



Προγραμματιστική Εργασία: Εφαρμογή Social Media App σε iOS με Node.js/MongoDB Backend

Αναγνωστόπουλος Βασίλειος 03153 και Γεωργιτζίκη Γαρυφαλιά 03218

Προχωρημένη Διαχείριση Δεδομένων 2023-24

8° Εξάμηνο

Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Πανεπιστήμιο Θεσσαλίας, Βόλος

{vanagnostop, ggeorgitziki}@e-ce.uth.gr



SocialWave

Περιεχόμενα

1	Επιλογή Θέματος	3
2	Περιγραφή και προδιαγραφές του θέματος	3
2.1	Περιγραφή	3
2.2	One-Time-Password (OTP)	4
2.3	Αυτόματη Σύνδεση με χρήστη Token	4
2.4	Friend Suggestion Algorithm	4
2.5	Socket Programming για Real-time Ειδοποίησεις και Chat	5
2.6	HTTPS Server	6
2.7	Hashing στον Κωδικό Πρόσβασης του χρήστη	6
3	Επιλογή Συστήματος NoSQL ΒΔ κατάλληλου για εφαρμογή Social Media ..	6
4	Εγκατάσταση του Συστήματος NoSQL MongoDB	6
5	Προσδιορισμός χρήσιμων ερωτημάτων για την εφαρμογή	7
6	Δημιουργία δεδομένων με χρήση data generator ή ανεύρεση έτοιμων δεδομένων	7
6.1	Γενικά	7
6.2	Ανάλυση python script <code>creationOfDatabase.py</code>	8
6.2.1	Τυχαία δεδομένα για τον χρήστη	
		8
6.2.2	Τυχαία δεδομένα για την δημοσίευση(post)	
		11
6.2.3	Τυχαία δεδομένα για τα σχόλια σε δημοσιεύσεις(comments)	
		12
7	Σχεδιασμός του μοντέλου της ΒΔ και υλοποίησή του στο Σύστημα NoSQL ΒΔ	13
8	Δημιουργία κώδικα για λειτουργίες CRUD (Create, read, update and delete) δεδομένων	20
8.1	Λειτουργίες create δεδομένων στην βάση δεδομένων	20
8.2	Λειτουργίες read δεδομένων στην βάση δεδομένων	23
8.3	Λειτουργίες update δεδομένων στην βάση δεδομένων	25
8.4	Λειτουργίες delete δεδομένων στην βάση δεδομένων	30
9	Υλοποίηση των ερωτημάτων στη ΒΔ	31
10	Λίστα με τα αρχεία που παραδίδονται και τι περιλαμβάνει το καθένα	35
11	Οδηγίες εγκατάστασης της εφαρμογής και φόρτωση δεδομένων	36
12	Οδηγίες χρήσης και παράδειγμα εκτέλεσης της εφαρμογής με παρουσίαση της διεπαφής χρήστη για λειτουργίες CRUD και για την εκτέλεση των ερωτημάτων στη ΒΔ, εκτέλεση των ερωτημάτων και εξέταση της ορθότητας των αποτελεσμάτων	43
13	Μελλοντικές επεκτάσεις και προσθήκες	62

14 Βιβλιογραφία	62
---------------------------	----

1 Επιλογή Θέματος

Το θέμα που επιλέχθηκε είναι η δημιουργία ενός πλήρως λειτουργικού mobile app Social Media σε λειτουργικό iOS, με χρήση της NoSQL βάσης δεδομένων MongoDB και σύνδεσή της με frontend σε Node.js. Το όνομα της εφαρμογής είναι “SocialWave” και παρακάτω φαίνεται το logo της.



Σχήμα 1. SocialWave App Logo

2 Περιγραφή και προδιαγραφές του θέματος

2.1 Περιγραφή

Η εφαρμογή “SocialWave” είναι ένα Social Media App, με σκοπό την άμεση αλληλεπίδραση και επικοινωνία των χρηστών. Συγκεκριμένα παρέχει στους χρήστες την δυνατότητα:

- Να δημιουργούν το προσωπικό τους λογαριασμό, με επιβεβαίωση του email τους με “One-Time-Password” (“OTP”), και να συνδέονται σε αυτόν.
- Να αλλάζουν τον κωδικό πρόσβασης τους, με επιβεβαίωση του email τους με “One-Time-Password” (“OTP”) για ασφάλεια από κακόβουλους χρήστες.
- Να στέλνουν αιτήματα φιλίας (“Friends Requests”) σε άτομα με τα οποία θέλουν να γίνουν φίλοι στο App.
- Να δέχονται (“Accept”) ή να απορρίπτουν (“Reject”) αιτήματα φιλίας άλλων χρηστών.
- Να διαγράφουν φίλους που δεν θέλουν πλέον.
- Να βρίσκουν πιθανούς φίλους με την λειτουργία “Suggest Friends” που θα αναλυθεί στην συνέχεια.
- Να αναζητούν χρήστες και εφόσον είναι φίλοι να περιηγούνται στο προφίλ τους, ειδάλλως το προφίλ φαίνεται «κλειδωμένο».
- Να ανεβάζουν δημοσιεύσεις (“Posts”) με φωτογραφίες και περιγραφή.

- Να αντιδρούν με “Like” σε δημοσιεύεις των φίλων τους που τους αρέσουν.
- Να σχολιάζουν (“Comment”) δημοσιεύσεις φύλων τους.
- Να αντιδρούν με “Like” στα “Comments” δικών τους ή άλλων δημοσιεύσεων.
- Να περιηγούνται στο δικό τους προφίλ και να επεξεργάζονται τα προσωπικά τους δεδομένα ή να διαγράφουν προηγούμενες δημοσιεύσεις τους.
- Να λαμβάνουν ειδοποιήσεις για “Likes” και “Comments” σε δημοσιεύσεις τους και για αιτήματα φιλίας που λάβανε.
- Να επικοινωνούν με άλλους χρήστες μέσω προσωπικής συζήτησης (“Chat”) και να λαμβάνουν ειδοποιήσεις για τυχόν νέα μηνύματα.

2.2 One-Time-Password (OTP)

Το One-Time-Password (OTP) είναι ένας εξαγήφιος ασφαλής κωδικός μίας χρήσης που αποστέλλεται μέσω του email no-reply.socialwave2024@gmail.com και χρησιμοποιείται για την επιβεβαίωση διαδικτυακών ενεργειών όπως συνδέσεις ή συναλλαγές. Στην περίπτωση της εφαρμογής μας ο κωδικός αυτός αποστέλλεται για την επιβεβαίωση του email του χρήστη κατά την διαδικασία της εγγραφής του. Όταν ένας χρήστης συμπληρώσει όλα τα στοιχεία του και πατήσει ”Sing Up” παράγεται ένας τυχαίος εξαγήφιος αριθμός και αποστέλλεται στο εγγεγραμμένο email του. Ο χρήστης έχει στην διάθεση του 10 λέπτα πριν λήξει ο κωδικός για να τον εισάγει στο ειδικό πεδίο της εφαρμογής. Αυτή η διαδικασία διασφαλίζει ότι το email που πληκτρολόγησε ο χρήστης στα στοιχεία εγγραφής του είναι σωστό και μπορεί να ολοκληρωθεί η διαδικασία εγγραφής. Ο κωδικός OTP χρησιμοποιείται επίσης κατά την αλλαγή κωδικού πρόσβασης ώστε να μην μπορεί οποιοσδήποτε, γνωρίζοντας μόνο το email κάποιου, να αλλάξει τον κωδικό πρόσβασης του. Με αυτόν τον τρόπο, προστατεύεται τον χρήστη από άλλους κακόβουλους χρήστες που μπορεί να προσπαθήσουν να δημιουργήσουν ένα ψεύτικο προφίλ με τα στοιχεία του χρήστη εν αγνοία του ή να του υποκλέψουν τον λογαριασμό.

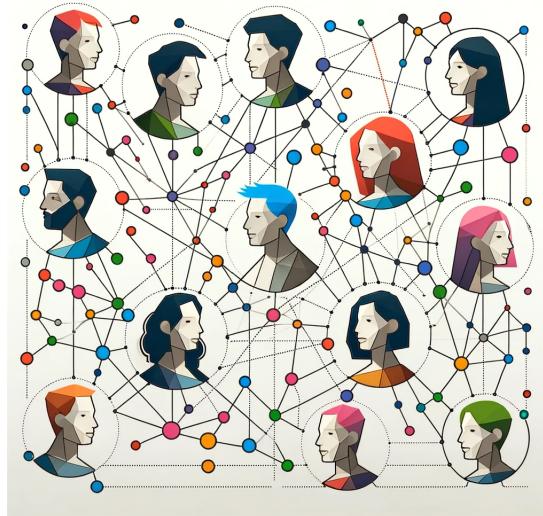
2.3 Αυτόματη Σύνδεση με χρήση Token

Στην εφαρμογή έχει προστεθεί η λειτουργία της Αυτόματης Σύνδεσης (Auto-Login) με χρήση token JWT (JSON Web Token) που επιτρέπει στους χρήστες να παραμένουν συνδεδεμένοι χωρίς να εισάγουν επανειλημμένα τα στοιχεία τους (email και password). Κατά την αρχική σύνδεση, ο server δημιουργεί ένα μοναδικό token και το στέλνει στη συσκευή του χρήστη. Αυτό το token αποθηκεύεται με ασφάλεια και αποστέλλεται ξανά στον server, ως αναγνωριστικό, σε επόμενες επισκέψεις στην εφαρμογή. Το token στέλνεται στο header οποιουδήποτε HTTPS Request. Αν είναι έγκυρο, ο server επαληθεύει τον χρήστη, αλλιώς ο client αποσυνδέεται αυτόματα και οδηγείται στο Login Screen. Τα tokens έχουν ορισμένη διάρκεια ζωής και εάν λήξουν ο χρήστης θα πρέπει να επαληθεύσει ξανά τα στοιχεία του πάλι μέσω του Login Screen. Αυτή η μέθοδος εξασφαλίζει μια ασφαλή εμπειρία για τον χρήστη.

2.4 Friend Suggestion Algorithm

Η λειτουργία Friend Suggestion έχει δημιουργηθεί για να προτείνει στους χρήστες άτομα που ίσως γνωρίζουν με βάση τους κοινούς τους φίλους. Συγκεκριμένα, πίσω από την λειτουργία αυτή τρέχει ένας αλγόριθμος που ακολουθεί τα εξής βήματα:

1. Αντλεί την λίστα φίλων του τρέχοντος χρήστη από την βάση δεδομένων.
2. Για κάθε φίλο του χρήστη, αντλεί την λίστα με τους φίλους του και δημιουργεί μία νέα λίστα με τους χρήστες αυτούς (οι φίλοι των φίλων του τρέχοντος χρήστη).
3. Χρήστες που είναι φίλοι με πάνω από έναν φίλο του χρήστη τοποθετούνται πιο ψηλά στην λίστα, με αποτέλεσμα να δημιουργείται μία λίστα όπου στην κορυφή της βρίσκονται χρήστες που έχουν τους περισσότερους κοινούς φίλους με το τρέχον χρήστη. Κατά αυτόν τον τρόπο, είναι πιο πιθανό ο τρέχων χρήστης να γνωρίζει τους πρώτους χρήστες της λίστας ή να ανήκουν σε ίδια ή παρόμοια κοινότητα.
4. Τέλος, εμφανίζει αυτούς τους χρήστες ανά 6 και μετά από δύο δημοσιεύσεις ή εάν υπάρχουν λιγότερες μετά από την μία δημοσίευση ή αν δεν υπάρχουν εμφανίζεται μόνο αυτή η λίστα, ξεκινώντας από τους πρώτους της λίστας, στην ειδική side scroll λίστα “Suggested For You” του “Home” Page της εφαρμογής.



Σχήμα 2. Friends Suggestion Algorithm

2.5 Socket Programming για Real-time Ειδοποιήσεις και Chat

Το Socket Programming χρησιμοποιήθηκε για την αποστολή Ειδοποιήσεων (Notifications) για ενέργειες όπως Likes, Comments και νέα μηνύματα, και για την επικοινωνία των χρηστών μέσω Chat. Συγκεκριμένα τα sockets που χρησιμοποιούνται ακούνε για εισερχόμενες ειδοποιήσεις ή μηνύματα και διαχειρίζονται αυτά τα δεδομένα. Η δημιουργία των sockets έγινε με την χρήση της βιβλιοθήκης ”socket” της Python.

2.6 HTTPS Server

Η χρήση ενός HTTPS Server στην εφαρμογή μας διασφαλίζει την ασφαλή επικοινωνία μεταξύ του app που χρησιμοποιεί ο χρήστης και του server. Το HTTPS κρυπτογραφεί τα δεδομένα, προστατεύοντας ευαίσθητες πληροφορίες όπως κωδικούς πρόσβασης, μηνύματα και προσωπικά στοιχεία από υποκλοπή και παραβίαση. Για να εφαρμοστεί το HTTPS, δημιουργήθηκε ένα πιστοποιητικό SSL από το Key Chain Access του MacBook που περιείχε ένα public και ένα private key, τα οποία τοποθετήθηκαν κατάλληλα στον Node.js server. Η επικοινωνία και το verification της σύνδεσης γίνεται μέσω του public key μεταξύ του client και του server, εξασφαλίζοντας ότι όλα τα μεταδιδόμενα δεδομένα είναι κρυπτογραφημένα και ασφαλή.

2.7 Hashing στον Κωδικό Πρόσβασης του χρήστη

Το hashing στον κωδικό πρόσβασης του χρήστη γίνεται σε 10 γύρους και περιλαμβάνει την επανειλημμένη εφαρμογή μιας κρυπτογραφικής συνάρτησης hashing στον κωδικό πρόσβασης, συνολικά 10 φορές. Κάθε γύρος χρησιμοποιεί την έξοδο του προηγούμενου γύρου ως είσοδο, καθιστώντας το τελικό hash πιο ασφαλή από επιθέσεις. Αυτή η διαδικασία πολλαπλών γύρων εξασφαλίζει μεγαλύτερη προστασία του κωδικού πρόσβασης.

3 Επιλογή Συστήματος NoSQL ΒΔ κατάλληλου για εφαρμογή Social Media

Το Σύστημα NoSQL βάσης δεδομένων που επιλέχθηκε είναι η MongoDB. Η επιλογή έγινε με βάση τις δυνατότητες που μπορεί να προσφέρει η MongoDB σε ένα Social Media App. Αναλυτικά, η MongoDB είναι κατάλληλη για εφαρμογές κοινωνικών μέσων λόγω της ευελιξίας και της απόδοσής της. Ως βάση δεδομένων NoSQL, επιτρέπει την αποθήκευση δεδομένων σε έγγραφα JSON-like, κάτι που ταιριάζει με τη δομή των δεδομένων κοινωνικών μέσων όπως προφίλ χρηστών, δημοσιεύσεις, likes, σχόλια και λίστες φίλων. Η δυνατότητα για γρήγορη αποθήκευση και ανάκτηση μεγάλων όγκων δεδομένων σε πραγματικό χρόνο είναι κρίσιμη για την εμπειρία του χρήστη. Επίσης, η κλιμάκωση της MongoDB είναι απλή, επιτρέποντας στην εφαρμογή να διαχειριστεί την αυξημένη χρήση και τα δεδομένα καθώς μεγαλώνει η βάση χρηστών. Επιπλέον, υποστηρίζει προηγμένα ερωτήματα, πλήρη αναζήτηση κειμένου και αναλύσεις σε πραγματικό χρόνο, παρέχοντας τη δυνατότητα ανάλυσης της συμπεριφοράς των χρηστών και την προσαρμογή του περιεχομένου. Αυτά τα χαρακτηριστικά καθιστούν τη MongoDB μια ισχυρή επιλογή για την ανάπτυξη της εφαρμογής μας.

4 Εγκατάσταση του Συστήματος NoSQL MongoDB

Το σύστημα NoSQL MongoDB, δεν απαιτεί την εγκατάσταση του λογισμικού τοπικά στον υπολογιστή καθώς παρέχει την υπηρεσία MongoDB Atlas στο διαδίκτυο. Κατά αυτόν τον τρόπο, η βάση δεδομένων δημιουργήθηκε στο cloud χρησιμοποιώντας την πλατφόρμα Atlas, η οποία παρέχει όλα τα χαρακτηριστικά και τις δυνατότητες της

MongoDB με διαχείριση, ασφάλεια και κλιμάκωση που προσφέρονται από την ίδια την MongoDB. Βήματα που ακολουθήθηκαν για την χρήση MongoDB Atlas:

1. Εγγραφή στην MongoDB Atlas: Δημιουργήθηκε ένας λογαριασμός στη MongoDB Atlas [εδώ](#).
2. Δημιουργία Cluster: Δημιουργήθηκε ένα νέο cluster μέσω του πίνακα ελέγχου της Atlas.
3. Ρύθμιση Βάσης Δεδομένων: Δημιουργήθηκε η βάση δεδομένων και τα collection της μέσω της διεπαφής του Atlas.
4. Σύνδεση με Εφαρμογή: Δημιουργήθηκε ένας Node.js server για την ένωση της βάσης δεδομένων με το UI της εφαρμογής, τον οποίο θα αναλύσουμε στην συνέχεια.

5 Προσδιορισμός χρήσιμων ερωτημάτων για την εφαρμογή

Χρήσιμα ερωτήματα για την εφαρμογή είναι ερωτήματα που σχετίζονται με την λειτουργία μίας Social Media εφαρμογής όπως:

- Επιβεβαίωση στοιχείων εγγεγραμμένου χρήστη με email και password.
- Αναζήτηση χρηστών με το “username” τους.
- Ανάκτηση πληροφοριών για το προφίλ του χρήστη ή άλλων χρηστών.
- Εμφάνιση φίλων του χρήστη και πληροφορίες τους.
- Εμφάνιση συνομιλιών του χρήστη με άλλους χρήστες καθώς και τα μηνύματα της κάθε συνομιλίας.
- Εμφάνιση ειδοποιήσεων για likes, comments και friend requests.
- Εμφάνιση friend requests του χρήστη.
- Εμφάνιση comments και οι πληροφορίες τους σε ένα post.
- Εμφάνιση δημοσιεύσεων στο home screen φίλων του χρήστη καθώς και τις δημοσιεύσεις στο προφίλ του χρήστη μαζί με τις πληροφορίες τους.
- Εμφάνιση friend suggestions για τον χρήστη.

6 Δημιουργία δεδομένων με χρήση data generator ή ανεύρεση έτοιμων δεδομένων

6.1 Γενικά

Η δημιουργία των δεδομένων της εφαρμογής έγινε με την χρήση ενός python script. Συγκεκριμένα, το script δημιουργεί τυχαία δεδομένα για τα μοντέλα User(που έχει τις πληροφορίες των users), Post(που έχει τις πληροφορίες των posts) και Comment(που έχει τις πληροφορίες των comments σε διάφορα posts). Τώρα για τους χρήστες δημιουργούνται τυχαία δεδομένα για το ονοματεπώνυμο, όνομα χρήστη, κωδικό πρόσβασης στην εφαρμογή, περιγραφή στο προφίλ του χρήστη, φωτογραφία προφίλ, λίστα φίλων, λίστα χρηστών που έχει στείλει ο χρήστης αίτημα φιλίας(friend request) σε αυτούς, λίστα χρηστών που έχουν στείλει αίτημα φιλίας στον χρήστη και δημοσιεύσεις του χρήστη καθώς και πληροφορίες τους.

6.2 Ανάλυση python script `creationOfDatabase.py`

6.2.1 Τυχαία δεδομένα για τον χρήστη

Βλέπουμε από την εικόνα παρακάτω ότι η συνάρτηση `generateUser()` δημιουργεί ψεύτικο fullname, ψεύτικο password και ψεύτικο description μέσω της βιβλιοθήκης της python Faker. Για το username χρησιμοποιεί την συνάρτηση `generateUsername(fullname)` μέσω του fullname χρησιμοποιεί το πρώτο γράμμα του firstname + όλο το lastname ή εάν έχει ενιαίο fullname τότε το τοποθετεί όλο και στο τέλος βάζει και ένα τυχαίο νούμερο από 10 έως 99. Για το email χρησιμοποιεί την συνάρτηση `generateEmail(username)` μέσω του username τοποθετεί και στο τέλος το "@gmail.com". Τέλος, για την φωτογραφία προφίλ χρησιμοποιεί την συνάρτηση `fetchRandomImage()` και λαμβάνει κάθε φορά τυχαία φωτογραφία όταν καλείται μέσω ενός link [GetRandomImageEachTime](#) και χρειάζεται από την συνάρτηση να επιστρέψει μόνο μία φωτογραφία για το προφίλ.

```
def generateUser():
    fullname = fake.name()
    username = generateUsername(fullname)
    email = generateEmail(username)
    return {
        'fullname': fullname,
        'username': username,
        'email': email,
        'password': fake.password(), #remember, in real scenarios, hash this password
        'description': fake.text(),
        'profileImage': fetchRandomImage(), #fetch only one image for profile
        'receiving': [], #requests from probably friends taht received the user
        'sending': [], #pending requests that sends the user to others
        'friends': [], #to be updated with actual user references after insertion
        'posts': [] #assuming you'll handle posts similarly; adjust as needed
    }
```

Συνάρτηση `generateUsername(fullname)`:

```
def generateUsername(fullname):
    parts = fullname.split()
    if len(parts) > 1:
        username_base = parts[0][0].lower() + parts[-1].lower()
    else:
        username_base = parts[0].lower()
    return username_base + str(random.randint(10, 99))
```

Συνάρτηση `generateEmail(username)`:

```
def generateEmail(username):
    return f"{username}@gmail.com"
```

Συνάρτηση `fetchRandomImage()`:

```
def fetchRandomImage(num_images=1, profile=True):
    '''Fetches a specified number of random images and return them as a list of bytes objects.'''
    images = []
    for _ in range(num_images):
        image_url = "https://picsum.photos/200" #image source URL
        response = requests.get(image_url)
        if response.status_code == 200:
            images.append(response.content)
    return images if num_images >= 1 and not profile else images[0] #return the first image directly if only one is requested
```

Συνάρτηση `createUsersWithFriends(num_users)` δημιουργεί τους χρήστες και εισάγει φίλους και sending και receiving requests:

```
def createUsersWithFriends(num_users):
    users_ids = []
```

- **Βήμα 1ο:** Δημιουργεί και εισάγει χρήστες χωρίς φίλους ακόμη.

```
for _ in range(num_users):
    user_data = generateUser()
    result = users_collection.insert_one(user_data)
    users_ids.append(result.inserted_id)
```

- **Βήμα 2^ο:** Τυχαία αναθέτει φίλους βεβαιώνοντας ότι υπάρχει αμοιβάια φιλία οπότε τοποθετεί αυτόν τον χρήστη και στον πίνακα των άλλων χρηστών.

```
for user_id in users_ids:
    #select 2 to 5 random friends for this user
    friends_ids = random.sample([uid for uid in users_ids if uid != user_id], random.randint(2, 5))

    #update the current user's friends list
    for friend_id in friends_ids:
        users_collection.update_one({'_id': user_id}, {'$addToSet': {'friends': friend_id}})

    #ensure mutual friendships by adding this user to their friends' lists
    for friend_id in friends_ids:
        users_collection.update_one({'_id': friend_id}, {'$addToSet': {'friends': user_id}})
```

- **Βήμα 3^ο:** Αναθέτει αιτήματα φιλίας που έχει στείλει αυτός ο χρήστης σε άλλους(sending) και αιτήματα φιλίας που έχει λάβει από άλλους χρήστες(receiving).

```
for user_id in users_ids:
    current_user_doc = users_collection.find_one({'_id': user_id})
    current_friends = current_user_doc['friends']

    potential_requests = [uid for uid in users_ids if uid != user_id and uid not in current_friends]
    random.shuffle(potential_requests)

    half_point = len(potential_requests) // 2
    sending = potential_requests[:half_point]

    #update sending for the current user
    users_collection.update_one({'_id': user_id}, {'$set': {'sending': sending}})

    #update receiving for other users based on current user's sending
    for receiver in sending:
        users_collection.update_one({'_id': receiver}, {'$addToSet': {'receiving': user_id}})
```

```
for user in users_ids:
    #find overlaps between 'friends' and 'receiving' and 'sending'
    current_user_doc = users_collection.find_one({'_id': user_id})
    friends = current_user_doc['friends']
    receiving = current_user_doc['receiving']
    sending = current_user_doc['sending']

    overlapsReceiving = [uid for uid in receiving if uid in friends]
    overlapsSending = [uid for uid in sending if uid in friends]

    #remove overlaps from 'receiving'
    if overlapsReceiving:
        users_collection.update_one(
            {'_id': user},
            {'$pullAll': {'receiving': overlapsReceiving}}
        )

    #remove overlaps from 'sending'
    if overlapsReceiving:
        users_collection.update_one(
            {'_id': user},
            {'$pullAll': {'sending': overlapsSending}}
        )
```

- **Βήμα 4ο:** Δημιουργεί δημοσιεύσεις(posts) για κάθε χρήστη.

```
createPostsForUsers(users_ids)
```

6.2.2 Τυχαία δεδομένα για την δημοσίευση(post)

Βλέπουμε από την εικόνα παρακάτω ότι η συνάρτηση `createPostsForUsers-(user_ids)` δημιουργεί έναν συγκεκριμένο αριθμό posts(μεταβάλλεται από 3 έως 7) για κάθε χρήστη και ενημερώνει το user document με postIDs των πίνακα posts. Επίσης, όπως φαίνεται χρησιμοποιείται η συνάρτηση `generatePost(user_id)` όπου δημιουργεί ένα μοναδικό post data structure για τον χρήστη με πολλές εικόνες(από 1 έως 5 εικόνες κάθε post), με username και userId από το user collection και τα text, createdAt, updatedAt μέσω της βιβλιοθήκης της python Faker. Τέλος, η συνάρτηση `likePost-(user_id, post_id)` τοποθετεί χρήστες(φίλοι αυτού του χρήστη και να μην υπερβαίνει ο αριθμός των likes τον αριθμό των φίλων), που έχουν κάνει like, στον πίνακα likes για κάθε δημοσίευση(post).

```
def createPostsForUsers(user_ids):
    """Creates a specified number of posts for each user and updates the user document with post IDs."""
    for user_id in user_ids:
        post_ids = []
        for _ in range(random.randint(3, 7)):
            post_data = generatePost(user_id)
            result = posts_collection.insert_one(post_data)
            post_ids.append(result.inserted_id)
            #new block to add 1 to 5 likes from friends
            author_document = users_collection.find_one({'_id': user_id})
            author_friends = author_document['friends']
            num_likes = random.randint(1, min(10, len(author_friends))) # Ensure not more friends than available
            liked_friends = random.sample(author_friends, num_likes)
            for friend_id in liked_friends:
                likePost(friend_id, result.inserted_id)

        #update the user's document with the new post IDs
        users_collection.update_one({'_id': user_id}, {'$push': {'posts': {'$each': post_ids}}})
```

Συνάρτηση `generatePost(user_id):`

```
def generatePost(user_id):
    '''Generates a single post data structure for a given user, now with multiple images.'''
    num_images = random.randint(1, 5) #decide how many images to fetch
    images = fetchRandomImage(num_images, profile=False) # Fetch multiple images
    return {
        'text': fake.sentence(),
        'username': users_collection.find_one({'_id': user_id})['username'],
        'userId': user_id,
        'images': images, #attach the list of images
        'likes': [],
        'createdAt': fake.date_time_this_year(),
        'updatedAt': fake.date_time_this_year()
    }
```

Συνάρτηση likePost(user_id, post_id):

```
def likePost(user_id, post_id):
    #retrieve the post and the author's friends list
    post_document = posts_collection.find_one({'_id': post_id})
    userId = post_document['userId']
    author_document = users_collection.find_one({'_id': userId})
    author_friends = author_document['friends']

    #check if the liking user is a friend of the author
    if user_id in author_friends:
        #record the like
        if user_id not in post_document.get('likes', []):
            posts_collection.update_one(
                {'_id': post_id},
                {'$push': {'likes': user_id}}
            )
        else:
            print("User has already liked this post.")
    else:
        print("User is not a friend of the post's author and cannot like this post.")
```

6.2.3 Τυχαία δεδομένα για τα σχόλια σε δημοσιεύσεις(comments)

Βλέπουμε από την εικόνα παρακάτω ότι η συνάρτηση addCommentsToPosts() δημιουργεί έναν συγκεκριμένο αριθμό comments για κάθε post ενός χρήστη και το περιεχόμενο μεταβάλλεται σε τρεις επιλογές ['Great post!', 'Really loved this!', 'Awesome!']. Τέλος, μέσω της συνάρτησης addLikeToComment(user_id, comment_id) τοποθετείται τυχαίος αριθμός από likes σε σχόλια αλλά να βρίσκεται μέσα στους φίλους του χρήστη.

```
def addCommentsToPosts():
    posts = posts_collection.find()
    for post in posts:
        user_id = post['userId']
        user_document = users_collection.find_one({'_id': user_id})
        if user_document and 'friends' in user_document:
            num_comments = random.randint(1, min(3, len(user_document['friends'])))
            friends_to_comment = random.sample(user_document['friends'], num_comments)

            for friend_id in friends_to_comment:
                comment_content = random.choice(["Great post!", "Really loved this!", "Awesome!"])
                comment = {
                    'content': comment_content,
                    'user': friend_id,
                    'post': post['_id'],
                    'likes': [],
                    'createdAt': datetime.now()
                }
                comments_collection.insert_one(comment)
                if random.choice([True, False]):
                    addLikeToComment(friend_id, comments_collection.find_one({'post': post['_id'], 'user': friend_id, 'content': comment_content})['_id'])
```

Συνάρτηση addLikeToComment(user_id, comment_id):

```
def addLikeToComment(user_id, comment_id):
    comment = comments_collection.find_one({"_id": comment_id})
    if user_id not in comment['likes']:
        comments_collection.update_one({"_id": comment_id}, {"$addToSet": {"likes": user_id}})
    return
```

Τέλος αφού έχουν οριστεί όλες οι συναρτήσεις καλούμε την συνάρτηση `createUsersWithFriends(num_users)` με `num_users` να είναι ο αριθμός των χρηστών που θέλουμε να δημιουργήσουμε και έπειτα αφού έχουν δημιουργηθεί και χρήστες και δημοσιεύσεις για κάθε χρήστη καλούμε την συνάρτηση `addCommentsToPosts()` για να βάλουμε και σχόλια στις δημοσιεύσεις με όλες τις απαραίτητες πληροφορίες.

```
createUsersWithFriends(10)
addCommentsToPosts()
```

7 Σχεδιασμός του μοντέλου της ΒΔ και υλοποίησή του στο Σύστημα NoSQL ΒΔ

Η βάση δεδομένων ”SocialWaveDB” αποτελείται από 7 collections τα οποία ορίζονται μέσα από την Node.js και τα 3(User, Post, Comment) από αυτά αρχικοποιούνται μέσα από το python αρχείο που εξηγήθηκε προηγουμένως. Παρακάτω φαίνεται πως ορίζονται στην Node.js και ένα παράδειγμα από την MongoDB πως είναι πραγματικά τα δεδομένα.

▼ SocialWaveDB

- Chats
- Comments
- Messages
- Notifications
- OTP
- Posts
- Users**

Ορισμός "User" model στην Node.js: Παρατηρούμε ότι οι πίνακες friends, receiving και sending αναφέρονται στο μοντέλο User(έχουν εσωτερικά user ids) ώστε μετά με τα queries με το populate να πάρουμε αντίστοιχα και τα στοιχεία των φίλων, και αυτών που έστειλαν αίτημα φιλίας και σε αυτούς που έστειλε ο χρήστης. Επίσης, και ο πίνακας posts(έχει εσωτερικά post ids) αναφέρεται στο μοντέλο Post για τον ίδιο λόγο με προηγουμένως.

```
const UserSchema = new mongoose.Schema({
  fullname: {
    type: String
  },
  username: {
    type: String,
    required: true,
    unique: true,
    trim: true
  },
  email: {
    type: String,
    required: true,
    unique: true,
    trim: true
  },
  password: {
    type: String,
    required: true
  },
  description: {
    type: String
  },
  profileImage: {
    type: Buffer
  },
  receiving: [
    {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User'
    }
  ],
  sending: [
    {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User'
    }
  ],
  friends: [
    {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User'
    }
  ],
  posts: [
    {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Post'
    }
  ]
})

const User = mongoose.model('User', UserSchema, 'Users')
```

"Users" collection στην MongoDB:

```
_id: ObjectId('662ea9be6ecd53710f828e78')
fullname : "David Neal"
username : "dneal70"
email : "dneal70@gmail.com"
password : "F%8e1@$x%H"
description : "Personal show level her.
    Wish weight increase question. Many ground en..."
profileImage : Binary.createFromBase64('/9j/4QDeRXhpZgAASUkqAAgAAAAGABIBAwABAAAAAQ...
▶ receiving : Array (empty)
▶ sending : Array (1)
▶ friends : Array (6)
▶ posts : Array (5)
```

Ορισμός "Post" model στην Node.js:

```
const PostSchema = new mongoose.Schema({
  text: {
    type: String,
    trim: true
  },
  username: {
    type: String,
    required: true,
    trim: true
  },
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    required: true,
    ref: 'User'
  },
  images: [
    {
      type: Buffer,
      required: true
    }
  ],
  likes: [
    {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User'
    }
  ],
  {
    timestamps: true
  }
}

const Post = mongoose.model('Post', PostSchema, 'Posts')
```

"Posts" collection στην MongoDB:

```
_id: ObjectId('662ea9d36ecd53710f828e82')
text : "Focus fly generation maybe marriage."
username : "dneal70"
userId : ObjectId('662ea9be6ecd53710f828e78')
▶ images : Array (2)
▶ likes : Array (3)
createdAt : 2024-01-13T02:54:39.000+00:00
```

Ορισμός "Comment" model στην Node.js:

```
const commentSchema = new mongoose.Schema({
  content: {
    type: String,
    required: true
  },
  user: {
    type: mongoose.Schema.Types.ObjectId,
    required: true,
    ref: 'User'
  },
  post: {
    type: mongoose.Schema.Types.ObjectId,
    required: true,
    ref: 'Post'
  },
  likes: [
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
  ],
  createdAt: {
    type: Date,
    default: Date.now
  }
})

const Comment = mongoose.model('Comment', commentSchema, 'Comments')
```

"Comments" collection στην MongoDB:

```
_id: ObjectId('662eaa856ecd53710f828eb4')
content : "Awesome!"
user : ObjectId('662ea9c36ecd53710f828e7e')
post : ObjectId('662ea9d36ecd53710f828e82')
▶ likes : Array (empty)
createdAt : 2024-04-28T22:59:01.531+00:00
```

Ορισμός "OTP" model στην Node.js:

```
const OTPSchema = new mongoose.Schema({
  email: {
    type: String,
    required: true,
  },
  otp: {
    type: String,
    required: true,
  },
  createdAt: {
    type: Date,
    default: Date.now,
    expires: 60 * 10, // The document will be automatically deleted after 10 minutes of its creation time
  }
})
```

"OTP" collection στην MongoDB:

```
_id: ObjectId('6647283278ac56fd8f580c72')
email : "basilisanagnostopoulos02@gmail.com"
otp : "908325"
createdAt : 2024-05-17T09:49:38.491+00:00
__v : 0
```

Ορισμός "Notification" model στην Node.js:

```
const notificationSchema = new mongoose.Schema({
  usernameReceiver: {
    type: String,
    required: true
  },
  notifSenderId: {
    type: mongoose.Schema.Types.ObjectId,
    required: true,
    ref: 'User'
  },
  notificationType: {
    type: String
  },
  notifReferringPost: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Post'
  },
  notificationSeen: {
    type: Boolean
  },
  createdAt: {
    type: Date,
    default: Date.now
  }
})

const Notification = mongoose.model('Notification', notificationSchema, 'Notifications')
```

"Notifications" collection στην MongoDB:

```
_id: ObjectId('66328938d3b65eaf2ef325ac')
usernameReceiver : "mgordon88"
notifSenderId : ObjectId('662ea9bf6ecd53710f828e79')
notificationType : "like"
notifReferringPost : ObjectId('662eaa5a6ecd53710f828ea9')
notificationSeen : true
createdAt : 2024-05-01T18:26:00.862+00:00
```

Ορισμός "Chat" model στην Node.js:

```
const chatSchema = new mongoose.Schema({
  username: {
    type: String,
    required: true
  },
  chatMember: {
    type: mongoose.Schema.Types.ObjectId,
    required: true,
    ref: 'User'
  },
  lastMessageSeen: {
    type: Boolean
  },
  lastMessage: {
    type: String
  },
  lastMessageCreatedAt: {
    type: Date
  },
  messages: [
    {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Message'
    }
  ],
  createdAt: {
    type: Date,
    default: Date.now
  }
})
const Chat = mongoose.model('Chat', chatSchema, 'Chats')
```

"Chats" collection στην MongoDB:

```
_id: ObjectId('6643788b6a6b020591fcf67d')
username : "sramirez81"
chatMember : ObjectId('662ea9bf6ecd53710f828e79')
lastMessageSeen : true
lastMessage : "hbaixbaxkaka Xanthi
kaxiabxiabixbxxhhxzjhhzhzjzjjxnznzbzbzbzbzbzbzbbz..."
lastMessageCreatedAt : 2024-05-14T15:25:02.735+00:00
► messages : Array (13)
createdAt : 2024-05-14T14:43:23.620+00:00
```

Ορισμός "Message" model στην Node.js:

```
const messageSchema = new mongoose.Schema({
  sender: {
    type: mongoose.Schema.Types.ObjectId,
    required: true,
    ref: 'User'
  },
  text: {
    type: String
  },
  createdAt: {
    type: Date,
    default: Date.now
  }
})

const Message = mongoose.model('Message', messageSchema, 'Messages')
```

"Messages" collection στην MongoDB:

```
_id: ObjectId('66433b41c8443d01ce14632d')
sender : ObjectId('662ea9c16ecd53710f828e7b')
text : "sdsdsdsds"
createdAt : 2024-05-14T10:21:53.525+00:00
```

8 Δημιουργία κώδικα για λειτουργίες CRUD (Create, read, update and delete) δεδομένων

8.1 Λειτουργίες create δεδομένων στην βάση δεδομένων

Εγγραφή νέων χρηστών:

```
export async function addUser(data) {
    const newPassword = await bcrypt.hash(data.password, 10)
    const user = await User.create({
        fullname: null,
        username: data.username,
        email: data.email,
        password: newPassword,
        following: [],
        followers: [],
        description: null,
        profileImage: null
    })
    return user
}
```

Δημιουργία νέων δημοσιεύσεων:

```
const post = await Post.create({
    text: data.caption,
    username: username,
    userId: data.userId,
    images: images
})
```

Δημιουργία ειδοποιήσεων για friend request:

```
await Notification.create(
  {
    usernameReceiver: receiver.username,
    notifSenderId: senderId,
    notificationSeen: false,
    notificationType: 'request',
    notifReferringPost: null
  }
)
```

Δημιουργία ειδοποιήσεων για like σε μία δημοσίευση:

```
await Notification.create(
  {
    usernameReceiver: updatedPost.username,
    notifSenderId: userId,
    notificationSeen: false,
    notificationType: 'like',
    notifReferringPost: updatedPost._id
  }
)
```

Δημιουργία ειδοποιήσεων για comment σε μία δημοσίευση:

```
await Notification.create(
  {
    usernameReceiver: post.username,
    notifSenderId: userId,
    notificationSeen: false,
    notificationType: 'comment',
    notifReferringPost: postId
  }
)
```

Δημιουργία OTP(One-Time Password):

```
export async function createOTP(otpPayload) {
  const otp = await OTP.create(otpPayload)
  return otp
}
```

Δημιουργία σχολίου(comment) για μία δημοσίευση:

```
await Comment.create(
  {
    content: content,
    user: userId,
    post: postId,
    likes: []
  }
)
```

Δημιουργία συνομιλίας(chat) μεταξύ δύο φώτων:

```
const newMessage = await Message.create({
  sender: currentUserId,
  text: text
})
```

Δημιουργία μηνυμάτων σε μία συνομιλία(chat) μεταξύ δύο φώτων:

```
const newMessage = await Message.create({
  sender: currentUserId,
  text: text
})
```

8.2 Λειτουργίες read δεδομένων στην βάση δεδομένων

Αναζήτηση χρήστη με το email:

```
export async function findUser(email) {
    let user = await User.findOne({
        email: email
    })

    return user
}
```

Έλεγχος χρήστη αν υπάρχει με το email ή το username:

```
export async function checkIfEmailOrUsernameExists(username, email) {
    let user = await User.findOne({
        email: email
    })

    if(user!==null) {
        return null
    }
    else {
        user = await User.findOne({
            username: username
        })
        if(user!==null) {
            return null
        }
    }
    return true
}
```

Έλεγχος χρήστη για το login:

```
export async function checkLoginUser(data) {
  let user = await User.findOne({
    email: data.email
  })
  if(user==null) {
    return null
  }

  const match = await bcrypt.compare(data.password,user.password)
  // const match = await User.findOne({
  //   password: data.password
  // })
  if(match) {
    return user
  }
  else {
    return null
  }
}
```

Ανάκτηση πληροφοριών του χρήστη για τα requests και τους φίλους:

```
export async function getSRF(username) {

  const user = await User.findOne({username: username})
  return {sending: user.sending, receiving: user.receiving, friends: user.friends}
}
```

Εύρεση του πιο πρόσφατου OTP για ένα συγκεκριμένο email:

```
export async function findOTP(otp, email, option) {
  let otpResponse

  if(option === 0) {
    otpResponse = await OTP.findOne({ otp: otp })
  }
  else {
    // Find the most recent OTP for the email
    otpResponse = await OTP.find({ email: email }).sort({ createdAt: -1 }).limit(1)
  }
  return otpResponse
}
```

Ανάκτηση τον αριθμό των ειδοποιήσεων που δεν έχει δει ακόμη ο χρήστης:

```
export async function getNumberOfUnseenNotif(username) {
    const unseenNotifications = await Notification.find({usernameReceiver: username, notificationSeen: false})
    return unseenNotifications.length
}
```

Ανάκτηση τον αριθμό των συνομιλιών που δεν έχει δει ακόμη ο χρήστης:

```
export async function getNumberOfUnseenChats(username) {
    const unseenChats = await Chat.find({username: username, lastMessageSeen: false})
    return unseenChats.length
}
```

8.3 Λειτουργίες update δεδομένων στην βάση δεδομένων

Ενημέρωση κωδικού πρόσβασης:

```
export async function updatePassword(email,password) {
    const newPassword = await bcrypt.hash(password, 10)
    await User.findOneAndUpdate({email: email}, {password: newPassword})
}
```

Ενημέρωση τελευταίου μηνύματος σε μία συνομιλία(chat):

```
await Chat.findOneAndUpdate(
    { username: username, chatMember: chatMemberId },
    {
        $set: { lastMessageSeen: true, lastMessage: text, lastMessageCreatedAt: newMessage.createdAt },
        $push: { messages: newMessage._id }
    },
    { new: true, upsert: true } //upsert option to create the chat if it doesn't exist
)

await Chat.findOneAndUpdate(
    { username: chatMemberUsername, chatMember: currentUserId },
    {
        $set: { lastMessageSeen: false, lastMessage: text, lastMessageCreatedAt: newMessage.createdAt },
        $push: { messages: newMessage._id }
    },
    { new: true, upsert: true } //upsert option to create the chat if it doesn't exist
)
```

Ενημέρωση ότι ο χρήστης είδε το τελευταίο μήνυμα σε μία συνομιλία(chat):

```
export async function updateLastMessageSeen(chatId) {
    await Chat.findByIdAndUpdate(chatId, { lastMessageSeen: true })
}
```

Ενημέρωση εισαγωγής like σε ένα σχόλιο(comment):

```
export async function addLikeToCommentForPost(postId, userId, content, currentUser) {
    const userComments = await Comment.findOneAndUpdate({
        post: postId,
        user: userId,
        content: content
    },
    { $addToSet: { likes: currentUser } },
    { new: true },
    { $project: { _id: 0, content: 0, user: 0, post: 0, createdAt: 0, likes: 1 } })

    return userComments.likes
}
```

Ενημέρωση διαγραφής like από ένα σχόλιο(comment):

```
export async function deleteLikeFromCommentForPost(postId, userId, content, currentUser) {
    const userComments = await Comment.findOneAndUpdate({
        post: postId,
        user: userId,
        content: content
    },
    { $pull: { likes: currentUser } },
    { new: true },
    { $project: { _id: 0, content: 0, user: 0, post: 0, createdAt: 0, likes: 1 } })

    return userComments.likes
}
```

Ενημέρωση ειδοποιήσεων ότι τις είδε ο χρήστης:

```
await Notification.updateMany({
    usernameReceiver: username, notificationSeen: false
},
{
    notificationSeen: true
})
```

Ενημέρωση περιγραφής δημοσίευσης(post):

```
const updatedPost = await Post.findOneAndUpdate(
  { _id: postId },
  { text: newCaption },
  {
    new: true,
    projection: '_id text username userId images likes createdAt',
    lean: true
  }
)
```

Ενημέρωση εισαγωγής like σε μία δημοσίευση(post):

```
export async function addLikeToPost(userId, postId) {
  const updatedPost = await Post.findOneAndUpdate(
    { _id: postId },
    { $addToSet: { likes: userId } },
    { new: true },
    { $project: { _id: 0, text: 0, username: 0, userId: 0, images: 0, likes: 1 } }
  ).lean()

  if(updatedPost.userId.toString()!==userId) {
    //creates the notification
    await Notification.create(
      {
        usernameReceiver: updatedPost.username,
        notifSenderId: userId,
        notificationSeen: false,
        notificationType: 'like',
        notifReferringPost: updatedPost._id
      }
    )
  }
  return updatedPost.likes
}
```

Ενημέρωση διαγραφής like σε μία δημοσίευση(post):

```
export async function deleteLikeFromPost(userId, postId) {
  const updatedPost = await Post.findOneAndUpdate(
    { _id: postId },
    { $pull: { likes: userId } },
    { new: true },
    { $project: { _id: 0, text: 0, username: 0, userId: 0, images: 0, likes: 1 } }
  ).lean()
  return updatedPost.likes
}
```

Ενημέρωση λεπτομερειών του προφίλ του χρήστη:

```
export async function updateUser(data, profileImage, username) {
  const user = await User.findOneAndUpdate({username: username}, {
    description: data.description != "" ? data.description : null,
    fullname: data.fullname != "" ? data.fullname : null,
    profileImage: profileImage != 0 ? profileImage : null
  },
  {
    new: true,
    projection: '_id fullname username description profileImage receiving sending friends posts',
    lean: true
  })
  user.id = user._id.toString()
  delete user._id
  return user
}
```

Ενημέρωση λόγω αποδοχής του αιτήματος φιλίας:

```
export async function acceptRequestOfTheCurrentUser(userId, requesterId) {
  //updates the user who is accepting the request
  const updatedUser = await User.findByIdAndUpdate(userId, {
    $pull: { receiving: requesterId, sending: requesterId },
    $addToSet: { friends: requesterId }
  }, { new: true })

  //updates the user who sent the request
  await User.findByIdAndUpdate(requesterId, {
    $pull: { sending: userId, receiving: userId },
    $addToSet: { friends: userId }
  })
  return { receiving: updatedUser.receiving, friends: updatedUser.friends }
}
```

Ενημέρωση λόγω απόρριψης του αιτήματος φιλίας:

```
export async function rejectRequestOfTheCurrentUser(userId, requesterId) {
  //updates the user who is accepting the request
  const updatedUser = await User.findByIdAndUpdate(userId, {
    $pull: { receiving: requesterId }
  }, { new: true })

  //updates the user who sent the request
  await User.findByIdAndUpdate(requesterId, {
    $pull: { sending: userId }
  })
  return { receiving: updatedUser.receiving, friends: updatedUser.friends }
}
```

Ενημέρωση λόγω αποστολής ενός αιτήματος φιλίας για έναν χρήστη:

```
//updates the user who is sending the request
const updatedUser = await User.findByIdAndUpdate(senderId, {
  $addToSet: { sending: receiverId }
}, { new: true })

//updates the user who received the request
const receiver = await User.findByIdAndUpdate(receiverId, {
  $addToSet: { receiving: senderId }
}, {new: true})
```

Ενημέρωση λόγω ακύρωσης αποστολής ενός αιτήματος φιλίας για έναν χρήστη:

```
export async function unsendRequestInAUser(senderId, receiverId) {
  //updates the user who is unsends the request
  const updatedUser = await User.findByIdAndUpdate(senderId, {
    $pull: { sending: receiverId }
  }, { new: true })

  //updates the user who unreceived the request
  await User.findByIdAndUpdate(receiverId, {
    $pull: { receiving: senderId }
  })
  return updatedUser.sending
}
```

Ενημέρωση λόγω διαγραφής ενός φίλου του χρήστη:

```
export async function deleteFriendship(currentUserId, friendUserId) {
  const updatedUser = await User.findByIdAndUpdate(currentUserId, {
    $pull: { friends: friendUserId }
  }, { new: true })

  await User.findByIdAndUpdate(friendUserId, {
    $pull: { friends: currentUserId }
  })

  return updatedUser.friends
}
```

Ενημέρωση με την εισαγωγή νέων δημοσιεύσεων τοποθετώντας τες στην αρχή σχετικά με τις προηγούμενες για να φαίνονται πρώτες:

```
const userPostsDetails = await User.findOneAndUpdate({username: username},
  {
    $push: {
      posts: {
        $each: [post._id],
        $position: 0
      }
    },
    { new: true }
  }
  .populate({
    path: 'posts',
    select: '_id text username userId images likes createdAt'
}).lean()
```

8.4 Λειτουργίες delete δεδομένων στην βάση δεδομένων

Διαγραφή ενός χρήστη:

```
export async function deleteUserByUsername(username) {
  await User.deleteOne({username: username})
}
```

Διαγραφή μίας δημοσίευσης(post) και των ειδοποιήσεων που σχετίζονται με αυτή:

```
await Notification.deleteMany({notifReferringPost: postId})
await Post.deleteOne({_id: postId})
```

Διαγραφή των OTP με το αντίστοιχο email:

```
export async function deleteOTP(email) {
  await OTP.deleteOne({email: email})
}
```

Διαγραφή μίας συνομιλίας μεταξύ δύο χρηστών(chat):

```
await Chat.findByIdAndDelete(chatId)
```

Διαγραφή ενός σχολίου(comment) σε μία δημοσίευση(post):

```
await Comment.deleteOne({
    content: content,
    user: userId,
    post: postId
})
```

Διαγραφή όλων των ειδοποιήσεων που έχουν σταλθεί για έναν χρήστη:

```
export async function deleteNotifsOfTheCurrentUser(username) {
    await Notification.deleteMany({usernameReceiver: username})
}
```

9 Υλοποίηση των ερωτημάτων στη ΒΔ

Αναζήτηση χρηστών χρησιμοποιώντας regular expressions:

```
export async function searchUsersUsingRegex(regexPattern) {
    const users = await User.find({ username: { $regex: regexPattern } },
        '_id fullname username description profileImage receiving sending friends posts'
    ).lean()
    for(let i=0; i < users.length; i++) {
        users[i].id = users[i]._id.toString()
        delete users[i]._id
    }
    return users
}
```

Ανάκτηση φίλων του χρήστη:

```
export async function findFriendsOfTheCurrentUser(username) {
  const friends = await User.findOne({username: username})
    .populate({
      path: 'friends',
      select: '_id fullname username profileImage description posts receiving sending friends'
    }).lean()
  for(let i=0; i < friends.friends.length; i++) {
    friends.friends[i].id = friends.friends[i]._id.toString()
    delete friends.friends[i]._id
  }
  return friends.friends
}
```

Ανάκτηση αιτημάτων φιλίας(friend requests) του χρήστη:

```
export async function findRequestsOfTheCurrentUser(username) {
  const requests = await User.findOne({username: username})
    .populate({
      path: 'receiving',
      select: '_id fullname username profileImage description posts receiving sending friends'
    }).lean()
  for(let i=0; i < requests.receiving.length; i++) {
    requests.receiving[i].id = requests.receiving[i]._id.toString()
    delete requests.receiving[i]._id
  }
  return requests.receiving
}
```

Ανάκτηση δημοσιεύσεων(posts) των φίλων του χρήστη για το home screen:

```
const friendsPosts = await Post.aggregate([
  { $match: { userId: { $in: currentUser.friends } } },
  { $sort: { createdAt: -1 } },
  {
    $lookup: {
      from: "Users",
      localField: "userId",
      foreignField: "_id",
      as: "userDetails"
    }
  },
  { $unwind: "$userDetails" },
  {
    $project: {
      id: "$_id",
      text: 1,
      userId: 1,
      username: 1,
      profileImage: "$userDetails.profileImage",
      images: 1,
      likes: 1,
      createdAt: 1,
      _id: 0
    }
  }
])
```

Ανάκτηση προτεινόμενων φίλων για τον χρήστη:

```
const suggestedFriends = await User.aggregate([
  {
    $match: {
      _id: { $in: currentUser.friends } // First level friends
    }
  },
  {
    $lookup: {
      from: 'Users',
      localField: 'friends',
      foreignField: '_id',
      as: 'friendsOfFriends'
    }
  },
  {
    $unwind: '$friendsOfFriends'
  },
  {
    $replaceRoot: { newRoot: "$friendsOfFriends" }
  },
  {
    $match: {
      _id: { $nin: currentUser.friends.concat([currentUser._id]) } // Exclude direct friends and self
    }
  },
  {
    $group: {
      _id: '$_id',
      fullname: { $first: '$fullname' },
      username: { $first: '$username' },
      profileImage: { $first: '$profileImage' },
      description: { $first: '$description' },
      posts: { $first: '$posts' },
      mutualConnections: { $sum: 1 } // Count occurrences to rank by common connections
    }
  },
  {
    $sort: { mutualConnections: -1 } // Sort by the number of mutual connections
  },
  {
    $limit: 6 // Limit the result to the top 6 suggested friends
  },
  {
    $project: { // Select only the necessary fields for the output
      id: '$_id',
      _id: 0,
      username: 1,
      fullname: 1,
      profileImage: 1
    }
  }
])
```

Ανάκτηση λεπτομερειών για το προφίλ του χρήστη:

```
const posts = await User.findOne({username: username})
  .populate({
    path: 'posts',
    select: '_id text username userId images likes createdAt',
    options: { sort: { 'createdAt': -1 } }
  }).lean()
```

Ανάκτηση ειδοποιήσεων για τον χρήστη:

```
const notifications = await Notification.aggregate([
  { $match: { usernameReceiver: username } },
  { $sort: { createdAt: -1 } },
  {
    $lookup: {
      from: "Users",
      localField: "notifSenderId",
      foreignField: "_id",
      as: "senderInfo"
    }
  },
  // Unwind the result since lookup returns an array
  { $unwind: "$senderInfo" },
  {
    $lookup: {
      from: "Posts",
      localField: "notifReferringPost",
      foreignField: "_id",
      as: "postDetails"
    }
  },
  // Unwind the result, including preserve empty results with preserveNullAndEmptyArrays
  { $unwind: { path: "$postDetails", preserveNullAndEmptyArrays: true } },
  {
    $project: {
      "id": "$_id",
      "notificationType": 1,
      "notificationSeen": 1,
      "createdAt": 1,
      "senderProfile": "$senderInfo.profileImage",
      "senderUsername": "$senderInfo.username",
      "postImage": { $arrayElemAt: ["$postDetails.images", 0] }
    }
  }
])
```

Ανάκτηση συνομιλιών(chats) του χρήστη:

```
const chats = await Chat.find({ username: username })
  .populate('chatMember', '_id username fullname profileImage')
  .sort({ lastMessageCreatedAt: -1 })
  .lean()
```

Ανάκτηση σχολίων(comments) για μία δημοσίευση(post):

```
const userComments = await Comment.aggregate([
  { $match: { post: new mongoose.Types.ObjectId(postId) } },
  { $sort: { createdAt: -1 } },
  {
    $lookup: {
      from: "Users",
      localField: "user",
      foreignField: "_id",
      as: "userDetails"
    }
  },
  { $unwind: "$userDetails" },
  {
    $project: {
      id: "$_id",
      content: 1,
      user: 1,
      post: 1,
      _id: 0,
      likes: 1,
      createdAt: 1,
      username: "$userDetails.username",
      profileImage: "$userDetails.profileImage"
    }
  }
])
```

Ανάκτηση μηνυμάτων μίας συνομιλίας του χρήστη:

```
const chat = await Chat.findById(chatId)
  .populate({
    path: 'messages',
    select: '_id sender text createdAt'
  }).lean()
```

10 Λίστα με τα αρχεία που παραδίδονται και τι περιλαμβάνει το καθένα

1. `creationOfDatabase.py` python αρχείο που περιέχει τον κώδικα για την δημιουργία της βάσης με φόρτωση δεδομένων όπως εξηγήθηκε και προηγουμένως σε προηγούμενη ενότητα.
2. `backend.zip` zip αρχείο που περιέχει όλο τον κώδικα για το backend(πάνω από 30 αρχεία).

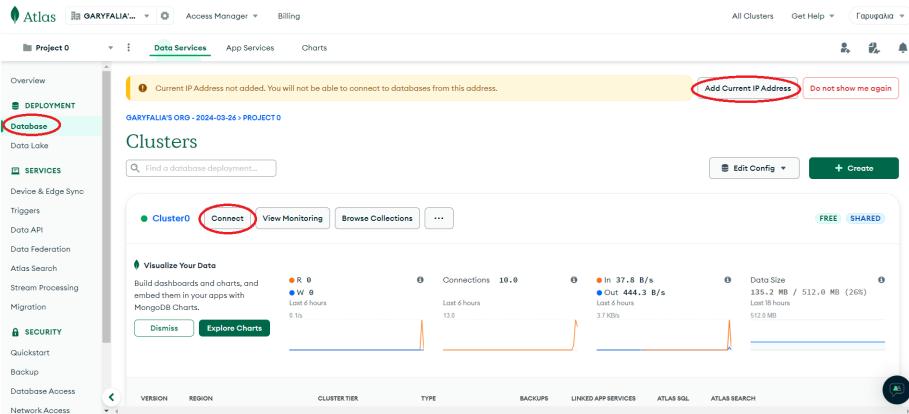
3. frontend.zip zip αρχείο που περιέχει όλο τον κώδικα για το frontend(πάνω από 130 αρχεία) μέσα στον φάκελο SocialWave.

11 Οδηγίες εγκατάστασης της εφαρμογής και φόρτωση δεδομένων

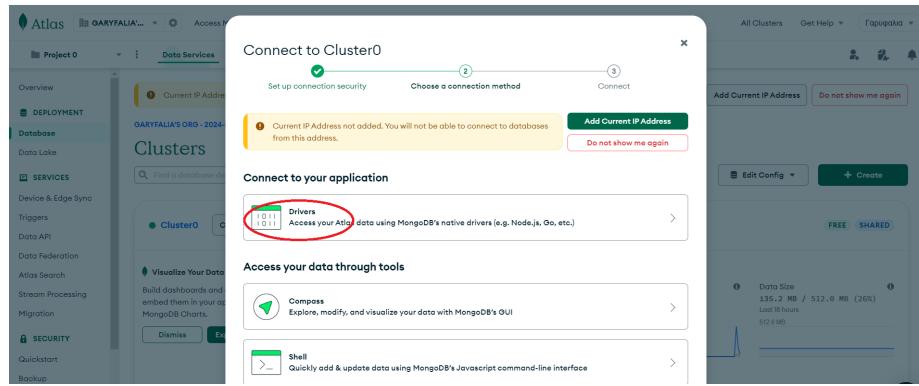
Η εγκατάσταση και χρήση της εφαρμογής μας προϋποθέτει την χρήση MacBook καθώς η ανάπτυξη εφαρμογών για iOS γίνεται με το Xcode, που είναι το επίσημο ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) της Apple, και το λειτουργικό σύστημα macOS, που είναι απαραίτητο για τη χρήση του Xcode, δεν υποστηρίζεται επίσημα από υπολογιστές διαφορετικούς της Apple. Επίσης, χρειάζεται η εφαρμογή Tailscale για την σύνδεση του υπολογιστή σε MacBook (σε περίπτωση που τρέξετε τον server σε διαφορετικό υπολογιστή από το MacBook).

Το Xcode μπορείτε να το κατεβάσετε [εδώ](#) και το Tailscale [εδώ](#). Στην συνέχεια, για την εγκατάσταση της εφαρμογής σε MacBook, ακολουθούμε τα παρακάτω βήματα:

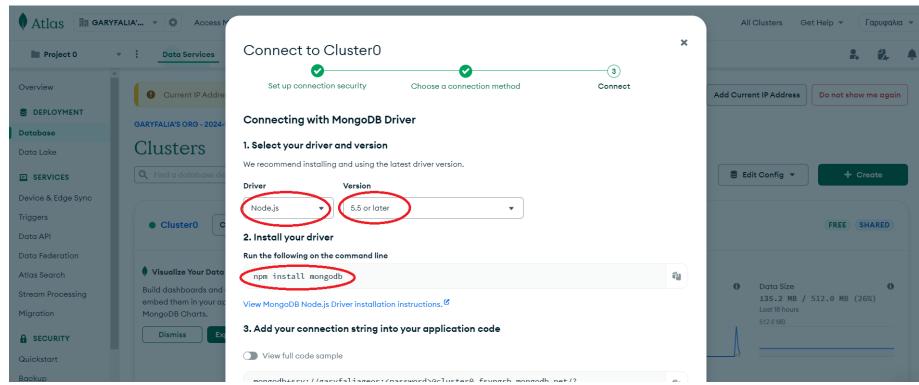
1. Κατεβάζουμε όλα τα αρχεία της εφαρμογής.
2. Αφού έχουμε εγγραφεί στην MongoDB Atlas, πατάμε ”Database” και ”Connect”. Εάν εμφανίζεται μήνυμα για την current IP πατάμε ”Add Current IP Address”.



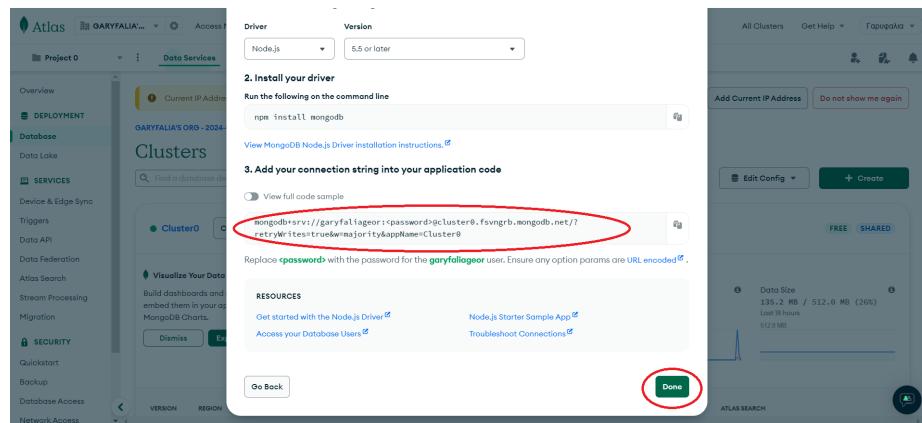
3. Κάνουμε κλικ στο πεδίο "Drivers".



4. Επιλέγουμε ως Driver την Node.js και την version που έχουμε και τρέχουμε στο terminal την εντολή που φαίνεται.



5. Αντιγράφουμε το connection string και το βάζουμε στο αρχείο `creationOfDatabase.py`, στο πεδίο "connectionString" και στο αρχείο `DatabaseConfig.js`.



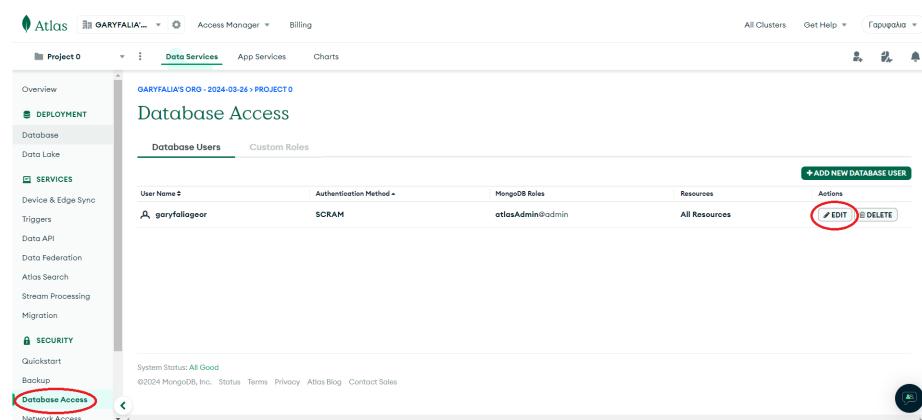
```
#connect to MongoDB
client = MongoClient("connectionString")
```

Σχήμα 3. Αρχείο `creationOfDatabase.py`

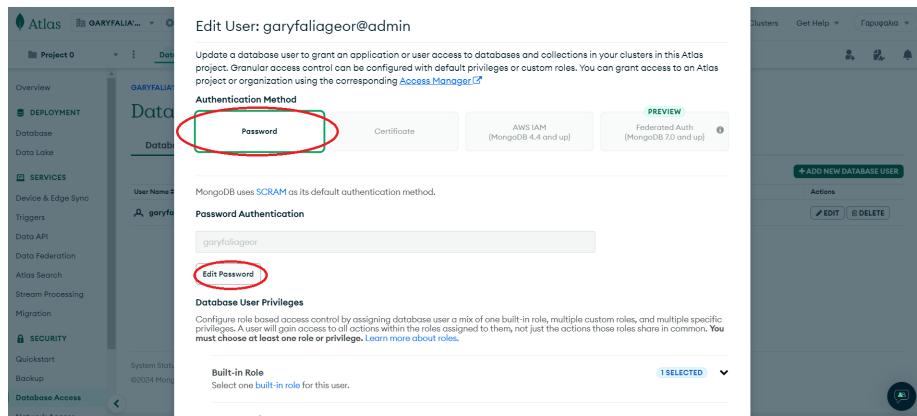
```
try {
  await mongoose.connect(`connectionString`)
  .then(console.log("MongoDB Connected"))
} catch(error) {
```

Σχήμα 4. Αρχείο `DatabaseConfig.js`

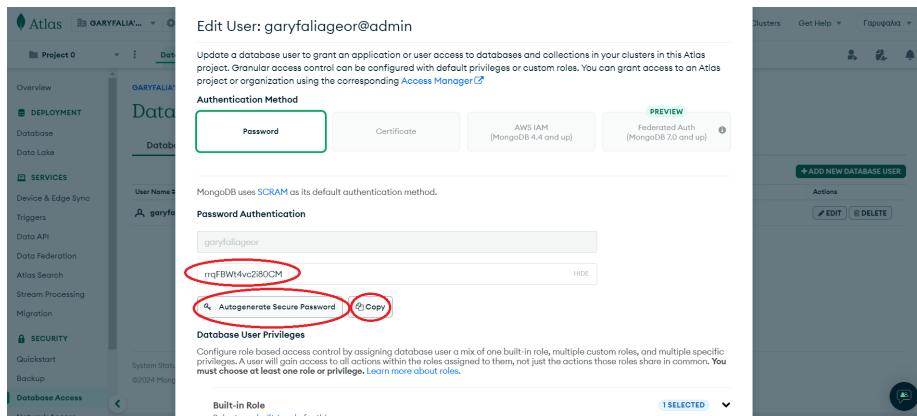
6. Παρατηρούμε ότι στο "connectionString" λείπει το "password". Για να δημιουργήσουμε ένα κρυπτογραφημένο password πατάμε "Database Access" και "Edit".



7. Πατάμε "Edit Password".



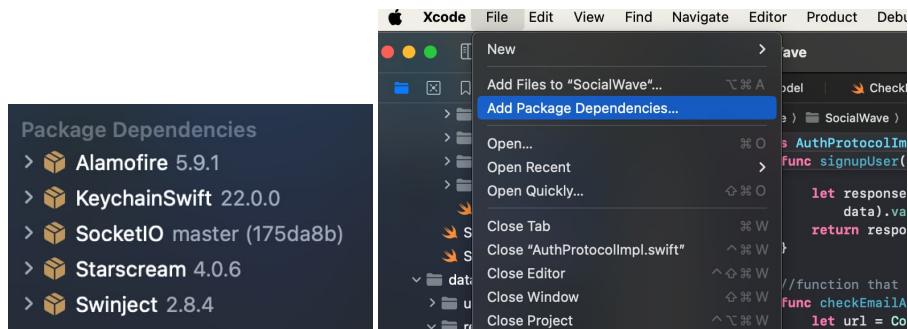
8. Πατάμε "Autogenerate Secure Password", κάνουμε "Copy" και στο κάτω μέρος του παραθύρου πατάμε "Update User". Τοποθετούμε το password που αντιγράψαμε στο πεδίο "password" του "connectionString".

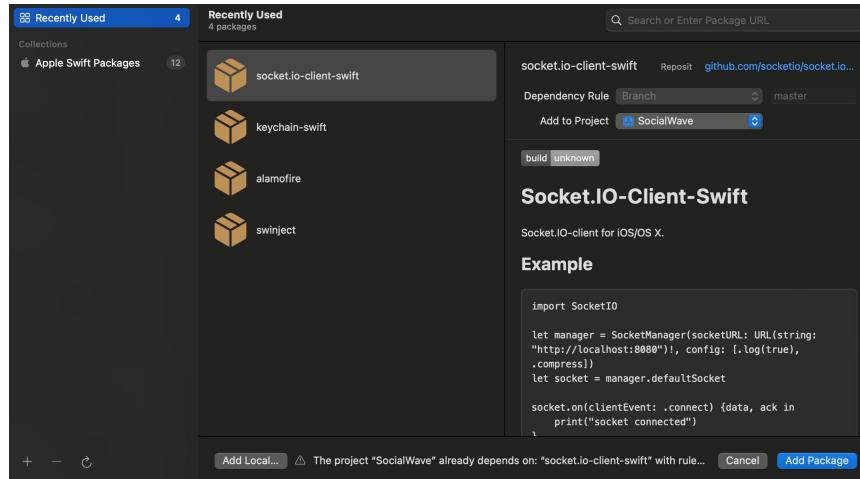


9. Ανοίγουμε ένα terminal και στο path όπου βρίσκεται το αρχείο `creationOfDatabase.py` τρέχουμε την εντολή `python creationOfDatabase.py` για να φορτωθούν τα δεδομένα και να ολοκληρωθεί η βάση δεδομένων.
10. Ανοίγουμε τον φάκελο "backend" στο Visual Studio Code, όπου τρέχουμε την εντολή `npm start` στο terminal (o server μπορεί να δημιουργηθεί και σε Windows/Linux υπολογιστή).
11. Συνεχίζουμε ανοίγοντας το Xcode που έχουμε εγκαταστήσει στο MacBook μας. Κάνουμε κλικ στο "Create New Project" και κάνουμε "Import" τον φάκελο (τα αρχεία) με το project.

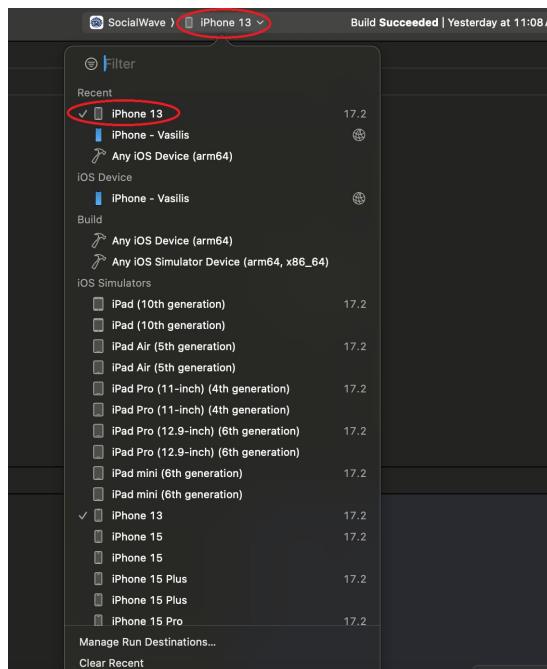


12. Για να προσθέσουμε τα "Package Dependencies" που φαίνονται παρακάτω, κάνουμε κλικ στο "File" -> "Add Package Dependencies...". Στην συνέχεια κάνουμε ένα search σε μία μηχανή αναζήτησης για τα παρακάτω dependencies και θα βρούμε κάποια github links. Τα τοποθετούμε στο "search" που φαίνεται παρακάτω και πατάμε "Add Package".

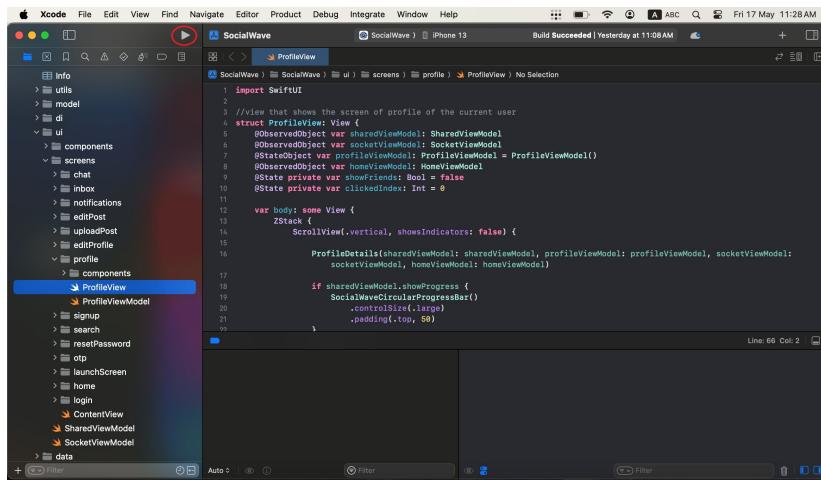




13. Επιλέγουμε ως simulator το "Iphone 13".



14. Κάνουμε κλικ στο κουμπί "Run" πάνω αριστερά και περιμένουμε να ανοίξει ο simulator με την εφαρμογή μας (αυτό ίσως διαρκέσει μερικά λεπτά).



15. Τέλος, για να τρέξουμε την εφαρμογή θα πρέπει αφού έχουμε εγκαταστήσει την εφαρμογή "Tailscale" και στους δύο υπολογιστές, να τρέξουμε την εντολή tailscale up.

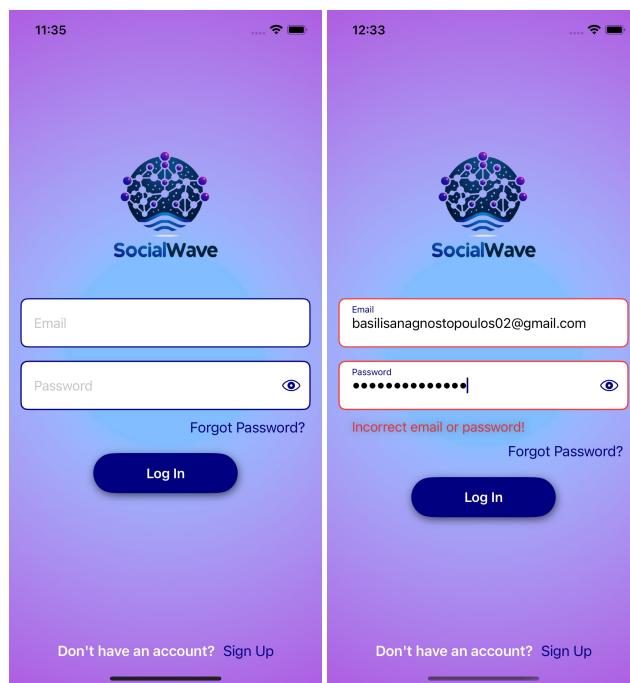
12 Οδηγίες χρήσης και παράδειγμα εκτέλεσης της εφαρμογής με παρουσίαση της διεπαφής χρήστη για λειτουργίες CRUD και για την εκτέλεση των ερωτημάτων στη ΒΔ, εκτέλεση των ερωτημάτων και εξέταση της ορθότητας των αποτελεσμάτων

Η εφαρμογή "SocialWave" αποτελείται από αρκετές σελίδες και εκτελεί πολλές λειτουργίες. Ακολουθεί μία ανάλυση της εφαρμογής:

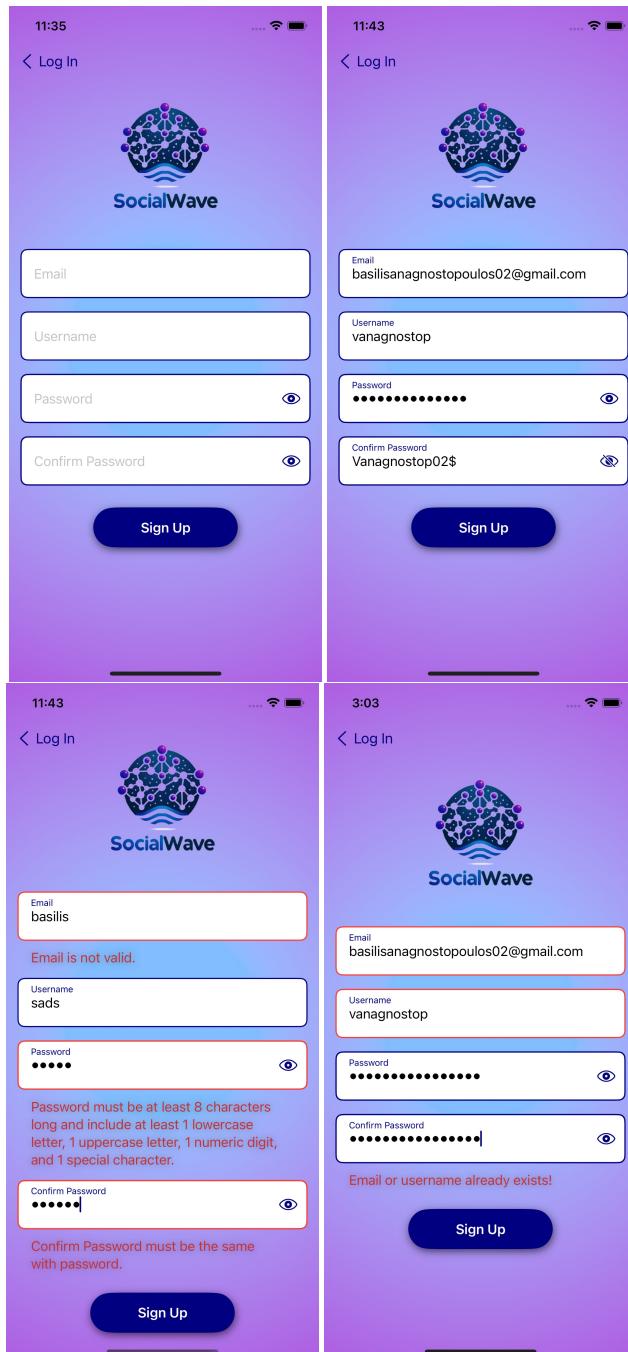
- **Splash Screen:** Σελίδα που εμφανίζεται κατά το άνοιγμα της εφαρμογής.



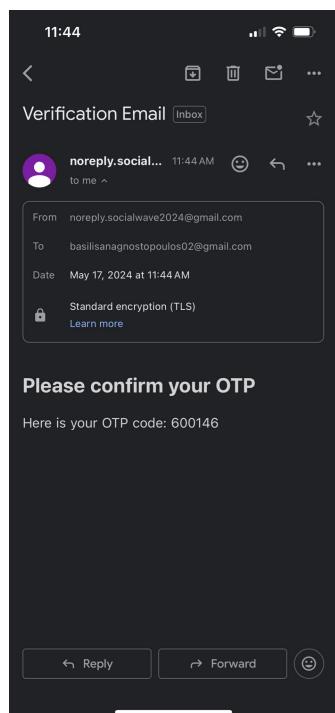
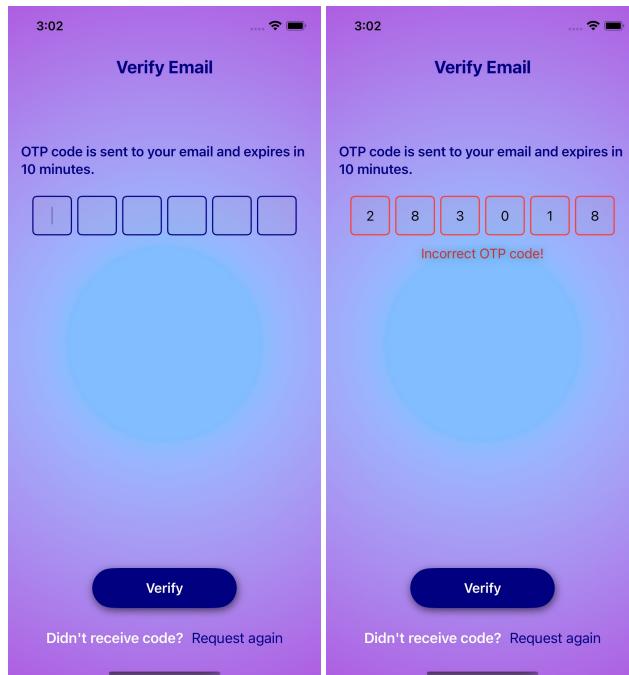
- **Log-In Screen:** Σελίδα που περιέχει το Log-In Form της εφαρμογής για χρήστες που έχουν ήδη account. Κάνοντας κλικ στο "Log In" ο χρήστης συνδέεται στο λογαριασμό του και μεταφέρεται στο "Home Screen". Σε περίπτωση εισαγωγής λανθασμένων στοιχείων εμφανίζεται σχετικό μήνυμα. Κάνοντας κλικ στο "ματακί" μπορεί να εμφανίσει ή να κρύψει τον κωδικό πρόσβασής του. Τέλος, με κλικ στο κουμπί "Sign Up" ο χρήστης μεταφέρεται στην σελίδα "Sign Up Screen", ενώ πατώντας το "Forgot Password?" εμφανίζεται η σελίδα "Reset Password".



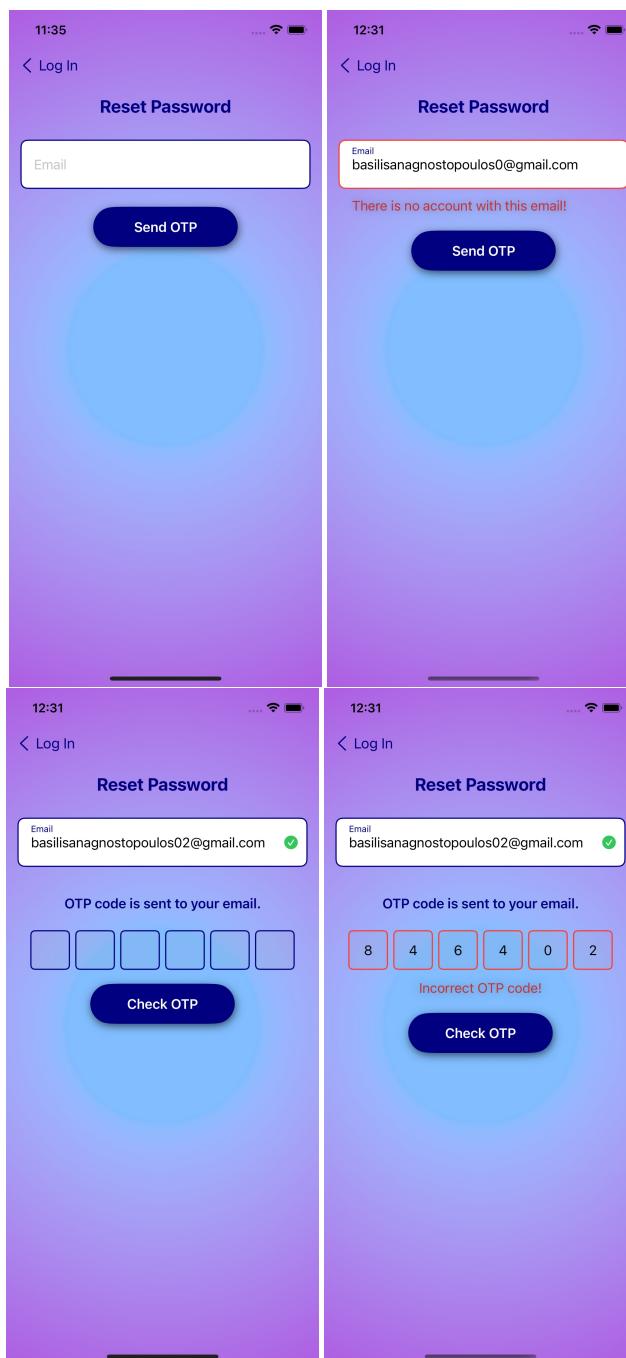
- **Sign-Up Screen:** Σελίδα που περιέχει το Sign-Up Form της εφαρμογής για χρήστες που συνδέονται πρώτη φορά και θέλουν να δημιουργήσουν account. Κάνοντας κλικ στο "Log In", ο χρήστης μεταφέρεται στο "Log In Screen", ενώ πατώντας το "Sign Up", ο χρήστης μεταφέρεται στη σελίδα "Verify Email with OTP", εφόσον τα στοιχεία που πληκτρολόγησε είναι σωστά. Σε αντίθετη περίπτωση, θα εμφανιστούν τα αντίστοιχα μηνύματα λάθους όπως φαίνεται παρακάτω.

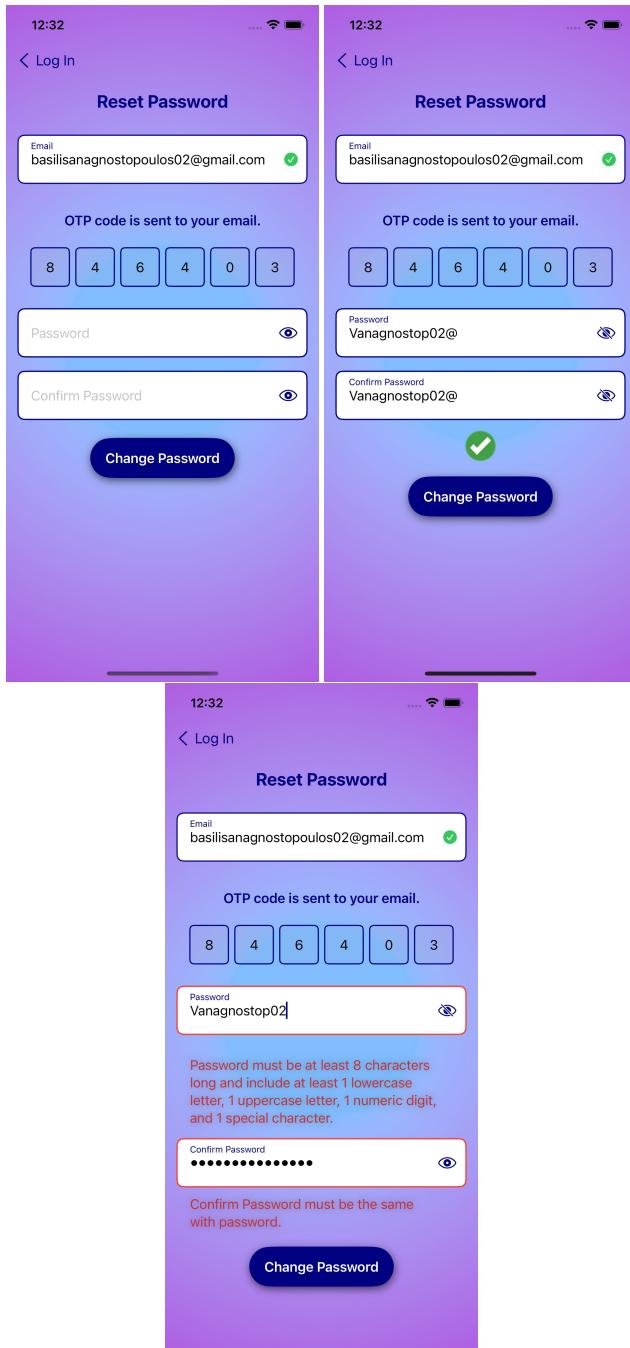


- **Verify Email with OTP (One-Time-Password):** Σελίδα για την επιβεβαίωση του email του χρήστη κατά την εγγραφή του, με χρήση του 6ψήφιου κωδικού που στέλνεται στο email που πληκτρολόγησε στα στοιχεία του. Κάνοντας κλικ στο "Request again", στέλνεται νέος κωδικός στο ίδιο email. Πατώντας "Verify", εάν ο κωδικός είναι σωστός, ο χρήστης μπορεί να συνεχίσει κάνοντας Log-In στον λογαριασμό του.

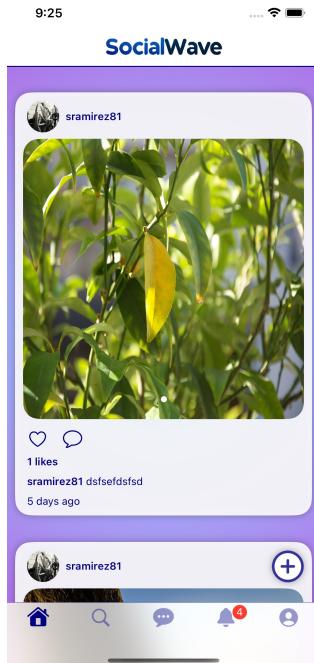


- **Reset Password Screen:** Σελίδα για την αλλαγή του κωδικού πρόσβασης του χρήστη. Ο χρήστης αρχικά θα πρέπει να συμπληρώσει το email με το οποίο έχει κάνει εγγραφή για να του σταλεί ένας 6ψήφιος OTP κωδικός και στην συνέχεια να τον συμπληρώσει στο αντίστοιχο πεδίο. Αφού ολοκληρωθεί η ταυτοποίηση του, ο χρήστης εισάγει το νέο password που επιθυμεί δύο φορές για επιβεβαίωση ορθότητας και εφόσον είναι ορθό και πληροί τις προϋποθέσεις το password αλλάζει και μεταφέρεται στο Log-In Screen για να συνδεθεί στην εφαρμογή.

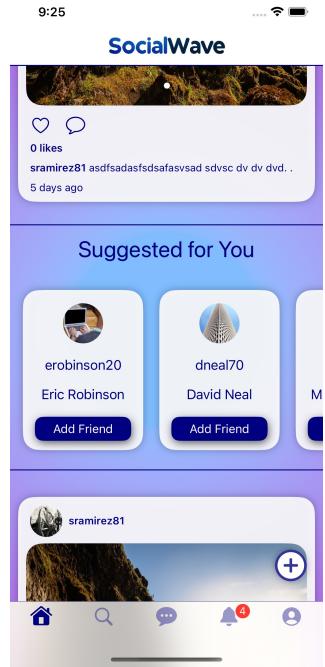




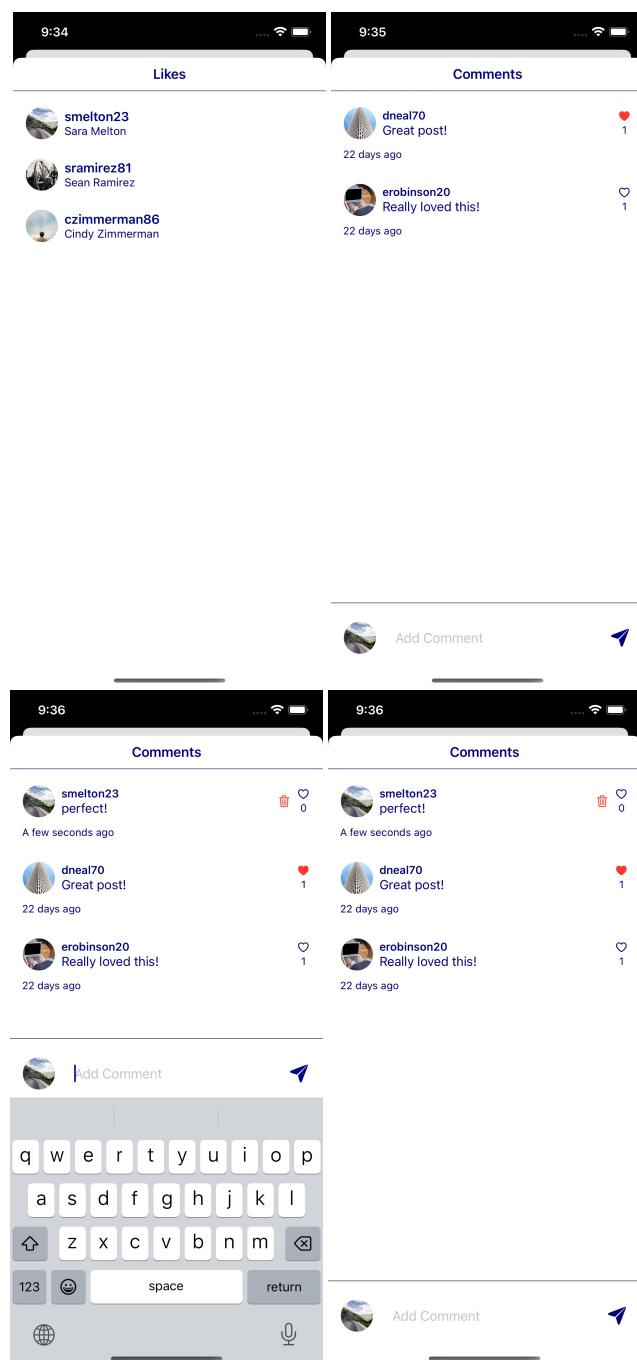
- **Home Screen:** Η αρχική σελίδα της εφαρμογής όπου παρουσιάζονται όλα τα post των φίλων του χρήστη. Τα post μπορεί να περιέχουν 1 έως 5 φωτογραφίες, οι οποίες εμφανίζονται σκρολάροντας από δεξιά προς τα αριστερά, ενώ υπάρχει και ο indicator που δείχνει σε ποιά φωτογραφία βρισκόμαστε. Στο κάτω μέρος της οθόνης υπάρχει το bottom navigation bar με το οποίο ο χρήστης μπορεί να περιηγηθεί σε όλες τις σειδες της εφαρμογής (Home, Search, Chats, Notifications, Profile).

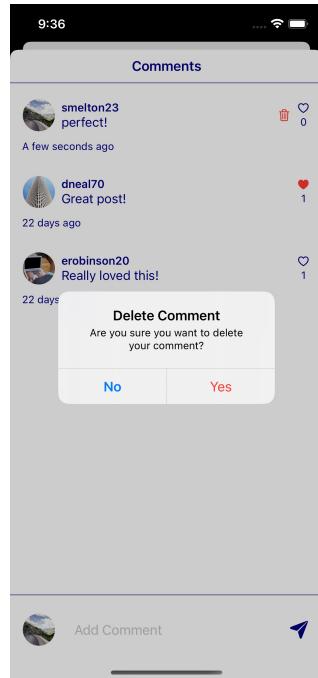


- **Suggest Friends in Home Screen:** Στο Home Screen υπάρχει το horizontal scroll bar "Suggested For You" που εμφανίζει στον χρήστη προτεινόμενα άτομα για να τους κάνει φίλους.

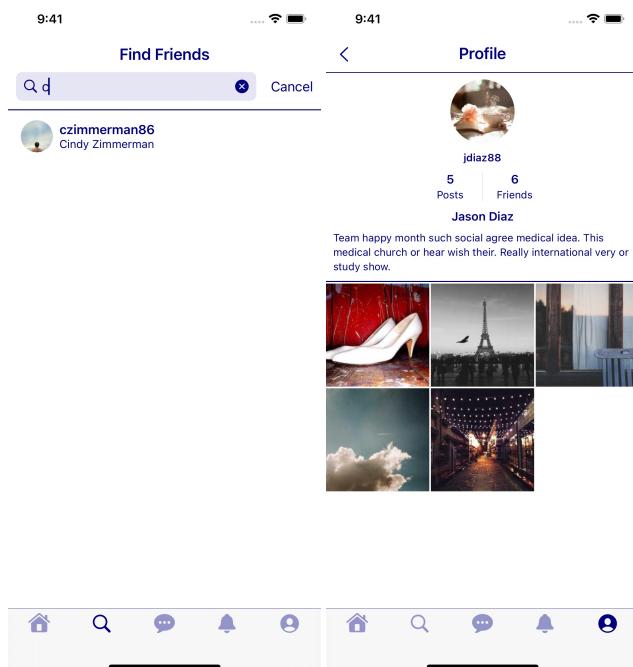


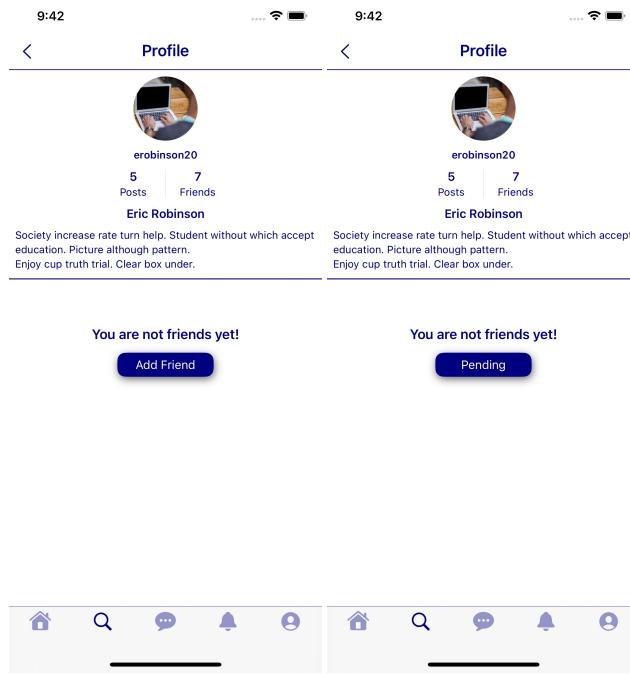
– Like και Comment στα Posts: Κάνοντας κλικ στο σύμβολο "καρδιά", ο χρήστης μπορεί να κάνει like ή να αφαιρέσει κάποιο like που έχει ήδη κάνει (unlike). Επίσης πατώντας τον αριθμό των likes ενός post, εμφανίζεται μία λίστα με τα άτομα που έχουν κάνει like στο post. Κάνοντας κλικ στο σύμβολο "σύννεφο", εμφανίζεται μία λίστα με τα άτομα που έχουν κάνει σχόλιο στο post και ο χρήστης μπορεί πάλι να κάνει like ή unlike σε κάποιο σχόλιο. Τέλος, ο χρήστης μπορεί να κάνει το δικό του σχόλιο στο post και να το διαγράψει όποτε αυτός θελήσει.



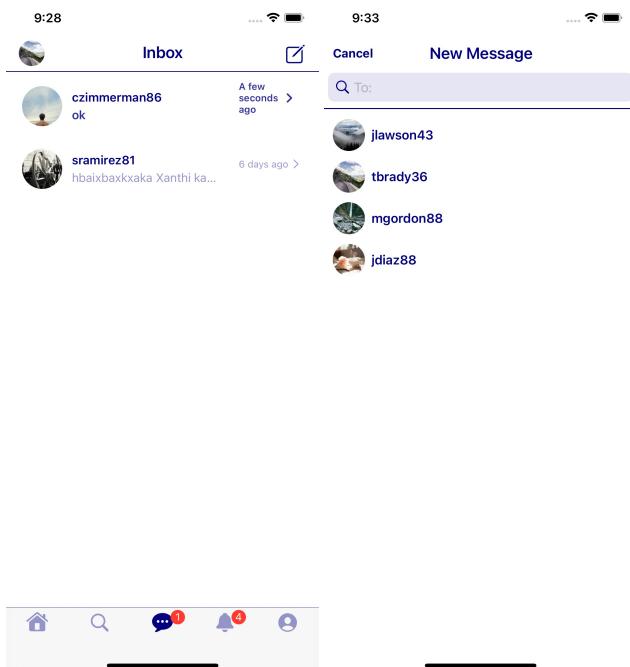


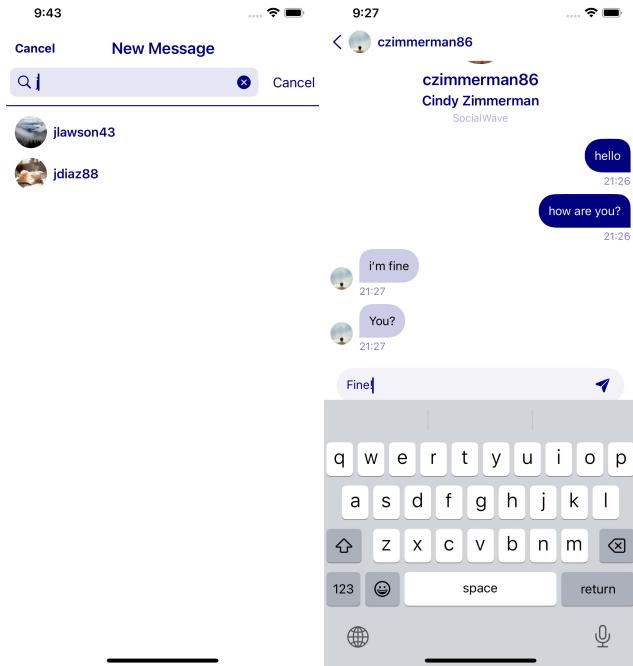
– **Search Screen:** Στο Search Screen, ο χρήστης μπορεί να αναζητήσει άλλους χρήστες, φίλους του ή και όχι, και να περιηγηθεί στα προφίλ τους εφόσον είναι φίλοι ή τους στείλει αίτημα φιλίας. Κάνοντας κλικ στο "Add Friend" στέλνεται ένα αίτημα φιλίας στον άλλον χρήστη ("Pending") και εάν εκείνος το δεχτεί, ο χρήστης θα έχει πρόσβαση πλέον στο προφίλ του φίλου του όπου θα μπορεί να δει όλα τα post του και να αλληλεπιδράσει με αυτά.



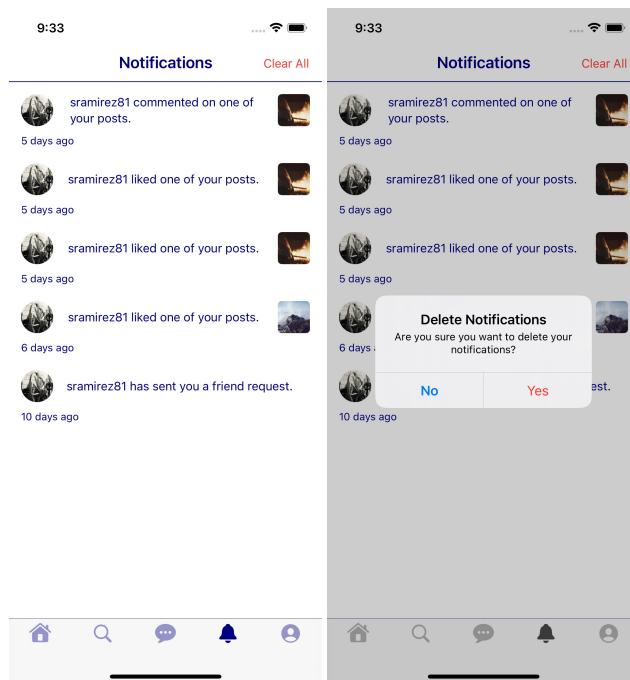


- **Chats(Inbox) Screen:** Σε αυτή την σελίδα εμφανίζονται όλες οι συνομιλίες που έχει ο τρέχων χρήστης με άλλους χρήστες. Εάν υπάρχει νέο μήνυμα, θα εμφανιστεί μία κόκκινη ένδειξη στο σύμβολο των chats κάτω στην μπάρα της εφαρμογής, με τον αριθμό των νέων μηνυμάτων. Κάνοντας κλικ στο κουμπί πάνω δεξιά, ο χρήστης μεταφέρεται στην σελίδα "New Message" όπου μπορεί να αναζητήσει φίλους του και να τους στείλει μήνυμα για να ξεκινήσει μία συνομιλία μαζί τους. Εάν έχει ήδη κάποια συνομιλία με άλλον χρήστη μπορεί να την δει και να στείλει νέα μηνύματα πατώντας πάνω και θα μεταφερθεί στο chat τους.

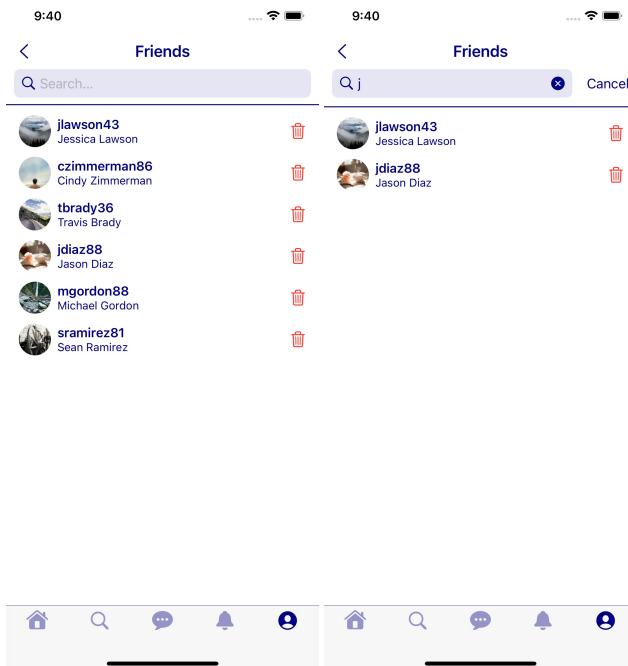
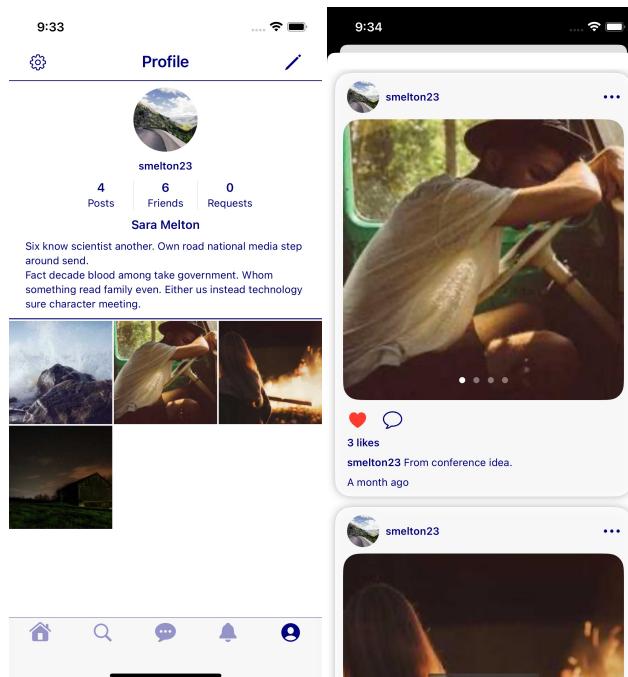


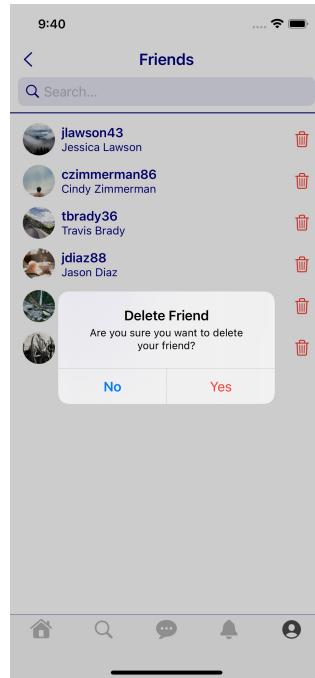


– **Notifications Screen:** Στη σελίδα αυτή εμφανίζονται όλες οι ειδοποιήσεις που μπορεί έχει ο χρήστης από likes, comments και αιτήματα φιλίας. Κάνοντας κλικ σε κάποια ειδοποίηση ο χρήστης μεταφέρεται στο συγκεκριμένο post στο προφίλ του για να δει τις αντιδράσεις των φίλων του. Με την επιλογή "Clear All" που βρίσκεται πάνω δεξιά στη σελίδα, μπορεί να διαγράψει όλες τις υπάρχουσες ειδοποιήσεις. Όπως και στο chat, εάν υπάρχουν νέες ειδοποιήσεις, θα εμφανιστεί μία κόκκινη ένδειξη στο σύμβολο των ειδοποιήσεων ("καμπανάκι") κάτω στην μπάρα της εφαρμογής, με τον αριθμό των νέων ειδοποιήσεων.

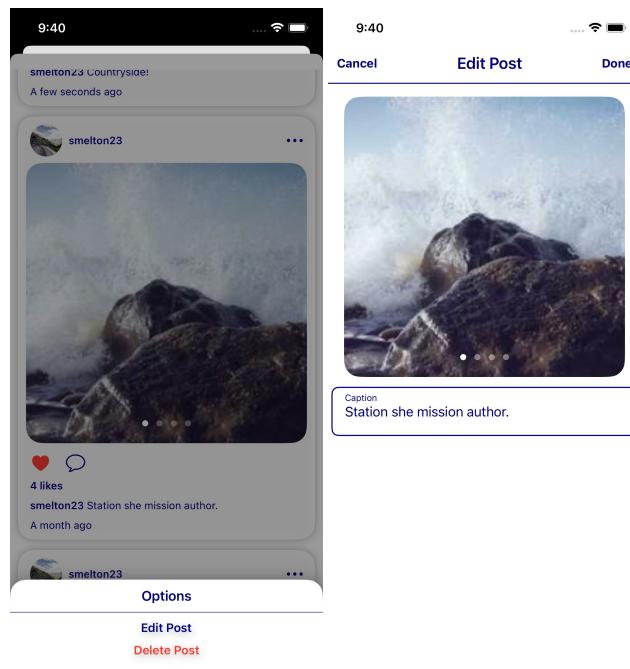


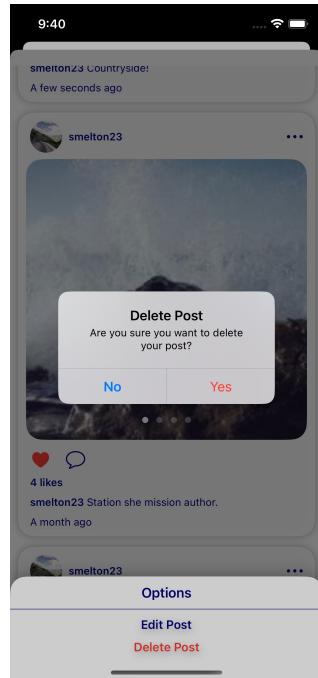
– **Profile Screen:** Η σελίδα Profile Screen αντιπροσωπεύει το προσωπικό προφίλ του χρήστη όπου μπορεί να δει όλες τις δημοσιεύσεις (μαζικά ή και μία μία σαν λίστα) του και τα προσωπικά δεδομένα του και να τα επεξεργαστεί. Ακόμη, έχει πρόσβαση στη λίστα με τους φίλους του και μπορεί να τους αναζητεί στο αντίστοιχο search bar και να πλοηγείται στα προφίλ τους. Εάν επιθυμεί να διαγράψει κάποιον φίλο του, μπορεί να κάνει κλικ στο εικονίδιο ”κάδο” δίπλα από το όνομα του και θα εμφανιστεί το σχετικό μήνυμα επιβεβαίωσης διαγραφής φίλου.



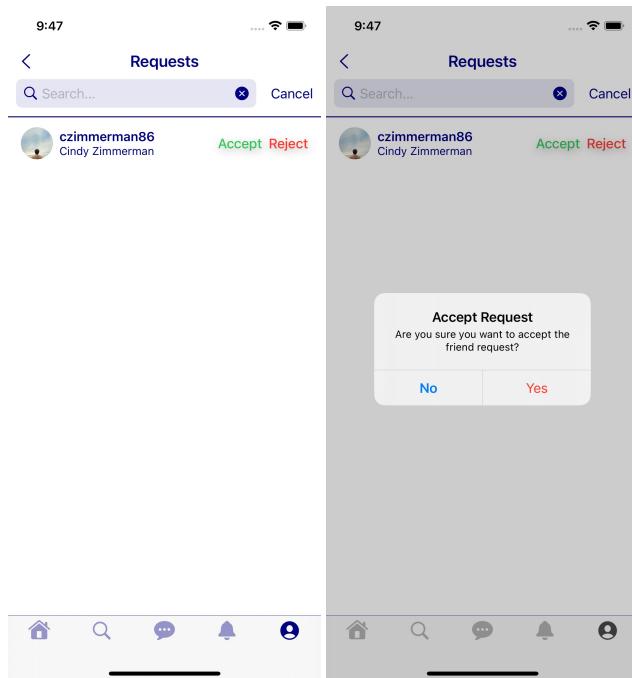


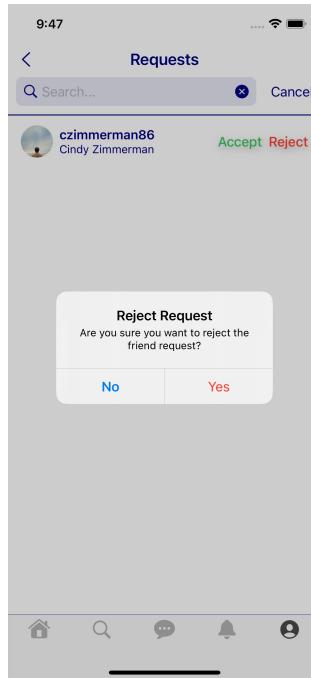
Επιπρόσθετα, ο χρήστης μπορεί να επεξεργαστεί μία δημοσίευση ή να την διαγράψει από το κουμπί με τις τρεις τελείες πάνω δεξιά στη δημοσίευση, που θα του εμφανίσει τις αντίστοιχες επιλογές.



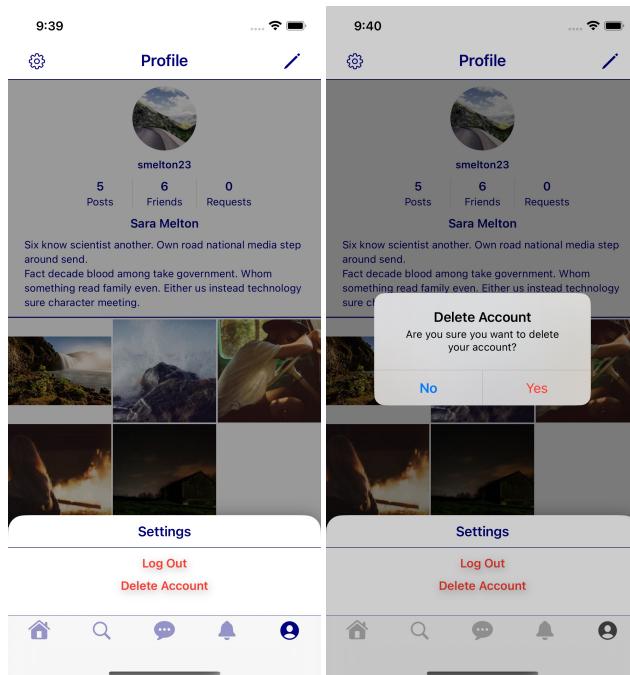


Τα αιτήματα φιλίας του χρήστη εμφανίζονται στο πεδίο "Requests" στην σελίδα του προφίλ του. Κάνοντας κλικ στο πεδίο αυτό εμφανίζεται μία λίστα με τα αιτήματα που έχει από άλλους χρήστες και μπορεί είτε να δεχτεί το αίτημα είτε να το απορρίψει, πατώντας το αντίστοιχο κουμπί "Accept" ή "Reject".

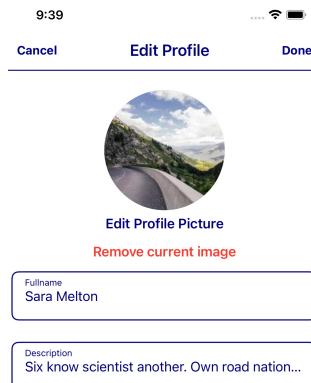




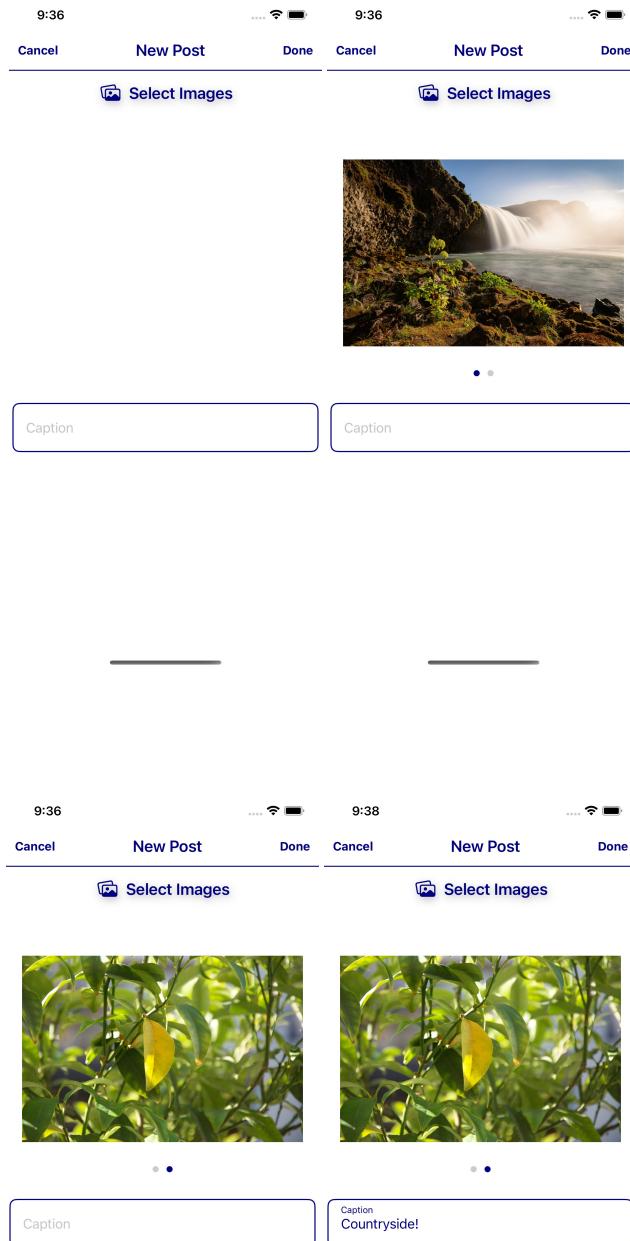
Οι ρυθμίσεις του προφίλ του χρήστη μπορούν να εμφανιστούν κάνοντας κλικ στο εικονίδιο "γρανάζι" πάνω αριστερά στο προφίλ του και οι επιλογές που έχει είναι να αποσυνδεθεί από το προφίλ του πατώντας στο "Log Out" ή να διαγράψει οριστικά το προφίλ του (αφού επιβεβαιώση την πράξη του στο σχετικό μήνυμα).



Τέλος, από το κουμπί πάνω δεξιά στο προφίλ του ο χρήστης μπορεί να επεξεργαστεί τα προσωπικά του δεδομένα όπως να ανεβάσει φωτογραφία προφίλ, να την διαγράψει ή να την αλλάξει, να αλλάξει το όνομα του, και την περιγραφή στο προφίλ του.



- **Upload Post Screen:** Στη σελίδα αυτή, ο χρήστης έχει την δυνατότητα να ανεβάσει μία νέα δημοσίευση κάνοντας κλικ στο "Select Images" και επιλέγοντας 1 έως 5 φωτογραφίες από το άλμπουμ του τηλεφώνου του, τις οποίες μπορεί να δει στην συνέχεια σκρολάροντας στο πλάι και στο πεδίο caption γράφει την περιγραφή που επιθυμεί. Τέλος, κάνοντας κλικ στο "Done", δημιουργείται το post και ανεβαίνει στην εφαρμογή.



13 Μελλοντικές επεκτάσεις και προσθήκες

Η εφαρμογή ”SocialWave”, ως εφαρμογή Social Media, θα μπορούσε να επεκταθεί περεταίρω μελλοντικά και να περιέχει επίσης λειτουργίες όπως:

- Ομαδικές Συζητήσεις: Οι χρήστες θα έχουν την δυνατότητα να δημιουργούν ομαδικές συζητήσεις με πολλούς φίλους τους, επιτρέποντάς τους να συνομιλούν και να μοιράζονται περιεχόμενο.
- Συμμετοχή σε κοινότητες: Οι χρήστες θα μπορούν να δημιουργούν και να συμμετέχουν σε κοινότητες/ομάδες με κοινά ενδιαφέροντα όπως αθλήματα, τέχνες, μόδα, επιστήμη κλπ και να μοιράζονται τις απόψεις/προβληματισμούς τους.
- Κοινές Δημοσιεύσεις: Οι χρήστες θα μπορούν να δημοσιεύουν περιεχόμενο μαζί με άλλους χρήστες, δημιουργώντας έτσι κοινές δημοσιεύσεις.
- Αποστολή Πολυμέσων στο Chat: Οι χρήστες θα μπορούν να στέλνουν στο chat φωτογραφίες, ηχητικά μηνύματα, βίντεο και αρχεία, πέρα από τα απλά μηνύματα.
- Εμπλουτισμός Αλγορίθμου ”Suggest Friends”: Ο αλγόριθμος μπορεί να εμπλουτιστεί έτσι ώστε οι προτεινόμενοι φίλοι να εμφανίζονται με βάση επιπλέον παράγοντες όπως συμμετοχή σε ομάδες, like ή comment σε δημοσιεύσεις, επίσκεψη στο προφίλ κλπ.
- Ιστορίες (Stories): Δυνατότητα για προσωρινές δημοσιεύσεις που διαρκούν μόνο 24 ώρες, παρόμοια με τα Stories σε άλλες πλατφόρμες κοινωνικής δικτύωσης.
- Στατιστικά Προφίλ: Οι χρήστες θα έχουν πρόσβαση σε αναλυτικά στατιστικά για το προφίλ τους, όπως αριθμό αλληλεπιδράσεων, προβολές, και στατιστικά για τις δημοσιεύσεις τους.

14 Βιβλιογραφία

Αναφορές

1. [Node.js](#)
2. [MongoDB Manual](#)
3. [Exercises in MongoDB](#)
4. [Swift Documentation](#)
5. [SwiftUI Tutorials](#)
6. [Socket Programming in Node.js](#)
7. [HTTPS Server in Node.js](#)
8. [Email Verification with OTP in Node.js](#)
9. [Hashing in Password in Node.js](#)
10. [JWT Authentication with Node.js](#)