ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόγραμμα Μεταπτυχιακών Σπουδών

«ΠΡΟΗΓΜΕΝΑ ΣΥΣΤΗΜΑΤΑ ΠΛΗΡΟΦΟΡΙΚΗΣ - ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ ΚΑΙ ΤΕΧΝΗΤΗΣ ΝΟΗΜΟΣΥΝΗΣ»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	Ανάπτυξη διαδικτυακής εφαρμογής «Κοινωνικό Δίκτυο για την Ψυχική Υγεία» με χρήση SpringBoot backend και Angular frontend (Ελληνικά) Development of web application “Social Media Network for Mental Health” using SpringBoot backend and Angular frontend (English)
Ονοματεπώνυμο Φοιτητή	Γαλιάτσος Κωνσταντίνος
Πατρώνυμο	Εμμανουήλ
Αριθμός Μητρώου	ΜΠΣΠ21009
Επιβλέπων	Ευθύμιος Αλέπης, Καθηγητής

Ημερομηνία Παράδοσης **Μάρτιος 2025**

Τριμελής εξεταστική επιτροπή

Ευθύμιος Αλέπης Καθηγητής	Μαρία Βίρβου Καθηγήτρια	Ευάγγελος Σακκόπουλος Αναπληρωτής Καθηγητής

Σύνοψη

Οι εφαρμογές κοινωνικών δικτύων έχουν καταφέρει να φέρουν σε επαφή σε κάποιο κοινό ενδιαφέρον ή πρόβλημα μεγάλο αριθμό ανθρώπων από διαφορετικά υπόβαθρα. Η παρούσα μελέτη εστιάζεται στην ανάπτυξη μιας διαδικτυακής εφαρμογή «Κοινωνικό Δίκτυο για την Ψυχική Υγεία».

Το Κοινωνικό Δίκτυο για την Ψυχική Υγεία σκοπεύει να προσφέρει έναν διαδικτυακό ασφαλή και υποστηρικτικό χώρο όπου άτομα που αντιμετωπίζουν ζητήματα με την ψυχική υγεία τους μπορούν να μοιράζονται εμπειρίες, να βρίσκουν υποστήριξη από ειδικούς και να αποκτούν πρόσβαση σε χρήσιμες πληροφορίες που ενισχύουν την ψυχική ευημερία τους.

Η εφαρμογή επιτρέπει στους χρήστες να δημιουργούν και να συμμετέχουν σε ομάδες, να δημιουργούν αναρτήσεις ανά ομάδα, να σχολιάζουν στις αναρτήσεις καθώς και να αντιδρούν στις αναρτήσεις και τα σχόλια με Like. Επιτρέπει επίσης σε ειδικούς να προσφέρουν τις γνώσεις τους πάνω στις αναρτήσεις, να προσφέρουν πληροφορίες καθώς και γραμμές επικοινωνίας στους χρήστες. Οι αναρτήσεις και τα σχόλια των ειδικών παρουσιάζονται με ξεχωριστό τρόπο.

Για την ασφάλεια των χρηστών έχουν υλοποιηθεί λειτουργίες, όπως η αναφορά αναρτήσεων και σχολίων, η από προεπιλογή απόκρυψη των στοιχείων του χρήστη.

Η εφαρμογή υλοποιήθηκε με την βάση δεδομένων MariaDB, το SpringBoot Framework με Java για το backend και Angular με TypeScript για το frontend.

Λέξεις – κλειδιά: Ανάπτυξη λογισμικού, Διαδικτυακή εφαρμογή, Κοινωνικό Δίκτυο, Ψυχική Υγεία, MariaDB, SpringBoot, Java, Angular, TypeScript

Abstract

Social media applications have succeeded in bringing together many people from different backgrounds around a common interest or problem. This study focuses on the development of a web application "Social Network for Mental Health".

The Social Network for Mental Health intends to provide an online safe and supportive space where people experiencing issues with their mental health can share experiences, find support from experts and access useful information that enhances their mental wellbeing.

The app allows users to create and participate in groups, create posts per group, comment on posts, and react to posts and comments with a Like. It also allows experts to offer their expertise on posts, offer information as well as contact lines to users. The posts and comments of experts are presented in a separate way.

For the security of users, functions such as reporting posts and comments, hiding user data by default.

The application was implemented with MariaDB database, SpringBoot Framework with Java for the backend and Angular with TypeScript for the frontend.

Keywords: Software development, Web application, Social Network, Mental Health MariaDB, SpringBoot, Java, Angular, TypeScript

Περιεχόμενα

Contents

Εισαγωγή

Καθώς ο κόσμος καλείται να ζήσει και να μάθει από τις εκτεταμένες επιπτώσεις της πανδημίας COVID-19, πρέπει όλοι να αναλογιστούμε μια από τις πιο εντυπωσιακές πτυχές της - το τεράστιο τίμημα που έχει πάρει στην ψυχική υγεία των ανθρώπων. Τα ποσοστά ήδη κοινών παθήσεων όπως η κατάθλιψη και το άγχος αυξήθηκαν κατά περισσότερο από 25% το πρώτο έτος της πανδημίας, προσθέτοντας στους σχεδόν ένα δισεκατομμύριο ανθρώπους που ζούσαν ήδη με ψυχική διαταραχή. Ταυτόχρονα, πρέπει να αναγνωρίσουμε την αδυναμία των συστημάτων υγείας που προσπαθούν να αντιμετωπίσουν τις ανάγκες των ατόμων με νεοεμφανιζόμενες αλλά και προϋπάρχουσες ψυχικές παθήσεις.

Η ψυχική υγεία είναι κάτι πολύ περισσότερο από την απουσία ασθένειας: είναι ένα εγγενές μέρος της ατομικής και συλλογικής υγείας και ευημερίας μας.

Το να έχει κανείς καλή ψυχική υγεία σημαίνει ότι είναι σε καλύτερη θέση να συνδεθεί, να λειτουργήσει, να ανταπεξέλθει και να ευδοκιμήσει. Η ψυχική υγεία υπάρχει σε ένα πολύπλοκο συνεχές, με εμπειρίες που κυμαίνονται από μια βέλτιστη κατάσταση ευημερίας έως εξουθενωτικές καταστάσεις μεγάλου πόνου και συναισθηματικού πόνου. Τα άτομα με προβλήματα ψυχικής υγείας είναι πιο πιθανό να βιώσουν χαμηλότερα επίπεδα ψυχικής ευεξίας, αλλά αυτό δεν συμβαίνει πάντα ή απαραίτητα.

Επειδή οι παράγοντες που καθορίζουν την ψυχική υγεία είναι πολυτομεακοί, οι παρεμβάσεις για την προώθηση και την προστασία της ψυχικής υγείας θα πρέπει επίσης να παρέχονται σε πολλούς τομείς. Και όταν πρόκειται για την παροχή φροντίδας, μια πολυτομεακή προσέγγιση είναι εξίσου απαραίτητη, επειδή τα άτομα με προβλήματα ψυχικής υγείας συχνά χρειάζονται υπηρεσίες και υποστήριξη που εκτείνονται πέρα από την κλινική θεραπεία.1

Μεθοδολογία

Σκοπός της εφαρμογής

Σκοπός αυτής της μελέτης είναι να αναπτύξει μια διαδικτυακή εφαρμογή για ένα κοινωνικό δίκτυο, το οποίο είναι εξειδικευμένο για χρήση σε ζητήματα ψυχική υγείας.

Η εφαρμογή αυτή δίνει την δυνατότητα σε χρήστες να συζητούν, να αναζητούν και να δίνουν λύσεις σε προβλήματά που αντιμετωπίζουν οι ίδιοι και άλλοι χρήστες.

Σύντομη περιγραφή της εφαρμογής

Σύμφωνα με τον σκοπό της εφαρμογής αναζητήθηκαν οι απαραίτητες λειτουργίες που θα πρέπει να έχει η εφαρμογή.

Αρχικά, η εφαρμογή αυτή απαιτεί σύνδεση με διαπιστευτήρια (όνομα χρήστη και κωδικός) επειδή πρώτον δεν είναι δυνατή η χρήση του από μη εγγεγραμμένους χρήστες και δεύτερον για την προσωποποίηση της εφαρμογής για κάθε χρήστη.

---

[1] World Health Organization. World mental health report: transforming mental health for all. Στο: https://iris.who.int/bitstream/handle/10665/356119/9789240049338-eng.pdf?sequence=1 (προσπελάστηκε στις 19/1/2025)

Κάθε χρήστης μπορεί να έχει διαφορετικούς ρόλους στην εφαρμογή και ο κάθε ρόλος έχει πρόσβαση σε διαφορετικές λειτουργίες. Οι ρόλοι είναι User (απλός χρήστης), Expert (ειδικός ψυχικής υγείας) και Administrator (Διαχειριστής).

Ο ρόλος User αλλά και οι υπόλοιποι, έχουν την δυνατότητα για την δημιουργία ομάδων ανά ζήτημα ή κατηγορία προβλήματος ψυχικής υγείας, όπου μπορούν να δημιουργούν αναρτήσεις και να πραγματοποιείται συζήτηση με σχόλια.

Κάθε χρήσης μπορεί να αντιδράσει θετικά σε μία ανάρτηση ή σχόλιο.

Κάθε χρήστης έχει την δυνατότητα να τροποποιήσει το προφίλ του συμπεριλαμβανομένων ενός σύντομου βιογραφικού – περιγραφής, και των ρυθμίσεων ιδιωτικότητας.

Υπάρχει η δυνατότητα ο χρήστης να κάνει «φίλο» έναν άλλον, δίνοντας έτσι την δυνατότητα να παρακολουθεί ο καθένας την δραστηριότητα του «φίλου» του.

Οι ειδικοί πάνω στην ψυχική υγεία, μπορούν να προσφέρουν την γνώση τους επίσης μέσω αναρτήσεων και σχολίων, αλλά και να αναρτούν πόρους, για παράδειγμα συνδέσμους σε ιστοσελίδες που περιέχουν πληροφορίες ή μελέτες για το ζήτημα, σε κάθε ομάδα. Μπορούν επίσης να αναρτούν στην αρχική σελίδα της εφαρμογής τηλεφωνικές γραμμές, email και τοποθεσίες για την άμεση πρόσβαση των χρηστών με βοήθεια εάν την χρειαστούν. Οι αναρτήσεις, τα σχόλια αλλά και το προφίλ των ειδικών παρουσιάζονται με ξεχωριστό τρόπο, κάνοντας γνωστή την ιδιότητά του στους υπόλοιπους χρήστες.

Η διασφάλιση των δεδομένων των χρηστών γίνεται με πολλούς τρόπους. Πρώτον, μόνο το όνομα χρήστη καθώς και ένα σύντομο βιογραφικό, εάν το συντάξει ο χρήστης, είναι ορατά στους υπόλοιπους χρήστες της εφαρμογής. Ο κάθε χρήστης έχει την δυνατότητα να κάνει ορατή την δραστηριότητά του στην εφαρμογή (αναρτήσεις και σχόλια) σύμφωνα με τις εξής επιλογές: α. σε κανέναν, β. μόνο σε φίλους και γ. σε όλους. Αρχικά η πρώτη επιλογή ισχύει.

Για να διασφαλιστεί ότι οι πληροφορίες που αναρτώνται δεν είναι επιβλαβείς, δεν διαδίδει παραπληροφόρηση και δεν παραβιάζει τις κατευθυντήριες γραμμές της κοινότητας, έχει υλοποιηθεί η δυνατότητα αναφοράς αναρτήσεων και σχολίων. Ο διαχειριστής της ιστοσελίδας μπορεί να πραγματοποιήσει ενέργειες πάνω στις αναφορές αυτές και είτε να διαγράψει την ανάρτηση ή σχόλιο είτε να αγνοήσει την αναφορά.

Ο διαχειριστής επίσης βλέπει στατιστικά χρήσης της εφαρμογής.

Τεχνολογίες και τεχνικές που χρησιμοποιήθηκαν

Περίληψη τεχνολογιών

Η εφαρμογή έχει την αρχιτεκτονική client-server, με την εφαρμογή να έχει τον ρόλο του server και οι φυλλομετρητές χρηστών, τον ρόλο του client.

Το κομμάτι του server χωρίζεται σε βάση δεδομένων, εφαρμογή backend και εφαρμογή frontend.

Η τεχνολογία της βάσης δεδομένων είναι MariaDB, η backend εφαρμογή είναι σε SpringBoot Framework με χρήση της γλώσσας Java, έχει χρησιμοποιηθεί η αρχιτεκτονική ανάπτυξης λογισμικού MVC (Model – View – Controller) και η εφαρμογή frontend είναι γραμμένη στο Framework Angular με χρήστη της γλώσσας Typescript.

Η επικοινωνία μεταξύ backend και frontend πραγματοποιείται μέσω κλήσεων HTTP, στις οποίες το frontend ζητάει ένα URI. Οι κλήσεις αυτές μπορεί να είναι HTTP GET, POST, ή DELETE. Το GET χρησιμοποιείται για λήψη πληροφοριών, το POST για την δημιουργία πόρων, και το DELETE για διαγραφή πόρων. Στο POST το frontend συμπεριλαμβάνει και δεδομένα, αναφερόμενα και ως φορτίο, στην κλήση, τα στοιχεία των δεδομένων αυτών χρησιμοποιούνται

για την δημιουργία πόρων. Σε όλες τις περιπτώσεις στις κλήσεις HTTP συμπεριλαμβάνεται και το JWT token για πιστοποίηση του χρήστη και του ρόλου του.

MariaDB

Η βάση δεδομένων MariaDB χρησιμοποιήθηκε για την αποθήκευση των δεδομένων της εφαρμογής, τα οποία θα παρουσιαστούν παρακάτω.

Το MariaDB είναι μια σχεσιακή βάση δεδομένων γενικής χρήστης, ανοικτού κώδικα. Είναι ένας από τους πιο δημοφιλείς διακομιστές βάσεων δεδομένων στον κόσμο, με αξιοσημείωτους χρήστες όπως η Wikipedia, η WordPress.com και η Google. Μπορεί να χρησιμοποιηθεί για δεδομένα συναλλαγών υψηλής διαθεσιμότητας, αναλυτικά στοιχεία, ως ενσωματωμένος διακομιστής και ένα ευρύ φάσμα εργαλείων και εφαρμογών που υποστηρίζουν τον διακομιστή MariaDB.2

Η συγκεκριμένη βάση δεδομένων επιλέχθηκε λόγω της συμβατότητας με το περιβάλλον SpringBoot.

Java

Η Java είναι μια γλώσσα Write On One Platform και Run on Many Platforms (WORM). Για εφαρμογές φιλικές προς το δίκτυο, ανεξάρτητες από πλατφόρμα, η Java είναι μια αντικειμενοστρεφής γλώσσα προγραμματισμού. Ο πηγαίος κώδικας Java μεταγλωττίζεται σε κώδικα εικονικής μηχανής ή bytecode. Αυτό καθιστά την πλατφόρμα Java ανεξάρτητη. Μπορεί να τοποθετηθεί σε μια τοποθεσία web και να εκτελεστεί στην πλευρά του πελάτη σε υπολογιστή PC-Intel, Mac-Motorola ή UNIX-Solaris χωρίς επαναμεταγλώττιση. Η Sun Microsystems ανακοίνωσε επίσημα την Java τον Μάιο του 1995. Η Java είναι η πρώτη γλώσσα που έχει ενσωματωμένες δυνατότητες για εφαρμογές δικτύωσης, ιδίως για τη δημιουργία δυναμικών ιστοσελίδων. Η Java μειώνει το κόστος ανάπτυξης και επιταχύνει την καμπύλη μάθησης. Ως αποτέλεσμα, τα παραδοσιακά εργαλεία ανάπτυξης πελάτη-διακομιστή, όπως οι Delphi, Power-Builder και Visual Basic, χάνουν έδαφος έναντι της Java. Μέχρι το τέλος του 1996, η Java είχε προχωρήσει στη χρήση τόσο της C όσο και της C++ και αυτών των εργαλείων ανάπτυξης εφαρμογών.3

TypeScript

Παρά την επιτυχία της, η JavaScript παραμένει μια κακή γλώσσα για την ανάπτυξη και τη συντήρηση μεγάλων εφαρμογών. Η TypeScript είναι μια επέκταση της JavaScript που προορίζεται για την αντιμετώπιση αυτής της παραβίασης. Συντακτικά, η TypeScript είναι ένα υπερσύνολο του EcmaScript 5, επομένως κάθε πρόγραμμα JavaScript είναι ένα πρόγραμμα TypeScript. Η TypeScript εμπλουτίζει τη JavaScript με ένα σύστημα μονάδων, κλάσεων, διεπαφών και ένα σύστημα στατικού τύπου. Καθώς η TypeScript στοχεύει να παρέχει ελαφριά βοήθεια στους προγραμματιστές, το σύστημα μονάδων και το σύστημα τύπων είναι ευέλικτα και εύχρηστα. Συγκεκριμένα, υποστηρίζουν πολλές κοινές πρακτικές προγραμματισμού JavaScript. Επιτρέπουν επίσης εμπειρίες εργαλείων και IDE που είχαν συσχετιστεί προηγουμένως με γλώσσες όπως η C και η Java. Για παράδειγμα, οι τύποι βοηθούν να συλληφθούν τα λάθη στατικά και επιτρέπουν άλλη υποστήριξη για την ανάπτυξη προγράμματος (για παράδειγμα, προτείνοντας ποιες μεθόδους θα μπορούσαν να καλούνται σε ένα αντικείμενο). Η υποστήριξη για

---

[2] MariaDB in brief. Στο https://mariadb.org/en/ (Προσπελάστηκε στις 21/1/2025)

[3] Sabharwal, C. L. (1998). Java, java, java. *IEEE Potentials*, *17*(3), 33-37.

τις κλάσεις ευθυγραμμίζεται με προτάσεις που τυποποιούνται επί του παρόντος για το EcmaScript 6.

Ο μεταγλωττιστής TypeScript ελέγχει τα προγράμματα TypeScript και εξάγει JavaScript, έτσι ώστε τα προγράμματα να μπορούν να εκτελούνται αμέσως σε μια τεράστια γκάμα περιβαλλόντων εκτέλεσης. Ο μεταγλωττιστής χρησιμοποιείται εκτενώς στη Microsoft για τη δημιουργία σημαντικών εφαρμογών με JavaScript.4

SpringBoot Framework

Το Java Spring Framework (Spring Framework) είναι ένα δημοφιλές, ανοιχτού κώδικα framework για τη δημιουργία αυτόνομων εφαρμογών που εκτελούνται στην εικονική μηχανή της Java (JVM). Το Spring Boot βελτιστοποιεί και απλοποιεί την ανάπτυξη του Spring Framework μέσω τριών βασικών χαρακτηριστικών:

1. Αυτόματη διαμόρφωση
2. Μια προσέγγιση γνώμης στη διαμόρφωση
3. Δυνατότητα δημιουργίας αυτόνομων εφαρμογών

Το Spring Framework προσφέρει λειτουργικότητα dependency injection που επιτρέπει σε κάθε αντικείμενο να ορίζει τις δικές του εξαρτήσεις τις οποίες το Spring Framework εισάγει αργότερα σε αυτά. Αυτή η λειτουργία επιτρέπει σε προγραμματιστές να δημιουργήσουν αρθρωτές εφαρμογές αποτελούμενες από χαλαρά συνδεδεμένα στοιχεία που είναι ιδανικά για μικροϋπηρεσίες και κατανεμημένες εφαρμογές δικτύου.

Το Spring Framework προσφέρει επίσης ενσωματωμένη υποστήριξη για τυπικές εργασίες που πρέπει να εκτελέσει μια εφαρμογή, όπως σύνδεση δεδομένων, μετατροπή τύπων, επικύρωση, χειρισμός εξαιρέσεων, διαχείριση πόρων και συμβάντων, διεθνοποίηση και πολλά άλλα. Ενσωματώνεται με διάφορες τεχνολογίες Java EE όπως RMI (Remote Method Invocation), AMQP (Advanced Message Queuing Protocol), Java Web Services και άλλες.

Όσο ικανό και περιεκτικό και αν είναι το Spring Framework, εξακολουθεί να απαιτεί σημαντικό χρόνο και γνώση για τη διαμόρφωση, τη ρύθμιση και την ανάπτυξη εφαρμογών Spring. Το Spring Boot μετριάζει αυτή την προσπάθεια με τρεις σημαντικές δυνατότητες.

Η αυτόματη ρύθμιση παραμέτρων προετοιμάζει εφαρμογές με προκαθορισμένες εξαρτήσεις, ώστε να μην χρειάζονται ρύθμιση οι παράμετροι με μη αυτόματο τρόπο. Το Java Spring Boot διαθέτει ενσωματωμένες δυνατότητες αυτόματης διαμόρφωσης, οι οποίες διαμορφώνουν αυτόματα τόσο το βασικό Spring Framework όσο και τα πακέτα τρίτων με βάση τις ρυθμίσεις. Αυτή η προσέγγιση, που βασίζεται στις βέλτιστες πρακτικές, βοηθά στην αποφυγή σφαλμάτων.5

Angular

Το Angular είναι ένα web framework που δίνει τη δυνατότητα στους προγραμματιστές να δημιουργούν γρήγορες, αξιόπιστες εφαρμογές.

---

[4] Bierman, G., Abadi, M., & Torgersen, M. (2014). Understanding typescript. Στο *ECOOP 2014–Object-Oriented Programming: 28th European Conference, Uppsala, Sweden, July 28–August 1, 2014. Proceedings 28* (pp. 257-281). Springer Berlin Heidelberg. σ. 1

[5] What is Java Spring Boot? Στο https://www.ibm.com/think/topics/java-spring-boot (Προσπελάστηκε στις 21/1/2025)

Συντηρούμενο από μια ειδική ομάδα της Google, το Angular παρέχει μια ευρεία σειρά εργαλείων, API και βιβλιοθηκών για την απλοποίηση και τον εξορθολογισμό της ροής εργασιών ανάπτυξης. Το Angular προσφέρει μια σταθερή πλατφόρμα στην οποία μπορεί κανείς να δημιουργήσει γρήγορες, αξιόπιστες εφαρμογές που κλιμακώνονται τόσο με το μέγεθος της ομάδας όσο και με το μέγεθος του κώδικα.

Τα components της Angular διευκολύνουν τον διαχωρισμό του κώδικα σε καλά ενσωματωμένα μέρη.

Το ευέλικτο dependency injection βοηθά να διατηρηθεί ο κώδικας αρθρωτός, χαλαρά συζευγμένος και ελεγχόμενος.6

Object Relational Mapping – ORM

Το Object Relational Mapping (ORM) παρέχει μια μεθοδολογία και έναν μηχανισμό για τα αντικειμενοστραφή συστήματα ώστε να διατηρούν τα μακροπρόθεσμα δεδομένα τους με ασφάλεια σε μια βάση δεδομένων, με έλεγχο των συναλλαγών πάνω σε αυτά, αλλά να τα εκφράζουν όταν χρειάζεται σε αντικείμενα του προγράμματος. Αντί για δέσμες ειδικού κώδικα για το σκοπό αυτό, το ORM ενθαρρύνει τα μοντέλα και τη χρήση περιορισμών για την εφαρμογή, η οποία στη συνέχεια εκτελείται σε ένα πλαίσιο που έχει δημιουργηθεί από το ORM. Οι σημερινές διαδικτυακές εφαρμογές είναι ιδιαίτερα κατάλληλες για αυτή την προσέγγιση, καθώς είναι αναγκαστικά πολυνηματικές και συνεπώς επιρρεπείς σε συνθήκες ανταγωνισμού, εκτός εάν η αλληλεπίδραση με τη βάση δεδομένων υλοποιηθεί πολύ προσεκτικά. Η προσέγγιση ORM υλοποιήθηκε για πρώτη φορά στο Hibernate, ένα έργο ανοικτού κώδικα για συστήματα Java που ξεκίνησε το 2002, και φέτος (2008) προστέθηκε το Entity Data Model της Microsoft για συστήματα .NET.7

JWT

Το JSON Web Token (JWT) είναι ένα συμπαγές, ασφαλές μέσο για την αναπαράσταση αξιώσεων που πρέπει να μεταφερθούν μεταξύ δύο μερών. Οι ισχυρισμοί σε ένα JWT κωδικοποιούνται ως αντικείμενο JSON που χρησιμοποιείται ως ωφέλιμο φορτίο ενός JSON Web Signature (JWS) ή ως απλό κείμενο ενός JSON Web κρυπτογράφησης (JWE), επιτρέποντας την ψηφιακή επεξεργασία των αξιώσεων υπογραφή ή προστασία ακεραιότητας με έναν κωδικό ελέγχου ταυτότητας μηνύματος (MAC) ή/και κρυπτογραφημένα.

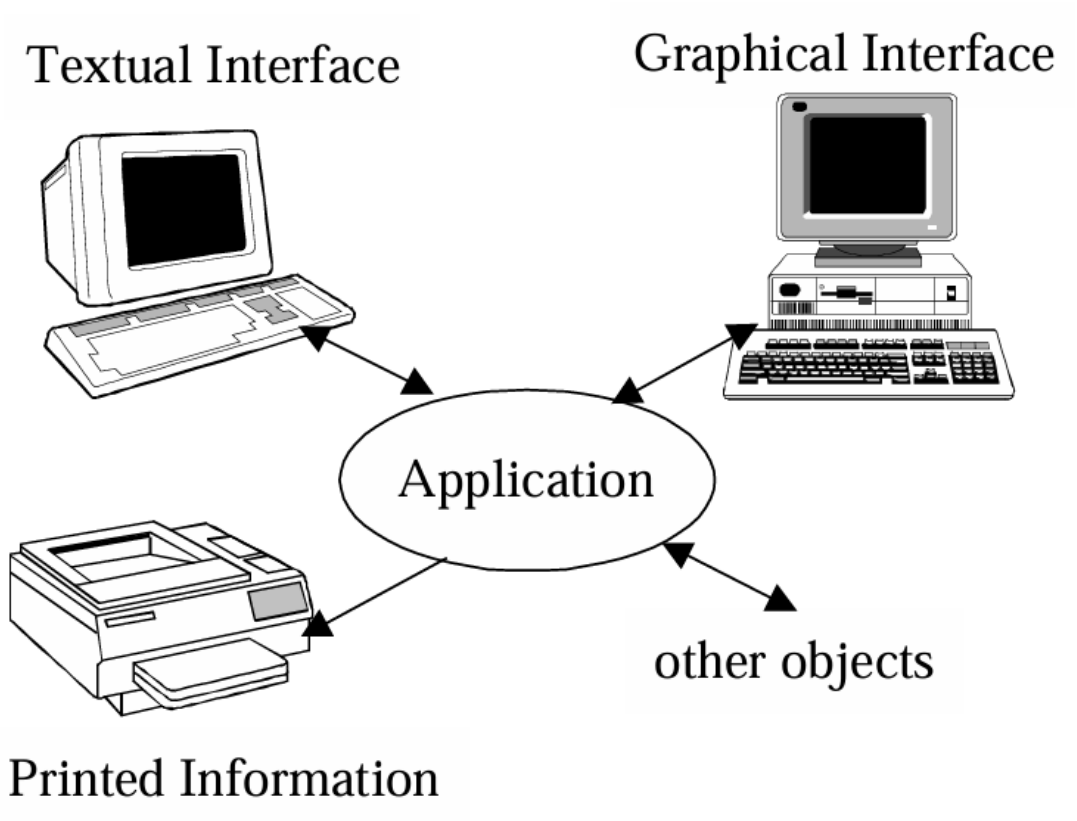
Αρχιτεκτονική Model – View – Controller (MVC)

Η αρχιτεκτονική Model – View – Controller βασίζεται στην έννοια του διαχωρισμού μιας εφαρμογής από τη διεπαφή χρήστη. Αυτό σημαίνει ότι διαφορετικές διεπαφές μπορούν να χρησιμοποιηθούν με την ίδια εφαρμογή, χωρίς η εφαρμογή να το γνωρίζει. Η πρόθεση είναι ότι οποιοδήποτε μέρος του συστήματος μπορεί να αλλάξει χωρίς να επηρεαστεί η λειτουργία του άλλου. Για παράδειγμα, στο Σχήμα 1, ο τρόπος με τον οποίο η γραφική διεπαφή εμφανίζει τις πληροφορίες θα μπορούσε να αλλάξει χωρίς τροποποίηση της πραγματικής εφαρμογής ή της διεπαφής κειμένου. Πράγματι, η εφαρμογή δεν χρειάζεται να γνωρίζει καθόλου τι είδους διεπαφή είναι συνδεδεμένη αυτήν τη στιγμή.

---

[6]What is Angular? Στο https://angular.dev/overview (Προσπελάστηκε στις 22/1/2025)

[7] O'Neil Elizabeth J. (2008). Object/relational mapping 2008: hibernate and the entity data model (edm). In Proceedings of the 2008 ACM SIGMOD international conference on Management of data (SIGMOD '08), New York, NY, USA, Association for Computing Machinery 1351–1356.



Σχήμα 1 Διαχωρίζοντας την διεπαφή

Η πρόθεση της αρχιτεκτονικής MVC είναι να διαχωρίσει τη διεπαφή χρήστη από το υποκείμενο μοντέλο πληροφοριών. Υπάρχουν διάφοροι λόγοι για τους οποίους αυτό είναι χρήσιμο:

- επαναχρησιμοποίηση στοιχείων εφαρμογής ή/και διεπαφής χρήστη,
- δυνατότητα ανάπτυξης της εφαρμογής και της διεπαφής χρήστη ξεχωριστά,
- ικανότητα κληρονομιάς από διαφορετικά μέρη της ιεραρχίας της κλάσης.

Ο καταμερισμός ευθύνης στο MVC αναλύεται ως εξής:

- • Model - Το μοντέλο πληροφοριών που χειρίζεται την αποθήκευση δεδομένων και την επεξεργασία πληροφοριών. Δηλαδή, διαχειρίζεται τη συμπεριφορά των δεδομένων στον τομέα της εφαρμογής.
- • View - που χειρίζεται την οπτική οθόνη (το τμήμα εξόδου). Δηλαδή, χειρίζεται τον τρόπο με τον οποίο εμφανίζονται οι πληροφορίες σχετικά με την εφαρμογή στην οθόνη.
- • Controller - Παρέχει την αλληλεπίδραση του χρήστη ή τον έλεγχο της επεξεργασίας μοντέλων πληροφοριών (το τμήμα εισόδου).8

Στην περίπτωση της εφαρμογής που μελετάται, το μοντέλο αφορά τα δεδομένα της εφαρμογής και την διεπαφή με την βάση δεδομένων με χρήση ORM. Ο Controller αφορά το κομμάτι την

[8] Hunt, J. (1997). The Model-View-Controller Architecture. In: Smalltalk and Object Orientation. Springer, London

εφαρμογής με την οποία μια εφαρμογή – πελάτης αλληλεπιδρά με την εφαρμογή με χρήση URIs με βάση το πρωτόκολλο HTTP. Ο Controller απαντάει στον πελάτη με δεδομένα JSON, τα οποία στην περίπτωσή μας αποτελούν το View κομμάτι της αρχιτεκτονικής.

Αντικείμενα Μεταφοράς Δεδομένων - DTO

Για τα δεδομένα που ανταλλάσσονται μεταξύ frontend και backend έχει χρησιμοποιηθεί η λογική των αντικειμένων μεταφοράς δεδομένων (Data Transfer Object - DTO). Σύμφωνα με την λογική αυτήν, μεταφέρονται τα απολύτως απαραίτητα δεδομένα. Αυτό έχει τα εξής πλεονεκτήματα. Πρώτον, την ασφάλεια αποφεύγοντας την διαρροή δεδομένων. Για παράδειγμα όταν ένας χρήστης επισκέπτεται την σελίδα προφίλ ενός τρίτου χρήστη, δεν επιστρέφονται όλα τα δεδομένα της εγγραφής του πίνακα user για αυτόν τον χρήστη, που περιλαμβάνει τον κωδικό και το email, αλλά μόνο τα στοιχεία που απαιτούνται για την παρουσίαση. Δεύτερο πλεονέκτημα είναι η μείωση του όγκου πληροφοριών που μεταφέρονται, καθώς δεν μεταφέρεται ολόκληρο το αντικείμενο.

Ιστορίες Χρήστη

Για τις λειτουργίες της εφαρμογής δημιουργήθηκαν ιστορίες χρήστη – user stories.

Οι ιστορίες χρήστη ορίζονται ως ένα κάτι που θέλει ο πελάτης να κάνει το σύστημα. Οι ιστορίες θα πρέπει να υπολογίζονται σε μία έως πέντε ιδανικές εβδομάδες προγραμματισμού. Οι ιστορίες πρέπει να είναι ελεγχόμενες.

Οι ιστορίες μπορούν να διαχωρίζονται σε εργασίες, μετρήσιμες μονάδες προσπάθειας ανάπτυξης. Ο διαχωρισμός γίνεται από τους προγραμματιστές που πρέπει επίσης να υπολογίσουν πόσο χρόνο θα χρειαζόταν 2 συντάκτες για την υλοποίηση κάθε εργασίας9

Ιστορίες χρήστη του ρόλου Χρήστη

Ως ανώνυμος χρήστης θέλω να κάνω εγγραφή. Ο χρήστης πατάει τον σύνδεσμο εγγραφής. Εισάγει το όνομα χρήστη, email, κωδικό και επανάληψη κωδικού.

- Ως ανώνυμος χρήστης θέλω να κάνω σύνδεση. Ο χρήστης εισάγει το όνομα χρήστη και τον κωδικό στην φόρμα σύνδεσης χρήστη.
- Ως χρήστης θέλω να δημιουργώ και να ενημερώνω το προφίλ μου. Το προφίλ πρέπει να περιλαμβάνει πεδία όπως βιογραφικό, εικόνα προφίλ και ρυθμίσεις ασφαλείας.
- Θέλω να συμμετέχω σε ομάδες που επικεντρώνονται σε συγκεκριμένα θέματα ψυχικής υγείας, ώστε να μπορώ να βρω υποστήριξη και πληροφορίες. Οι χρήστες θα πρέπει να μπορούν να αναζητούν, να συμμετέχουν και να δημιουργούν ομάδες, με ομαδικές αναρτήσεις ορατές στα μέλη
- Ως χρήστης μπορώ να καταργώ την εγγραφή μου από ομάδες. Οι χρήστες θα πρέπει να έχουν μια λίστα από τις ομάδες στις οποίες είναι εγγεγραμμένοι και να μπορούν να αποχωρούν από τις ομάδες.
- Ως χρήστης θέλω να δημιουργώ δημοσιεύσεις για να μπορώ να μοιραστώ τις σκέψεις και τις εμπειρίες μου. Η ανάρτηση θα πρέπει να επιτρέπει περιεχόμενο κειμένου, προαιρετική επισύναψη εικόνας και να έχει χρονική σήμανση

[9] Breitman, K., & Leite, J. C. S. P. (2002). Managing user stories. Στο *International Workshop on Time-Constrained Requirements Engineering* σ. 168.

- Ως χρήστης θέλω να σχολιάζω αναρτήσεις για να μπορώ να συμμετέχω σε συζητήσεις. Τα σχόλια πρέπει να επιτρέπουν περιεχόμενο κειμένου, να συνδέονται με μια ανάρτηση και να φέρουν χρονική σήμανση.
- Ως χρήστης, θέλω να μου αρέσουν οι αναρτήσεις και τα σχόλια για να μπορώ να δείξω την υποστήριξή μου. Οι χρήστες θα πρέπει να μπορούν να κάνουν like σε αναρτήσεις και σχόλια, με τον αριθμό των likes να εμφανίζεται.
- Ως χρήστης, θέλω να ορίσω τις προτιμήσεις απορρήτου μου, ώστε να μπορώ να ελέγχω ποιος βλέπει το προφίλ και τις αναρτήσεις μου. Οι ρυθμίσεις απορρήτου θα πρέπει να επιτρέπουν στους χρήστες να επιλέγουν μεταξύ δημόσιου, μόνο για φίλους ή ιδιωτικού για το προφίλ και τις αναρτήσεις τους.
- Ως χρήστης , θέλω να στείλω αιτήματα φιλίας σε άλλους χρήστες, ώστε να μπορώ να συνδεθώ μαζί τους. Οι χρήστες θα πρέπει να μπορούν να στέλνουν και να αποδέχονται/απορρίπτουν αιτήματα φιλίας, με ένα σύστημα ειδοποιήσεων.
- Ως χρήστης, θέλω να δω μια λίστα με τους φίλους μου, ώστε να έχω εύκολη πρόσβαση στα προφίλ και τις αναρτήσεις τους.
- Ως χρήστης θέλω να αναφέρω αναρτήσεις ή σχόλια που είναι ακατάλληλα ή επιβλαβή, ώστε η πλατφόρμα να παραμείνει ένας ασφαλής χώρος. Κάθε ανάρτηση και σχόλιο θα πρέπει να έχει ένα κουμπί αναφοράς που ειδοποιεί τους διαχειριστές

Ιστορίες χρήστη ρόλου Ειδικού

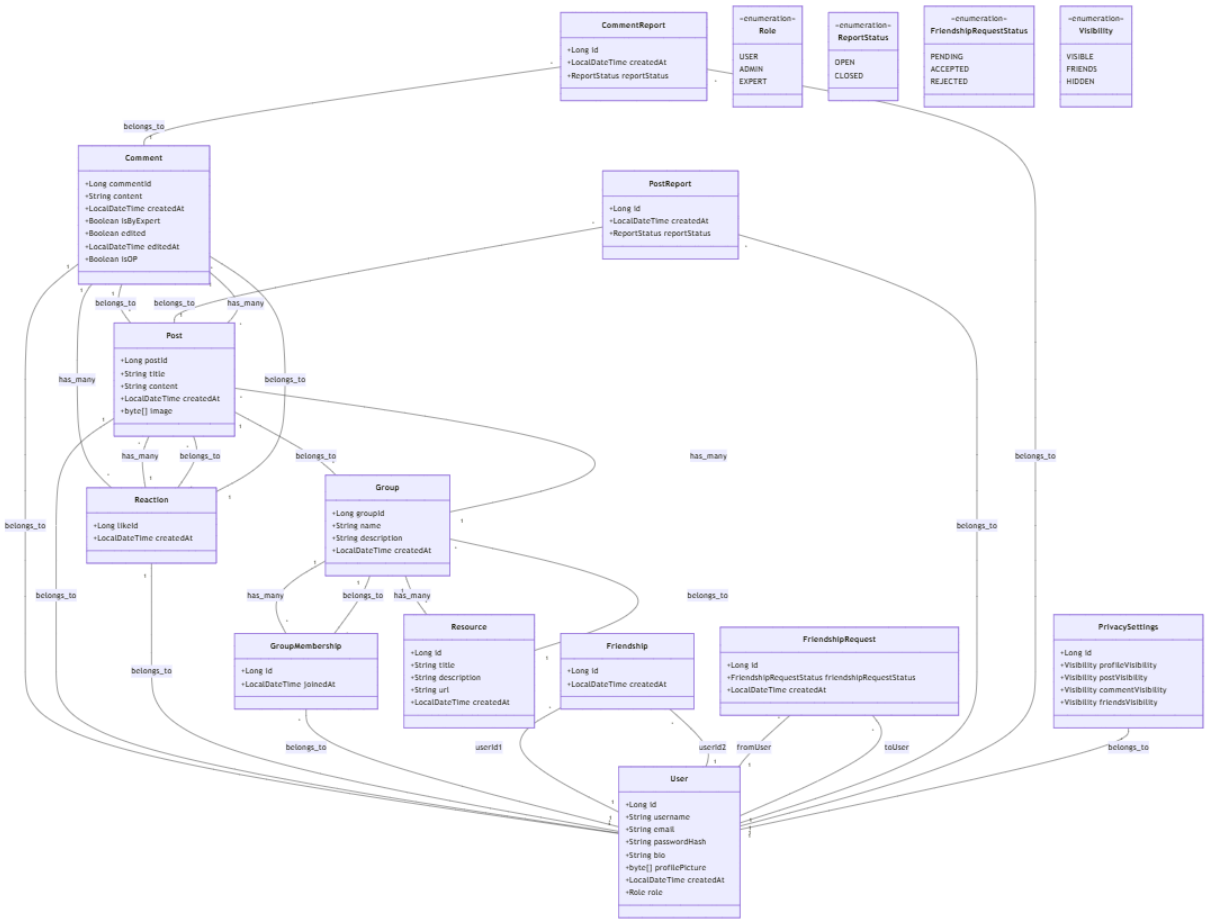
- Ως ειδικός, θέλω να σχολιάσω τις δημοσιεύσεις για να παρέχω πληροφορίες/καθοδήγηση. ένα σχόλιο ειδικού επισημαίνεται συγκεκριμένα ως τέτοιο.
- Ως ειδικός ψυχικής υγείας, θέλω να έχω ένα επαληθευμένο σήμα στο προφίλ μου, ώστε οι χρήστες να μπορούν να εμπιστεύονται το περιεχόμενο και τις συμβουλές μου.
- Ως ειδικός ψυχικής υγείας, θέλω να ανεβάζω και να μοιράζομαι πόρους όπως άρθρα, βίντεο, ώστε οι χρήστες να έχουν πρόσβαση σε χρήσιμο υλικό. Οι ειδικοί θα πρέπει να διαθέτουν μια ειδική ενότητα για πόρους, στους οποίους οι χρήστες μπορούν να έχουν εύκολη πρόσβαση.
- Ως ειδικός ψυχικής υγείας, θέλω να παρέχω συνδέσμους σε ανοιχτές γραμμές επικοινωνίας, ώστε οι χρήστες να μπορούν να λαμβάνουν επαγγελματική βοήθεια όταν χρειάζεται. Οι ειδικοί θα πρέπει να είναι σε θέση να δημιουργούν σημαντικές επαφές μια ανοιχτή γραμμή επικοινωνίας σε μια συγκεκριμένη ενότητα.

Ιστορίες χρήστη ρόλου Διαχειριστή

- Ως διαχειριστής θέλω να αλλάξω τον ρόλο ενός χρήστη. Ο διαχειριστής βρίσκει τον χρήστη και επιλέγει έναν ρόλο εκ των: Χρήστης, Ειδικός, Διαχειριστής.
- Ως διαχειριστής, θέλω να αναλάβω δράση σε αναφορές σχολίων και αναρτήσεων. Ένας διαχειριστής θα δει τις αναφορές και μπορεί να διαγράψει τα περιεχόμενα των αναρτήσεων ή των σχολίων ή να αγνοήσει την αναφορά. Στη συνέχεια θα ολοκληρωθεί η αναφορά.
- Ως διαχειριστής θέλω να διαγράψω έναν χρήστη. Ο διαχειριστής θα πρέπει να μπορεί να βλέπει όλους τους χρήστες και να έχει την δυνατότητα διαγραφής τους από την εφαρμογή.
- Ως διαχειριστής θέλω να βλέπω τα στατιστικά της εφαρμογής. Τα στατιστικά περιλαμβάνουν: αριθμό χρηστών, αριθμό ομάδων, αριθμό αναρτήσεων και ανά ομάδα, αριθμό σχολίων ανά ανάρτηση.

Διάγραμμα κλάσεων

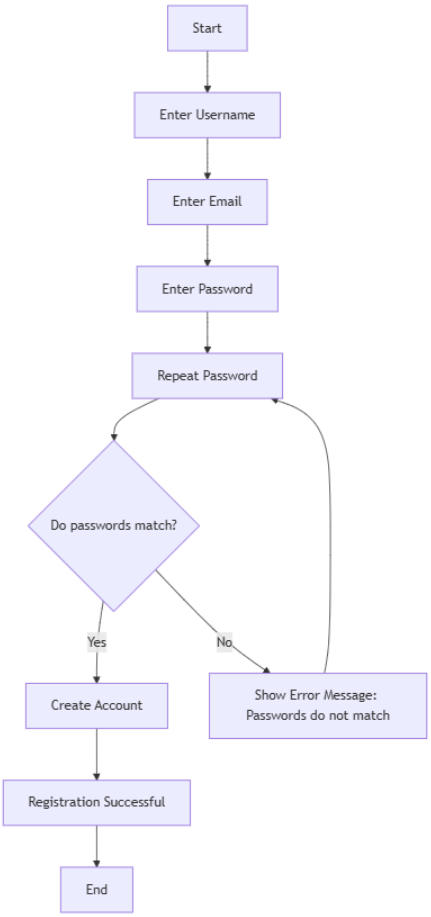
Με βάση τις παραπάνω απαιτήσεις δημιουργήθηκε το εξής διάγραμμα κλάσεων:



Σχήμα 2 Διάγραμμα κλάσεων

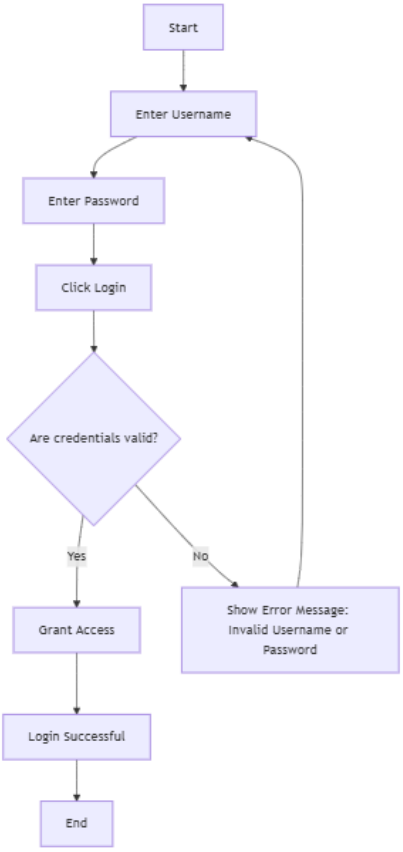
Διαγράμματα ροής

Εγγραφή νέου χρήστη



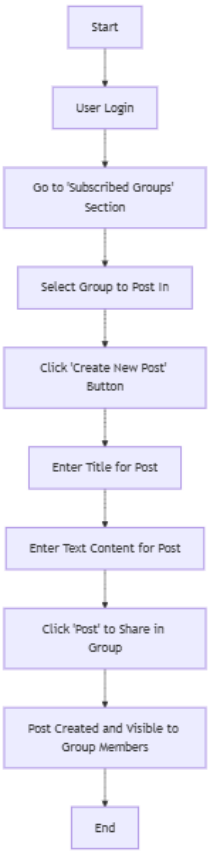
Σχήμα 3 Διάγραμμα ροής εγγραφής νέου χρήστη

Σύνδεση χρήστη



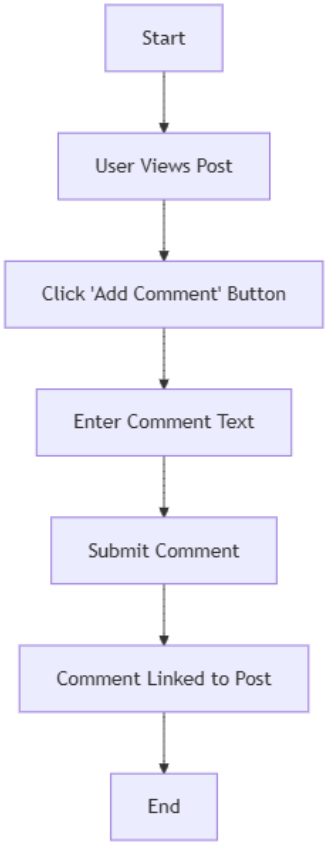
Σχήμα 4 Διάγραμμα ροής σύνδεσης χρήστη

Δημιουργία νέας ανάρτησης



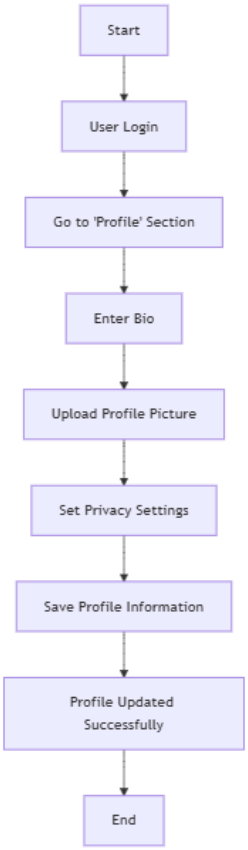
Σχήμα 5 Διάγραμμα ροής προσθήκης νέας ανάρτησης

Δημιουργία σχολίου



Σχήμα 6 Διάγραμμα ροής Δημιουργία σχολίου

Ενημέρωση προφίλ χρήστη

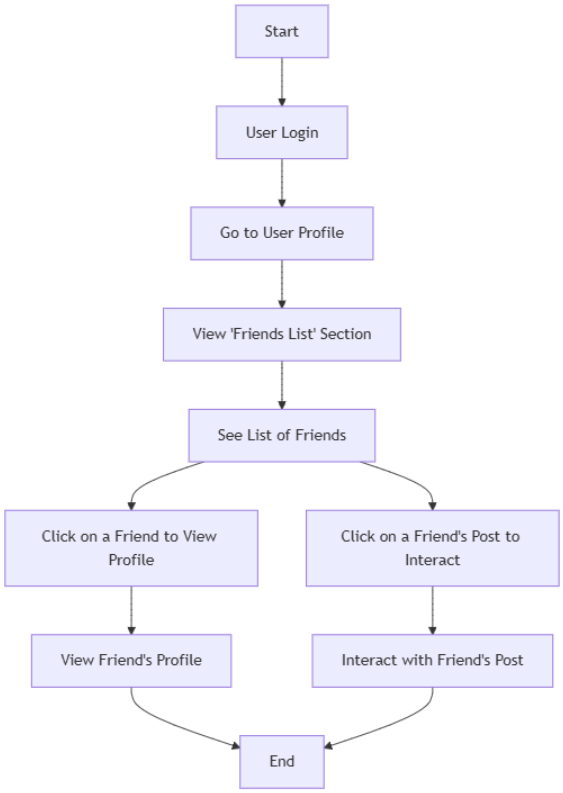


Σχήμα 7 Διάγραμμα ροής ενημέρωσης προφίλ χρήστη

Επεξεργασία ρυθμίσεων ασφαλείας

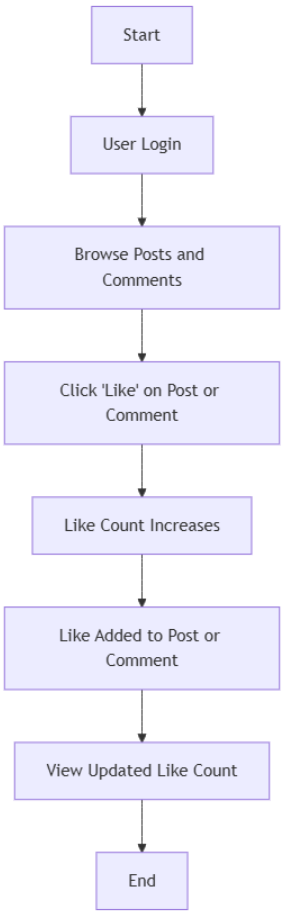
Σχήμα 8 Διάγραμμα ροής επεξεργασίας ρυθμίσεων ασφαλείας

Προβολή λίστας φίλων



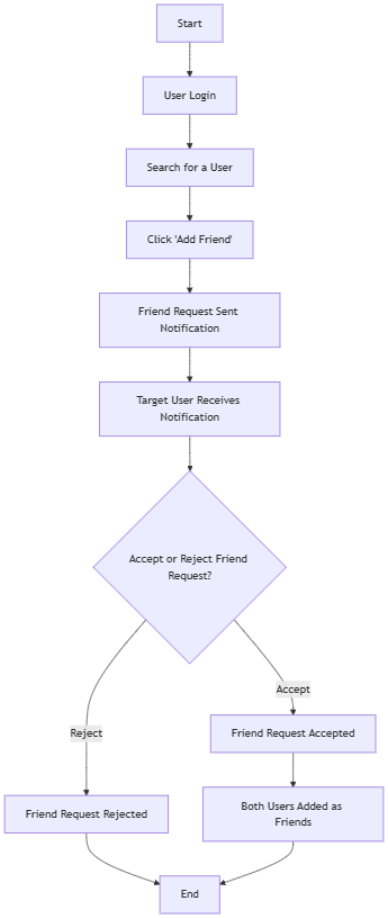
Σχήμα 9 Διάγραμμα ροής προβολής λίστας φίλων

Προσθήκη like σε ανάρτηση ή σχόλιο



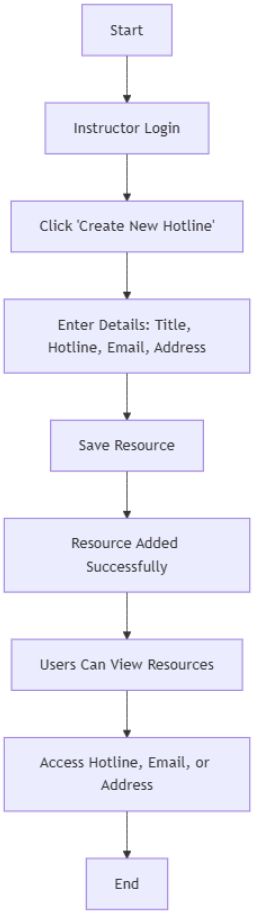
Σχήμα 10 Διάγραμμα ροής προσθήκης like σε ανάρτηση ή σχόλιο

Προσθήκη φίλου



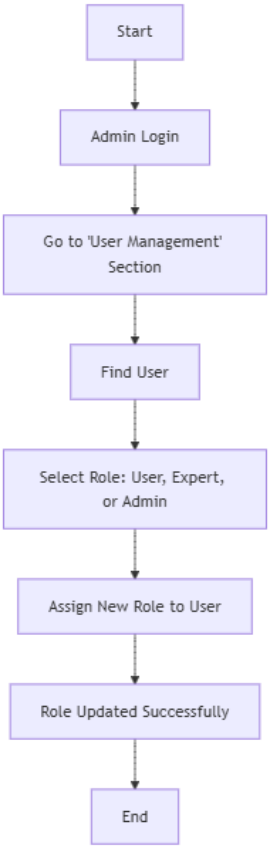
Σχήμα 11 Διάγραμμα ροής προσθήκη φίλου

Διάγραμμα ροής προσθήκη νέας τηλεφωνικής γραμμής



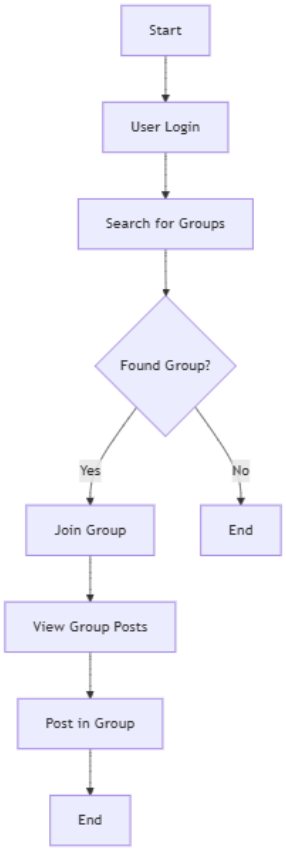
Σχήμα 12 Διάγραμμα ροής προσθήκη νέας γραμμής

Αλλαγή ρόλου ενός χρήστη



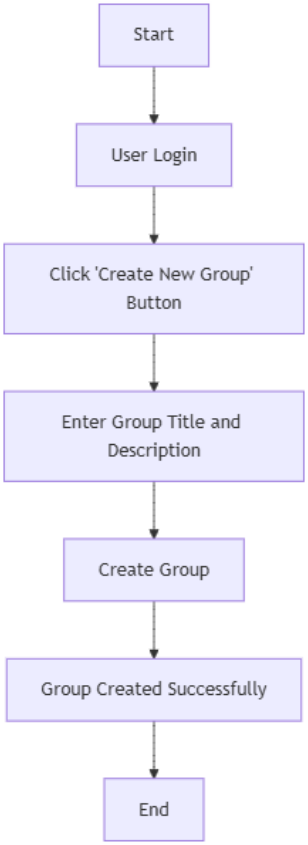
Σχήμα 13 Διάγραμμα ροής αλλαγή ρόλου ενός χρήστη

Συμμετοχή σε ομάδα



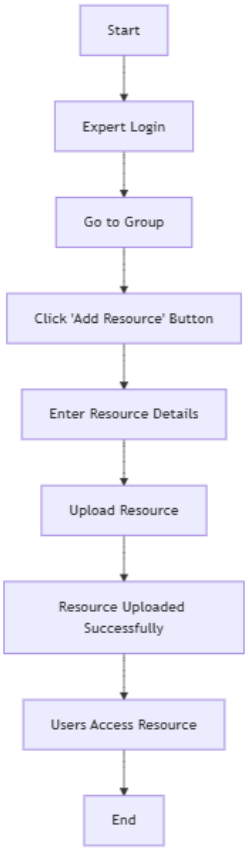
Σχήμα 14 Διάγραμμα ροής συμμετοχή σε ομάδα

Δημιουργία ομάδας



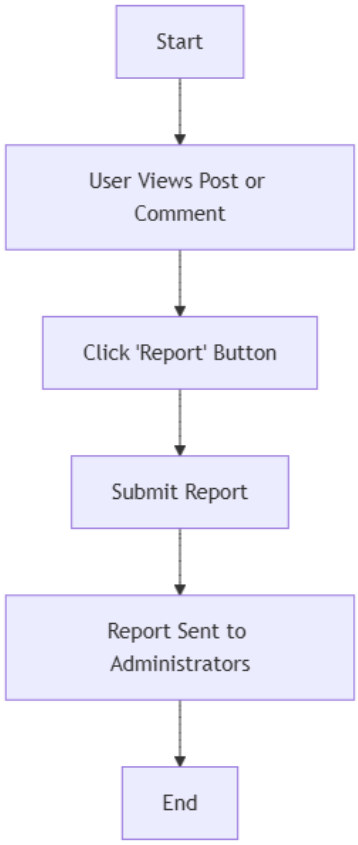
Σχήμα 15 Διάγραμμα ροής δημιουργία ομάδας

Προσθήκη πόρων



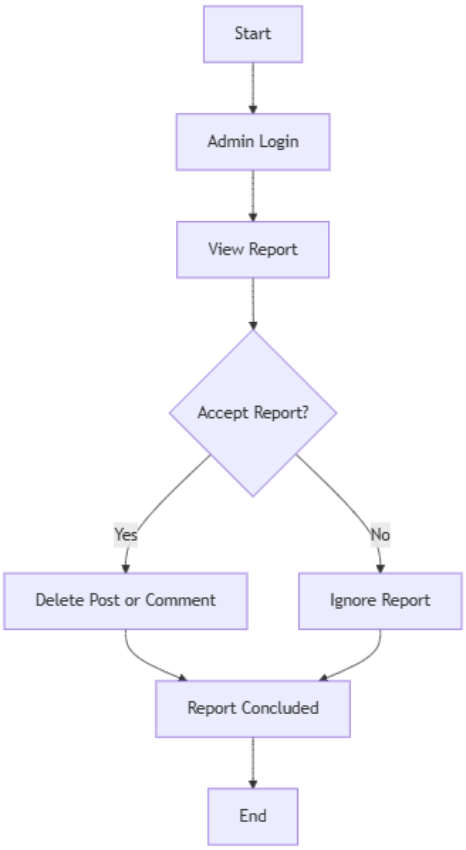
Σχήμα 16 Διάγραμμα ροής προσθήκη πόρων

Δημιουργία αναφοράς



Σχήμα 17 Διάγραμμα ροής Δημιουργία αναφοράς

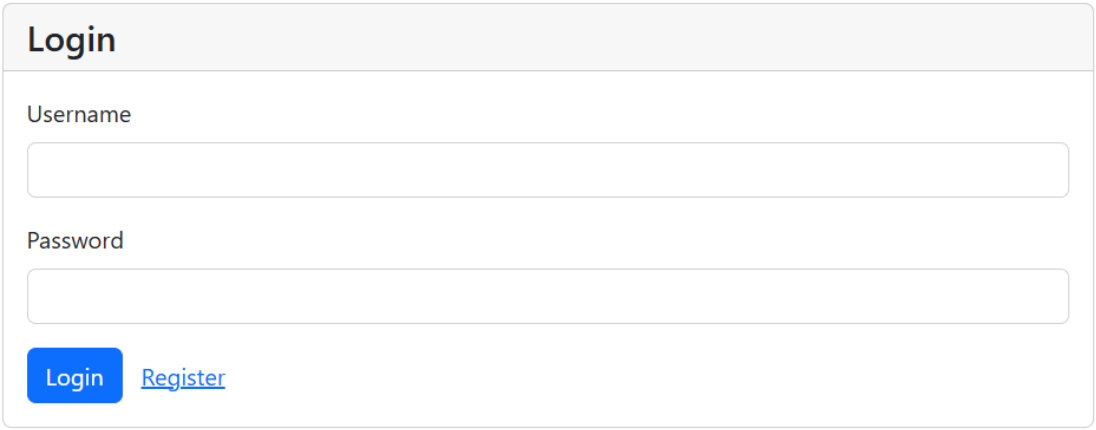
Διαχείριση Αναφορών



Σχήμα 18 Διάγραμμα ροής αναφορών

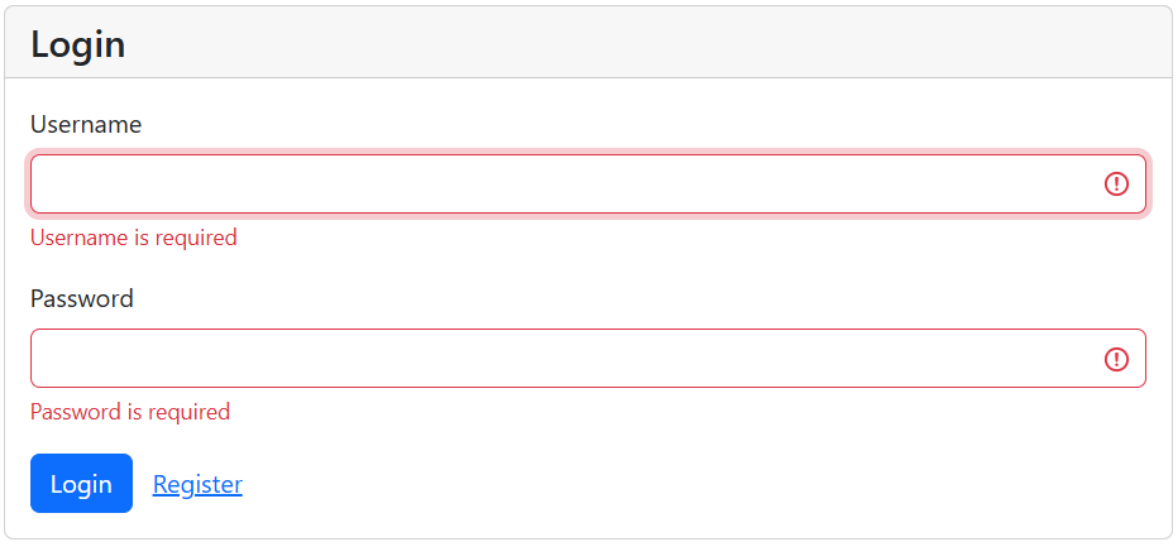
Παρουσίαση της εφαρμογής

Σύνδεση χρήστη



Σχήμα 19 Οθόνη σύνδεσης χρήστη

Ανοίγοντας την εφαρμογή, παρουσιάζεται στον χρήστη η οθόνη σύνδεσης. Τα πεδία username και password είναι υποχρεωτικά να τα συμπληρώσει ο χρήστης και παρουσιάζεται ένα κατάλληλο μήνυμα στην περίπτωση που είναι κενά.



Σχήμα 20 Οθόνη σύνδεσης χρήστη με κενά πεδία

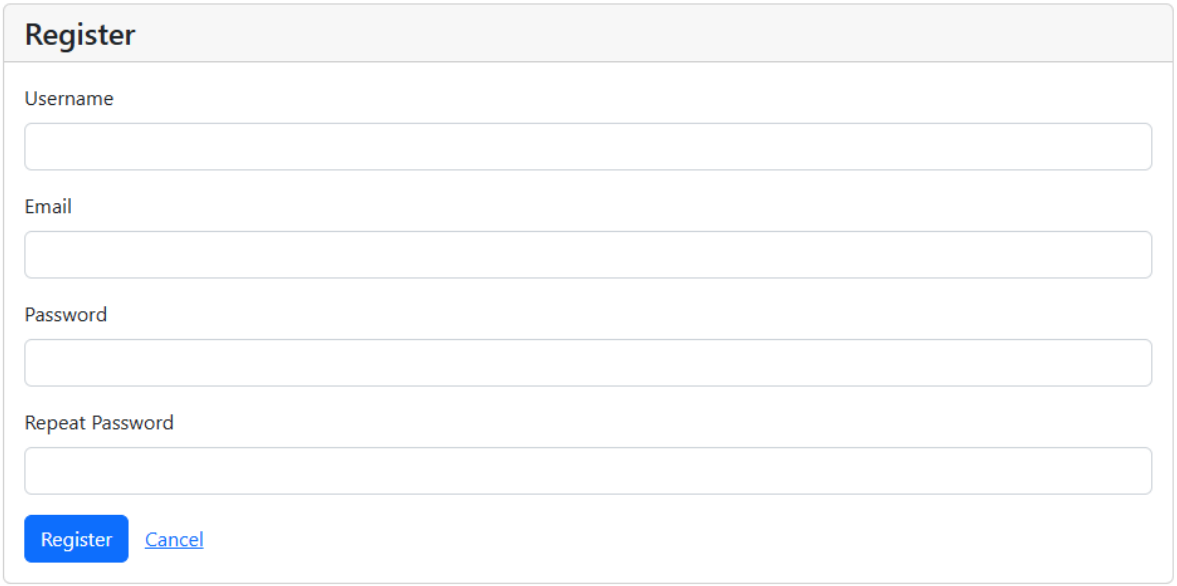
Συμπληρώνοντας ο χρήστης τα στοιχεία του μπορεί να εισέλθει στην εφαρμογή.

Η υλοποίηση της λειτουργικότητας πιστοποίησης χρήστη έγινε με την βιβλιοθήκη spring-boot-starter-security του SpringBoot.

Τα διαπιστευτήρια του χρήστη, δηλαδή το όνομα χρήστη και ο κωδικός πρόσβασης είναι αποθηκευμένα στην βάση δεδομένων, στον πίνακα User και στα πεδία username και passwordHash αντίστοιχα. Η εφαρμογή έχει ρυθμιστεί ώστε ο κωδικός του χρήστη να αποθηκεύεται κατακερματισμένος (hashed) με την κρυπτογραφική συνάρτηση κατακερματισμού BCrypt.

Με την επιτυχή σύνδεση η backend εφαρμογή παράγει ένα JWT Token, το οποίο λαμβάνει η frontend εφαρμογή, η οποία σε κάθε αίτημα που κάνει από εκείνη την στιγμή μέχρι την αποσύνδεση του χρήστη, συμπεριλαμβάνει και αυτό το JWT Token για την διαπίστευση του χρήστη.

Εγγραφή χρήστη



Σχήμα 21 Οθόνη εγγραφής χρήστη

Στην εγγραφή χρήστη ο χρήστης, πρέπει να εισάγει το όνομα επιθυμητό όνομα χρήστη, το email του, κωδικό και επανάληψη κωδικού. Αντίστοιχα, όπως και οθόνη σύνδεσης, κι εδώ όλα τα πεδία είναι υποχρεωτικά. Αφού η εγγραφή του χρήστη είναι επιτυχημένη, του ανατίθεται ο ρόλος User, δημιουργούνται οι προεπιλεγμένες ρυθμίσεις ασφαλείας του χρήστη, δηλαδή αναρτήσεις, σχόλια, και προφίλ κρυμμένα σε όλους.

Λόγω χρήσης ORM οι ενέργειες αυτές εκτελούνται ως πρόσβαση σε αντικείμενα Java. Για παράδειγμα στην SpringBoot πραγματοποιείται ως εξής:

```
public User signup(RegisterRequest input) {
    User user = User.builder()
            .username(input.getUsername())
            .email(input.getEmail())
            .passwordHash(passwordEncoder.encode(input.getPassword()))
            .role(Role.USER)
            .build();
    userRepository.save(user);
```

```
    PrivacySettings privacySettings = new PrivacySettings();
    privacySettings.setUser(user);
    privacySettingsRepository.save(privacySettings);

    userRepository.save(user);
    return user;
}
```

και στην frontend εφαρμογή με Angular πραγματοποιείται ένα αίτημα HTTP POST:

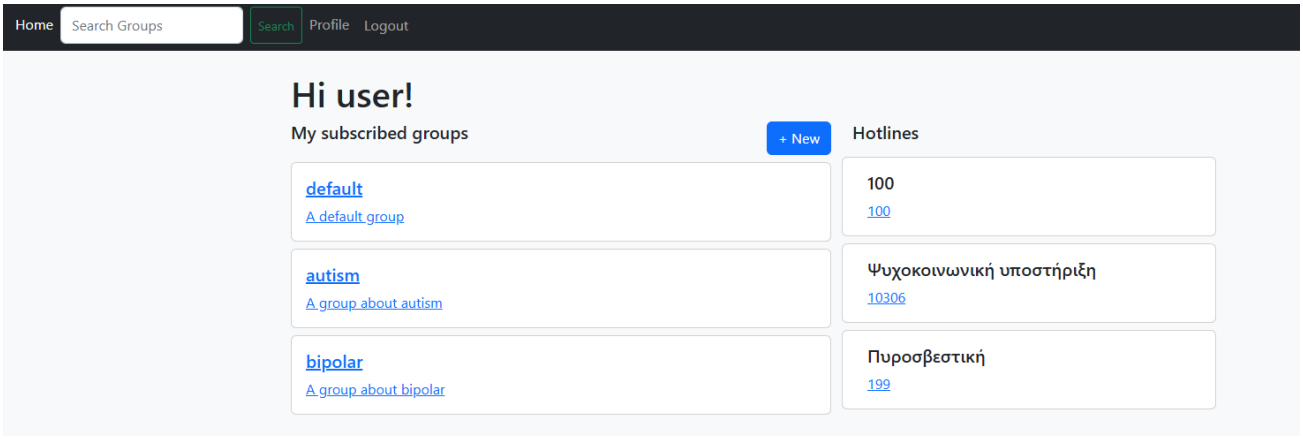
```
this.http.post(`${environment.apiUrl}/auth/signup`, user);
```

και στην συνέχεια ενημερώνεται ο χρήστης για το αποτέλεσμα και μεταβαίνει στην οθόνη σύνδεσης:

```
this.accountService.register(this.form.value)
            .pipe(first())
            .subscribe({
                next: () => {
                    console.info("Successful register");
                    this.alertService.success('Registration successful',
{ keepAfterRouteChange: true });
                    this.router.navigate(['../login'], { relativeTo:
this.route });
                },
                error: (error: any) => {
                    console.error("Register not successful: " + error);
                    this.alertService.error(error);
                    this.loading = false;
                }
            });
```

Κάνοντας σύνδεση ο χρήστης με τα στοιχεία στην οθόνη σύνδεσης βλέπει την αρχική οθόνη της εφαρμογής.

Αρχική οθόνη συνδεδεμένου χρήστη



Σχήμα 22 Αρχική οθόνη συνδεδεμένου χρήστη

Στην αρχική οθόνη, αφού συνδεθεί ο χρήστης, υπάρχει η γραμμή εργαλείων, που περιέχει σύνδεσμο για την αρχική οθόνη, ένα πεδίο αναζήτησης για ομάδες, σύνδεσμο στο προφίλ και ρυθμίσεις του χρήστη και τέλος κουμπί για αποσύνδεση από την εφαρμογή.

Το σώμα της οθόνης χωρίζεται σε 2 στήλες, μία για τις εγγεγραμμένες ομάδες και άλλη μια για τηλέφωνα και επαφές έκτακτης ανάγκης.

Για τις εγγεγραμμένες ομάδες η εφαρμογή φορτώνει αρχικά από την βάση δεδομένων όλες τις εγγραφές του πίνακα GROUP_MEMBERSHIP με user_id αυτό του συνδεδεμένου χρήστη. Στην συνέχεια φορτώνονται όλες οι ομάδες με id το group_id από τα προηγούμενα αποτελέσματα. Έπειτα μετασχηματίζονται με το κατάλληλο DTO και παρουσιάζονται στον χρήστη.

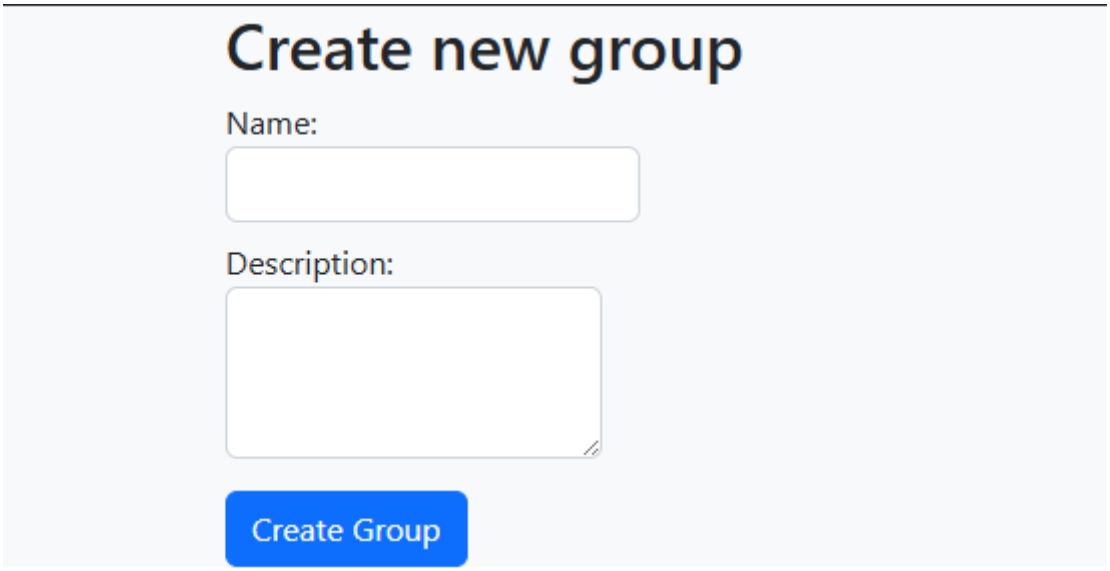
Παράδειγμα κώδικα από την backend εφαρμογή:

```
User user = userService.getLoggedInUser();
List<GroupMembership> groupMemberships =
groupMembershipRepository.findByUser(user);
List<Group> groups =
groupMemberships.stream().map(GroupMembership::getGroup).toList();
```

Η εφαρμογή frontend αρχικά κάνει ένα αίτημα HTTP GET για να λάβει τα αποτελέσματα:

```
this.http.get<Group[]>(`${environment.apiUrl}/group/my`);
```
και στην συνέχεια τα κρατάει σε μεταβλητή:

```
 this.groupsService.getSubscribedGroups().subscribe(response => {
      this.groups = response;
    });
```
Πάνω από τις εγγεγραμμένες ομάδες υπάρχει κουμπί δημιουργίας νέας ομάδας.



Σχήμα 23 Δημιουργία νέας ομάδας

Κατά την δημιουργία νέας ομάδας ο χρήστης εισάγει το όνομα και μια περιγραφή.

Πραγματοποιείται ένα αίτημα HTTP POST με το σώμα του αιτήματος να περιέχει το όνομα και την περιγραφή. Αφού δημιουργηθεί, ο χρήστης εγγράφεται αυτόματα στην ομάδα αυτή.

Παράδειγμα κώδικα για την δημιουργία νέας ομάδας στην εφαρμογή backend:

```
Group group = Group.builder()
        .name(request.getName())
        .description(request.getDescription())
        .build();
groupRepository.save(group);
```

Και στην συνέχεια για την εγγραφή στην ομάδα:

```
Group group = groupRepository.findById(id).orElseThrow();
User user = userService.getLoggedInUser();
GroupMembership groupMembership = GroupMembership.builder()
        .user(user)
        .group(group)
        .build();
groupMembershipRepository.save(groupMembership);
```

Στην εφαρμογή backend πραγματοποιείται ένα αίτημα HTTP POST:

```
this.http.post<Group>(`${environment.apiUrl}/group/`, group);
```

και στην συνέχεια ο χρήστης εγγράφεται στην ομάδα και μεταφέρεται στην σελίδα της ομάδας:

```
this.groupService.createGroup(this.form.value).subscribe({
     next: data => {
       this.loading = false;
```

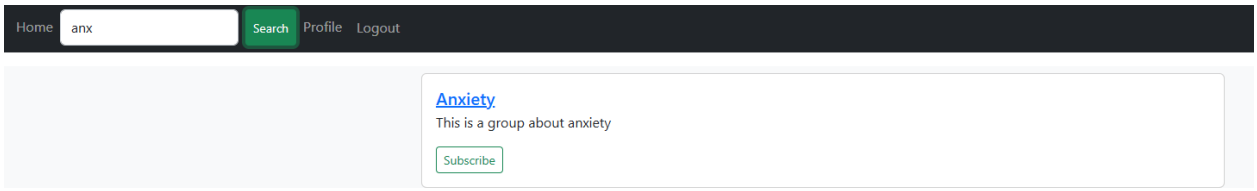
```
        if(data) {
          this.alertService.info("Group Created");
          const id = data.id;
          if(id) {
            this.groupService.subscribe(id).subscribe(() => {});
            this.router.navigateByUrl(`/groups/${id}`);
          }
        }
        else {
          this.alertService.error("Couldn't create group");
        }
      },
      error: error => {
        this.loading = false;
        this.alertService.error("Couldn't create group: " + error);

      }
    });
```

Για την εγγραφή σε ομάδα ο κώδικας της frontend εφαρμογής πραγματοποιεί ένα HTTP GET αίτημα:

```
this.http.get<GroupMembership>(`${environment.apiUrl}/group/subscribe/
${groupid}`);
```

Αναζήτηση ομάδας



Σχήμα 24 Αναζήτηση ομάδας

Ο χρήστης μπορεί να αναζητήσει μια ομάδα με το όνομά της ή κομμάτι αυτής πληκτρολογώντας στο πεδίο αναζήτησης και πατώντας το πλήκτρο Search στην εργαλειοθήκη. Στην συνέχεια ο χρήστης μεταφέρεται στην σελίδα αναζήτησης όπου παρουσιάζεται μια λίστα με τα αποτελέσματα. Το κάθε αποτέλεσμα περιέχει τον τίτλο, την περιγραφή της ομάδας και ένα κουμπί για εγγραφή. Με την εγγραφή του χρήστη, δημιουργείται μια νέα εγγραφή στον πίνακα GROUP_MEMBERSHIP, με στοιχεία το id του χρήστη και το id της ομάδας.

Με ORM η λήψη αυτών των αποτελεσμάτων γίνεται με χρήση αντικειμένων Java.

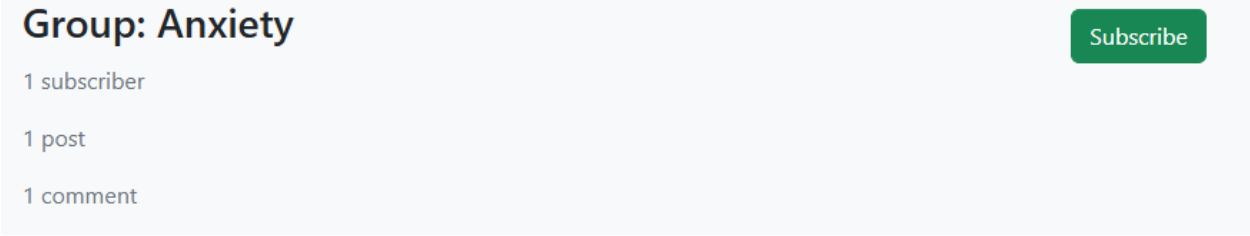
Παράδειγμα κώδικα αναζήτησης στο όνομα της ομάδας:

```
List<Group> groups = groupRepository.findByNameContains(term);
```

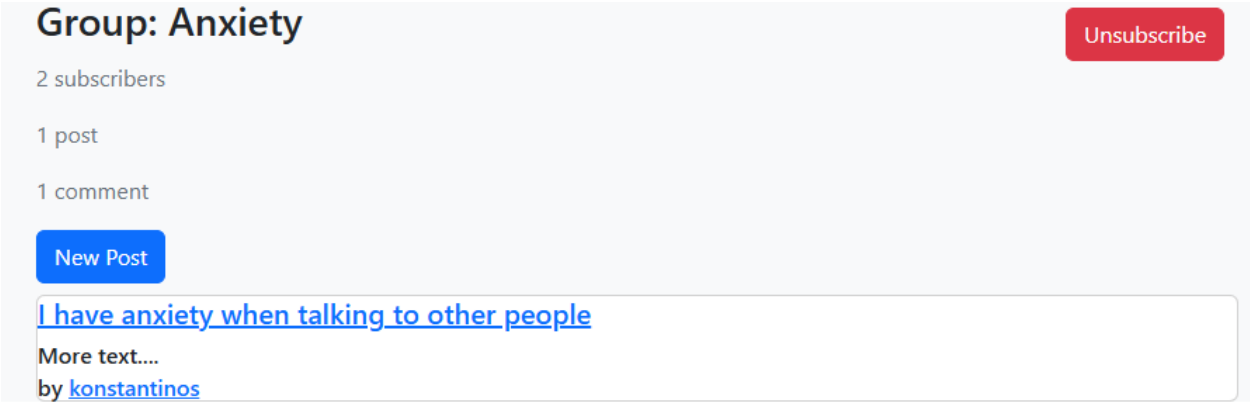
Η εφαρμογή frontend πραγματοποιεί ένα αίτημα HTTP GET με παράμετρο το πεδίο προς αναζήτηση:

```
this.http.get<Group[]>(`${environment.apiUrl}/group/search/${searchTerm}`);
```

Σελίδα ομάδας



Σχήμα 25 Σελίδα ομάδας χωρίς εγγγραφή



Σχήμα 26 Σελίδα ομάδας με εγγραφή

Η σελίδα ομάδας παρουσιάζει τον τίτλο της ομάδας, στοιχεία όπως το πλήθος των συνδρομητών, των αναρτήσεων και των σχολίων. Ένα κουμπί για απεγγραφή ή εγγραφή στην την ομάδα ανάλογα εάν ο τρέχων χρήστης είναι μέλος ή όχι αντίστοιχα. Επίσης εάν είναι μέλος της ομάδας εμφανίζεται ένα κουμπί δημιουργίας νέας ανάρτησης καθώς και μια λίστα με τις αναρτήσεις. Η λίστα των αναρτήσεων, για την κάθε μία περιέχει τον τίτλο της, ένα απόσπασμα από το κείμενο και το όνομα χρήστη του δημιουργού της ανάρτησης με σύνδεσμο προς την σελίδα προφίλ του.

Με την χρήση ORM αυτές οι ενέργειες γίνονται με χρήση αντικειμένων Java. Παρατίθεται παράδειγμα από την εγγραφή σε ομάδα:

```
GroupMembership groupMembership = GroupMembership.builder()
        .user(user)
        .group(group)
        .build();
groupMembershipRepository.save(groupMembership);
```

Η frontend εφαρμογή πραγματοποιεί ένα HTTP GET αίτημα με παράμετρο το ID της ομάδας για την εγγραφή:

```
this.http.get<GroupMembership>(`${environment.apiUrl}/group/subscribe/
${groupid}`);
```

Και στην συνέχεια ενημερώνεται ο χρήστης με ένα μήνυμα και ανανεώνεται η σελίδα αναζήτησης.

```
onSubscribeClick(groupId: number | undefined) {
    this.groupsService.subscribe(groupId).subscribe(x => {
      this.alertService.success("Subscribed");
    });
    this.ngOnInit();
  }
```

Νέα ανάρτηση

Σχήμα 27 Σελίδα δημιουργίας νέας ανάρτησης

Στην σελίδα νέας ανάρτησης, ο χρήστης εισάγει τον τίτλο και το σώμα του κειμένου. Με το κουμπί Cancel επιστρέφει στην σελίδα της ομάδας, και με το κουμπί Submit δημιουργείται νέα ανάρτηση με χρήση HTTP POST με το σώμα του αιτήματος να περιέχει τον τίτλο, το περιεχόμενο και το ID της ομάδας. Στην βάση δεδομένων δημιουργείται μια νέα εγγραφή του πίνακα Post, όπου πέραν των παραπάνω στοιχείων εισάγεται αυτόματα η ημερομηνία δημιουργίας και συνδέεται με τον χρήστη που την δημιούργησε. Υπάρχει επίσης έλεγχος για το εάν ο χρήστης είναι μέλος της ομάδας.

Ο κώδικας εφαρμογής backend για την δημιουργία ανάρτησης:

```
@PostMapping("/")
public ResponseEntity<PostResponse> createPost(@RequestBody
CreatePostRequest request) throws Exception {
    Group group =
groupRepository.findById(request.getGroup()).orElseThrow();
    User user = userService.getLoggedInUser();

    if (!groupMembershipRepository.existsByGroupAndUser(group, user))
{
        throw new Exception("User: " + user.getUsername() + " not a
member of group: " + group.getName());
    } else {
        Post post = Post.builder()
                .title(request.getTitle())
                .content(request.getContent())
```

```
                .user(user)
                .group(group)
                .image(request.getImage())
                .build();
        postRepository.save(post);
        PostResponse response = PostResponse.from(post);
        return ResponseEntity.ok(response);
    }
}
```

Ο κώδικας εφαρμογής frontend για την δημιουργία ανάρτησης:

Αρχικά δημιουργείται ένα αίτημα HTTP POST με τα δεδομένα που εισήγαγε ο χρήστης:

```
const data = {'title': title, 'content': content, 'group':
Number(group)}
this.http.post<Post>(`${environment.apiUrl}/post/`, data);
```

Στην συνέχεια λαμβάνεται το αποτέλεσμα και ο χρήστης μεταφέρεται στην σελίδα της ανάρτησης:

```
this.postService.createPost(this.f['title'].value,
this.f['content'].value, this.groupId)
      .subscribe({
        next: (data: Post) => {
          console.log(data);
          const id = data.id;
          this.router.navigateByUrl(`/posts/${id}`);
        },
        error: error => {
          this.alertService.error(error);
          this.loading = false;
        }
      })
```

Σελίδα ανάρτησης

Σχήμα 28 Σελίδα ανάρτησης

Η σελίδα της ανάρτησης περιέχει τα στοιχεία της ανάρτησης, όπως ο τίτλος, ο δημιουργός και το κείμενο. Περιέχει επίσης πλήκτρο για Like με το πλήθος τους να αναγράφεται δίπλα του. Κάτω από την ανάρτηση συμπεριλαμβάνονται όλα τα σχόλια αυτής της ανάρτησης, με στοιχεία, τον δημιουργό, την ημερομηνία και το κείμενο του σχολίου. Συμπεριλαμβάνεται επίσης κουμπί για Like με το πλήθος του να αναγράφεται δίπλα.

Στην κορυφή των σχολίων υπάρχει πεδίο για δημιουργία νέου σχολίου.

Παρακάτω είναι ο κώδικας προσθήκης σχολίου σε ανάρτηση στην backend εφαρμογή:

```
Post post =
postRepository.findById(request.getPostId()).orElseThrow();
Group group =
groupRepository.findById(post.getGroup().getGroupId()).orElseThrow();
User loggedInUser = userService.getLoggedInUser();
if(!groupMembershipRepository.existsByGroupAndUser(group,
loggedInUser)) {
    throw new Exception("User: "+ loggedInUser.getUsername() +" not a
member of group: " + group.getName());
}
Boolean isByExpert = userService.isExpert(loggedInUser);
Boolean isOP = post.getUser() == loggedInUser;
Comment comment = Comment.builder()
        .content(request.getContent())
        .post(post)
        .user(loggedInUser)
        .isByExpert(isByExpert)
        .edited(false)
        .isOP(isOP)
        .build();
commentRepository.save(comment);
post.getComments().add(comment);
postRepository.save(post);
```

Στην εφαρμογή frontend πραγματοποιείται ένα HTTP POST αίτημα,

```
const data = {'content': comment, 'postId': Number(postId)};
this.http.post<Comment>(`${environment.apiUrl}/comment/`, data);
```

Στην συνέχεια ανανεώνονται τα δεδομένα της σελίδας της ανάρτησης:

```
this.commentService.addComment(this.f['comment'].value, this.post?.id)
    .subscribe({
      next: (data: Comment) => {
        console.log(data);
        this.ngOnInit();
      },
      error: (error) => {
        this.alertService.error(error);
        this.loading = false;
      }
    });
```

Για κάθε ανάρτηση και σχόλιο υπάρχει κουμπί αναφοράς, όπου δημιουργείται μία νέα αναφορά με κατάσταση Ανοιχτό.

Κατά την αναφορά δημιουργείται νέα εγγραφή στον πίνακα Post_Report για αναρτήσεις και Comment_Report για σχόλια αντίστοιχα με στοιχεία τον χρήστη που έκανε την αναφορά, την ανάρτηση ή το σχόλιο, την κατάσταση, που όταν δημιουργείται έχει την κατάσταση Open και όταν κάνει κάποια ενέργεια ο διαχειριστής έχει την κατάσταση Κλειστό.

Τα κουμπιά Like έχουν υλοποιηθεί ως κουμπιά εναλλαγής, όπου το πρώτο πάτημα προσθέτει το like και το δεύτερο το αφαιρεί.

Παράδειγμα κώδικα προσθήκης Like:

```
Reaction reaction = Reaction.builder()
        .post(post)
        .user(loggedInUser)
        .build();
reactionRepository.save(reaction);
```

Παράδειγμα κώδικα αφαίρεσης Like:

```
Reaction reaction = reactionRepository.findByUserAndPost(loggedInUser,
post).orElseThrow();
ReactionResponse response = PostReactionResponse.from(reaction);
reactionRepository.delete(reaction);
```

Στην frontend εφαρμογή για την προσθήκη και αφαίρεση like δημιουργείται αίτημα HTTP GET:

Για την προσθήκη like:

```
this.http.get<Like>(`${environment.apiUrl}/post/${id}/like`);
```

Για την αφαίρεση like:

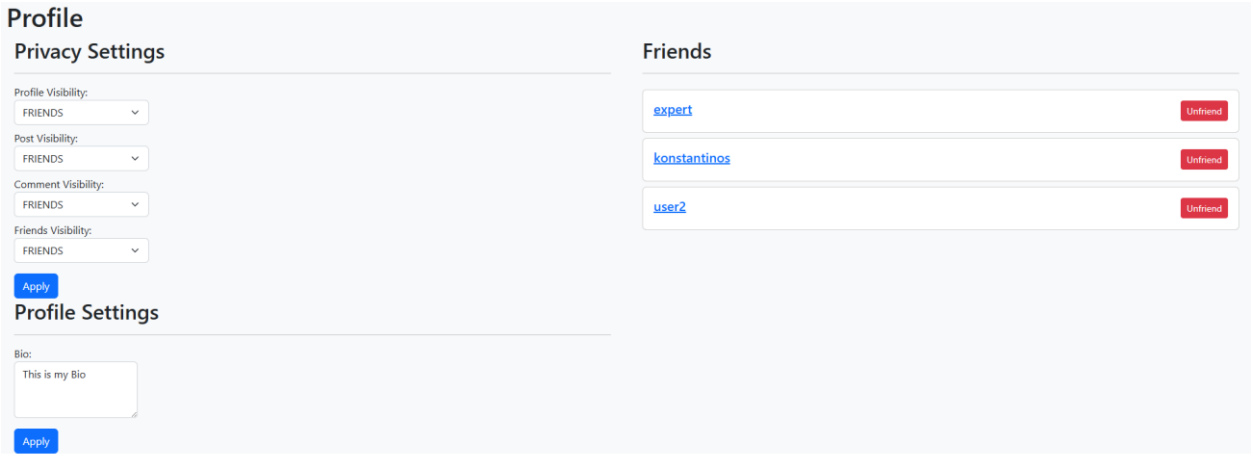
```
this.http.get<Like>(`${environment.apiUrl}/post/${id}/unlike`);
```

Και στην συνέχεια, ανανεώνεται το γραφικό του Like ώστε να δείχνει την τρέχουσα κατάσταση και ανανεώνεται ο αριθμός των Likes.

```
likePost() {
    this.loadingLike = true;
    if(this.hasLike) {
      this.postsService.unlikePost(this.post?.id).subscribe({
        next: () => {
          this.hasLike = false;
          this.loadingLike = false;
          this.countLikes();
        },
        error: error => {
          this.alertService.error(error);
          this.loading = false;
        }
      });
    }
    else {
      this.postsService.likePost(this.post?.id).subscribe({
        next: () => {
          this.hasLike = true;
          this.loadingLike = false;
          this.countLikes();
        },
        error: error => {
          this.alertService.error(error);
          this.loading = false;
        }
      });
    }
  }
```

Έχει υλοποιηθεί μόνο το θετικό like και όχι το αρνητικό για λόγους αποφυγής αρνητικής αλληλεπίδρασης και ατμόσφαιρας στο κοινωνικό δίκτυο.

Ρυθμίσεις προφίλ



Σχήμα 29 Οθόνη ρυθμίσεων προφίλ

Η οθόνη προφίλ χωρίζεται σε τρεις ενότητες.

Η πρώτη ενότητα περιέχει τις ρυθμίσεις ιδιωτικότητας, όπου ο χρήστης μπορεί να ρυθμίσει ποιο από τα προφίλ, αναρτήσεις, σχόλια, και φίλοι θα φαίνονται σε άλλους χρήστες. Οι επιλογές είναι, κρυφό, όπου η επιλογή δεν είναι ορατή σε τρίτους, φίλοι, όπου η επιλογή είναι ορατή σε όσους είναι φίλοι, με τον τρέχων χρήστη και τέλος, δημόσια, όπου η επιλογή είναι ορατή σε όλους τους συνδεδεμένους χρήστες. Η επιλογή αφορά την σελίδα του χρήστη και όχι τις αναρτήσεις ή τα σχόλια όπως φαίνονται στην οθόνη κάποιας ομάδας.

Ο κώδικας frontend εφαρμογής για αλλαγή ρυθμίσεων ιδιωτικότητας:

Πραγματοποιείται ένα HTTP POST αίτημα με τα δεδομένα ρυθμίσεων:

```
this.http.post<Boolean>(`${environment.apiUrl}/user/privacy`,
privacySettings);
```

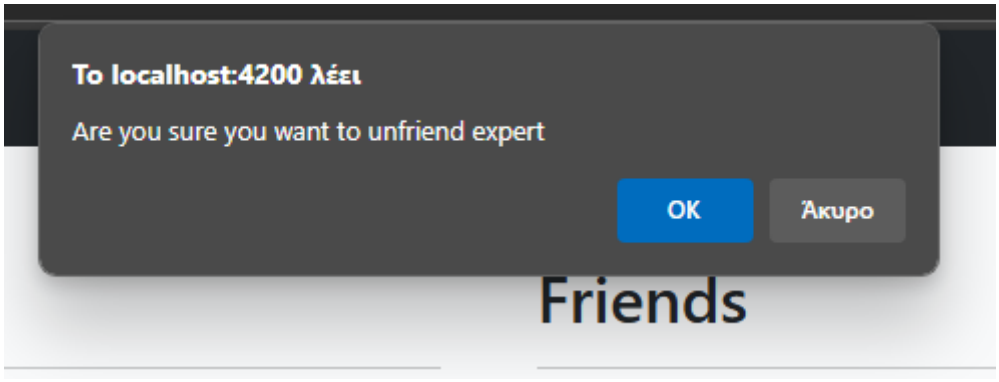
Στην συνέχεια, στην εφαρμογή backend εφαρμόζονται οι αλλαγές:

```
Visibility postVisibility =
Visibility.valueOf(request.getPostVisibility());
Visibility commentVisibility =
Visibility.valueOf(request.getCommentVisibility());
Visibility profileVisivility =
Visibility.valueOf(request.getProfileVisibility());
Visibility friendsVisibility =
Visibility.valueOf(request.getFriendsVisibility());
privacySettings.setPostVisibility(postVisibility);
privacySettings.setCommentVisibility(commentVisibility);
privacySettings.setProfileVisibility(profileVisivility);
privacySettings.setFriendsVisibility(friendsVisibility);

privacySettingsRepository.save(privacySettings);
```

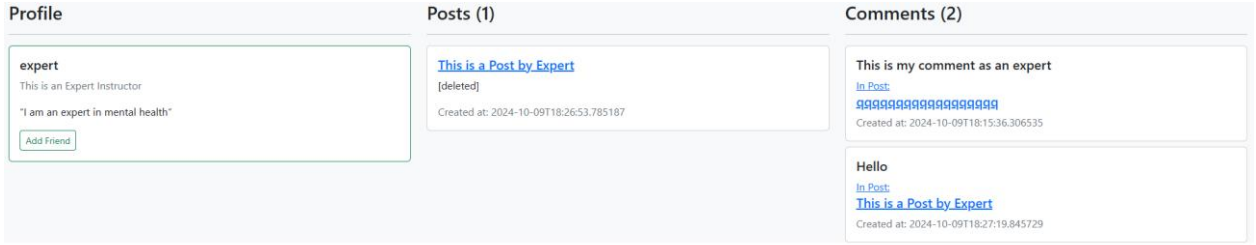
Η δεύτερη ενότητα αφορά τις ρυθμίσεις ιδιωτικότητας, όπου έχει υλοποιηθεί η αλλαγή του πεδίου του χρήστη: Βιογραφικό.

Τέλος, η τρίτη ενότητα αφορά τους φίλους του συνδεδεμένου χρήστη και για τον καθένα περιέχει το όνομα του χρήστη, το οποίο αποτελεί σύνδεσμο προς την σελίδα χρήστη και ένα κουμπί για διαγραφή της σχέσης φίλος με αυτόν. Κατά το πάτημα του κουμπιού αυτού ερωτάται ο χρήστης για επιβεβαίωση προς αποφυγή λαθών.



Σχήμα 30 Επιβεβαίωση διαγραφής φίλου

Προφίλ τρίτου χρήστη



Σχήμα 31 Οθόνη προφίλ τρίτου χρήστη

Στην σελίδα προφίλ τρίτου χρήστη, δηλαδή όχι του συνδεδεμένου, παρουσιάζονται τα εξής στοιχεία:

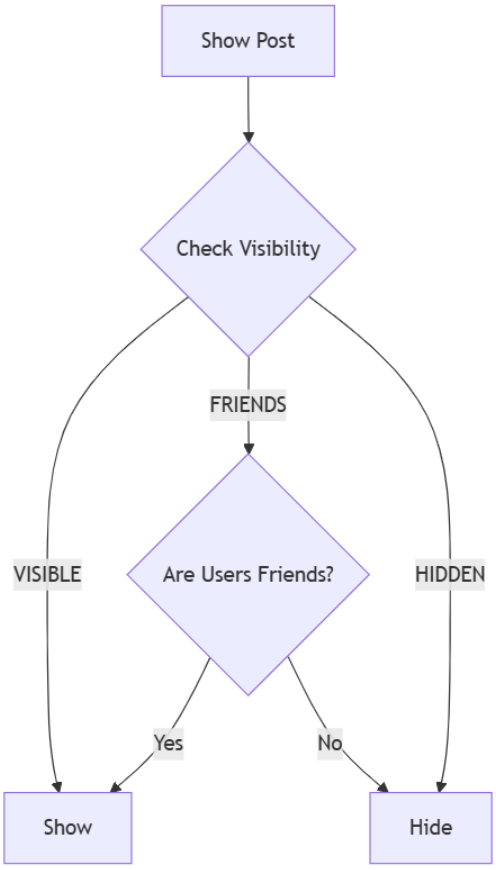
Το προφίλ του χρήστη που αναφέρονται ο όνομα χρήστη και το σύντομο βιογραφικό. Στο Σχήμα 31 παρουσιάζεται το προφίλ χρήστη ενός ειδικού, δηλαδή με ρόλο EXPERT, οπότε στις λεπτομέρειες προφίλ εμφανίζεται το διευκρινιστικό κείμενο «This is an Expert Instructor» και το πλαίσιο έχει πράσινο χρώμα.

Παρουσιάζονται επίσης οι αναρτήσεις του χρήστη, όπου ο τίτλος της ανάρτησης αποτελεί και σύνδεσμο προς την ανάρτηση αυτήν, και τα σχόλια του, που περιέχουν σύνδεσμο προς την ανάρτηση που σχολιάστηκε.

Τα στοιχεία ενός χρήστη Α που αναφέρθηκαν μπορούν να εμφανίζονται ή όχι στον χρήστη Β που επισκέπτεται την σελίδα προφίλ του πρώτου, ανάλογα τις ρυθμίσεις ιδιωτικότητας που έχει επιλέξει ο χρήστης Α.

Η λογική για να εμφανιστούν τα στοιχεία αυτά έχουν ως εξής: Για καθένα από τα στοιχεία του προφίλ ελέγχεται πρώτα η ανάλογη ρύθμιση. Εάν είναι κρυφή τότε δεν εμφανίζει τίποτα. Εάν η

ρύθμιση είναι το στοιχείο να είναι ορατό μόνο σε φίλους, τότε ελέγχεται εάν οι δύο χρήστες έχουν την σχέση φίλος, και μόνο αν η απάντηση είναι θετική επιστρέφει αποτέλεσμα. Τέλος εάν η ρύθμιση είναι να είναι δημόσιο τότε το στοιχείο εμφανίζεται πάντα.



Σχήμα 32 Διάγραμμα ροής για απόφαση εμφάνισης στοιχείου προφίλ

Παράδειγμα κώδικα όπου προσθέτει στην απάντηση του αιτήματος για το προφίλ του χρήστη, τις αναρτήσεις του.

```
if(force || shouldShowPosts(user)) {
    List<Post> userPosts = postRepository.findByUser(user);
    List<PostResponse> userPostResponses =
userPosts.stream().map(PostResponse::from).toList();
    builder.posts(userPostResponses);
}
```

Ο διαχειριστής του συστήματος μπορεί πάντα να βλέπει τα στοιχεία προφίλ χρήστη. Επιλέχθηκε αυτή λογική, για παράδειγμα σε περίπτωση αναφοράς σε ανάρτηση ή σχόλιο του χρήστη, ο διαχειριστής να μπορεί δει τα στοιχεία του προφίλ του, ασχέτως ρυθμίσεων.

Η μέθοδος shouldShowPosts ελέγχει εάν ο συνδεδεμένος χρήστης έχει το δικαίωμα να δει τις αναρτήσεις τρίτου χρήστη. Παράδειγμα κώδικα:

```
User loggedInUser = getLoggedInUser();
if(user.equals(loggedInUser)) {
    return true;
}
if(loggedInUser.getRole() == Role.ADMIN) {
    return true;
}
PrivacySettings privacySettings =
privacySettingsRepository.findByUser(user).orElseThrow();
if(privacySettings.getPostVisibility() == Visibility.VISIBLE) {
    return true;
}
return privacySettings.getPostVisibility() == Visibility.FRIENDS &&
        friendshipService.areFriends(user, loggedInUser);
```

Στην εφαρμογή frontend λαμβάνονται όσα στοιχεία στέλνει η εφαρμογή backend:

Το αίτημα HTTP GET προς την εφαρμογή backend:

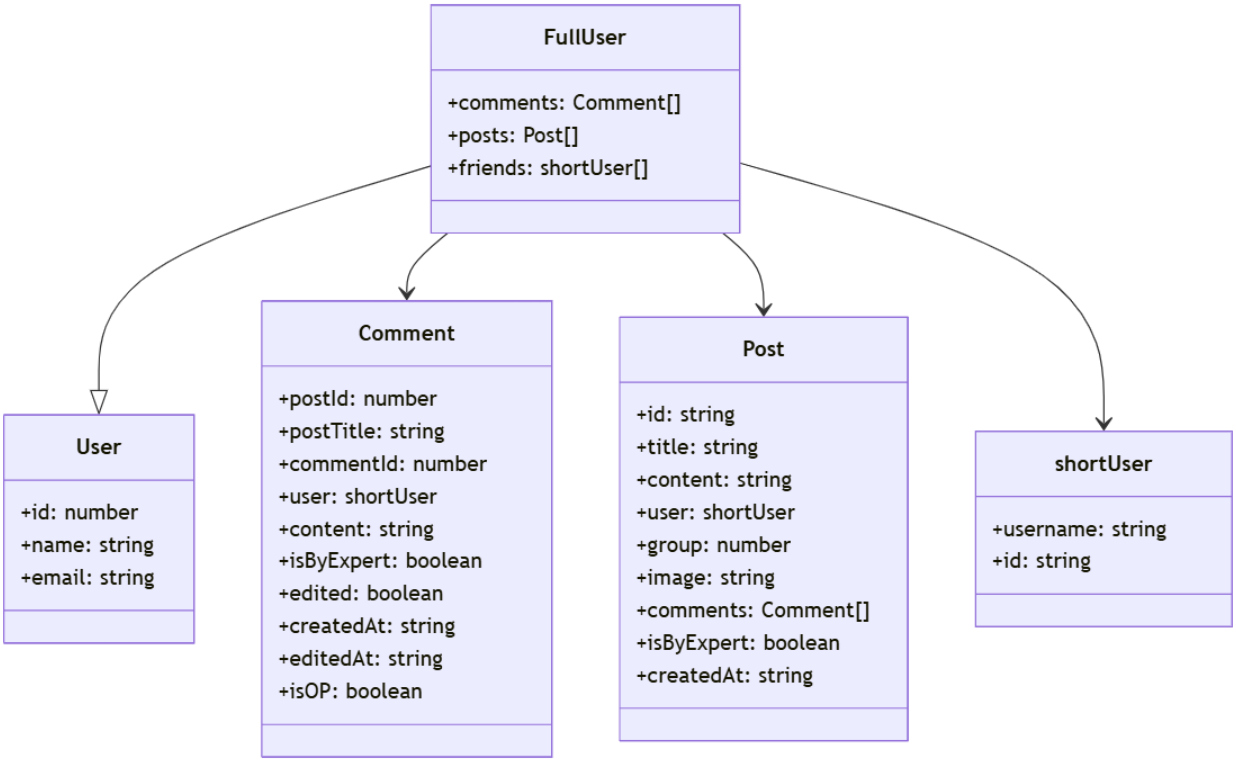
```
this.http.get<User>(`${environment.apiUrl}/users/${id}`);
```

και η αποθήκευση των πληροφοριών σε μεταβλητή:

```
this.userService.getUserById(this.id).subscribe({
      next: data => {
        this.user = data;
        this.isExpert = data?.role == Roles.EXPERT;

      },
      error: error => {
        this.alertService.error(`Couldn't retrieve info for user with
id=${this.id}. Error: ${error}`);
      }
    });
```

Όπου το αντικείμενο user, είναι της μορφής όπως παρουσιάζεται στο Σχήμα 33.



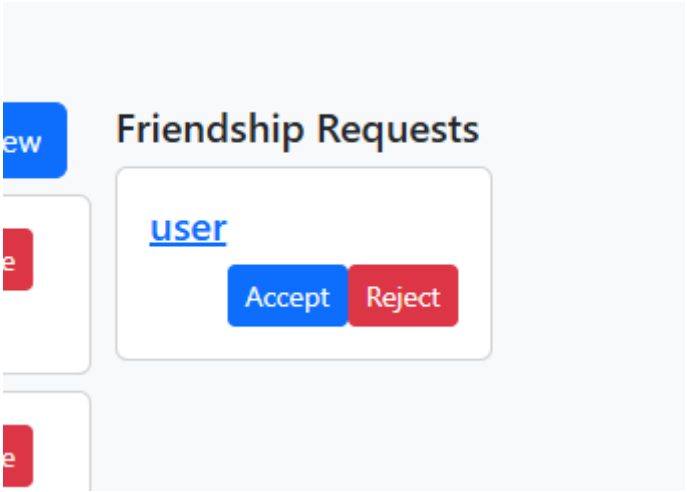
Σχήμα 33 Διάγραμμα κλάσεων αντικειμένου FullUser

Προσθήκη φίλου

Στο προφίλ χρήστη, όπως φαίνεται στο Σχήμα 31, υπάρχει κουμπί προσθήκης φίλου και πατώντας το, ο χρήστης δημιουργεί ένα νέο αίτημα φιλίας και ενημερώνεται ότι το αίτημα αυτό έχει αποσταλεί.

Σχήμα 34 Μήνυμα ενημέρωσης για αποστολή αιτήματος φιλίας

Ο αποδέκτης του αιτήματος φιλίας μπορεί να ενημερωθεί για αυτό στην αρχική του σελίδα, όπου εμφανίζεται το αίτημα αυτό με επιλογές την αποδοχή ή μη.



Σχήμα 35 Αποδοχή ή όχι αιτήματος φιλίας

Η υλοποίηση έχει γίνει χρησιμοποιώντας τον πίνακα Friendship_Request με πεδία τον αποστολέα, τον παραλήπτη και την κατάσταση του αιτήματος. Για την προβολή των αιτημάτων φιλίας ζητούνται οι εγγραφές του πίνακα Friendship_Request στις οποίες ο παραλήπτης είναι ο τρέχων χρήστης και η κατάσταση του αιτήματος είναι PENDING.

Παράδειγμα κώδικα στην εφαρμογή backend:

```
User fromUser = userService.getLoggedInUser();
User toUser = userRepository.findById(id).orElseThrow();
if(fromUser.equals(toUser)) {
    throw new Exception(fromUser.getUsername() + " and " +
toUser.getUsername() + " are the same");
}
if(friendshipService.areFriends(fromUser, toUser)) {
    throw new Exception(fromUser.getUsername() + " and " +
toUser.getUsername() + " are already friends");
}
if(friendshipService.hasPendingRequest(fromUser, toUser)) {
    throw new Exception(fromUser.getUsername() + " already has a
pending friend request to " + toUser.getUsername());

}
FriendshipRequest friendshipRequest = FriendshipRequest.builder()
                .fromUser(fromUser)
                .toUser(toUser)

.friendshipRequestStatus(FriendshipRequestStatus.PENDING)
                .build();
friendshipRequestRepository.save(friendshipRequest);
```

Ενώ στην εφαρμογή frontend πραγματοποιείται ένα αίτημα HTTP GET:

```
this.http.get<Boolean>(`${environment.apiUrl}/friends/sendRequest/${id
}`);
```

και ενημερώνεται ο χρήστης:

```
this.friendsService.sendFriendRequest(this.user?.id).subscribe({
      next:data => {
        this.loading = false;
        if(data) {
          this.alertService.info("Friend request sent");
        }
        else {
          this.alertService.error("Couldn't sent error request");
        }
      },
      error: error => {
        this.loading = false;
        this.alertService.error(`Couldn't sent error request:
${error}`);
      }
    })
  }
```

Εάν ο χρήστης έχει άλλο αίτημα προς τον ίδιο χρήστη σε εξέλιξη δεν μπορεί να αιτηθεί νέο.

Στην περίπτωση που αποδεκτεί ο παραλήπτης το αίτημα, τότε δημιουργείται μια νέα εγγραφή στον πίνακα Friendship με στοιχεία, τους δύο χρήστες και στην αντίστοιχη εγγραφή του πίνακα Friendship_Request το πεδίο Status σημειώνεται ως Accepted.

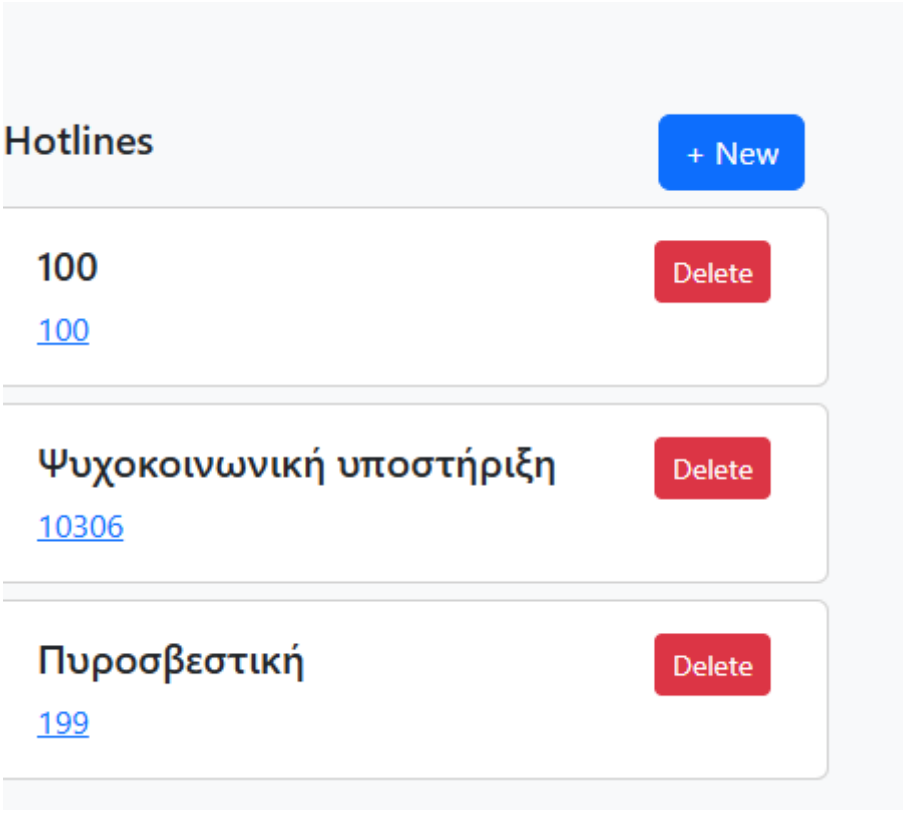
Παράδειγμα κώδικα:

```
Friendship friendship = Friendship.builder()
        .userId1(fromUser)
        .userId2(toUser)
        .build();
friendshipRepository.save(friendship);
friendshipRequest.setFriendshipRequestStatus(FriendshipRequestStatus.A
CCEPTED);
```

Οι δύο αυτοί χρήστες μπορούν να δουν τον άλλον στην λίστα φίλων και να επισκεφτούν το προφίλ του.

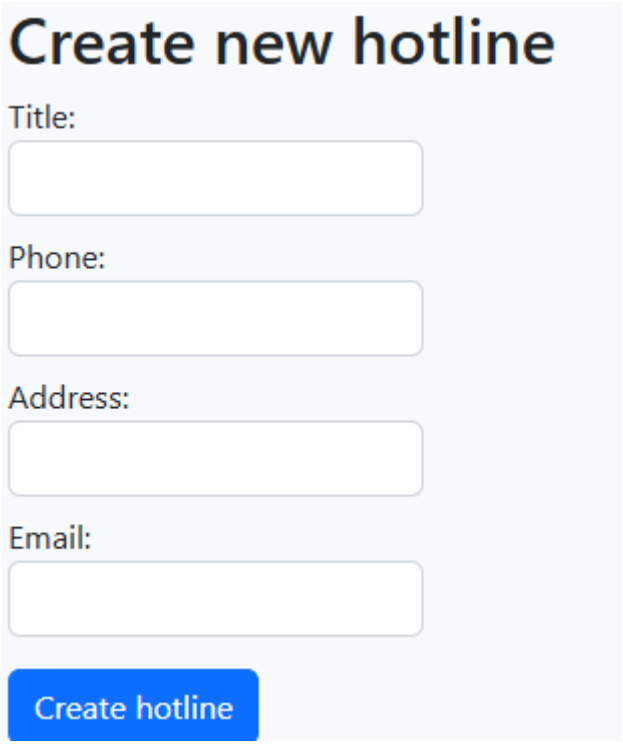
Δημιουργία επαφής έκτακτης ανάγκης

Ένας χρήστης με δικαιώματα Ειδικού, έχει δυνατότητα να δημιουργήσει νέες επαφές έκτακτης ανάγκης οι οποίες είναι ορατές σε όλους τους χρήστες στην αρχική τους οθόνη.



Σχήμα 36 Επαφές έκτακτης ανάγκης και κουμπί δημιουργία νέας

Στην δημιουργία νέας επαφής, ο ειδικός εισάγει τον τίτλο και ένα εκ των: τηλέφωνο, email, διεύθυνση. Ο κάθε τρόπος επικοινωνίας έχει και το ανάλογο URI schema για εύκολο άνοιγμα της κατάλληλης εφαρμογής. Το τηλέφωνο έχει το πρόθεμα tel:, το email mailto:.



Σχήμα 37 Δημιουργία νέας επαφής έκτακτης ανάγκης

Παράδειγμα κώδικα:

```
@PostMapping("/")
@PreAuthorize("hasRole('EXPERT')")
public ResponseEntity<Hotline> addHotline(@RequestBody HotlineRequest
request) {
    Hotline hotline = HotlineRequest.from(request);
    hotlineRepository.save(hotline);
    return ResponseEntity.ok(hotline);
}
```

Στην εφαρμογή frontend πραγματοποιείται αίτημα HTTP POST με τα δεδομένα που έχει εισάγει ο χρήστης:

```
this.http.post<Hotline>(`${environment.apiUrl}/hotlines/`, hotline);
```

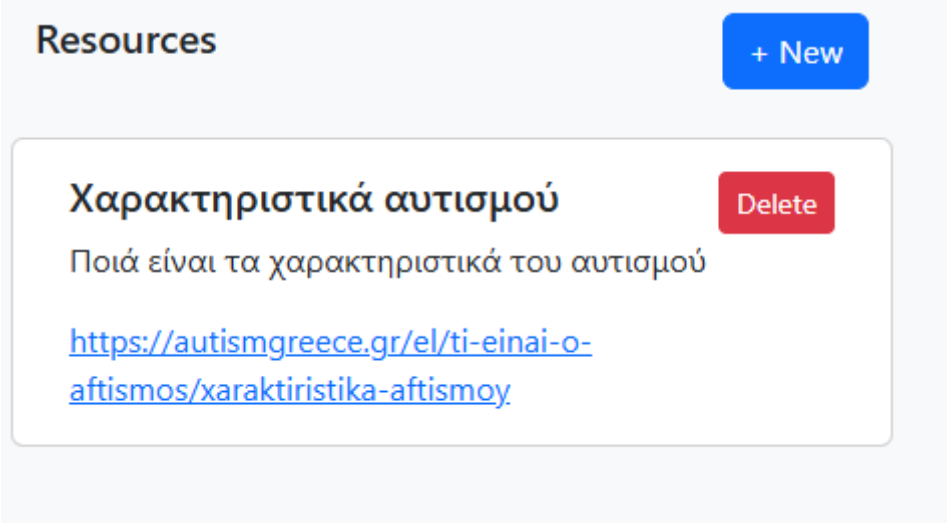
και στην συνέχεια ενημερώνεται ο χρήστης για το αποτέλεσμα και μεταφέρεται στην αρχική σελίδα:

```
this.hotlineService.createHotline(this.form.value).subscribe({
      next: data => {
        this.loading = false;
        if(data) {
          this.alertService.info("Hotline created");
```

```
          this.router.navigateByUrl("/");
        }
        else {
          this.alertService.error("Couldn't create hotline");
        }
      },
      error: error => {
        this.loading = false;
        this.alertService.error("Couldn't create hotline: " + error);
      }
    });
```

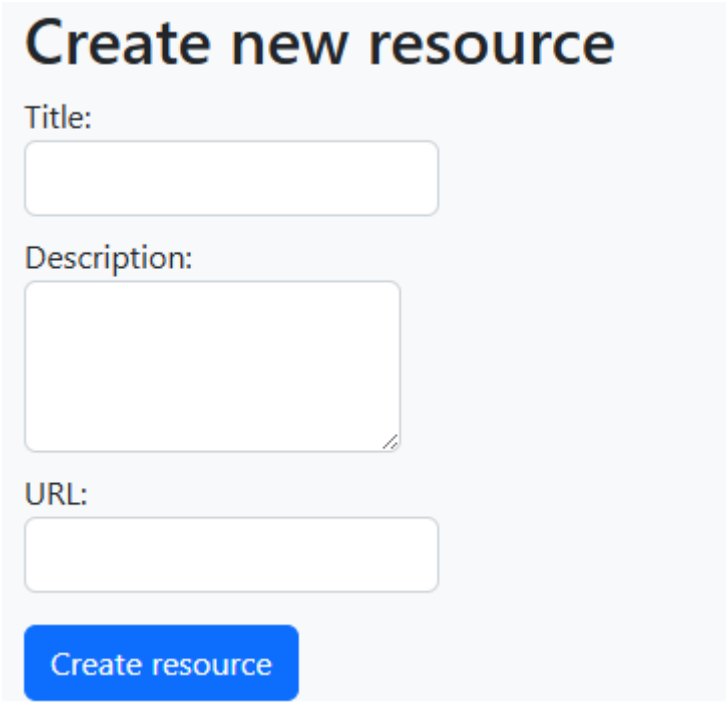
Πόροι ομάδας

Σε κάθε ομάδα στην οποία είναι εγγεγραμμένος ο ειδικός, μπορεί να δημιουργήσει και να διαγράψει πόρους για εύκολη πρόσβαση σε πληροφορίες από τους χρήστες.



Σχήμα 38 Πόροι ομάδας

Κάθε πόρος συνδέεται με μια ομάδα και περιέχει τίτλο, περιγραφή και URL.

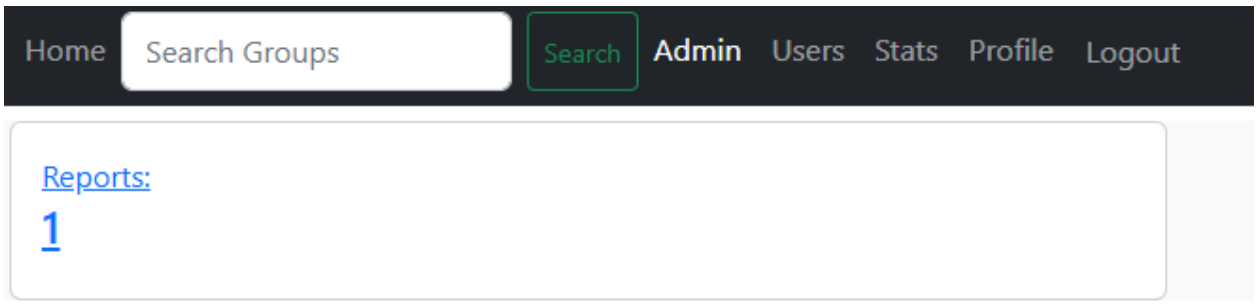


Σχήμα 39 Δημιουργία νέου πόρου

Διαχείριση αναφορών

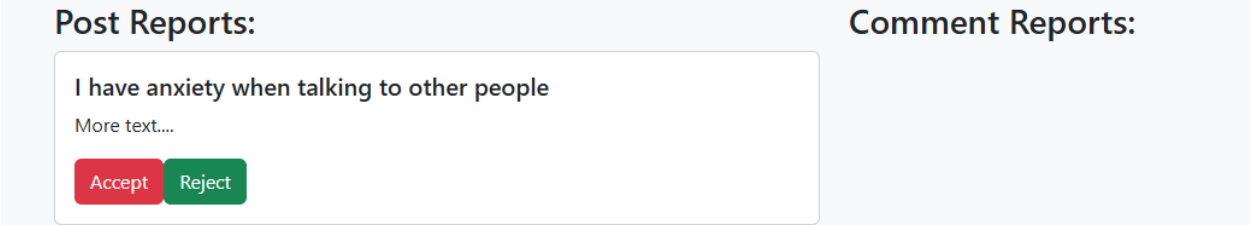
Ο διαχειριστής έχει την δυνατότητα να κάνει ενέργειες πάνω στις αναφορές αναρτήσεων και σχολίων που έχουν κάνει οι χρήστες.

Από τον σύνδεσμο Admin στην εργαλειοθήκη, ο διαχειριστής βλέπει αρχικά το πλήθος των αναφορών.



Σχήμα 40 Πλήθος αναφορών

Κάνοντας κλικ στον σύνδεσμο αυτό, ο διαχειριστής βλέπει ξεχωριστά τις αναφορές σχολίων και αναρτήσεων. Για κάθε μία από αυτές μπορεί να δει μια σύντομη προβολή του περιεχομένου, καθώς και ένα κουμπί για αποδοχή και ένα για απόρριψη της αναφοράς.



Σχήμα 41 Αναφορές

Για την προβολή των ανοικτών αναφορών ζητούνται από την εφαρμογή όλες οι εγγραφές των πινάκων Post_Report και Comment_Report με την ιδιότητα Status σε τιμή Open.

Παράδειγμα κώδικα δημιουργίας αναφοράς μέσω ORM:

```
PostReport postReport = new PostReport();
postReport.setUser(user);
postReport.setPost(post);
postReport.setReportStatus(ReportStatus.OPEN);
postReportRepository.save(postReport);
```

Στην frontend εφαρμογή πραγματοποιείται ως εξής:

Πραγματοποιείται ένα αίτημα HTTP GET:

```
this.http.get<PostReport>(`${environment.apiUrl}/post/${id}/report`);
```

και στην συνέχεια ενημερώνεται ο χρήστης για το αποτέλεσμα:

```
this.postsService.report(this.postId).subscribe({
      next: data => {
        this.alertService.warn("Reported!");
        this.loadingReport = false;
      },
      error: error => {
        this.alertService.error(error);
        this.loadingReport = false;
      }
    });
```

Κάθε ένα από αυτά τα κουμπιά θέτουν την κατάσταση της αναφοράς στον πίνακα Post_Report ή Comment_Report σε κλειστό. Η αποδοχή της αναφοράς διαγράφει το περιεχόμενο της ανάρτησης ή του σχολίου.

```
PostReport postReport = getReport(id);
Post post = postReport.getPost();
post.setContent("[deleted]");
postRepository.save(post);
postReport.setReportStatus(ReportStatus.CLOSED);
postReportRepository.save(postReport);
```

Στην εφαρμογή frontend για την αποδοχή της αναφοράς:

```
this.http.get<PostReport>(`${environment.apiUrl}/reports/post/${id}/ac
cept`);
```

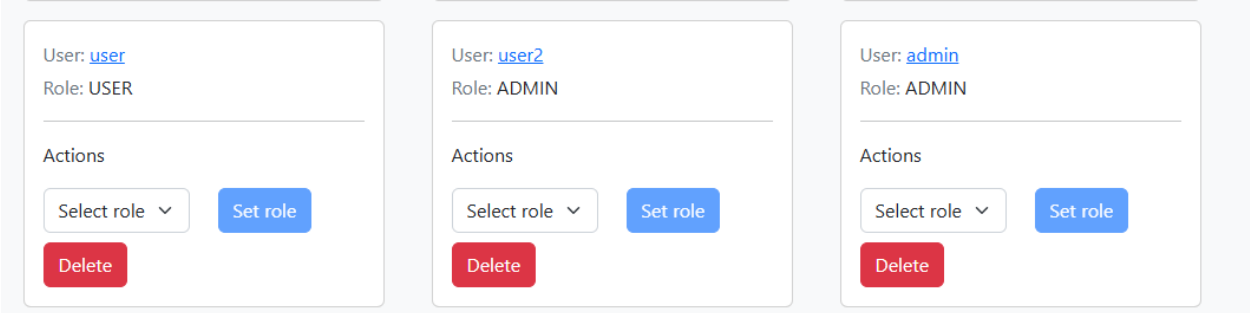
και άρνηση της αναφοράς:

```
this.http.get<PostReport>(`${environment.apiUrl}/reports/post/${id}/re
ject`);
```

πραγματοποιείται ένα αίτημα HTTP GET και στην συνέχεια ενημερώνεται ο χρήστης για το αποτέλεσμα, και ανανεώνονται τα δεδομένα:

```
this.reportsService.acceptPostReport(id).subscribe({
      next: data => {
        this.alertService.info("accepted");
        this.ngOnInit();
      },
      error: error => {
        this.alertService.error(error);
      }
    });
```

Διαχείριση χρηστών



Σχήμα 42 Διαχείριση χρηστών

Στην σελίδα διαχείρισης χρηστών της εφαρμογής, διαθέσιμο με τον σύνδεσμο ‘Users’ στην γραμμή εργαλείων, ένας διαχειριστής μπορεί να δει όλους τους χρήστες της εφαρμογής και να εκτελέσει διάφορες εργασίες.

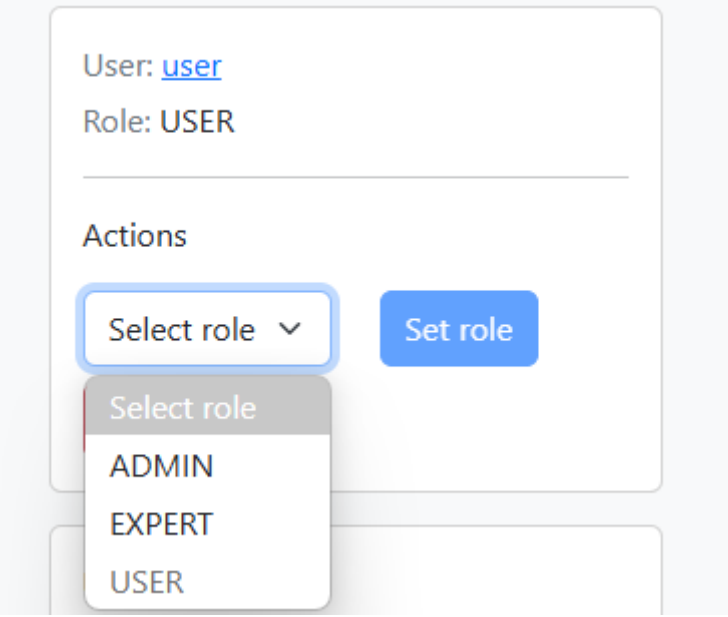
Όπως φαίνεται στην εικόνα για κάθε χρήστη ο διαχειριστής βλέπει το όνομα χρήστη, το οποίο είναι και σύνδεσμος προς το προφίλ του, τον ρόλο του, και στην συνέχεια, μπορεί να αλλάξει τον ρόλο του χρήστη καθώς και να τον διαγράψει.

Σε αυτήν την σελίδα ένας διαχειριστής μπορεί να αλλάξει τον ρόλο ενός απλού χρήστη σε ειδικό, με τις επιπλέον λειτουργίες που αυτό φέρνει, εάν ο χρήστης δώσει τα απαραίτητα δικαιολογητικά ότι είναι ειδικός της ψυχικής υγείας, στον διαχειριστή.

Στην εφαρμογή υπάρχουν τρεις προκαθορισμένοι ρόλοι:

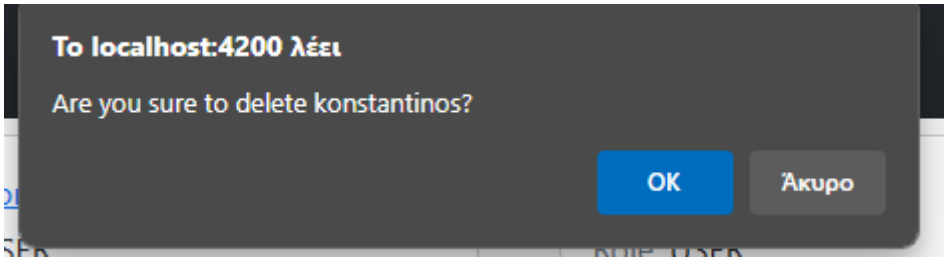
1. USER, απλός χρήστης
2. EXPERT, ειδικός ψυχικής υγείας
3. ADMIN, διαχειριστής εφαρμογής

Κατά την αλλαγή ρόλου ενός χρήστη, στον πίνακα User, αλλάζει η τιμή της ιδιότητας role σε μία από τις παραπάνω τιμές. Λόγω χρήσης ORM στην εφαρμογή αυτή η αλλαγή πραγματοποιείται ως αλλαγή της ιδιότητας role του αντικειμένου user: user.setRole(role);



Σχήμα 43 Επιλογή ρόλου χρήστη

Τέλος, ο διαχειριστής μπορεί να διαγράψει έναν χρήστη. Για αποφυγή λαθών παρουσιάζεται ένα αναδυόμενο παράθυρο επιβεβαίωσης.



Σχήμα 44 Παράθυρο επιβεβαίωσης διαγραφής χρήστη

Στατιστικά εφαρμογής

Ο διαχειριστής μπορεί να δει τα συγκεντρωτικά στατιστικά στοιχεία της εφαρμογής, που περιλαμβάνουν, το πλήθος των χρηστών, το πλήθος των ομάδων, το πλήθος των αναρτήσεων το πλήθος των σχολίων καθώς και το πλήθος των χρηστών ανά ρόλο.

Σχήμα 45 Συγκεντρωτικά στοιχεία εφαρμογής

Βιβλιογραφία

Ξενόγλωσση

1. Sabharwal, C. L. (1998). Java, java, java. IEEE Potentials, 17(3), σ. 33-37.
2. Bierman, G., Abadi, M., & Torgersen, M. (2014). Understanding typescript. Στο ECOOP 2014–Object-Oriented Programming: 28th European Conference, Uppsala, Sweden, July 28–August 1, 2014. Proceedings 28 (pp. 257-281). Springer Berlin Heidelberg. σ. 1
3. O'Neil Elizabeth J. (2008). Object/relational mapping 2008: hibernate and the entity data model (edm). In Proceedings of the 2008 ACM SIGMOD international conference on Management of data (SIGMOD '08), New York, NY, USA, Association for Computing Machinery 1351–1356.
4. Hunt, J. (1997). The Model-View-Controller Architecture. In: Smalltalk and Object Orientation. Springer, London
5. Breitman, K., & Leite, J. C. S. P. (2002). Managing user stories. Στο *International Workshop on Time-Constrained Requirements Engineering* σ. 168
6. Kontogianni, Aristea & Kabassi, Katerina & Virvou, Maria & Alepis, Efthymios. (2018). Smart Tourism Through Social Network User Modeling: A Literature Review. 1-4. 10.1109/IISA.2018.8633633.
7. Papageorgiou, Achilleas & Strigkos, Michael & Politou, Eugenia & Alepis, Efthymios & Solanas, Agusti & Patsakis, Constantinos. (2018). Security and Privacy Analysis of Mobile Health Applications: The Alarming State of Practice. IEEE Access. PP. 1-1. 10.1109/ACCESS.2018.2799522.
8. Alepis, Efthymios & Lambrinidis, Christos. (2013). M-health: Supporting automated diagnosis and electonic health records. SpringerPlus. 2. 103. 10.1186/2193-1801-2-103.
9. Virvou, Maria & Alepis, Efthymios. (2006). Mobile and electronic medical support and education for dyslexic students. Proceedings of the 7th International Workshop on Mathematical Methods in Scattering Theory and Biomedical Engineering. 431-438. 10.1142/9789812773197_0043.
10. A. Karavokyris and E. Alepis, "Software Measures for Common Design Patterns Using Visual Studio Code Metrics," *2018 9th International Conference on Information, Intelligence, Systems and Applications (IISA)*, Zakynthos, Greece, 2018, pp. 1-7, doi: 10.1109/IISA.2018.8633694.
11. P. Pavlakis, E. Alepis and M. Virvou, "Intelligent Mobile Multimedia Application for the Support of the Elderly," *2012 Eighth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, Piraeus-Athens, Greece, 2012, pp. 297-300, doi: 10.1109/IIH-MSP.2012.78.
12. Hatzilygeroudis, Ioannis & Tsihrintzis, George & Virvou, Maria & Perikos, Isidoros. (2022). Special issue on information, intelligence, systems and applications. Neural Computing and Applications. 35. 10.1007/s00521-022-07954-3.
13. Politou, Eugenia & Alepis, Efthymios & Virvou, Maria & Patsakis, Constantinos. (2022). Privacy and Data Protection Challenges in the Distributed Era. 10.1007/978-3-030-85443-0.
14. Krouska, Akrivi & Troussas, Christos & Virvou, Maria. (2017). Social networks as a learning environment: Developed applications and comparative analysis. 1-6. 10.1109/IISA.2017.8316430.
15. Virvou, Maria & Alepis, Efthymios. (2013). User Modeling in Mobile Learning Environments for Learners with Special Needs. Smart Innovation, Systems and Technologies. 25. 7-17. 10.1007/978-3-319-00375-7_2.
16. Kabassi, Katerina & Virvou, Maria & Tsihrintzis, George. (2006). Hybrid Intelligent Medical Tutor for Atheromatosis. 4252. 1289-1296. 10.1007/11893004_163.
17. Tsihrintzis, George & Virvou, Maria & Bourbakis, Nikolaos & Jain, Lakhmi. (2024). Introduction to Advances in Information, Intelligence, Systems and Applications. 10.1007/978-3-031-67426-6_1.
18. Politou, Eugenia & Alepis, Efthymios & Virvou, Maria & Patsakis, Constantinos. (2021). Privacy in the COVID-19 Era. 10.1007/978-3-030-85443-0_9.

19. Kastanas, George & Sakkopoulos, Evangelos. (2022). Mobile student information services: A case study in Greek Open University. 1-4. 10.1109/IISA56318.2022.9904343.
20. Piotopoulos, Stavros & Sakkopoulos, Evangelos. (2021). Transforming procedures to web applications using IFML: The new Greek Citizenship Test. 1-6. 10.1109/IISA52424.2021.9555545.
21. Paschou, Mersini & Nodarakis, Nikolaos & Tsakalidis, Athanassios & Sakkopoulos, Evangelos. (2013). MOBILE HEALTHCARE SYSTEMS: GENERATING DYNAMIC SMARTPHONE APPS TO SERVE MULTIPLE MEDICAL SPECIALIZATIONS 6 TH INTERNATIONAL CONFERENCE ON HEALTH INFORMATICS FEBRUARY 11-14, BARCELONA, SPAIN.
22. Dimitris, Antoniou & Garofalakis, John & Makris, Christos & Panagis, Yannis & Sakkopoulos, Evangelos. (2010). Context-similarity based hotlinks assignment: Model, metrics and algorithm. Data & Knowledge Engineering. 69. 357-370. 10.1016/j.datak.2009.04.007.
23. Sakkopoulos, Evangelos & Sourla, Efrosini & Tsakalidis, Athanasios & Lytras, Miltiadis. (2008). Integrated system for e-health advisory web services provision using broadband networks. International Journal of Social and Humanistic Computing. 1. 10.1504/IJSHC.2008.020479.
24. Garofalakis, John & Matsoukas, T. & Panagis, Yannis & Sakkopoulos, Evangelos & Tsakalidis, Athanasios. (2005). Personalization Techniques for Web Search Results Categorization. 148 - 151. 10.1109/EEE.2005.101.
25. Paschou, Mersini & Sakkopoulos, Evangelos & Tsakalidis, Athanasios. (2013). easyHealthApps: e-Health Apps Dynamic Generation for Smartphones & Tablets. Journal of medical systems. 37. 9951. 10.1007/s10916-013-9951-6.

Ιστοσελίδες

1. World Health Organization. World mental health report: transforming mental health for all. Στο: https://iris.who.int/bitstream/handle/10665/356119/9789240049338-eng.pdf?sequence=1 (προσπελάστηκε στις 19/1/2025)
2. MariaDB in brief. Στο https://mariadb.org/en/ (Προσπελάστηκε στις 21/1/2025)
3. What is Java Spring Boot? Στο https://www.ibm.com/think/topics/java-spring-boot (Προσπελάστηκε στις 21/1/2025)
4. What is Angular? Στο https://angular.dev/overview (Προσπελάστηκε στις 22/1/2025)