οργανωμένη ανά βασική λειτουργία, ώστε να αποδοθεί με ακρίβεια η τεχνική ροή της εφαρμογής και οι αντίστοιχες σχεδιαστικές αποφάσεις.

4.4.1 Εγγραφή Χρήστη (Signup)

Η εγγραφή νέου χρήστη υλοποιείται μέσω του HTTP endpoint:

```
POST /user/register
```

Από το frontend αποστέλλεται ένα αντικείμενο τύπου UserCreateDto, το οποίο περιέχει τα εξής πεδία: Το DTO έχει την εξής δομή:

```
public class UserCreateDto {
    private String username;
    private String password;
    private String firstName;
    private String lastName;
    private String email;
    private String phone;
}
```

Η αίτηση παραλαμβάνεται από τον controller:

```
@PostMapping("register")
public ResponseEntity<AuthenticationResponseDto> register(
        @RequestBody UserCreateDto createDto
) {
    return userService
            .register(UserMapper.userCreateDtoToUser(createDto))
            .map(ResponseEntity::ok)
            .onFailure(Throwable::printStackTrace)
            .get();
}
```

Το DTO μετατρέπεται σε οντότητα User μέσω του UserMapper και προωθείται στην υπηρεσία UserService.

Η μέθοδος register() στο service εκτελεί:

- έλεγχο για ύπαρξη του username ή του email,
- κρυπτογράφηση του κωδικού πρόσβασης,

- αποθήκευση του χρήστη στη βάση δεδομένων,
- αποστολή email καλωσορίσματος,
- δημιουργία access και refresh token,
- αποθήκευση του access token,
- και τελική επιστροφή AuthenticationResponseDto.

Το βασικό μέρος της υλοποίησης είναι το εξής:

```
public Try<AuthenticationResponseDto> register(User user) {
    return Option.ofOptional(userRepository.findByUsername(user.getUsername()))
        .fold(() -> Option.ofOptional(userRepository.findByEmail(user.getEmail()))
            .fold(() -> Try.of(() -> user)
                .map(u -> {
                    u.setId(UUID.randomUUID());
                    u.setPassword(passwordEncoder.encode(u.getPassword()));
                    u.setCreatedAt(OffsetDateTime.now());
                    u.setUpdatedAt(OffsetDateTime.now());
                    return u;
                })
                .map(userRepository::saveAndFlush)
                .map(savedUser -> {
                    emailService.sendEmail(...);
                    return savedUser;
                })
                .map(user -> Tuple.of(
                        user,
                        jwtUtils.generateToken(user),
                        jwtUtils.generateRefreshToken(user)))
                .map(tuple -> Tuple.of(saveAccessToken(tuple._1, tuple._2).get(),
tuple._3))
                .map(tuple -> AuthenticationResponseDto.builder()
                        .accessToken(tuple._1.getToken())
                        .refreshToken(tuple._2)
                        .build())
            , emailExists -> Try.failure(new IllegalArgumentException("Email
already exists")))
        , usernameExists -> Try.failure(new IllegalArgumentException("Username
already exists")));
}
```

Η απάντηση που επιστρέφεται στο frontend είναι:

```
public class AuthenticationResponseDto {
    @JsonProperty("access_token")
    private String accessToken;

    @JsonProperty("refresh_token")
    private String refreshToken;
}
```

Το access token χρησιμοποιείται σε μελλοντικές αιτήσεις του χρήστη, ενώ το refresh token επιτρέπει την ανανέωση της σύνδεσης χωρίς επαναλαμβανόμενη είσοδο.

4.4.2 Σύνδεση Χρήστη (Login)

Η λειτουργία σύνδεσης χρήστη είναι υπεύθυνη για την αυθεντικοποίηση και την έναρξη της συνεδρίας ενός ήδη εγγεγραμμένου χρήστη στην εφαρμογή. Η υλοποίηση βασίζεται στην ανταλλαγή JSON δεδομένων μέσω HTTP και στην τεχνολογία JWT για την απονομή και χρήση tokens ασφαλείας.

Η επικοινωνία ξεκινά από το frontend, το οποίο αποστέλλει HTTP POST αίτημα προς το backend endpoint:

```
POST /user/login
```

Το σώμα της αίτησης περιλαμβάνει το αντικείμενο AuthenticationRequestDto, το οποίο περιέχει δύο πεδία: username και password. Παράδειγμα αιτήματος:

```
{
  "username": "student42",
  "password": "mypassword123"
}
```

Το αίτημα παραλαμβάνεται από τον controller UserController, ο οποίος προωθεί το αντικείμενο στην υπηρεσία UserService για επεξεργασία:

```
@PostMapping("login")
public ResponseEntity<AuthenticationResponseDto> login(
        @RequestBody AuthenticationRequestDto authenticationRequestDto
) {
    return userService
            .authenticate(authenticationRequestDto)
```

```
            .map(ResponseEntity::ok)
            .onFailure(Throwable::printStackTrace)
            .get();
}
```

Η χρήση του Try API επιτρέπει τον ασφαλή χειρισμό εξαιρέσεων και την επιστροφή του αποτελέσματος με λειτουργικό τρόπο.

Η μέθοδος authenticate() αποτελεί την κύρια είσοδο για τον έλεγχο ταυτότητας. Η λογική που ακολουθείται είναι η εξής:

Ο χρήστης αναζητείται στη βάση δεδομένων με βάση το username. Αν δεν βρεθεί, επιστρέφεται εξαίρεση. Αν βρεθεί, επιχειρείται ταυτοποίηση μέσω του authenticationManager. Αν ο κωδικός είναι λανθασμένος, εμφανίζεται εξαίρεση BadCredentialsException. Αν τα διαπιστευτήρια είναι έγκυρα, δημιουργείται access token και refresh token, ανακαλούνται προηγούμενα tokens, αποθηκεύεται το νέο και τελικά επιστρέφεται η απάντηση στο frontend.

Ο πλήρης κώδικας της μεθόδου είναι ο εξής:

```
public Try<AuthenticationResponseDto> authenticate(AuthenticationRequestDto
requestDto) {
    return Try.of(() ->
Option.ofOptional(userRepository.findByUsername(requestDto.getUsername()))
        .getOrElseThrow(() -> new IllegalArgumentException("There is no user with
username: " + requestDto.getUsername())))
    .flatMap(user -> Try.of(() -> authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                    requestDto.getUsername(),
                    requestDto.getPassword()))))
        .map(auth -> user)
        .recover(BadCredentialsException.class, e -> {
            throw new IllegalArgumentException("Wrong password provided", e);
        }))
    .map(user -> Tuple.of(
            user,
            jwtUtils.generateToken(user),
            jwtUtils.generateRefreshToken(user)))
    .flatMap(tuple -> revokeAllAccessTokens(tuple._1).map(ignored -> tuple))
    .map(tuple -> Tuple.of(
            saveAccessToken(tuple._1, tuple._2).getOrElseThrow(
                    ex -> new RuntimeException("Access Token didn't save in
database.", ex)),
            tuple._3))
    .map(tuple -> AuthenticationResponseDto.builder()
```

```
            .accessToken(tuple._1.getToken())
            .refreshToken(tuple._2)
            .build());
}
```

Η απάντηση που αποστέλλεται προς το frontend περιέχει τα δύο tokens:

```
public class AuthenticationResponseDto {
    @JsonProperty("access_token")
    private String accessToken;

    @JsonProperty("refresh_token")
    private String refreshToken;
}
```

Το access token αποθηκεύεται προσωρινά (π.χ. στο sessionStorage) και χρησιμοποιείται για την επικύρωση κάθε μελλοντικής αίτησης. Το refresh token μπορεί να αξιοποιηθεί για την αυτόματη ανανέωση της σύνδεσης.

Σε περίπτωση αποτυχίας ελέγχου ταυτότητας, το σύστημα επιστρέφει κατάλληλο μήνυμα σφάλματος προς το frontend, με σαφείς πληροφορίες για το τι πήγε λάθος (π.χ. λανθασμένος κωδικός ή ανύπαρκτος χρήστης). Η ασφάλεια της διαδικασίας διασφαλίζεται μέσω μηχανισμών αυθεντικοποίησης και διαχείρισης tokens, των οποίων η τεχνική υλοποίηση θα παρουσιαστεί σε επόμενη ενότητα.

Η σύνδεση χρήστη αποτελεί την βασική λειτουργία εισόδου στην εφαρμογή και η σωστή υλοποίησή της εγγυάται την προστασία των προσωπικών δεδομένων και των διαθέσιμων λειτουργιών του συστήματος.

4.4.3 Επαναφορά Κωδικού Πρόσβασης

Η εφαρμογή υποστηρίζει μηχανισμό επαναφοράς κωδικού πρόσβασης μέσω προσωρινού token, το οποίο αποστέλλεται στον χρήστη με email. Η διαδικασία επιτρέπει στον χρήστη να επανακτήσει την πρόσβαση στον λογαριασμό του χωρίς παρέμβαση διαχειριστή, και είναι σχεδιασμένη με γνώμονα την ασφάλεια και την ευχρηστία.

Η ροή περιλαμβάνει δύο στάδια: την αίτηση για επαναφορά και την καταχώριση νέου κωδικού.

Η διαδικασία ξεκινά με αίτημα POST από το frontend προς το endpoint:

```
POST /user/forgot-password
```

Το αίτημα περιλαμβάνει το username του χρήστη ως query parameter. Το backend το παραλαμβάνει μέσω της εξής μεθόδου στον controller:

```
@PostMapping("forgot-password")
public ResponseEntity<Void> requestToken(
        @RequestParam(name = "username") String username
) {
    return userService.requestToken(username)
            .map(ResponseEntity::ok)
            .get();
}
```

Η υπηρεσία `UserService` εκτελεί τα εξής βήματα:

- Αναζητά τον χρήστη με βάση το username.
- Δημιουργεί νέο reset token συνδεδεμένο με τον χρήστη.
- Αποθηκεύει το token στη βάση δεδομένων με χρονική ισχύ 30 λεπτών.
- Αποστέλλει email στον χρήστη με σύνδεσμο επαναφοράς.

Το βασικό απόσπασμα κώδικα είναι το εξής:

```
public Try<Void> requestToken(String username) {
    return Try.of(() -> userRepository.findByUsername(username)
            .orElseThrow(() -> new IllegalArgumentException("Username not found")))
        .map(user -> resetTokenRepository.saveAndFlush(ResetToken.newToken(user)))
        .flatMap(resetToken -> emailService.sendEmail(SendEmailDto.builder()
            .sendTo(resetToken.getUser().getEmail())
            .subject("Reset Your Password – Uniboost")
            .body("""
                Hello %s,

                We received a request to reset your password for your Uniboost
account.

                Click the link below to choose a new password:
                %s

                This link will expire in 30 minutes. If you didn't request a
password reset, you can safely ignore this email.

                Thanks,
```

```
                The Uniboost Team
                """.formatted(resetToken.getUser().getFirstname(), resetLink +
resetToken.getToken()))
            .build()));
}
```

Η δημιουργία του reset token γίνεται μέσω της στατικής μεθόδου `ResetToken.newToken(User user)`. Η μέθοδος δημιουργεί ένα αντικείμενο τύπου `ResetToken` με τα εξής χαρακτηριστικά:

- μοναδικό αναγνωριστικό (UUID)
- ασφαλές token 64 bytes, παραγόμενο μέσω `SecureRandom`
- χρονική ισχύ 30 λεπτών
- κατάσταση unused
- σύνδεση με τον χρήστη που το αιτήθηκε

Η υλοποίηση έχει ως εξής:

```
public static ResetToken newToken(User user) {
    long thirtyMinutes = 30 * 60 * 1000;
    return ResetToken.builder()
            .id(UUID.randomUUID())
            .token(token())
            .expiredAt(new Date(System.currentTimeMillis() + thirtyMinutes))
            .createdAt(new Date(System.currentTimeMillis()))
            .user(user)
            .used(false)
            .build();
}

private static String token() {
    SecureRandom random = new SecureRandom();
    byte[] bytes = new byte[64];
    random.nextBytes(bytes);
    return Base64.getUrlEncoder().withoutPadding().encodeToString(bytes);
}
```

Η χρήση του SecureRandom και η παραγωγή Base64 URL-safe token εγγυώνται υψηλό επίπεδο ασφάλειας και μη προβλεψιμότητας.

Ο χρήστης λαμβάνει email που περιέχει προσωρινό σύνδεσμο επαναφοράς, ο οποίος οδηγεί στη σελίδα καταχώρισης νέου κωδικού. Το email περιλαμβάνει επίσης ενημέρωση για τη χρονική ισχύ του συνδέσμου και οδηγίες ασφαλείας. Το περιεχόμενο και η μορφή του email αποτυπώνονται και στην ενότητα 4.3.13.

Στη συνέχεια, το frontend αποστέλλει νέο POST αίτημα για να οριστεί ο νέος κωδικός:

```
POST /user/reset-password
```

Το αίτημα περιλαμβάνει το token ως query parameter και το νέο password ως JSON body. Το backend διαχειρίζεται το αίτημα με την παρακάτω μέθοδο:

```
@PostMapping("reset-password")
public ResponseEntity<Void> resetPassword(
        @RequestParam(name = "token") String token,
        @RequestBody ResetPasswordRequestDto requestDto
) {
    return userService.resetPassword(token, requestDto.getPassword())
            .map(ResponseEntity::ok)
            .get();
}
```

Η μέθοδος resetPassword() στο UserService:

- εντοπίζει το reset token στη βάση
- ελέγχει αν είναι ήδη χρησιμοποιημένο ή έχει λήξει
- κρυπτογραφεί και αποθηκεύει τον νέο κωδικό
- μαρκάρει το token ως used

Η αντίστοιχη υλοποίηση:

```
public Try<Void> resetPassword(String token, String password) {
    return Try.of(() -> resetTokenRepository.findByToken(token)
            .orElseThrow(() -> new RuntimeException("Invalid Token!")))
        .filter(resetToken -> !resetToken.isUsed() &&
                              resetToken.getExpiredAt().after(new
Date(System.currentTimeMillis())),
                ex -> new RuntimeException("This token is used!"))
        .map(resetToken -> {
            resetToken.getUser().setPassword(passwordEncoder.encode(password));
            resetToken.setUsed(true);
            return resetTokenRepository.saveAndFlush(resetToken);
        });
}
```

Η επαναφορά κωδικού αποτελεί κρίσιμη λειτουργία υποστήριξης χρηστών. Η χρήση μοναδικού token, ο χρονικός περιορισμός, οι έλεγχοι εγκυρότητας και η ενημέρωση του χρήστη μέσω email διασφαλίζουν την αξιοπιστία και την ασφάλεια της διαδικασίας.

4.4.4 Διαχείριση Μαθημάτων (CRUD)

Η διαχείριση των μαθημάτων αποτελεί μία από τις βασικότερες λειτουργίες της εφαρμογής. Οι χρήστες έχουν τη δυνατότητα να δημιουργούν, να προβάλλουν, να επεξεργάζονται και να διαγράφουν μαθήματα, ανάλογα με τον ρόλο και τη σχέση τους με κάθε μάθημα (δημιουργός ή συμμετέχων).

Το backend υλοποιεί την πλήρη λειτουργικότητα CRUD (Create, Read, Update, Delete) μέσω RESTful endpoints, ενώ κάθε ενέργεια συνοδεύεται από κατάλληλους ελέγχους ταυτότητας και δικαιωμάτων.

- Οι λειτουργίες διαχωρίζονται σε τέσσερις κύριες κατηγορίες:
- Δημιουργία νέου μαθήματος από τον χρήστη-διδάσκοντα
- Προβολή μαθημάτων από όλους τους χρήστες, με διαφορετικό επίπεδο λεπτομέρειας
- Επεξεργασία μαθημάτων από τον δημιουργό τους
- Διαγραφή μαθημάτων με ενημέρωση όλων των συμμετεχόντων

Στις επόμενες υποενότητες περιγράφονται αναλυτικά οι τεχνικές λεπτομέρειες για κάθε ενέργεια, περιλαμβάνοντας τα αντίστοιχα endpoints, τα DTOs εισόδου/εξόδου και τη ροή στο backend.

4.4.4.1 Δημιουργία Μαθήματος

Η δημιουργία νέου μαθήματος πραγματοποιείται από χρήστες που επιθυμούν να λειτουργήσουν ως διδάσκοντες στην πλατφόρμα. Η λειτουργία υλοποιείται μέσω POST αιτήματος στο εξής endpoint:

```
POST /post
```

Το αίτημα αποστέλλεται από το frontend και περιλαμβάνει στο σώμα του τα στοιχεία του μαθήματος σε μορφή JSON, καθώς και το access token στο πεδίο Authorization του header. Τα δεδομένα του μαθήματος αποτυπώνονται στο αντικείμενο τύπου PostCreateDto:

```
public class PostCreateDto {
    private String title;
    private String previewDescription;
    private String description;
    private Integer maxEnrolls;
```

```
    private Boolean isPersonal;
    private String place;
}
```

Το αίτημα παραλαμβάνεται στον controller και προωθείται στην υπηρεσία PostService, αφού πρώτα εξαχθεί το username από το JWT:

```
@PostMapping()
public ResponseEntity<UUID> create(
        @RequestBody PostCreateDto createDto,
        @RequestHeader(HttpHeaders.AUTHORIZATION) String auth
) {
    String username = jwtUtils.extractUsername(auth.substring(7));
    return postService.create(createDto, username)
            .map(ResponseEntity::ok)
            .get();
}
```

Η μέθοδος create() στο service εκτελεί τη βασική λογική δημιουργίας του μαθήματος. Συγκεκριμένα:

- αναζητά τον χρήστη με βάση το username
- ελέγχει αν έχει ήδη δημιουργήσει μάθημα με ίδιο τίτλο
- δημιουργεί αντικείμενο Post με όλα τα πεδία από το DTO
- θέτει ημερομηνίες δημιουργίας και ενημέρωσης
- αποθηκεύει το νέο μάθημα στη βάση δεδομένων
- αποστέλλει email επιβεβαίωσης στον δημιουργό

Το σχετικό απόσπασμα κώδικα έχει ως εξής:

```
public Try<UUID> create(PostCreateDto createDto, String username) {
    return Try.of(() -> Option.ofOptional(userRepository.findByUsername(username))
            .getOrElseThrow(() -> new IllegalArgumentException("There is no user
with username: " + username)))
        .filter(user -> user.getPostsOwnedByMe().stream()
            .noneMatch(post -> Objects.equals(post.getTitle(),
createDto.getTitle())),
            () -> new IllegalArgumentException("The title is already used by this
user."))
        .map(user -> {
            Post post = new Post();
            post.setId(UUID.randomUUID());
```

```
            post.setTitle(createDto.getTitle());
            post.setPreviewDescription(createDto.getPreviewDescription());
            post.setDescription(createDto.getDescription());
            post.setMaxEnrolls(createDto.getMaxEnrolls());
            post.setIsPersonal(createDto.getIsPersonal());
            post.setPlace(createDto.getPlace());
            post.setCreatedBy(user);
            post.setCreatedAt(OffsetDateTime.now());
            post.setUpdatedAt(OffsetDateTime.now());
            return postRepository.saveAndFlush(post);
        })
        .map(post -> {
            emailService.sendEmail(...); // αποστολή επιβεβαίωσης δημιουργίας
μαθήματος
            return post.getId();
        });
}
```

Το αποτέλεσμα της λειτουργίας είναι η επιστροφή του UUID του νεοδημιουργηθέντος μαθήματος στο frontend, το οποίο μπορεί να χρησιμοποιηθεί για πλοήγηση ή ενημέρωση της κατάστασης του χρήστη.

Η υλοποίηση ακολουθεί τις αρχές REST και τηρεί ελέγχους μοναδικότητας για τον τίτλο εντός του ίδιου δημιουργού, καθώς και ασφάλειας μέσω ταυτοποίησης JWT. Η άμεση αποστολή email επιβεβαίωσης βελτιώνει την εμπειρία χρήστη και την αξιοπιστία του συστήματος.

4.4.4.2 Προβολή Μαθημάτων

Η προβολή μαθημάτων αποτελεί βασική λειτουργία της εφαρμογής, καθώς επιτρέπει στους χρήστες να ενημερώνονται για τα διαθέσιμα μαθήματα, να αναζητούν όσα τους ενδιαφέρουν και να διαχειρίζονται όσα έχουν δημιουργήσει ή παρακολουθούν. Η εφαρμογή υλοποιεί διακριτά endpoints για:

- την παρουσίαση όλων των μαθημάτων (feed),
- την εμφάνιση των μαθημάτων που έχει δημιουργήσει ο χρήστης (my courses),
- την προβολή των μαθημάτων στα οποία έχει εγγραφεί (enrolled),
- και την αναλυτική προβολή κάθε μαθήματος (detailed view).

Κάθε προβολή παρέχει συγκεκριμένα δεδομένα ανάλογα με τον ρόλο και τη σχέση του χρήστη με το μάθημα. Τα δεδομένα εξυπηρετούνται μέσω RESTful endpoints με υποστήριξη σελιδοποίησης, αναζήτησης και ταξινόμησης.

4.4.4.2.1

Η αρχική προβολή μαθημάτων (feed) επιτρέπει σε κάθε χρήστη να περιηγηθεί σε όλα τα διαθέσιμα μαθήματα που έχουν δημιουργηθεί από άλλους χρήστες. Η λειτουργία αυτή αποτελεί τον βασικό τρόπο ανακάλυψης μαθημάτων και υποστηρίζει αναζήτηση, ταξινόμηση και σελιδοποίηση.

Το αίτημα πραγματοποιείται μέσω HTTP GET προς το εξής endpoint:

```
GET /post/feed
```

Το endpoint δέχεται τις παρακάτω παραμέτρους:

- search (προαιρετικό): λέξη-κλειδί που χρησιμοποιείται για φιλτράρισμα με βάση τον τίτλο
- page: αριθμός σελίδας (προεπιλογή 0)
- size: μέγεθος σελίδας (προεπιλογή 10)
- sort: πεδίο ταξινόμησης (π.χ. updatedAt)
- Authorization header: περιέχει το JWT για εξαγωγή του username

Η υλοποίηση στον controller είναι η εξής:

```
@GetMapping("feed")
public ResponseEntity<PostResponseContainerDto> getPosts(
        @RequestParam(name = "search", required = false, defaultValue = "") String
search,
        @RequestParam(name = "page", required = false, defaultValue = "0") int
page,
        @RequestParam(name = "size", required = false, defaultValue = "10") int
size,
        @RequestParam(name = "sort", required = false, defaultValue = "updatedAt")
String sort,
        @RequestHeader(HttpHeaders.AUTHORIZATION) String auth
) {
    String username = jwtUtils.extractUsername(auth.substring(7));
    return postService.findByTitle(search, page, size, sort, username)
            .map(postPage -> new PostResponseContainerDto(
                    postPage.stream()
                            .map(PostMapper::postToPostResponseDto)
                            .collect(Collectors.toList()),
                    postPage.getTotalElements()))
            .map(ResponseEntity::ok)
            .getOrElseThrow(ex -> new RuntimeException("Something went wrong while
fetching posts.", ex));
}
```

Το service καλεί τη μέθοδο findByTitle(...), η οποία εκτελεί αναζήτηση με βάση τον τίτλο, ενώ ταυτόχρονα εξαιρεί τα μαθήματα που ανήκουν στον ίδιο τον χρήστη:

```
public Try<Page<Post>> findByTitle(String title, Integer page, Integer size, String
sort, String username) {
    return Try.of(() -> postRepository.findByTitle(
            title.toLowerCase(),
            username,
            PageRequest.of(page, size, Sort.by(sort).descending())));
}
```

Η απάντηση που επιστρέφεται στον client περιλαμβάνει μια λίστα από αντικείμενα τύπου PostResponseDto και τον συνολικό αριθμό αποτελεσμάτων. Το συνολικό αντικείμενο απάντησης είναι τύπου PostResponseContainerDto:

```
public class PostResponseContainerDto {
    private List<PostResponseDto> content;
    private Long total;
}
```

Κάθε PostResponseDto περιέχει τα βασικά χαρακτηριστικά ενός μαθήματος:

```
public class PostResponseDto {
    private UUID id;
    private String title;
    private String previewDescription;
    private Integer maxEnrolls;
    private Boolean isPersonal;
    private String place;
    private UserPostResponseDto userOwner;
    private Integer enrollments;
}
```

Το πεδίο userOwner είναι τύπου UserPostResponseDto και περιέχει τα στοιχεία του δημιουργού του μαθήματος:

```
public class UserPostResponseDto {
    private UUID id;
    private String username;
    private String firstname;
    private String lastname;
    private String email;
    private String phone;
}
```

Η προβολή feed δίνει στον χρήστη τη δυνατότητα να ενημερωθεί για όλα τα διαθέσιμα μαθήματα, να τα φιλτράρει κατά τίτλο, να ταξινομήσει τα αποτελέσματα και να περιηγηθεί εύκολα με χρήση σελιδοποίησης. Αποτελεί βασικό μηχανισμό εισόδου στη λειτουργία εγγραφής και γενικότερα στην αλληλεπίδραση με την πλατφόρμα.

4.4.4.2.2 Μαθήματα που Έχω Δημιουργήσει (My Courses)

Η εφαρμογή παρέχει στον χρήστη τη δυνατότητα να προβάλει συγκεντρωτικά όλα τα μαθήματα που έχει δημιουργήσει ο ίδιος. Η λειτουργία αυτή επιτρέπει στον διδάσκοντα να έχει πλήρη εικόνα της διδασκαλίας του, καθώς και να μεταβεί εύκολα στη διαχείριση, επεξεργασία ή διαγραφή κάθε μαθήματος.

Η πρόσβαση στα δεδομένα πραγματοποιείται μέσω HTTP GET αιτήματος στο εξής endpoint:

```
GET /post/myposts
```

Το αίτημα υποστηρίζει παραμέτρους σελιδοποίησης και ταξινόμησης:

- page: αριθμός σελίδας (προεπιλογή 0)
- size: μέγεθος σελίδας (προεπιλογή 10)
- sort: πεδίο ταξινόμησης (π.χ. updatedAt)
- Authorization header: περιέχει το JWT του χρήστη

Η αντίστοιχη μέθοδος στον controller:

```
@GetMapping("myposts")
public ResponseEntity<PostResponseContainerDto> getOwnersPosts(
        @RequestParam(name = "page", required = false, defaultValue = "0") int
page,
        @RequestParam(name = "size", required = false, defaultValue = "10") int
size,
        @RequestParam(name = "sort", required = false, defaultValue = "updatedAt")
String sort,
        @RequestHeader(HttpHeaders.AUTHORIZATION) String auth
) {
    String username = jwtUtils.extractUsername(auth.substring(7));
    return postService.findByUsername(page, size, sort, username)
            .map(posts -> new PostResponseContainerDto(
                    posts.stream()
                         .map(PostMapper::postToPostResponseDto)
```

```
                        .collect(Collectors.toList()),
                posts.getTotalElements()))
        .map(ResponseEntity::ok)
        .get();
}
```

Η μέθοδος findByUsername(...) αναζητά στη βάση δεδομένων όλα τα μαθήματα που έχουν δημιουργηθεί από τον συγκεκριμένο χρήστη, με χρήση σελιδοποίησης:

```
public Try<Page<Post>> findByUsername(int page, int size, String sort, String
username) {
    return Try.of(() -> postRepository.findByUsername(
            username,
            PageRequest.of(page, size, Sort.by(sort).ascending())));
}
```

Για κάθε μάθημα που ανήκει στον χρήστη, επιστρέφεται ένα αντικείμενο PostResponseOwnerDto, το οποίο περιλαμβάνει αναλυτικά στοιχεία, μεταξύ των οποίων:

```
public class PostResponseOwnerDto {
    private UUID id;
    private String title;
    private String previewDescription;
    private String description;
    private Integer enrollments;
    private Integer maxEnrolls;
    private Boolean isPersonal;
    private String place;
    private List<UserPostResponseDto> enrolledStudents;
}
```

Σε αντίθεση με την προβολή feed, εδώ παρέχεται επιπλέον η λίστα των εγγεγραμμένων φοιτητών (enrolledStudents) με τα πλήρη στοιχεία τους, μέσω του αντικειμένου UserPostResponseDto:

```
public class UserPostResponseDto {
    private UUID id;
    private String username;
    private String firstname;
    private String lastname;
    private String email;
    private String phone;
}
```

Η προβολή αυτή ενδείκνυται για την πλήρη εποπτεία και διαχείριση των μαθημάτων που έχει δημιουργήσει ο χρήστης. Παρέχει τις απαραίτητες πληροφορίες για να μεταβεί σε επεξεργασία, να δει τη σύνθεση των φοιτητών ή να καταργήσει κάποιο μάθημα αν χρειαστεί.

4.4.4.2.3 Μαθήματα στα Οποία Συμμετέχω (Enrolled)

Η εφαρμογή δίνει τη δυνατότητα στους χρήστες να προβάλλουν όλα τα μαθήματα στα οποία είναι εγγεγραμμένοι ως συμμετέχοντες. Η προβολή αυτή επιτρέπει στον φοιτητή να έχει σαφή εικόνα των μαθημάτων που παρακολουθεί, να τα εντοπίζει γρήγορα και να διαχειρίζεται τη συμμετοχή του.

Το σχετικό αίτημα πραγματοποιείται μέσω HTTP GET προς το endpoint:

```
GET /post/enrolledposts
```

Όπως και στις υπόλοιπες προβολές, υποστηρίζονται παράμετροι σελιδοποίησης και ταξινόμησης:

- page: αριθμός σελίδας
- size: μέγεθος σελίδας
- sort: πεδίο ταξινόμησης
- Authorization header: περιέχει το JWT του χρήστη

Η μέθοδος στον controller:

```
@GetMapping("enrolledposts")
public ResponseEntity<PostResponseContainerDto> getEnrolledPosts(
        @RequestParam(name = "page", required = false, defaultValue = "0") int
page,
        @RequestParam(name = "size", required = false, defaultValue = "10") int
size,
        @RequestParam(name = "sort", required = false, defaultValue = "updatedAt")
String sort,
        @RequestHeader(HttpHeaders.AUTHORIZATION) String auth
) {
    String username = jwtUtils.extractUsername(auth.substring(7));
    return postService.findEnrolledByUsername(page, size, sort, username)
            .map(ResponseEntity::ok)
            .get();
}
```

Το service χρησιμοποιεί τη μέθοδο findEnrolledByUsername(...), η οποία αναζητά στη βάση δεδομένων όλα τα μαθήματα στα οποία συμμετέχει ο χρήστης ως φοιτητής:

```
public Try<PostResponseContainerDto> findEnrolledByUsername(int page, int size,
String sort, String username) {
    return Try.of(() -> postRepository.findEnrolledPostsByUsername(
            username,
            PageRequest.of(page, size, Sort.by(sort).ascending())))
        .map(posts -> new PostResponseContainerDto(
                posts.stream().map(PostMapper::postToPostResponseDto).toList(),
                posts.getTotalElements()));
}
```

Η απάντηση που επιστρέφεται περιλαμβάνει αντικείμενο τύπου PostResponseContainerDto, το οποίο περιέχει:

- λίστα με αντικείμενα PostResponseDto
- συνολικό πλήθος εγγραφών για σκοπούς σελιδοποίησης

Τα δεδομένα για κάθε μάθημα είναι ίδια με την προβολή feed και περιλαμβάνουν τίτλο, περιγραφή, τοποθεσία, τύπο μαθήματος, αριθμό εγγραφών και στοιχεία του δημιουργού.

Η συγκεκριμένη προβολή αποτελεί βασικό εργαλείο παρακολούθησης για κάθε φοιτητή, καθώς του παρέχει κεντρική πρόσβαση σε όλα τα ενεργά μαθήματα στα οποία έχει εγγραφεί.

4.4.4.2.4 Αναλυτική Προβολή Μαθήματος (Detailed View)

Η εφαρμογή προσφέρει σε κάθε χρήστη τη δυνατότητα αναλυτικής προβολής ενός συγκεκριμένου μαθήματος. Η προβολή αυτή ενεργοποιείται όταν ο χρήστης επιλέξει κάποιο μάθημα από το feed ή από τη λίστα των μαθημάτων του και του επιτρέπει να δει αναλυτικά στοιχεία όπως περιγραφή, δημιουργό, πλήθος εγγεγραμμένων φοιτητών και διαθεσιμότητα.

Το σχετικό αίτημα πραγματοποιείται μέσω HTTP GET προς το εξής endpoint:

```
GET /post/{postId}
```

Το postId αντιστοιχεί στο αναγνωριστικό του μαθήματος που επιλέγει ο χρήστης. Το αίτημα απαιτεί header εξουσιοδότησης (Authorization) με το JWT του χρήστη. Η μέθοδος στον controller:

```
@GetMapping("{postId}")
public ResponseEntity<PostDetailsResponseDto> getPostById(
        @RequestHeader(HttpHeaders.AUTHORIZATION) String auth,
        @PathVariable("postId") UUID postId
) {
```

```
    String username = jwtUtils.extractUsername(auth.substring(7));
    return postService.getPostById(username, postId)
            .map(ResponseEntity::ok)
            .get();
}
```

Η μέθοδος getPostById(...) στην υπηρεσία χρησιμοποιεί το όνομα χρήστη ώστε να προσαρμόσει την απόκριση ανάλογα με το αν ο χρήστης:

- είναι ο δημιουργός του μαθήματος,
- είναι εγγεγραμμένος στο μάθημα,
- ή είναι επισκέπτης.

Η απάντηση επιστρέφεται σε μορφή PostDetailsResponseDto, το οποίο περιλαμβάνει τα παρακάτω πεδία:

```
public class PostDetailsResponseDto {
    private UUID id;
    private String title;
    private String previewDescription;
    private String Description;
    private Integer maxEnrolls;
    private Boolean isPersonal;
    private String place;
    private UserPostResponseDto userOwner;
    private Integer enrollments;
    private boolean isEnrolled;
    private boolean isOwner;
}
```

Το πεδίο userOwner περιέχει τις βασικές πληροφορίες του δημιουργού, ενώ τα πεδία isEnrolled και isOwner καθορίζουν τη σχέση του χρήστη με το μάθημα. Αυτές οι τιμές χρησιμοποιούνται στο frontend για να προσαρμόζεται η διεπαφή χρήστη, εμφανίζοντας ανάλογες επιλογές (π.χ. εγγραφή, αποεγγραφή, διαχείριση).

Η προβολή αυτή αποτελεί κεντρικό σημείο από το οποίο ο χρήστης μπορεί να αποφασίσει αν θέλει να εγγραφεί σε ένα μάθημα ή να διαχειριστεί ένα μάθημα που έχει δημιουργήσει. Αποτελεί επίσης τη βάση για τις επόμενες λειτουργίες (εγγραφή, επεξεργασία, διαγραφή), οι οποίες αναλύονται στις επόμενες ενότητες.

4.4.4.3 Διαχείριση Συμμετοχής σε Μάθημα

Η εφαρμογή υποστηρίζει πλήρη μηχανισμό διαχείρισης συμμετοχής φοιτητών σε μαθήματα, τόσο από την πλευρά του απλού χρήστη όσο και από την πλευρά του δημιουργού κάθε μαθήματος. Συγκεκριμένα, υποστηρίζονται:

- Εγγραφή ενός χρήστη σε μάθημα (από τον ίδιο)
- Αποχώρηση από μάθημα (disenroll, επίσης από τον ίδιο)
- Αφαίρεση εγγεγραμμένου φοιτητή από τον δημιουργό του μαθήματος

Κάθε μία από αυτές τις ενέργειες υλοποιείται μέσω διαφορετικού endpoint, με τους αντίστοιχους ελέγχους ταυτοποίησης και πρόσβασης. Επιπλέον, όλες οι αλλαγές συνοδεύονται από αποστολή email ενημέρωσης στους εμπλεκόμενους χρήστες.

4.4.4.3.1 Εγγραφή Χρήστη σε Μάθημα

Η εγγραφή σε μάθημα αποτελεί τη βασική ενέργεια μέσω της οποίας ένας χρήστης αποκτά συμμετοχή σε μάθημα που προσφέρεται από άλλον φοιτητή. Η λειτουργία υλοποιείται μέσω HTTP POST αιτήματος προς το εξής endpoint:

```
POST /post/enroll/{postId}
```

Το postId είναι το μοναδικό αναγνωριστικό του μαθήματος. Ο χρήστης πρέπει να είναι αυθεντικοποιημένος, καθώς το JWT εξάγεται από το Authorization header για την αναγνώριση του username.

Η μέθοδος στον controller έχει ως εξής:

```
@PostMapping("enroll/{postId}")
public ResponseEntity<PostDetailsResponseDto> enroll(
        @RequestHeader(HttpHeaders.AUTHORIZATION) String auth,
        @PathVariable("postId") UUID postId
) {
    String username = jwtUtils.extractUsername(auth.substring(7));
    return postService.enroll(username, postId)
            .map(ResponseEntity::ok)
            .get();
}
```

Η μέθοδος enroll(...) στο service εκτελεί τους απαραίτητους ελέγχους και ενημερώσεις:

- Εύρεση του μαθήματος με βάση το postId

- Ανάκτηση του χρήστη από το username
- Έλεγχος ότι:
    - ο χρήστης δεν είναι ο δημιουργός του μαθήματος
    - ο χρήστης δεν είναι ήδη εγγεγραμμένος
    - το μάθημα έχει διαθέσιμες θέσεις
- Εγγραφή του χρήστη στο μάθημα
- Αποθήκευση της νέας κατάστασης στη βάση
- Αποστολή email στον χρήστη και στον δημιουργό

Η μέθοδος είναι επισημασμένη με @Transactional, ώστε όλες οι παραπάνω ενέργειες (ανάκτηση δεδομένων, τροποποίηση λίστας εγγεγραμμένων, αποθήκευση, αποστολή email) να εκτελούνται ως ενιαία μονάδα συναλλαγής. Αν οποιοδήποτε βήμα αποτύχει, η αλλαγή ακυρώνεται, διασφαλίζοντας τη συνοχή της βάσης δεδομένων.

Η υλοποίηση έχει ως εξής:

```
@Transactional
public Try<PostDetailsResponseDto> enroll(final String username, final UUID postId)
{
    return Try.of(() -> Option.ofOptional(postRepository.findById(postId))
            .getOrElseThrow(() -> new PostNotFoundException("There is no post with
id: " + postId)))
        .flatMap(post -> Try.of(() ->
Option.ofOptional(userRepository.findByUsername(username))
                .getOrElseThrow(() -> new IllegalArgumentException("There is no
user with username: " + username)))
            .flatMap(user -> {
                io.vavr.collection.List<User> enrolledUsers =
io.vavr.collection.List.ofAll(post.getEnrolledUsers());

                return Try.of(() -> post)
                    .filter(p -> p.getMaxEnrolls() > enrolledUsers.size(),
                            () -> new IllegalStateException("Cannot enroll: The
lesson is full."))
                    .filter(p -> !p.getCreatedBy().equals(user),
                            () -> new IllegalArgumentException("You cannot enroll
if you are the owner of the post."))
                    .filter(p -> !enrolledUsers.contains(user),
                            () -> new IllegalArgumentException("User already
enrolled."))
                    .map(p -> {
                        io.vavr.collection.List<User> updatedUsers =
enrolledUsers.append(user);
                        p.setEnrolledUsers(new ArrayList<>(updatedUsers.asJava()));
```

```
                        return postRepository.saveAndFlush(p);
                    })
                    .map(post2 -> {
                        emailService.sendEmail(...); // email στον χρήστη
                        emailService.sendEmail(...); // email στον δημιουργό
                        return post2;
                    });
            }))
        .map(post -> PostMapper.postToPostDetailsResponseDto(post, username))
        .onFailure(Throwable::printStackTrace);
}
```

Η απόκριση επιστρέφεται σε μορφή PostDetailsResponseDto, το οποίο περιέχει τα ενημερωμένα δεδομένα του μαθήματος, μεταξύ των οποίων:

- τον συνολικό αριθμό συμμετεχόντων
- το αν ο χρήστης είναι πλέον εγγεγραμμένος
- το αν είναι ο δημιουργός

Η λειτουργία εγγραφής περιλαμβάνει πλήρεις ελέγχους ώστε να διασφαλίζεται η ακεραιότητα των δεδομένων και να αποφεύγονται περιπτώσεις διπλοεγγραφής ή υπέρβασης της διαθεσιμότητας. Η αποστολή ειδοποιήσεων μέσω email ενισχύει τη διαφάνεια και κρατά όλους τους εμπλεκόμενους ενημερωμένους.

4.4.4.3.2 Αποεγγραφή Χρήστη από Μάθημα

Η εφαρμογή επιτρέπει στους χρήστες να αποχωρούν από μαθήματα στα οποία είναι εγγεγραμμένοι, μέσω λειτουργίας αποεγγραφής. Η ενέργεια αυτή εξασφαλίζει την αυτονομία του χρήστη και προσφέρει καθαρή έξοδο από μαθήματα που δεν τον ενδιαφέρουν πλέον.

Η αποεγγραφή πραγματοποιείται μέσω HTTP DELETE αιτήματος προς το εξής endpoint:

```
DELETE /post/disenroll/{postId}
```

Το postId αντιστοιχεί στο μάθημα από το οποίο θέλει να αποχωρήσει ο χρήστης. Η ταυτοποίηση γίνεται μέσω JWT token που παρέχεται στο Authorization header.

Η μέθοδος στον controller είναι:

```
@DeleteMapping("disenroll/{postId}")
public ResponseEntity<PostDetailsResponseDto> disenroll(
        @RequestHeader(HttpHeaders.AUTHORIZATION) String auth,
        @PathVariable("postId") UUID postId
) {
```

```
    String username = jwtUtils.extractUsername(auth.substring(7));
    return postService.disenroll(username, postId)
            .map(ResponseEntity::ok)
            .get();
}
```

Η λογική της αποεγγραφής στο PostService περιλαμβάνει τα εξής βήματα:

- Εντοπισμός του μαθήματος με βάση το postId
- Ανάκτηση του χρήστη από τη βάση με βάση το username
- Έλεγχος ότι ο χρήστης είναι πράγματι εγγεγραμμένος στο μάθημα
- Αφαίρεση του χρήστη από τη λίστα εγγεγραμμένων
- Αποθήκευση των αλλαγών
- Αποστολή ενημερωτικού email τόσο στον χρήστη όσο και στον δημιουργό

Η μέθοδος είναι επισημασμένη με @Transactional, ώστε όλες οι ενέργειες να εκτελούνται ατομικά ως συναλλαγή:

```
@Transactional
public Try<PostDetailsResponseDto> disenroll(final String username, final UUID
postId) {
    return Try.of(() -> postRepository.findById(postId))
        .flatMap(postOpt -> Option.ofOptional(postOpt)
            .toTry(() -> new IllegalArgumentException("There is no post with id: "
+ postId)))
        .flatMap(post -> Try.of(() -> userRepository.findByUsername(username))
            .flatMap(userOpt -> Option.ofOptional(userOpt)
                .toTry(() -> new IllegalArgumentException("There is no user with
username: " + username)))
            .flatMap(user -> {
                List<User> enrolledUsers = new
ArrayList<>(post.getEnrolledUsers());

                return Try.success(post)
                    .filter(p -> enrolledUsers.contains(user),
                            () -> new IllegalArgumentException("User is not
enrolled in post: " + postId))
                    .map(p -> {
                        enrolledUsers.remove(user);
                        p.setEnrolledUsers(enrolledUsers);
                        return postRepository.saveAndFlush(p);
                    })
                    .map(post2 -> {
                        emailService.sendEmail(...); // email στον χρήστη
                        emailService.sendEmail(...); // email στον δημιουργό
```

```
                        return post2;
                    });
            }))
        .map(post -> PostMapper.postToPostDetailsResponseDto(post, username));
}
```

Το αποτέλεσμα της αποεγγραφής είναι η ενημέρωση της λίστας συμμετεχόντων στο backend και η επιστροφή του αντικειμένου PostDetailsResponseDto με τις ενημερωμένες πληροφορίες για το μάθημα.

Η ενέργεια αποεγγραφής προσφέρει στους χρήστες την ευελιξία να διαχειριστούν τη συμμετοχή τους ανάλογα με τις ανάγκες τους, χωρίς τη μεσολάβηση τρίτων. Παράλληλα, η αποστολή ενημερώσεων διατηρεί και τον δημιουργό ενήμερο για τις αλλαγές που αφορούν το μάθημά του.

4.4.4.3.3 Απομάκρυνση Φοιτητή από Δημιουργό Μαθήματος

Η εφαρμογή δίνει τη δυνατότητα στον δημιουργό ενός μαθήματος να αφαιρέσει εγγεγραμμένο φοιτητή από το μάθημά του. Η ενέργεια αυτή μπορεί να χρησιμοποιηθεί σε περιπτώσεις ακύρωσης συμμετοχής, παραβίασης κανόνων ή γενικότερης διαχείρισης της ομάδας των φοιτητών.

Η απομάκρυνση υλοποιείται με HTTP DELETE αίτημα προς το εξής endpoint:

```
DELETE /post/disenroll/{userId}/{postId}
```

Το userId είναι το αναγνωριστικό του φοιτητή που πρόκειται να απομακρυνθεί και το postId είναι το αναγνωριστικό του μαθήματος. Η ταυτοποίηση του δημιουργού γίνεται μέσω του JWT που περιέχεται στο Authorization header.

Η μέθοδος στον controller:

```
@DeleteMapping("disenroll/{userId}/{postId}")
public ResponseEntity<Void> deleteEnrolledStudent(
        @RequestHeader(HttpHeaders.AUTHORIZATION) String auth,
        @PathVariable("postId") UUID postId,
        @PathVariable("userId") UUID userId
) {
    String username = jwtUtils.extractUsername(auth.substring(7));
    return postService.deleteEnrolledStudent(username, postId, userId)
            .map(ResponseEntity::ok)
            .get();
}
```

Η μέθοδος deleteEnrolledStudent(...) στο PostService εκτελεί τα εξής βήματα:

- Επιβεβαιώνει ότι ο χρήστης που εκτελεί την ενέργεια είναι ο **δημιουργός** του μαθήματος

- Αναζητά το μάθημα και τον φοιτητή στη βάση
- Αφαιρεί τον φοιτητή από τη λίστα των εγγεγραμμένων
- Αποθηκεύει την τροποποιημένη κατάσταση
- Αποστέλλει email τόσο στον φοιτητή όσο και στον δημιουργό

Η υλοποίηση είναι επισημασμένη με @Transactional ώστε να διασφαλίζεται ότι η διαδικασία ολοκληρώνεται ατομικά:

```
@Transactional
public Try<Void> deleteEnrolledStudent(String username, UUID postId, UUID userId) {
    return Try.of(() -> Option.ofOptional(postRepository.findById(postId))
            .getOrElseThrow(() -> new IllegalArgumentException("There is no post
with id: " + postId)))
        .filter(post -> post.getCreatedBy().getUsername().equals(username),
                () -> new IllegalArgumentException("User is not the owner of the
post."))
        .map(post -> {
            post.setEnrolledUsers(post.getEnrolledUsers().stream()
                    .filter(user -> !user.getId().equals(userId))
                    .collect(Collectors.toList()));
            return postRepository.saveAndFlush(post);
        })
        .map(post -> Tuple.of(
                Option.ofOptional(userRepository.findById(userId))
                        .getOrElseThrow(() -> new IllegalArgumentException("There
is no user with id: " + userId)),
                post))
        .map(tuple -> {
            User user = tuple._1;
            Post post = tuple._2;

            emailService.sendEmail(...); // email στον φοιτητή
            emailService.sendEmail(...); // email στον δημιουργό

            return null;
        });
}
```

Η ενέργεια απομάκρυνσης προσφέρει στον διδάσκοντα απόλυτο έλεγχο για τη διαχείριση των συμμετεχόντων στο μάθημα, εξασφαλίζοντας ότι η δυναμική του τμήματος διατηρείται λειτουργική και συνεπής με τους στόχους του μαθήματος. Ταυτόχρονα, οι ειδοποιήσεις μέσω email διατηρούν τη διαφάνεια και ενημερώνουν όλα τα εμπλεκόμενα μέρη με σαφήνεια.

4.4.4.4 Επεξεργασία Μαθήματος

Η εφαρμογή δίνει τη δυνατότητα στον δημιουργό ενός μαθήματος να επεξεργαστεί συγκεκριμένα πεδία του μαθήματος, διατηρώντας όμως τον τίτλο σταθερό. Η δυνατότητα αυτή παρέχει ευελιξία και υποστηρίζει σενάρια αλλαγής χώρου, περιγραφής ή πλήθους εγγραφών.

Η επεξεργασία πραγματοποιείται μέσω HTTP PUT αιτήματος προς το εξής endpoint:

```
PUT /post
```

Το αίτημα απαιτεί το JWT του χρήστη στο Authorization header και περιλαμβάνει τα νέα στοιχεία του μαθήματος στο σώμα του, με χρήση του DTO PostUpdateDto:

```
public class PostUpdateDto {
    private UUID id;
    private String previewDescription;
    private String description;
    private Integer maxEnrolls;
    private Boolean isPersonal;
    private String place;
}
```

Όπως φαίνεται, το DTO περιλαμβάνει μόνο τα πεδία που είναι τροποποιήσιμα. Ο τίτλος (title) δεν περιλαμβάνεται, διασφαλίζοντας ότι η ταυτότητα του μαθήματος παραμένει σταθερή.

Η μέθοδος στον controller:

```
@PutMapping
public ResponseEntity<PostResponseOwnerDto> update(
        @RequestBody PostUpdateDto updateDto,
        @RequestHeader(HttpHeaders.AUTHORIZATION) String auth
) {
    String username = jwtUtils.extractUsername(auth.substring(7));
    return postService.update(updateDto, username)
            .map(ResponseEntity::ok)
            .get();
}
```

Στο backend, η μέθοδος update(...) στο PostService εκτελεί:

- Ανάκτηση του μαθήματος από τη βάση μέσω του id
- Επιβεβαίωση ότι ο αιτών είναι ο δημιουργός

- Ενημέρωση των πεδίων περιγραφής, τοποθεσίας, τύπου μαθήματος και μέγιστου αριθμού εγγραφών
- Ανανέωση του timestamp updatedAt
- Αποθήκευση των αλλαγών
- Αποστολή email:
    - στον δημιουργό (επιβεβαίωση αλλαγής)
    - σε όλους τους εγγεγραμμένους φοιτητές (ειδοποίηση για την αλλαγή)

Η μέθοδος είναι επισημασμένη με @Transactional, διασφαλίζοντας ότι όλες οι αλλαγές και ειδοποιήσεις εκτελούνται ενιαία:

```
@Transactional
public Try<PostResponseOwnerDto> update(PostUpdateDto updateDto, String username) {
    return Try.of(() ->
Option.ofOptional(postRepository.findById(updateDto.getId()))
            .getOrElseThrow(() -> new IllegalArgumentException("There is no post
with id: " + updateDto.getId())))
        .filter(post -> Objects.equals(post.getCreatedBy().getUsername(),
username),
                () -> new PostOwnershipException("User is not the owner of the
post."))
        .map(post -> {
            post.setPreviewDescription(updateDto.getPreviewDescription());
            post.setDescription(updateDto.getDescription());
            post.setMaxEnrolls(updateDto.getMaxEnrolls());
            post.setIsPersonal(updateDto.getIsPersonal());
            post.setPlace(updateDto.getPlace());
            post.setUpdatedAt(OffsetDateTime.now());
            return postRepository.saveAndFlush(post);
        })
        .map(post -> {
            post.getEnrolledUsers().forEach(user -> {
                emailService.sendEmail(...); // ειδοποίηση προς φοιτητές
            });

            emailService.sendEmail(...); // επιβεβαίωση προς δημιουργό
            return post;
        })
        .map(PostMapper::postToPostResponseOwnerDto)
        .onFailure(Throwable::printStackTrace);
}
```

Η απόκριση επιστρέφεται ως PostResponseOwnerDto, το οποίο περιλαμβάνει τα επικαιροποιημένα στοιχεία του μαθήματος και τη λίστα εγγεγραμμένων φοιτητών.

Η δυνατότητα επεξεργασίας επιτρέπει στον διδάσκοντα να προσαρμόσει το μάθημα ανάλογα με τις εξελίξεις ή τις ανάγκες του τμήματος. Οι ειδοποιήσεις εξασφαλίζουν ότι όλοι οι εμπλεκόμενοι χρήστες παραμένουν πλήρως ενημερωμένοι για οποιαδήποτε αλλαγή.

4.4.4.5 Διαχείριση Προφίλ Χρήστη

Η εφαρμογή προσφέρει στον χρήστη τη δυνατότητα να διαχειριστεί το προφίλ του με τρόπο ασφαλή, άμεσο και πλήρως ελεγχόμενο. Οι βασικές λειτουργίες περιλαμβάνουν την τροποποίηση προσωπικών στοιχείων, την αλλαγή του username και του email, καθώς και τη δυνατότητα οριστικής διαγραφής του λογαριασμού.

Κάθε μία από αυτές τις ενέργειες υλοποιείται με ξεχωριστό endpoint, συνοδεύεται από τους απαραίτητους ελέγχους ταυτοποίησης μέσω JWT και εφαρμόζει κανόνες εγκυρότητας, όπως έλεγχο μοναδικότητας για username και email.

Η δομή της λειτουργικότητας διαχείρισης προφίλ καλύπτει:

- Την **αλλαγή προσωπικών στοιχείων** (όνομα, επώνυμο, τηλέφωνο), χωρίς επίδραση στην ταυτοποίηση
- Την **τροποποίηση του username**, με πλήρη έλεγχο μοναδικότητας
- Την **τροποποίηση του email**, επίσης με έλεγχο για διπλοεγγραφές
- Τη **διαγραφή του λογαριασμού**, η οποία συνοδεύεται από πλήρη καθαρισμό των σχετικών δεδομένων και αποστολή email επιβεβαίωσης

Οι ενέργειες αυτές ενισχύουν την αυτονομία του χρήστη και τηρούν τις αρχές ασφάλειας και διαφάνειας της πλατφόρμας.

4.4.4.5.1 Αλλαγή Username

Η εφαρμογή επιτρέπει στον χρήστη να αλλάξει το username του λογαριασμού του, εφόσον το νέο όνομα δεν είναι ήδη καταχωρημένο. Η λειτουργία αυτή είναι απαραίτητη για την προσωποποίηση του προφίλ και την ευελιξία διαχείρισης ταυτότητας χρήστη.

Η αλλαγή υλοποιείται με HTTP PUT αίτημα προς το εξής endpoint:

```
PUT /user/username
```

Η ταυτοποίηση του χρήστη γίνεται μέσω JWT που περιέχεται στο Authorization header, ενώ το νέο username αποστέλλεται στο σώμα της αίτησης σε μορφή JSON:

```
{
  "newUsername": "alex42"
}
```

Η μέθοδος στον controller:

```
@PutMapping("username")
public ResponseEntity<UserPostResponseDto> updateUsername(
        @RequestHeader(HttpHeaders.AUTHORIZATION) String auth,
        @RequestBody Map<String, String> body
) {
    String username = jwtUtils.extractUsername(auth.substring(7));
    String newUsername = body.get("newUsername");
    return userService.updateUsername(username, newUsername)
            .map(ResponseEntity::ok)
            .get();
}
```

Η μέθοδος updateUsername(...) στο UserService εκτελεί τα εξής βήματα:

- Αναζητά τον χρήστη με βάση το τρέχον username
- Ελέγχει αν το νέο username είναι ήδη καταχωρημένο σε άλλον χρήστη
- Εφόσον είναι διαθέσιμο:
- ενημερώνει το πεδίο username
- ενημερώνει το updatedAt
- αποθηκεύει τη νέα κατάσταση στη βάση
- Επιστρέφει το νέο προφίλ του χρήστη σε μορφή DTO

Η υλοποίηση είναι η εξής:

```
public Try<UserPostResponseDto> updateUsername(String oldUsername, String
newUsername) {
    return Try.of(() ->
Option.ofOptional(userRepository.findByUsername(oldUsername))
            .getOrElseThrow(() -> new IllegalArgumentException("There is no user
with this username: " + oldUsername)))
```

```
        .flatMap(user ->
Option.ofOptional(userRepository.findByUsername(newUsername))
            .map(existingUser -> Try.<UserPostResponseDto>failure(new
IllegalArgumentException("Username already exists!")))
            .getOrElse(() -> {
                user.setUsername(newUsername);
                user.setUpdatedAt(OffsetDateTime.now());
                return Try.of(() -> userRepository.saveAndFlush(user))
                        .map(UserMapper::usertoUserPostResponseDto);
            }));
}
```

Η ενέργεια αλλαγής username επιτρέπει στον χρήστη να διατηρεί τον λογαριασμό του ενημερωμένο και προσωποποιημένο, ενώ ο έλεγχος μοναδικότητας διασφαλίζει την ακεραιότητα των δεδομένων. Η χρήση της βιβλιοθήκης Try εξασφαλίζει την ασφαλή εκτέλεση της διαδικασίας, με πλήρη χειρισμό των σφαλμάτων.

4.4.4.5.2 Αλλαγή Email

Η εφαρμογή επιτρέπει στον χρήστη να αλλάξει το email που έχει συνδεδεμένο με τον λογαριασμό του. Η αλλαγή αυτή είναι κρίσιμη για τη διατήρηση έγκυρης επικοινωνίας, καθώς το email χρησιμοποιείται για λειτουργίες ειδοποίησης, επαναφοράς κωδικού και επιβεβαιώσεων.

Η αλλαγή υλοποιείται με HTTP PUT αίτημα προς το εξής endpoint:

```
PUT /user/email
```

Το JWT του χρήστη αποστέλλεται μέσω του Authorization header για έλεγχο ταυτότητας. Το νέο email αποστέλλεται στο σώμα της αίτησης σε μορφή JSON:

```
{
  "email": "example@uniboost.academy"
}
```

Η μέθοδος στον controller:

```
@PutMapping("email")
public ResponseEntity<UserPostResponseDto> updateEmail(
        @RequestHeader(HttpHeaders.AUTHORIZATION) String auth,
        @RequestBody Map<String, String> body
) {
    String username = jwtUtils.extractUsername(auth.substring(7));
    String email = body.get("email");
    return userService.updateEmail(username, email)
```

```
            .onFailure(Throwable::printStackTrace)
            .map(ResponseEntity::ok)
            .get();
}
```

Η μέθοδος updateEmail(...) στο UserService εκτελεί τα εξής βήματα:

- Αναζητά τον χρήστη με βάση το username
- Ελέγχει αν το νέο email είναι ήδη καταχωρημένο στη βάση
- Εφόσον είναι διαθέσιμο:
- ενημερώνει το πεδίο email
- ενημερώνει το updatedAt
- αποθηκεύει τη νέα κατάσταση
- Επιστρέφει το νέο προφίλ του χρήστη σε μορφή DTO

Η υλοποίηση της μεθόδου είναι:

```
public Try<UserPostResponseDto> updateEmail(String username, String email) {
    return Try.of(() -> Option.ofOptional(userRepository.findByUsername(username))
            .getOrElseThrow(() -> new IllegalArgumentException("There is no user
with this username: " + username)))
        .flatMap(user -> Option.ofOptional(userRepository.findByEmail(email))
            .map(existingUser -> Try.<UserPostResponseDto>failure(new
IllegalArgumentException("Email already exists!")))
            .getOrElse(() -> {
                user.setEmail(email);
                user.setUpdatedAt(OffsetDateTime.now());
                return Try.of(() -> userRepository.saveAndFlush(user))
                        .map(UserMapper::usertoUserPostResponseDto);
            }))
        .onFailure(Throwable::printStackTrace);
}
```

Η λειτουργία αλλαγής email περιλαμβάνει αυστηρό έλεγχο μοναδικότητας και ασφαλή τροποποίηση των στοιχείων του χρήστη. Η μέθοδος αξιοποιεί το Try για να διασφαλίσει τη σταθερότητα της διαδικασίας και την αποφυγή σφαλμάτων κατά την αποθήκευση ή την επικύρωση. Η αλλαγή εφαρμόζεται άμεσα και ενημερώνει το σύστημα ώστε οι μελλοντικές επικοινωνίες να αποστέλλονται στη νέα διεύθυνση.

4.4.4.5.3 Διαγραφή Λογαριασμού Χρήστη

Η εφαρμογή προσφέρει στον χρήστη τη δυνατότητα οριστικής διαγραφής του λογαριασμού του από την πλατφόρμα. Η ενέργεια αυτή διαγράφει όλα τα προσωπικά στοιχεία και τις εγγραφές του χρήστη από τη βάση, σύμφωνα με τις αρχές προστασίας δεδομένων. Η διαδικασία είναι μη αναστρέψιμη και συνοδεύεται από ενημερωτικό email επιβεβαίωσης.

Η διαγραφή εκτελείται με HTTP DELETE αίτημα προς το εξής endpoint:

```
DELETE /user
```

Το JWT του χρήστη αποστέλλεται στο Authorization header για να πραγματοποιηθεί η ταυτοποίηση. Δεν απαιτείται σώμα αίτησης.

Η μέθοδος στον controller:

```
@DeleteMapping
public ResponseEntity<Void> delete(
        @RequestHeader(HttpHeaders.AUTHORIZATION) String auth
) {
    String username = jwtUtils.extractUsername(auth.substring(7));
    return userService.delete(username)
            .onFailure(Throwable::printStackTrace)
            .map(ResponseEntity::ok)
            .get();
}
```

Η μέθοδος delete(...) στο UserService εκτελεί τα εξής βήματα:

- Αναζητά τον χρήστη στη βάση με βάση το username
- Διαγράφει την αντίστοιχη εγγραφή από τη βάση
- Αποστέλλει email επιβεβαίωσης στον χρήστη με αναλυτική ενημέρωση
- Επιστρέφει επιβεβαίωση επιτυχούς διαγραφής

Η υλοποίηση είναι η εξής:

```
public Try<Void> delete(String username) {
    return Try.of(() -> Option.ofOptional(userRepository.findByUsername(username))
```

```
            .getOrElseThrow(() -> new IllegalArgumentException("There is no user
with this username: " + username)))
        .flatMap(user -> Try.run(() -> userRepository.delete(user))
            .onFailure(Throwable::printStackTrace)
            .flatMap(ignored -> emailService.sendEmail(SendEmailDto.builder()
                    .sendTo(user.getEmail())
                    .subject("""
                            Account Deletion Confirmation – Your Uniboost profile
has been removed
                            """)
                    .body("""
                            Dear %s,

                            This message is to confirm that your Uniboost profile
has been successfully deleted.

                            All associated course enrollments, personal
information, and activity history have been permanently removed from our system in
accordance with our data handling policies.

                            If this action was performed in error, please note that
account recovery is not possible. You are welcome to re-register at any time by
creating a new profile.

                            This is an automated message sent by the Uniboost
academic system. Please do not reply to this email.

                            –
                            Uniboost Academic Services
                            """.formatted(user.getFirstname()))
                    .build())));
}
```

Η χρήση της δομής Try σε συνδυασμό με την αποστολή αυτοματοποιημένου email επιβεβαιώνει την ασφαλή εκτέλεση της διαγραφής και διατηρεί τον χρήστη πλήρως ενημερωμένο. Η λειτουργία αυτή προσφέρει στον χρήστη πλήρη έλεγχο των δεδομένων του και ανταποκρίνεται στις απαιτήσεις διαφάνειας και ψηφιακών δικαιωμάτων.

4.4.5 Ασφάλεια Εφαρμογής

Η ασφάλεια της εφαρμογής βασίζεται στον μηχανισμό authentication μέσω JWT (JSON Web Token) και υλοποιείται πλήρως από τη Spring Security. Η ταυτοποίηση των χρηστών γίνεται με τη χρήση

tokens, τα οποία αποστέλλονται από το frontend σε κάθε αίτημα μέσω του HTTP Authorization header. Το backend επαληθεύει το token και αποδίδει τα αντίστοιχα δικαιώματα στον χρήστη.

Η προσέγγιση που ακολουθείται είναι stateless authentication, χωρίς χρήση server-side sessions, κάτι που καθιστά την αρχιτεκτονική πιο ευέλικτη και κατάλληλη για εφαρμογές τύπου REST. Η δομή της ασφάλειας περιλαμβάνει μηχανισμό δημιουργίας και ελέγχου token, φίλτρο που παρεμβάλλεται στην αλυσίδα αιτημάτων, διαμόρφωση κανόνων πρόσβασης ανά endpoint, και διαχείριση αποσύνδεσης.

4.4.5.1 Δημιουργία και Έλεγχος Token (JWT)

Ο χειρισμός των tokens υλοποιείται στην κλάση JwtUtils, η οποία είναι υπεύθυνη για τη δημιουργία, αποκωδικοποίηση και επικύρωση των JWT access και refresh tokens. Η κλάση χρησιμοποιεί ως κρυπτογραφικό μηχανισμό τον αλγόριθμο HMAC-SHA, με μυστικό κλειδί που ορίζεται στο αρχείο ρυθμίσεων της εφαρμογής.

Η δημιουργία token βασίζεται σε πληροφορίες του UserDetails και μπορεί να συνοδεύεται από πρόσθετες claims:

```
public String generateToken(UserDetails userDetails) {
    return generateToken(new HashMap<>(), userDetails);
}
```

Η δημιουργία refresh token υλοποιείται με ανάλογο τρόπο, αλλά με διαφορετική διάρκεια ισχύος:

```
public String generateRefreshToken(UserDetails userDetails) {
    return buildToken(new HashMap<>(), userDetails, refreshExpiration);
}
```

Και στις δύο περιπτώσεις, η τελική μέθοδος buildToken(...) κατασκευάζει το token με χρήση του κρυπτογραφημένου κλειδιού:

```
private String buildToken(Map<String, Object> claims, UserDetails userDetails, Long
expiration) {
    return Jwts
            .builder()
            .claims(claims)
            .subject(userDetails.getUsername())
            .issuedAt(new Date(System.currentTimeMillis()))
            .expiration(new Date(System.currentTimeMillis() + expiration))
            .signWith(getSignInKey())
            .compact();
}
```

Η κρυπτογράφηση βασίζεται σε κλειδί που παρέχεται ως Base64 και αποκωδικοποιείται κατά την εκκίνηση:

```
private Key getSignInKey() {
    byte[] keyBytes = Decoders.BASE64.decode(secretKey);
    return Keys.hmacShaKeyFor(keyBytes);
}
```

Η μέθοδος extractUsername(...) ανακτά το subject του token (το οποίο αντιστοιχεί στο username του χρήστη):

```
public String extractUsername(String token) {
    return getClaim(token, Claims::getSubject);
}
```

Η εγκυρότητα του token ελέγχεται με δύο κριτήρια: ότι το subject ταυτίζεται με το username του χρήστη και ότι το token δεν έχει λήξει:

```
public boolean isTokenValid(String token, UserDetails userDetails) {
    final String username = extractUsername(token);
    return username.equals(userDetails.getUsername()) && !isTokenExpired(token);
}
```

Ο μηχανισμός αυτός εξασφαλίζει ότι το backend μπορεί να αποδώσει ρόλους και ταυτότητα στον χρήστη χωρίς επιπλέον βάση ή session, κάνοντας την αρχιτεκτονική της εφαρμογής πλήρως RESTful και ανεξάρτητη από server-side κατάσταση.

4.4.5.2 Έλεγχος Πρόσβασης με JwtAuthenticationFilter

Η επιβεβαίωση ταυτότητας των χρηστών για κάθε αίτημα υλοποιείται μέσω του JwtAuthenticationFilter. Πρόκειται για custom φίλτρο που επεκτείνει την κλάση OncePerRequestFilter, και ελέγχει αν το εισερχόμενο αίτημα περιέχει έγκυρο JWT token στο Authorization header. Αν το token είναι έγκυρο και αντιστοιχεί σε υπαρκτό χρήστη, δημιουργείται sessionless authentication context στο SecurityContextHolder.

Η βασική ροή του φίλτρου doFilterInternal(...) είναι η εξής:

- Ανιχνεύει το Authorization header και ελέγχει αν ξεκινά με Bearer
- Εξάγει το JWT token από το header
- Καλεί τη μέθοδο extractUsername(...) από το JwtUtils

- Αν υπάρχει username και ο χρήστης δεν είναι ήδη authenticated:
- Φορτώνει τα στοιχεία του χρήστη από τη βάση
- Επαληθεύει την εγκυρότητα του token
- Ορίζει authentication context στην εφαρμογή

Η βασική δομή του φίλτρου:

```
@Override
protected void doFilterInternal(HttpServletRequest request,
                                HttpServletResponse response,
                                FilterChain filterChain) throws ServletException,
IOException {
    final String authHeader = request.getHeader(HttpHeaders.AUTHORIZATION);
    String jwt = null;
    String username = null;

    if (authHeader != null && authHeader.startsWith("Bearer ")) {
        jwt = authHeader.substring(7);
        username = jwtUtils.extractUsername(jwt);
    }

    if (username != null && SecurityContextHolder.getContext().getAuthentication()
== null) {
        UserDetails userDetails = userDetailsService.loadUserByUsername(username);

        if (jwtUtils.isTokenValid(jwt, userDetails) &&
tokenService.isTokenValid(jwt)) {
            UsernamePasswordAuthenticationToken authentication =
                    new UsernamePasswordAuthenticationToken(userDetails, null,
userDetails.getAuthorities());
            authentication.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
            SecurityContextHolder.getContext().setAuthentication(authentication);
        }
    }

    filterChain.doFilter(request, response);
}
```

Επιπλέον, το φίλτρο περιλαμβάνει μηχανισμό χειρισμού σφαλμάτων ώστε σε περίπτωση ληγμένου ή μη έγκυρου token να επιστρέφεται 401 Unauthorized με κατάλληλο ProblemDetail response, σύμφωνα με το πρότυπο application/problem+json:

```
private void writeProblemDetailResponse(HttpServletResponse response, HttpStatus
status, String detail) throws IOException {
```

```
    ProblemDetail problemDetail = ProblemDetail.forStatusAndDetail(status, detail);
    problemDetail.setTitle("Authentication Failed");

    response.setStatus(status.value());
    response.setContentType("application/problem+json");
    response.getWriter().write(objectMapper.writeValueAsString(problemDetail));
}
```

Το φίλτρο εγγράφεται στην αλυσίδα ασφάλειας μέσω της κλάσης SecurityConfig και εκτελείται πριν από το UsernamePasswordAuthenticationFilter, ώστε να ελέγχει όλα τα αιτήματα που απαιτούν authentication.

Η υλοποίηση αυτή εξασφαλίζει ότι κάθε προστατευόμενο endpoint προσπελαύνεται μόνο από χρήστες με έγκυρα tokens, και ότι οι αποτυχίες authentication διαχειρίζονται με τυποποιημένο τρόπο.

4.4.5.3 Διαμόρφωση Πρόσβασης με SecurityConfig

Η κλάση SecurityConfig είναι υπεύθυνη για τον πλήρη καθορισμό της πολιτικής ασφαλείας της εφαρμογής. Χρησιμοποιώντας το SecurityFilterChain της Spring Security, ορίζονται οι κανόνες πρόσβασης στα endpoints, οι authentication μηχανισμοί, καθώς και η ρύθμιση της συμπεριφοράς του logout.

Η εφαρμογή ακολουθεί αρχιτεκτονική stateless authentication, κάτι που σημαίνει ότι δεν χρησιμοποιούνται HTTP Sessions. Κάθε αίτημα πρέπει να περιλαμβάνει έγκυρο JWT token για να επιτραπεί η πρόσβαση σε προστατευόμενους πόρους.

Η βασική υλοποίηση της μεθόδου securityFilterChain(...) περιλαμβάνει τα εξής:

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception
{
    http
        .csrf(AbstractHttpConfigurer::disable)
        .authorizeHttpRequests(req -> req
            .requestMatchers(
                "/user/login",
                "/user/register",
                "/user/refresh-token",
                "/user/forgot-password",
                "/user/reset-password"
            ).permitAll()
            .requestMatchers("/post/**", "/user/**").authenticated()
        )
```

```
        .sessionManagement(session -> session
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        )
        .authenticationProvider(authenticationProvider)
        .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class)
        .logout(logout -> logout
            .logoutUrl("/user/logout")
            .addLogoutHandler(logoutService)
            .logoutSuccessHandler((request, response, authentication) ->
SecurityContextHolder.clearContext())
        );

    return http.build();
}
```

Η πολιτική πρόσβασης ορίζεται ως εξής:

- Τα endpoints /user/login, /user/register, /user/refresh-token, /user/forgot-password και /user/reset-password είναι δημόσια προσβάσιμα (permitAll).
- Όλα τα υπόλοιπα endpoints που αρχίζουν με /post/ και /user/ απαιτούν επαλήθευση ταυτότητας (authenticated).
- Η διαχείριση συνεδριών ορίζεται ως STATELESS, πράγμα που σημαίνει ότι η ασφάλεια βασίζεται αποκλειστικά στα tokens και όχι σε sessions.
- Το φίλτρο JwtAuthenticationFilter εγγράφεται πριν το φίλτρο UsernamePasswordAuthenticationFilter, ώστε να αναλύει τα tokens πριν την κλασική login ροή.
- Η λειτουργία logout ρυθμίζεται χειροκίνητα, μέσω custom handler (logoutService) και μηδενισμού του security context.

Η κλάση αυτή είναι το κεντρικό σημείο ελέγχου για όλα τα θέματα πολιτικής πρόσβασης, και εξασφαλίζει ότι μόνο αυθεντικοποιημένοι χρήστες έχουν πρόσβαση στα προστατευόμενα δεδομένα της εφαρμογής. Παράλληλα, η σαφής διαχωριστική γραμμή ανάμεσα σε public και secured endpoints προσφέρει απλότητα στη συντήρηση και ευκολία επέκτασης στο μέλλον.

4.4.5.4 Φόρτωση Χρηστών και Αυθεντικοποίηση (AppConfig)

Η διαδικασία αυθεντικοποίησης στην εφαρμογή βασίζεται στη χρήση του UserDetailsService και του AuthenticationProvider, τα οποία διαμορφώνονται στην κλάση AppConfig. Μέσω αυτών, το Spring Security μπορεί να εντοπίζει χρήστες και να επαληθεύει τα credentials τους κατά την είσοδο στο σύστημα.

Η μέθοδος userDetailsService() υλοποιείται με @Bean και επιστρέφει έναν custom UserDetailsService, ο οποίος αναζητά τον χρήστη στη βάση με βάση το username:

```
@Bean
public UserDetailsService userDetailsService() {
    return username -> userRepository
            .findByUsername(username)
            .orElseThrow(() -> new IllegalArgumentException("The user : " +
username + " was not found."));
}
```

Η μέθοδος authenticationProvider() ορίζει τον μηχανισμό αυθεντικοποίησης και χρησιμοποιεί DaoAuthenticationProvider, ο οποίος:

- χρησιμοποιεί την παραπάνω μέθοδο για να φορτώσει τον χρήστη
- συγκρίνει τον κωδικό που δόθηκε από τον χρήστη με τον αποθηκευμένο στη βάση, χρησιμοποιώντας PasswordEncoder

```
@Bean
public AuthenticationProvider authenticationProvider() {
    DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
    provider.setUserDetailsService(userDetailsService());
    provider.setPasswordEncoder(passwordEncoder());
    return provider;
}
```

Η passwordEncoder() καθορίζει τον αλγόριθμο κρυπτογράφησης των κωδικών πρόσβασης. Στην προκειμένη περίπτωση χρησιμοποιείται ο αλγόριθμος BCrypt:

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

Τέλος, η μέθοδος authenticationManager() παρέχει τον κεντρικό διαχειριστή authentication, όπως απαιτείται από το Spring Security:

```
@Bean
public AuthenticationManager authenticationManager(AuthenticationConfiguration
config) throws Exception {
    return config.getAuthenticationManager();
}
```

Οι παραπάνω ρυθμίσεις επιτρέπουν στην εφαρμογή να ελέγχει credentials χρηστών, να κρυπτογραφεί με ασφάλεια τους κωδικούς και να υποστηρίζει μηχανισμούς authentication πλήρως εναρμονισμένους με τις πρακτικές του Spring Security.

4.4.5.5 Angular HTTP Interceptor και Διαχείριση Token

Η εφαρμογή διαθέτει μηχανισμό διαχείρισης αυθεντικοποίησης και ανανέωσης tokens μέσω custom HTTP Interceptor στο frontend. Ο μηχανισμός αυτός παρεμβαίνει σε κάθε εξερχόμενο HTTP αίτημα και προσθέτει αυτόματα το access token στο Authorization header. Η πληροφορία αυτή αντλείται από το localStorage και προστίθεται στο header του αιτήματος, όπως φαίνεται παρακάτω:

```
const accessToken = localStorage.getItem('access_token');
const cloned = accessToken
  ? req.clone({ headers: req.headers.set('Authorization', `Bearer ${accessToken}`)
})
  : req;
```

Εξαίρεση αποτελεί το endpoint /api/user/refresh-token, το οποίο αγνοείται ρητά ώστε να αποτραπεί οποιαδήποτε αναδρομή ανανέωσης. Το αιτηθέν αντικείμενο (cloned) προωθείται στον server μέσω της εντολής next(cloned).pipe(...).

Όταν η απόκριση του server επιστρέψει σφάλμα 401, δηλαδή όταν το access token έχει λήξει ή καταστεί άκυρο, ο μηχανισμός αναλαμβάνει να ανανεώσει το token εφόσον υπάρχει έγκυρο refresh token στο localStorage. Η μεταβλητή isRefreshing αποτρέπει ταυτόχρονες προσπάθειες ανανέωσης, ενώ το αρχικό αίτημα αποθηκεύεται προσωρινά στην retryRequest:

```
if (error.status === 401 && !isRefreshing) {
  isRefreshing = true;
  retryRequest = req;

  const refreshToken = localStorage.getItem('refresh_token');
  ...
}
```

Αν το refresh token είναι διαθέσιμο, αποστέλλεται POST αίτημα στο backend:

```
return http.post<{ access_token: string; refresh_token: string }>(
  '/api/user/refresh-token',
  null,
  {
    headers: { Authorization: `Bearer ${refreshToken}` }
  }
)
```

Με την επιτυχή απάντηση του server, τα νέα tokens αποθηκεύονται ξανά στο localStorage και το αρχικό αίτημα επαναλαμβάνεται, αυτή τη φορά με το νέο access token:

```
switchMap((res) => {
  localStorage.setItem('access_token', res.access_token);
  localStorage.setItem('refresh_token', res.refresh_token);
```

```
  const retryReq = retryRequest!.clone({
    headers: retryRequest!.headers.set('Authorization', `Bearer
${res.access_token}`)
  });

  isRefreshing = false;
  retryRequest = null;

  return next(retryReq);
})
```

Σε περίπτωση αποτυχίας της ανανέωσης, είτε λόγω λήξης του refresh token είτε για άλλον λόγο, το σύστημα προχωρά σε πλήρη εκκαθάριση των αποθηκευμένων credentials και προκαλεί ανανέωση της εφαρμογής:

```
catchError(refreshErr => {
  localStorage.clear();
  location.reload();
  return throwError(() => refreshErr);
});
```

Αυτόματα, ο χρήστης μεταφέρεται ξανά στη διαδικασία login, διατηρώντας τη συνοχή της εφαρμογής και αποτρέποντας κάθε ενδεχόμενο μη εξουσιοδοτημένης πρόσβασης.

Η τεχνική υλοποίηση αυτού του interceptor προσφέρει στο frontend έναν πλήρως αυτόνομο και αξιόπιστο μηχανισμό διαχείρισης authentication. Οι ενέργειες εκτελούνται με πλήρη διαφάνεια και χωρίς διακοπή στην εμπειρία χρήστη, ενώ ο συνδυασμός access και refresh tokens επιτρέπει τη διατήρηση μιας σταθερής και ασφαλούς συνεδρίας. Η λογική της ανανέωσης και του retry εγγυάται ότι κάθε αίτημα που απέτυχε λόγω ληγμένου token, θα επαναληφθεί χωρίς να απαιτείται ανθρώπινη παρέμβαση, εφόσον η ανανέωση είναι δυνατή.

4.4.5.6 Logout και Εκκαθάριση Token

Η εφαρμογή υποστηρίζει logout μηχανισμό ο οποίος διαγράφει ενεργά tokens από τη βάση και ακυρώνει κάθε μελλοντική χρήση τους. Με αυτόν τον τρόπο διασφαλίζεται ότι, ακόμα και αν κάποιος έχει στην κατοχή του έγκυρο JWT token, αυτό θα πάψει να ισχύει εφόσον ο χρήστης πραγματοποιήσει αποσύνδεση. Το logout υλοποιείται μέσω του endpoint /user/logout και υποστηρίζεται από custom handler LogoutService.

Η ρύθμιση του logout γίνεται μέσα από την κλάση SecurityConfig, όπου ορίζεται το URL αποσύνδεσης, ο handler και η τελική ενέργεια που εκτελείται:

```
.logout(logout -> logout
    .logoutUrl("/user/logout")
```

```
    .addLogoutHandler(logoutService)
    .logoutSuccessHandler((request, response, authentication) ->
SecurityContextHolder.clearContext())
)
```

Όταν ο client αποστείλει αίτημα POST /user/logout, ορίζεται από το σύστημα ότι:

- εκτελείται η κλάση LogoutService, η οποία αναλαμβάνει να εντοπίσει το token και να το ακυρώσει
- το context του χρήστη εκκαθαρίζεται μέσω SecurityContextHolder.clearContext()
- δεν απαιτείται αποστολή body στο αίτημα

Η βασική λογική που υλοποιείται στον logout handler είναι η εξής: ανακτά το JWT token από το Authorization header, αναζητά το token στη βάση δεδομένων και, αν υπάρχει, θέτει τις σημαίες revoked και expired σε true. Με τον τρόπο αυτό, το token παύει να είναι έγκυρο ακόμα κι αν δεν έχει λήξει χρονικά.

Εφόσον η εφαρμογή βασίζεται σε stateless authentication, το logout δεν συνεπάγεται διαγραφή HTTP session, αλλά ακύρωση των tokens. Η επιλογή αυτή είναι πλήρως συμβατή με RESTful αρχιτεκτονική και προσφέρει επιπλέον έλεγχο, καθώς η εφαρμογή παρακολουθεί τη ζωή κάθε token με ακρίβεια και δυνατότητα ανάκλησης.

Ο μηχανισμός του logout ολοκληρώνει τον κύκλο ασφάλειας της εφαρμογής, παρέχοντας ένα ξεκάθαρο σημείο εξόδου για τον χρήστη και ενισχύοντας τη συνολική αξιοπιστία του συστήματος authentication.

Κεφάλαιο 5ο: Συμπεράσματα ή/και προτάσεις βελτίωσης

Το παρόν κεφάλαιο αποσκοπεί στην αποτίμηση της εργασίας τόσο σε επίπεδο υλοποίησης όσο και ως προς τον αντίκτυπό της στο εκάστοτε περιβάλλον χρήσης. Γίνεται συνοπτική ανασκόπηση των επιτευγμάτων του συστήματος, αξιολογούνται οι τεχνικές επιλογές, αναδεικνύονται τα οφέλη για τον τελικό χρήστη και επισημαίνονται τα σημεία που θα μπορούσαν να βελτιωθούν ή να επεκταθούν στο μέλλον. Παράλληλα, προτείνονται συγκεκριμένες κατευθύνσεις για την εξέλιξη της εφαρμογής, τόσο λειτουργικά όσο και τεχνολογικά, εστιάζοντας σε πιθανές νέες δυνατότητες και περιβάλλοντα χρήσης.

5.1 Ανακεφαλαίωση και αποτίμηση έργου

Η παρούσα εργασία είχε ως αντικείμενο την ανάπτυξη μιας διαδικτυακής πλατφόρμας η οποία επιτρέπει τη δημιουργία και τη διαχείριση ιδιωτικών ή ομαδικών μαθημάτων μεταξύ φοιτητών. Η κεντρική ιδέα προέκυψε από την ανάγκη υποστήριξης της φοιτητικής κοινότητας μέσω ενός ευέλικτου, λειτουργικού και προσβάσιμου συστήματος αλληλοβοήθειας. Η εφαρμογή δίνει τη δυνατότητα σε έναν φοιτητή που έχει εξειδικευμένες γνώσεις (π.χ. στην Ανάλυση, στη Java ή σε άλλα πανεπιστημιακά αντικείμενα) να αναλάβει το ρόλο του διδάσκοντα και να προσφέρει βοήθεια σε συμφοιτητές του μέσα από την πλατφόρμα. Παράλληλα, κάθε χρήστης μπορεί να συμμετέχει και ως μαθητής σε κάποιο από τα διαθέσιμα μαθήματα άλλων χρηστών, ενισχύοντας τη δυναμική αλληλεπίδραση και αυτοοργάνωση της κοινότητας.

Κατά την υλοποίηση, δόθηκε ιδιαίτερη έμφαση στην ευχρηστία του περιβάλλοντος χρήστη και στην ευκολία πλοήγησης. Η λειτουργικότητα καλύπτει πλήρως τις ανάγκες που τέθηκαν στον αρχικό σχεδιασμό: εγγραφή και σύνδεση χρηστών, προβολή και αναζήτηση μαθημάτων, δημιουργία και διαχείριση μαθήματος, εγγραφή και αποεγγραφή φοιτητών, επεξεργασία στοιχείων προφίλ, ειδοποιήσεις μέσω email και ασφαλής πρόσβαση σε προστατευόμενα δεδομένα.

Η εφαρμογή υλοποιήθηκε με σύγχρονες τεχνολογίες: Spring Boot στο backend και Angular στο frontend. Η διεπαφή χρήστη σχεδιάστηκε εξολοκλήρου στα αγγλικά, προκειμένου να είναι προσβάσιμη και σε φοιτητές του προγράμματος Erasmus ή άλλους αλλοδαπούς χρήστες, υποστηρίζοντας έτσι μια πολυγλωσσική και συμπεριληπτική προσέγγιση. Ο μηχανισμός authentication βασίστηκε σε JSON Web Tokens (JWT), με σύστημα αυτόματης ανανέωσης access token μέσω refresh token. Επιπλέον, υλοποιήθηκε interceptor στο frontend ώστε η διαδικασία authentication να λειτουργεί με πλήρη διαφάνεια προς τον χρήστη.

Συνολικά, η εργασία πέτυχε τον αρχικό της στόχο, παραδίδοντας ένα λειτουργικό, χρηστικό και τεχνολογικά σύγχρονο εργαλείο που μπορεί να καλύψει πραγματικές ανάγκες της φοιτητικής κοινότητας. Το σύστημα λειτουργεί χωρίς εξάρτηση από εξωτερικές πλατφόρμες και μπορεί εύκολα να

εγκατασταθεί σε πανεπιστημιακό server. Η εμπειρία που αποκομίστηκε από την υλοποίηση κάλυψε ένα ευρύ φάσμα τεχνολογιών και προγραμματιστικών προσεγγίσεων, προσφέροντας ουσιαστική εμβάθυνση στην ανάπτυξη εφαρμογών πλήρους στοίβας.

5.2 Αξιολόγηση λειτουργικότητας και χρηστικότητας

Η εφαρμογή αξιολογείται ως προς δύο βασικούς άξονες: την πληρότητα της λειτουργικότητας που προσφέρει και τον βαθμό χρηστικότητας για τον τελικό χρήστη. Η συνδυαστική προσέγγιση backend–frontend, σε συνδυασμό με τις επιλογές σχεδίασης, εξασφαλίζει ένα σταθερό και αποτελεσματικό σύστημα για τη διαχείριση μαθημάτων μεταξύ φοιτητών.

Σε επίπεδο λειτουργικότητας, η πλατφόρμα καλύπτει πλήρως τον κύκλο ζωής ενός μαθήματος. Από τη δημιουργία του από τον διδάσκοντα, την επεξεργασία και τη διαγραφή του, μέχρι και την εγγραφή, αποεγγραφή και απομάκρυνση φοιτητών. Οι ενέργειες αυτές πραγματοποιούνται με ασφάλεια και συνοδεύονται από αυτόματες ειδοποιήσεις μέσω email, οι οποίες ενισχύουν την επικοινωνία και την πληροφόρηση των εμπλεκομένων μερών.

Από άποψη χρηστικότητας, το περιβάλλον χρήστη είναι σχεδιασμένο με στόχο την απλότητα και την άμεση κατανόηση. Η αρχική σελίδα παρέχει άμεση πρόσβαση σε όλα τα διαθέσιμα μαθήματα, ενώ μέσω απλών φίλτρων και λειτουργίας αναζήτησης ο χρήστης μπορεί να εντοπίσει εύκολα τα μαθήματα που τον ενδιαφέρουν. Οι ενότητες "Τα μαθήματά μου" και "Μαθήματα που παρακολουθώ" οργανώνουν την πληροφορία με τρόπο λειτουργικό και αποδοτικό, επιτρέποντας στον χρήστη να ελέγχει άμεσα τη δραστηριότητά του.

Η δυνατότητα δημιουργίας και διαχείρισης μαθημάτων προσφέρεται με διαισθητικό τρόπο, χωρίς να απαιτείται τεχνική εξοικείωση. Η σχεδίαση προβλέπει φόρμες με κατάλληλη επικύρωση, σαφείς οδηγίες και ελεγχόμενα πεδία. Επιπλέον, η διαχείριση του προφίλ (όπως αλλαγή email, username ή προσωπικών στοιχείων) υλοποιείται με ανεξάρτητες, σαφώς διαχωρισμένες ροές, προσφέροντας ακρίβεια και έλεγχο.

Επιπλέον, η επιλογή της αγγλικής γλώσσας για το περιβάλλον χρήστη καθιστά την πλατφόρμα προσβάσιμη σε ευρύτερο κοινό, συμπεριλαμβανομένων των φοιτητών Erasmus ή άλλων αγγλόφωνων φοιτητών του ιδρύματος. Η γλώσσα του περιβάλλοντος λειτουργεί έτσι υποστηρικτικά προς την εξωστρέφεια και τη συμπερίληψη.

Τέλος, η λειτουργία επαναφοράς κωδικού μέσω token παρέχει αξιοπιστία σε περιπτώσεις απώλειας πρόσβασης, ενώ η αυτόματη αποστολή επιβεβαιωτικών emails ενισχύει την εμπιστοσύνη του χρήστη προς την εφαρμογή. Η εμπειρία χρήστη συνολικά χαρακτηρίζεται από σταθερότητα, ταχύτητα και σαφήνεια στις ενέργειες που καλείται να εκτελέσει.

5.3 Τεχνικά οφέλη και αρχιτεκτονική

Η αρχιτεκτονική της εφαρμογής βασίζεται στον διαχωρισμό frontend και backend, ακολουθώντας τις αρχές του μοντέλου πλήρους στοίβας (full stack) και υλοποιήθηκε με χρήση τεχνολογιών αιχμής. Το backend αναπτύχθηκε με το πλαίσιο Spring Boot, το οποίο επέτρεψε την ταχεία κατασκευή RESTful APIs με σαφή οργάνωση, ασφάλεια και ευκολία επέκτασης. Το frontend δημιουργήθηκε με Angular, προσφέροντας μία σύγχρονη και αποδοτική εμπειρία χρήστη, με έμφαση στη δυναμική αλληλεπίδραση και την ταχύτητα απόκρισης.

Ο διαχωρισμός των ρόλων και η οργάνωση του backend σε controller–service–repository προσέφερε ευκρινή λογική δομή και δυνατότητα κλιμάκωσης. Οι λειτουργίες αυθεντικοποίησης και εξουσιοδότησης υλοποιήθηκαν μέσω JWT tokens, με χρήση access και refresh tokens, καλύπτοντας τις απαιτήσεις για stateless authentication. Η αποθήκευση των tokens και η δυνατότητα ανάκλησής τους προσέθεσε επίπεδο ελέγχου και ασφάλειας πέραν της βασικής υλοποίησης.

Ο σχεδιασμός και η υλοποίηση του frontend περιλαμβάνουν μηχανισμό Angular HTTP Interceptor, ο οποίος χειρίζεται αυτόματα την αποστολή και ανανέωση των tokens, προσφέροντας αδιάκοπη εμπειρία χρήστη. Η χρήση reactive προγραμματισμού (RxJS) διασφαλίζει την ομαλή ροή των αιτημάτων και των αποκρίσεων, ενώ το routing και η αρχιτεκτονική με modules καθιστούν το frontend απολύτως επεκτάσιμο.

Σε επίπεδο ασφάλειας, η εφαρμογή υποστηρίζει granular έλεγχο πρόσβασης, με role-based authorization και custom φίλτρα. Όλες οι προστατευόμενες ενέργειες ελέγχονται πριν την εκτέλεση, ενώ οι κρίσιμες συναλλαγές (π.χ. αλλαγές στα μαθήματα ή στους χρήστες) υλοποιούνται με @Transactional annotations ώστε να εξασφαλίζεται ατομικότητα και ακεραιότητα δεδομένων.

Αξιοσημείωτο είναι επίσης το γεγονός ότι η εφαρμογή έχει σχεδιαστεί ώστε να μπορεί να εκτελεστεί σε περιβάλλον πανεπιστημιακού server χωρίς την ανάγκη για εξωτερικά APIs ή υπηρεσίες. Το σύστημα είναι ανεξάρτητο, φορητό και μπορεί να επαναχρησιμοποιηθεί ή να προσαρμοστεί σε άλλες δομές εκπαιδευτικών οργανισμών, χωρίς αλλαγή στον βασικό του πυρήνα.

Η συνολική αρχιτεκτονική επιλογή προσφέρει υψηλό βαθμό σταθερότητας, επεκτασιμότητας και ασφάλειας, καθιστώντας την εφαρμογή τεχνολογικά έτοιμη να υποστηρίξει μεγαλύτερα σύνολα χρηστών και πιο σύνθετες λειτουργίες στο μέλλον.

5.4 Περιορισμοί και αδυναμίες

Παρά την πληρότητα και τη σταθερότητα της εφαρμογής, είναι σημαντικό να αναγνωριστούν τα σημεία όπου εντοπίζονται λειτουργικοί ή τεχνολογικοί περιορισμοί, είτε λόγω της έκτασης της εργασίας είτε λόγω συνειδητών επιλογών σχεδιασμού.

Η εφαρμογή προς το παρόν δεν περιλαμβάνει δυνατότητα πραγματοποίησης διαλέξεων ή απευθείας βιντεοκλήσεων εντός της πλατφόρμας. Η επικοινωνία μεταξύ φοιτητών και διδασκόντων περιορίζεται σε εξωτερικά κανάλια (π.χ. email ή άλλες πλατφόρμες) τα οποία μπορεί να συμφωνούνται μεταξύ τους ανεξάρτητα από το σύστημα. Αν και αυτό δεν αναιρεί τη χρησιμότητα της εφαρμογής ως εργαλείου οργάνωσης, αποτελεί έναν τομέα που θα μπορούσε να επεκταθεί στο μέλλον.

Επιπλέον, η τρέχουσα έκδοση δεν περιλαμβάνει ρόλο διαχειριστή (admin) ή μηχανισμούς εποπτείας. Όλοι οι χρήστες έχουν ίσα δικαιώματα ως προς τη δημιουργία και συμμετοχή σε μαθήματα. Αν και αυτό υποστηρίζει τη φιλοσοφία της αυτοοργάνωσης, ενδέχεται να περιορίζει την ικανότητα παρέμβασης σε περιπτώσεις κατάχρησης ή αναγκών διαχείρισης περιεχομένου.

Ένα ακόμη σημείο που απουσιάζει είναι η δυνατότητα αξιολόγησης των μαθημάτων ή των διδασκόντων. Η προσθήκη ενός μηχανισμού ανατροφοδότησης (π.χ. με βαθμολογίες ή σχόλια) θα μπορούσε να ενισχύσει τη διαφάνεια, να καλλιεργήσει κίνητρα και να βοηθήσει τους φοιτητές να επιλέγουν πιο συνειδητά τα μαθήματα που παρακολουθούν.

Τεχνικά, η εφαρμογή βασίζεται αποκλειστικά σε web περιβάλλον και δεν διαθέτει ακόμη mobile έκδοση ή native υποστήριξη για μικρές οθόνες. Αν και η σχεδίαση είναι responsive, η εμπειρία σε κινητές συσκευές δεν είναι βελτιστοποιημένη για πλήρως mobile-first χρήση.

Τέλος, δεν έχει ενσωματωθεί μηχανισμός ειδοποιήσεων σε πραγματικό χρόνο (π.χ. push notifications ή live alerts), γεγονός που περιορίζει την αμεσότητα σε σημαντικές αλλαγές, όπως ακυρώσεις μαθημάτων ή τροποποιήσεις συμμετοχών.

Οι παραπάνω περιορισμοί δεν αναιρούν την αξία της εφαρμογής, αλλά αναγνωρίζονται ως σημεία δυνητικής ενίσχυσης σε μελλοντικές εκδόσεις, ανάλογα με τις ανάγκες της κοινότητας χρηστών και την εξέλιξη του λογισμικού.

5.5 Προτάσεις μελλοντικής εξέλιξης

Με αφετηρία την πλήρως λειτουργική και σταθερή έκδοση της εφαρμογής, διαμορφώνονται πολλαπλές δυνατότητες εξέλιξης και εμπλουτισμού, τόσο σε επίπεδο λειτουργιών όσο και ως προς τη στρατηγική αξιοποίησής της. Η εμπειρία που αποκτήθηκε μέσα από την ανάπτυξη της παρούσας έκδοσης επιτρέπει την εξαγωγή ρεαλιστικών και ουσιαστικών προτάσεων για την περαιτέρω ωρίμανση του συστήματος.

Μία βασική προτεραιότητα θα μπορούσε να είναι η προσθήκη ρόλου διαχειριστή (admin), με δικαιώματα εποπτείας και δυνατότητα παρέμβασης σε περιπτώσεις καταχρήσεων, ψευδών δηλώσεων ή γενικότερης διαχείρισης περιεχομένου. Ένα αντίστοιχο dashboard θα μπορούσε να περιλαμβάνει στατιστικά χρήσης, αναφορές και μηχανισμούς moderation, ενισχύοντας την αξιοπιστία της πλατφόρμας.

Παράλληλα, η ενσωμάτωση μηχανισμού αξιολόγησης μαθημάτων θα προσέφερε σημαντική προστιθέμενη αξία. Οι φοιτητές θα μπορούσαν να εκφράζουν την εμπειρία τους από κάθε μάθημα, με τη μορφή βαθμολογίας ή σχολίων, δίνοντας τη δυνατότητα στους διδάσκοντες να βελτιώνονται και στους νέους χρήστες να κάνουν πιο τεκμηριωμένες επιλογές.

Ένα ακόμη ουσιαστικό βήμα θα ήταν η ενσωμάτωση εργαλείων για την απευθείας πραγματοποίηση των μαθημάτων, όπως υποστήριξη για Zoom, Jitsi ή άλλες video-conference πλατφόρμες, με δυνατότητα σύνδεσης ή προγραμματισμού συναντήσεων μέσα από την ίδια την εφαρμογή. Αυτό θα καθιστούσε το σύστημα ολοκληρωμένο ως προς τη διδακτική εμπειρία και θα εξάλειφε την ανάγκη για εξωτερικά εργαλεία.

Η προσθήκη υποστήριξης για push notifications ή άμεσες ειδοποιήσεις εντός της εφαρμογής θα ενίσχυε την αμεσότητα στην επικοινωνία. Οι χρήστες θα μπορούσαν να ενημερώνονται άμεσα για σημαντικές αλλαγές σε μαθήματα, νέα σχόλια, εγγραφές ή διαγραφές.

Τέλος, η υλοποίηση native mobile εφαρμογής (ή προοδευτικής web εφαρμογής – PWA) θα καθιστούσε την πλατφόρμα απόλυτα λειτουργική και προσβάσιμη από φορητές συσκευές, κάτι που ανταποκρίνεται στη σύγχρονη πραγματικότητα και στις προσδοκίες των φοιτητών.

Η ευελιξία της αρχιτεκτονικής και η αυτονομία της εφαρμογής επιτρέπουν την κλιμάκωσή της και την ενδεχόμενη επέκτασή της σε επίπεδο ιδρύματος ή ακόμα και διαπανεπιστημιακής χρήσης. Η πορεία που έχει ήδη χαραχθεί αποτελεί ένα σταθερό θεμέλιο για ένα σύστημα που μπορεί να εξελιχθεί, να επεκταθεί και να προσφέρει ουσιαστική υποστήριξη σε ακαδημαϊκές κοινότητες.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Oracle – The Java Programming Language: https://www.oracle.com/java/technologies/
[2] Baeldung – Introduction to Java: https://www.baeldung.com/java-tutorial
[3] Wikipedia – Java (programming language): https://en.wikipedia.org/wiki/Java_(programming_language)

[4] Spring Boot Reference Guide – Auto-Configuration: https://docs.spring.io/spring-boot/docs/current/reference/html/features.html#features.developing-auto-configuration
[5] Spring Boot – Embedded Web Servers: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#web.embedded-container

[6] Spring Data JPA Reference Documentation: https://docs.spring.io/spring-data/jpa/reference/index.html
[7] Baeldung – Introduction to Spring Data JPA: https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa
[8] GeeksforGeeks – Spring Data JPA Tutorial: https://www.geeksforgeeks.org/spring-data-jpa-tutorial/

[9] Spring Security – Official Project Page: https://spring.io/projects/spring-security
[10] Spring Security with JWT – Tutorialspoint: https://www.tutorialspoint.com/spring_security/spring_security_with_jwt.htm
[11] Spring Security JWT Authentication Tutorial – CodeJava.net: https://www.codejava.net/frameworks/spring-boot/spring-security-jwt-authentication-tutorial

[12] Functional Programming Concepts – Baeldung: https://www.baeldung.com/java-functional-programming
[13] Vavr Documentation – Introduction: https://docs.vavr.io
[14] Vavr Library – GitHub Repository: https://github.com/vavr-io/vavr

[15] Project Lombok – Official Website: https://projectlombok.org
[16] Hypersistence Utils – GitHub Repository: https://github.com/vladmihalcea/hypersistence-utils

[17] JavaScript language overview – MDN Web Docs: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Language_overview
[18] TypeScript: JavaScript With Syntax For Types – Official Website: https://www.typescriptlang.org/

[19] Angular – Architecture Overview: https://angular.io/guide/architecture
[20] Angular – Routing & Navigation: https://angular.io/guide/router
[21] Angular – Reactive Forms Guide: https://angular.io/guide/reactive-forms
[22] Angular Components Overview – MDN: https://developer.mozilla.org/en-US/docs/Web/Guide/Component_Model/Angular
[23] Angular CLI Overview: https://angular.io/cli

[24] RxJS Official Documentation: https://rxjs.dev/guide/overview
[25] RxJS Operators Guide: https://rxjs.dev/guide/operators
[26] Angular - The RxJS library: https://angular.io/guide/rx-library
[27] Learn RxJS: Introduction: https://www.learnrxjs.io/

[28] Reactive Programming with RxJS - Medium: https://medium.com/@rahul.a1739/reactive-programming-with-rxjs-f3e52d0ca1eb

[29] PostgreSQL – Official Documentation: https://www.postgresql.org/docs/
[30] Baeldung – Introduction to PostgreSQL: https://www.baeldung.com/intro-to-postgresql
[31] PostgreSQL – Wikipedia: https://en.wikipedia.org/wiki/PostgreSQL

[32] Baeldung – Introduction to Spring Boot: https://www.baeldung.com/spring-boot-start

[33] Spring Boot Mail Support – Official Docs: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#io.email

[34] Flyway – Database Migrations Made Easy: https://flywaydb.org/documentation/

[35] Eastern Illinois University, "Knack Peer Tutoring & Mentoring," https://www.eiu.edu/academicsupport/knack.php

[36] GoPeer, "Online Tutoring Platform," διαθέσιμο στο: https://gopeer.org

[37] Schoolhouse.world, "Free Virtual Peer Tutoring," https://schoolhouse.world

[38] YTeach, "Peer Tutoring Platform for Schools," https://yteach.com

[39] Tutorpeers, "Online Peer Tutoring For College Students," https://tutorpeers.com

[40] GitHub – CAHLR/OATutor, "An Open-source Adaptive Tutoring System," https://github.com/CAHLR/OATutor

Παράρτημα Α: Κώδικας PlantUML Διαγραμμάτων Ροής Εφαρμογής

4.3.1 Δημιουργία Λογαριασμού Χρήστη

```
@startuml
actor User
boundary "SignupComponent\n(Angular)" as UI
control "UserService" as Service
entity "Backend API\n/auth/signup" as API
database "PostgreSQL DB" as DB

User -> UI : Συμπλήρωση φόρμας\n(όνομα, email, username, password)
UI -> Service : this.userService.signup(userData)
Service -> API : POST /api/auth/signup\n(userDTO)
API -> DB : INSERT INTO users(...)
DB --> API : Επιτυχής αποθήκευση
API --> Service : 201 Created + μήνυμα επιβεβαίωσης
Service --> UI : Επιστροφή\nresponse επιτυχίας
UI -> User : Εμφάνιση μηνύματος επιτυχίας\n+ ανακατεύθυνση σε login
@enduml
```

4.3.2 Σύνδεση Χρήστη

```
@startuml
actor User
boundary "LoginComponent\n(Angular)" as UI
control "AuthService" as Service
entity "Backend API\n/auth/login" as API
database "PostgreSQL DB" as DB
collections "sessionStorage" as Storage

User -> UI : Συμπλήρωση email & password
UI -> Service : this.authService.login(credentials)
Service -> API : POST /api/auth/login
API -> DB : SELECT user\nWHERE email = ?
DB --> API : Χρήστης βρέθηκε\n+ έλεγχος password
API --> Service : 200 OK\n+ accessToken, refreshToken
Service -> Storage : sessionStorage.setItem(...tokens...)
Service --> UI : Επιστροφή επιτυχίας
UI -> User : Redirect στην αρχική σελίδα
@enduml
```

4.3.3 Αρχική Σελίδα – Λίστα Διαθέσιμων Μαθημάτων

```
@startuml
actor User
boundary "HomeComponent\n(Angular)" as UI
control "CourseService" as Service
entity "Backend API\n/courses/public" as API
database "PostgreSQL DB" as DB

User -> UI : Είσοδος στην αρχική σελίδα
UI -> Service : this.courseService.getAvailableCourses()
Service -> API : GET /api/courses/public\n+ Authorization: Bearer token
API -> DB : SELECT * FROM courses\nWHERE creator ≠ userId\nAND user NOT enrolled
DB --> API : Λίστα διαθέσιμων μαθημάτων
API --> Service : 200 OK + List<Course>
Service --> UI : Επιστροφή μαθημάτων
UI -> User : Προβολή λίστας με διαθέσιμα μαθήματα
@enduml
```

4.3.4 Αναζήτηση Μαθημάτων

```
@startuml
actor User
boundary "SearchComponent\n(Angular)" as UI
control "CourseService" as Service
entity "Backend API\n/courses/search?query=" as API
database "PostgreSQL DB" as DB

User -> UI : Εισαγωγή λέξης-κλειδιού (π.χ. "C++")
User -> UI : Πάτημα κουμπιού "Αναζήτηση"
UI -> Service : this.courseService.searchCourses("C++")
Service -> API : GET /api/courses/search?query=C%2B%2B\n+ Bearer token
API -> DB : SELECT * FROM courses\nWHERE title ILIKE '%C++%'
DB --> API : Λίστα μαθημάτων
API --> Service : 200 OK + μαθήματα
Service --> UI : Επιστροφή αποτελεσμάτων
UI -> User : Προβολή φιλτραρισμένων μαθημάτων
@enduml
```

4.3.5 Προβολή Λεπτομερειών Μαθήματος (Μη Εγγεγραμμένος Χρήστης)

```
@startuml
actor User
boundary "CreateCourseComponent\n(Angular)" as UI
control "CourseService" as Service
entity "Backend API\n/courses" as API
database "PostgreSQL DB" as DB

User -> UI : Συμπλήρωση φόρμας\n(τίτλος, περιγραφή, group size, μέθοδος)
User -> UI : Πάτημα κουμπιού "Καταχώριση"
UI -> Service : this.courseService.createCourse(courseData)
Service -> API : POST /api/courses\n+ Authorization token\n+ JSON σώμα
API -> DB : INSERT INTO courses (...)
DB --> API : Επιτυχής καταχώριση
API --> Service : 201 Created + courseId
Service --> UI : Επιστροφή επιτυχίας
UI -> User : Μήνυμα επιβεβαίωσης ή redirect σε λίστα
@enduml
```

4.3.6 Προβολή Λεπτομερειών Μαθήματος (Μη Εγγεγραμμένος Χρήστης)

```
@startuml
actor User
boundary "CourseDetailsComponent\n(Angular)" as UI
control "CourseService" as Service
entity "Backend API\n/courses/{id}" as API
database "PostgreSQL DB" as DB

User -> UI : Επιλογή μαθήματος από λίστα
UI -> Service : this.courseService.getCourseById(id)
Service -> API : GET /api/courses/{id}\n+ Bearer token
API -> DB : SELECT * FROM courses\nWHERE id = ?
DB --> API : Στοιχεία μαθήματος
API --> Service : 200 OK + course details
Service --> UI : Παράδοση δεδομένων
UI -> User : Προβολή τίτλου, περιγραφής, δημιουργού\n+ Κουμπί "Εγγραφή"
@enduml
```

4.3.7 Λεπτομέρειες Μαθήματος (μετά την εγγραφή)

```
@startuml
actor User
```

```
boundary "CourseDetailsComponent (Angular)" as UI
control "CourseService" as Service
entity "Backend API /courses/{id}" as API
database "PostgreSQL DB" as DB

User -> UI : Επιλογή μαθήματος από "Συμμετέχω"
UI -> Service : this.courseService.getCourseById(id)
Service -> API : GET /api/courses/{id} + Bearer token
API -> DB : SELECT * FROM courses WHERE id = ?\n+ συμμετέχοντες
DB --> API : Course + enrollments
API --> Service : 200 OK + course details
Service --> UI : Παράδοση δεδομένων
UI -> User : Εμφάνιση τίτλου, περιγραφής,\nλίστα συμμετεχόντων, αποεγγραφή
@enduml
```

4.3.8 Προφίλ Χρήστη – Μαθήματα στα Οποία Συμμετέχω

```
@startuml
actor User
boundary "ProfileComponent\n(Angular)" as UI
control "CourseService" as Service
entity "Backend API\n/courses/enrolled" as API
database "PostgreSQL DB" as DB

User -> UI : Μετάβαση στο προφίλ (tab "Συμμετέχω")
UI -> Service : this.courseService.getEnrolledCourses()
Service -> API : GET /api/courses/enrolled\n+ Bearer token
API -> DB : SELECT * FROM enrollments\nWHERE user_id = ?
DB --> API : Λίστα course_id\n→ JOIN με πίνακα courses
API --> Service : 200 OK + List<Course>
Service --> UI : Παράδοση λίστας
UI -> User : Προβολή συμμετοχών\n+ κουμπί αποεγγραφής
@enduml
```

4.3.9 Προφίλ Χρήστη – Μαθήματα που Έχω Δημιουργήσει

```
@startuml
actor User
boundary "ProfileComponent\n(tab: My Courses)" as UI
control "CourseService" as Service
entity "Backend API\nGET /post/myposts" as API
database "PostgreSQL DB" as DB

User -> UI : Επιλογή tab "My Courses"
UI -> Service : this.courseService.getMyCreatedCourses()
Service -> API : GET /post/myposts?page=0&size=10&sort=updatedAt\n+ Bearer token
API -> DB : SELECT * FROM courses\nWHERE creator_username = ?
DB --> API : Λίστα μαθημάτων (PostResponseDto[])
API --> Service : 200 OK + createdCourses[]
Service --> UI : Παράδοση δεδομένων
UI -> User : Εμφάνιση σε κάρτες με τίτλο, περιγραφή, χώρο, συμμετοχές
@enduml
```

4.3.10 Προβολή Μαθήματος από τον Δημιουργό

```
@startuml
actor Creator as User
boundary "CourseDetailsComponent (Angular)" as UI
control "CourseService" as Service
entity "Backend API GET /post/{postId}" as API
database "PostgreSQL DB" as DB

User -> UI : Επιλογή μαθήματος από My Courses
```

```
UI -> Service : this.courseService.getCourseById(id)
Service -> API : GET /post/{postId} + Bearer token
API -> DB : SELECT * FROM posts WHERE id = ? AND creator = ?
DB --> API : PostDetailsResponseDto
API --> Service : 200 OK + δεδομένα
Service --> UI : Παράδοση πλήρων στοιχείων
UI -> User : Προβολή τίτλου, περιγραφής,\nτύπου, τοποθεσίας, συμμετοχών
@enduml
```

4.3.11 Επεξεργασία Προφίλ Χρήστη (Edit Mode)

Επεξεργασία προσωπικών πληροφοριών

```
@startuml
actor User
boundary "ProfileComponent (Angular)" as UI
control "UserService" as Service
entity "Backend API /user/edit/personal" as API
database "PostgreSQL DB" as DB

User -> UI : Πατάει "Edit" στα προσωπικά στοιχεία
UI -> UI : Ενεργοποίηση input fields + Save
User -> UI : Επεξεργασία στοιχείων
User -> UI : Πάτημα Save
UI -> Service : updatePersonalInfo(data)
Service -> API : PUT /user/edit/personal
API -> DB : UPDATE users SET name, phone,... WHERE username = ?
DB --> API : OK
API --> Service : 200 OK
Service --> UI : Success feedback
UI -> User : Προβολή νέων στοιχείων
@enduml
```

Επεξεργασία Username

```
@startuml
actor User
boundary "ProfileComponent (Angular)" as UI
control "UserService" as Service
entity "Backend API /user/username" as API
database "PostgreSQL DB" as DB

User -> UI : Πατάει "Edit Username"
UI -> UI : Ενεργοποίηση πεδίου + Save
User -> UI : Νέο username
User -> UI : Πάτημα Save
UI -> Service : updateUsername(newUsername)
Service -> API : PUT /user/username
API -> DB : SELECT * FROM users WHERE username = ?
DB --> API : Αν υπάρχει → Error\nΑλλιώς → UPDATE users SET username = ?
API --> Service : 200 OK ή error "Username already exists"
Service --> UI : Feedback
UI -> User : Εμφάνιση νέου username ή μήνυμα σφάλματος
@enduml
```

Επεξεργασία Email

```
@startuml
actor User
boundary "ProfileComponent (Angular)" as UI
control "UserService" as Service
entity "Backend API /user/email" as API
database "PostgreSQL DB" as DB
```

```
User -> UI : Πατάει "Edit Email"
UI -> UI : Ενεργοποίηση input + Save
User -> UI : Νέο email
User -> UI : Πάτημα Save
UI -> Service : updateEmail(newEmail)
Service -> API : PUT /user/email
API -> DB : SELECT * FROM users WHERE email = ?
DB --> API : Αν υπάρχει → Error\nΑλλιώς → UPDATE users SET email = ?
API --> Service : 200 OK ή error "Email already exists"
Service --> UI : Feedback
UI -> User : Εμφάνιση νέου email ή μήνυμα σφάλματος
@enduml
```

4.3.12 Επεξεργασία Μαθήματος (Edit Mode)

```
@startuml
actor Creator as User
boundary "EditCourseComponent (Angular)" as UI
control "CourseService" as Service
entity "Backend API\nPUT /post" as API
database "PostgreSQL DB" as DB

User -> UI : Μετάβαση σε Edit Mode
UI -> UI : Προσυμπληρωμένα πεδία + disabled Save

== Αν γίνει αλλαγή ==

User -> UI : Αλλάζει description/location/type...
UI -> UI : Ενεργοποίηση κουμπιού Save
User -> UI : Πατάει Save
UI -> Service : updateCourse(postUpdateDto)
Service -> API : PUT /post + Bearer token
API -> DB : UPDATE posts SET ... WHERE id = ? AND creator = ?
DB --> API : OK
API --> Service : 200 OK
Service --> UI : Επιτυχής ενημέρωση
UI -> User : Επιστροφή στην προβολή + μήνυμα

== Αν δεν γίνει αλλαγή ==

User -> UI : Καμία αλλαγή
UI -> UI : Save παραμένει disabled
@enduml
```

4.3.13 Επαναφορά Κωδικού Πρόσβασης

```
@startuml
actor User
boundary "LoginComponent\n(Angular)" as LoginUI
boundary "ForgotPasswordComponent" as ForgotUI
boundary "ResetPasswordComponent" as ResetUI
control "UserService" as Service
entity "Backend API" as API
database "PostgreSQL DB" as DB
entity "Email Service" as Mail

== Βήμα 1: Αίτηση επαναφοράς ==

User -> LoginUI : Πατάει "Forgot your password?"
LoginUI -> ForgotUI : Redirect στη φόρμα reset
User -> ForgotUI : Συμπλήρωση username + click "Send"
ForgotUI -> Service : requestPasswordReset(username)
Service -> API : POST /user/forgot-password?username=
API -> DB : SELECT user WHERE username = ?
```

```
DB --> API : User entity
API -> DB : INSERT reset token (UUID, token, expiresIn)
API -> Mail : Send email με link + token
Mail --> User : Email με σύνδεσμο reset

== Βήμα 2: Άνοιγμα συνδέσμου ==

User -> ResetUI : Click στο email link (token στο URL)
ResetUI -> ResetUI : Προβολή φόρμας νέου κωδικού

== Βήμα 3: Υποβολή νέου κωδικού ==

User -> ResetUI : Εισαγωγή password + confirm
ResetUI -> Service : resetPassword(token, password)
Service -> API : POST /user/reset-password?token=
API -> DB : SELECT * FROM reset_tokens WHERE token = ?
DB --> API : ResetToken entity
API -> API : Validation (not expired & not used)
API -> DB : UPDATE user SET password = hash\nUPDATE token SET used = true
DB --> API : OK
API --> Service : 200 OK
Service --> ResetUI : Success feedback
ResetUI -> User : Redirect σε login + μήνυμα επιτυχίας
@enduml
```