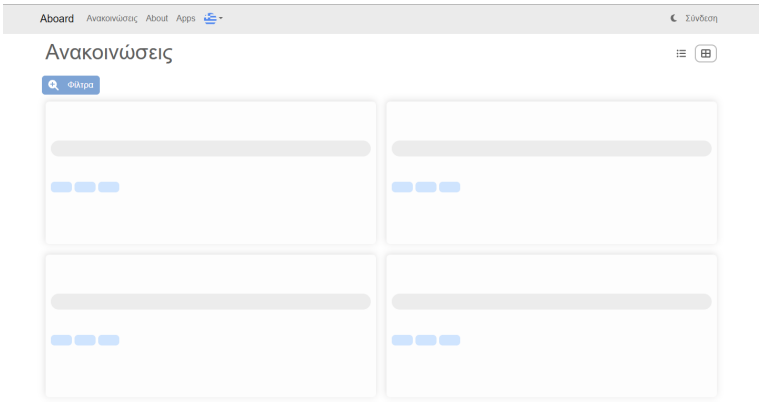
Διεθνές Πανεπιστήμιο Ελλάδος
Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Βελτίωση και Επέκταση του Συστήματος Ανακοινώσεων Aboard



Φοιτητής:
Χρήστος Ντομπρουγκίδης
Αριθμός Μητρώου: 185404

Επιβλέπων:
Αντώνης Σιδηρόπουλος

12 Σεπτεμβρίου 2025

Τίτλος Π.Ε.: Βελτίωση και Επέκταση του Συστήματος Ανακοινώσεων Aboard του τμήματος Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων
Κωδικός Π.Ε.: 24163
Ονοματεπώνυμο φοιτητή: Χρήστος Ντομπρουγκίδης
Ονοματεπώνυμο εισηγητή: Αντώνης Σιδηρόπουλος
Ημερομηνία ανάληψης Π.Ε.: 22-03-2024
Ημερομηνία περάτωσης Π.Ε.: 12-09-2025

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 πτυχιακή εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Πρόλογος

Η παρούσα πτυχιακή εργασία εκπονήθηκε με στόχο να αποτελέσει ένα θεμέλιο για ένα έργο που μπορεί να επεκταθεί και να εξελιχθεί περαιτέρω στο μέλλον από το τμήμα. Ένας από τους λόγους που επέλεξα αυτό το θέμα ήταν η προοπτική να δημιουργήσω μια βάση πάνω στην οποία θα μπορούν να στηριχθούν μελλοντικές βελτιώσεις και επεκτάσεις, προσφέροντας έτσι κάτι διαχρονικά χρήσιμο στη σχολή που μου παρείχε τόσο πολύτιμες γνώσεις κατά τη διάρκεια των σπουδών μου.

Κατά την ανάπτυξη της εφαρμογής, επικεντρώθηκα ιδιαίτερα στον σχεδιασμό και στον τρόπο υλοποίησης, ώστε να διασφαλιστεί η ευκολία συντήρησης και μελλοντικής επεκτασιμότητας. Μέσα από αυτή τη διαδικασία, εμβάθυνα σε τεχνικές που ενισχύουν την ευελιξία του κώδικα καθώς είναι ένα απαραίτητο στοιχείο για κάθε διαδικτυακή εφαρμογή που στοχεύει στη μακροχρόνια χρήση και εξέλιξη.

Αρχικά, γίνεται ανάλυση των απαιτήσεων του έργου και αναλύονται οι τεχνολογίες που χρησιμοποιούνται για την υλοποίηση του. Στην συνέχεια τονίζονται οι διαδικασίες που υλοποιήθηκαν ώστε να γίνει η επέκταση διατηρώντας την υπάρχουσα λειτουργικότητα. Τέλος, γίνεται μία πλήρης παρουσίαση της εφαρμογής.

Περίληψη

Η παρούσα εργασία παρουσιάζει τον επανασχεδιασμό και την επέκταση της εφαρμογής *aboard*, ενός συστήματος διαχείρισης ανακοινώσεων που χρησιμοποιείται για την ενημέρωση φοιτητών. Στο πλαίσιο της νέας υλοποίησης δόθηκε έμφαση στην υποστήριξη πολλαπλών τμημάτων του ίδιου πανεπιστημίου, με στόχο την κλιμακωσιμότητα και την ευελιξία. Για να επιτευχθεί αυτό, ανασχεδιάστηκε η δομή της βάσης δεδομένων με εισαγωγή ιεραρχικών ομάδων (`groups`) και πίνακα συσχέτισης χρήστη–ομάδας (`user_has_group`) με ρόλους ανά πλαίσιο, ενώ μεταφέρθηκαν τα καθολικά flags ρόλων από τον πίνακα χρηστών στη σχέση, βελτιώνοντας την κανονικοποίηση και τη συντηρησιμότητα.
Παράλληλα, αναπτύχθηκαν και τεκμηριώθηκαν εκτενώς τα REST endpoints του API για τις βασικές οντότητες (ανακοινώσεις, ομάδες, χρήστες), με έμφαση στην αυθεντικοποίηση μέσω SSO και στην ανταλλαγή δεδομένων σε μορφή JSON και iCalendar όπου απαιτείται. Η διεπαφή χρήστη σε ReactJS προσφέρει σύγχρονη εμπειρία πλοήγησης, με λειτουργίες φιλτραρίσματος, σελιδοποίησης και εναλλαγής προβολών. Τα αποτελέσματα καταδεικνύουν ότι ο νέος σχεδιασμός επιτρέπει την ασφαλή και καθαρή οριοθέτηση δεδομένων ανά τμήμα, διευκολύνοντας την υιοθέτηση από περισσότερες ακαδημαϊκές μονάδες και την περαιτέρω εξέλιξη της εφαρμογής.

Περιεχόμενα

Κατάλογος σχημάτων

Κατάλογος πινάκων

Κατάλογος Κώδικα

Κεφάλαιο 1

Απαιτήσεις εφαρμογής

1.1 Εισαγωγή στις απαιτήσεις της εφαρμογής

Οι απαιτήσεις ενός έργου αποτελούν ένα από τα σημαντικότερα βήματα για την επιτυχημένη υλοποίηση μιας εφαρμογής. Ως απαιτήσεις ορίζονται οι δηλώσεις που περιγράφουν είτε πτυχές της λειτουργικότητας ενός συστήματος είτε περιορισμούς που πρέπει να ενσωματώνονται σε αυτό [1]. Συγκεκριμένα, οι απαιτήσεις διατυπώνονται με τρόπο που να διευκολύνει την κατανόηση των διαφόρων λειτουργιών από όλα τα εμπλεκόμενα μέρη, αλλά και να καθοδηγεί τη συνολική διαδικασία ανάπτυξης, αποτελώντας τη βάση για όλα τα επόμενα στάδια.

1.2 Κατηγορίες

Οι απαιτήσεις μιας εφαρμογής διακρίνονται σε δύο κύριες κατηγορίες: λειτουργικές και μη λειτουργικές. Οι λειτουργικές απαιτήσεις περιγράφουν συγκεκριμένες δυνατότητες που πρέπει να διαθέτει το σύστημα, όπως για παράδειγμα η διαδικασία εγγραφής και αυθεντικοποίησης ενός χρήστη. Οι μη λειτουργικές απαιτήσεις δεν αφορούν συγκεκριμένες λειτουργίες, αλλά γενικότερες αρχές και χαρακτηριστικά που πρέπει να πληρούνται για να εξασφαλιστεί η ομαλή και αποδοτική λειτουργία του συστήματος. Για παράδειγμα, η ασφάλεια του συστήματος αποτελεί μη λειτουργική απαίτηση, καθώς επηρεάζει σχεδόν κάθε πτυχή του.

1.3 Ανάλυση

Ο στόχος της παρούσας εργασίας είναι ο επανασχεδιασμός της εφαρμογής Aboard. Ένα από τα πρώτα βήματα για την κατανόηση του έργου είναι η ανάλυση της υφιστάμενης εφαρμογής, ώστε να εντοπιστούν και να υλοποιηθούν οι αλλαγές που απαιτούνται για μια επιτυχημένη μετάβαση.

Η τρέχουσα εφαρμογή υλοποιεί πλήρως ένα ολοκληρωμένο σύστημα ανακοινώσεων, το οποίο περιλαμβάνει διαδικασία αυθεντικοποίησης χρηστών μέσω του πρωτοκόλλου OAuth2.0 για σύνδεση με το σύστημα login.iee.ihu.gr, που παρέχεται από το τμήμα. Κάθε χρήστης διαθέτει διαφορετικά δικαιώματα και διακρίνεται σε μία από τις τέσσερις κατηγορίες:

- Μη συνδεδεμένοι χρήστες, που έχουν πρόσβαση μόνο στις δημόσιες ανακοινώσεις (με εξαίρεση τη χρήση της εφαρμογής εντός των εγκαταστάσεων του τμήματος, όπου αποκτούν και πρόσβαση σε μη δημόσιες ανακοινώσεις).
- Απλοί συνδεδεμένοι χρήστες (φοιτητές), με πρόσβαση σε όλες τις ανακοινώσεις και στις ρυθμίσεις λογαριασμού.
- Συγγραφείς, οι οποίοι μπορούν να δημιουργούν ανακοινώσεις.
- Διαχειριστής (admin), που διαθέτει όλα τα δικαιώματα στην εφαρμογή.

Η τρέχουσα υλοποίηση υποστηρίζει επιπλέον τη δυνατότητα καρφιτσώματος σημαντικών ανακοινώσεων στην κορυφή για ορισμένο χρονικό διάστημα.

Από την ανάλυση του συστήματος προκύπτει ότι πρόκειται για μια ολοκληρωμένη λύση, χωρίς ουσιαστικές ελλείψεις, που παρέχει όλες τις κρίσιμες λειτουργίες που απαιτούνται σε εφαρμογές αυτού του τύπου. Ο λόγος για τον επανασχεδιασμό δεν σχετίζεται με λειτουργικά ελαττώματα ή ελλείψεις, αλλά με την ανάγκη υποστήριξης της επεκτασιμότητας και τον εκσυγχρονισμό του έργου.

1.4 Λειτουργικές απαιτήσεις

Σε αντίθεση με προηγούμενες εργασίες που εστίαζαν στον επανασχεδιασμό και την αναδόμηση του κώδικα, ο στόχος της παρούσας εργασίας είναι η επέκταση της βάσης δεδομένων και της επιχειρησιακής λογικής της εφαρμογής, ώστε να μπορεί να υποστηρίξει πολλαπλά τμήματα του πανεπιστημίου αντί για μόνο ένα.

Η υλοποίηση αυτή απαιτεί:

- Προσθήκη νέων οντοτήτων και σχέσεων στη βάση δεδομένων για την καταγραφή και διαχείριση των τμημάτων.
- Προσαρμογή του υπάρχοντος μοντέλου δεδομένων ώστε οι ανακοινώσεις, οι χρήστες και οι ρυθμίσεις τους να συσχετίζονται με το αντίστοιχο τμήμα.
- Ενημέρωση των μηχανισμών αυθεντικοποίησης και εξουσιοδότησης για να λαμβάνεται υπόψη η συσχέτιση χρήστη–τμήματος.
- Τροποποίηση του backend API και του frontend ώστε να υποστηρίζεται η προβολή, αναζήτηση και διαχείριση ανακοινώσεων ανά τμήμα.

- Διατήρηση της επεκτασιμότητας, ώστε στο μέλλον να μπορούν να προστεθούν επιπλέον ακαδημαϊκές μονάδες ή άλλοι φορείς χωρίς ριζικές αλλαγές στην αρχιτεκτονική.

1.5 Επίλογος

Σε αυτό το κεφάλαιο παρουσιάστηκε η έννοια των απαιτήσεων, έγινε διάκριση μεταξύ λειτουργικών και μη λειτουργικών απαιτήσεων και αναλύθηκε η υπάρχουσα εφαρμογή Aboard. Στη συνέχεια, προσδιορίστηκε ο σκοπός της παρούσας εργασίας, ο οποίος εστιάζει στην επέκταση της βάσης δεδομένων και της λογικής της εφαρμογής ώστε να καλύπτει πολλαπλά τμήματα του πανεπιστημίου, διατηρώντας παράλληλα την επεκτασιμότητα, την ασφάλεια και την αξιοπιστία του συστήματος.

Κεφάλαιο 2

Οι τεχνολογίες του έργου

Εισαγωγή

Για την ανάπτυξη της εφαρμογής εξετάστηκαν οι υφιστάμενες τεχνολογίες και η τρέχουσα υποδομή, καθώς οι νέες απαιτήσεις δεν επέβαλαν τη χρήση επιπρόσθετων ή διαφορετικών τεχνολογιών. Η επιλογή διατήρησης του υπάρχοντος τεχνολογικού συνόλου διασφαλίζει τη συνέπεια με την τρέχουσα υλοποίηση και επιτρέπει μια ομαλή μετάβαση στη νέα έκδοση. Παρακάτω παρουσιάζονται οι τεχνολογίες που χρησιμοποιήθηκαν, με αναφορά σε εναλλακτικές ως σημεία σύγκρισης.

2.1 PHP

Η PHP (Hypertext Preprocessor) είναι μια δημοφιλής γλώσσα προγραμματισμού που χρησιμοποιείται κυρίως για την ανάπτυξη δυναμικών ιστοσελίδων και διαδικτυακών εφαρμογών. Δημιουργήθηκε το 1994 από τον Rasmus Lerdorf και από τότε έχει εξελιχθεί σε μία από τις πιο διαδεδομένες γλώσσες στον χώρο του web development, με μια μεγάλη κοινότητα προγραμματιστών και πλούσια συλλογή βιβλιοθηκών [1].

2.1.1 Εξέλιξη της PHP

Η PHP ξεκίνησε το 1994 ως ένα σύνολο εργαλείων του Rasmus Lerdorf για την παρακολούθηση προσωπικών σελίδων (PHP/FI – Personal Home Page / Forms Interpreter). Με την κοινότητα να μεγαλώνει, η γλώσσα εξελίχθηκε ραγδαία: η **PHP 3** (1997–1998) ήταν η πρώτη πραγματικά αρθρωτή και επεκτάσιμη έκδοση, ενώ η **PHP 4** (2000) εισήγαγε τον *Zend Engine 1*, βελτιώνοντας σημαντικά τις επιδόσεις και το μοντέλο εκτέλεσης. Η **PHP 5** (2004) αποτέλεσε κομβικό σημείο, φέρνοντας τον *Zend Engine 2* και ώριμη αντικειμενοστραφή υποστήριξη (ορατότητες, `exceptions`, `interfaces`, `abstract` κλάσεις), μαζί με *extensions* όπως PDO

για ενοποιημένη πρόσβαση σε βάσεις δεδομένων. Η πορεία αυτή συνέβαλε στη μετάβαση από «σκριπτάκια» σε ολοκληρωμένες, μεγάλες εφαρμογές Μετά από μια περίοδο σταθεροποίησης, η **PHP 7** (2015) πρόσφερε άλμα επιδόσεων (νέος *engine*, μειωμένη κατανάλωση μνήμης), ενισχυμένη *typing* υποστήριξη (scalar type hints, return types) και βελτιωμένο σύστημα λαθών μέσω `Throwable`. Σε συνδυασμό με σύγχρονα *frameworks* (π.χ. Laravel) και διαχείριση εξαρτήσεων μέσω Composer, η PHP 7 εδραίωσε την PHP ως λύση υψηλής απόδοσης. Η **PHP 8** (2020 και μεταγενέστερες 8.x) εισήγαγε *JIT compilation*, *attributes* (annotations σε επίπεδο γλώσσας), *union types*, `match` expressions, *constructor property promotion*, καθώς και σταδιακή ενίσχυση των τύπων και της εργονομίας API. Οι εκδόσεις 8.1/8.2 συνέχισαν την πορεία αυτή με δυνατότητες όπως *enums*, *readonly* ιδιότητες/κλάσεις και περαιτέρω βελτιώσεις στο *type system*. Συνολικά, η εξέλιξη της PHP αντικατοπτρίζει μια συνεχή μετάβαση προς ισχυρότερη τυποποίηση, καλύτερη απόδοση και εργαλεία κατάλληλα για μεγάλες, συντηρήσιμες εφαρμογές. Σήμερα, η PHP συνδυάζει ώριμο οικοσύστημα (Composer/Packagist), σύγχρονες πρακτικές ανάπτυξης και ευρεία υποστήριξη σε υποδομές παραγωγής, παραμένοντας μια από τις πιο πρακτικές επιλογές για διαδικτυακές εφαρμογές μεγάλης κλίμακας [1].

2.1.2 Δυνατότητες της php

Η κύρια δύναμη της PHP βρίσκεται στην ευκολία εκμάθησης και χρήσης της, στην εξαιρετική συμβατότητά της με διάφορα λειτουργικά συστήματα και εξυπηρετητές ιστού, καθώς και στη μεγάλη γκάμα διαθέσιμων επεκτάσεων. Επιπλέον, η PHP υποστηρίζει αντικειμενοστραφή προγραμματισμό, ενσωματώνεται εύκολα με βάσεις δεδομένων όπως η MySQL και διαθέτει εκτενή τεκμηρίωση [1].

2.2 Laravel

Το Laravel είναι ένα σύγχρονο, ανοιχτού κώδικα PHP framework, το οποίο κυκλοφόρησε το 2011 από τον Taylor Otwell. Είναι σχεδιασμένο για να κάνει την ανάπτυξη web εφαρμογών πιο γρήγορη, απλή και ευχάριστη, προσφέροντας καθαρή σύνταξη και έτοιμα εργαλεία που μειώνουν την ανάγκη για επαναλαμβανόμενο κώδικα.

2.2.1 Εξέλιξη Laravel

Από την πρώτη του κυκλοφορία το 2011, το Laravel γνώρισε ραγδαία εξέλιξη και καθιερώθηκε ως ένα από τα πιο δημοφιλή frameworks για την ανάπτυξη εφαρμογών σε PHP [2], [3]. Η **έκδοση 1.x** προσέφερε μια βασική δρομολόγηση και απλή διαχείριση προτύπων, ενώ η **έκδοση 2.x** εισήγαγε το *Blade templating engine*, το οποίο έδωσε μεγαλύτερη ευελιξία στη δημιουργία Views. Η σημαντική καμπή ήρθε με το **Laravel 3** (2012), που πρόσθεσε το *Artisan command-line tool*, την υποστήριξη για migrations και το σύστημα package bundles. Το **Laravel 4** αποτέλεσε πλήρη επανεγγραφή πάνω στο Composer, καθιστώντας το framework πιο επεκτάσιμο και

συμβατό με το οικοσύστημα πακέτων της PHP. Με το **Laravel 5** (2015), το framework εμπλουτίστηκε με middleware, βελτιωμένο σύστημα authentication, job queues, event broadcasting και integration με το Laravel Elixir (πλέον Laravel Mix) για asset management. Οι συνεχείς αναβαθμίσεις των εκδόσεων 5.x εδραίωσαν τη φήμη του ως εργαλείο κατάλληλο για μεγάλες και σύνθετες εφαρμογές. Το **Laravel 6** (2019) καθιέρωσε τον κύκλο εκδόσεων LTS (Long Term Support) και εισήγαγε τα *Semantic Versioning*, ενώ το **Laravel 7** και **8** πρόσθεσαν δυνατότητες όπως improved routing, job batching και το Laravel Jetstream για authentication scaffolding. Στη σύγχρονη εκδοχή του (**Laravel 9 και 10**), το framework αξιοποιεί χαρακτηριστικά της PHP 8 όπως τα attributes, τα enums και τον JIT compiler, ενώ συνεχίζει να ενσωματώνει βελτιώσεις σε επιδόσεις, δομή κώδικα και developer experience. Έτσι, το Laravel παραμένει επίκαιρο, ευέλικτο και ιδανικό για την ανάπτυξη μοντέρνων web εφαρμογών μεγάλης κλίμακας.

2.2.2 Δυνατότητές Laravel

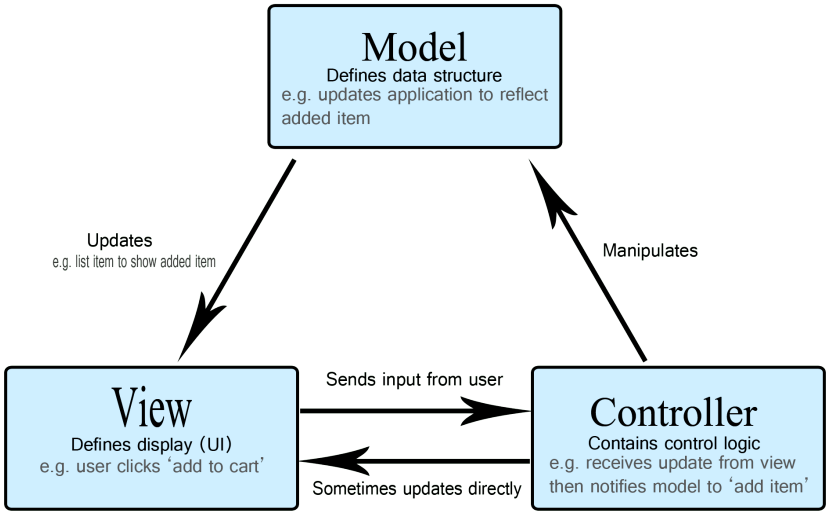
Το Laravel ακολουθεί την αρχιτεκτονική MVC (Model–View–Controller) και παρέχει ενσωματωμένα χαρακτηριστικά όπως ισχυρό σύστημα δρομολόγησης, ORM (Eloquent) για αλληλεπίδραση με τη βάση δεδομένων, μηχανισμό migration, queue σύστημα, καθώς και υποστήριξη για authentication, caching και REST APIs. Ένα από τα βασικά του πλεονεκτήματα είναι η μεγάλη κοινότητα και το οικοσύστημα πακέτων που το συνοδεύει, όπως το Laravel Horizon, Passport και Nova.

2.3 Η αρχιτεκτονική MVC

Η αρχιτεκτονική **MVC (Model–View–Controller)** είναι ένα από τα πιο διαδεδομένα πρότυπα σχεδίασης λογισμικού και χρησιμοποιείται εκτενώς στην ανάπτυξη web εφαρμογών. Η Laravel υιοθετεί την αρχιτεκτονική MVC ώστε να επιτυγχάνεται σαφής διαχωρισμός της επιχειρησιακής λογικής, της παρουσίασης και του ελέγχου της ροής, διευκολύνοντας την οργάνωση και τη συντήρηση του κώδικα [3], [4].

2.3.1 Model

Το **Model** αντιπροσωπεύει την επιχειρησιακή λογική και τη διαχείριση των δεδομένων της εφαρμογής. Είναι υπεύθυνο για την αλληλεπίδραση με τη βάση δεδομένων (ανάγνωση, αποθήκευση, ενημέρωση, διαγραφή), καθώς και για την εφαρμογή των κανόνων που διέπουν την πληροφορία. Στη Laravel, το Model υλοποιείται συνήθως μέσω του **Eloquent ORM**, το οποίο παρέχει έναν απλό και εκφραστικό τρόπο διαχείρισης δεδομένων με τη χρήση αντικειμένων αντί για απλά SQL queries.



Σχήμα 2.1: Το πρότυπο αρχιτεκτονικής MVC (Model–View–Controller).

2.3.2 View

Το **View** αφορά την παρουσίαση των δεδομένων στον τελικό χρήστη. Περιλαμβάνει την οπτική αναπαράσταση των πληροφοριών (HTML, CSS, JavaScript), χωρίς να περιέχει επιχειρησιακή λογική. Στο Laravel, το View υλοποιείται μέσω του **Blade templating engine**, το οποίο επιτρέπει τη χρήση καθαρής και επαναχρησιμοποιήσιμης σύνταξης για τη δημιουργία σελίδων, υποστηρίζοντας ταυτόχρονα συνθήκες, βρόχους και layouts.

2.3.3 Controller

Ο **Controller** λειτουργεί ως ο συνδετικός κρίκος μεταξύ του Model και του View. Δέχεται τα αιτήματα των χρηστών, επεξεργάζεται την εισαγόμενη πληροφορία και αποφασίζει ποια δεδομένα πρέπει να ανακτηθούν από το Model. Στη συνέχεια, στέλνει τα δεδομένα στο View ώστε να παρουσιαστούν στον χρήστη. Στο Laravel, οι Controllers οργανώνουν τη ροή της εφαρμογής, επιτρέποντας καλύτερη διαχείριση της λογικής σε σύγκριση με το να ενσωματώνεται η λογική απευθείας στις διαδρομές (routes).

2.3.4 Ο ρόλος της συνεργασίας Model–View–Controller

Η συνεργασία των τριών συνιστωσών μπορεί να περιγραφεί μέσα από την τυπική ροή μιας εφαρμογής:

1. Ο χρήστης στέλνει ένα αίτημα μέσω του προγράμματος περιήγησης (π.χ. φόρμα σύνδεσης).
2. Το αίτημα φτάνει στον Controller, ο οποίος το επεξεργάζεται και αποφασίζει ποιο Model θα χρησιμοποιήσει.
3. Το Model ανακτά ή τροποποιεί δεδομένα από τη βάση και τα επιστρέφει στον Controller.
4. Ο Controller περνάει τα δεδομένα στο View.
5. Το View δημιουργεί την κατάλληλη οπτική αναπαράσταση και την επιστρέφει στον χρήστη ως απάντηση.

Με αυτόν τον τρόπο, η MVC αρχιτεκτονική εξασφαλίζει διαχωρισμό ανησυχιών (separation of concerns), καθιστώντας την εφαρμογή πιο ευέλικτη, επεκτάσιμη και εύκολα συντηρήσιμη.

2.4 Composer

Ο **Composer** είναι ένα εργαλείο διαχείρισης εξαρτήσεων για την PHP, το οποίο εισήχθη το 2012 και άλλαξε ριζικά τον τρόπο με τον οποίο οι προγραμματιστές ενσωματώνουν βιβλιοθήκες και πακέτα στις εφαρμογές τους [5], [6]. Πριν από τον Composer, η διαδικασία προσθήκης εξωτερικών βιβλιοθηκών απαιτούσε χειροκίνητο κατέβασμα αρχείων, αντιγραφή στον κώδικα του έργου και διαχείριση ενημερώσεων, κάτι που οδηγούσε σε προβλήματα συντήρησης, ασυμβατότητες και δυσκολία στον συγχρονισμό μεταξύ διαφορετικών περιβαλλόντων ανάπτυξης.

2.4.1 Τα προβλήματα που έλυσε ο Composer

Ο Composer έλυσε μια σειρά από χρόνιες δυσκολίες στην ανάπτυξη εφαρμογών με PHP:

- **Αυτόματη διαχείριση εκδόσεων:** Δεν χρειάζεται πλέον χειροκίνητη ενημέρωση των βιβλιοθηκών· ο Composer αναλαμβάνει να εγκαταστήσει την κατάλληλη έκδοση κάθε πακέτου, λαμβάνοντας υπόψη τις εξαρτήσεις του.
- **Αναπαραγωγιμότητα περιβάλλοντος:** Με τη χρήση του αρχείου `composer.lock`, όλες οι εγκαταστάσεις του ίδιου έργου έχουν ακριβώς τις ίδιες εκδόσεις βιβλιοθηκών, αποφεύγοντας ασυμβατότητες.
- **Ενοποιημένο οικοσύστημα:** Μέσω του **Packagist**, του επίσημου αποθετηρίου πακέτων της PHP, οι προγραμματιστές έχουν πρόσβαση σε χιλιάδες βιβλιοθήκες με τυποποιημένο τρόπο εγκατάστασης.
- **Autoloading:** Ο Composer παρέχει μηχανισμό αυτόματης φόρτωσης (PSR-4 autoloading), ώστε οι κλάσεις να είναι διαθέσιμες χωρίς ρητές `require` ή `include`.

2.4.2 Πώς χρησιμοποιείται

Η χρήση του Composer βασίζεται κυρίως σε δύο αρχεία:

- Το `composer.json`, στο οποίο ορίζονται οι εξαρτήσεις μιας εφαρμογής, οι εκδόσεις τους και άλλες ρυθμίσεις.
- Το `composer.lock`, που δημιουργείται αυτόματα και «κλειδώνει» τις εκδόσεις όλων των εγκατεστημένων βιβλιοθηκών.

Η βασική ροή χρήσης έχει ως εξής:

1. Δημιουργία ή επεξεργασία του αρχείου `composer.json` με τις απαιτούμενες βιβλιοθήκες.
2. Εκτέλεση της εντολής `composer install` για την πρώτη εγκατάσταση των εξαρτήσεων ή `composer update` για την ανανέωση στην πιο πρόσφατη επιτρεπτή έκδοση.
3. Χρήση του μηχανισμού `autoload` που δημιουργείται αυτόματα στο `vendor/autoload.php`, ώστε να είναι διαθέσιμες όλες οι βιβλιοθήκες στον κώδικα της εφαρμογής.

Με τον τρόπο αυτό, ο Composer καθιστά την ανάπτυξη πιο αξιόπιστη, επαναλήψιμη και ευέλικτη, προσφέροντας έναν κοινό τρόπο διαχείρισης βιβλιοθηκών και διευκολύνοντας την συνεργασία σε ομάδες ανάπτυξης.

2.4.3 Συνηθισμένες εντολές του Composer

Στον Πίνακα 2.1 παρουσιάζονται μερικές από τις πιο συχνά χρησιμοποιούμενες εντολές του Composer, οι οποίες διευκολύνουν την εγκατάσταση, την ενημέρωση και τη διαχείριση πακέτων.

Πίνακας 2.1: Συνηθισμένες εντολές του Composer

Εντολή	**Περιγραφή**
`composer init`	Δημιουργεί διαδραστικά ένα νέο αρχείο `composer.json` για τον ορισμό εξαρτήσεων και ρυθμίσεων έργου.
`composer install`	Εγκαθιστά όλες τις εξαρτήσεις που ορίζονται στο `composer.json`. Δημιουργεί επίσης το αρχείο `composer.lock`.
`composer update`	Ενημερώνει τις υπάρχουσες εξαρτήσεις στην πιο πρόσφατη έκδοση, με βάση τους περιορισμούς που ορίζονται στο `composer.json`.
`composer require vendor/package`	Προσθέτει μια νέα εξάρτηση στο έργο και την εγκαθιστά άμεσα.
`composer remove vendor/package`	Αφαιρεί μια υπάρχουσα εξάρτηση από το έργο.
`composer dump-autoload`	Αναδημιουργεί το αρχείο autoloader (`vendor/autoload.php`), χρήσιμο μετά από αλλαγές σε namespaces ή classes.
`composer show`	Εμφανίζει πληροφορίες για τα εγκατεστημένα πακέτα.

2.5 HTML

Η **HTML (HyperText Markup Language)** αποτελεί τη βασική γλώσσα σήμανσης για την ανάπτυξη ιστοσελίδων. Μέσω της HTML καθορίζεται η δομή και το περιεχόμενο μίας ιστοσελίδας, όπως τίτλοι, παράγραφοι, εικόνες, πίνακες και σύνδεσμοι. Κάθε στοιχείο ορίζεται με τη χρήση ετικετών (tags), οι οποίες δίνουν στο πρόγραμμα περιήγησης οδηγίες για τον τρόπο εμφάνισης του περιεχομένου [7].

2.6 CSS

Η **CSS (Cascading Style Sheets)** είναι η γλώσσα που χρησιμοποιείται για τον καθορισμό της εμφάνισης και της μορφοποίησης μιας ιστοσελίδας. Μέσω της CSS, οι προγραμματιστές μπορούν να ελέγξουν χαρακτηριστικά όπως χρώματα, γραμματοσειρές, αποστάσεις, διάταξη και responsive σχεδιασμό. Η χρήση της CSS επιτρέπει τον διαχωρισμό της δομής (HTML) από την εμφάνιση, καθιστώντας τον κώδικα πιο οργανωμένο και ευκολότερο στη συντήρηση [8].

2.7 JavaScript

Η **JavaScript** είναι μια γλώσσα προγραμματισμού που εκτελείται στον φυλλομετρητή (client-side) και χρησιμοποιείται για να προσθέτει διαδραστικότητα στις ιστοσελίδες. Με τη JavaScript είναι δυνατή η υλοποίηση δυναμικών λειτουργιών όπως η επικύρωση φορμών, η διαχείριση γεγονότων (events), οι ασύγχρονες κλήσεις σε διακομιστές (AJAX) και η ενημέρωση

2.8 MySQL

Η **MySQL** είναι ένα από τα πιο διαδεδομένα συστήματα διαχείρισης σχεσιακών βάσεων δεδομένων (RDBMS), ανοιχτού κώδικα, που χρησιμοποιεί τη γλώσσα **SQL (Structured Query Language)** για την αλληλεπίδραση με τα δεδομένα. Αναπτύχθηκε αρχικά το 1995 από την σουηδική εταιρεία MySQL AB και στη συνέχεια εξαγοράστηκε από τη Sun Microsystems και αργότερα από την Oracle Corporation. Η MySQL είναι ιδιαίτερα δημοφιλής λόγω της αξιοπιστίας, της ταχύτητας και της ευκολίας χρήσης της, ενώ χρησιμοποιείται ευρέως σε web εφαρμογές, σε συνδυασμό με γλώσσες όπως η PHP [9].

2.8.1 Εξέλιξη της MySQL

Από την πρώτη της έκδοση, η MySQL έχει περάσει από πολλές φάσεις ανάπτυξης. Οι πρώτες εκδόσεις ήταν ελαφριές και προσανατολισμένες σε μικρές εφαρμογές. Σταδιακά, με την αύξηση της ζήτησης για πιο πολύπλοκα και μεγάλης κλίμακας συστήματα, η MySQL απέκτησε υποστήριξη για συναλλαγές (transactions), αποθηκευμένες διαδικασίες (stored procedures), triggers, views και βελτιωμένη διαχείριση δεικτών. Σήμερα, η MySQL παραμένει κορυφαία επιλογή για μικρές αλλά και μεγάλες εφαρμογές, και συχνά αποτελεί μέρος της γνωστής στοίβας ανάπτυξης **LAMP** (Linux, Apache, MySQL, PHP).

2.8.2 Βασικές Έννοιες της MySQL

Πίνακας (Table): Ο πίνακας αποτελεί τη βασική δομή αποθήκευσης δεδομένων στη MySQL. Κάθε πίνακας περιέχει γραμμές (rows/records) και στήλες (columns/fields), όπου οι στήλες ορίζουν το είδος των δεδομένων (π.χ. αριθμοί, κείμενο, ημερομηνίες) και οι γραμμές αποθηκεύουν τις πραγματικές τιμές.

Πρωτεύον Κλειδί (Primary Key): Είναι ένα ή περισσότερα πεδία ενός πίνακα που προσδιορίζουν μοναδικά κάθε γραμμή. Δεν μπορεί να περιέχει τιμές `NULL` και δεν επιτρέπονται διπλότυπα. Ένα παράδειγμα είναι το πεδίο `id` που χρησιμοποιείται συχνά για την ταυτοποίηση εγγραφών.

Ξένο Κλειδί (Foreign Key): Είναι ένα πεδίο που συνδέει δύο πίνακες μεταξύ τους. Το ξένο

κλειδί αναφέρεται στο πρωτεύον κλειδί ενός άλλου πίνακα και εξασφαλίζει την ακεραιότητα των δεδομένων. Για παράδειγμα, ένας πίνακας `announcements` μπορεί να περιέχει ένα πεδίο `user_id` που είναι ξένο κλειδί και αναφέρεται στον πίνακα `users`.

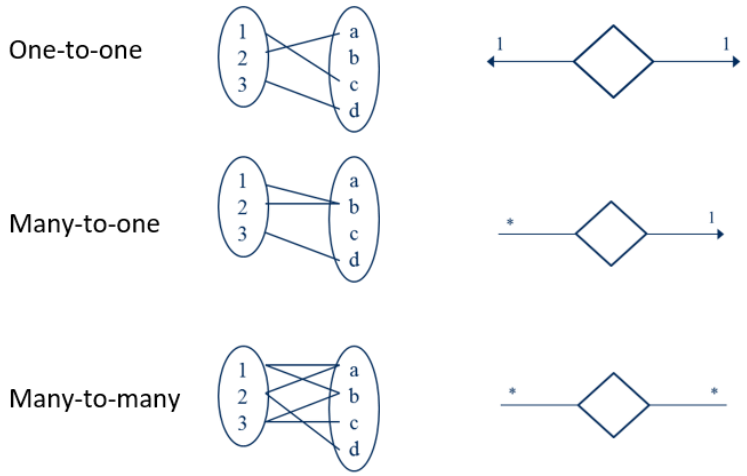
Περιορισμοί (Constraints): Οι περιορισμοί χρησιμοποιούνται για να καθορίσουν κανόνες που πρέπει να ακολουθούν τα δεδομένα ενός πίνακα. Μερικοί βασικοί περιορισμοί είναι:

- `NOT NULL`: Δεν επιτρέπει να είναι μια στήλη κενή.
- `UNIQUE`: Εξασφαλίζει ότι όλες οι τιμές σε μια στήλη είναι μοναδικές.
- `DEFAULT`: Ορίζει προεπιλεγμένη τιμή αν δεν δοθεί άλλη.
- `CHECK`: Ελέγχει ότι οι τιμές ικανοποιούν μία συνθήκη.
- `FOREIGN KEY`: Εξασφαλίζει τη σχέση με έναν άλλο πίνακα.

2.8.3 Τύποι Σχέσεων

Στη MySQL (και στις σχεσιακές βάσεις δεδομένων γενικότερα) οι σχέσεις μεταξύ πινάκων περιγράφουν τον τρόπο με τον οποίο συνδέονται τα δεδομένα:

- **Ένα-προς-Ένα (One-to-One):** Κάθε γραμμή σε έναν πίνακα συνδέεται με μία μόνο γραμμή σε έναν άλλο πίνακα. Παράδειγμα: Ένας χρήστης έχει ένα μόνο προφίλ.
- **Ένα-προς-Πολλά (One-to-Many):** Μία γραμμή σε έναν πίνακα μπορεί να συνδέεται με πολλές γραμμές σε έναν άλλο πίνακα. Παράδειγμα: Ένας καθηγητής μπορεί να έχει πολλές ανακοινώσεις.
- **Πολλά-προς-Πολλά (Many-to-Many):** Πολλές γραμμές σε έναν πίνακα συνδέονται με πολλές γραμμές σε άλλον πίνακα. Αυτό υλοποιείται συνήθως με έναν ενδιάμεσο πίνακα. Παράδειγμα: Οι φοιτητές μπορούν να συμμετέχουν σε πολλά μαθήματα και κάθε μάθημα έχει πολλούς φοιτητές.



Σχήμα 2.2: Σημειογραφία βέλους για τύπους σχέσεων: 1-1, 1-N, N-1, N-N

Κεφάλαιο 3

Αλλαγές στην εφαρμογή

3.1 Αλλαγές στη Βάση Δεδομένων

Η σημαντικότερη αλλαγή που πραγματοποιήθηκε στο πλαίσιο της παρούσας εργασίας αφορά την επέκταση του υπάρχοντος συστήματος ώστε να μπορεί να υποστηρίζει περισσότερες σχολές ή και πανεπιστήμια. Στην αρχική υλοποίηση, το σύστημα είχε σχεδιαστεί για να εξυπηρετεί μόνο ένα τμήμα και οι ρόλοι των χρηστών (π.χ. `admin`, `author`) ήταν αποθηκευμένοι απευθείας στον πίνακα χρηστών. Αυτό περιόριζε την ευελιξία, καθώς δεν υπήρχε η δυνατότητα ιεραρχικής ομαδοποίησης ή πολλαπλής συμμετοχής ενός χρήστη σε διαφορετικά πλαίσια (π.χ. διαφορετικές σχολές ή πανεπιστήμια).

Για την αντιμετώπιση αυτού του περιορισμού σχεδιάστηκε μια πιο ευέλικτη δομή στη βάση δεδομένων, η οποία επιτρέπει:

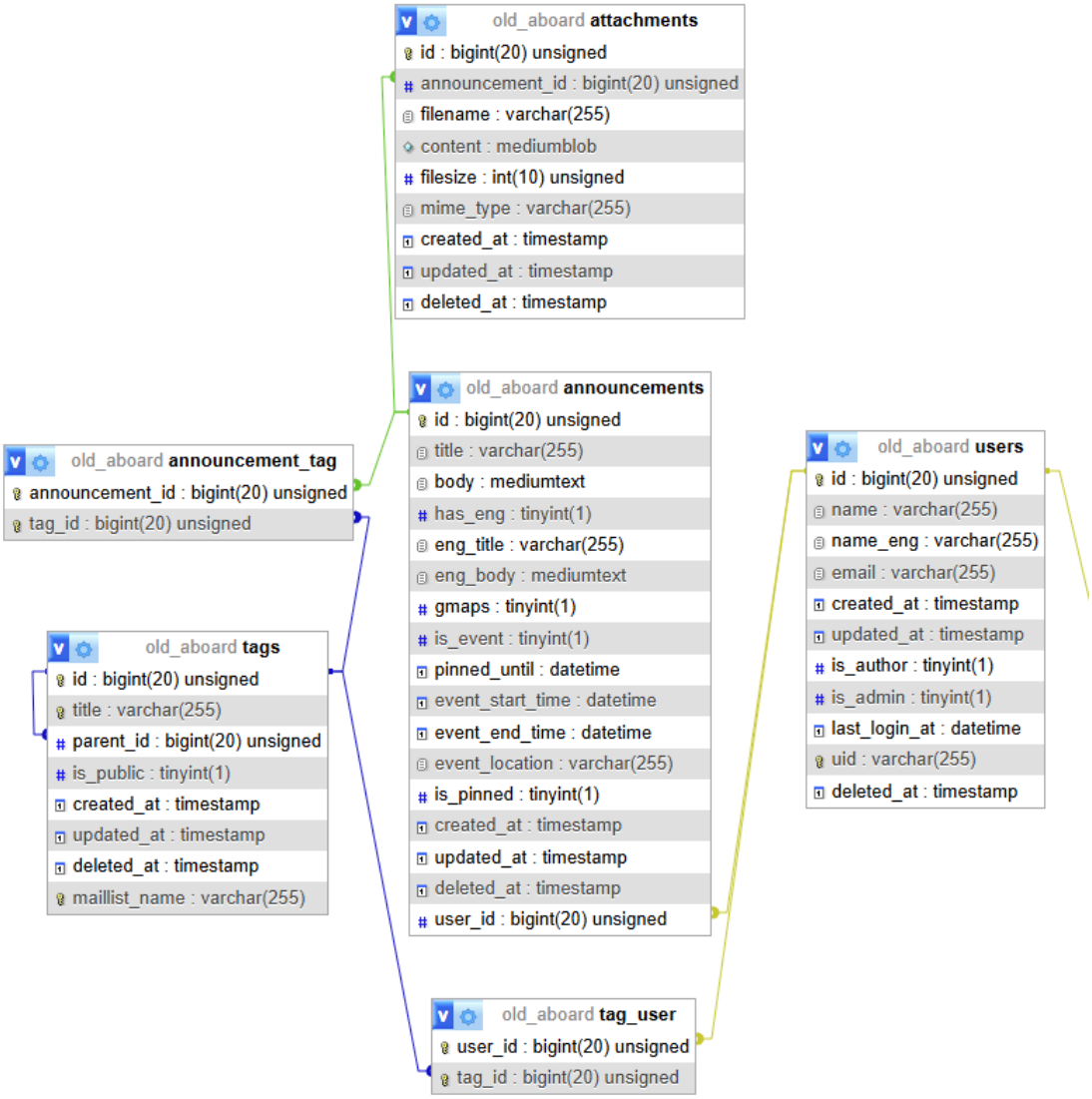
- Τη δημιουργία και ιεραρχική οργάνωση ομάδων (`groups`).
- Τη σύνδεση ενός χρήστη με πολλές ομάδες, μέσω πίνακα συσχέτισης.
- Τον ορισμό ρόλων ανά ομάδα (π.χ. ένας χρήστης μπορεί να είναι φοιτητής σε μία ομάδα και `admin` σε άλλη).

3.1.1 Παλαιό Σχήμα Βάσης Δεδομένων

Στην παλαιά έκδοση του συστήματος, η βάση δεδομένων περιλάμβανε τους βασικούς πίνακες `users`, `announcements`, `attachments`, `tags` και βοηθητικούς πίνακες όπως `announcement_tag`. Οι ρόλοι (`is_author`, `is_admin`) αποθηκεύονταν ως πεδία στον πίνακα `users`, ενώ κάθε ανακοίνωση συνδεόταν με έναν και μόνο χρήστη μέσω ξένου κλειδιού `user_id`.

Ενδεικτικά χαρακτηριστικά:

- Ο πίνακας `users` διαχειριζόταν όλους τους ρόλους σε παγκόσμιο επίπεδο.
- Δεν υπήρχε έννοια ομάδων ή τμημάτων· κάθε χρήστης ανήκε ουσιαστικά σε έναν χώρο.
- Η σχέση χρήστη–ανακοίνωσης ήταν ένα-προς-πολλά.

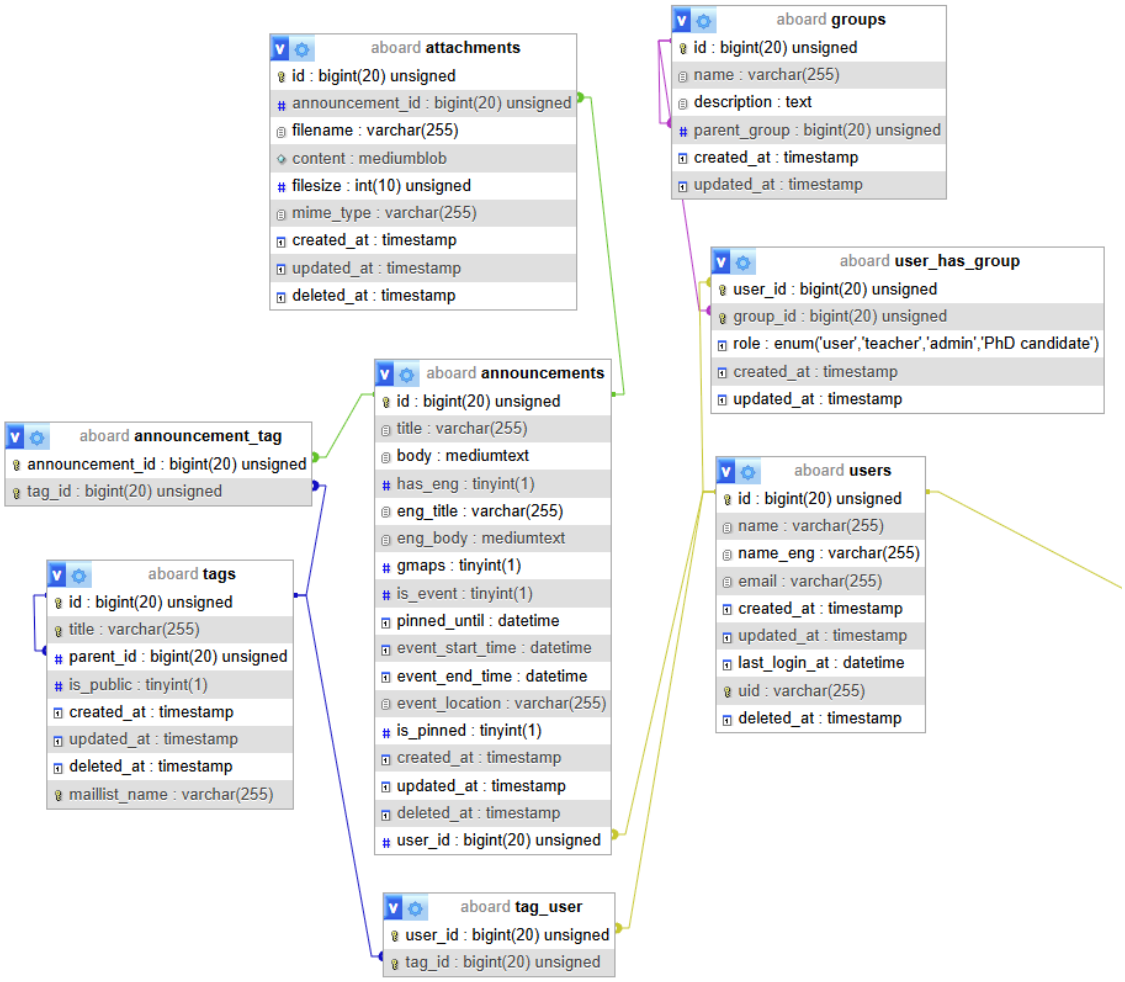


Σχήμα 3.1: Διάγραμμα της παλαιάς δομής της βάσης δεδομένων

3.1.2 Νέο Σχήμα Βάσης Δεδομένων

Η νέα έκδοση εισάγει την έννοια των ομάδων (`groups`) και των σχέσεων χρήστη–ομάδας (`user_has_group`), ενώ αφαιρεί τους παγκόσμιους ρόλους από τον πίνακα `users`. Με αυτό τον τρόπο:

- Ο πίνακας `groups` επιτρέπει την ιεραρχική οργάνωση (π.χ. Πανεπιστήμιο → Σχολή → Τμήμα) με πεδίο `parent_group`.
- Ο πίνακας `user_has_group` επιτρέπει πολλαπλές σχέσεις ενός χρήστη με διαφορετικές ομάδες, ορίζοντας και ρόλο μέσω `role` (`user`, `teacher`, `admin`, `PhD candidate`).
- Οι στήλες `is_author`, `is_admin` αφαιρούνται από τον πίνακα `users`.



Σχήμα 3.2: Διάγραμμα της νέας δομής της βάσης δεδομένων

3.1.3 Σύγκριση Παλαιού και Νέου Σχήματος

Εννοιολογική μοντελοποίηση Στο παλαιό σχήμα, οι ρόλοι χρηστών (`is_author`, `is_admin`) ήταν ιδιότητες της οντότητας `users` σε *παγκόσμιο* επίπεδο. Αυτό συνεπάγεται έναν μονοδιάστατο χώρο δικαιωμάτων: ο χρήστης έχει ή δεν έχει έναν ρόλο για *όλο* το σύστημα. Το νέο σχήμα μεταφέρει τους ρόλους στη *σχέση* μεταξύ χρήστη και πλαισίου (`user_has_group`),

καθιστώντας τους *τοπικούς* ως προς μία ομάδα (`group`). Έτσι, η έννοια «ρόλος» παύει να είναι ιδιότητα του `users` και γίνεται ιδιότητα της *συμμετοχής* του χρήστη σε ένα οργανωτικό πλαίσιο (π.χ. τμήμα, σχολή, ίδρυμα).

Ιεραρχική οργάνωση και scoping Η εισαγωγή του πίνακα `groups` με `parent_group` καθιστά το οργανωτικό μοντέλο *ιεραρχικό* (Ίδρυμα → Σχολή → Τμήμα), επιτρέποντας σαφή οριοθέτηση (scoping) δεδομένων και δικαιωμάτων. Οποιαδήποτε επιχειρησιακή λειτουργία μπορεί πλέον να εκτελείται στο κατάλληλο επίπεδο (π.χ. ανά τμήμα) και να *προβάλλεται* συνοπτικά σε ανώτερα επίπεδα, χωρίς να απαιτείται ανασχεδιασμός σχήματος.

Κανονικοποίηση και αποφυγή πλεονασμών Η μεταφορά των ρόλων από πεδία του `users` σε ξεχωριστή σχέση `user_has_group` βελτιώνει την κανονικοποίηση: εξαλείφεται η ανάγκη για πρόσθετα boolean/flags όταν προκύπτουν νέες κατηγορίες ρόλων, ενώ αποφεύγονται πλεονασμοί και ασυνέπειες (π.χ. «admin» σε ένα πλαίσιο αλλά όχι σε άλλο). Οι ρόλοι γίνονται εγγραφές δεδομένων κι όχι «στήλες σχήματος», με άμεση θετική επίπτωση στη συντηρησιμότητα.

Ακεραιότητα αναφορών και συμπεριφορά διαγραφών Τα ξένα κλειδιά (`FK`) και οι ρητές πολιτικές διαγραφής (`ON DELETE`) στηρίζουν την επιχειρησιακή λογική: `SET NULL` στο `parent_group` επιτρέπει ασφαλή αναδιάρθρωση ιεραρχιών χωρίς απώλεια θυγατρικών, ενώ `CASCADE` σε πίνακες συσχέτισης διασφαλίζει καθαρή απομάκρυνση συσχετισμών όταν αφαιρείται ένας βασικός κόμβος (χρήστης ή ομάδα). Έτσι, η βάση παραμένει συνεπής κατά την αλλαγή οργανωτικών δομών.

Έλεγχος πρόσβασης προσανατολισμένος στο πλαίσιο Με τους ρόλους να είναι ορισμένοι ανά `group`, ο μηχανισμός ελέγχου πρόσβασης ευθυγραμμίζεται με την επιχειρησιακή πραγματικότητα: ο ίδιος χρήστης μπορεί να έχει διαφορετικά δικαιώματα σε διαφορετικά τμήματα, και αυτά τεκμηριώνονται ρητά στο `user_has_group`. Οι πολιτικές πρόσβασης μπορούν να αξιοποιούν άμεσα αυτή τη σχέση, αποτρέποντας τη διαρροή δεδομένων μεταξύ οργανωτικών πλαισίων.

Κλιμάκωση και επαναχρησιμοποίηση Το νέο σχήμα είναι *γενικευμένο*: το `groups` λειτουργεί ως αφηρημένη αναπαράσταση «πλαισίου λειτουργίας». Η ίδια λογική εφαρμόζεται εξίσου καλά είτε το πλαίσιο είναι «τμήμα», είτε «σχολή», είτε «ίδρυμα». Η κλιμάκωση από ένα τμήμα σε πολλά δεν απαιτεί διαφοροποίηση του σχήματος – μόνο προσθήκη δεδομένων (*data-driven scaling*). Αυτό μειώνει το κόστος αλλαγής (change cost) και τον κίνδυνο σφαλμάτων.

Εκτελεσιμότητα ερωτημάτων (queryability) Οι συνηθισμένες ανάγκες αναφορών (π.χ. «οι ανακοινώσεις ενός τμήματος», «οι χρήστες που ανήκουν σε μια σχολή») εκφράζονται πλέον

με φυσικά joins πάνω σε `groups` και `user_has_group`. Η ύπαρξη ξεκάθαρης ιεραρχίας επιτρέπει και συγκεντρωτικές προβολές (roll-up) ανά επίπεδο, ενώ τα στοχευμένα ευρετήρια (indexes) σε `(user_id, group_id)` και `group`-σχετικά πεδία βελτιώνουν τον χρόνο απόκρισης.

Συντηρησιμότητα και καθαρός διαχωρισμός ευθυνών Η απλοποίηση του `users` (χωρίς application-specific flags) και ο σαφής διαχωρισμός «ταυτότητας» (users) από «συμμετοχή/ρόλο σε πλαίσιο» (`user_has_group`) καθιστούν τον κώδικα που εδράζεται στο σχήμα πιο προβλέψιμο: κάθε αλλαγή ρόλων, μετονομασία τμημάτων ή αναδιάρθρωση οργανισμού εντάσσεται σε δεδομένα, όχι σε μεταβολές σχήματος.

Η παρακάτω σύνοψη συγκρίνει τους δύο σχεδιασμούς κατά άξονα:

Πίνακας 3.1: Σύγκριση παλαιού και νέου σχήματος

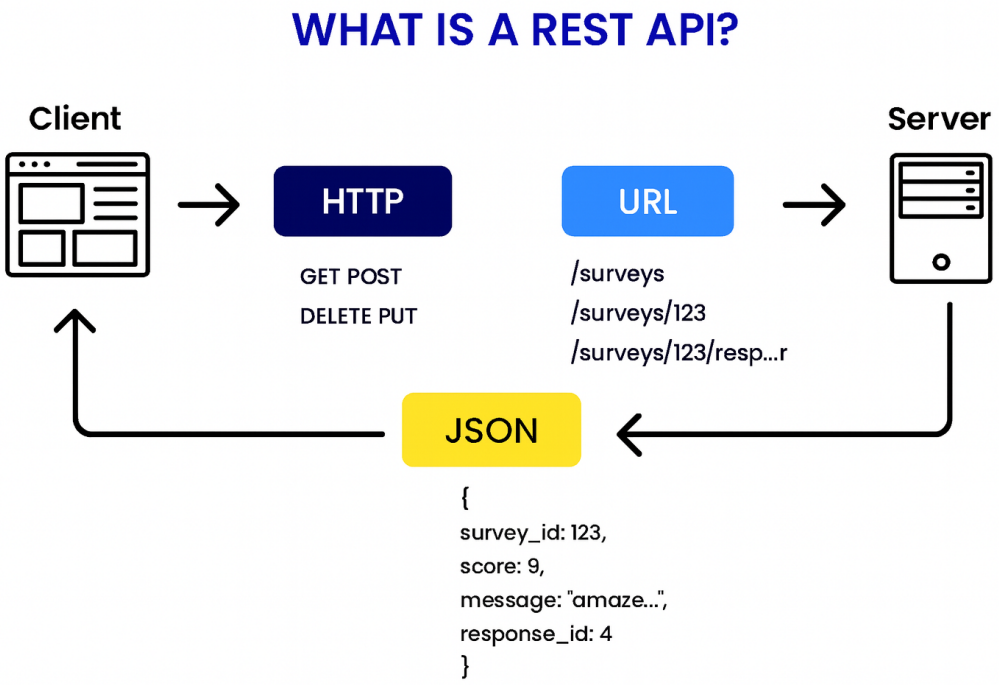
Άξονας	**Παλαιό Σχήμα**	**Νέο Σχήμα**
Μοντελοποίηση ρόλων	Ρόλοι ως πεδία στον `users` (*global flags*).	Ρόλοι ως ιδιότητα της σχέσης `user_has_group` (*contextual roles*).
Οργανωτική δομή	Μη ιεραρχικό μοντέλο· απουσία πλαισίων (τμημάτων/σχολών).	`groups` με `parent_group` για ιεραρχία (Ίδρυμα → Σχολή → Τμήμα).
Scoping δεδομένων	Μονολιθικό· δύσκολη οριοθέτηση ανά τμήμα.	Ρητό scoping ανά `group`; φυσικός διαχωρισμός δεδομένων ανά πλαίσιο.
Κανονικοποίηση	Περιορισμένη: προσθήκη ρόλων ως νέες στήλες/flags.	Βελτιωμένη: ρόλοι ως εγγραφές δεδομένων στη σχέση, χωρίς αλλαγές σχήματος.
Ακεραιότητα/Διαγραφές	Ασαφής διάδοση αλλαγών.	`FK` + `ON DELETE` πολιτικές (π.χ. `SET NULL`, `CASCADE`) για συνεπή συντήρηση.
Επεκτασιμότητα	Προσανατολισμός σε ένα τμήμα.	Γενικό μοντέλο για πολλαπλά τμήματα/σχολές/ιδρύματα.
Συντηρησιμότητα	Υψηλό κόστος αλλαγής ρόλων/δομών.	*Data-driven* αλλαγές: νέα πλαίσια/ρόλοι χωρίς μεταβολές schema.
Εκφραστικότητα ερωτημάτων	Φιλτράρισμα/αναφορές δύσκολες ανά οργανωτική ενότητα.	Φυσικά joins και roll-ups ανά `group` και ιεραρχία.

Συνολικά, το νέο σχήμα μετατοπίζει το βάρος από παγκόσμιους ρόλους και μονολιθική οντολο-

γία σε ένα *πλαισιοκεντρικό* μοντέλο με ιεραρχία, τοπικούς ρόλους και ξεκάθαρο scoping. Αυτή η αρχιτεκτονική επιλογή ευθυγραμμίζεται με την επιχειρησιακή ανάγκη υποστήριξης πολλαπλών τμημάτων του ίδιου ιδρύματος, προσφέροντας ταυτόχρονα καλύτερη κανονικοποίηση, ισχυρότερη ακεραιότητα και υψηλότερη συντηρησιμότητα.

3.2 Τι είναι το API

Ο όρος **API (Application Programming Interface)** αναφέρεται σε ένα σύνολο κανόνων και πρωτοκόλλων που επιτρέπουν την επικοινωνία μεταξύ διαφορετικών εφαρμογών ή συστημάτων. Μέσα από ένα API, ένας προγραμματιστής μπορεί να χρησιμοποιήσει τις λειτουργίες ενός λογισμικού χωρίς να γνωρίζει τις εσωτερικές λεπτομέρειες της υλοποίησής του. Στον χώρο των web εφαρμογών, τα APIs χρησιμοποιούνται εκτενώς ώστε να συνδέουν τον client (frontend) με τον server (backend), αλλά και για την επικοινωνία με εξωτερικές υπηρεσίες τρίτων [10].



Σχήμα 3.3: Απεικόνιση της επικοινωνίας μέσω REST API μεταξύ frontend και backend

3.2.1 API Requests

Τα πιο συνηθισμένα APIs για το διαδίκτυο είναι τα **REST APIs**, τα οποία βασίζονται στο πρωτόκολλο HTTP. Η επικοινωνία γίνεται μέσω **αιτημάτων (requests)** από τον client προς

τον server και κάθε αίτημα αντιστοιχεί σε μία συγκεκριμένη ενέργεια. Οι βασικοί τύποι HTTP requests είναι:

- **GET:** Χρησιμοποιείται για την ανάκτηση δεδομένων από τον server. Δεν τροποποιεί τα δεδομένα και θεωρείται "ασφαλής" μέθοδος. Παράδειγμα: ανάκτηση όλων των ανακοινώσεων.
- **POST:** Χρησιμοποιείται για την αποστολή νέων δεδομένων στον server, συνήθως για τη δημιουργία νέων εγγραφών στη βάση δεδομένων. Παράδειγμα: δημιουργία νέας ανακοίνωσης από έναν καθηγητή.
- **PUT:** Χρησιμοποιείται για την ενημέρωση (update) μίας υπάρχουσας εγγραφής, αντικαθιστώντας τα δεδομένα της με νέα. Παράδειγμα: ενημέρωση του περιεχομένου μίας ήδη υπάρχουσας ανακοίνωσης.
- **PATCH:** Παρόμοιο με το PUT, αλλά χρησιμοποιείται για μερική ενημέρωση δεδομένων αντί για πλήρη αντικατάσταση. Παράδειγμα: αλλαγή μόνο του τίτλου μίας ανακοίνωσης.
- **DELETE:** Χρησιμοποιείται για τη διαγραφή δεδομένων από τον server. Παράδειγμα: διαγραφή μίας ανακοίνωσης από το σύστημα.

Η χρήση αυτών των μεθόδων προσφέρει μία τυποποιημένη και προβλέψιμη διαδικασία επικοινωνίας, η οποία βοηθά στη δημιουργία εφαρμογών που είναι ευέλικτες, επεκτάσιμες και εύκολες στη συντήρηση.

3.3 Κωδικοί Απόκρισης API

Όταν ένας client επικοινωνεί με ένα API μέσω HTTP αιτημάτων, ο server επιστρέφει έναν **κωδικό κατάστασης (status code)** που δηλώνει το αποτέλεσμα της ενέργειας. Οι κωδικοί χωρίζονται σε κατηγορίες ανάλογα με το πρώτο τους ψηφίο:

- **2xx – Επιτυχία:** Η αίτηση ολοκληρώθηκε με επιτυχία.
- **3xx – Ανακατεύθυνση:** Ο client πρέπει να κάνει επιπλέον ενέργεια, όπως ανακατεύθυνση σε άλλο URL.
- **4xx – Σφάλματα Client:** Υπάρχει πρόβλημα στο αίτημα που έστειλε ο client.
- **5xx – Σφάλματα Server:** Προέκυψε σφάλμα στην πλευρά του server κατά την επεξεργασία της αίτησης.

Οι πιο συχνοί κωδικοί απόκρισης που εμφανίζονται σε REST APIs παρουσιάζονται στον παρακάτω πίνακα:

Πίνακας 3.2: Περιγραφή κωδικών απόκρισης.

Κωδικός	**Κατηγορία**	**Περιγραφή**
200 OK	Επιτυχία	Η αίτηση εκτελέστηκε σωστά και ο server επέστρεψε την απάντηση.
201 Created	Επιτυχία	Δημιουργήθηκε επιτυχώς ένας νέος πόρος (π.χ. νέα ανακοίνωση).
204 No Content	Επιτυχία	Η αίτηση ολοκληρώθηκε αλλά δεν υπάρχει περιεχόμενο στην απάντηση (συχνό σε DELETE).
301 Moved Permanently	Ανακατεύθυνση	Ο πόρος έχει μετακινηθεί μόνιμα σε νέο URL.
302 Found	Ανακατεύθυνση	Ο πόρος βρίσκεται προσωρινά σε άλλο URL.
400 Bad Request	Σφάλμα Client	Το αίτημα ήταν λανθασμένο ή ατελές.
401 Unauthorized	Σφάλμα Client	Ο client δεν έχει έγκυρα credentials (π.χ. λάθος ή ληγμένο token).
403 Forbidden	Σφάλμα Client	Ο client είναι αυθεντικοποιημένος αλλά δεν έχει δικαίωμα πρόσβασης.
404 Not Found	Σφάλμα Client	Ο ζητούμενος πόρος δεν βρέθηκε.
409 Conflict	Σφάλμα Client	Υπάρχει σύγκρουση (π.χ. διπλοεγγραφή ή πόρος που υπάρχει ήδη).
422 Unprocessable Entity	Σφάλμα Client	Το αίτημα ήταν συντακτικά σωστό αλλά απέτυχε σε επίπεδο λογικής/validation.
429 Too Many Requests	Σφάλμα Client	Ο client υπερέβη το επιτρεπτό όριο αιτημάτων (rate limit).
500 Internal Server Error	Σφάλμα Server	Γενικό σφάλμα στον server.
502 Bad Gateway	Σφάλμα Server	Ο server acting as gateway έλαβε μη έγκυρη απάντηση από άλλο server.
503 Service Unavailable	Σφάλμα Server	Η υπηρεσία δεν είναι διαθέσιμη προσωρινά (συντήρηση, υπερφόρτωση).
504 Gateway Timeout	Σφάλμα Server	Ο upstream server δεν απάντησε εγκαίρως.

3.3.1 API Endpoints

Στον παρακάτω πίνακα παρουσιάζονται όλα τα endpoints που προσφέρει η νέα έκδοση του API. Όλα τα endpoints είναι σχετικά ως προς τη βάση `{BASE_URL}` και απαιτούν, όπου αναφέρεται, κεφαλίδα `Authorization: Bearer {TOKEN}`.

Πίνακας 3.3: Λίστα διαθέσιμων endpoints της εφαρμογής

Endpoint	**Method**	**Περιγραφή**
`/api/v2/authenticate`	GET	Ανταλλαγή `code` από `login.iee.ihu.gr` με JWT access token.
`/api/v3/announcements`	GET	Λίστα ανακοινώσεων με φίλτρα (π.χ. `title`, `page`, `perPage`, `tags[]`, `users[]`, `updatedAfter`, `updatedBefore`, `sortId`).
`/api/v3/announcements` *(Content-Type: text/calendar)*	GET	Επιστροφή ημερολογίου (ICS) όταν το `Content-Type` είναι `text-calendar`. Ίδια φίλτρα με το απλό GET.
`/api/v3/announcements/{ANNOUNCEMENT_ID}`	GET	Ανάκτηση μίας συγκεκριμένης ανακοίνωσης.
`/api/v3/announcements`	POST	Δημιουργία ανακοίνωσης (τίτλος, σώμα, tags, συνημμένα, pin/event). Απαιτεί `Authorization` & δικαιώματα συγγραφέα.
`/api/v3/announcements/{id}`	PUT	Ενημέρωση ανακοίνωσης. Απαιτεί `Authorization` & ο χρήστης να είναι ο συγγραφέας.
`/api/v3/announcements/{id}`	DELETE	Διαγραφή ανακοίνωσης. Απαιτεί `Authorization` & ιδιοκτησία.
`/api/v3/announcements/{ANNOUNCEMENT_ID}/attachments/\{ATTACHMENT_ID\}`	GET	Λήψη/stream συνημμένου αρχείου ανακοίνωσης.
`/api/v3/attachments/{ATTACHMENT_ID}`	GET	Μεταδεδομένα για συγκεκριμένο συνημμένο.
`/api/v3/attachments/{ATTACHMENT_ID}/download`	GET	Λήψη αρχείου συνημμένου.
`/api/v3/tags`	GET	Όλα τα tags.
`/api/v3/filtertags`	GET	Tags σε nested μορφή (υποστηρίζει φίλτρα όπως ημερομηνίες/κείμενο).
`/api/v3/users`	GET	Όλοι οι χρήστες.
`/api/v3/users/{USER_ID}`	GET	Στοιχεία συγκεκριμένου χρήστη.
`/api/v3/users/{USER_ID}/groups`	GET	Ομάδες στις οποίες ανήκει ο χρήστης.
`/api/v3/users/{USER_ID}/announcements`	GET	Ανακοινώσεις που σχετίζονται με τον χρήστη.

Συνέχεια από την προηγούμενη σελίδα

Endpoint	Method	Περιγραφή
`/api/v3/groups`	GET	Όλες οι ομάδες.
`/api/v3/groups/{GROUP_ID}`	GET	Στοιχεία συγκεκριμένης ομάδας.
`/api/v3/groups/{GROUP_ID}/users`	GET	Χρήστες που ανήκουν στην ομάδα.
`/api/v3/groups/{GROUP_ID}/announcements`	GET	Ανακοινώσεις που ανήκουν στην ομάδα.
`/api/v3/groups/{GROUP_ID}/tags`	GET	Tags που σχετίζονται με την ομάδα.
`/api/v3/groups`	POST	Δημιουργία νέας ομάδας (όνομα, περιγραφή, προαιρετικό `parent_group`).
`/api/v3/groups/{GROUP_ID}`	PUT	Ενημέρωση στοιχείων ομάδας (π.χ. `name`, `description`, `parent_group`).
`/api/v3/groups/{GROUP_ID}`	DELETE	Διαγραφή ομάδας.
`/api/v3/user-group-roles`	POST	Δημιουργία σχέσης χρήστη–ομάδας με ρόλο (`user_id`, `group_id`, `role`).
`/api/v3/user-group-roles`	GET	Λίστα όλων των ρόλων χρηστών σε ομάδες.
`/api/v3/user-group-roles/{user_id}/{group_id}`	PUT	Ενημέρωση ρόλου χρήστη σε συγκεκριμένη ομάδα.
`/api/v3/user-group-roles/{user_id}/{group_id}`	DELETE	Αφαίρεση ρόλου/σχέσης χρήστη από συγκεκριμένη ομάδα.

3.4 Παράδειγμα Endpoint: GET Announcements

Το endpoint `/api/v3/announcements` επιστρέφει όλες τις ανακοινώσεις. Δέχεται μια σειρά από παραμέτρους φιλτραρίσματος:

Πίνακας 3.4: Διαθέσιμες παράμετροι για το endpoint GET /api/v3/announcements

Παράμετρος	Τύπος	Περιγραφή
title	string	Αναζήτηση με βάση τον τίτλο.
body	string	Αναζήτηση με βάση το σώμα της ανακοίνωσης.
page	integer	Σελίδα αποτελεσμάτων (π.χ. 1, 2, 3).
perPage	integer	Πόσες ανακοινώσεις επιστρέφονται ανά σελίδα.
tags[]	array(int)	Λίστα από IDs tags.

Συνέχεια από την προηγούμενη σελίδα

Παράμετρος	**Τύπος**	**Περιγραφή**
users[]	array(int)	Λίστα από IDs χρηστών (συγγραφέων).
updatedAfter	date (Y-m-d)	Ανακοινώσεις που ενημερώθηκαν μετά από αυτή την ημερομηνία.
updatedBefore	date (Y-m-d)	Ανακοινώσεις που ενημερώθηκαν πριν από αυτή την ημερομηνία.
sortId	integer	Τρόπος ταξινόμησης: 0 = Important first, 1 = Latest first, 2 = Oldest first.

Απάντηση:

Κώδικας 3.1: Παράδειγμα response του GET /api/v3/announcements

```
{
  "data": [
    {
      "id": 3,
      "title": "test2",
      "body": "<p>test</p>",
      "created_at": "2023-06-25 18:44",
      "updated_at": "2023-06-25 18:47",
      "tags": [
        { "id": 1, "title": Όλες" οιανακοινώσεις " },
        { "id": 9, "title": Εξάμηνο" Η" }
      ],
      "author": {
        "id": 1,
        "name": ΝΤΟΜΠΡΟΥΓΚΙΔΗΣ" ΧΡΗΣΤΟΣ"
      }
    }
  ],
  "meta": {
    "current_page": 1,
    "per_page": 20,
    "total": 55
  }
}
```

3.4.1 GET announcement by id

Το endpoint `/api/v3/announcements/{ANNOUNCEMENT_ID}` επιστρέφει μία συγκεκριμένη ανακοίνωση βάσει του μοναδικού της αναγνωριστικού. Η κλήση απαιτεί `Authorization: Bearer {TOKEN}`.

Πίνακας 3.5: Παράμετροι για το endpoint GET /api/v3/announcements/{ANNOUNCEMENT_ID}

Παράμετρος	**Τύπος**	**Περιγραφή**
ANNOUNCEMENT_ID	integer (path)	Το μοναδικό αναγνωριστικό της ανακοίνωσης που θα ανακτηθεί.

Απάντηση:

Κώδικας 3.2: Παράδειγμα response του GET /api/v3/announcements/{id}


```
{
  "data": {
    "id": 3,
    "title": "test2",
    "eng_title": "test2",
    "body": "<p>test</p>",
    "eng_body": "test2",
    "preview": "test",
    "eng_preview": "test2",
    "has_eng": null,
    "created_at": "2023-06-25 18:44",
    "updated_at": "2023-06-25 18:47",
    "is_pinned": 1,
    "pinned_until": "2023-06-28 00:00",
    "is_event": 1,
    "event_start_time": "2023-06-28 00:00",
    "event_end_time": "2023-06-29 00:00",
    "event_location": "test",
    "gmaps": null,
    "tags": [
      {
        "id": 1,
        "title": Όλες" οιανακοινώσεις ",
        "parent_id": null,
        "is_public": false,
        "maillist_name": null
      },
      {
        "id": 9,
        "title": "ξάμηνοΕ Η",
        "parent_id": 1,
        "is_public": false,
        "maillist_name": null
      }
    ],
    "attachments": [],
    "author": {
      "name": ΝΤΟΜΠΡΟΥΓΚΙΔΗΣ" ΧΡΗΣΤΟΣ",
      "id": 1
    },
    "announcement_url": "http://locahost:8000/announcements/3"
  }
}
```

3.5 Παράδειγμα Endpoint: GET filtertags

Το endpoint `/api/v3/filtertags` επιστρέφει όλα τα διαθέσιμα tags σε μορφή δενδροειδούς δομής (nested). Αυτό επιτρέπει την καλύτερη οργάνωση και φιλτράρισμα των ανακοινώσεων βάσει θεματικών κατηγοριών.

Πίνακας 3.6: Παράμετροι για το endpoint GET /api/v3/user-group-roles

Παράμετρος	**Τύπος**	**Περιγραφή**
–	–	Το endpoint δεν απαιτεί παραμέτρους.

Απάντηση:

Κώδικας 3.3: Παράδειγμα response του GET /api/v3/filtertags

```
[
  {
    "id": 1,
    "title": Όλες" οιανακοινώσεις ",
    "parent_id": null,
    "is_public": false,
    "maillist_name": null,
    "children": [
      {
        "id": 9,
        "title": Εξάμηνο" Η",
        "parent_id": 1,
        "is_public": false,
        "maillist_name": null,
        "children": []
      },
      {
        "id": 10,
        "title": Εξάμηνο" Θ",
        "parent_id": 1,
        "is_public": false,
        "maillist_name": null,
        "children": []
      }
    ]
  }
]
```

3.5.1 POST announcements

Το endpoint `/api/v3/announcements` επιτρέπει τη δημιουργία νέας ανακοίνωσης. Ο χρήστης πρέπει να είναι συνδεδεμένος και να διαθέτει δικαιώματα συγγραφέα.

Πίνακας 3.7: Διαθέσιμα πεδία για το σώμα αιτήματος (POST /api/v3/announcements)

Παράμετρος	**Τύπος**	**Περιγραφή**
title	string	Τίτλος της ανακοίνωσης.
body	string (HTML)	Το περιεχόμενο της ανακοίνωσης.
eng_title	string	Τίτλος στα Αγγλικά (προαιρετικό).
eng_body	string (HTML)	Περιεχόμενο στα Αγγλικά (προαιρετικό).
attachments[]	file	Συνημμένα αρχεία (προαιρετικό).
tags[]	array(int)	IDs tags στα οποία ανήκει η ανακοίνωση.
is_pinned	boolean/int	Αν η ανακοίνωση είναι καρφιτσωμένη.
pinned_until	datetime (Y-m-d H:i)	Ημερομηνία λήξης καρφιτσώματος.
is_event	boolean/int	Αν η ανακοίνωση αφορά εκδήλωση.
event_start_time	datetime (Y-m-d H:i)	Έναρξη εκδήλωσης.
event_end_time	datetime (Y-m-d H:i)	Λήξη εκδήλωσης.
event_location	string	Τοποθεσία εκδήλωσης.

Απάντηση:

Κώδικας 3.4: Παράδειγμα response του endpoint POST /api/v3/announcements

```
{
  "data": {
    "id": 3,
    "title": "test",
    "eng_title": "test",
    "body": "<p>test</p>",
    "eng_body": "test",
    "preview": "test",
    "is_pinned": 1,
    "pinned_until": "2023-06-28 00:00",
    "is_event": 1,
    "event_start_time": "2023-06-28 00:00",
    "event_end_time": "2023-06-29 00:00",
    "event_location": "test",
    "tags": [
      { "id": 1, "title": Όλες" οιανακοινώσεις " },
      { "id": 9, "title": Εξάμηνο" Η" }
    ],
    "attachments": [],
```

```
    "author": {
      "id": 1,
      "name": ΝΤΟΜΠΡΟΥΓΚΙΔΗΣ" ΧΡΗΣΤΟΣ"
    },
    "announcement_url": "http://localhost:8000/announcements/3"
  }
}
```

3.5.2 PUT announcements

Το endpoint `/api/v3/announcements/{id}` επιτρέπει την ενημέρωση υπάρχουσας ανακοίνωσης. Απαιτείται ο χρήστης να είναι ο συγγραφέας της ανακοίνωσης.

Πίνακας 3.8: Διαθέσιμα πεδία για το σώμα αιτήματος (PUT /api/v3/announcements/{id})

Παράμετρος	**Τύπος**	**Περιγραφή**
title	string	Νέος τίτλος της ανακοίνωσης.
body	string (HTML)	Νέο περιεχόμενο της ανακοίνωσης.
eng_title	string	Νέος τίτλος στα Αγγλικά (προαιρετικό).
eng_body	string (HTML)	Νέο περιεχόμενο στα Αγγλικά (προαιρετικό).
tags[]	array(int)	IDs tags της ανακοίνωσης.
is_pinned	boolean/int	Αν παραμένει καρφιτσωμένη η ανακοίνωση.
pinned_until	datetime (Y-m-d H:i)	Νέα ημερομηνία λήξης καρφιτσώματος.
is_event	boolean/int	Αν παραμένει εκδήλωση.
event_start_time	datetime (Y-m-d H:i)	Νέα ώρα έναρξης εκδήλωσης.
event_end_time	datetime (Y-m-d H:i)	Νέα ώρα λήξης εκδήλωσης.
event_location	string	Νέα τοποθεσία εκδήλωσης.

Απάντηση:

Κώδικας 3.5: Παράδειγμα response του endpoint PUT /api/v3/announcements/{id}

```
{
  "data": {
    "id": 3,
    "title": "test2",
    "eng_title": "test2",
    "body": "<p>test</p>",
    "eng_body": "test2",
    "is_pinned": 1,
    "pinned_until": "2023-06-28 00:00",
    "is_event": 1,
```

```
    "event_start_time": "2023-06-28 00:00",
    "event_end_time": "2023-06-29 00:00",
    "event_location": "test",
    "tags": [
      { "id": 1, "title": Όλες" οιανακοινώσεις " },
      { "id": 9, "title": Εξάμηνο" Η" }
    ],
    "author": {
      "id": 1,
      "name": ΝΤΟΜΠΡΟΥΓΚΙΔΗΣ" ΧΡΗΣΤΟΣ"
    },
    "announcement_url": "http://localhost:8000/announcements/3"
  }
}
```

3.6 Ομάδες (Groups)

Η ενότητα `groups` της API επιτρέπει τη δημιουργία, ανάκτηση, ενημέρωση και διαγραφή ιεραρχικών ομάδων (π.χ. Πανεπιστήμιο → Σχολή → Τμήμα). Τα διαθέσιμα endpoints που προκύπτουν από τη συλλογή Postman είναι τα εξής:

Πίνακας 3.9: Διαθέσιμα endpoints για τη διαχείριση ομάδων (groups)

Endpoint	**Method**	**Περιγραφή**
`/api/v3/groups`	GET	Επιστρέφει όλες τις ομάδες.
`/api/v3/groups/{id}`	GET	Επιστρέφει τα στοιχεία μιας συγκεκριμένης ομάδας.
`/api/v3/groups`	POST	Δημιουργεί νέα ομάδα.
`/api/v3/groups/{id}`	PUT	Ενημερώνει τα στοιχεία ομάδας.
`/api/v3/groups/{id}`	DELETE	Διαγράφει ομάδα.

3.6.1 GET `/api/v3/groups`

Επιστρέφει λίστα όλων των ομάδων. Δεν απαιτεί σώμα αιτήματος. Στη συλλογή Postman εμφανίζεται ως *fetch*.

Πίνακας 3.10: Παράμετροι για το endpoint GET /api/v3/groups

Παράμετρος	**Τύπος**	**Περιγραφή**
–	–	Δεν υπάρχουν παράμετροι query στην τρέχουσα υλοποίηση.

Κώδικας 3.6: Παράδειγμα response του endpoint GET /api/v3/groups

```
[
  {
    "id": 15,
    "name": "International Hellenic University",
    "description": "Root org",
    "parent_group": null,
    "created_at": "2025-01-10 12:00",
    "updated_at": "2025-01-20 09:30",
    "children_recursive": [],
    "announcements_count": 3
  }
]
```

3.6.2 GET `/api/v3/groups/{id}`

Επιστρέφει τα στοιχεία μίας συγκεκριμένης ομάδας με βάση το `id`. Στη συλλογή Postman εμφανίζεται ως *fetch by Id*

Πίνακας 3.11: Παράμετροι για το endpoint GET /api/v3/groups/{id}

Παράμετρος	**Τύπος**	**Περιγραφή**
`id`	integer (path)	Μοναδικό αναγνωριστικό της ομάδας.

Κώδικας 3.7: Παράδειγμα response του GET /api/v3/groups/{id}

```
{
  "id": 20,
  "name": "School of Science",
  "description": "Node under IHU",
  "parent_group": 15,
  "created_at": "2025-02-11 10:22",
  "updated_at": "2025-02-18 08:00",
  "children_recursive": []
}
```

3.6.3 POST `/api/v3/groups`

Δημιουργεί νέα ομάδα. Βλέπε *create* στη συλλογή Postman

Πίνακας 3.12: Πεδία σώματος αιτήματος για το endpoint POST /api/v3/groups

Πεδίο	**Τύπος**	**Περιγραφή**
`name`	string	Υποχρεωτικό. Το όνομα της ομάδας.
`description`	string	Προαιρετικό. Περιγραφή ομάδας.
`parent_group`	integer|null	Προαιρετικό. `id` γονικής ομάδας ή `null` για ρίζα.

Παράδειγμα:

Κώδικας 3.8: Παράδειγμα σώματος αιτήματος για POST /api/v3/groups

```
{
  "name": "AI Club",
  "description": "Subgroup for AI interests",
  "parent_group": 15
}
```

Κώδικας 3.9: Παράδειγμα response (201) του POST /api/v3/groups

```
{
  "data": {
    "id": 42,
    "name": "AI Club",
    "description": "Subgroup for AI interests",
    "parent_group": 15,
    "created_at": "2025-02-20 14:05",
    "updated_at": "2025-02-20 14:05"
  }
}
```

3.6.4 PUT `/api/v3/groups/{id}`

Ενημερώνει υπάρχουσα ομάδα. Στη συλλογή Postman εμφανίζεται ως *update by Id*

Πίνακας 3.13: Παράμετροι διαδρομής για το endpoint PUT /api/v3/groups/{id}

Παράμετρος	**Τύπος**	**Περιγραφή**
`id`	integer (path)	Μοναδικό αναγνωριστικό ομάδας.

Πίνακας 3.14: Πεδία σώματος αιτήματος για το endpoint PUT /api/v3/groups/{id}

Πεδίο	**Τύπος**	**Περιγραφή**
`name`	string	Νέο όνομα (προαιρετικό).

Συνέχεια από την προηγούμενη σελίδα

Πεδίο	Τύπος	Περιγραφή
`description`	string	Νέα περιγραφή (προαιρετικό).
`parent_group`	integer|null	Νέος γονέας ή `null`.

Παράδειγμα:

Κώδικας 3.10: Παράδειγμα σώματος αιτήματος για PUT /api/v3/groups/{id}

```
{
  "name": "AI Club (updated)",
  "description": "Updated description",
  "parent_group": 15
}
```

Κώδικας 3.11: Παράδειγμα response (200) του PUT /api/v3/groups/{id}

```
{
  "data": {
    "id": 42,
    "name": "AI Club (updated)",
    "description": "Updated description",
    "parent_group": 15,
    "created_at": "2025-02-20 14:05",
    "updated_at": "2025-02-21 10:12"
  }
}
```

3.6.5 DELETE `/api/v3/groups/{id}`

Διαγράφει ομάδα με βάση το `id`. Στη συλλογή Postman εμφανίζεται ως *delete by Id*.

Πίνακας 3.15: Παράμετροι διαδρομής για το endpoint DELETE /api/v3/groups/{id}

Παράμετρος	Τύπος	Περιγραφή
`id`	integer (path)	Μοναδικό αναγνωριστικό ομάδας για διαγραφή.

Κώδικας 3.12: Παράδειγμα response (ενδεικτικό) του DELETE /api/v3/groups/{id}

```
{
  "status": "deleted"
}
```

3.7 Σχέσεις Χρήστη–Ομάδας (user_has_group)

Η ενότητα `user_has_group` της API διαχειρίζεται τους ρόλους των χρηστών μέσα σε συγκεκριμένες ομάδες. Τα διαθέσιμα endpoints είναι:

Πίνακας 3.16: Διαθέσιμα endpoints για τη διαχείριση ρόλων σε ομάδες (user_has_group)

Endpoint	Method	Περιγραφή
`/api/v3/user-group-roles`	GET	Επιστρέφει όλους τους ρόλους χρηστών σε ομάδες.
`/api/v3/user-group-roles/{user_id}/{group_id}`	GET	Επιστρέφει τον ρόλο συγκεκριμένου χρήστη σε ομάδα.
`/api/v3/user-group-roles`	POST	Δημιουργεί νέο ρόλο χρήστη σε ομάδα.
`/api/v3/user-group-roles/{user_id}/{group_id}`	PUT	Ενημερώνει τον ρόλο χρήστη σε ομάδα.
`/api/v3/user-group-roles/{user_id}/{group_id}`	DELETE	Διαγράφει τη σχέση χρήστη–ομάδας.

3.7.1 GET `/api/v3/user-group-roles`

Επιστρέφει λίστα όλων των σχέσεων χρήστη–ομάδας και των αντίστοιχων ρόλων.

Πίνακας 3.17: Παράμετροι για το endpoint GET /api/v3/user-group-roles

Παράμετρος	Τύπος	Περιγραφή
–	–	Δεν υπάρχουν παράμετροι query στην τρέχουσα υλοποίηση.

Κώδικας 3.13: Παράδειγμα response του GET /api/v3/user-group-roles

```
[
  { "user_id": 2, "group_id": 5,  "role": "teacher", "created_at": "2025-02-20 12:01" },
  { "user_id": 2, "group_id": 10, "role": "admin",   "created_at": "2025-02-20 12:05" },
  { "user_id": 7, "group_id": 5,  "role": "user",    "created_at": "2025-02-21 09:10" }
]
```

3.7.2 GET `/api/v3/user-group-roles/{user_id}/{group_id}`

Επιστρέφει τη σχέση και τον ρόλο ενός συγκεκριμένου χρήστη σε μία συγκεκριμένη ομάδα.

Πίνακας 3.18: Παράμετροι για το endpoint GET /api/v3/user-group-roles/{user_id}/{group_id}

Παράμετρος	**Τύπος**	**Περιγραφή**
`user_id`	integer (path)	Το `id` του χρήστη.
`group_id`	integer (path)	Το `id` της ομάδας.

Κώδικας 3.14: Παράδειγμα response του GET /api/v3/user-group-roles/{user_id}/{group_id}

```
{
  "user_id": 2,
  "group_id": 5,
  "role": "teacher",
  "created_at": "2025-02-20 12:01",
  "updated_at": "2025-02-20 12:01"
}
```

3.7.3 POST `/api/v3/user-group-roles`

Δημιουργεί νέα σχέση χρήστη–ομάδας με συγκεκριμένο ρόλο.

Πίνακας 3.19: Πεδία σώματος αιτήματος για το endpoint POST /api/v3/user-group-roles

Πεδίο	**Τύπος**	**Περιγραφή**
`user_id`	integer	Υποχρεωτικό. Αναγνωριστικό χρήστη.
`group_id`	integer	Υποχρεωτικό. Αναγνωριστικό ομάδας.
`role`	enum	Υποχρεωτικό. Ένας από: `user`, `teacher`, `admin`, `PhD candidate`.

Παράδειγμα:

Κώδικας 3.15: Παράδειγμα αιτήματος για POST /api/v3/user-group-roles

```
{
  "user_id": 2,
  "group_id": 5,
  "role": "teacher"
}
```

Κώδικας 3.16: Παράδειγμα response (201) του POST /api/v3/user-group-roles

```
{
  "data": {
    "user_id": 2,
    "group_id": 5,
    "role": "teacher",
    "created_at": "2025-02-22 10:15",
```

```
    "updated_at": "2025-02-22 10:15"
  }
}
```

3.7.4 PUT /api/v3/user-group-roles/{user_id}/{group_id}

Ενημερώνει τον ρόλο ενός χρήστη σε συγκεκριμένη ομάδα.

Πίνακας 3.20: Παράμετροι διαδρομής για το endpoint PUT /api/v3/user-group-roles/{user_id}/{group_id}

Παράμετρος	Τύπος	Περιγραφή
`user_id`	integer (path)	Το `id` του χρήστη.
`group_id`	integer (path)	Το `id` της ομάδας.

Πίνακας 3.21: Πεδία σώματος αιτήματος για το endpoint PUT /api/v3/user-group-roles/{user_id}/{group_id}

Πεδίο	Τύπος	Περιγραφή
`role`	enum	Νέα τιμή ρόλου: `user`, `teacher`, `admin`, `PhD candidate`.

Παράδειγμα:

Κώδικας 3.17: Παράδειγμα αιτήματος για PUT /api/v3/user-group-roles/{user_id}/{group_id}

```
{
  "role": "admin"
}
```

Κώδικας 3.18: Παράδειγμα response (200) του PUT /api/v3/user-group-roles/{user_id}/{group_id}

```
{
  "data": {
    "user_id": 2,
    "group_id": 5,
    "role": "admin",
    "created_at": "2025-02-20 12:01",
    "updated_at": "2025-02-22 11:40"
  }
}
```

3.7.5 DELETE `/api/v3/user-group-roles/{user_id}/{group_id}`

Αφαιρεί μια σχέση χρήστη–ομάδας (και κατά συνέπεια τον ρόλο του χρήστη σε αυτή την ομάδα).

Πίνακας 3.22: Παράμετροι διαδρομής για το endpoint GET /api/v3/user-group-roles/{user_id}/{group_id}

Παράμετρος	**Τύπος**	**Περιγραφή**
`user_id`	integer (path)	Το `id` του χρήστη.
`group_id`	integer (path)	Το `id` της ομάδας.

Κώδικας 3.19: Παράδειγμα response του DELETE /api/v3/user-group-roles/{user_id}/{group_id}

```
{
  "status": "deleted"
}
```

Κεφάλαιο 4

Παρουσίαση της εφαρμογής

4.1 Εισαγωγή

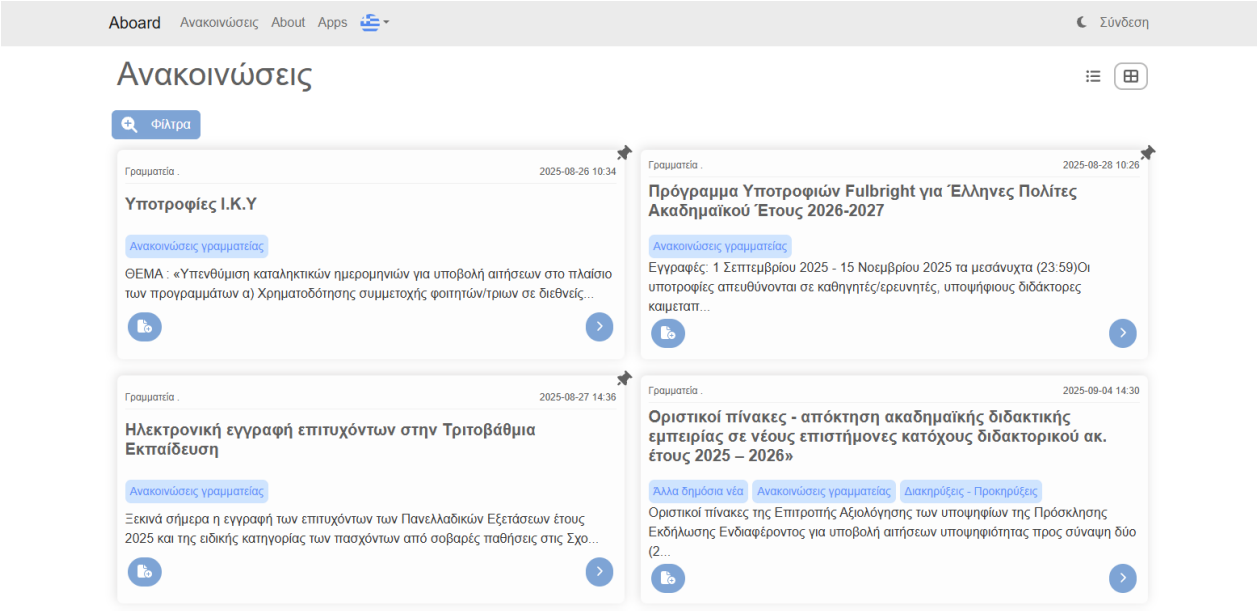
Αυτό το κεφάλαιο εξετάζει τη λειτουργικότητα της εφαρμογής από δύο διαφορετικές οπτικές: του επισκέπτη και του συνδεδεμένου χρήστη. Η παρουσίαση γίνεται μέσα από στιγμιότυπα της διεπαφής χρήστη, με παράλληλη αναφορά στις κλήσεις του API που υποστηρίζουν κάθε προβολή.

4.2 Ανακοινώσεις

Στην προβολή των ανακοινώσεων παρουσιάζεται η λίστα με όλες τις διαθέσιμες ανακοινώσεις, ταξινομημένες προεπιλεγμένα κατά ημερομηνία σε φθίνουσα σειρά, δίνοντας προτεραιότητα στις σημαντικές. Στο κάτω μέρος της σελίδας εμφανίζεται μηχανισμός σελιδοποίησης, με τον προεπιλεγμένο αριθμό ανακοινώσεων ανά σελίδα να έχει οριστεί στις δέκα.

Σε επίπεδο API, η προβολή αυτή βασίζεται στο endpoint:

- `GET /api/v3/announcements` με παραμέτρους `page` και `perPage`.



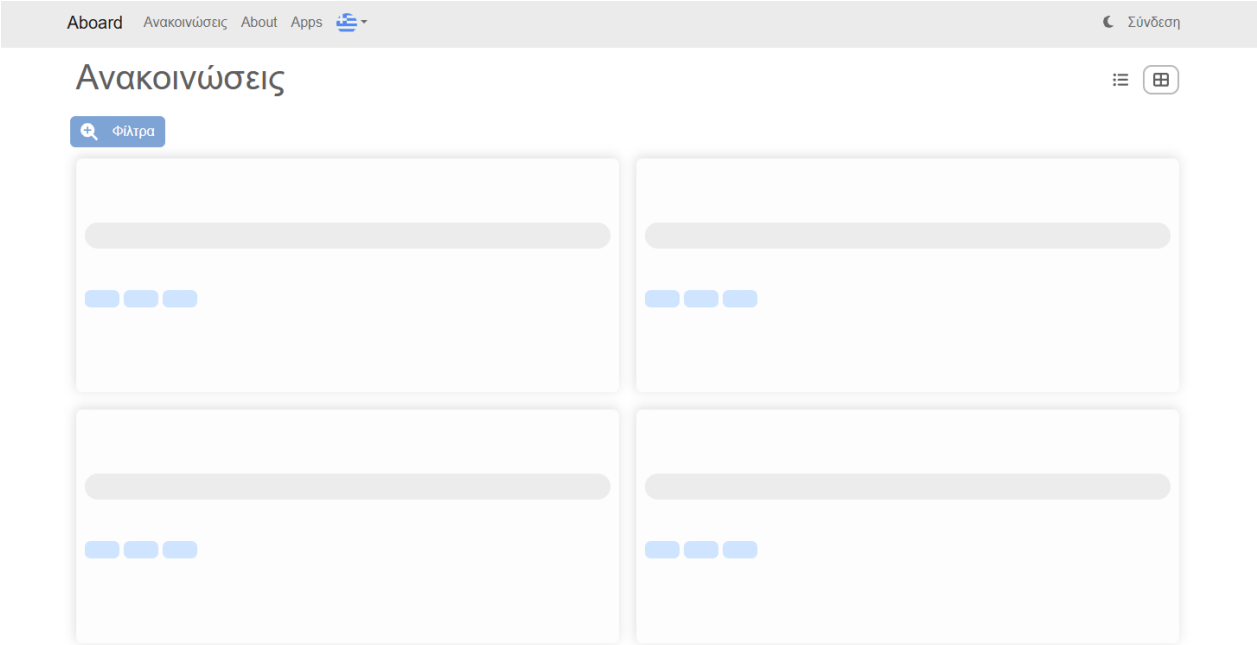
Σχήμα 4.1: Προβολή λίστας ανακοινώσεων με προεπιλεγμένη ταξινόμηση

4.2.1 Σκελετός φόρτωσης

Για την ομαλή εμπειρία περιήγησης έχει υλοποιηθεί η λειτουργία των “σκελετών” ανακοινώσεων, οι οποίοι προβάλλονται προσωρινά πριν από τη φόρτωση του κανονικού περιεχομένου. Η τεχνική αυτή είναι ευρέως διαδεδομένη (π.χ. YouTube, Facebook).

Κατά τη διάρκεια αυτή γίνεται αίτηση προς:

- `GET /api/v3/announcements`, με τα δεδομένα να αντικαθιστούν τους σκελετούς μόλις φορτωθούν.



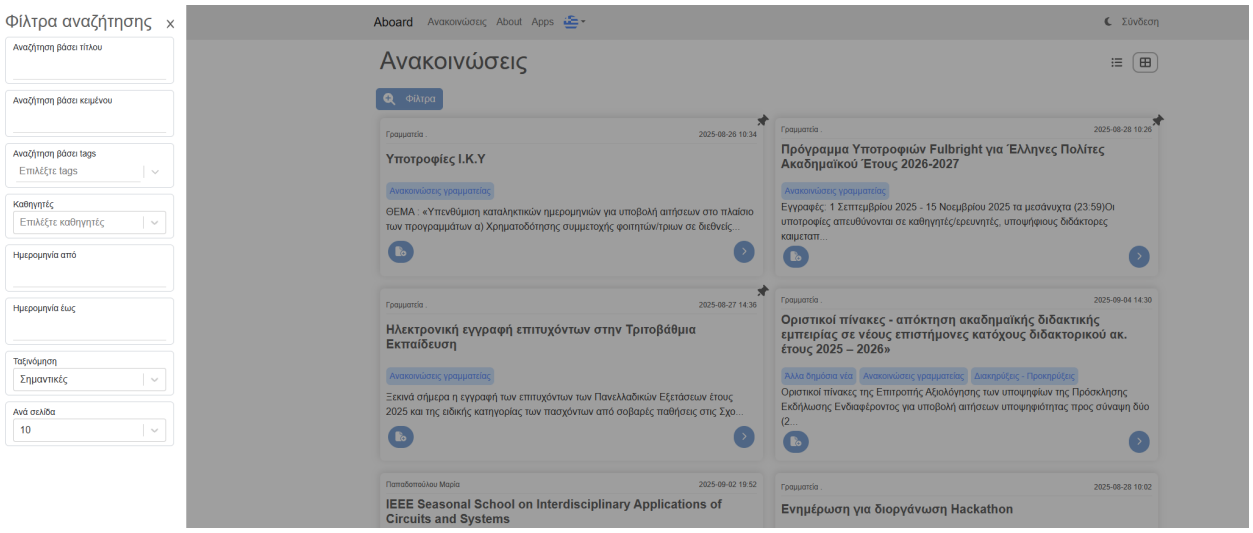
Σχήμα 4.2: Σκελετοί φόρτωσης πριν την εμφάνιση ανακοινώσεων

4.2.2 Φίλτρα ανακοινώσεων

Με την επιλογή του κουμπιού Φίλτρα, εμφανίζεται μενού που προσφέρει δυνατότητες φιλτραρίσματος των ανακοινώσεων. Ο χρήστης μπορεί να φιλτράρει με βάση τον τίτλο, το κείμενο, τα tags, τους διδάσκοντες και εύρος ημερομηνιών. Τα φίλτρα tags και διδάσκοντες υποστηρίζουν πολλαπλή επιλογή. Τέλος, στο ίδιο μενού παρέχονται επιλογές για την ταξινόμηση και τον αριθμό εμφανιζόμενων ανακοινώσεων ανά σελίδα.

Σε επίπεδο API, οι σχετικές κλήσεις είναι:

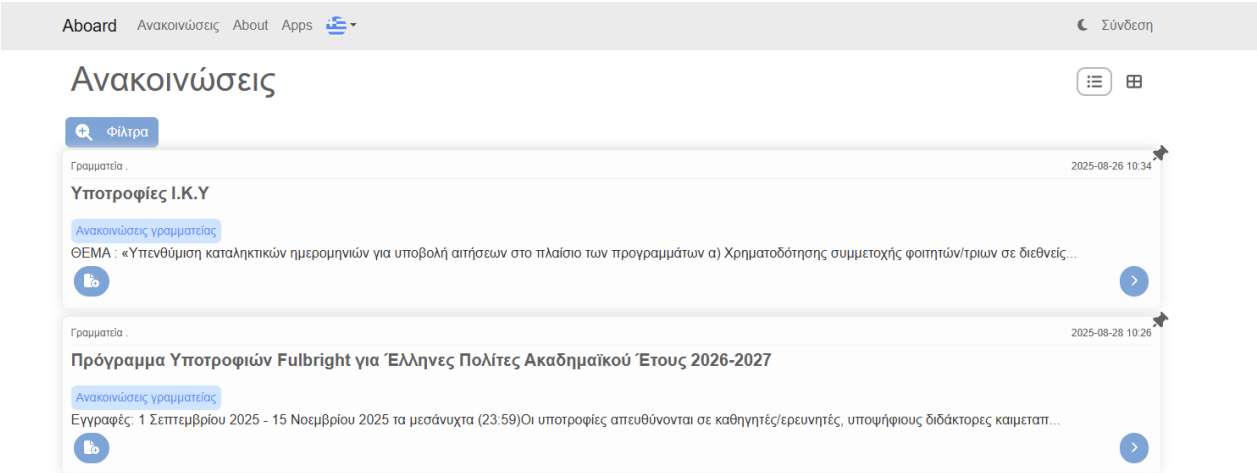
- `GET /api/v3/announcements` με query παραμέτρους (`title`, `tags[]`, `users[]`, `updatedAfter`, `updatedBefore`, `sortId`).
- `GET /api/v3/filtertags` για ανάκτηση διαθέσιμων tags.



Σχήμα 4.3: Μενού φίλτρων για αναζητήσεις στις ανακοινώσεις

4.2.3 Εμφάνιση ανακοινώσεων σε λίστα ή πλέγμα

Στο επάνω δεξί τμήμα της σελίδας βρίσκονται δύο κουμπιά, τα οποία επιτρέπουν την εναλλαγή του τρόπου προβολής των ανακοινώσεων: είτε σε μορφή λίστας είτε σε μορφή πλέγματος. Το API παραμένει το ίδιο (`GET /api/v3/announcements`), αλλά η παρουσίαση διαφοροποιείται στο frontend.



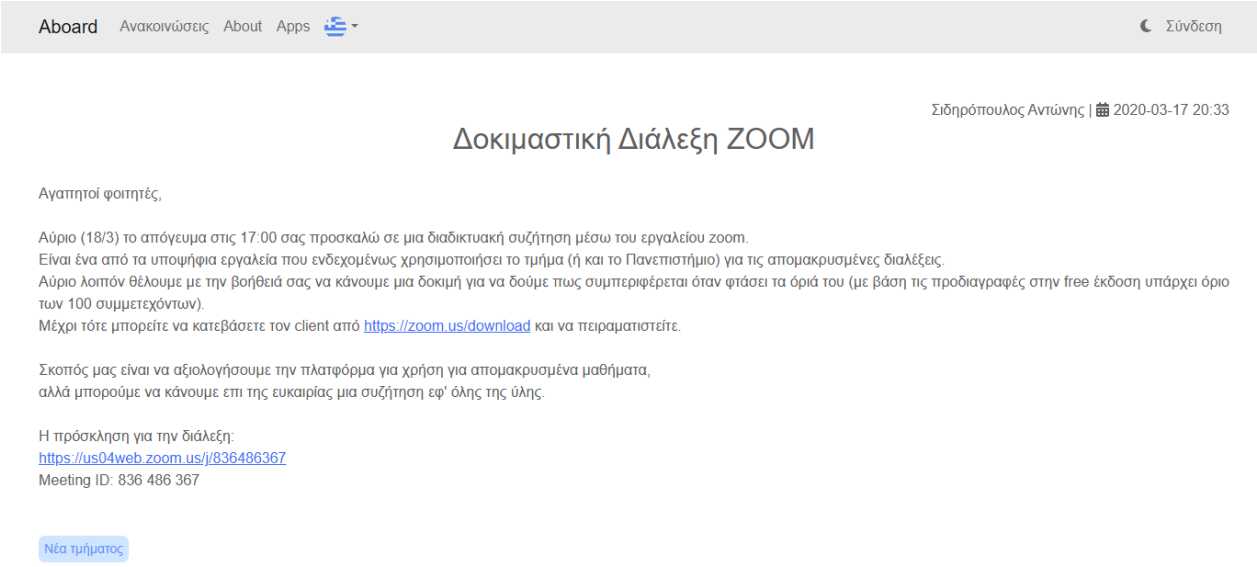
Σχήμα 4.4: Εναλλαγή τρόπου εμφάνισης ανακοινώσεων (λίστα ή πλέγμα)

4.2.4 Προβολή ανακοίνωσης

Η σελίδα προβολής μίας ανακοίνωσης εμφανίζει τον πλήρη τίτλο, το σώμα, τον συγγραφέα, τις ετικέτες και τυχόν συνημμένα αρχεία.

Η συγκεκριμένη προβολή βασίζεται στο endpoint:

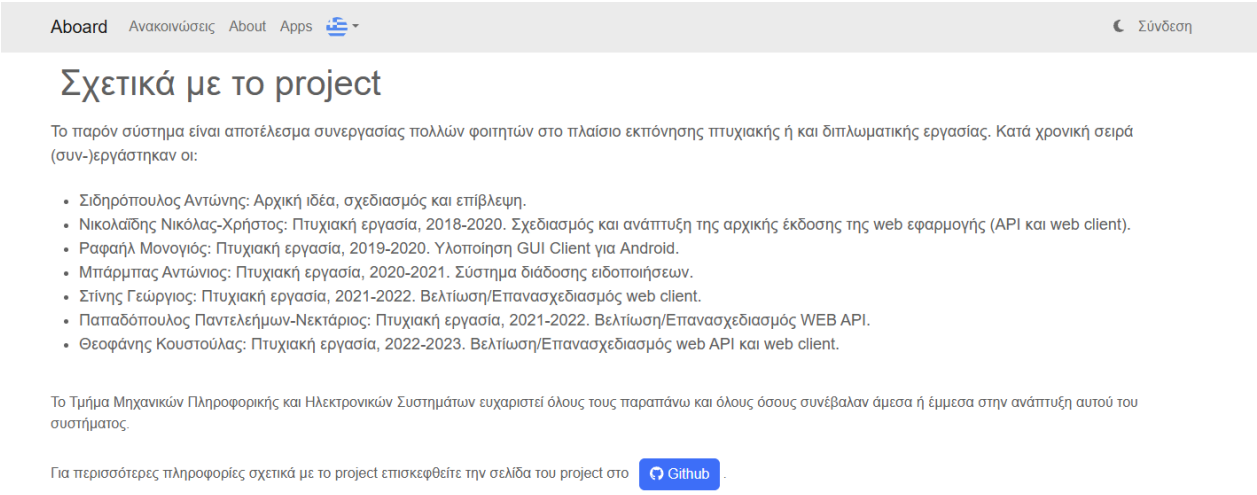
- `GET /api/v3/announcements/{id}`



Σχήμα 4.5: Σελίδα προβολής μιας ανακοίνωσης

4.2.5 Παρουσίαση σελίδας About

Η ενότητα About παρέχει πληροφορίες για το έργο, καθώς και τη λίστα με τα άτομα που εργάστηκαν και συνεργάστηκαν. Δεν απαιτεί κλήσεις προς το API, καθώς πρόκειται για στατικό περιεχόμενο.



Σχήμα 4.6: Σελίδα παρουσίασης πληροφοριών του έργου (About)

4.2.6 Συνδεδεμένος χρήστης

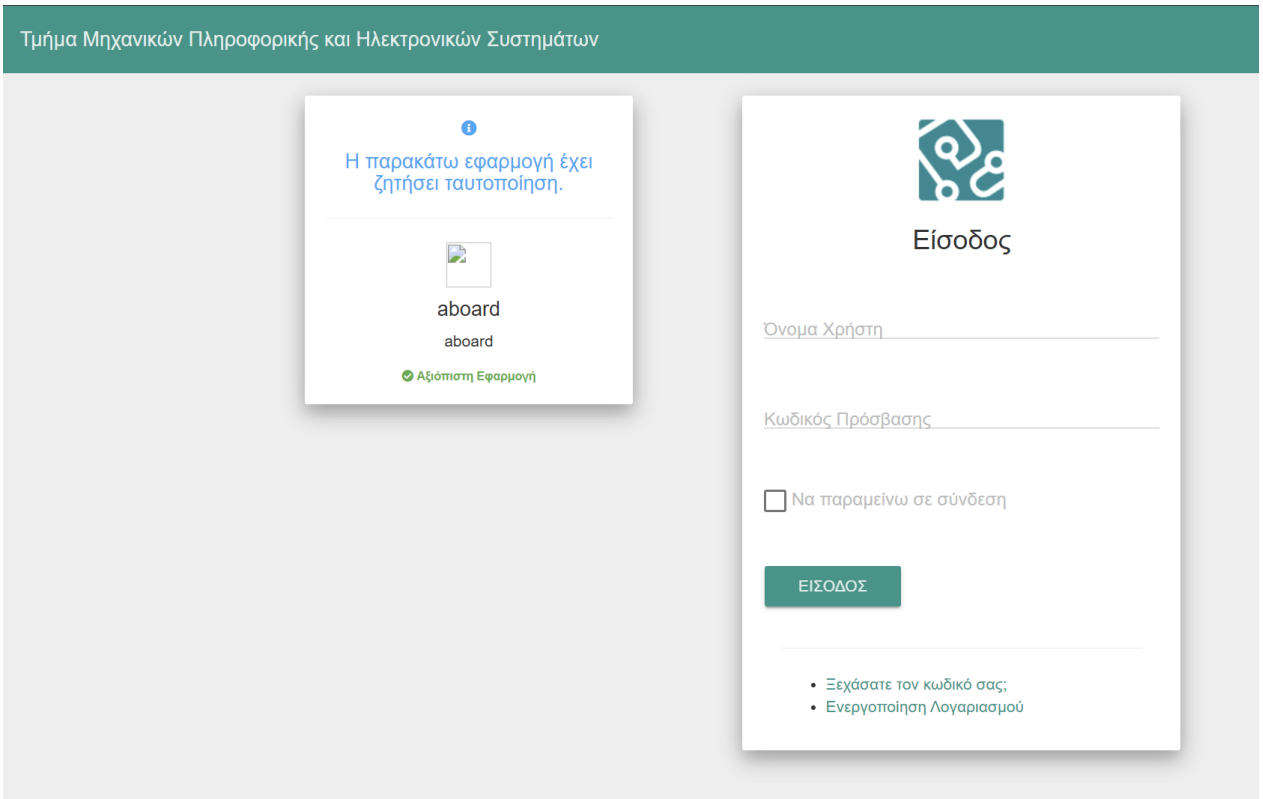
Ένας συνδεδεμένος χρήστης, χωρίς δικαιώματα author ή admin, έχει πρόσβαση σε όλες τις παραπάνω προβολές. Η μοναδική διαφορά εντοπίζεται στη λίστα των ανακοινώσεων, όπου εμφανίζονται και οι ανακοινώσεις που ανήκουν σε ιδιωτικές ετικέτες. Η αναγνώριση γίνεται με βάση το JWT token, το οποίο αποστέλλεται σε κάθε αίτηση στο API.

4.2.7 Σύνδεση του χρήστη

Η σελίδα σύνδεσης κατευθύνει τον χρήστη προς την υπηρεσία `login.iee.ihu.gr` (SSO). Μετά την επιτυχή αυθεντικοποίηση επιστρέφεται JWT token που χρησιμοποιείται σε όλες τις επόμενες αιτήσεις.

Βασικά endpoints:

- `POST /api/v3/auth/login` (SSO login).
- `GET /api/v3/auth/profile` (πληροφορίες συνδεδεμένου χρήστη).

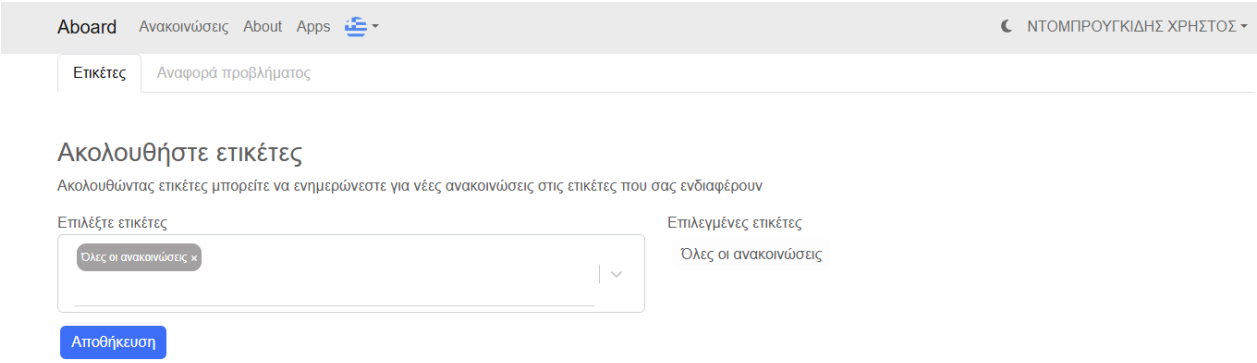


Σχήμα 4.7: Σελίδα σύνδεσης χρήστη μέσω SSO

4.2.8 Προσωπική σελίδα λογαριασμού

Η προσωπική σελίδα λογαριασμού εμφανίζει τα στοιχεία του συνδεδεμένου χρήστη, όπως όνομα, email και τις ομάδες στις οποίες ανήκει. Τα δεδομένα αντλούνται μέσω:

- `GET /api/v3/auth/profile`
- `GET /api/v3/user-group-roles/{user_id}/{group_id}`



Σχήμα 4.8: Προσωπική σελίδα λογαριασμού συνδεδεμένου χρήστη

Κεφάλαιο 5

Συμπεράσματα και προτάσεις

5.1 Εισαγωγή

Σε αυτό το κεφάλαιο παρουσιάζεται μια συνοπτική αποτίμηση της εφαρμογής και προτείνονται πιθανές μελλοντικές βελτιώσεις. Στη συνέχεια, αναλύονται τα τελικά συμπεράσματα που προέκυψαν από τον επανασχεδιασμό και την ανάπτυξή της.

5.2 Σύνοψη

Η εφαρμογή aboard αποτελεί ένα ολοκληρωμένο σύστημα διαχείρισης ανακοινώσεων, με κύριο σκοπό την έγκυρη ενημέρωση των φοιτητών. Στην αρχική της έκδοση είχε σχεδιαστεί για το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων, ενώ με τη νέα υλοποίηση επεκτάθηκε ώστε να μπορεί να υποστηρίζει ταυτόχρονα περισσότερα τμήματα και σχολές του ίδιου πανεπιστημίου.

Η εφαρμογή βασίζεται σε ένα API, το οποίο είναι υπεύθυνο για τη διαχείριση των πληροφοριών των χρηστών και των ρόλων τους, των ανακοινώσεων, των ετικετών, των συνδρομών σε ετικέτες καθώς και των συνημμένων αρχείων. Το API υλοποιεί πλήρεις λειτουργίες CRUD (Create, Read, Update, Delete) για όλες τις παραπάνω οντότητες και τις μεταξύ τους σχέσεις. Επιπλέον, ενσωματώνει μηχανισμό αυθεντικοποίησης μέσω των υπηρεσιών SSO του `login.iee.ihu.gr`, ενώ η ίδια η αυθεντικοποίηση είναι stateless, χρησιμοποιώντας JWT tokens για τη διατήρηση της σύνδεσης των χρηστών [11], [12].

Παράλληλα, η εφαρμογή περιλαμβάνει μία ιστοσελίδα σε ReactJS, η οποία αλληλεπιδρά με το API. Η ιστοσελίδα προσφέρει λίστα ανακοινώσεων με δυνατότητες φιλτραρίσματος, σελιδοποίησης και διαφορετικές μορφές παρουσίασης (λίστα ή πλέγμα). Οι χρήστες μπορούν να δηλώσουν ετικέτες για να ενημερώνονται άμεσα για νέες σχετικές ανακοινώσεις. Για τους authors παρέχεται ειδικό μενού διαχείρισης, μέσω του οποίου μπορούν να δημιουργούν, να επεξεργά-

ζονται και να διαγράφουν τις ανακοινώσεις τους. Τέλος, οι admins διαθέτουν πλήρη έλεγχο στο σύστημα, με δυνατότητα εισαγωγής, τροποποίησης, διαγραφής καθώς και μεταφοράς ανακοινώσεων από έναν author σε άλλον.

5.3 Προτάσεις βελτίωσης

Στην παρούσα ενότητα παρουσιάζονται ορισμένες προτάσεις που θα μπορούσαν να συμβάλουν στη μελλοντική βελτίωση της εφαρμογής, τόσο σε τεχνικό επίπεδο (υποδομή, λογισμικό) όσο και σε επίπεδο λειτουργικότητας και ποιότητας κώδικα.

5.3.1 Αναβάθμιση έκδοσης PHP

Η τρέχουσα υλοποίηση βασίζεται σε **PHP 7**, η οποία, αν και σταθερή, δεν υποστηρίζεται πλέον πλήρως με ενημερώσεις ασφαλείας και έχει πλέον φτάσει στο *end of life*. Αυτό σημαίνει ότι ενδέχεται να προκύψουν προβλήματα ασφάλειας και συμβατότητας με σύγχρονα εργαλεία ανάπτυξης, βιβλιοθήκες και frameworks.

Η αναβάθμιση σε **PHP 8.x** δεν αποτελεί μόνο ζήτημα ασφάλειας, αλλά και στρατηγική επιλογή για τη μακροχρόνια συντήρηση της εφαρμογής. Οι σημαντικότερες βελτιώσεις που φέρνει η νέα έκδοση περιλαμβάνουν:

- **Αυξημένη απόδοση:** Ο νέος Just-In-Time (JIT) compiler μειώνει τον χρόνο εκτέλεσης και βελτιώνει την ταχύτητα σε υπολογιστικά βαριές διεργασίες.
- **Νέες δυνατότητες στη γλώσσα:** Υποστήριξη για *union types*, *attributes (annotations)*, *constructor property promotion* και *named arguments*, που καθιστούν τον κώδικα πιο ευανάγνωστο, λιγότερο επαναλαμβανόμενο και πιο συντηρήσιμο.
- **Βελτιωμένη διαχείριση σφαλμάτων:** Εισαγωγή `ValueError` και `TypeError` exceptions σε περιπτώσεις που παλαιότερα επιστρέφονταν αθόρυβα `null` ή `false`, μειώνοντας έτσι τα κρυφά bugs.
- **Συμβατότητα με σύγχρονα frameworks:** Τα πιο πρόσφατα frameworks (π.χ. Laravel 9/10, Symfony 6) απαιτούν **PHP 8.x**, επομένως η αναβάθμιση είναι προϋπόθεση για μελλοντικές επεκτάσεις.
- **Βελτιώσεις ασφάλειας:** Νέα features και συνεχής υποστήριξη με security patches μειώνουν την έκθεση της εφαρμογής σε γνωστές επιθέσεις.

Συνολικά, η μετάβαση σε PHP 8 θα εξασφαλίσει ότι η εφαρμογή **aboard** παραμένει ασφαλής, ταχύτερη και πιο εύκολη στη συντήρηση, ενώ παράλληλα θα ανοίξει τον δρόμο για αξιοποίηση νεότερων βιβλιοθηκών και βέλτιστων πρακτικών ανάπτυξης [13].

5.3.2 Αναβάθμιση έκδοσης Laravel

Η εφαρμογή βασίζεται σε **Laravel**, το οποίο εξελίσσεται συνεχώς. Οι εκδόσεις **Laravel 9** και **Laravel 10** εισήγαγαν σημαντικές βελτιώσεις, όπως:

- Υποστήριξη για **PHP 8** και εκτεταμένη χρήση νέων χαρακτηριστικών της γλώσσας.
- Νέο **query builder interface** με βελτιωμένη υποστήριξη type-hinting, διευκολύνοντας την ανάπτυξη και μειώνοντας σφάλματα.
- Αναβαθμισμένο **Job batching** και βελτιώσεις στο σύστημα **queues**.
- Εκτεταμένη υποστήριξη για **test coverage** και εργαλεία debugging.
- Νέα χαρακτηριστικά στο **Eloquent ORM**, τα οποία απλοποιούν τη διαχείριση των σχέσεων και την εκτέλεση πολύπλοκων queries.
- Στη Laravel 10 προστέθηκε **native type declarations** σε όλο το framework, γεγονός που ενισχύει τη σταθερότητα και τη σαφήνεια του κώδικα.

Η μετάβαση σε νεότερη έκδοση θα ενισχύσει την ασφάλεια, την απόδοση και θα βελτιώσει τη δυνατότητα συντήρησης του έργου [14].

5.3.3 Προσθήκη test

Μία από τις πιο σημαντικές βελτιώσεις που θα μπορούσαν να υλοποιηθούν αφορά την προσθήκη αυτοματοποιημένων **tests**.

- Στο **backend**, με χρήση του PHPUnit και των εργαλείων που παρέχει το Laravel, μπορούν να γραφούν tests για endpoints του API, καλύπτοντας λειτουργίες CRUD και σενάρια ασφάλειας.
- Στο **frontend**, μπορούν να ενσωματωθούν βιβλιοθήκες όπως το Jest ή το React Testing Library, ώστε να διασφαλιστεί η σωστή λειτουργία των components και η εμπειρία του χρήστη.

Η ενσωμάτωση tests αυξάνει την αξιοπιστία του συστήματος, διευκολύνει την ανίχνευση σφαλμάτων και μειώνει τον κίνδυνο εμφάνισης regression bugs σε μελλοντικές αλλαγές.

5.4 Συμπεράσματα

Η εφαρμογή **aboard**, τόσο στην αρχική της μορφή όσο και στη νέα υλοποίηση, ακολουθεί καλές πρακτικές ανάπτυξης λογισμικού. Η προηγούμενη έκδοση είχε μια καθαρή δομή και σωστή

οργάνωση του κώδικα, κάτι που αποτέλεσε σταθερό θεμέλιο για τη συνέχιση και την επέκταση του έργου.

Με τον επανασχεδιασμό της βάσης δεδομένων και την εισαγωγή πιο ευέλικτης αρχιτεκτονικής, το σύστημα μπορεί πλέον να υποστηρίξει πολλαπλά τμήματα ή σχολές, αυξάνοντας σημαντικά τις δυνατότητές του. Παράλληλα, το API παραμένει συνεπές, ασφαλές και επεκτάσιμο, ενώ η διεπαφή ReactJS προσφέρει μια σύγχρονη και φιλική εμπειρία χρήσης.

Η εργασία ανέδειξε τη σημασία της σωστής αρχιτεκτονικής σχεδίασης, της χρήσης σύγχρονων τεχνολογιών και της προσαρμοστικότητας των πληροφοριακών συστημάτων στις πραγματικές ανάγκες των χρηστών.

5.5 Επίλογος

Συνοψίζοντας, η παρούσα εργασία παρουσίασε τον επανασχεδιασμό και την επέκταση της εφαρμογής ανακοινώσεων **aboard**, με στόχο την κάλυψη ευρύτερων αναγκών εντός του πανεπιστημίου. Ο συνδυασμός ενός σύγχρονου API, μιας δυναμικής διεπαφής χρήστη και μιας πιο ευέλικτης βάσης δεδομένων καθιστά την εφαρμογή κατάλληλη για υιοθέτηση από περισσότερα τμήματα και σχολές.

Η εμπειρία ανάπτυξης ανέδειξε τις προκλήσεις αλλά και τα οφέλη της συνεχούς βελτίωσης λογισμικού, επιβεβαιώνοντας ότι οι καλές πρακτικές στον αρχικό σχεδιασμό επιτρέπουν μελλοντικές επεκτάσεις χωρίς να απαιτείται πλήρης αναδόμηση. Μελλοντικά, η ενσωμάτωση των προτάσεων βελτίωσης θα ενισχύσει περαιτέρω την αξιοπιστία, την ασφάλεια και τη χρηστικότητα της εφαρμογής.

Βιβλιογραφία

[1] The PHP Group, *PHP Manual*, Accessed 2025-08-14, 2025. διεύθν.: https://www.php.net/manual/en/.

[2] M. Stauffer, *Laravel: Up & Running*, 2η έκδοση. O'Reilly Media, 2019, isbn: 978-1492041215.

[3] T. Otwell και L. Contributors, *Laravel Documentation*, Accessed 2025-08-14, 2025. διεύθν.: https://laravel.com/docs.

[4] T. Reenskaug κ.ά., *Model–View–Controller (MVC) Design Pattern*, Accessed 2025-08-14, 2025. διεύθν.: https://developer.mozilla.org/en-US/docs/Glossary/MVC.

[5] Composer Contributors, *Composer Documentation*, Accessed 2025-08-14, 2025. διεύθν.: https://getcomposer.org/doc/.

[6] Composer/Packagist Contributors, *Packagist - The PHP Package Repository*, Accessed 2025-08-14, 2025. διεύθν.: https://packagist.org.

[7] WHATWG, *HTML Standard*, Accessed 2025-08-18, 2025. διεύθν.: https://html.spec.whatwg.org/.

[8] W3C CSS Working Group, *CSS Cascading Style Sheets — W3C*, Accessed 2025-08-18, 2025. διεύθν.: https://www.w3.org/Style/CSS/.

[9] Oracle Corporation, *MySQL Documentation*, Accessed 2025-08-18, 2025. διεύθν.: https://dev.mysql.com/doc/.

[10] REST API Tutorial Contributors, *REST API Tutorial*, Accessed 2025-08-18, 2025. διεύθν.: https://restfulapi.net/.

[11] D. Hardt, *The OAuth 2.0 Authorization Framework*, IETF RFC 6749, 2012. διεύθν.: https://datatracker.ietf.org/doc/html/rfc6749.

[12] M. Jones, J. Bradley και N. Sakimura, *JSON Web Token (JWT)*, IETF RFC 7519, 2015. διεύθν.: https://datatracker.ietf.org/doc/html/rfc7519.

[13] The PHP Group, *PHP 8 Release Notes*, Accessed 2025-08-20, 2020. διεύθν.: https://www.php.net/releases/8.0/.

[14] T. Otwell και L. Contributors, *Laravel Release Notes*, Accessed 2025-08-20, 2025. διεύθν.: https://laravel.com/docs/releases.