Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής
και Υπολογιστών

Ανάλυση και Οπτικοποίηση Δεδομένων από Μέσα Κοινωνικής Δικτύωσης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ηλίας Ν. Αντωνίου

Επιβλέπων : Ιωάννης Βασιλείου
Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2015

Ηλίας Ν. Αντωνίου

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογιας Πληροφορικής
και Υπολογιστων

Ανάλυση και Οπτικοποίηση Δεδομένων από Μέσα Κοινωνικής Δικτύωσης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ηλίας Ν. Αντωνίου

Επιβλέπων : Ιωάννης Βασιλείου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 27η Μαρτίου 2015.

.....	.....	.....
Ιωάννης Βασιλείου	Κωνσταντίνος Κοντογιάννης	Στέφανος Κόλλιας
Καθηγητής Ε.Μ.Π.	Αν. Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2015

(*Υπογραφή*)

.....
ΑΝΤΩΝΙΟΥ ΗΛΙΑΣ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Ηλίας Ν. Αντωνίου

Περίληψη

Οι πλατφόρμες μικρο-ιστολογίου έχουν γίνει στις μέρες μας ένα πολύ δημοφιλές εργαλείο επικοινωνίας μεταξύ των χρηστών του Διαδικτύου. Εκατομμύρια χρήστες μοιράζονται τις απόψεις τους πάνω σε θέματα της καθημερινότητας και συνεπώς οι ιστοσελίδες αυτές κατακλύζονται από πληθώρα δεδομένων. Μια από τις δημοφιλέστερες πλατφόρμες αυτού του τύπου είναι η ιστοσελίδα κοινωνικής δικτύωσης *Twitter*. Τα δεδομένα που υπάρχουν στο *Twitter* αποτελούν σημαντική πηγή πληροφοριών, που για να αξιοποιηθούν όμως πρέπει πρώτα να οργανωθούν και να αναλυθούν με κάποιο αποτελεσματικό τρόπο.

Ο όγκος των δεδομένων που υπάρχουν στο *Twitter,* καθώς και ο ρυθμός που αυτά παράγονται καθιστά την διαχείριση τους μια σημαντική πρόκληση. Η ακριβής γεωγραφική θέση δημιουργίας ενός τιτιβίσματος μπορεί πλέον να προσδιοριστεί αυτόματα με την βοήθεια τεχνολογιών όπως *WiFi* ή *GPS* που τα σύγχρονα κινητά τηλέφωνα παρέχουν. Σκοπός της παρούσας διπλωματικής είναι η ανάπτυξη μιας διαδικτυακής εφαρμογής που θα διαχειρίζεται τα δεδομένα του *Twitter* σε πραγματικό χρόνο, δίνοντας έμφαση τόσο στην γεωγραφική πληροφορία που συνοδεύει αυτά τα δεδομένα, όσο και στην εξαγωγή γλωσσικού περιεχομένου (συναίσθημα, *hashtags*, *URLs*) από αυτά.

Για την συλλογή των δεδομένων πραγματικού χρόνου αξιοποιήσαμε τις δυνατότητες που μας δίνει η εξαιρετικά οργανωμένη και λειτουργική διεπαφή προγραμματισμού εφαρμογών (*API*) του *Twitter*. Στη συνέχεια, αναλύσαμε τα δεδομένα αυτά με ένα εργαλείο επεξεργασίας φυσικής γλώσσας, το *AlchemyAPI*, προκειμένου να κάνουμε ανάλυση συναισθημάτων (*Sentiment Analysis*) στο κείμενο των τιτιβισμάτων. Οι πληροφορίες που εξάγονται από την ανάλυση, μαζί με αυτές που προσφέρει η διεπαφή του *Twitter*, αποθηκεύονται σε μια βάση δεδομένων. Ως βάση δεδομένων χρησιμοποιήσαμε την μη σχεσιακή βάση δεδομένων γράφου *Neo4j*, καθώς ο τρόπος που αποθηκεύει και ανασύρει τα δεδομένα ταιριάζει στον έντονα δομημένο γύρω από τις σχέσεις τομέα των κοινωνικών δικτύων.

Για την ανάπτυξη της εφαρμογής μας χρησιμοποιήσαμε την πλατφόρμα *Node.js*. Το *Node.js* χρησιμοποιεί την *Javascript* στην πλευρά του εξυπηρετητή, για την δημιουργία γρήγορων και με υψηλά επίπεδα κλιμάκωσης εφαρμογών. Τα υψηλά επίπεδα κλιμάκωσης, είναι ιδιαίτερα σημαντικό στοιχείο για μια εφαρμογή διαχείρισης δεδομένων, αφού αυτή απαιτείται να ανταποκρίνεται σε μεγάλο όγκο δεδομένων. Στην πλευρά του εξυπηρετούμενου χρησιμοποιήσαμε ένα σύνολο γνωστών διαδικτυακών τεχνολογιών (*HTML*, CSS, Javascript, JQuery) και βιβλιοθηκών (*OpenLayers*, *Sigma*, *Raphäel*), προκειμένου να εμφανίσουμε τα αποτελέσματα στον χρήστη.

Λέξεις κλειδιά

κοινωνικά δίκτυα, Twitter, μεταδεδομένα τοποθεσίας, ανάλυση συναισθημάτων, βάσεις δεδομένων γράφου, Neo4j, Cypher, πλατφόρμα Node.js

Abstract

The micro-blogging platforms have become today a very popular communication tool among Internet users. Millions of users share opinions on different aspects of everyday life and therefore micro-blogging websites are rich sources of data. One of the most popular micro-blogging platform is the Twitter social network. The data that exists in Twitter can be considered as a valuable source of information. In order to exploit this information though, it must firstly be organized and analyzed in an efficient way.

The amount of data produced in Twitter and the fact that these data are produced in real-time make their management really challenging. The purpose of this thesis is to enhance this section of research and the development of a data management web application. With technologies such as WiFi or GPS, modern cell phones can automatically determine the exact geographical location that a tweet was created at. The aim of our application is to manage real-time data that comes from Twitter emphasizing both in geospatial information, and in lingual content (sentiment, hashtags, URLs) that comes along with these data.

During the process of data collection, we used in a large extent the capabilities of the highly organized and functional Twitter API. After that, the collected data are processed with a Natural Language Processing (NLP) tool, such as AlchemyAPI, in order to perform sentiment analysis to the text of the tweets. The information that is retrieved from this analysis, as well as the one that the Twitter API offers, are stored in a database. As a database we used the NoSQL graph database Neo4j. The way Neo4j stores and retrieves data is a natural fit for the overtly relationship-centered domain of social networks.

Our application is developed using the Node.js platform. Node.js uses server-side Javascript for easily building fast, scalable networks applications. Scalability is very important for data management applications, since they are supposed to handle a huge amount of data. At the client's side, we used an amount of popular web technologies (HTML, CSS, Javascript, JQuery) and libraries (OpenLayers, Sigma, Raphäel), in order to visualize results to user.

Key words

social networks, Twitter, geospatial data, sentiment analysis, graph databases, Neo4j, Cypher, Node.js platform

Ηλίας Ν. Αντωνίου

Ευχαριστήριο Σημείωμα

Θα ήθελα να ευχαριστήσω θερμά τον υποψήφιο διδάκτορα Ευσταθιάδη Χριστόδουλο και τον ερευνητή Σκούτα Δημήτριο για την πολύτιμη καθοδήγηση τους καθ' όλη την διάρκεια εκπόνησης της παρούσας εργασίας καθώς και για τον χρόνο που αφιέρωσαν.

Ευχαριστώ επίσης τον επιβλέποντα καθηγητή κ. Ιωάννη Βασιλείου, για την δυνατότητα που μου έδωσε να ασχοληθώ με την διπλωματική αυτή εργασία.

Θέλω να ευχαριστήσω του γονείς μου Νικόλαο και Ελένη για την συνεχή και κάθε είδους υποστήριξη καθ' όλα τα χρόνια των προπτυχιακών μου σπουδών.

Τέλος, να ευχαριστήσω την αδερφή μου Αικατερίνη για την υπομονή και υποστήριξη της καθ' όλα τα χρόνια της συγκατοίκησης μας και κυρίως τον καιρό της ενασχόλησής μου με την παρούσα εργασία, και τον αδερφό μου Σωτήριο για την υπομονή και κατανόηση του κατά τη διάρκεια της απουσίας μου.

Πίνακας περιεχομένων

Πίνακας σχημάτων

1 Εισαγωγή

1.1 Γενικό Πλαίσιο

Την τελευταία δεκαπενταετία έχει παρατηρηθεί παγκοσμίως μια τεράστια αύξηση του αριθμού των χρηστών που χρησιμοποιούν το Διαδίκτυο. Ενδεικτικά αναφέρουμε ότι 360 εκατομμύρια χρήστες χρησιμοποιούσαν το Διαδίκτυο στην αρχή του αιώνα μας. Στις αρχές του 2014 ο αριθμός αυτός ξεπέρασε τα 2,8 δισεκατομμύρια, δηλαδή αύξηση της τάξης του 676% [*S*].

Στην αύξηση αυτή των χρηστών του Διαδικτύου σημαντικό ρόλο παίζουν οι υπηρεσίες κοινωνικής δικτύωσης. Οι ιστοσελίδες κοινωνικής δικτύωσης κερδίζουν συνεχώς περισσότερους χρήστες, καθώς τους παρέχουν την δυνατότητα να αλληλεπιδρούν και να επικοινωνούν μεταξύ τους. Έτσι οι χρήστες αυτών των ιστοσελίδων μπορούν να μοιραστούν ενδιαφέροντα, ιδέες, γεγονότα, φωτογραφίες ή ακόμα και βίντεο, επιτρέποντας τους να αξιολογήσουν και να ανακαλύψουν ο ένας τον άλλον καθώς και τα κοινά τους ενδιαφέροντα.

Χαρακτηριστικά παραδείγματα κοινωνικών δικτύων είναι κατά κύριο λόγο το *Facebook* και δευτερευόντως το *Twitter* [*MSGL*]. Με βάση τα σημερινά επίσημα στατιστικά στοιχεία το *Facebook* απαριθμεί 1.32 δισεκατομμύρια ενεργούς χρήστες [*FBC*], ενώ το *Twitter* περισσότερους από 270 εκατομμύρια [*ATC*]. Για να πάρουμε μια ιδέα του πλήθους των δεδομένων που παράγουν αυτοί οι χρήστες, στο *Facebook* καθημερινά ανεβαίνουν περισσότερες από 350 εκατομμύρια φωτογραφίες, ενώ στο *Twitter* καθημερινά δημοσιεύονται 500 εκατομμύρια τιτιβίσματα (*tweets*).

Γίνεται εύκολα λοιπόν αντιληπτό ότι αυτές οι ιστοσελίδες κατακλύζονται από πληθώρα δεδομένων. Πολλές φορές μάλιστα αυτά τα δεδομένα, συνοδεύονται από μεταδεδομένα, όπως η γεωγραφική θέση (*geotagging*) ή κάποια ετικέτα. Λόγω της μαζικής παραγωγής προσιτών σε κόστος κινητών τηλεφώνων με ενσωματωμένη μονάδα παγκοσμίου συστήματος εντοπισμού θέσης (*Global Positioning System* ή *GPS*), τα μεταδεδομένα τοποθεσίας, όπως το γεωγραφικό μήκος και πλάτος συνδέονται αυτόματα με τα υπόλοιπα δεδομένα που οι χρήστες παράγουν.

Είναι λοιπόν εμφανής η ανάγκη να μπορέσουμε να συλλέξουμε αυτά τα δεδομένα και να τα διαχειριστούμε κατάλληλα ώστε να εξάγουμε χρήσιμες πληροφορίες τόσο σχετικά με την κατανομή των δεδομένων στον χώρο, όσο και διάφορες συμπεριφορές των χρηστών.

1.2 Σκοπός και Περιγραφή

Σκοπός της παρούσας διπλωματικής είναι να εξαχθούν και να αξιοποιηθούν οι πληροφορίες που αναφέρθηκαν παραπάνω. Έχοντας αυτό στο μυαλό μας αναπτύξαμε μια εφαρμογή η οποία χρησιμοποιεί δεδομένα από το *Twitter* για την εξαγωγή πληροφορίας από αυτά.

Πιο συγκεκριμένα ενδιαφερόμαστε για την συλλογή τιτιβισμάτων (*tweets*) σε πραγματικό χρόνο, που συνοδεύονται από γεωγραφική πληροφορία, με σκοπό την ανάπτυξη ενός συστήματος που παρέχει τα εξής στο χρήστη:

- Εμφάνιση αυτών των τιτιβισμάτων σε πραγματικό χρόνο μαζί με πληροφορίες που σχετίζονται με αυτά τα τιτιβίσματα. Τέτοιες πληροφορίες μπορεί να είναι το πόσο ενεργοί είναι οι χρήστες ανά χρονική στιγμή, ποιοι χρήστες είναι περισσότερο ενεργοί καθώς και τα θέματα συζήτησης που είναι περισσότερο δημοφιλή.
- Ένα προχωρημένο είδος αναζήτησης τιτιβισμάτων με βάση τόσο τη χρονική στιγμή όσο και την ακριβή τοποθεσία δημιουργίας τους. Ουσιαστικά μια εύκολη στη χρήση αλλά και ταυτόχρονα αποδοτική μορφή αναζήτησης στο χρόνο και στο χώρο.
- Ένα προχωρημένο είδος αναζήτησης τιτιβισμάτων βάση του θέματος τους αλλά και του χρήστη δημιουργού τους. Η συγκεκριμένη μορφή αναζήτησης μπορεί να συνδυαστεί με την αναζήτηση στο χώρο και στο χρόνο.
- Ένα περιβάλλον εκμάθησης της γλώσσας ερωτημάτων *Cypher*, μέσω έτοιμων ερωτημάτων που μπορούν πολύ εύκολα να προσαρμοστούν στις ανάγκες του χρήστη. Με αυτόν τον τρόπο ο χρήστης μπορεί να εστιάσει το ενδιαφέρον του στα δεδομένα που επιθυμεί.

Αφού συλλέξουμε τα δεδομένα, τα επεξεργαζόμαστε κατάλληλα βάση διαφόρων χαρακτηριστικών τους και στην συνέχεια τα αποθηκεύουμε σε μια βάση δεδομένων. Ως βάση δεδομένων χρησιμοποιήσαμε την *Neo4j*. Η *Neo4j* είναι μια μη-σχεσιακή (*NoSQL*) βάση δεδομένων, και πιο συγκεκριμένα μια βάση δεδομένων γράφου (*graph database*). Η επιλογή της έγινε ώστε να μπορέσουμε στην συνέχεια εύκολα και αποδοτικά να ανακτήσουμε την αποθηκευμένη πληροφορία, εκμεταλλευόμενοι την υψηλή της απόδοση σε ερωτήματα που αφορούν κάποιο τμήμα του γράφου, καθώς και την ευελιξία των βάσεων δεδομένων αυτού του τύπου.

Την εφαρμογή μας την αναπτύξαμε στο *Node.js*, μια πλατφόρμα που μας επιτρέπει να χρησιμοποιούμε *JavaScript* και στις δύο άκρες, τόσο στην πλευρά του εξυπηρετητή (*server-side JavaScript*) όσο και στην πλευρά του εξυπηρετούμενου (*client*). Η συγκεκριμένη πλατφόρμα μας παρέχει μια βιβλιοθήκη ώστε να μπορέσουμε να χρησιμοποιήσουμε την *REST* διεπαφή (*REST API*) της *Neo4j* βάσης δεδομένων μας μέσα από το *Node.js*.

Τέλος δημιουργήσαμε ένα διαδικτυακό γραφικό περιβάλλον ,το οποίο θα δίνει στον χρήστη την δυνατότητα να λάβει εύκολα και αποτελεσματικά την ζητούμενη

πληροφορία. Για αυτόν τον σκοπό χρησιμοποιούμε διάφορα μέσα για την οπτικοποίηση της πληροφορίας όπως χάρτες, γραφήματα, σύννεφα λέξεων (*tag clouds*) και χάρτες θερμότητας (*heatmaps*) . Η εφαρμογή θα είναι δυναμική καθώς ο χρήστης μπορεί να αλληλεπιδράσει μαζί της είτε μέσω ενός ερωτήματος (*query*) στην βάση δεδομένων, είτε μέσω διαφόρων επιλογών αναζήτησης (χρόνος, χώρος, *hashtag*).

1.3 Οργάνωση Κειμένου

Στο δεύτερο κεφάλαιο της παρούσας διπλωματικής θα περιγράψουμε το κοινωνικό δίκτυο του *Twitter*. Θα παρουσιάσουμε τον τρόπο με τον οποίο συνδέονται μεταξύ τους οι χρήστες σε αυτό, καθώς και το είδος των πληροφοριών που ρέουν στο εσωτερικό του. Θα καταλάβουμε τη χρησιμότητα των πληροφοριών αυτών και για ποιον λόγο επιβάλλεται η ανάπτυξη εφαρμογών που αξιοποιούν αυτές τις πληροφορίες. Θα δούμε παραδείγματα τέτοιων εφαρμογών καθώς και τις σύγχρονες ερευνητικές τάσεις που αναπτύσσονται πάνω σε αυτό το δίκτυο. Στην συνέχεια του κεφαλαίου, θα παρουσιάσουμε τον τρόπο με τον οποίο μπορούμε να αποκτήσουμε πρόσβαση στα δεδομένα του *Twitter* μέσω 4 διεπαφών προγραμματισμού εφαρμογών (*Application Programming Interfaces – APIs*). Θα δούμε αναλυτικά κάθε μία από αυτές τις διεπαφές και τους λόγους που μας οδήγησαν στην τελική επιλογή μας. Στο τέλος του κεφαλαίου θα παρουσιάσουμε το πρωτόκολλο *OAuth*, μια σύγχρονη μέθοδο πιστοποίησης που χρησιμοποιούμε για την σύνδεση της εφαρμογής μας στο *Twitter*.

Στο τρίτο κεφάλαιο θα δούμε την ανάλυση συναισθημάτων (*Sentiment Analysis*), έναν από τους πλέον δραστήριους ερευνητικά κλάδους του τομέα της επεξεργασίας φυσικής γλώσσας (*Natural Language Processing – NLP*), προκειμένου να προσδιορίσουμε και να εξαγάγουμε δομημένη πληροφορία από το κείμενο. Μέσα από την ανάλυση συναισθημάτων θα αποφασίσουμε για την διάθεση του ομιλητή σχετικά με κάποιο θέμα ή την σχετική με τα συμφραζόμενα πολικότητα ενός εγγράφου. Στο τέλος θα δούμε το *AlchemyAPI*, μια διεπαφή που μας προσφέρει έτοιμα εργαλεία επεξεργασίας φυσικής γλώσσας.

Στο κεφάλαιο 4 θα παρουσιάσουμε τη *Neo4j*, τη βάση δεδομένων που χρησιμοποιήσαμε για την αποθήκευση των δεδομένων μας. Θα δούμε τα χαρακτηριστικά των βάσεων δεδομένων αυτού του τύπου (*graph databases*), σε ποια από αυτά η *Neo4j* διαφέρει από τις υπόλοιπες βάσεις δεδομένων του ίδιου τύπου καθώς και τα ιδιαίτερα χαρακτηριστικά της που οδήγησαν στην επιλογή της. Στην συνέχεια του κεφαλαίου θα κάνουμε μια συνοπτική παρουσίαση της γλώσσας ερωτημάτων *Cypher*, με σκοπό ο αναγνώστης να είναι σε θέση να δημιουργήσει και να εκτελέσει κάποια απλά ερωτήματα μέσα από την εφαρμογή μας. Στο τέλος, θα δούμε τα χωρικά ευρετήρια που χρησιμοποιούνται για την αποτελεσματική ανάσυρση χωρικών δεδομένων και αποδοτική απάντηση ερωτημάτων στο χώρο. Θα περιγράψουμε αναλυτικά τα R-δέντρα, τον πιο διαδεδομένο τύπο χωρικών ευρετηρίων. Θα δούμε με ποιόν τρόπο αποτελούν γενίκευση των B-δέντρων στις δύο διαστάσεις καθώς και τον τρόπο που εκτελούνται οι βασικές λειτουργίες ευρετηρίου σε αυτού του είδους τα δέντρα.

Αμέσως μετά θα περιγράψουμε την πλατφόρμα *Node.js*. Θα δούμε αναλυτικά τα

προτερήματα που μας δίνει η χρησιμοποίηση της *Javascript* στην πλευρά του εξυπηρετητή (*server side*), και το που οφείλεται η ικανότητα του *Node* να πετυχαίνει πολύ υψηλά επίπεδα κλιμάκωσης (*scalability*). Στην συνέχεια, θα περιγράψουμε τον διαχειριστή πακέτων του *Node* (*Node Package Manager – NPM*)˙ έναν από τους λόγους της επιτυχίας του *Node*. Στο σημείο αυτό θα δούμε βήμα-βήμα την κατασκευή του εξυπηρετητή μας και τα θέματα που κληθήκαμε να αντιμετωπίσουμε. Στο τέλος του κεφαλαίου, θα μεταφερθούμε στην πλευρά του εξυπηρετούμενου (*client side*), όπου θα περιγράψουμε τις βασικές τεχνολογίες και βιβλιοθήκες που χρησιμοποιήσαμε προκειμένου να πετύχουμε την οπτικοποίηση των δεδομένων μας.

Στο έκτο κεφάλαιο και έχοντας δει αναλυτικά τα βασικά συστατικά της εφαρμογής μας, είμαστε πλέον σε θέση να περιγράψουμε τον τρόπο που αυτά συνδυάζονται για να επιτευχθεί το επιθυμητό αποτέλεσμα. Αρχικά θα δούμε την αρχιτεκτονική της εφαρμογής και θα προσπαθήσουμε να εξηγήσουμε τους λόγους που μας οδήγησαν στις βασικές σχεδιαστικές επιλογές μας. Στο τέλος του κεφαλαίου, θα περιγράψουμε τόσο την διαδικασία συλλογής και ανάλυσης δεδομένων μέσω μιας ανεξάρτητης εφαρμογής που τρέχει στο παρασκήνιο, όσο και το σχήμα των δεδομένων μας και πως αυτό ταιριάζει με την βάση δεδομένων που χρησιμοποιήσαμε.

Στο τελευταίο κεφάλαιο θα παρουσιάσουμε την εφαρμογή μας. Θα δούμε αναλυτικά όλες τις δυνατότητες που αυτή δίνει στο χρήστη και θα περιγράψουμε τον τρόπο που αυτός μπορεί να χρησιμοποιήσει αυτές τις δυνατότητες. Τέλος, θα αξιολογήσουμε το βαθμό που αυτή ανταποκρίνεται στο σκοπό της για αναζήτηση τιτιβισμάτων στο χώρο και στο χρόνο.

2 Twitter

2.1 Περιγραφή

Το *Twitter* είναι μια ιστοσελίδα κοινωνικής δικτύωσης με βάση το *San Francisco* των ΗΠΑ. Ιδρύθηκε τον Μάρτιο του 2006 από τον *Jack Dorsey* και από τα πρώτα στάδια της ζωής του γνώρισε ραγδαία ανάπτυξη. Σήμερα μετρά περισσότερους από 270 εκατομμύρια ενεργούς χρήστες που δημοσιεύουν 500 εκατομμύρια τιτιβίσματα (*tweets*) την ημέρα.

Κατά πόσο το *Twitter* αποτελεί κοινωνικό δίκτυο (*social network*) ή δίκτυο πληροφοριών (*information network*) αποτελεί αντικείμενο αντιπαράθεσης. Οι *Myers*, *Sharma*, *Gupta* και *Lin* στην εργασία τους [*MSGL*] μελετάνε τα χαρακτηριστικά του γράφου των ακολούθων (*follow graph*) του *Twitter* ώστε να αποφανθούν αν πρόκειται για κοινωνικό ή πληροφοριακό δίκτυο. Ο γράφος ακολούθων ενός κοινωνικού δικτύου χαρακτηρίζεται από μικρά συντομότερα μονοπάτια μεταξύ των χρηστών, μεγάλες συνδεδεμένες συνιστώσες (*connected components*), μεγάλο συντελεστή ομαδοποίησης (*clustering coefficient*) και υψηλό βαθμό αμοιβαιότητας (*reciprocity*), δηλαδή υψηλή τάση οι σχέσεις μεταξύ των χρηστών να εμφανίζονται σε ζεύγη. Το *Facebook* είναι χαρακτηριστικό παράδειγμα κοινωνικού δικτύου. Αντίθετα στα δίκτυα πληροφοριών ο γράφος ακολούθων χαρακτηρίζεται από μεγάλο αριθμό ακμών στους κόμβους (*vertex degree*), έλλειψη αμοιβαιότητας και μεγάλο αριθμό γειτόνων δύο επιπέδων (*two-hop neighbors*). Τα αποτελέσματα της έρευνας που έγινε σε 175 εκατομμύρια χρήστες του *Twitter*, έδειξε ότι αυτό παρουσιάζει χαρακτηριστικά τόσο κοινωνικού όσο και πληροφοριακού δικτύου. Αρχικά συμπεριφέρεται σαν δίκτυο πληροφοριών και εξελίσσεται σε κοινωνικό δίκτυο. Συγκεκριμένα, οι πρώτοι χρήστες τους οποίους ένας καινούριος χρήστης αποφασίζει να ακολουθήσει είναι συνήθως διάσημοι. Με τον καιρό και καθώς αποκτά εμπειρία στην χρήση του *Twitter,* γίνεται πιο επιλεκτικός και επιλέγει ποιους θα ακολουθήσει με άλλα κριτήρια πέρα από την διασημότητα. Έτσι εισέρχεται σε κοινότητες χρηστών με τους οποίους έχει κοινά ενδιαφέροντα ή άλλες κοινωνικές σχέσεις, μετατρέποντας με αυτόν τον τρόπο το *Twitter* σε κοινωνικό δίκτυο.

Η βασική του ιδέα είναι ότι οι χρήστες μπορούν να δημοσιεύσουν ανά πάσα στιγμή μηνύματα κειμένου μήκους έως και 140 χαρακτήρων, γνωστά και ως τιτιβίσματα ή "*tweets*". Αυτό του έχει δώσει και τον χαρακτηρισμό "*SMS* του *Internet*".

Όπως σε κάθε κοινωνικό δίκτυο έτσι και στο *Twitter* υπάρχουν σχέσεις μεταξύ των χρηστών του δικτύου. Στο *Twitter* βασική έννοια στις σχέσεις μεταξύ των χρηστών είναι αυτή του ακόλουθου (*follower*). Κάθε χρήστης μπορεί να ακολουθήσει (κάνει *follow*) οποιονδήποτε άλλον χρήστη επιθυμεί, χωρίς να χρειάζεται η συγκατάθεση του τελευταίου. Με αυτόν τον τρόπο γίνεται ακόλουθος (*follower*) και ενημερώνεται

αυτόματα για τα τιτιβίσματα του χρήστη που ακολουθεί.

Άλλη μια βασική έννοια στο *Twitter* που θα μας απασχολήσει ιδιαιτέρως είναι αυτή του *hashtag*. *Hashtag* είναι η λέξη ενός τιτιβίσματος που ακολουθεί τον ειδικό χαρακτήρα "#". Το *hashtag* εκφράζει το θέμα του τιτιβίσματος και βοηθάει στην ομαδοποίηση των τιτιβισμάτων που έχουν το ίδιο θέμα.

Όμοια ο ειδικός χαρακτήρας "@" ακολουθούμενος από το όνομα κάποιου χρήστη χρησιμοποιείται για να απευθυνθούμε σε αυτόν τον χρήστη και να ξεκινήσουμε μια συζήτηση μαζί του. Επίσης υπάρχει η δυνατότητα να προωθήσουμε ένα τιτίβισμα στους ακόλουθους μας κάνοντας το αναδημοσίευση ή "*retweet*". Σε αυτήν την περίπτωση χρησιμοποιούμε το σύμβολο "*RT*".

Από το 2009 οι χρήστες του *Twitter* μπορούν προαιρετικά να συνοδεύσουν κάθε τους τιτίβισμα με την ακριβή τους τοποθεσία. Αυτό μπορεί να γίνει εξαιρετικά εύκολα με την ενεργοποίηση της σχετικής επιλογής. Ο εντοπισμός της τοποθεσίας γίνεται είτε μέσω της μονάδας παγκοσμίου συστήματος εντοπισμού θέσης (μονάδα *GPS*) της συσκευής είτε, σε περίπτωση που δεν υπάρχει τέτοιο, μέσω του *WiFi* ή της *IP* διεύθυνσης της συσκευής. Στην δεύτερη περίπτωση φυσικά η ακρίβεια είναι αρκετά μειωμένη.

Τέλος να σημειώσουμε ότι το *Twitter* δεν δίνει την δυνατότητα στους χρήστες να κάνουν αναδημοσίευση (*retweet*) που να συνοδεύεται από πληροφορία τοποθεσίας. Μέσω των αναδημοσιεύσεων μπορούμε να παρακολουθήσουμε το πως διαχέεται ένα τιτίβισμα στο δίκτυο του *Twitter*. Θα παρουσίαζε ιδιαίτερο ενδιαφέρον λοιπόν η παρακολούθηση αυτής της διάχυσης όχι μόνο στο δίκτυο του *Twitter*, αλλά και στο χώρο. Δηλαδή πως συνδέονται γεωγραφικά τα αναδημοσιευμένα τιτιβίσματα σε σχέση με το αρχικό τιτίβισμα. Επειδή αυτή η πληροφορία δεν είναι άμεσα διαθέσιμη, η παρούσα εργασία δεν εμβαθύνει προς αυτή την κατεύθυνση. Παρ' όλα αυτά, η παρακολούθηση της πορείας ενός τιτιβίσματος στο χώρο αποτελεί σίγουρα μια από τις πλέον προκλητικές και ενδιαφέρουσες δυνατότητες επέκτασης της εφαρμογής μας.

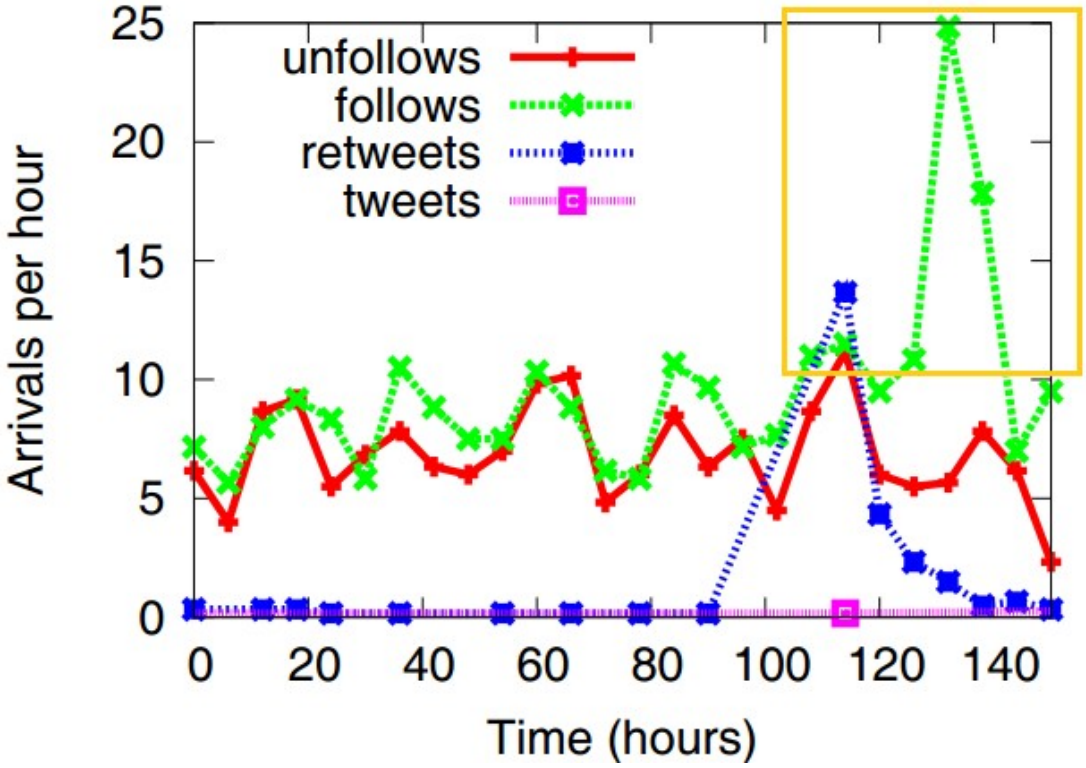
2.2 Έρευνα

Τα τελευταία χρόνια έχει γίνει αντικείμενο μελέτης τόσο ο τρόπος διάδοσης της πληροφορίας στο *Twitter* μέσω της δημοσίευσης και της αναδημοσίευσης περιεχομένου από τους χρήστες, όσο και η αλλαγή του υποκείμενου κοινωνικού δικτύου μέσω της δημιουργίας και της καταστροφής σχέσεων μεταξύ των χρηστών. Μεγάλο ενδιαφέρον παρουσιάζει η σύνδεση που υπάρχει μεταξύ αυτών των δυο δυναμικών. Δηλαδή πως επηρεάζει η διάδοση της πληροφορίας την δομή του ίδιου του δικτύου.

Οι *Myers* και *Leskovec* μελετούν στην εργασία τους [*ML*] τον τρόπο που η δομή του δικτύου του *Twitter* αντιδρά καθώς οι χρήστες δημοσιεύουν και αναδημοσιεύουν περιεχόμενο. Όταν οι χρήστες επιλέγουν ποιους χρήστες θα ακολουθήσουν, επιλέγουν εμμέσως και σε ποιες πληροφορίες θα έχουν πρόσβαση. Συνεπώς η εξέλιξη της δομής του δικτύου δεν είναι ανεξάρτητη από την διάχυση της πληροφορίας σε αυτό.

Οι *Myers* και *Leskovec* παρατήρησαν ότι η δομή του δικτύου του *Twitter* αλλάζει με έναν

σταθερό ρυθμό, ο οποίος διακόπτεται από απότομα ξεσπάσματα που οφείλονται στη διάχυση της πληροφορίας σε αυτό. Πρώτον, οι χρήστες σταματάνε να ακολουθούν κάποιον χρήστη όταν ο τελευταίος δημοσιεύει τιτιβίσματα με πολύ υψηλό ρυθμό. Αυτό ονομάζεται ξέσπασμα "*tweet-unfollow*". Δεύτερον, αν τα τιτιβίσματα ενός χρήστη αναδημοσιεύονται (*retweet*) συχνά από τους ακόλουθους του, τότε είναι πιθανόν αυτός ο χρήστης να κερδίσει περισσότερους ακόλουθους. Καθώς το τιτίβισμα διαχέεται στο δίκτυο μέσω των αναδημοσιεύσεων, όλο και περισσότεροι χρήστες έρχονται σε επαφή με το αρχικό τιτίβισμα. Αν αυτοί οι χρήστες το θεωρήσουν ενδιαφέρον είναι πιθανό να ακολουθήσουν τον χρήστη που το δημιούργησε. Αυτό ονομάζεται ξέσπασμα "*retweet-follow*". Στο σχήμα 2.1 φαίνεται ένα τέτοιο ξέσπασμα γύρω στην ώρα 110. Αρχικά ο χρήστης δέχεται έναν μεγάλο αριθμό αναδημοσιεύσεων, ακολουθούμενο από μεγάλη αύξηση στον αριθμό των ακολούθων του. Παρατηρούμε επίσης ότι πριν την ώρα 110 ο χρήστης χάνει και κερδίζει συνεχώς ακόλουθους, χωρίς να έχει δημοσιεύσει κάποιο περιεχόμενο. Φαίνεται δηλαδή έντονα ο τρόπος που το ξέσπασμα "*retweet-follow*" διακόπτει τον σταθερό ρυθμό αλλαγής του δικτύου.



Σχήμα *2.1 [ML]* - Ένα ξέσπασμα *"retweet-follow"* για έναν χρήστη με *266,844* ακόλουθους

Αφού πρότειναν μια μέθοδο για την αναγνώριση τέτοιων ξεσπασμάτων, που βασίζεται στη διαφορά των πραγματικών με των προσδοκώμενων ακολούθων, οι *Myers* και *Leskovec* προσπάθησαν να εξετάσουν την αλλαγή στη συνεκτικότητα του δικτύου μετά από τέτοια ξεσπάσματα. Και για τους δύο τύπους ξεσπασμάτων παρατηρήθηκε μια μεγάλη αύξηση στην ομοιότητα των χρηστών με τον αρχικό χρήστη, και συνεπώς αύξηση στην ομοιογένεια και συνεκτικότητα του δικτύου στη γειτονία αυτού του χρήστη. Για την μέτρηση της ομοιότητας των χρηστών, μετρήθηκε η ομοιότητα του

κειμένου των τιτιβισμάτων τους. Η ιδέα είναι ότι όσο πιο πολύ μοιάζουν δυο χρήστες μεταξύ τους, τόσο πιο όμοια θα είναι και η πληροφορία η οποία θα διαδίδεται από αυτούς. Τα ευρήματα δείχνουν ότι καινούργιοι ακόλουθοι που αποκτούνται κατά την διάρκεια ενός ξεσπάσματος "*retweet-follow*" έχουν 76.6% μεγαλύτερη ομοιότητα με τον αρχικό χρήστη, από τους ακόλουθους που αποκτήθηκαν χωρίς να εκτεθούν σε κάποια αναδημοσίευση. Όμοια, στα ξεσπάσματα "*tweet-unfollow*" η ομοιότητα ενός χρήστη, που σταματάει να ακολουθεί, με τον αρχικό χρήστη είναι 36.1% μικρότερη σε σχέση με τους χρήστες που συνεχίζουν να τον ακολουθούν. Εδώ η αύξηση της ομοιογένειας πετυχαίνεται λόγω του γεγονότος ότι οι πιο ανόμοιοι χρήστες με τον αρχικό χρήστη σταματάνε να τον ακολουθούν.

Ένα άλλο ενδιαφέρον στοιχείο που παρατηρήθηκε, είναι ότι μετά από τέτοιου είδους ξεσπάσματα, η γειτονιά του αρχικού χρήστη γίνεται πιο πυκνή. Δηλαδή ξεκινάνε να δημιουργούνται δεσμοί μεταξύ των ακολούθων αυτού του χρήστη. Αυτά τα ξεσπάσματα μπορούν να παρομοιαστούν με μια δόνηση για τη δομή του δικτύου στην γειτονιά του αρχικού χρήστη. Για μέρες μετά από αυτήν την δόνηση, το δίκτυο συνεχίζει να γίνεται όλο και πιο πυκνό, καθώς δημιουργούνται συνεχώς καινούργιοι δεσμοί.

Τελικά οι *Myers* και *Leskovec* εστίασαν στα "*retweet-follow*" ξεσπάσματα, και προσπάθησαν να προβλέψουν την πιθανότητα ένα ξέσπασμα αναδημοσιεύσεων να οδηγήσει στην δημιουργία καινούργιων σχέσεων. Αρχικά, επικεντρώνονται στους γείτονες δύο-επιπέδων (*two-hop neighbors*) στους οποίους φτάνει (μέσω αναδημοσιεύσεων) το αρχικό τιτίβισμα, και οι οποίοι δεν έχουν ξανάρθει σε επαφή με κάποιο τιτίβισμα του αρχικού χρήστη. Αν η μεταπήδηση αυτών των χρηστών στην γειτονία του αρχικού χρήστη (δηλαδή από γείτονες δύο-επιπέδων γίνουν γείτονες ενός-επιπέδου) αυξάνει την ομοιογένεια του δικτύου τότε υπάρχουν αρκετές πιθανότητες να οδηγηθούμε σε ξέσπασμα "*retweet-follow*". Η ομοιογένεια του δικτύου υπολογίζεται, όπως και προηγουμένως, σαν την ομοιότητα των τιτιβισμάτων των χρηστών.

Ένα άλλο αντικείμενο έρευνας αφορά την γεωγραφική πληροφορία που δημιουργείται στο *Twitter*. Λόγω των κινήτρων που δίνονται στους χρήστες να μοιράζονται την ακριβή τους τοποθεσία, έχει παρατηρηθεί πολλοί χρήστες να προσποιούνται ότι βρίσκονται σε κάποιο μέρος δημιουργώντας ψευδή πληροφορία.

Στην εργασία τους, οι Παπαλεξάκης, Πελεχρίνης και Φαλούτσος [*PPF*] προσπαθούν να εντοπίσουν τέτοιες συμπεριφορές αναπτύσσοντας μια μέθοδο που βασίζεται στην ανάλυση τανυστών (*tensor decomposition*). Ένας τανυστής ν-τάξης είναι η γενίκευση ενός πίνακα (τανυστής 2ης τάξης) στις ν διαστάσεις. Στη προσέγγιση τους, χρησιμοποιήθηκε ένα σύνολο τανυστών τριών διαστάσεων (χρήστης, χώρος, χρόνος) για τη μοντελοποίηση 6,7 εκατομμυρίων τιτιβισμάτων σε 460 χιλιάδες τοποθεσίες. Στη συνέχεια αυτό το σύνολο διασπάστηκε σε τρία διανύσματα (τανυστές 1ης τάξης). Με αυτό τον τρόπο κατάφεραν να συνοψίσουν τα δεδομένα σε μια αναπαράσταση μικρότερης τάξης.

Σε αυτό το σημείο μπορούν εύκολα να εντοπιστούν ανωμαλίες απλά εξετάζοντας την συνιστώσα του χρόνου. Μια περιοδική συμπεριφορά, ή μια απότομη έκρηξη (*burst*) δημιουργούν έντονη υποψία ότι ο συγκεκριμένος χρήστης παράγει ψευδείς πληροφορίες όσον αφορά την γεωγραφική του θέση.

2.3 Εφαρμογές

Το *Twitter* όμως, πέρα από αντικείμενο έρευνας τόσο ως κοινωνικό δίκτυο όσο και ως δίκτυο πληροφοριών, αποτελεί σήμερα την βάση για την ανάπτυξη πολλών εφαρμογών. Συνεπώς η αξιοποίηση του στα πλαίσια αυτών των εφαρμογών μπορεί να φανεί ιδιαίτερα χρήσιμη.

Λόγω της ευκολίας που οι πληροφορίες εξαπλώνονται, μια σημαντική πτυχή του *Twitter* αφορά την προστασία της ιδιωτικής ζωής των χρηστών του. Πολλοί χρήστες, λόγω άγνοιας, δημοσιεύουν περιεχόμενο που μπορεί να βάλει σε κίνδυνο ακόμα και την δουλειά τους. Για τον σκοπό αυτό οι *Kawase*, *Siehel* και *Herder* [*KSH*] προσπάθησαν να ερμηνεύσουν τα χαρακτηριστικά αυτών των χρηστών ώστε να δημιουργήσουν ένα πλαίσιο, το "*FireMe!*" (fireme.l3s.uni-hannover.de/fireme.php), που ενημερώνει τους χρήστες για το αν έχουν δημοσιεύσει παράπονα ή αρνητικά σχόλια σχετικά με την δουλειά ή το αφεντικό τους. Τα αποτελέσματα της έρευνας έδειξαν ότι τέτοιοι χρήστες έχουν λιγότερους φίλους και ακόλουθους (*followers*), σε σχέση με τους χρήστες που εκφράζονται θετικά για την δουλειά τους. Επίσης είναι πολύ λιγότεροι αλλά και πολύ πιο ενεργοί, όσον αφορά την συχνότητα που δημοσιεύουν περιεχόμενο.

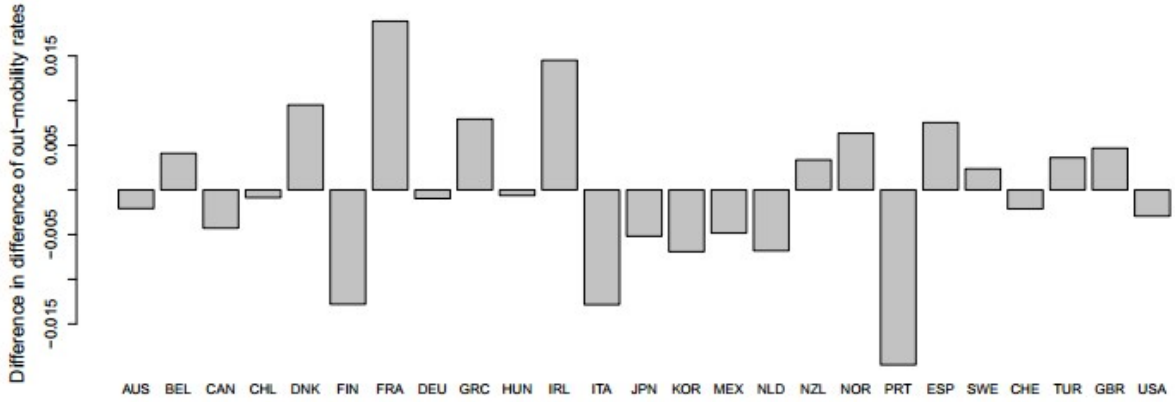
Άλλη μια ενδιαφέρουσα εφαρμογή είναι ο υπολογισμός των μεταναστευτικών ροών, βασιζόμενοι στην γεωγραφική πληροφορία που δημιουργείται στο *Twitter*. Αν και οι μεταναστευτικές ροές αποτελούν μια βασική πηγή δημογραφικής αλλαγής, τα δεδομένα που έχουμε για αυτές προέρχονται κυρίως από απογραφές και συνεπώς τις περισσότερες φορές είναι ξεπερασμένα και ανεπαρκή. Ειδικά όταν μιλάμε για διεθνής μεταναστευτικές ροές, υπάρχει μεγάλη ασυμφωνία ανάμεσα στα δεδομένα των απογραφών των δύο χωρών (χώρα-αφετηρία και χώρα-προορισμός). Ακόμα και σε περιπτώσεις που οι ασυμφωνίες είναι μικρές και μπορούν να επιλυθούν, περνάνε αρκετά χρόνια μέχρι τα δεδομένα των απογραφών να γίνουν διαθέσιμα. Αυτή η καθυστέρηση δημιουργεί μεγάλο πρόβλημα στην προσπάθεια μας να προβλέψουμε αυτές τις ροές, καθώς οι προβλέψεις δεν λαμβάνουν υπόψιν τους τις σύγχρονες τάσεις, και αυτό μας οδηγεί σε μεγάλα σφάλματα.

Οι *Zagheni*, *Garimella*, *Weber* και *State* μελετούν στην εργασία τους [*ZGWS*] τη διεθνή και εσωτερική κινητικότητα στις χώρες του ΟΟΣΑ (Οργανισμός Οικονομικής Συνεργασίας και Ανάπτυξης). Αφού συνέλεξαν γεωγραφικά δεδομένα για 2 χρόνια από 500 χιλιάδες χρήστες, τα χώρισαν σε έξι τετράμηνα. Από αυτούς κρατήσανε μόνο όσους είχαν κάνει τουλάχιστον 3 τιτιβίσματα με γεωγραφικό περιεχόμενο σε κάθε ένα από τα έξι τετράμηνα, μειώνοντας έτσι το δείγμα στους 15 χιλιάδες χρήστες. Σε κάθε μία από αυτές τις έξι περιόδους, υπολογίστηκε η χώρα κατοικίας κάθε χρήστη, ως αυτή από την οποία προέρχονται τα περισσότερα του τιτιβίσματα. Αν τα τιτιβίσματα από δύο χώρες είναι σχεδόν ίσα τότε δεν θεωρήθηκε κάποια από τις δύο ως χώρα κατοικίας του. Αν η χώρα κατοικίας ενός χρήστη αλλάξει από την μια περίοδο στην επόμενη, τότε θεωρούμε πως ο χρήστης μετανάστευσε.

Για τον υπολογισμό των μεταναστευτικών ροών οι *Zagheni*, *Garimella*, *Weber* και *State* χρησιμοποίησαν τις διαφορές δεύτερης τάξης ανάμεσα στην προς τα έξω μετανάστευση μιας χώρας με τον μέσο όρο της προς τα έξω μετανάστευσης όλων των χωρών, ανάμεσα

σε δύο διαδοχικές χρονικές περιόδους. Με αυτόν τον τρόπο αποφεύγουμε να κάνουμε στατιστική αναφορά σε μια συγκεκριμένη περίοδο, παρά μόνο να συγκρίνουμε σχετικές αλλαγές στην ροή ανάμεσα σε μια χώρα και τον μέσο όρο των χωρών. Έτσι για παράδειγμα αν το ποσοστό των χρηστών του *Twitter* που είναι 25 χρονών είναι μεγαλύτερο από το ποσοστό του πληθυσμού που είναι 25 χρονών, τότε οι εκτιμήσεις των μεταναστευτικών ροών με βάση τα δεδομένα του *Twitter* θα τείνουν να υπερεκτιμούν αυτές τις ροές. Αυτό θα συμβαίνει επειδή οι άνθρωποι αυτής της ηλικίας μεταναστεύουν συχνότερα. Αν όμως σε όλες τις χώρες το ποσοστό των χρηστών του *Twitter* που είναι 25 χρονών είναι μεγαλύτερο από το ποσοστό του πληθυσμού που είναι 25 χρονών, τότε αποφεύγουμε την προκατάληψη που οφείλεται στην επιλογή του δείγματος (*selection bias*) καθώς αναφερόμαστε σε σχετικά νούμερα.

Μπορούμε να εργαστούμε αντίστοιχα για τον υπολογισμό των εσωτερικών μεταναστευτικών ροών. Εδώ, εκμεταλλευόμενοι το γεγονός ότι τα γεωγραφικά δεδομένα που παίρνουμε από το *Twitter* έχουν τεράστια ακρίβεια, μπορούμε να υπολογίσουμε τον μέσο όρο της απόστασης των δεδομένων από το βαρύκεντρο τους, για κάθε χρήστη που δεν άλλαξε χώρα κατοικίας.



Σχήμα *2.2 [ZGWS]* – Οι δείκτες προς τα έξω μετανάστευσης για τις χώρες του ΟΟΣΑ

Στην εικόνα 2.2, φαίνονται οι προς τα έξω μεταναστευτικοί δείκτες για όσες χώρες του ΟΟΣΑ υπήρχε δείγμα τουλάχιστον 100 χρηστών για κάθε ένα από τα έξι προς εξέταση τετράμηνα. Θετικές τιμές υποδεικνύουν αύξηση στην μετανάστευση από την συγκεκριμένη χώρα σε μια άλλη χώρα του ΟΟΣΑ. Παρατηρούμε ότι στο Μεξικό υπάρχει μια μείωση στην προς τα έξω μετανάστευση. Η παρατήρηση αυτή είναι συνεπής με τις εκτιμήσεις του ερευνητικού κέντρου του *Pew*. Αυτού του είδους η πληροφορία θα είναι διαθέσιμη στις επίσημες στατιστικές μόνο μετά από αρκετά χρόνια. Συνεπώς όλες οι προβλέψεις για την μετανάστευση στο Μεξικό θα τείνουν να υπερεκτιμήσουν τις μεταναστευτικές ροές. Να σημειώσουμε εδώ ότι δεν υπάρχουν μέχρι στιγμής επίσημα στοιχεία για να επαληθεύσουν ή όχι τα αποτελέσματα της έρευνας.

2.4 Twitter APIs

Είδαμε λοιπόν κάποιες εφαρμογές όπου η αξιοποίηση των δεδομένων του *Twitter* μπορεί να φανεί πολύ χρήσιμη έως και κερδοφόρα. Για να μπορέσουμε να αξιοποιήσουμε αυτή την πληροφορία πρέπει οι εφαρμογές μας να αποκτήσουν με κάποιον τρόπο πρόσβαση σε αυτά τα δεδομένα.

Για αυτόν τον σκοπό, το *Twitter* παρέχει στους προγραμματιστές 4 διεπαφές προγραμματισμού εφαρμογών (*Application Programming Interfaces* ή πιο γνωστές ως *APIs*), ώστε οι εφαρμογές τους να μπορούν να επικοινωνούν με το *Twitter*. Οι εφαρμογές αυτές μπορούν να αλληλεπιδρούν με το *Twitter* έχοντας είτε δικαιώματα μόνο για ανάγνωση (*read only permission*) είτε δικαιώματα ανάγνωσης/εγγραφής (*read/write permission*). Στην πρώτη περίπτωση μπορούμε απλά να διαβάσουμε δεδομένα από το *Twitter*, ενώ στην δεύτερη μπορούμε επιπλέον να κάνουμε και αλλαγές στο προφίλ μας. Στα πλαίσια της εφαρμογής μας, τα δικαιώματα μόνο για ανάγνωση είναι αρκετά, καθώς μας ενδιαφέρει μόνο να παίρνουμε δεδομένα από το *Twitter*.

Για να μπορέσουμε να δημιουργήσουμε μια εφαρμογή που να επικοινωνεί με το *Twitter* πρέπει αρχικά να έχουμε έναν λογαριασμό στο *Twitter*. Στην συνέχεια πηγαίνουμε στην σελίδα dev.twitter.com δηλώνουμε την εφαρμογή μας και ορίζουμε τα δικαιώματα που θέλουμε να έχει. Στην συνέχεια λαμβάνουμε από το *Twitter* κάποια διαπιστευτήρια. Μέχρι τις 31 Αυγούστου 2010 τα διαπιστευτήρια αυτά ήταν το όνομα και ο κωδικός εισόδου. Έκτοτε το *Twitter* χρησιμοποιεί το *OAuth*, μια μέθοδο πιστοποίησης που δεν απαιτεί από τον χρήστη να δώσει τον κωδικό του στην εφαρμογή. Θα δούμε αναλυτικά την μέθοδο πιστοποίησης *OAuth* στην τελευταία παράγραφο της παρούσας ενότητας.

Στην συνέχεια της παρούσας ενότητας θα δούμε τα βασικά χαρακτηριστικά της καθεμιάς από τις 4 διεπαφές εφαρμογών (*APIs*) του *Twitter*.

2.4.1 Twitter for Websites

Η συγκεκριμένη διεπαφή παρέχει μια σειρά από υπηρεσίες που προσφέρουν την δυνατότητα σε ιστοσελίδες να χρησιμοποιούν κάποιες πολύ βασικές λειτουργίες του *Twitter*. Χαρακτηριστικά παραδείγματα είναι τα κουμπιά "*tweet*" και "*follow*" που βλέπουμε πολλές φορές σε διάφορες ιστοσελίδες. Η συγκεκριμένη διεπαφή δεν θα μας απασχολήσει καθόλου στην παρούσα διπλωματική.

2.4.2 Search API

Το *Search API* μας δίνει την δυνατότητα να υποβάλουμε στο *Twitter* ένα ερώτημα για κάποιο περιεχόμενο που μας ενδιαφέρει. Μπορούμε δηλαδή να αναζητήσουμε κάποιο τιτίβισμα με βάση κάποια λέξη ή φράσεις κλειδιά, με βάση κάποιο *hashtag*, τον χρήστη

που το δημοσίευσε, τους χρήστες που αναφέρονται σε αυτό, τη γλώσσα στην οποία είναι γραμμένο, τη γεωγραφική θέση και άλλα. Να σημειώσουμε ότι η συγκεκριμένη διεπαφή είναι μόνο για ανάγνωση (*read only*). Δεν έχουμε δηλαδή την δικαιοδοσία να δημιουργήσουμε, να ενημερώσουμε ή να διαγράψουμε τιτιβίσματα μέσω του *Search API*.

Ένας σημαντικός περιορισμός της συγκεκριμένης διεπαφής είναι το όριο χρήσης (*rate limit*). Με άλλα λόγια δεν μπορούμε να υποβάλουμε στο *Twitter* περισσότερα από 180 ερωτήματα ανά 15 λεπτά. Αυτός ο περιορισμός είναι ανά χρήστη και διαπιστευτήριο.

2.4.3 REST API

Το *REST API* μας δίνει την δυνατότητα να χρησιμοποιήσουμε όλες τις βασικές λειτουργίες που μπορεί να κάνει κανείς στο *Twitter*. Μπορούμε να δημοσιεύσουμε ή να αναδημοσιεύσουμε (*retweet*) κάποιο μήνυμα και να ακολουθήσουμε (*follow*) ή να πάψουμε να ακολουθούμε (*unfollow*) κάποιον χρήστη. Το *REST API* μας δίνει ακόμα περισσότερες δυνατότητες. Μπορούμε να πάρουμε το χρονολόγιο (*timeline*) ˙ κάτι σαν το ιστορικό των τιτιβισμάτων ενός χρήστη, ή ακόμα και να βρούμε την λίστα των χρηστών που ακολουθούν ή ακολουθούνται από έναν χρήστη. Είναι ιδανικό δηλαδή για να χτίσουμε το προφίλ ενός χρήστη.

Παρά τις μεγάλες του δυνατότητες υπόκειται και αυτό σε περιορισμό ορίου χρήσης (*rate limit*). Δεν μπορεί δηλαδή ένας χρήστης ή ένα πιστοποιητικό να θέσει στο *Twitter* περισσότερες από 180 ερωτήσεις ανά 15 λεπτά. Για κάποια ερωτήματα, όπως η λίστα με τους ακόλουθους (*followers*) ενός χρήστη, ο αριθμός των επιτρεπτών ερωτήσεων μειώνεται στις 15 ανά 15 λεπτά.

2.4.4 Streaming API

Το *Streaming API* απευθύνεται σε εφαρμογές που ενδιαφέρονται για μεγάλο όγκο δεδομένων σε πραγματικό χρόνο. Ουσιαστικά μας δίνει ένα πραγματικού χρόνου (*real time*) δείγμα της τάξης του 1% του δημοσίου χρονολογίου (*public timeline*). Το δημόσιο χρονολόγιο είναι το σύνολο των τιτιβισμάτων που γίνονται στο *Twitter*, γνωστό και ως *Firehose*. Αν και σε πρώτη ανάγνωση το νούμερο μοιάζει μικρό, τα δεδομένα που μας παρέχονται με αυτόν τον τρόπο είναι υπεραρκετά για τα πλαίσια των περισσοτέρων εφαρμογών. Όπως γίνεται κατανοητό από τα παραπάνω, η συγκεκριμένη διεπαφή (όπως και το *Search API*) είναι μόνο για ανάγνωση (*read only*).

Η συγκεκριμένη διεπαφή διαφέρει από τις δύο προηγούμενες στον τρόπο που επικοινωνεί με το *Twitter*. Δημιουργεί αρχικά μια μόνιμη σύνδεση με το *Twitter*, και έκτοτε τραβάει τιτιβίσματα από εκεί για όσο χρόνο η σύνδεση παραμένει ενεργή. Εδώ πρέπει να προσέξουμε τους λόγους για τους οποίους το *Twitter* μπορεί να διακόψει μια σύνδεση. Πρώτον, δεν επιτρέπεται να δημιουργήσουμε δύο συνδέσεις με τα ίδια διαπιστευτήρια. Αυτό θα οδηγήσει την πρώτη σύνδεση σε διακοπή. Δεύτερον, αν ο ρυθμός με τον οποίο λαμβάνουμε τα δεδομένα είναι πολύ αργός πάλι θα οδηγηθούμε σε

διακοπή. Κάθε σύνδεση έχει μια ουρά με τιτιβίσματα τα οποία πρέπει να παρέχει στην εφαρμογή. Αν αυτή η ουρά αυξάνεται πολύ γρήγορα με τον χρόνο, τότε αυτή η σύνδεση θα διακοπεί.

Το *Streaming API* χωρίζεται στο *Sample stream* και στο *Filter stream*. Το *Sample stream* τραβάει τυχαία [*MPL*] τιτιβίσματα από το δημόσιο χρονολόγιο (*public timeline*). Αντίθετα το *Filter stream* αποτελεί ένα υποσύνολο του *Firehose* με βάση κάποια κριτήρια. Τα κριτήρια αυτά μπορεί να είναι μια λέξη ή φράση κλειδί, ο δημιουργός του τιτιβίσματος, ή η τοποθεσία που γίνεται αυτό. Να σημειώσουμε εδώ ότι αν τα τιτιβίσματα που ταιριάζουν στα κριτήρια μας, ξεπερνάνε το 1% όλων των τιτιβισμάτων που γίνονται στο *Twitter*, τότε η απάντηση που λαμβάνουμε περιορίζεται ώστε να μην ξεπερνάει αυτό το όριο.

Τέλος η συγκεκριμένη διεπαφή δεν υπόκειται σε περιορισμούς ορίου χρήσης (*rate limit*), με την μορφή που τους συναντήσαμε στις δύο προηγούμενες διεπαφές. Εδώ ο περιορισμός αφορά το πλήθος των φορών που δοκιμάζουμε να δημιουργήσουμε μια σύνδεση στο *Twitter* με τα ίδια διαπιστευτήρια. Το *Twitter* δεν έχει κάνει δημοσίως γνωστό αυτό το νούμερο, αν και λέγεται ότι υπάρχει ανεκτικότητα της τάξης των μερικών δεκάδων συνδέσεων. Σε περίπτωση που ξεπεράσουμε το όριο, το *Twitter* μας στέλνει μια *HTTP 420* απάντηση σε κάθε αίτημα μας για νέα σύνδεση. Καλό θα είναι εδώ να σταματήσουμε για λίγο να προσπαθούμε να συνδεθούμε, καθώς περαιτέρω δοκιμές μπορεί να οδηγήσουν σε μόνιμη απαγόρευση της εισόδου μας στους εξυπηρετητές (*servers*) του *Twitter*.

2.4.5 Επιλογή API

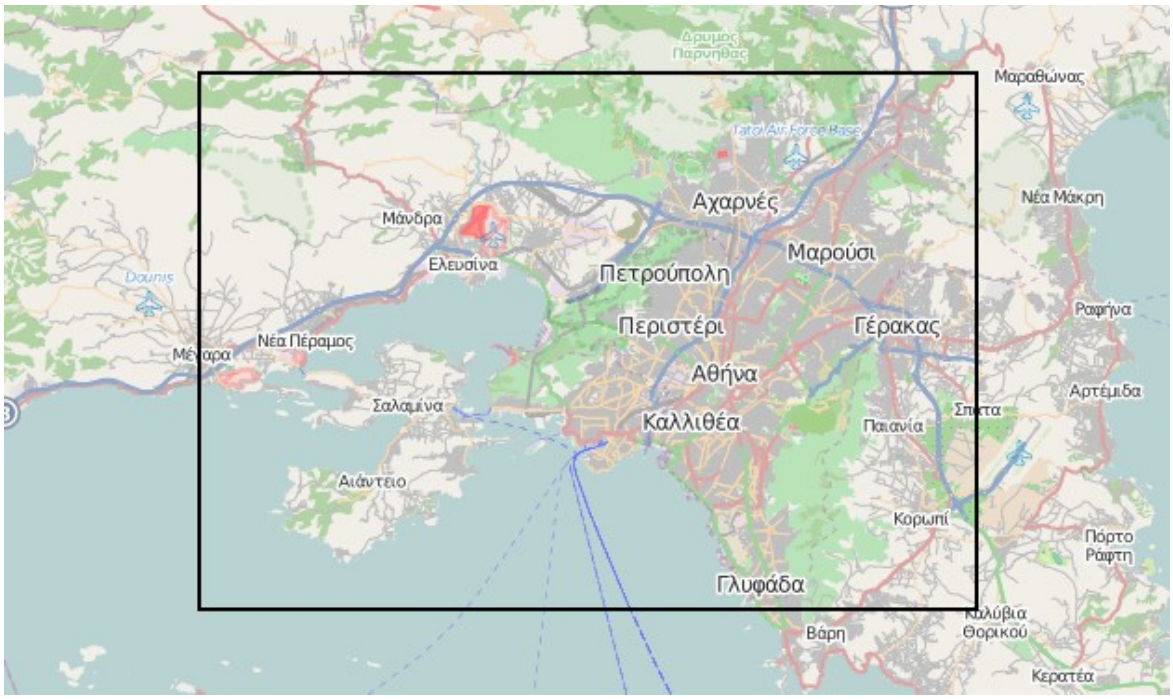
Αφού είδαμε αναλυτικά τις τέσσερις διεπαφές εφαρμογών (*APIs*) που παρέχει το *Twitter*, ήρθε η ώρα να αποφασίσουμε ποια καλύπτει καλύτερα τις ανάγκες της εφαρμογής μας. Η διεπαφή εφαρμογών που θα διαλέξουμε πρέπει να μπορεί να εφοδιάσει την εφαρμογή μας με μεγάλο όγκο δεδομένων πραγματικού χρόνου, που να περιέχουν γεωγραφική πληροφορία.

Αρχικά αποκλείουμε γρήγορα το *Twitter for Websites* καθώς ο σκοπός του δεν είναι να παρέχει δεδομένα του *Twitter* σε εφαρμογές. Από τα υπόλοιπα, το *Search API* και το *REST API* επικοινωνούν με το *Twitter* μέσω ερωτημάτων σε μια βάση δεδομένων. Αυτό δημιουργεί δύο σημαντικούς περιορισμούς. Πρώτον, όλα τα δεδομένα που μαζεύουμε με αυτόν τον τρόπο τοποθετούνται χρονικά στο παρελθόν, δεν είναι δεδομένα πραγματικού χρόνου. Δεύτερον, αυτά τα ερωτήματα απαιτούν την δοσοληψία με μια βάση δεδομένων και συνεπώς είναι χρονοβόρα. Αλλά δεν είναι αυτός ο βασικός λόγος που η συλλογή δεδομένων από το *Twitter* μέσω αυτών των διεπαφών γίνεται με πολύ αργούς ρυθμούς. Τα όρια χρήσης (*rate limits*) που έχουν οι δύο αυτές διεπαφές θέτουν σημαντικά όρια στην ταχύτητα συλλογής δεδομένων από το *Twitter*.

Αντίθετα, το *Streaming API* λειτουργεί διαφορετικά. Δημιουργεί αρχικά μια σύνδεση με το *Twitter* μέσω ενός *HTTP* αιτήματος (*HTTP request*) και στη συνέχεια δεν ξαναστέλνουμε κάτι σε αυτό. Το *Twitter* φροντίζει από μόνο του, να μας στείλει όσα

τιτιβίσματα από το *Firehose* ικανοποιούν τα κριτήρια μας. Με αυτόν τον τρόπο καταφέρνουμε να τροφοδοτήσουμε την εφαρμογή μας με μεγάλο όγκο πραγματικού χρόνου δεδομένα, μιας και είμαστε απαλλαγμένοι από τους δύο βασικούς περιορισμούς των άλλων διεπαφών (δεδομένα παρελθόντος, *rate limits*). Όπως γίνεται εύκολα αντιληπτό για την επικοινωνία μας με το *Twitter* χρησιμοποιήσαμε το *Streaming API*.

Το μόνο κομμάτι πού έμεινε να απαντήσουμε είναι πως θα επικεντρωθούμε σε τιτιβίσματα που περιέχουν γεωγραφική πληροφορία. Η μία επιλογή είναι να χρησιμοποιήσουμε το *Sample stream* του *Streaming API* και από το σύνολο των τιτιβισμάτων που προμηθευόμαστε από αυτό, να κρατάμε μόνο αυτά που περιέχουν τέτοιου είδους πληροφορία. Η δεύτερη επιλογή είναι να χρησιμοποιήσουμε το *Filter stream* και να θέσουμε ως κριτήριο αναζήτησης την γεωγραφική περιοχή που γίνονται τα τιτιβίσματα. Η επιλογή της περιοχής γίνεται μέσω ενός ή περισσοτέρων περιβαλλόντων πλαισίων (*bounding boxes*). Περιβάλλον πλαίσιο στις δύο διαστάσεις είναι μια περιοχή σχήματος ορθογωνίου παραλληλόγραμμου, που ορίζεται από 4 αριθμούς. Οι 2 αντιπροσωπεύουν γεωγραφικά μήκη (*longitudes*) και οι άλλοι 2 γεωγραφικά πλάτη (*latitudes*). Το γεωγραφικό μήκος παίρνει τιμές μεταξύ -180.0 και +180.0, ενώ το γεωγραφικό πλάτος μεταξύ -90.0 και +90.0. Στο σχήμα 2.3 φαίνεται το περιβάλλον πλαίσιο για την ευρύτερη περιοχή της Αθήνας, όπως αυτή χρησιμοποιήθηκε στα πλαίσια της εφαρμογής μας.



Σχήμα 2.3 – Περιβάλλον πλαίσιο για την πόλη της Αθήνας

Τελικά αποφασίσαμε να χρησιμοποιήσουμε το *Filter stream*, καθώς θέλουμε να επικεντρωθούμε σε *tweets* που προέρχονται από 5 γεωγραφικές περιοχές (Αθήνα, Λονδίνο, Άμστερνταμ, Νέα Υόρκη και Αίγυπτος). Αντίθετα το *Sample stream* θα μας έδινε τυχαία *tweets* από όλο τον κόσμο δυσκολεύοντας την προσπάθεια μας να επικεντρωθούμε σε κάποια περιοχή. Να σημειώσουμε ότι μερικές φορές το *Filter Stream* επιστρέφει τιτιβίσματα που βρίσκονται εκτός των δοσμένων ορίων (*bounding boxes*).

Οπότε με την χρήση του *Filter stream* δεν γλιτώνουμε τον έλεγχο για το εάν ένα τιτίβισμα ανήκει σε μια περιοχή ενδιαφέροντος˙ τέτοιος έλεγχος είναι απαραίτητος και με την χρήση του *Sample stream*. Αυτό που κερδίζουμε είναι ότι με το *Filter stream* τα περισσότερα τιτιβίσματα περνάνε αυτόν τον έλεγχο ενώ με το *Sample stream* τα περισσότερα κόβονται.

Το *Filter Stream* μας παρέχει λοιπόν, όλα τα τιτιβίσματα που γίνονται στις γεωγραφικές περιοχές ενδιαφέροντος που ορίσαμε (Αθήνα, Λονδίνο, Άμστερνταμ, Νέα Υόρκη και Αίγυπτος). Εδώ υπάρχει όμως ένας περιορισμός˙ αν ο όγκος αυτών των τιτιβισμάτων για κάποια χρονική στιγμή ξεπεράσει το 1% του δημοσίου χρονολογίου (*public timeline*), τότε η απάντηση μας δειγματοληπτείται (δηλαδή κάποια τιτιβίσματα δεν τα λαμβάνουμε). Αν και ο τρόπος που το *Twitter* δειγματοληπτεί τα δεδομένα του *Filter stream* δεν είναι γνωστός, οι *Morstatter, Pfeffer* και *Liu* στην εργασία τους [*MPL*] έδειξαν ότι υπάρχουν σοβαρές ενδείξεις προκατάληψης στην διαδικασία της δειγματοληψίας. Για παράδειγμα κάποια *hashtags* εμφανίζονται σημαντικά περισσότερο ή λιγότερο στα αποτελέσματα, σε σχέση με την πραγματική τάση στο *Twitter*.

Continent	*Firehose*	*Streaming API*	*Error*
Africa	5.74%	3.10%	-2.64%
Antarctica	0.00%	0.00%	±0.00%
Asia	34.26%	30.11%	-4.15%
Europe	11.03%	13.04%	+2.01%
Mid-Ocean	28.12%	27.67%	-0.45%
N. America	22.32%	27.49%	+5.17%
Oceania	1.98%	1.41%	-0.57%
S. America	0.11%	0.19%	+0.08%

Πίνακας *2.1 [MPLC]* – Τιτιβίσματα με γεωγραφική πληροφορία κατανεμημένα ανά ήπειρο. Συγκρίνοντας το *Firehose* με το *Filter Stream*

Πέρα από τα *hashtags* όμως, προκατάληψη υπάρχει και στην τοποθεσία των τιτιβισμάτων που επιστρέφονται [*MPLC*]. Όπως φαίνεται στον πίνακα 2.1, υπάρχει μια σαφής προτίμηση στα τιτιβίσματα που προέρχονται από τη Β. Αμερική, και αντίστοιχα μια αποστροφή σε αυτά που προέρχονται από την Ασία. Η σύγκριση γίνεται μεταξύ των τιτιβισμάτων που επιστρέφονται από το *Filter stream* του *Streaming API* και του συνόλου των τιτιβισμάτων στο *Twitter* (*Firehose*).

Συνεπώς για να αποφύγουμε σφάλματα που οφείλονται στην προκατάληψη, πρέπει να είμαστε σίγουροι ότι δεν ζητάμε από το *Streaming API* περισσότερα από το 1% των τιτιβισμάτων του δημοσίου χρονολογίου (*public timeline*). Ο αριθμός των τιτιβισμάτων που περιέχουν ακριβή γεωγραφική τοποθεσία δεν ξεπερνάει το 2% του συνόλου των τιτιβισμάτων σε μια τυπική μέρα [*FM, MPLC*]. Αυτό σημαίνει ότι αν μαζεύαμε δεδομένα από όλον τον κόσμο, τα μισά τιτιβίσματα δεν θα μας επιστρεφόντουσαν και ακόμα χειρότερα θα υπήρχε προκατάληψη σε αυτά που τελικά θα μας επιστρεφόντουσαν τόσο ως προς τα *hashtags* όσο και ως προς την περιοχή. Αυτός είναι ένας από τους λόγους που επιλέξαμε να επικεντρωθούμε σε συγκεκριμένες περιοχές αντί να συλλέξουμε δεδομένα από όλο τον κόσμο.

Ο αριθμός των τιτιβισμάτων με γεωγραφική πληροφορία που προέρχονται από τις πέντε γεωγραφικές περιοχές ενδιαφέροντος που ορίσαμε (Αθήνα, Λονδίνο, Άμστερνταμ, Νέα

Υόρκη και Αίγυπτος) αποτελεί το 1.93% του συνολικού αριθμού τιτιβισμάτων με γεωγραφική πληροφορία και συνεπώς είναι πολύ μικρότερος του 1% του συνόλου των τιτιβισμάτων στο *Twitter* (δημόσιο χρονολόγιο). Πιο συγκεκριμένα συλλέξαμε 1,024,504 τιτιβίσματα με γεωγραφική πληροφορία. Από αυτά:

- 294 (0.03%) προέρχονται από την Αθήνα
- 8463 (0.84%) προέρχονται από το Λονδίνο
- 443 (0.04%) προέρχονται από το Άμστερνταμ
- 8031 (0.78%) από την Νέα Υόρκη
- 2638 (0.26%) από την Αίγυπτο

Συνεπώς η συλλογή δεδομένων αποκλειστικά από αυτές τις πέντε περιοχές δεν αντιμετωπίζει προβλήματα προκατάληψης.

Στα πλαίσια της πληρότητας της εφαρμογής μας αποφασίσαμε, μαζί με την συλλογή τιτιβισμάτων πραγματικού χρόνου που συνοδεύονται από γεωγραφική πληροφορία, να δημιουργήσουμε και τον γράφο των ακολούθων (*followers graph*) των χρηστών–δημιουργών αυτών των τιτιβισμάτων. Για τον σκοπό αυτό χρησιμοποιήσαμε το *REST API*, καθώς είναι η μόνη διεπαφή που μπορεί να μας δώσει πληροφορίες για τους ακόλουθους ενός χρήστη. Η χρήση του *REST API* όμως, όπως έχουμε ήδη αναφέρει, θέτει στην εφαρμογή έναν σημαντικό περιορισμό ορίου χρήσης (*rate limit*). Ειδικά για ερωτήματα που αφορούν την εύρεση των ακολούθων ενός συγκεκριμένου χρήστη, αυτό το όριο είναι 15 ερωτήσεις ανά 15 λεπτά.

Αυτές οι 15 ερωτήσεις αφορούν συνήθως λιγότερους από 15 χρήστες. Αυτό οφείλεται στο γεγονός ότι οι απαντήσεις που λαμβάνουμε σε αυτές είναι οργανωμένες σε σελίδες, μέγιστης χωρητικότητας 200 χρηστών (με την πρώτη σελίδα να έχει τους πιο πρόσφατους ακόλουθους και η τελευταία τους παλαιότερους). Μάλιστα η προκαθορισμένη χωρητικότητα είναι 20 χρήστες, οπότε χρειάζεται ειδική μέριμνα ώστε να αυξήσουμε αυτόν τον αριθμό. Για κάθε χρήστη λοιπόν, με περισσότερους από 200 ακόλουθους, "σπαταλάμε" περισσότερα του ενός ερωτήματα για την εύρεση αυτών. Για παράδειγμα, για έναν χρήστη με 1000 ακόλουθους χρειαζόμαστε 5 ερωτήματα.

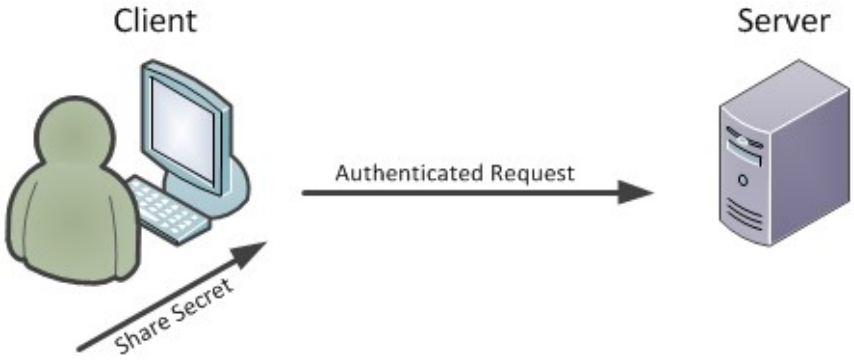
Αυτός ο περιορισμός χρήσης έχει ως συνέπεια ο ρυθμός με τον οποίο σχηματίζεται ο γράφος των ακολούθων (το πολύ 15 χρήστες ανά 15 λεπτά) να είναι πολύ μικρός σε σχέση με τον ρυθμό που λαμβάνουμε τιτιβίσματα (περίπου 500 τιτιβίσματα στο ίδιο χρονικό διάστημα) και συνεπώς που εισάγουμε καινούργιους χρήστες στην εφαρμογή μας. Αποτέλεσμα αυτού, είναι να μην υπάρχει πληροφορία για τους ακόλουθους των περισσότερων χρηστών που υπάρχουν στην εφαρμογή μας.

Η υλοποίηση των δύο διεπαφών (*Streaming API* και *REST API*) είναι ένα πρόβλημα που δεν μας απασχόλησε. Έχουν δημιουργηθεί βιβλιοθήκες που κάνουν αυτή την εργασία. Στα πλαίσια της παρούσας διπλωματικής χρησιμοποιήσαμε την *Twitter4j*, μια *Java* βιβλιοθήκη για την διεπαφή εφαρμογών του *Twitter*.

2.5 OAuth

Το *OAuth* είναι ένα πρωτόκολλο πιστοποίησης, που χρησιμοποιείται και από το *Twitter* προκειμένου να συνδεθεί μια εφαρμογή σε αυτό. Η χρησιμοποίηση του γίνεται για λόγους ασφαλείας, καθώς δεν χρειάζεται πλέον ο χρήστης να μοιραστεί τους κωδικούς του με την εφαρμογή.

Στην παραδοσιακή εκδοχή του μοντέλου εξυπηρετούμενου – εξυπηρετητή (*client – server*), ο εξυπηρετούμενος (*client*) χρησιμοποιεί τα διαπιστευτήρια του χρήστη για να αποκτήσει πρόσβαση σε πόρους που βρίσκονται στον εξυπηρετητή (*server*). Ουσιαστικά στέλνει μέσω ενός αιτήματος τα διαπιστευτήρια στον εξυπηρετητή. Οι πόροι μπορεί να είναι είτε δεδομένα (φωτογραφίες, συμβόλαια) είτε υπηρεσίες (δημοσίευση υλικού, μεταφορά χρημάτων) είτε οτιδήποτε άλλο έχει περιορισμούς πρόσβασης. Τα διαπιστευτήρια είναι ο συνδυασμός ονόματος χρήστη (*username*) και κωδικού πρόσβασης (*password*). Όσον αφορά τον εξυπηρετητή, δεν τον ενδιαφέρει ποιος κάνει το αίτημα ˙ αν το κάνει όντως ο χρήστης ή κάποια άλλη οντότητα, αν τα διαπιστευτήρια ταιριάζουν επιτρέπει την πρόσβαση στους πόρους. Ο εξυπηρετούμενος λοιπόν χρησιμοποιεί τα διαπιστευτήρια του χρήστη, παριστάνοντας ότι είναι αυτός, για να αποκτήσει πρόσβαση στους πόρους. Ο εξυπηρετούμενος είναι συνήθως μια υπηρεσία, στην οποία θέλει ο χρήστης να δώσει πρόσβαση στους πόρους.

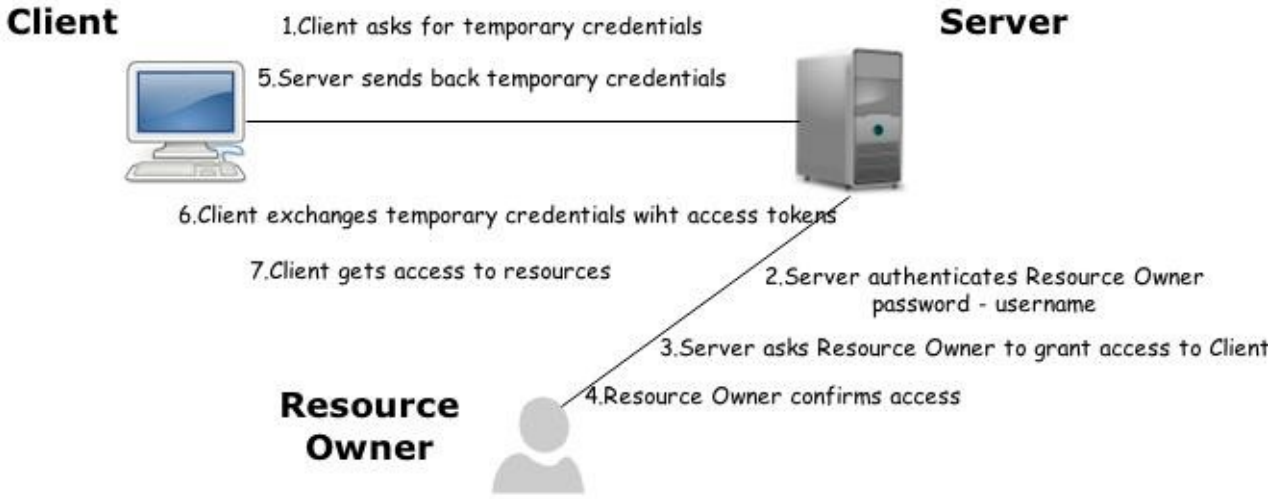


Σχήμα *2.4 [*HOA*]* – Το παραδοσιακό μοντέλο πιστοποίησης εξυπηρετούμενου-εξυπηρετητή

Το πρόβλημα με το παραδοσιακό μοντέλο είναι ότι ο χρήστης μοιράζεται τα διαπιστευτήρια του με την υπηρεσία (εξυπηρετούμενος), δίνοντας της απεριόριστη πρόσβαση να κάνει ότι θέλει - ακόμα και να αλλάξει τους κωδικούς του! Το *OAuth* λύνει αυτό το πρόβλημα, διαχωρίζοντας την έννοια του χρήστη από αυτήν του εξυπηρετούμενου. Για το *OAuth* ο χρήστης είναι ο κάτοχος των πόρων (*resource owner*) και είναι μια ξεχωριστή οντότητα σε κάθε συναλλαγή. Μπορεί ο εξυπηρετούμενος να είναι επίσης και ο κάτοχος των πόρων, όπως στο παραδοσιακό μοντέλο, αλλά γενικά αυτό δεν ισχύει. Το *OAuth* παρέχει μια μέθοδο με την οποία ο κάτοχος των πόρων μπορεί να επιτρέψει σε κάποιον τρίτο (τον εξυπηρετούμενο) μερική (ή σε κάθε περίπτωση ελεγχόμενη) πρόσβαση στους πόρους του χωρίς να μοιραστεί μαζί του τα διαπιστευτήρια του. Η πρόσβαση αυτή είναι περιορισμένη τόσο από άποψη δυνατοτήτων

(πχ. μόνο *read* δικαιώματα), όσο και από άποψη χρόνου (πχ. για 1 ώρα) και μπορεί να ανακληθεί οποιαδήποτε στιγμή θελήσει ο κάτοχος των πόρων.

Το *OAuth* χρησιμοποιεί τρία είδη διαπιστευτηρίων. Τα διαπιστευτήρια του εξυπηρετούμενου (*client credentials*), τα προσωρινά διαπιστευτήρια (*temporary credentials*) και τα τεκμήρια πρόσβασης (*access tokens*).



Σχήμα 2.5 – Το μοντέλο πιστοποίησης *OAuth*

Τα διαπιστευτήρια εξυπηρετουμένου χρησιμοποιούνται για να πιστοποιηθεί ο εξυπηρετούμενος. Επιτρέπουν στον εξυπηρετητή να συλλέξει πληροφορίες για τον εξυπηρετούμενο, ώστε να ενημερώσει τον κάτοχο των πόρων για το ποιος είναι ο εξυπηρετούμενος που ζητάει πρόσβαση στους πόρους του.

Τα τεκμήρια πρόσβασης χρησιμοποιούνται για να αποκτήσει ο εξυπηρετούμενος πρόσβαση στους πόρους. Αντί ο κάτοχος των πόρων να μοιραστεί με τον εξυπηρετούμενο τα διαπιστευτήρια του (συνδυασμό ονόματος χρήστη – κωδικού πρόσβασης), εξουσιοδοτεί τον εξυπηρετητή να εκδώσει μια ειδική κατηγορία διαπιστευτηρίων στον εξυπηρετούμενο. Αυτά τα διαπιστευτήρια είναι τα τεκμήρια πρόσβασης και αντιπροσωπεύουν την χορήγηση πρόσβασης στους πόρους, από τον κάτοχο των πόρων στον εξυπηρετούμενο. Όπως φαίνεται και στο σχήμα 2.5, σε κανένα σημείο δεν μοιράζεται ο κάτοχος των πόρων τους κωδικούς του με τον εξυπηρετούμενο.

3 Ανάλυση Συναισθημάτων

Το κοινωνικό δίκτυο του *Twitter* δίνει καθημερινά την δυνατότητα σε εκατομμύρια χρήστες να μοιράζονται τις απόψεις τους για θέματα της επικαιρότητας. Αυτοί οι χρήστες παράγουν έναν τεράστιο όγκο δεδομένων, κυρίως κειμένου, μέσω των τιτιβισμάτων (*tweets*) τους. Αυτά τα δεδομένα κειμένου δεν είναι απρόσωπα, αλλά αντίθετα φανερώνουν το προσωπικό ύφος, τις ιδέες και τις απόψεις του χρήστη-δημιουργού τους. Αποτελεί λοιπόν, τεράστια πρόκληση η αυτόματη εξαγωγή χρήσιμων πληροφοριών από τα δεδομένα αυτά. Στα πλαίσια αυτά, βρίσκει εφαρμογή η επιστημονική περιοχή της ανάλυσης συναισθημάτων ή ανάλυσης σχολίων (*sentiment analysis*).

3.1 Γενικά

Η ανάλυση συναισθημάτων (*sentiment analysis*) ή αλλιώς εξόρυξη γνώμης-άποψης (*opinion mining*) είναι η επιστημονική περιοχή που έχει ως σκοπό την ανάλυση της άποψης, των συναισθημάτων, των εκτιμήσεων και των στάσεων των ανθρώπων προς διάφορες οντότητες (προϊόντα, υπηρεσίες, οργανισμούς, διάφορα θέματα, άλλους ανθρώπους) και τα χαρακτηριστικά αυτών των οντοτήτων. Από τις αρχές του 2000, ο κλάδος της ανάλυσης συναισθημάτων έχει εξελιχθεί στον πιο ενεργό επιστημονικό κλάδο της επεξεργασίας φυσικής γλώσσας.

Η επεξεργασία φυσικής γλώσσας (*Natural Language Processing – NLP*), αποτελεί την επιστημονική περιοχή της Τεχνητής Νοημοσύνης (*Artificial Intelligence – AI*) που ασχολείται με την διάδραση μεταξύ υπολογιστών και ανθρώπινων γλωσσών. Ουσιαστικά είναι ο κλάδος που μελετά τους τρόπους με τους οποίους ένα πρόγραμμα υπολογιστή μπορεί να καταλάβει την ανθρώπινη φυσική γλώσσα ή αλλιώς το μη δομημένο κείμενο (*unstructured data*). Με τον όρο ανθρώπινη φυσική γλώσσα εννοούμε την σύνθετη και μη δομημένη ανθρώπινη γλώσσα, με τον τρόπο που αυτή ομιλείται ή γράφεται.

Παραδοσιακά η επικοινωνία του ανθρώπου με τον υπολογιστή γίνεται μέσω κάποιας γλώσσας προγραμματισμού, η οποία είναι ακριβής, σαφής και αυστηρά δομημένη. Αντίθετα, η ανθρώπινη γλώσσα είναι εξαιρετικά ασαφής και διφορούμενη, με το νόημα πολλές φορές να εξαρτάται από διάφορες μεταβλητές. Τέτοιες μεταβλητές μπορεί να είναι κάποια λαϊκή έκφραση, το ύφος του ομιλητή και διάφορα σημεία στίξης στην περίπτωση του γραπτού λόγου.

Η ανάλυση συναισθημάτων είναι ένα πρόβλημα της επεξεργασίας φυσικής γλώσσας. Αν και έχει στενή σχέση με αρκετά ανοιχτά προβλήματα αυτής, όπως ο χειρισμός της άρνησης (*negation handling*) και η αποσαφήνιση του νοήματος της λέξης (*word-sense*

disambiguation - WSD), το πεδίο της ανάλυσης συναισθημάτων είναι αρκετά περιορισμένο. Αυτό οφείλεται στο γεγονός ότι στην ανάλυση συναισθημάτων δεν χρειάζεται να κατανοήσουμε πλήρως την σημασιολογία κάθε πρότασης, παρά μόνο κάποιες πτυχές αυτής. Για παράδειγμα, θέλουμε να μάθουμε αν το περιεχόμενο μιας πρότασης είναι συνολικά ή ως προς κάποιο θέμα, θετικό ή αρνητικό.

Υπάρχουν πολλοί λόγοι που έχουν οδηγήσει στην ραγδαία αύξηση της επιστημονικής δραστηριότητας στον κλάδο της ανάλυσης συναισθημάτων τα τελευταία δέκα χρόνια. Αρχικά, οι επιχειρήσεις και οι οργανισμοί πάντα ήθελαν να γνωρίζουν την άποψη του κοινού και των πελατών τους, σχετικά με τα προϊόντα και τις υπηρεσίες που προσφέρουν. Αντίστοιχα, οι καταναλωτές θέλουν να γνωρίζουν την άποψη των χρηστών ενός προϊόντος σχετικά με αυτό το προϊόν πριν το αγοράσουν. Στο παρελθόν, όταν κάποιος καταναλωτής χρειαζόταν κάποια γνώμη έπρεπε να ρωτήσει τους γνωστούς του. Όμοια και οι επιχειρήσεις, διεξήγαγαν έρευνες και δημοσκοπήσεις προκειμένου να μάθουν τα "θέλω" και την άποψη των καταναλωτών, ώστε να τα ανατροφοδοτήσουν στο παραγόμενο προϊόν ή υπηρεσία. Με την εξάπλωση των μέσων κοινωνικής δικτύωσης, οι πληροφορίες αυτές είναι πλέον διαθέσιμες στο διαδίκτυο.

Ο όγκος των δεδομένων που παράγονται στα μέσα κοινωνικής δικτύωσης είναι τόσο μεγάλος όμως, που καθίσταται αδύνατο ένας οργανισμός ή κάποιο άτομο να μπορέσει να αποσπάσει και να συνοψίσει τις απαραίτητες πληροφορίες. Στο σημείο αυτό πρέπει να σημειώσουμε, ότι είναι αναγκαίο να συλλέξουμε και να συνοψίσουμε απόψεις από πολλά άτομα, καθώς η άποψη ενός ή λίγων ατόμων είναι υποκειμενική και όχι αρκετή για τα πλαίσια των περισσοτέρων εφαρμογών. Ειδικά στα πλαίσια των μέσων κοινωνικής δικτύωσης, όπου ο οποιοσδήποτε μπορεί ελεύθερα να δημοσιοποιήσει την άποψη του χωρίς να αποκαλύψει την πραγματική του ταυτότητα, παρατηρείται το φαινόμενο της διάδοσης λανθασμένων πληροφοριών (*opinion spamming*) [*JL*]. Δηλαδή, κάποιος χρήστης έχοντας κρυφούς σκοπούς, δημοσιοποιεί ψεύτικες απόψεις που αφορούν την προώθηση ή την δυσφήμιση επιλεγμένων προϊόντων και υπηρεσιών. Όλα τα παραπάνω καθιστούν αναγκαία την χρησιμοποίηση αυτόματων συστημάτων για ανάλυση συναισθημάτων.

Η ανάλυση συναισθημάτων μελετάται κυρίως σε τρία επίπεδα [*L*] :

1. Επίπεδο εγγράφου (*document level*) : Εδώ μελετάμε εάν ένα ολόκληρο έγγραφο εκφράζει θετικό ή αρνητικό συναίσθημα (*document-level sentiment classification*). Για παράδειγμα δεδομένης μιας κριτικής για ένα προϊόν, αν αυτή η κριτική αναφέρεται στο σύνολο της θετικά ή αρνητικά για το προϊόν. Σε περίπτωση όμως που το έγγραφο εκφράζει άποψη για περισσότερες από μία οντότητες (στο παράδειγμα προϊόντα) τέτοια ανάλυση δεν είναι δυνατή. Δεν μπορεί να εφαρμοστεί σε έγγραφα που συγκρίνουν μεταξύ τους οντότητες.
2. Επίπεδο πρότασης (*sentence level*) : Εδώ μελετάμε σε επίπεδο πρότασης, αν αυτή εκφράζει θετικό, αρνητικό ή ουδέτερο συναίσθημα. Ουδέτερο συναίσθημα σημαίνει απουσία συναισθήματος. Αυτού του επιπέδου την ανάλυση συναισθημάτων θα χρησιμοποιήσουμε για τα τιτιβίσματα (*tweets*) στα πλαίσια της εφαρμογής μας.

3. Επίπεδο οντότητας (*entity and aspect level*) : Σε αντίθεση με τα δύο προηγούμενα επίπεδα ανάλυσης, σε αυτό το επίπεδο η ανάλυση συναισθημάτων αφορά το θέμα ή την οντότητα πάνω στην οποία είναι εκφρασμένη η άποψη. Αντί λοιπόν, να ψάχνουμε γλωσσολογικές κατασκευές (έγγραφο, παράγραφος, πρόταση), πάνω στις οποίες θα γίνει η ανάλυση, αναζητούμε απευθείας το αντικείμενο για το οποίο εκφέρεται η άποψη. Η βάση αυτής της προσέγγισης είναι ότι κάθε άποψη αποτελείται από το συναίσθημα (*sentiment*) που είναι θετικό ή αρνητικό, και το στόχο (*target*) της άποψης. Μια άποψη χωρίς στόχο δεν έχει ιδιαίτερο νόημα, και συνεπώς είναι σημαντικό να αναζητήσουμε αυτόν τον στόχο. Για παράδειγμα η πρόταση "παρόλο που η εξυπηρέτηση δεν είναι τέλεια, αγαπάω αυτό το εστιατόριο", αν αναλυθεί σε επίπεδο πρότασης θα μας δώσει θετικό συναίσθημα με κάποιο βαθμό βεβαιότητας, παρόλο που η πρόταση δεν είναι τελείως θετική. Στην πραγματικότητα, η άποψη είναι θετική για το στόχο εστιατόριο (*entity*), αλλά αρνητική για το στόχο εξυπηρέτηση (*aspect*). Συνεπώς ο σκοπός της ανάλυσης σε αυτό το επίπεδο είναι να βρει τις οντότητες για τις οποίες εκφράζεται το συναίσθημα. Με αυτόν τον τρόπο η ανάλυση συναισθημάτων μετατρέπει το μη δομημένο κείμενο, σε δομημένη πληροφορία της μορφής (*sentiment – target*).

Όπως είναι εύκολα φανερό η ανάλυση συναισθημάτων σε επίπεδο οντότητας είναι μια πιο σύνθετη διαδικασία από τις αναλύσεις στα άλλα δύο επίπεδα. Η ανάλυση συναισθημάτων μπορεί να εμβαθύνει και άλλο, χωρίζοντας τις απόψεις σε κανονικές (*regular*) και συγκριτικές (*comparative*). Οι συγκριτικές απόψεις είναι αυτές που εκφράζουν την άποψη για μια οντότητα σε σύγκριση με μια άλλη οντότητα. Στα πλαίσια της παρούσας διπλωματικής τέτοια θέματα δεν θα μας απασχολήσουν καθώς δεν θα εμβαθύνουμε περισσότερο από την ανάλυση συναισθημάτων σε επίπεδο πρότασης.

Ειδικά για τα δεδομένα του *Twitter* η ανάλυση συναισθημάτων μπορεί να πετύχει πολύ υψηλά επίπεδα ακρίβειας. Το κείμενο σε ένα τιτίβισμα στο *Twitter* είναι μικρό (μέγιστο μέγεθος 140 χαρακτήρες) και γραμμένο σε ανεπίσημο ύφος, καθώς χρησιμοποιούνται σε μεγάλο βαθμό η αργκό του διαδικτύου (διάφορες συντομογραφίες όπως το *brb*, το *lol*, το *wtf* κτλ.) και εκφράσεων συναισθήματος (*emoticons*). Το μικρό τους μέγεθος μαζί με τη χρήση εκφράσεων συναισθήματος κάνει τα κείμενα των τιτιβισμάτων ιδανικά για τέτοιου είδους ανάλυση, καθώς ο χρήστης αναφέρεται συγκεκριμένα σε κάποιο θέμα και πολλές φορές "προδίδει" μέσω των εκφράσεων συναισθήματος το συναίσθημα που διακατέχει το κείμενο.

3.2 AlchemyAPI

Η υλοποίηση της ανάλυσης συναισθημάτων (*sentiment analysis*) στα κείμενα των τιτιβισμάτων (*tweets*) που λαμβάνουμε στα πλαίσια της εφαρμογής μας, έγινε με την χρήση του *AlchemyAPI.*

Το *AlchemyAPI* μας δίνει την δυνατότητα να χρησιμοποιήσουμε πολλά εργαλεία

επεξεργασίας φυσικής γλώσσας (*Natural Language Processing – NLP*), μέσω μιας διεπαφής προγραμματισμού εφαρμογών (*API*). Πρόκειται για μια εμπορική εφαρμογή, η οποία όμως δίνει την δυνατότητα σε μεμονωμένους χρήστες να το χρησιμοποιήσουν κάνοντας αίτηση για ένα κλειδί εγγραφής (*register key*).

Το κλειδί αυτό δίνει στον χρήστη-κάτοχο ένα ανώτατο όριο 1000 κλήσεων ανά ημέρα. Ο αριθμός αυτός είναι αρκετά μικρός, αν αναλογιστούμε ότι στο ίδιο διάστημα (24 ωρών) μπορούμε να λάβουμε υπερπολλαπλάσια τιτιβίσματα. Ένας άλλος περιορισμός αφορά την γλώσσα του κειμένου. Το *AlchemyAPI* υποστηρίζει 8 γλώσσες (αγγλικά, γαλλικά, γερμανικά, ισπανικά, ιταλικά, σουηδικά, ρώσικα και πορτογαλικά). Αυτό σημαίνει ότι δεν υποστηρίζει ολλανδικά, ελληνικά και αιγυπτιακά δηλαδή τρεις από τις πέντε γλώσσες κειμένου της εφαρμογής μας.

Για τη χρήση του *AlchemyAPI* έχουν δημιουργηθεί βιβλιοθήκες για τις δημοφιλέστερες γλώσσες προγραμματισμού (*Python, Java, PHP, C++, Ruby*). Στα πλαίσια της εφαρμογής μας θα χρησιμοποιήσουμε την Java βιβλιοθήκη[1], καθώς σε αυτό το περιβάλλον γίνεται η ανάπτυξη της εφαρμογής που είναι υπεύθυνη για την συλλογή και επεξεργασία των δεδομένων του *Twitter*. Χρησιμοποιώντας την *Java* βιβλιοθήκη η αρχικοποίηση της σύνδεσης με τη διεπαφή γίνεται μέσω της εξής *Java* εντολής :

```
AlchemyAPI alchemy = AlchemyAPI.GetInstanceFromString(ALCHEMY_KEY);
```

Για την υλοποίηση της ανάλυσης συναισθημάτων το *AlchemyAPI* χρησιμοποιεί μεθόδους από την επιστημονική περιοχή της μηχανικής μάθησης (*machine learning*). Πιο συγκεκριμένα, χρησιμοποιεί αλγορίθμους από την σε βάθος μάθηση (*deep learning*) και τα νευρωνικά δίκτυα (*neural networks*), προκειμένου να μπορέσει να κατηγοριοποιήσει το κείμενο (*classification*) ως θετικό ή αρνητικό με κάποιον βαθμό βεβαιότητας. Η ανάλυση συναισθημάτων μπορεί να γίνει και στα τρία επίπεδα ανάλυσης που αναφέρθηκαν στην προηγούμενη ενότητα (επίπεδο εγγράφου, επίπεδο πρότασης, επίπεδο οντότητας), και οι απαντήσεις μπορούν να δωθούν σε *XML, JSON*, ή και *RDF* μορφή.

Στην εφαρμογή μας χρησιμοποιήσαμε ανάλυση συναισθημάτων σε επίπεδο πρότασης με τις απαντήσεις να γίνονται σε μορφή *XML*, η οποία είναι και η προκαθορισμένη. Χρησιμοποιώντας την βιβλιοθήκη *Java*, ένα ερώτημα για ανάλυση συναισθημάτων επιπέδου πρότασης προς την διεπαφή είναι το εξής :

```
Document analysisXML = alchemy.TextGetTextSentiment(analysisText);
```

Το προς ανάλυση κείμενο είναι το κείμενο του τιτιβίσματος χωρίς αυτό να έχει υποστεί προηγουμένως κάποια επεξεργασία. Παρακάτω φαίνεται η απάντηση σε μορφή *XML* που λαμβάνουμε από ένα τέτοιο ερώτημα.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<results>
    <status>OK</status>
    <usage>By accessing AlchemyAPI or using information generated by
           AlchemyAPI, you are agreeing to be bound by the AlchemyAPI Terms
           of Use: http://www.alchemyapi.com/company/terms.html
    </usage>
```

1 github.com/AlchemyAPI/alchemyapi_java

```
    <totalTransactions>1</totalTransactions>
    <language>english</language>
    <docSentiment>
        <score>0.889117</score>
        <type>positive</type>
    </docSentiment>
</results>
```

Από την παραπάνω απάντηση μας ενδιαφέρουν τα εξής πεδία:

1. *status*: το συγκεκριμένο πεδίο μπορεί να πάρει τιμές *OK ή ERROR* και μας ενημερώνει για την ομαλή ή μη εκτέλεση του ερωτήματος.
2. *type*: το συγκεκριμένο πεδίο μπορεί να πάρει τιμές *positive*, *negative ή neutral* και μας ενημερώνει για την πολικότητα του κειμένου σε περίπτωση που η ανάλυση ολοκληρώθηκε με επιτυχία.
3. *score*: το συγκεκριμένο πεδίο μπορεί να πάρει τιμές από το -1 έως και το 1. Μας δίνει ένα βαθμό βεβαιότητας και ακρίβειας για το αποτέλεσμα της ανάλυσης. Τιμές κοντά στο 1 αντιπροσωπεύουν θετικό συναίσθημα ενώ αντίθετα τιμές κοντά στο -1 αρνητικό συναίσθημα. Τιμή 0 σημαίνει ουδέτερο συναίσθημα ή απουσία συναισθήματος.

Να σημειώσουμε εδώ ότι το *Alchemy API* ως εργαλείο επεξεργασίας φυσικής γλώσσας, δεν στοχεύει την ανάλυση του σε κείμενα που προέρχονται αποκλειστικά από το *Twitter*. Οι δημοσιεύσεις των χρηστών του *Twitter* τείνουν να είναι συντακτικά λανθασμένες και χαρακτηρίζονται από το μικρό τους μέγεθος (έως 140 χαρακτήρες), την χρήση συντμήσεων λέξεων και την χρήση εκφράσεων συναισθήματος (*emoticons*). Συνεπώς ένα εργαλείο που απευθύνεται αποκλειστικά σε τέτοιου είδους κείμενα θα μπορούσε να πετύχει πολύ υψηλά επίπεδα ακρίβειας [*GSCDMEHYFS*]. Ο σκοπός της παρούσας εργασίας δεν είναι η ανάπτυξη εργαλείων επεξεργασίας φυσικής γλώσσας, αλλά η εφαρμογή μιας ιδέας όσο το δυνατόν πιο ολοκληρωμένης, για τη διαχείριση των δεδομένων που προέρχονται από το *Twitter*. Στα πλαίσια αυτά, το *Alchemy API* μας παρέχει μια ολοκληρωμένη λύση για την υλοποίηση της ανάλυσης συναισθημάτων στα κείμενα των τιτιβισμάτων.

4 Βάση Δεδομένων

4.1 Εισαγωγή

Στα προηγούμενα κεφάλαια είδαμε πως θα συλλέξουμε και θα αναλύσουμε τα δεδομένα μας από το *Twitter*. Σε αυτό το κεφάλαιο θα δούμε πως θα τα αποθηκεύσουμε κατάλληλα ώστε στην συνέχεια να μπορέσουμε να τα επεξεργαστούμε. Για τον σκοπό αυτό θα χρησιμοποιήσουμε μια βάση δεδομένων. Μια βάση δεδομένων είναι μια συλλογή δεδομένων οργανωμένη με τέτοιο τρόπο ώστε να καθιστά εύκολη την πρόσβαση, την διαχείριση και την ενημέρωση των δεδομένων αυτών. Αν και ο πιο διαδεδομένος τύπος είναι οι σχεσιακές (*relational*) βάσεις δεδομένων, δηλαδή η οργάνωση των δεδομένων σε μορφή πινάκων με τα ξένα κλειδιά να φανερώνουν τις σχέσεις μεταξύ αυτών, εμείς θα ακολουθήσουμε μια διαφορετική προσέγγιση.

Τα τελευταία χρόνια έχει παρατηρηθεί μια ραγδαία αύξηση στη δημοτικότητα μιας οικογένειας τεχνολογιών αποθήκευσης δεδομένων, γνωστών και ως *NoSQL* (ακρωνύμιο του *Not Only SQL*). Αλλά ο όρος *NoSQL* χαρακτηρίζει περισσότερο τι δεν είναι – δεν είναι *SQL*-κεντρικές σχεσιακές βάσεις δεδομένων – παρά τι στην ουσία είναι αυτές οι βάσεις δεδομένων.

Ιστορικά οι περισσότερες διαδικτυακές εφαρμογές τρέχουν πάνω σε σχεσιακές βάσεις δεδομένων. Την τελευταία δεκαετία όμως τα δεδομένα που αντιμετωπίζουμε είναι πολύ μεγαλύτερα σε μέγεθος, αλλάζουν γρηγορότερα, και έχουν μεγάλη ποικιλία στη δομή τους, κάνοντας τα παραδοσιακά *RDBMS* συστήματα δύσκολο να ανταποκριθούν. Αυτές είναι και οι προκλήσεις που αντιμετωπίζουν οι *NoSQL* βάσεις δεδομένων.

Ο χρόνος εκτέλεσης των ερωτημάτων (*queries*) στις σχεσιακές βάσεις δεδομένων αυξάνεται δραματικά, καθώς αυξάνεται το μέγεθος των πινάκων και ο αριθμός των ενώσεων (*joins*) που εκτελούνται. Αυτό δεν είναι λάθος των ίδιων των βάσεων, αλλά του τρόπου που αντιμετωπίζουν τα δεδομένα καθώς για να απαντήσουν σε ένα ερώτημα αρχικά βρίσκουν όλα τα πιθανά αποτελέσματα και στην συνέχεια τα φιλτράρουν. Αντίθετα οι *NoSQL* βάσεις δεδομένων υιοθετούν διαφορετικές προσεγγίσεις για να αντιμετωπίσουν μεγάλο όγκο δεδομένων, προκειμένου να αποφύγουν τις πολλές ενώσεις.

Εκτός από το μεγάλο τους μέγεθος, τα σημερινά δεδομένα συχνά αλλάζουν πολύ γρήγορα. Αυτός ο υψηλός ρυθμός αλλαγής των δεδομένων έχει υψηλό υπολογιστικό κόστος στις σχεσιακές βάσεις δεδομένων, καθώς για να γίνει μια αλλαγή πρέπει να προσπελάσουμε πολλούς πίνακες.

Πέρα όμως από τον υψηλό ρυθμό αλλαγής των ίδιων των δεδομένων, μπορεί να αλλάξει ακόμα και η δομή τους. Με άλλα λόγια, εκτός από την τιμή μιας συγκεκριμένης

ιδιότητας μπορεί να αλλάξει και η ίδια η δομή των στοιχείων που έχουν αυτή την ιδιότητα. Οι σχεσιακές βάσεις δεδομένων έχουν αυστηρά καθορισμένο σχήμα, και απαιτούν τον αυστηρό ορισμό αυτού του σχήματος πριν οποιοδήποτε δεδομένο αποθηκευτεί σε αυτές. Η αλλαγή του σχήματος, αφού μπουν δεδομένα στην βάση, απαιτεί μεγάλο λειτουργικό κόστος και είναι μια διαδικασία που συχνά αποφεύγεται. Αντίθετα οι *NoSQL* βάσεις είναι απελευθερωμένες από αυτόν τον περιορισμό, καθώς δεν έχουν κάποιο προκαθορισμένο σχήμα (είναι *schema-free*). Έτσι επιτρέπουν στους προγραμματιστές εύκολα να ενσωματώνουν νέους τύπους δεδομένων ώστε να εμπλουτίζουν τις εφαρμογές τους.

Οι *NoSQL* βάσεις δεδομένων χωρίζονται σε 4 κατηγορίες με βάση τον τρόπο που μοντελοποιούν τα δεδομένα. Είναι οι *Document Stores*, οι *Key-Value Stores*, οι *Column Family* και οι βάσεις δεδομένων γράφου (*graph databases*).

Η βάση δεδομένων που θα χρησιμοποιήσουμε πρέπει να ταιριάζει με την φύση των δεδομένων μας. Πρώτον, τα δεδομένα μας προέρχονται από ένα κοινωνικό δίκτυο, το *Twitter*. Όπως τα δεδομένα κάθε δικτύου, έτσι και αυτά του *Twitter*, έχουν πυκνές σχέσεις μεταξύ τους και μπορούν εύκολα να αναπαρασταθούν από έναν γράφο. Έτσι μια βάση δεδομένων γράφου ταιριάζει από την φύση της σε αυτόν τον πολύ έντονα δομημένο γύρω από τις σχέσεις (*relationship-centered*) τομέα. Δεύτερον, τα δεδομένα μας περιέχουν έντονη γεωγραφική πληροφορία. Τα χωρικά προβλήματα είναι η αρχική περίπτωση χρήσης των γράφων. Ο *Leonhard Euler* έλυσε το πρόβλημα των επτά γεφυρών του Κόνιγκσμπεργκ (*Seven Bridges of Königsberg*) υποθέτοντας ένα μαθηματικό θεώρημα που έγινε η βάση της θεωρίας των γράφων (*graph theory*). Οι χωρικές εφαρμογές των βάσεων δεδομένων γράφου ποικίλουν από τον υπολογισμό ενός μονοπατιού μέσα σε έναν δίκτυο έως την εύρεση όλων των σημείων ενδιαφέροντος σε μία συγκεκριμένη περιοχή. Έχοντας αυτά κατά νου, επιλέξαμε μια βάση δεδομένων γράφου, καθώς θεωρήσαμε ότι μπορεί να μοντελοποιήσει καλύτερα τον τομέα του προβλήματος μας. Τελικά καταλήξαμε στην *Neo4j* την δημοφιλέστερη βάση δεδομένων γράφου αυτήν τη στιγμή.

4.2 Βάσεις Δεδομένων Γράφου – Neo4j

Αφού είδαμε τους λόγους για τους οποίους επιλέξαμε μια βάση δεδομένων γράφου (*graph database*) για την ανάπτυξη της εφαρμογής μας, στην ενότητα αυτή θα δούμε τα βασικά χαρακτηριστικά των βάσεων δεδομένων αυτού του τύπου, επικεντρωνόμενοι στην *Neo4j*.

Γράφος είναι μια συλλογή κορυφών και ακμών ή πιο απλά ένα σύνολο κόμβων και σχέσεων που ενώνουν αυτούς τους κόμβους. Μια βάση δεδομένων γράφου είναι μια βάση δεδομένων ή οποία χρησιμοποιεί δομές γράφου για να αναπαραστήσει και να αποθηκεύσει δεδομένα. Ένα σύστημα διαχείρισης μιας βάσης δεδομένων γράφου (*Graph Database Management System*) είναι ένα σύστημα διαχείρισης βάσεων δεδομένων με *CREATE*, *READ*, *UPDATE*, *DELETE* (*CRUD*) μεθόδους, οι οποίες εφαρμόζονται πάνω σε δεδομένα τα οποία είναι δομημένα σε μορφή γράφου.

Μια ιδιαιτερότητα των βάσεων δεδομένων γράφου είναι ο τρόπος που μοντελοποιούν το πρόβλημα μας. Μοντελοποίηση είναι η αφαιρετική διαδικασία κατά την οποία περνάμε συγκεκριμένες πτυχές ενός τομέα του πραγματικού κόσμου σε έναν τομέα στον οποίο μπορούν να δομηθούν και να τις χειριστούμε. Ο γράφος είναι ο φυσικός τρόπος με τον οποίο ο άνθρωπος προσπαθεί να αναπαραστήσει ένα πρόβλημα, χρησιμοποιώντας κύκλους και τετράγωνα και στην συνέχεια βελάκια που τα ενώνουν για να δείξει τις σχέσεις μεταξύ αυτών. Με αυτόν τον τρόπο οι βάσεις δεδομένων γράφου μας επιτρέπουν να δημιουργήσουμε εξελιγμένα μοντέλα που ταιριάζουν πολύ καλά με τον τομέα του προβλήματος μας. Αυτά τα μοντέλα είναι απλά και συγχρόνως πολύ πιο εκφραστικά από τα μοντέλα που δημιουργούνται από τις παραδοσιακές σχεσιακές βάσεις δεδομένων ή άλλες *NoSQL* δομές.

Το βασικό χαρακτηριστικό των βάσεων δεδομένων γράφου είναι ότι οι σχέσεις μεταξύ των κόμβων είναι κανονικές οντότητες. Αυτό σημαίνει ότι μπορούμε να τις δούμε απευθείας, απλά κοιτώντας την μνήμη. Αντίθετα σε άλλες τεχνολογίες βάσεων δεδομένων οι σχέσεις μεταξύ των οντοτήτων υπονοούνται, και πρέπει εμείς να τις συμπεράνουμε συχνά χρησιμοποιώντας επινοημένες ιδιότητες όπως το ξένο κλειδί (*foreign key*), ή επεξεργασίες όπως η *map-reduce*.

Ειδικά όταν μιλάμε για δεδομένα πολύ στενά συνδεδεμένα με σχέσεις διαφορετικών τύπων ή για ερωτήματα διάσχισης του γράφου, οι βάσεις αυτές έχουν πολύ υψηλή απόδοση ˙ συνέπεια του υποκείμενου μοντέλου. Αυτό συμβαίνει επειδή τα ερωτήματα εξετάζουν ένα τμήμα του γράφου αντί το σύνολο των δεδομένων και έτσι αποφεύγονται οι πολλές ενώσεις (*joins*) των σχεσιακών βάσεων. Έτσι η καθυστέρηση απόκρισης (*latency*) εξαρτάται από το μέγεθος του τμήματος του γράφου που επιλέγουμε να εξερευνήσουμε και όχι από το όγκο των αποθηκευμένων δεδομένων.

Όπως έχουμε ήδη αναφέρει οι βάσεις δεδομένων γράφου ταιριάζουν εξαιρετικά καλά στον τομέα των κοινωνικών δικτύων. Τα δεδομένα αυτών των δικτύων σχετίζονται στενά και με πολλαπλούς τρόπους μεταξύ τους. Περιμένουμε λοιπόν οι βάσεις δεδομένων γράφου να έχουν πολύ καλύτερη απόδοση από τις σχεσιακές σε ερωτήματα πάνω σε αυτά τα δεδομένα. Έστω λοιπόν ένα κοινωνικό δίκτυο που περιέχει 1 εκατομμύριο ανθρώπους όπου ο καθένας έχει 50 φίλους. Το ερώτημα που θέτουμε είναι, δεδομένου δύο ανθρώπων επιλεγμένων στην τύχη αν υπάρχει μονοπάτι που να τους ενώνει που να είναι το πολύ μήκους 5 σχέσεων. Τα αποτελέσματα φαίνονται στον παρακάτω πίνακα [*PVW*].

Depth	*RDBMS execution time (sec)*	*Neo4j execution time (sec)*	*Records returned*
2	0.016	0.01	~2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

Πίνακας *4.1 [PVW]* – Βρίσκοντας έναν απομακρυσμένο φίλο σε μια σχεσιακή βάση δεδομένων ενάντια στην αποδοτική υλοποίηση στην *Neo4j*

Όπως παρατηρούμε από τον πίνακα 4.1 η *Neo4j* έχει πολύ καλύτερη απόδοση για δεδομένα που είναι έντονα συνδεδεμένα. Για βάθος 2 (φίλοι φίλων) η *Neo4j* τρέχει στα δύο τρίτα του χρόνου που τρέχει η σχεσιακή βάση. Παρόλα αυτά η διαφορά αυτή δεν

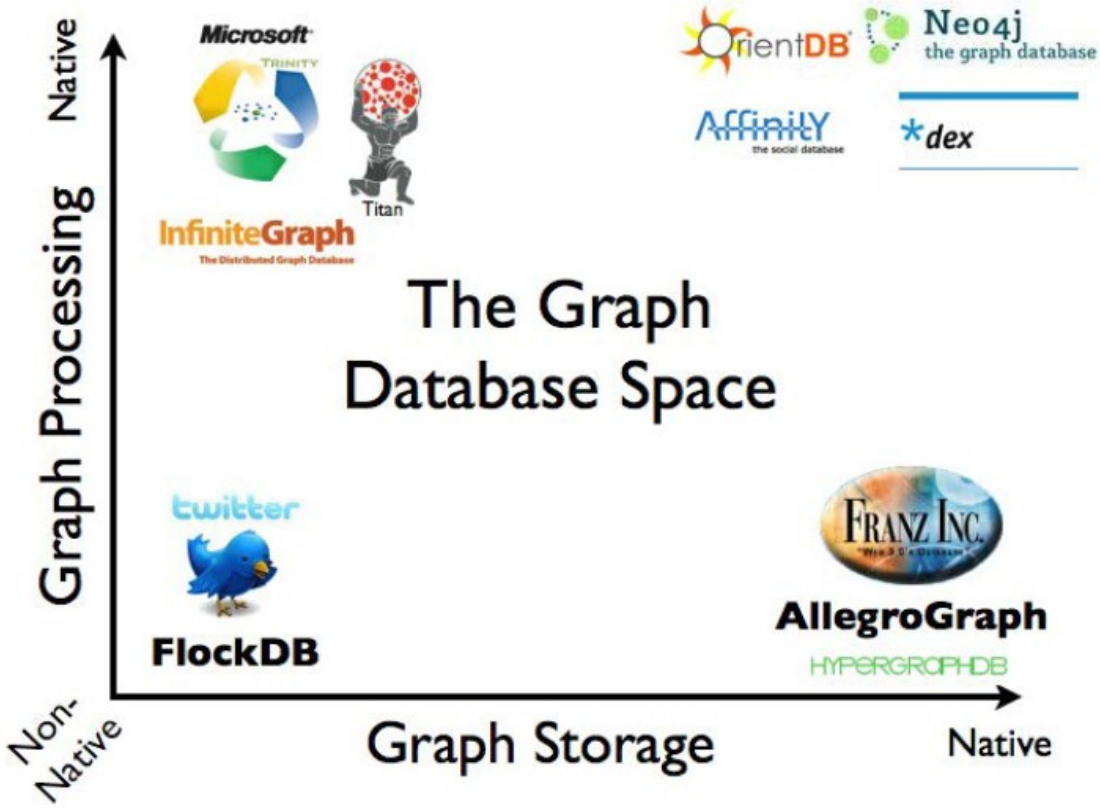
γίνεται αισθητή στο χρήστη. Καθώς μεγαλώνει το βάθος όμως, η σχεσιακή βάση δεδομένων δεν ανταποκρίνεται στο ερώτημα σε χρόνο που να μπορεί να θεωρηθεί αποδεκτός για μια διαδικτυακή εφαρμογή. Αντίθετα η *Neo4j* (αν και οι χρόνοι της χειροτερεύουν λίγο) συνεχίζει να ανταποκρίνεται ικανοποιητικά.

Πέρα από την απόδοση τους, ένα άλλο χαρακτηριστικό των βάσεων δεδομένων γράφου είναι το μη προκαθορισμένο σχήμα τους (*schema-free*). Έτσι, είναι πολύ εύκολο να προσθέσουμε νέου τύπου δεδομένα στην υπάρχουσα δομή, όπως κάποιο νέο είδος κόμβων, σχέσεων και ιδιοτήτων ή ακόμα και κάποιον ολόκληρο υπογράφο χωρίς να πειράξουμε τα υπάρχοντα ερωτήματα και την λειτουργικότητα της εφαρμογής. Αυτό οδηγεί σε αύξηση της ευελιξίας. Λόγω αυτής της ευελιξίας δεν χρειάζεται να μοντελοποιήσουμε από την αρχή πλήρως το πρόβλημα μας, κάτι που στο πλαίσιο των σύγχρονων συνεχώς μεταβαλλόμενων απαιτήσεων είναι και υπερβολικό.

Αφού είδαμε τα βασικά χαρακτηριστικά των βάσεων δεδομένων γράφου, θα επικεντρωθούμε στην *Neo4j*, μιας και αυτήν τελικά θα χρησιμοποιήσουμε στην εφαρμογή μας. Η *Neo4j* είναι μια βάση δεδομένων γράφου με ιδιότητες (*property graph database*). Ως συνέπεια αυτού, τα βασικά χαρακτηριστικά της είναι τα εξής:

- Περιέχει κόμβους, σχέσεις και ιδιότητες (ζευγάρια *"key-value"*).
- Οι κόμβοι αντιπροσωπεύουν τις οντότητες, έχουν ταμπέλες και περιέχουν ιδιότητες. Τα "*keys"* των ιδιοτήτων είναι συμβολοσειρές (*strings)* και τα "*values"* είναι αυθαίρετοι τύποι δεδομένων.
- Οι σχέσεις συνδέουν τους κόμβους. Έχουν πάντα μια κατεύθυνση, μια ταμπέλα, και έναν αρχικό και τελικό κόμβο - δεν υπάρχει αιωρούμενη σχέση.
- Όπως και οι κόμβοι έτσι και οι σχέσεις μπορούν να έχουν ιδιότητες. Αυτή η ικανότητα των σχέσεων να έχουν ιδιότητες είναι πολύ χρήσιμη τόσο σε διάφορους αλγορίθμους που εφαρμόζονται πάνω στους γράφους όσο και στον περιορισμό των ερωτημάτων καθώς αυτά εκτελούνται, καθώς προσθέτει στις σχέσεις σημασιολογία (βάρος, ποιότητα).

Το μοντέλο πάνω στο οποίο είναι δομημένη η *Neo4j* δεν διαφέρει σε πολλά από αυτό των περισσότερων βάσεων δεδομένων. Να σημειώσουμε εδώ ότι υπάρχουν και βάσεις που είναι δομημένες πάνω σε άλλα μοντέλα γράφου όπως οι *Hypergraphs* ή οι δομές *Triple*. Η *Neo4j* έχει επιπλέον δύο χαρακτηριστικά. Χρησιμοποιεί κανονική επεξεργασία γράφου (*native graph processing*) και κανονική αποθήκευση γράφου (*native graph storage*). Το πρώτο αφορά στον τρόπο που διαχειρίζεται τα ερωτήματα ενώ το δεύτερο στον τρόπο που αποθηκεύει την πληροφορία, και θα τα δούμε αναλυτικότερα στην συνέχεια. Όπως φαίνεται και στο σχήμα 4.1 δεν έχουν όλες οι βάσεις δεδομένων γράφου αυτά τα χαρακτηριστικά.



Σχήμα *4.1 [RWE]* – Επισκόπηση του χώρου των βάσεων δεδομένων γράφου

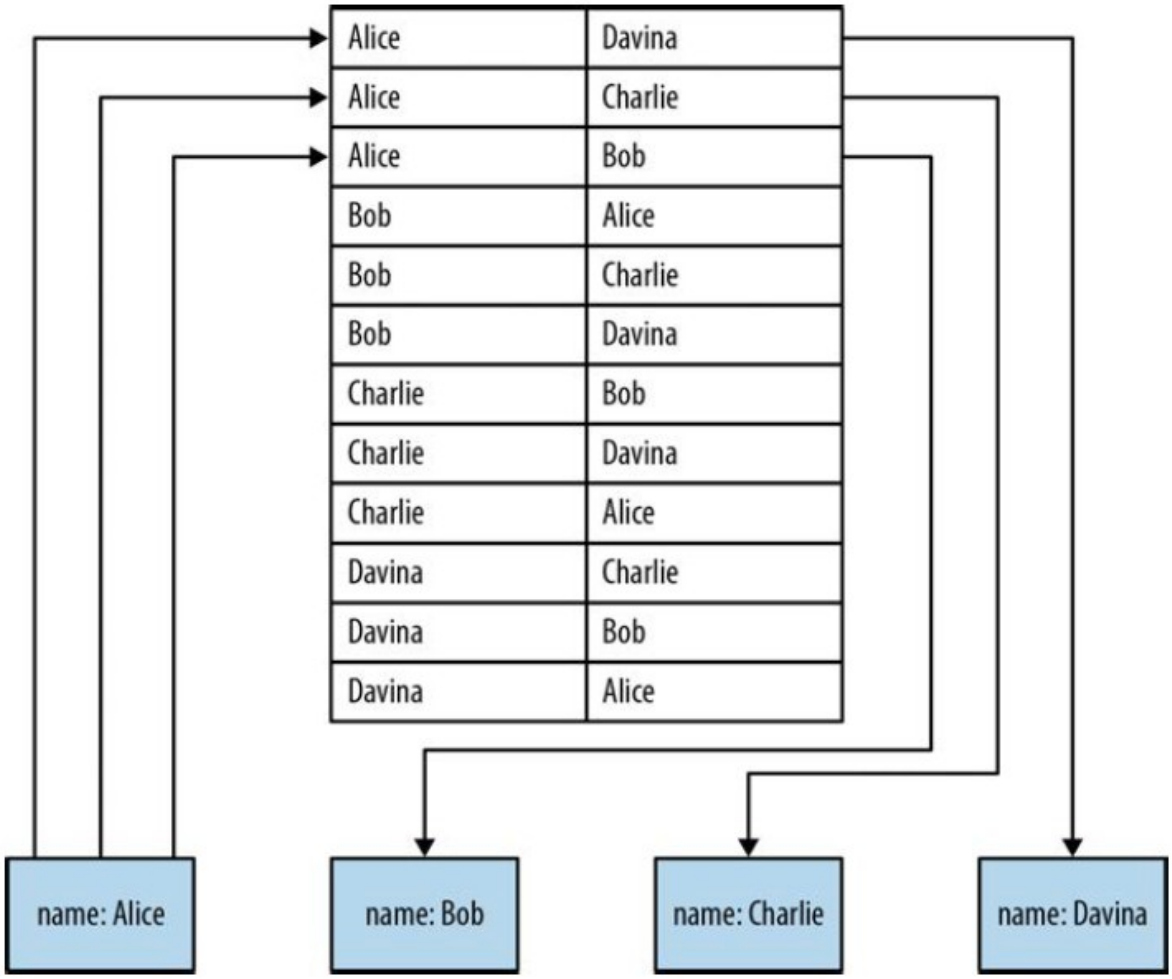
4.2.1 Κανονική Επεξεργασία Γράφου

Μια βάση δεδομένων γράφου λέμε ότι χρησιμοποιεί κανονική επεξεργασία γράφου (*native graph processing*) αν διαθέτει μια ιδιότητα που ονομάζεται γειτνίαση χωρίς ευρετήριο (*index-free adjacency*). Αυτό σημαίνει ότι κάθε κόμβος διατηρεί απευθείας αναφορές σε όλους τους γειτονικούς του κόμβους, και έτσι λειτουργεί σαν ένα μικρό ευρετήριο για την γειτονία του. Έτσι ο χρόνος απόκρισης σε ερωτήματα (*queries*) εξαρτάται αποκλειστικά από το μέγεθος αυτής της γειτονιάς και όχι από το μέγεθος του γράφου.

Αντίθετα, μια βάση δεδομένων που δεν έχει αυτήν την ιδιότητα χρησιμοποιεί ευρετήρια για να για να συνδέσει τους κόμβους μεταξύ τους. Αυτά τα ευρετήρια αυξάνουν το υπολογιστικό κόστος όταν προσπαθούμε να διασχίσουμε τον γράφο. Για να γίνει αυτό καλύτερα κατανοητό ακολουθεί το παρακάτω παράδειγμα.

Έστω λοιπόν ότι έχουμε ένα ευρετήριο για την σχέση "*FOLLOWS*". Αυτό θα μοιάζει με το ευρετήριο του σχήματος 4.2 (με τα μπλε είναι οι κόμβοι). Για να βρούμε ποιους ακολουθεί κάποιο άτομο χρειαζόμαστε ένα κοίταγμα στο ευρετήριο, δηλαδή *O(logn)* σε αλγοριθμική πολυπλοκότητα. Αυτή η διαδικασία σε μια βάση που χρησιμοποιεί κανονική επεξεργασία γράφου απαιτεί χρονική πολυπλοκότητα *O(1)*. Απλά πάμε στον κόμβο που μας ενδιαφέρει και κοιτάμε τις εξερχόμενες ακμές τύπου "*FOLLOWS*".

Ακόμα χειρότερα, έστω ότι αντιστρέφουμε το ερώτημα, και ψάχνουμε από ποιον ακολουθείται ένα συγκεκριμένο άτομο, έστω η Αλίκη. Σε αυτήν την περίπτωση θα έπρεπε να κοιτάξουμε πολλές φορές το ευρετήριο, μια φορά για κάθε κόμβο του δικτύου που είναι πιθανό να ακολουθεί την Αλίκη. Αν λοιπόν στο δίκτυο μας έχουμε m κόμβους ατόμων, η πολυπλοκότητα θα γινόταν *O(mlogn)*. Για κάθε άτομο θα έπρεπε να πάμε στο ευρετήριο και να δούμε αν ακολουθεί την Αλίκη. Αντίθετα, αν η βάση μας χρησιμοποιεί κανονική επεξεργασία γράφου η διαδικασία αυτή είναι πάλι *O(1)* για κάθε άτομο που ακολουθεί την Αλίκη! Δεν θα είχαμε παρά να πάμε στον κόμβο της Αλίκης και να δούμε αυτήν την φορά τις εισερχόμενες ακμές τύπου "*FOLLOWS*".



Σχήμα *4.2 [RWE]* – Ευρετήριο για την σχέση *"FOLLOWS"*

Με βάση τα παραπάνω γίνεται σαφές, στην θεωρία τουλάχιστον, ότι στις βάσεις που χρησιμοποιούν κανονική επεξεργασία γράφου (*native graph processing*) τα ερωτήματα διάσχισης μπορούν να απαντηθούν πολύ αποτελεσματικά.

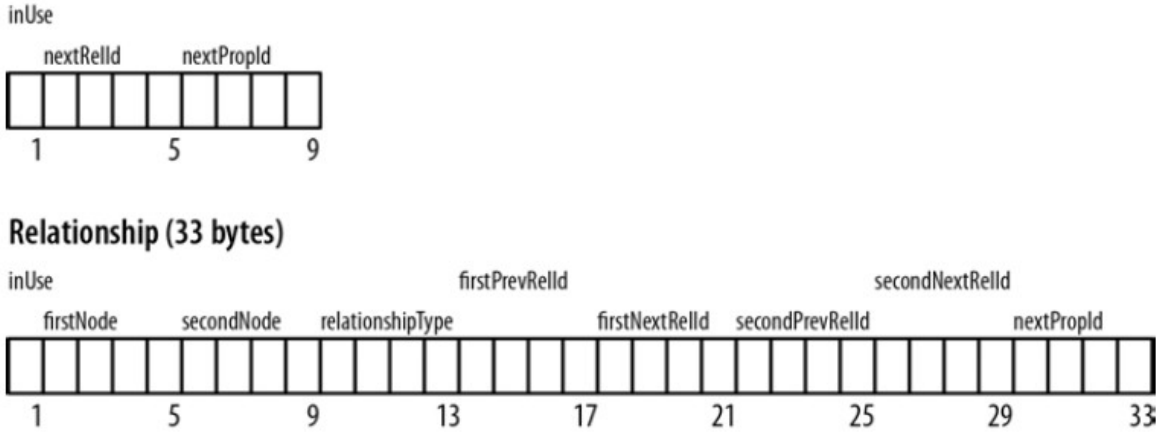
4.2.2 Κανονική Αποθήκευση Γράφου

Μια βασική πτυχή του σχεδιασμού μια βάσης δεδομένων είναι και ο τρόπος που αποθηκεύονται τα δεδομένα (στην περίπτωση μας οι γράφοι) στην μνήμη. Όπως αναφέραμε η *Neo4j* χρησιμοποιεί κανονική αποθήκευση γράφου (*native graph storage*), κάτι που οδηγεί σε υψηλή απόδοση στα ερωτήματα διάσχισης.

Η *Neo4j* αποθηκεύει τα διάφορα δεδομένα σε διαφορετικά αρχεία αποθήκευσης. Κάθε ένα από αυτά τα αρχεία έχει πληροφορία για ένα τμήμα του γράφου (αλλού οι κόμβοι, αλλού οι σχέσεις, αλλού οι ιδιότητες και τα λοιπά). Το γεγονός αυτό, και κυρίως το γεγονός ότι διαχωρίζει τα δομικά στοιχεία του γράφου από τις ιδιότητες, οδηγεί σε πολύ γρήγορες διασχίσεις του γράφου. Βέβαια αυτό συνεπάγεται ότι η άποψη του χρήστη της βάσης για την δομή του γράφου, διαφέρει από το πώς είναι στην πραγματικότητα αποθηκευμένος ο γράφος.

Οι κόμβοι αποθηκεύονται στο αρχείο αποθήκευσης κόμβων (*neostore.nodestore.db*). Κάθε εγγραφή σε αυτό το αρχείο έχει σταθερό μέγεθος (στην *Neo4j* 9 *bytes* [*RWE*]). Αυτό το σταθερό μέγεθος των εγγραφών, μας δίνει την δυνατότητα να βρίσκουμε απευθείας τον κόμβο που χρειαζόμαστε απλά γνωρίζοντας το *id* του. Για παράδειγμα ο κόμβος με *id* 100 βρίσκεται 900 (9 x 100) *bytes* από την αρχή του αρχείου. Από αυτά τα 9 *bytes* το πρώτο είναι μια σημαία. Τα επόμενα 4 είναι το *id* της πρώτης σχέσης που συνδέεται σε αυτόν τον κόμβο, και τα τελευταία 4 είναι το *id* της πρώτης ιδιότητας αυτού του κόμβου. Λόγω της σχεδίασης της βάσης, όλα αυτά τα *id* λειτουργούν ακριβώς σαν δείκτες στα αντίστοιχα αρχεία.

Ακριβώς αντίστοιχα με τους κόμβους, και οι σχέσεις αποθηκεύονται στο αρχείο αποθήκευσης σχέσεων (*neostore.relationshipstore.db*). Εδώ κάθε εγγραφή στο αρχείο έχει σταθερό μέγεθος 33 *bytes* [*RWE*]. Κάθε εγγραφή περιλαμβάνει το *id* του κόμβου αρχής, του κόμβου πέρατος, του τύπου της σχέσης (οι τύποι σχέσεων αποθηκεύονται ακριβώς αντίστοιχα σε άλλο αρχείο) καθώς και τα *id* των προηγούμενων και επόμενων σχέσεων τόσο του κόμβου αρχής όσο και πέρατος. Και πάλι, όλα αυτά τα *id* λειτουργούν ακριβώς σαν δείκτες στα αντίστοιχα αρχεία. Στο σχήμα 4.3 φαίνεται αναλυτικά η δομή κάθε εγγραφής στα αρχεία αποθήκευσης κόμβων και σχέσεων.



Σχήμα *4.3 [RWE]*- Εγγραφές των αρχείων αποθήκευσης κόμβων και σχέσεων στην *Neo4j*

Παρατηρούμε λοιπόν ότι τα αρχεία αποθήκευσης κόμβων και σχέσεων έχουν μόνο πληροφορία για τη δομή του γράφου, και απολύτως καμία για τις ιδιότητες. Με αυτόν τον τρόπο μπορούμε πολύ γρήγορα να διασχίσουμε τον γράφο απλά ακολουθώντας δείκτες μεταξύ κόμβων και σχέσεων. Για να γίνει αυτό το μόνο που απαιτείται είναι φτηνοί υπολογισμοί μεταξύ των *id* των οντοτήτων. Αντίθετα σε βάσεις που δεν χρησιμοποιούν κανονική αποθήκευση γράφου (αποθηκεύουν κόμβους και σχέσεις μαζί με τις ιδιότητες τους σε μορφή πίνακα) απαιτούνται υπολογισμοί σε ευρετήρια για να μπορέσουμε να διασχίσουμε τον γράφο.

Τέλος οι ιδιότητες αποθηκεύονται στο αρχείο αποθήκευσης ιδιοτήτων (*neostore.propertystore.db*). Αυτές οι ιδιότητες είναι της μορφής "*key-value*". Να υπενθυμίσουμε εδώ ότι η *Neo4j* είναι μια βάση δεδομένων γράφου με ιδιότητες (*property graph database*) που σημαίνει ότι τόσο οι κόμβοι όσο και οι σχέσεις μπορούν να έχουν ιδιότητες. Κάθε εγγραφή στο αρχείο ιδιοτήτων έχει σταθερό μέγεθος, και περιλαμβάνει έναν δείκτη προς το αρχείο ευρετηρίου ιδιοτήτων (*neostore.propertystore.db.index*) όπου αποθηκεύεται το όνομα της ιδιότητας, την τιμή της ιδιότητας και έναν δείκτη προς την επόμενη ιδιότητα της οντότητας.

4.3 Τα Ερωτήματα στην Neo4j

4.3.1 Cypher

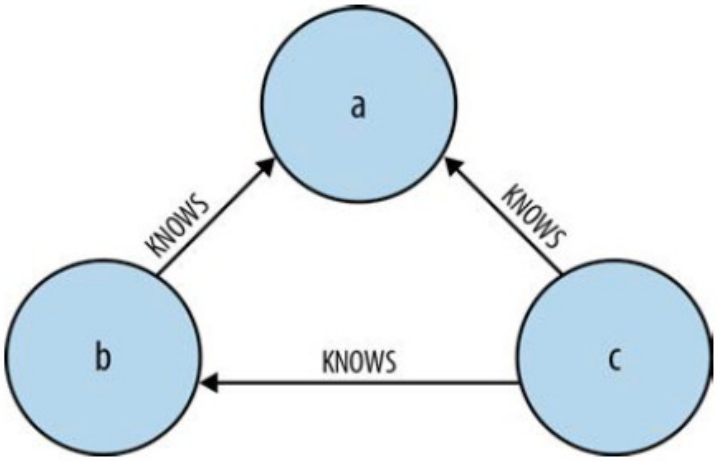
Μέχρι σήμερα δεν υπάρχει κάποια πρότυπη γλώσσα για ερωτήματα (*queries*) στις βάσεις δεδομένων γράφου (*graph databases*), όπως υπάρχει η *SQL* για τα παραδοσιακά *RDBMS* συστήματα. Η *Cypher* είναι μια από τις πολλές γλώσσες που χρησιμοποιούνται για να περιγράψουμε και να θέσουμε ερωτήματα σε βάσεις δεδομένων γράφου με ιδιότητες (*property graph databases*). Η *Neo4j* υποστηρίζει τόσο την *RDF* γλώσσα ερωτημάτων *SPARQL*[2], όσο και την προστακτική, βασισμένη στα ερωτήματα διάσχισης γλώσσα *Gremlin*[3], οι οποίες είναι δύο από της δημοφιλέστερες γλώσσες ερωτημάτων για αυτού του είδους τις βάσεις δεδομένων. Παρόλα αυτά παροτρύνει τους προγραμματιστές τις να χρησιμοποιούν την *Cypher*.

Η *Cypher* είναι μια δηλωτική γλώσσα ερωτημάτων για βάσεις δεδομένων γράφου. Είναι πολύ εκφραστική, από την άποψη ότι μοιάζει πολύ με τον τρόπο που ο άνθρωπος αντιλαμβάνεται διαισθητικά τους γράφους. Επειδή ακριβώς είναι δηλωτική, επικεντρώνεται στο να ορίσει με σαφήνεια τι να ανασύρουμε από τον γράφο και όχι πώς να το ανασύρουμε, σε αντίθεση με τις προστακτικές γλώσσες όπως η *Gremlin*.

Η *Cypher* επιτρέπει στον χρήστη να ζητήσει δεδομένα από την βάση που ταιριάζουν σε ένα συγκεκριμένο μοτίβο (*pattern*). Έτσι ρωτάμε την βάση "βρες μου πράγματα όπως αυτό". Και ο ευκολότερος τρόπος για να περιγράψουμε τα "όπως αυτό πράγματα" είναι ζωγραφίζοντας τα. Στο σχήμα 4.4 έχουμε ένα παράδειγμα ενός τέτοιου μοτίβου που

2 www.w3.org/TR/rdf-sparql-query
3 github.com/tinkerpop/gremlin/wiki

περιγράφει τρεις κοινούς φίλους.



Σχήμα *4.4 [RWE]* – Ένα απλό μοτίβο γράφου, εκφρασμένο χρησιμοποιώντας διάγραμμα

Σε *Cypher* αυτό το μοτίβο μπορεί να περιγραφεί ως εξής:

```
(c)-[:KNOWS]->(b)-[:KNOWS]->(a), (c)-[:KNOWS]->(a)
```

Αυτό το μοτίβο περιγράφει ένα μονοπάτι που ενώνει τους c με b, b με a και c με a. Το μόνο που μας εμποδίζει είναι ότι το γράφημα βρίσκεται σε χώρο δύο διαστάσεων ενώ η γλώσσα μας έχει μόνο μία διάσταση. Για αυτόν τον λόγο χρησιμοποιούμε και το κόμμα για να χωρίσουμε το μοτίβο μας σε δύο υπομοτίβα.

Για να εκφράσει τα ερωτήματα της η γλώσσα *Cypher* χρησιμοποιεί τακτικές διαφόρων γλωσσών ερωτημάτων. Οι περισσότερες από τις λέξεις κλειδιά της ("*WHERE*", "*ORDER BY*") είναι εμπνευσμένες από την *SQL*, ενώ το ταίριασμα μοτίβων (*pattern matching*) είναι δανεισμένο από την *SPARQL*. Πάμε να δούμε συνοπτικά τις βασικότερες λέξεις κλειδιά που χρησιμοποιεί η *Cypher*.

START

Με το *START* μπορούμε να ορίσουμε ένα ή περισσότερα σημεία αρχής (κόμβους ή ακμές) στον γράφο. Ουσιαστικά πρόκειται για τα σημεία του γράφου από όπου θα ξεκινήσουμε για να απαντήσουμε σε κάποιο ερώτημα. Πρέπει λοιπόν να μπορούμε να φτάσουμε σε αυτά τα σημεία πολύ γρήγορα. Οπότε χρησιμοποιούμε κάποιο ευρετήριο ή το *id* των κόμβων και των ακμών από όπου θέλουμε να ξεκινήσουμε την ερώτηση μας. Ας θυμηθούμε εδώ ότι η *Neo4j* χρησιμοποιεί κανονική αποθήκευση γράφου (*native graph storage*) και συνεπώς μπορούμε σε *O(1)* να προσπελάσουμε τους ζητούμενους κόμβους και ακμές όταν χρησιμοποιούμε το *id* τους.

MATCH

Με το *MATCH* ορίζουμε ποιο είναι το μοτίβο στο γράφο που θέλουμε να βρούμε. Όπως είδαμε και προηγουμένως ουσιαστικά "ζωγραφίζουμε" το μοτίβο που ψάχνουμε. Μέσα

σε παρένθεση εκφράζουμε τους κόμβους. Με δύο παύλες και το σύμβολο του μικρότερου (<--) ή του μεγαλύτερου (-->) εκφράζουμε τις σχέσεις. Οπτικά λοιπόν ένας κόμβος μοιάζει με κύκλο και μια σχέση με βέλος. Μετά τον ειδικό χαρακτήρα ":" ακολουθεί η ταμπέλα (*label*), δηλαδή ο τύπος της σχέσης (στο παράδειγμα του σχήματος 4.4 "*KNOWS*") ή του κόμβου. Με κεφαλαία γράφουμε τις λέξεις κλειδιά, ενώ με μικρά τις μεταβλητές. Στις μεταβλητές ουσιαστικά "δένουμε" κόμβους ή σχέσεις του γράφου, ώστε να ικανοποιείται το μοτίβο.

WHERE

Το *WHERE* δεν μπορεί να εμφανιστεί μόνο του, αλλά μόνο ως μέρος μιας *MATCH* πρότασης. Ουσιαστικά εισάγει περιορισμούς στο μοτίβο που ορίζεται στην πρόταση *MATCH*, περιορίζοντας τα τμήματα του γράφου που ικανοποιούν αυτό το μοτίβο.

RETURN

Το *RETURN* ορίζει ποιοι κόμβοι, σχέσεις ή και ιδιότητες των δεδομένων που ικανοποιούν το μοτίβο θέλουμε να επιστραφούν στον χρήστη.

Για να γίνουν τα παραπάνω κατανοητά ας δούμε ένα παράδειγμα. Έστω ότι ψάχνουμε να βρούμε ποιους χρήστες ακολουθεί ο Ηλίας. Έχουμε 2 επιλογές ˙να χρησιμοποιήσουμε ευρετήριο ή όχι.

Example 1

```
START a=node:user(name='Helias')
MATCH (a)-[:FOLLOWS]->(b)
RETURN b
```

Example 2

```
MATCH (a)-[:FOLLOWS]->(b)
WHERE a.name='Helias'
RETURN b
```

Στην πρώτη υλοποίηση υποθέτουμε ότι έχουμε το ευρετήριο *user* για την ιδιότητα *name*. Ως κόμβο αρχής, ζητάμε από το ευρετήριο να μας βρει τον κόμβο που έχει όνομα Ηλίας. Αφού τον βρει, "δένει" αυτόν τον κόμβο στην μεταβλητή a. Με αυτόν τον τρόπο αγκιστρώνουμε το μοτίβο που ακολουθεί, σε ένα συγκεκριμένο σημείο του γράφου. Επίσης, μέσω της μεταβλητής a μπορούμε να αναφερόμαστε στον κόμβο αρχής για όλο το υπόλοιπο ερώτημα. Στην πρόταση *MATCH* βλέπουμε το μοτίβο. Πρόκειται για ένα μονοπάτι που περιλαμβάνει 2 κόμβους, εκ των οποίων ο ένας ξέρουμε ότι είναι ο Ηλίας. Στην πρόταση *RETURN* επιστρέφονται όσοι κόμβοι ικανοποιούν το παραπάνω μοτίβο,

δηλαδή όσους χρήστες ακολουθεί ο Ηλίας.

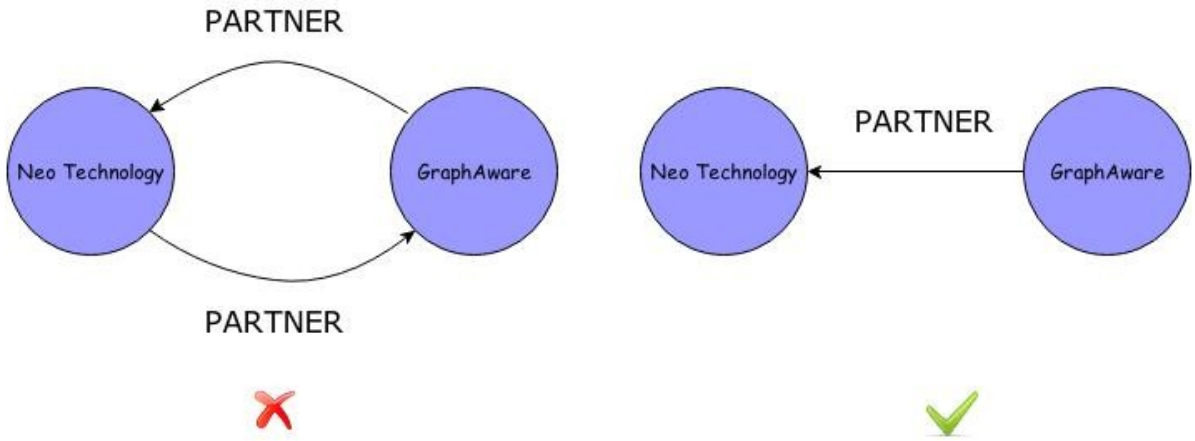
Αντίθετα, η δεύτερη υλοποίηση λειτουργεί διαφορετικά. Το μοτίβο είναι ακριβώς το ίδιο. Αυτή την φορά όμως, δεν ψάχνουμε σε ένα σημείου του γράφου για κόμβους που ικανοποιούν το μοτίβο. Ψάχνουμε σε ολόκληρο τον γράφο για τέτοια μοτίβα. Θεωρητικά θα βρούμε πολλά τέτοια. Εδώ λοιπόν έρχεται η πρόταση *WHERE*. Μας λέει ποια από αυτά τα μοτίβα θέλουμε να κρατήσουμε - αυτά που ο αριστερός κόμβος έχει όνομα Ηλίας. Με αυτόν τον τρόπο περιορίζουμε τα μοτίβα που ικανοποιούν το ερώτημα μας. Στην πρόταση *RETURN* επιστρέφονται οι δεξιοί κόμβοι αυτών των μοτίβων.

Να τονίσουμε εδώ ότι τα δύο ερωτήματα θα μας επιστρέψουν ακριβώς το ίδιο αποτέλεσμα. Το πρώτο ερώτημα εκτελείται φυσικά σε πολύ μικρότερο χρόνο, καθώς εκμεταλλεύεται την ύπαρξη ευρετηρίου στην ιδιότητα *name*.

4.3.2 Χειρισμός Σχέσεων

Η *Neo4j* είναι μια βάση δεδομένων γράφου με ιδιότητες (*property graph database*). Αυτό σημαίνει ότι κάθε σχέση έχει, πέρα από ιδιότητες και ταμπέλες, έναν κόμβο αρχής και έναν κόμβο πέρατος και συνεπώς κατεύθυνση.

Πολλές φορές όμως, στα πλαίσια μιας εφαρμογής, δεν θέλουμε κάποια σχέση να έχει κατεύθυνση. Στο παράδειγμα του σχήματος 4.5, η σχέση "*PARTNER*" που συνδέει δύο κόμβους εταιριών δεν έχει συγκεκριμένη κατεύθυνση. Αν μια εταιρία είναι συνεργάτης μιας άλλης, τότε και η δεύτερη είναι συνεργάτης της πρώτης. Για να ξεπεράσουμε αυτόν τον περιορισμό θα μπορούσαμε να δημιουργήσουμε δύο σχέσεις. Μία από την εταιρία *Neo Technologies* στην *GraphAware* και μία από την *GraphAware* στην *Neo Technologies,* για το παράδειγμα του σχήματος 4.5. Κάτι τέτοιο δεν είναι αποδοτικό, καθώς δημιουργούμε μεγάλο αριθμό σχέσεων που δεν έχουν καμία πληροφορία. Στην *Neo4j* αντί για σχέσεις διπλής κατεύθυνσης, μπορούμε απλά να αγνοήσουμε την κατεύθυνση των σχέσεων κατά την διάρκεια των ερωτημάτων (*queries*). Συνεπώς αντί για δύο σχέσεις, δημιουργούμε μια σχέση με τυχαία κατεύθυνση, και απλά φροντίζουμε να αδιαφορούμε για την κατεύθυνση αυτή όταν συναντάμε την συγκεκριμένη σχέση.



Σχήμα *4.5* – Τρόπος χειρισμού των διμερών σχέσεων στην *Neo4j*

Άλλη μια σημαντική παράμετρος που πρέπει να σκεφτούμε κατά τον σχεδιασμό του σχήματος μιας βάσης δεδομένων, αφορά τις ταμπέλες των σχέσεων. Πρέπει να επιλέξουμε ανάμεσα στην χρησιμοποίηση καλά προσδιορισμένων (*fine-grained*) σχέσεων ή γενικότερων σχέσεων που προσδιορίζονται από τις ιδιότητες. Για παράδειγμα υπάρχει τεράστια διαφορά ανάμεσα στην χρησιμοποίηση δύο ειδών σχέσεων με ταμπέλες `HOME_ADDRESS` και `WORK_ADDRESS`, και στην χρησιμοποίηση ενός είδους σχέσεων με ταμπέλα `ADDRESS` αλλά με διαφοροποίηση στις ιδιότητες (μερικές σχέσεις θα έχουν ιδιότητα {type: `HOME`} και άλλες {type: `WORK`}).Η επιλογή βασίζεται σε τι είδους ερωτήματα διάσχισης θέλουμε να εκτελέσουμε.

Αν για παράδειγμα, έχουμε ένα ερώτημα διάσχισης που ακολουθεί μόνο τις διευθύνσεις κατοικίας και αγνοεί τις διευθύνσεις εργασίας, τότε η υλοποίηση με καλά προσδιορισμένες σχέσεις είναι πιο αποδοτική. Αυτό συμβαίνει επειδή στην υλοποίηση με γενικές σχέσεις υπάρχει κόστος για να επισκεφτούμε τις ιδιότητες μιας σχέσης ώστε να ξεχωρίσουμε τις διευθύνσεις εργασίας από αυτές της κατοικίας. Στην ενότητα 4.2.2 αναφέραμε ότι η *Neo4j* χρησιμοποιεί φυσική αποθήκευση γράφου (*native graph storage*) και συνεπώς οι ιδιότητες μιας σχέσης είναι αποθηκευμένες ξεχωριστά από την ίδια την σχέση.

Αντίθετα, αν είχαμε ένα ερώτημα διάσχισης που ακολουθεί τις διευθύνσεις όλων των ειδών η υλοποίηση με καλά προσδιορισμένες σχέσεις θα μας δυσκόλευε. Θα έπρεπε να χρησιμοποιήσουμε όλων των ειδών τις σχέσεις σε ένα ερώτημα της μορφής:

```
(u)-[:WORK_ADDRESS|HOME_ADDRESS|OTHER...]->(a)
```

Βέβαια σε περίπτωση ύπαρξης πολλών ειδών σχέσεων κάτι τέτοιο θα ήταν δύσκολο να γίνει.

Εναλλακτικά θα μπορούσαμε να προσθέσουμε στο σχήμα της βάσης πέρα από τις καλά προσδιορισμένες σχέσεις, και γενικές σχέσεις. Έτσι κάθε κόμβος που αντιπροσωπεύει έναν χρήστη θα συνδέεται με έναν κόμβο που αντιπροσωπεύει μια διεύθυνση με δύο ειδών σχέσεων. Μια γενική (`ADDRESS`) και μια ειδικότερη (πχ. `WORK_ADDRESS`). Σε κάθε περίπτωση το είδος των σχέσεων που θα χρησιμοποιήσουμε εξαρτάται από το είδος των ερωτημάτων διάσχισης που θα εκτελέσουμε.

4.3.3 Ερωτήματα Διάσχισης

Ένας από τους λόγους που οι βάσεις δεδομένων γράφου (*graph databases*) έχουν γίνει ιδιαίτερα δημοφιλείς είναι η ευκολία και η υψηλή απόδοση με την οποία μπορούμε να εκτελέσουμε ερωτήματα που αφορούν τη διάσχιση του γράφου. Η *Neo4j* είναι δομημένη ώστε να μπορεί να υποστηρίξει αποτελεσματικά έναν μεγάλο αριθμό αλγορίθμων γράφου (*graph algorithms*) που πετυχαίνουν αυτόν τον σκοπό. Η αποτελεσματική αντιμετώπιση τέτοιου είδους ερωτημάτων, είναι ένας από τους βασικότερους λόγους για τους οποίους η *Neo4j* χρησιμοποιεί την κανονική επεξεργασία γράφου (*native graph processing*).

Ανάμεσα στους αλγορίθμους γράφου που η *Neo4j* έχει ενσωματωμένους (*built-in*

algorithms) είναι η αναζήτηση κατά βάθος (*Depth First Search – DFS*), η αναζήτηση κατά πλάτος (*Breadth First Search – BFS*), ο αλγόριθμος εύρεσης συντομότερων διαδρομών *Dijkstra*, καθώς και ο αλγόριθμος *A**. Φυσικά πέρα από αυτούς τους αλγορίθμους, μπορούμε να υλοποιήσουμε προγραμματιστικά οποιονδήποτε αλγόριθμο γράφου επιθυμούμε. Να σημειώσουμε εδώ ότι προς το παρόν, οι προαναφερθέντες αλγόριθμοι δεν είναι διαθέσιμοι μέσω της γλώσσας ερωτημάτων *Cypher*, κάτι που θα παρουσίαζε ιδιαίτερο ενδιαφέρον.

Στα πλαίσια της εφαρμογής μας θα χρειαστεί να υπολογίσουμε τους γείτονες ν-επιπέδων (*n-hop neighbors*) κάποιου κόμβου, καθώς και τα συντομότερα μονοπάτια (*shortest paths*) μεταξύ δύο κόμβων. Για τον σκοπό αυτό είναι χρήσιμο να εισάγουμε την έννοια του μονοπατιού (*path*). Ένα μονοπάτι στη *Neo4j* είναι μια σειρά κορυφών οι οποίες συνδέονται μέσω ακμών. Πιο συγκεκριμένα, ένα μονοπάτι στην *Neo4j* έχει υποχρεωτικά έναν κόμβο αρχής και έναν κόμβο πέρατος (οι κόμβοι αυτοί μπορεί και να ταυτίζονται στα μονοπάτια μηδενικού μήκους). Επίσης μπορεί να έχει μία ή περισσότερες σχέσεις, η κάθε μια από τις οποίες ακολουθείται από έναν κόμβο. Ένα μονοπάτι στη *Neo4j* μπορούμε να το αναθέσουμε σε μία μεταβλητή.

Για την επεξεργασία ενός μονοπατιού (δηλαδή την εξαγωγή χρήσιμης πληροφορίας από την δομή αυτή), μπορούμε να χρησιμοποιήσουμε κάποιες έτοιμες συναρτήσεις, που ονομάζονται συναρτήσεις συλλογής. Αυτές οι συναρτήσεις ονομάζονται έτσι επειδή παίρνουν ως παράμετρο ή επιστρέφουν ως αποτέλεσμα μια συλλογή (*collection*). Μια συλλογή στη *Neo4j* είναι αντικείμενα (ακέραιοι αριθμοί, συμβολοσειρές, κόμβοι, σχέσεις και άλλα) που χωρίζονται με κόμμα και βρίσκονται ανάμεσα σε αγκύλες. Για παράδειγμα, μια συλλογή ακεραίων είναι η `[0,1,2,3,4,5,6,7,8,9]`, η οποία μπορεί να γραφτεί και ως `range(0,9)`. Οι συναρτήσεις συλλογής που θα μας φανούν χρήσιμες είναι οι `NODES`(*path*), `RELATIONSHIPS`(*path*), `LENGTH`(*path*) και `REDUCE`(*accumulator = init_value, id in collection | expression*). Οι πρώτες 2 συναρτήσεις επιστρέφουν μια συλλογή κόμβων και σχέσεων ενός μονοπατιού, με την σειρά που αυτά εμφανίζονται στο μονοπάτι. Η τρίτη επιστρέφει το μήκος του μονοπατιού, ενώ η `REDUCE` εφαρμόζει μια έκφραση σε όλα τα στοιχεία μιας συλλογής (συνήθως οι κόμβοι ή οι σχέσεις ενός μονοπατιού).

n-hop neighbors

Για τον υπολογισμό των γειτόνων ν-επιπέδων χρησιμοποιούμε το παρακάτω ερώτημα *Cypher* (συγκεκριμένα για τους γείτονες του τρίτου επιπέδου).

3-hop neighbors

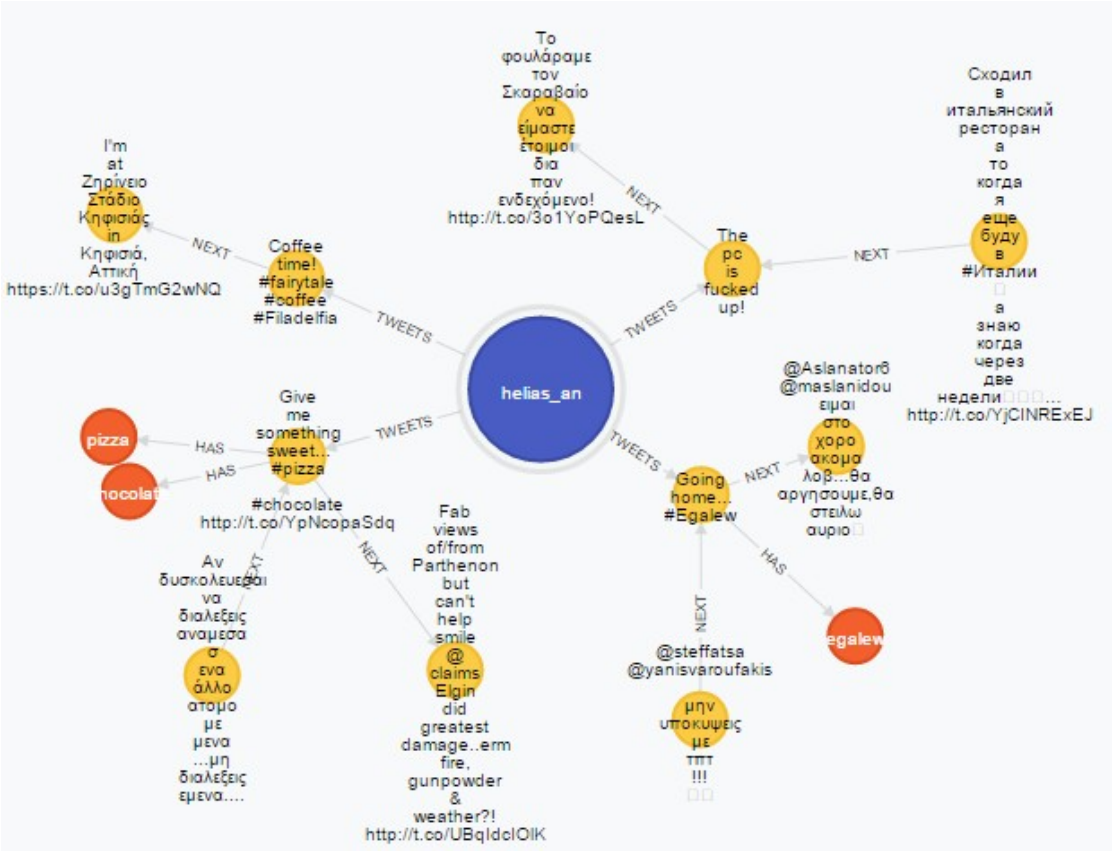
```
MATCH path=((u:User)-[*3..3]-(anything))
WHERE u.Username='helias_an'
RETURN nodes(path), length(path)
```

Αρχικά θέτουμε στην μεταβλητή `path` το μονοπάτι. Ορίζουμε τον κόμβο αρχής (μέσω της έκφρασης `WHERE`) και αφήνουμε ελεύθερο τον κόμβο πέρατος να πάρει οποιαδήποτε τιμή. Η *Neo4j* ψάχνει να ταιριάξει το ζητούμενο μοτίβο στα δεδομένα του γράφου (έκφραση `MATCH`). Αλλά αντί να μας γυρίσει κάποιον συγκεκριμένο κόμβο ή σχέση (όπως για παράδειγμα τον κόμβο `anything`), μας γυρίζει ολόκληρο το μοτίβο που είναι αποθηκευμένο στην μεταβλητή `path`.

Το αποθηκευμένο μοτίβο θα μπορούσε να είναι ένα κλασικό μοτίβο της μορφής:

```
(u:User)<-[:FOLLOWS]-(anything)
```

Σε αυτήν την περίπτωση η μεταβλητή `path` θα ήταν αναγκαστικά ένα μονοπάτι μήκους 1 με την σχέση ανάμεσα στους 2 κόμβους να είναι σχέση "*FOLLOWS*". Αντίθετα εμείς επιλέγουμε να αδιαφορήσουμε για την κατεύθυνση αλλά και τον τύπο της σχέσης (ενότητα *4.3.2*). Τέλος επιλέγουμε το μοτίβο μας να αφορά όλα τα μονοπάτια με ελάχιστο μήκος τρία και μέγιστο τρία. Δηλαδή όλα τα μονοπάτια ακριβώς μήκους τρία. Όλες αυτές οι επιλογές έγιναν, προκειμένου να υπολογίσουμε τους γείτονες ν-επιπέδου ενός κόμβου συνολικά στον γράφο. Αν για παράδειγμα θέλαμε να υπολογίσουμε το δίκτυο των ακολούθων θα περιορίζαμε την σχέση τόσο ως προς την ταμπέλα της (πλέον μόνο "*FOLLOWS*") όσο και ως προς την κατεύθυνση της, αλλά θα διευρύναμε το δυνατό εύρος τιμών για το μήκος των μονοπατιών (για παράδειγμα από 1 έως 10 ή από 5 έως *).



Σχήμα *4.6* – Γειτονιά *2*-επιπέδων για τον κόμβο τύπου *User "helias_an"*

Στο σχήμα 4.6 μπορούμε να δούμε οπτικά το παραπάνω ερώτημα. Προκειμένου τα

αποτελέσματα να είναι εμφανέστερα επιλέξαμε να δείξουμε τους γείτονες του δευτέρου επιπέδου περιορίζοντας τους και αυτούς στον αριθμό 10.

shortest path

Για τον υπολογισμό του ή των συντομότερων μονοπατιών ανάμεσα σε δύο δεδομένους κόμβους χρησιμοποιούμε το παρακάτω ερώτημα *Cypher*.

Shortest path

```
MATCH path=shortestPath((u:User)-[*]-(h:HashTag))
WHERE u.Username='helias_an' AND h.Hashtag='ny'
RETURN nodes(path), length(path)
```

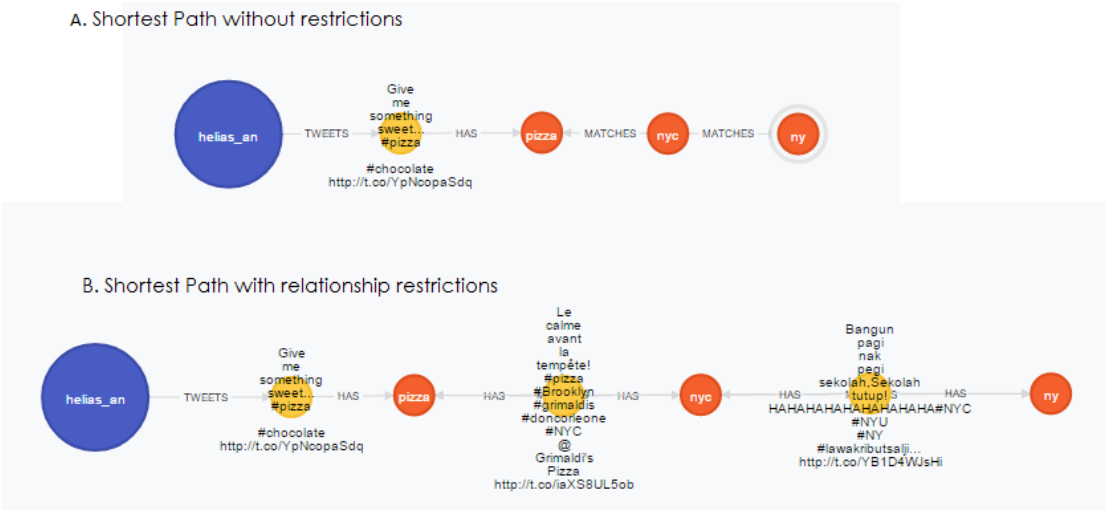
Στο παραπάνω ερώτημα αναζητούμε το συντομότερο μονοπάτι ανάμεσα σε έναν χρήστη και ένα *hashtag*. Όπως και στους γείτονες ν-επιπέδων, θέτουμε το μονοπάτι στην μεταβλητή `path`. Η έννοια του συντομότερου μονοπατιού απαιτεί την ύπαρξη συγκεκριμένου κόμβου αρχής και συγκεκριμένου κόμβου πέρατος. Αυτό το πετυχαίνουμε μέσω της έκφρασης `WHERE`. Με τη δεσμευμένη λέξη `shortestPath` ενημερώνουμε την *Neo4j* ότι αναζητούμε το συντομότερο μονοπάτι μεταξύ των δύο κόμβων. Σε περίπτωση ύπαρξης περισσότερων του ενός συντομότερων μονοπατιών η συγκεκριμένη έκφραση θα επιστρέψει μόνο ένα.

Αν θέλαμε να όλα τα συντομότερα μονοπάτια ανάμεσα σε δύο κόμβους (μονοπάτια ίδιου μήκους), θα χρησιμοποιούσαμε την έκφραση `allshortestPaths`. Σε περίπτωση που θέλαμε το συντομότερο μονοπάτι να διασχίζει συγκεκριμένους τύπους σχέσεων θα μπορούσαμε να αλλάξουμε το ερώτημα μας θέτοντας τους αντίστοιχους περιορισμούς. Για παράδειγμα το παρακάτω μοτίβο υπολογίζει τα μονοπάτια που αποτελούνται αποκλειστικά από σχέσεις τύπου "*TWEETS*" και "HAS". Μάλιστα θέτει ως ανώτατο όριο αναζήτησης τα μονοπάτια μήκους 8.

```
(u:User)-[:TWEETS | HAS *..8]-(h:HashTag)
```

Στο σχήμα φαίνεται η σύγκριση των συντομότερων μονοπατιών ανάμεσα στους ίδιους κόμβους όταν θέτουμε και όταν όχι, περιορισμούς στη σχέση. Τέλος να σημειώσουμε ότι δεν είναι υποχρεωτικό να δεσμεύσουμε και τις δύο άκρες του μονοπατιού σε κάποιο κόμβο. Σε μια τέτοια περίπτωση θα μπορούσαμε να υπολογίσουμε τα συντομότερα μονοπάτια από τον κόμβο ενδιαφέροντος σε όλους τους κόμβους του γραφήματος ή όλους του κόμβους του γραφήματος κάποιου τύπου σε περίπτωση που βάζαμε κάποιον περιορισμό ταμπέλας (*label*).

Για τον υπολογισμό των συντομότερων μονοπατιών η *Neo4j* χρησιμοποιεί τον αλγόριθμο αναζήτησης κατά πλάτος – *BFS*. Δηλαδή ξεκινώντας από μια μια κορυφή επισκέπτεται αρχικά τους γείτονες σε βάθος ένα, μετά αυτούς σε βάθος δύο και συνεχίζει έτσι μέχρι να επισκεφτεί τον κόμβο – στόχο.



Σχήμα *4.7* – Συντομότερα μονοπάτια χωρίς (A) και με (B) περιορισμούς στις σχέσεις

Υλοποίηση shortest path σε γράφο με βάρη στις ακμές

Η *Neo4j* έχει ενσωματωμένο αλγόριθμο υπολογισμού του ελαφρύτερου (με βάση το βάρος των ακμών) μονοπατιού μεταξύ δύο δεδομένων κορυφών (*Dijkstra*). Ο αλγόριθμος *Dijkstra* όμως δεν είναι διαθέσιμος μέσω της γλώσσας ερωτημάτων *Cypher*. Εξαιτίας αυτού του γεγονότος, θα αναφέρουμε μια εναλλακτική υλοποίηση για τον υπολογισμό του ελαφρύτερου μονοπατιού σε κάποιο γράφο που έχει βάρη στις ακμές, χρησιμοποιώντας την συνάρτηση συλλογής *REDUCE*. Η συνάρτηση *REDUCE* εφαρμόζει μια έκφραση πάνω σε όλα τα στοιχεία μιας συλλογής. Ως συλλογή εδώ, θα χρησιμοποιήσουμε το σύνολο των σχέσεων ενός μονοπατιού. Στα πλαίσια της εφαρμογής μας, οι μόνες ακμές που έχουν βάρος είναι αυτές του τύπου "*MATCHES*" που ενώνουν δύο κόμβους *Hashtag*.

Shortest path in weighted graph

```
MATCH path=((u:HashTag)-[:MATCHES*]-(h:HashTag))
WHERE u.Hashtag='egypt' AND h.Hashtag='ny'
RETURN nodes(path), length(path),
REDUCE(sum=0, x in relationships(path)| sum +
x.Frequency) as Weight,
ORDER BY Weight, LIMIT 1
```

Αρχικοποιούμε λοιπόν την μεταβλητή `sum` , και στην συνέχεια για κάθε στοιχείο της συλλογής των σχέσεων του μονοπατιού `x` (`x in relationships(path)`) προσθέτουμε σε αυτή την τιμή της ιδιότητας *Frequency* των σχέσεων του μονοπατιού. Έχουμε φροντίσει από πριν (έκφραση *MATCH* στο ερώτημα) κάθε σχέση του

μονοπατιού να είναι τύπου "*MATCHES*", και συνεπώς να έχει τέτοια ιδιότητα. Τελικά ταξινομούμε τα μονοπάτια βάση αυτής της ιδιότητας και επιστρέφουμε μόνο το ελαφρύτερο. Είναι εύκολα κατανοητό, ότι η συγκεκριμένη υλοποίηση δεν μπορεί να πετύχει την αποδοτικότητα του αλγορίθμου *Dijkstra*, αλλά μας δίνει την δυνατότητα να υπολογίσουμε το ελαφρύτερο μονοπάτι ανάμεσα σε δύο δεδομένους κόμβους, χρησιμοποιώντας μόνο ένα ερώτημα σε γλώσσα *Cypher*.

4.4 Λειτουργίες

Οι περισσότερες βάσεις δεδομένων σήμερα τρέχουν σαν εξυπηρετητές (*servers*) που μπορούν να προσπελαστούν μέσω μιας βιβλιοθήκης εξυπηρετούμενου (*client library*). Η *Neo4j* εκτός από την λειτουργία εξυπηρετητή, μπορεί να τρέξει και σε ενσωματωμένη λειτουργία (*embedded mode*).

Ενσωματωμένη Λειτουργία (Embedded Mode)

Στην ενσωματωμένη λειτουργία η βάση δεδομένων μας τρέχει στην ίδια διεργασία με την εφαρμογή μας. Επειδή ακριβώς η εφαρμογή μας επικοινωνεί απευθείας με την βάση δεδομένων, η επιλογή αυτής της λειτουργίας έχει ένα μεγάλο πλεονέκτημα: ταχύτητα. Σε αυτήν την λειτουργία, είναι η εφαρμογή μας υπεύθυνη για όλες τις συναλλαγές (*transactions*). Με αυτόν τον τρόπο έχουμε την δυνατότητα να κάνουμε αρκετά σύνθετες λειτουργίες στη βάση δεδομένων στα πλαίσια μιας συναλλαγής, καθώς εμείς αποφασίζουμε πότε μια συναλλαγή θα ξεκινήσει και πότε θα ολοκληρωθεί.

Πέρα από την υψηλή απόδοση, το να μοιράζεται η εφαρμογή μας με την βάση δεδομένων την ίδια διεργασία θέτει και αρκετούς περιορισμούς. Στην ενσωματωμένη λειτουργία η εφαρμογή μας είναι υπεύθυνη για τον έλεγχο του κύκλου ζωής της βάσης δεδομένων μας, που περιλαμβάνει το ομαλό ξεκίνημα και κλείσιμο της. Έτσι αν η εφαρμογή μας για κάποιον λόγο καταρρεύσει απότομα (*crashes*) θα καταρρεύσει και η βάση και οτιδήποτε άλλο είναι συνδεδεμένο στην βάση δεν θα λειτουργεί. Επίσης με την κατάρρευση της βάσης υπάρχει κίνδυνος στην συνέχεια να μην μπορούμε να την επαναφέρουμε.

Λειτουργία Εξυπηρετητή (Server Mode)

Η λειτουργία εξυπηρετητή διαχωρίζει την εφαρμογή μας από την βάση δεδομένων. Η επικοινωνία της εφαρμογής μας με την βάση γίνεται μέσω της προσφερόμενης *REST* διεπαφής (*REST API*). Αυτή μας δίνει την δυνατότητα να στέλνουμε αιτήματα σε μορφή *JSON* πάνω στο *HTTP*. Κάθε τέτοιο αίτημα είναι και μια συναλλαγή με την βάση. Ο τρόπος αυτός επικοινωνίας κάνει την βάση δεδομένων προσπελάσιμη από οποιοδήποτε εξυπηρετούμενο (*client*), που μπορεί να τρέχει σε οποιαδήποτε πλατφόρμα. Το μόνο που χρειάζεται είναι μια *HTTP* βιβλιοθήκη. Στα πλαίσια της εφαρμογής μας χρησιμοποιήσαμε τόσο μια *Java* εφαρμογή όσο και μια εφαρμογή στο *Node.js* για να

προσπελάσουμε την βάση.

Όπως γίνεται αντιληπτό η λειτουργία εξυπηρετητή δεν είναι τόσο αποδοτική από άποψη ταχύτητας. Ο διαχωρισμός της εφαρμογής από την βάση συνεπάγεται μια επιβάρυνση επικοινωνίας για κάθε αίτημα *HTTP* (*HTTP request*). Αυτή η επιβάρυνση αν και ελάχιστη, από την άποψη ότι μετά το πρώτο αίτημα του εξυπηρετούμενου μια σύνδεση TCP παραμένει ανοιχτή, παραμένει υπολογίσιμη.

Επίσης πλέον πολύπλοκες λειτουργίες είναι δύσκολο να γίνουν στα πλαίσια μιας συναλλαγής. Για να επιτευχθούν τέτοιες λειτουργίες χρειαζόμαστε πολλές συναλλαγές, που σημαίνει πολλά αιτήματα *HTTP* και συνεπώς μεγάλη επιβάρυνση. Για αυτόν τον σκοπό μπορούν να αναπτυχθούν διάφορες επεκτάσεις στον εξυπηρετητή (*server extensions*). Οι επεκτάσεις είναι *Java* κώδικας που τρέχει μέσα στον εξυπηρετητή. Μας επιτρέπουν να επεκτείνουμε την *REST* διεπαφή του εξυπηρετητή, ώστε με ένα αίτημα να μπορούμε να πετύχουμε πολύπλοκες λειτουργίες στην βάση.

Στα πλαίσια της εφαρμογής μας επιλέξαμε να χρησιμοποιήσουμε την λειτουργία εξυπηρετητή, ώστε να έχουμε πρόσβαση στην βάση μας από πολλά περιβάλλοντα. Η επικοινωνία της *Java* εφαρμογής μας με την βάση πετυχαίνεται μέσω της βιβλιοθήκης *Java REST Bindings*, η οποία περιτυλίγει τις *REST* κλήσεις πίσω από την διεπαφή *GraphDatabaseService* που χρησιμοποιείται στην ενσωματωμένη λειτουργία (*embedded mode*). Αντίστοιχα η επικοινωνία του *Node.js* εξυπηρετητή μας (κεφάλαιο 5) με την βάση γίνεται μέσω του πακέτου *neo4j*. Αυτό μας παρέχει μια διεπαφή για να χρησιμοποιήσουμε την REST διεπαφή της βάσης μας. Τέλος χρησιμοποιήσαμε μια επέκταση στον εξυπηρετητή για να πετύχουμε λειτουργίες χωρικού ευρετηρίου (*spatial index*).

4.5 Χωρικό Ευρετήριο

Τα δεδομένα της εφαρμογής μας συνοδεύονται από έντονη γεωγραφική πληροφορία. Καλούμαστε να αντιμετωπίσουμε αποδοτικά λοιπόν, τους εξής τύπους ερωτημάτων [*SKS*]:

- Ερωτήματα Εγγύτητας. Αυτά τα ερωτήματα ζητούν αντικείμενα που βρίσκονται κοντά σε μια συγκεκριμένη θέση. Το ερώτημα “Ποια εστιατόρια βρίσκονται σε απόσταση 5 χιλιομέτρων από το κέντρο της Αθήνας?” είναι χαρακτηριστικό παράδειγμα ερωτήματος αυτή της κατηγορίας.
- Ερωτήματα περιοχών. Αυτά τα ερωτήματα σχετίζονται με χωροταξικές περιοχές. Ουσιαστικά αναζητάμε τα αντικείμενα που βρίσκονται ολικά ή εν μέρη, μέσα σε μια συγκεκριμένη περιοχή. Ένα τέτοιο παράδειγμα είναι ένα ερώτημα που ζητά όλα τα καταστήματα στα γεωγραφικά όρια μιας πόλης.
- Ερωτήματα τομών και ενώσεων περιοχών. Για παράδειγμα, έχοντας δεδομένα για την πυκνότητα του πληθυσμού διαφόρων περιοχών, ένα τέτοιο ερώτημα μπορεί να αναζητήσει όλες τις περιοχές με υψηλή πυκνότητα πληθυσμού.

Για να πετύχουμε υψηλή απόδοση σε ερωτήματα τέτοιου τύπου απαιτείται η χρησιμοποίηση χωρικών ευρετηρίων (*Spatial indexes*). Οι παραδοσιακές δομές ευρετηρίων όπως τα ευρετήρια *hash* και τα B-δέντρα, δεν είναι κατάλληλα, αφού σχετίζονται μόνο με μονοδιάστατα δεδομένα, ενώ τα χωροταξικά δεδομένα είναι τυπικά δύο ή περισσοτέρων διαστάσεων. Η *Neo4j* μας παρέχει μια χωρική επέκταση, την χωρική *Neo4j* (*Neo4j Spatial*). Πρόκειται ουσιαστικά για μια βιβλιοθήκη που υλοποιεί ένα χωρικό ευρετήριο.

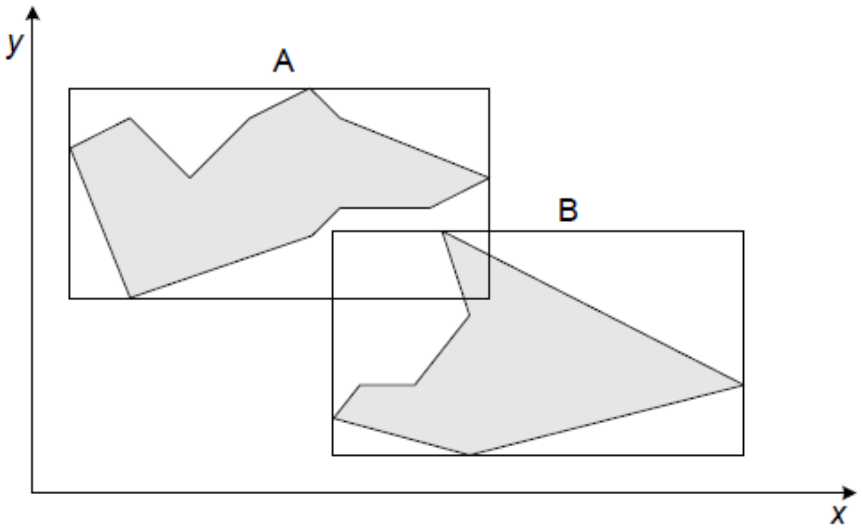
Τα χωρικά ευρετήρια υλοποιούνται από διάφορες δομές δέντρων. Παραδείγματα τέτοιων δέντρων είναι τα k-d δέντρα, τα τετρα-δέντρα και τα R-δέντρα. Στην συνέχεια της παρούσας ενότητας θα περιγράψουμε τα R-δέντρα, καθώς τα περισσότερα χωρικά ευρετήρια (ανάμεσά τους και αυτό της *Neo4j*) χρησιμοποιούν αυτή την δομή.

4.5.1 R-Δέντρα

Τα R-δέντρα (*R-trees*) είναι μια δομή αποθήκευσης, που έχει ως σκοπό να αποθηκεύσει και να ανασύρει αποτελεσματικά γεωμετρικά αντικείμενα όπως σημεία, ευθύγραμμα τμήματα, επιφάνειες ακόμα και όγκους σε χώρους πολλών διαστάσεων. Προτάθηκαν αρχικά από τον *Antonin Guttman* το 1984, με αρχικό αντικείμενο εφαρμογής τον σχεδιασμό συστημάτων *VLSI* (*Very Large Scale Integration*), προσπαθώντας να απαντήσει αποδοτικά στο ερώτημα αν υπάρχει κάποιο τσιπ σε μία επιφάνεια[G].

Ένα R-δέντρο είναι μια ισορροπημένη δομή δέντρου, με τα αντικείμενα να είναι στα φύλλα, όπως τα B^+-δέντρα. Χρησιμοποιείται για την οργάνωση ενός αριθμού γεωμετρικών αντικειμένων κ-διαστάσεων το καθένα, σε ορθογώνια περιβάλλοντα πλαίσια (*Minimum Bounding Rectangles* πιο γνωστών και ως *MBRs*) κ-διαστάσεων το καθένα. Στην περίπτωση των δύο διαστάσεων τα περιβάλλοντα πλαίσια έχουν σχήμα ορθογωνίου παραλληλογράμμου. Από αυτήν ακριβώς τους την ιδιότητα πήραν και το όνομα τους τα R-δέντρα. Το γράμμα "R" είναι για την αγγλική λέξη "*Rectangle*", δηλαδή ορθογώνιο.

Το περιβάλλον πλαίσιο κάθε κόμβου στο R-δέντρο, είναι το μικρότερο ορθογώνιο που περιέχει και είναι παράλληλο με τα περιβάλλοντα πλαίσια των παιδιών του. Τα φύλλα του R-δέντρου περιέχουν δείκτες σε αντικείμενα της βάσης δεδομένων όσο και τα περιβάλλοντα πλαίσια αυτών των αντικειμένων. Οι κόμβοι είναι υλοποιημένοι σαν σελίδες στον δίσκο. Να σημειώσουμε εδώ ότι τα περιβάλλοντα πλαίσια μπορούν να επικαλύπτονται μεταξύ τους. Επίσης κάθε περιβάλλον πλαίσιο μπορεί γεωγραφικά να ανήκει σε πολλούς κόμβους, αλλά θα συνδέεται μόνο με έναν ˙ έχει αναγκαστικά μόνο έναν πατέρα στο δέντρο.



Σχήμα *4.8 [MNPT]* – Παράδειγμα δύο επικαλυπτόμενων πλαισίων, όπου τα αντίστοιχα αντικείμενα δεν επικαλύπτονται

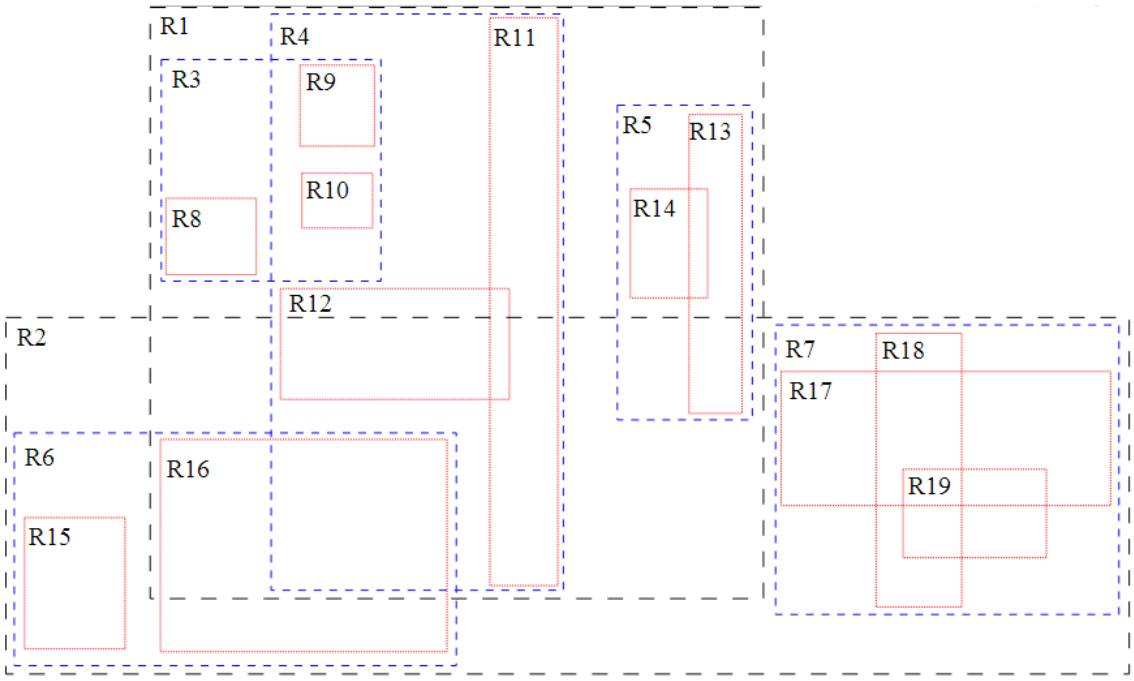
Τα περιβάλλοντα πλαίσια βοηθούν να επιταχυνθούν οι έλεγχοι για επικάλυψη των αντικειμένων, δηλαδή αν ένα ορθογώνιο δεν επικαλύπτεται με το περιβάλλον πλαίσιο ενός αντικειμένου, δεν μπορεί να επικαλύπτεται ούτε με το αντικείμενο. Βέβαια, όπως φαίνεται και στο σχήμα 4.8, δύο αντικείμενα μπορεί να μην επικαλύπτονται μεταξύ τους αλλά τα αντίστοιχα περιβάλλοντα πλαίσια να επικαλύπτονται. Σε αυτήν την περίπτωση χρειαζόμαστε επιπλέον έλεγχο για να δούμε αν τα αντικείμενα επικαλύπτονται.

Ένα R-δέντρο τάξεως (m,M) έχει τα ακόλουθα χαρακτηριστικά [*MNPT*]:

- Κάθε φύλλο μπορεί να έχει μέχρι M εγγραφές. Ο ελάχιστος αριθμός εγγραφών που μπορεί να έχει είναι m=M/2. Κάθε εγγραφή είναι της μορφής (*mbr*, *oid*). Το *mbr* αντιστοιχεί στο περιβάλλον πλαίσιο του αντικειμένου, ενώ το *oid* είναι δείκτης προς το ίδιο το αντικείμενο.
- Αντίστοιχα με τα φύλλα, οι εσωτερικοί κόμβοι έχουν από m=M/2 έως M εγγραφές. Κάθε εγγραφή είναι της μορφής (mbr, p). Το p είναι δείκτης προς το αντίστοιχο παιδί και mbr είναι το περιβάλλον πλαίσιο του παιδιού αυτού.
- Ο ελάχιστος αριθμός εγγραφών της ρίζα του δέντρου είναι 2, εκτός αν είναι φύλλο οπότε μπορεί να έχει μία ή και καμία εγγραφή.
- Όλα τα φύλλα του R-δέντρου βρίσκονται στο ίδιο επίπεδο.

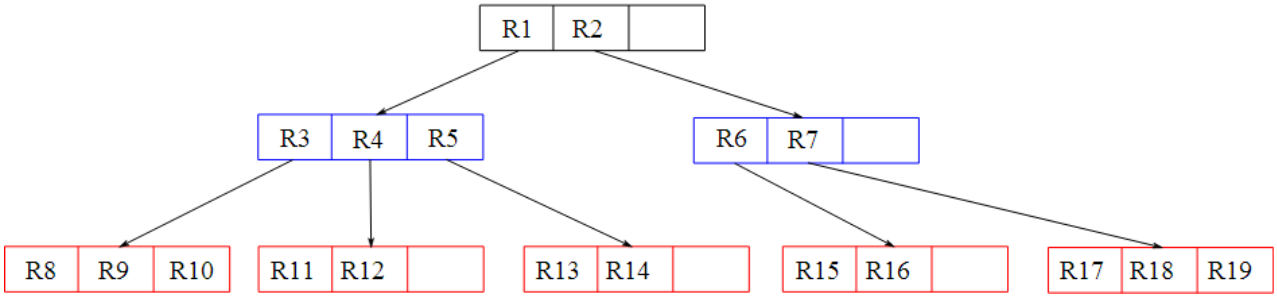
Από τον παραπάνω ορισμό του R-tree γίνεται κατανοητό ότι αυτό είναι ισορροπημένο ως προς το ύψος (*height-balanced tree*). Υποθέτοντας N γεωμετρικά αντικείμενα, το μέγιστο ύψος ενός R-δέντρου (έστω h) δίνεται από την σχέση:

$$h_{max} = \log_m N - 1$$



Σχήμα *4.9 [WRT]* – Παράδειγμα πλαισίων αντικειμένων, και τα πλαίσια αυτών των πλαισίων

Στο σχήμα 4.9 φαίνονται με κόκκινο χρώμα τα περιβάλλοντα πλαίσια κάποιων γεωμετρικών αντικειμένων – τα αντικείμενα δεν φαίνονται. Αυτά τα πλαίσια είναι τα R8-R19 και θα αποθηκευτούν στα φύλλα του R-δέντρου. Επίσης στην εικόνα φαίνονται με μπλε και μαύρο χρώμα τα περιβάλλοντα πλαίσια που περιβάλλουν τα προηγούμενα πλαίσια. Ουσιαστικά πρόκειται για τα περιβάλλοντα πλαίσια που θα αποθηκευτούν στους εσωτερικούς κόμβους του R-δέντρου. Υποθέτοντας M=3, το σχήμα 4.10 δείχνει το παραγόμενο R-δέντρο.



Σχήμα *4.10 [WRT]* – Το αντίστοιχο *R*-Δέντρο

Στην συνέχεια θα δούμε πως να χειριζόμαστε τις αναζητήσεις, τις εισαγωγές και τις διαγραφές σε ένα R-δέντρο.

Αναζήτηση

Δεδομένου ενός ορθογωνίου παραλληλογράμμου Q, ψάχνουμε όλα τα περιβάλλοντα πλαίσια αντικειμένων που επικαλύπτονται πλήρως ή μερικώς από το Q. Για μια εγγραφή ενός κόμβου στο R-δέντρο έστω e, το *e.p* είναι δείκτης προς το παιδί αυτής της εγγραφής και το *e.mbr* αντιπροσωπεύει το περιβάλλον πλαίσιο αυτού του παιδιού.

Να σημειώσουμε εδώ ότι ο παραπάνω αλγόριθμος βρίσκει τα αντικείμενα των οποίων τα περιβάλλοντα πλαίσια επικαλύπτονται από το *Q*. Για να βρούμε τα πραγματικά γεωμετρικά αντικείμενα που επικαλύπτονται *Q* πρέπει να κάνουμε ακόμα ένα βήμα.

Αλγόριθμος **Αναζήτηση** (Κόμβος RN, Πλαίσιο Q)

/**Βρίσκει όλα τα περιβάλλοντα πλαίσια με που είναι αποθηκευμένα κάτω από τον κόμβο RN και επικαλύπτονται από την περιοχή Q*/*

1. Αν το RN δεν είναι φύλλο
2. *Εξετάζουμε κάθε εγγραφή e του RN, για να βρούμε αυτές που το e.mbr επικαλύπτεται από το Q*
3. *Για αυτές τις εγγραφές καλούμε την* ***Αναζήτηση****(e.p, Q)*
4. Αλλιώς //Το RN είναι φύλλο
5. *Εξετάζουμε κάθε εγγραφή e του RN, για να βρούμε αυτές που το e.mbr επικαλύπτεται από το Q*
6. *Προσθέτουμε τις εγγραφές*
7. Τέλος Αν

Εισαγωγή

Για να εισάγουμε ένα αντικείμενο σε ένα R-δέντρο διασχίζουμε το δέντρο για να βρούμε το κατάλληλο φύλλο για την εισαγωγή αυτού του αντικειμένου. Η διαδικασία επιλογής του κατάλληλου φύλλου περιγράφεται αργότερα στον αλγόριθμο εισαγωγής. Αφού γίνει η εισαγωγή ενημερώνονται οι τιμές των περιβαλλόντων πλαισίων όλων των κόμβων του μονοπατιού, από το φύλλο που έγινε η εισαγωγή ως την ρίζα του δέντρου. Αν το φύλλο που διαλέξαμε για εισαγωγή του αντικειμένου είναι γεμάτος (περιέχει ήδη M εγγραφές), τότε ακολουθούμε μια διαδικασία που ονομάζεται διαίρεση (*split*) και χωρίζουμε το φύλλο σε δύο φύλλα.

Για την διαδικασία της διαίρεσης έχουν προταθεί διάφοροι αλγόριθμοι. Βασικό μέλημα κάθε τέτοιου αλγορίθμου είναι να μειώσει την πιθανότητα ο αλγόριθμος αναζήτησης να επισκεφτεί και τους δύο δημιουργηθέντες κόμβους.

Ο αλγόριθμος γραμμικής διαίρεσης (*linear split algorithm*) προσπαθεί να πετύχει αυτόν τον σκοπό ελαχιστοποιώντας το συνολικό εμβαδόν των δύο δημιουργηθέντων κόμβων. Αρχικά, επιλέγει τα δύο αντικείμενα που τα περιβάλλοντα πλαίσια τους απέχουν περισσότερο και τα χωρίζει σε δύο κόμβους. Στην συνέχεια αναθέτει κάθε άλλο αντικείμενο, σε τυχαία σειρά, σε έναν από τους δύο κόμβους. Ο κόμβος που επιλέγεται

είναι αυτός που υφίσταται την μικρότερη αύξηση του περιβάλλοντος πλαισίου του με την εισαγωγή του καινούργιου αντικειμένου. Η διαδικασία σταματάει όταν έχουν εκχωρηθεί όλα τα στοιχεία, ή όταν ένας από τους δύο κόμβους έχει αρκετά στοιχεία που όλα τα υπόλοιπα στοιχεία πρέπει να προστεθούν στον άλλον κόμβο, ώστε και οι δύο κόμβοι να έχουν την ελάχιστη χωρητικότητα m. Ο αλγόριθμος γραμμικής διαίρεσης είναι γραμμικής πολυπλοκότητας.

Αλγόριθμος **Εισαγωγή** (Κόμβος RN, Εγγραφή E)
/*Εισάγει μια εγγραφή E στο R-δέντρο με ρίζα τον κόμβο RN*/
1. Διασχίζουμε το δέντρο από την ρίζα RN για να βρούμε το κατάλληλο φύλλο για την εισαγωγή. Σε κάθε βήμα επιλέγουμε τον κόμβο, ο οποίος χρειάζεται την μικρότερη αύξηση στο περιβάλλον πλαίσιο του για να υποδεχτεί την E. Σε περίπτωση ισοπαλίας επιλέγουμε τον κόμβου του οποίου το περιβάλλον πλαίσιο έχει το μικρότερο εμβαδόν.
2. Αν το επιλεγμένο φύλλο δεν είναι γεμάτο
3. *Εισάγουμε το E στο φύλλο.*
4. *Ενημερώνουμε το μονοπάτι από το φύλλο στην ρίζα, ώστε όλα τα περιβάλλοντα πλαίσια να καλύπτουν το E.mbr.*
5. Αλλιώς //Το φύλλο είναι γεμάτο
6. *Εφαρμόζουμε έναν αλγόριθμο διαίρεσης στο φύλλο.*
7. *Ενημερώνουμε τα περιβάλλοντα πλαίσια όλων των κόμβων του μονοπατιού από τα δύο πλέον φύλλα έως την ρίζα.*
8. *Εφαρμόζουμε διαιρέσεις σε όσους κόμβους του μονοπατιού από τα φύλλα στην ρίζα είναι απαραίτητο.*
9. *Σε περίπτωση που η ρίζα πρέπει να χωριστεί δημιουργούμε καινούργια ρίζα και αυξάνουμε το ύψος του δέντρου.*
10. Τέλος Αν

Ο αλγόριθμος τετραγωνικής διαίρεσης (*quadratic split algorithm*) επιλέγει αρχικά δύο αντικείμενα, έτσι ώστε αν τα βάλει στον ίδιο κόμβο να καταλήξουμε σε ένα περιβάλλον πλαίσιο με όσο τον δυνατό περισσότερο αχρησιμοποίητο χώρο. Αχρησιμοποίητος χώρος είναι ο χώρος που περισσεύει από το περιβάλλον πλαίσιο αν αφαιρεθούν τα περιβάλλοντα πλαίσια των δύο αντικειμένων. Αυτά τα δύο αντικείμενα χωρίζονται σε δύο κόμβους. Στην συνέχεια από τα αντικείμενα που έχουν απομείνει, επιλέγουμε αυτό που η διαφορά του αχρησιμοποίητου χώρου αν το προσθέσουμε στους δύο κόμβους μεγιστοποιείται. Εισάγουμε το αντικείμενο αυτό στον κόμβο που υφίσταται την μικρότερη αύξηση του περιβάλλοντος πλαισίου του. Επαναλαμβάνουμε την διαδικασία και για τα υπόλοιπα αντικείμενα μέχρις ότου ένας από τους δύο κόμβους να έχει m-λ εγγραφές και περισσεύουν λ αντικείμενα για εισαγωγή. Εισάγουμε όλα τα αντικείμενα σε αυτόν τον κόμβο. Η πολυπλοκότητα του αλγορίθμου τετραγωνικής διαίρεσης είναι τετραγωνική.

Τέλος ο αλγόριθμος εκθετικής διαίρεσης (*exponential split algorithm*) δημιουργεί όλους

τους πιθανούς συνδυασμούς αντικειμένων και επιλέγει αυτόν που ελαχιστοποιεί το συνολικό εμβαδόν των περιβαλλόντων πλαισίων. Η πολυπλοκότητα του συγκεκριμένου αλγορίθμου είναι εκθετική.

Στα πλαίσια των περισσοτέρων εφαρμογών χρησιμοποιείται ο τετραγωνικός αλγόριθμος διαίρεσης, καθώς πετυχαίνει ικανοποιητική απόδοση τόσο στην ανάκτηση των αντικειμένων από το δέντρο όσο και στην απάντηση ερωτημάτων αναζήτησης.

Διαγραφή

Η διαγραφή ενός αντικειμένου από ένα R-δέντρο μπορεί να εκτελεστεί όπως μια διαγραφή σε ένα B^+-δέντρο, δανειζόμενη στοιχεία από αδερφικούς κόμβους ή συγχωνεύοντας τους. Αν και η διαγραφή συγχωνεύοντας κόμβους του ίδιου επιπέδου είναι εφικτή, για την διαγραφή στοιχείων από R-δέντρα έχει αναπτυχθεί μια διαφορετική προσέγγιση.

Αλγόριθμος **Διαγραφή**(Κόμβος RN, Εγγραφή E)
/*Διαγράφει την εγγραφή E από το R-δέντρο με ρίζα τον κόμβο RN*/
1. Ψάξε όλες τις εγγραφές του RN που καλύπτουν πλήρως το E.mbr
2. Ακολούθησε αυτές τις εγγραφές μέχρι να βρεθεί το φύλλο L που να περιέχει την E.
3. Διέγραψε το E από το L.
4. Αν μετά την διαγραφή το φύλλο L έχει λιγότερες από m εγγραφές
5. *Διέγραψε τελείως το φύλλο L.*
6. *Εισήγαγε όλες τις εγγραφές στο δέντρο μέσω του αλγορίθμου* ***Εισαγωγή***
7. Αλλιώς //Το φύλλο L έχει περισσότερες ή ίσες από m εγγραφές
8. *Ενημερώνουμε με τα καινούργια περιβάλλοντα πλαίσια το μονοπάτι από το φύλλο στην ρίζα.*
9. Τέλος Αν

Η διαγραφή αντικειμένων από ένα R-δέντρο με τον τρόπο που περιγράφεται στον παραπάνω αλγόριθμο, δηλαδή με την διαγραφή ολόκληρου του φύλλου και την εισαγωγή στο δέντρο των εγγραφών του, παρουσιάζει τα εξής πλεονεκτήματα σε σχέση με την διαγραφή μέσω συγχωνεύσεων (όπως στα B^+-δέντρα):

- Τα B^+-δέντρα χρησιμοποιούνται για αποθήκευση μονοδιάστατων δεδομένων και συνεπώς οι αδερφικοί κόμβοι έχουν συνεχόμενες καταχωρίσεις. Λόγω αυτού, οι συγχωνεύσεις γειτονικών κόμβων είναι εύκολο να γίνουν και η ποιότητα του δέντρου διατηρείται. Αντίθετα για δεδομένα πολλών διαστάσεων αυτή η ιδιότητα δεν ισχύει, και η διαγραφή αντικειμένων μέσω συγχώνευσης γειτονικών κόμβων μπορεί να οδηγήσει σε μείωση της ποιότητας του δέντρου.
- Όπως περιγράψαμε, ο αλγόριθμος εισαγωγής προσπαθεί να διατηρήσει την καλή ποιότητα του δέντρου, δηλαδή την ικανότητα του να ανταποκρίνεται αποδοτικά

σε ερωτήματα αναζήτησης. Λόγω αυτού, το να ξαναεισάγουμε δεδομένα είναι μια διαδικασία που βοηθάει στην συντήρηση της ποιότητας του δέντρου.

- Οι σελίδες του δίσκου που απαιτούνται για τις εισαγωγές στοιχείων στο δέντρο υπάρχουν ήδη στην ενδιάμεση μνήμη (*buffer memory*), καθώς προσπελάστηκαν κατά την αναζήτηση του προς διαγραφήν στοιχείου. Άρα η εισαγωγή στοιχείων στο δέντρο δεν σημαίνει περισσότερες προσπελάσεις στην μνήμη ούτε χειρότερη απόδοση.

Συνοψίζοντας λοιπόν, η αποτελεσματικότητα αποθήκευσης των R-δέντρων είναι πολύ καλή, αφού ένα αντικείμενο αποθηκεύεται μόνο μια φορά και μπορούμε να εξασφαλίσουμε ότι κάθε κόμβος είναι τουλάχιστον μισογεμάτος. Ωστόσο τα ερωτήματα μπορεί να εκτελούνται αργά αφού θα πρέπει να γίνει αναζήτηση σε πολλές διαδρομές (οι αλγόριθμοι διαίρεσης προσπαθούν να αυξήσουν την απόδοση αυτών των ερωτημάτων). Τέλος, λόγω της μεγάλης τους ομοιότητας με τα B-δέντρα, τα R-δέντρα έχουν αποδειχτεί δημοφιλείς σε βάσεις δεδομένων που υποστηρίζουν χωροταξικά δεδομένα.

5 Η πλατφόρμα Node.js

Το *Node.js* είναι ένα πρότζεκτ που παρουσίασε στο ευρωπαϊκό συνέδριο της *JavaScript* το 2009 (2009 European JSConf) στο Βερολίνο, ο προγραμματιστής *Ryan Dahl*. Το *Node.js* ή πιο απλά *Node*, είναι μια πλατφόρμα που μας επιτρέπει να γράψουμε *JavaScript* στην πλευρά του εξυπηρετητή (*server-side JavaScript*). Από το 2009 μέχρι σήμερα η πλατφόρμα έχει αναπτυχθεί ραγδαία και έχει χρησιμοποιηθεί σε πολλές μεγάλες εφαρμογές. Αποτελεί αυτήν την στιγμή το δεύτερο σε επισκεψιμότητα πρότζεκτ στο *GitHub* και μετράει περισσότερα από 90000 πακέτα (*modules*) δημοσιευμένα στον διαχειριστή πακέτων του *Node* (*Node Package Manager*, πιο γνωστού και ως *NPM*).

Η ιδέα ήταν να δημιουργηθεί μια οδηγούμενη από τα γεγονότα (*event-driven*) πλατφόρμα, που θα χρησιμοποιεί λειτουργίες Εισόδου/Εξόδου χωρίς αναμονή (*non blocking I/O*) - ή ισοδύναμα θα χρησιμοποιεί ασύγχρονη Ε/Ε (*asynchronous I/O*) - με σκοπό να αυξήσει την απόδοση και την δυνατότητα κλιμάκωσης (*scalability*) διαδικτυακών εφαρμογών πραγματικού χρόνου. Το *Node* χρησιμοποιεί την *V8*, μια εικονική μηχανή που αναπτύχθηκε από την *Google* για τον *Google Chrome* και τρέχει κώδικα *JavaScript*. Η *V8* δίνει στο *Node* μεγάλη αύξηση στην απόδοση, καθώς κάνει απευθείας μεταγλώττιση σε γλώσσα μηχανής αντί να εκτελεί *bytecode* ή να χρησιμοποιεί διερμηνευτή (*interpreter*).

Για αρκετό καιρό, ήταν γνωστό στην προγραμματιστική κοινότητα ότι η ανάπτυξη ενός εξυπηρετητή σε μια πλατφόρμα που χρησιμοποιεί ασύγχρονες, οδηγούμενες από τα γεγονότα λειτουργίες Ε/Ε θα ήταν πολύ αποδοτική (τόσο από άποψη ταχύτητας όσο και απαιτήσεων μνήμης). Την γνώση αυτήν, αξιοποίησαν διάφορες γνωστές πλατφόρμες και γλώσσες προγραμματισμού. Μερικές από τις διασημότερες υλοποιήσεις είναι το *Event Machine* της *Ruby*, το *AnyEvent* της *Perl* και το *Twisted* της *Python*.

Η υλοποίηση όμως μιας εφαρμογής σε ένα από αυτά τα πλαίσια απαιτεί τόσο την ειδική γνώση του ίδιου του πλαισίου όσο και την ειδική γνώση των βιβλιοθηκών του. Για παράδειγμα στο *Event Machine*, για να πετύχουμε λειτουργίες Ε/Ε χωρίς αναμονή περιοριζόμαστε στην χρησιμοποίηση μόνο ασύγχρονων βιβλιοθηκών φτιαγμένων ειδικά για το *Event Machine*. Αν χρησιμοποιήσουμε κάποια σύγχρονη βιβλιοθήκη, όπως οι περισσότερες βιβλιοθήκες της *Ruby*, η απόδοση του εξυπηρετητή μας θα πέσει, καθώς θα αναμένουμε συνεχώς κάποια λειτουργία Ε/Ε. Αντίθετα, το *Node* δημιουργήθηκε ως μια πλατφόρμα εξυπηρετητή με λειτουργίες Ε/Ε χωρίς αναμονή. Έτσι οποιαδήποτε λειτουργία στο *Node* είναι χωρίς αναμονή (*non blocking*) από την φύση της, χωρίς εμείς να χρειάζεται να χρησιμοποιήσουμε κάποια εξειδικευμένη βιβλιοθήκη για αυτό το σκοπό. Αυτός είναι ένας από τους βασικούς λόγους της μεγάλης αποδοχής που έτυχε το συγκεκριμένο πρότζεκτ.

5.1 Η πλευρά του Εξυπηρετητή

5.1.1 Η Javascript στον Εξυπηρετητή

Όπως αναφέραμε, το *Node* μας επιτρέπει να γράψουμε *JavaScript* στην πλευρά του εξυπηρετητή (*server-side JavaScript*). Η *JavaScript*, αν και παραδοσιακά έτρεχε αποκλειστικά στους φυλλομετρητές (*browsers*), τα τελευταία χρόνια έχει ξεκινήσει να χρησιμοποιείται και έξω από αυτούς. Το γεγονός ότι το *Node* μας δίνει την δυνατότητα να χρησιμοποιήσουμε την ίδια γλώσσα και στις δύο άκρες, τόσο στον εξυπηρετητή (*server*) όσο και στον εξυπηρετούμενο (*client*), απλοποιεί την ανάπτυξη διαδικτυακών εφαρμογών. Με αυτόν τον τρόπο, η γνώση *JavaScript* γίνεται το μόνο προαπαιτούμενο για να αναπτύξει κανείς μια τέτοια εφαρμογή.

Η χρήση *JavaScript* και στις δύο άκρες, όχι μόνο διευκολύνει την ανάπτυξη διαδικτυακών εφαρμογών αλλά ταυτόχρονα αυξάνει και την απόδοση τους. Από την ημέρα που κυκλοφόρησε ο φυλλομετρητής *Google Chrome* το 2008, η ταχύτητα εκτέλεσης της *JavaScript* αυξάνεται με απίστευτα γρήγορους ρυθμούς. Αυτό οφείλεται κυρίως στον ανταγωνισμό μεταξύ των εταιριών που παρέχουν φυλλομετρητές (*Mozilla, Microsoft, Apple, Opera* και *Google*). Καταλαβαίνουμε λοιπόν, το μεγάλο πλεονέκτημα απόδοσης που έχουμε χρησιμοποιώντας *JavaScript* και στην πλευρά του εξυπηρετητή.

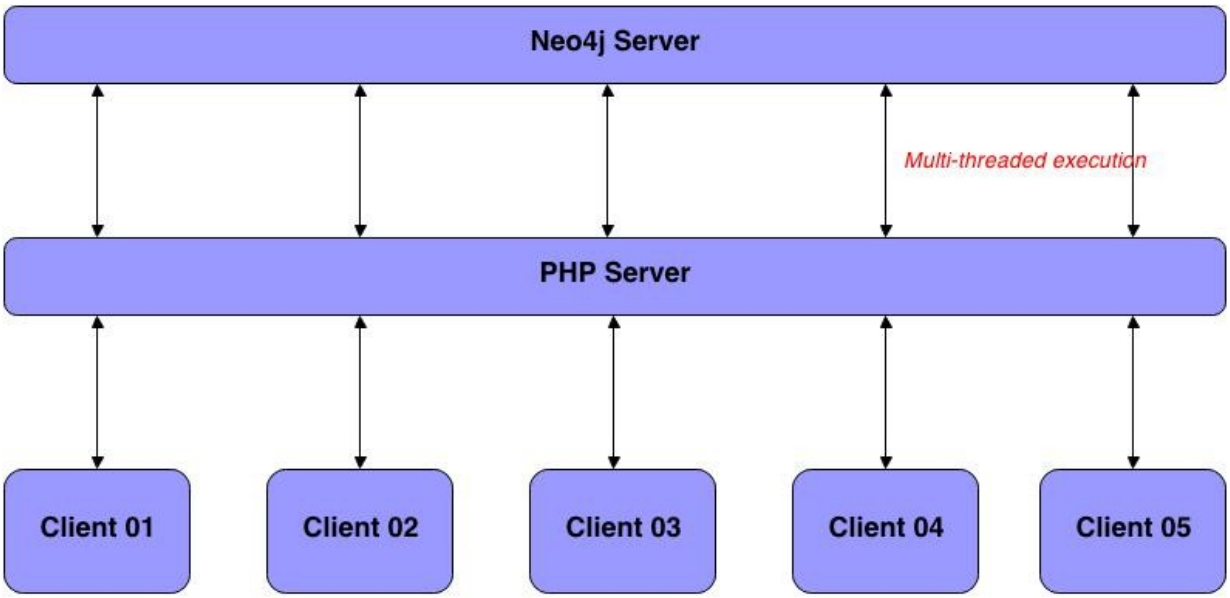
Δεν είναι όμως αυτοί οι βασικοί λόγοι για τους οποίους το *Node* χρησιμοποιεί *JavaScript*. Αρχικά, ο δημιουργός του *Node Ryan Dahl*, ξεκίνησε να φτιάχνει μια πλατφόρμα σε C. Η ιδέα εγκαταλείφθηκε καθώς η διατήρηση του περιβάλλοντος λειτουργίας (*context*) κατά την κλήση των συναρτήσεων ήταν μια αρκετά πολύπλοκη διαδικασία που οδήγησε και σε πολύπλοκο κώδικα. Η επόμενη επιλογή ήταν η *Lua*, η οποία όμως διέθετε αρκετές σύγχρονες βιβλιοθήκες Ε/Ε. Ο συνδυασμός αυτός, των σύγχρονων και των ασύγχρονων βιβλιοθηκών, θεωρήθηκε ότι θα μπέρδευε τους προγραμματιστές και θα τους απέτρεπε από την χρήση της πλατφόρμας.

Τελικά επιλέχτηκε η *JavaScript*. Η *JavaScript* δεν διαθέτει κάποια σύγχρονη βιβλιοθήκη για λειτουργίες Ε/Ε. Αυτό έδωσε την δυνατότητα στον *Dahl* να ορίσει έναν ασύγχρονο, οδηγούμενο από τα γεγονότα τρόπο με τον οποίο θα γίνονται οι λειτουργίες Ε/Ε. Η *JavaScript* λοιπόν είναι ο λόγος, που κάθε λειτουργία στην πλατφόρμα *Node* είναι από την φύση της χωρίς αναμονή (*non blocking*).

Στην *JavaScript* οι συναρτήσεις είναι αντικείμενα πρώτης τάξης (*first-class functions*). Αυτό σημαίνει ότι η γλώσσα υποστηρίζει το πέρασμα μιας συνάρτησης σαν παράμετρο σε μια άλλη συνάρτηση και την επιστροφή μιας συνάρτησης σαν αποτέλεσμα από μια άλλη συνάρτηση. Επίσης η *JavaScript* υποστηρίζει το μοτίβο κλεισίματος (*closure pattern*). Με αυτόν τον τρόπο οι συναρτήσεις επιστρεφόμενης κλήσης (*callback functions*) έχουν την ιδιότητα να θυμούνται, σχεδόν με μαγικό τρόπο, το περιβάλλον λειτουργίας στο οποίο δηλώθηκαν καθώς και τις μεταβλητές που υπήρχαν σε αυτό το περιβάλλον. Αυτά τα δύο χαρακτηριστικά της *JavaScript* βρίσκονται στην καρδιά της επιτυχίας του *Node*, καθώς (όπως θα δούμε και στην συνέχεια) δίνουν στην πλατφόρμα την δυνατότητα να πετύχει ασύγχρονες, οδηγούμενες από τα γεγονότα λειτουργίες Ε/Ε.

5.1.2 Το βασισμένο στα Νήματα Μοντέλο

Οι παραδοσιακές προγραμματιστικές τεχνικές εμποδίζουν την συνέχιση της εκτέλεσης μιας διεργασίας μέχρι μια λειτουργία Ε/Ε να ολοκληρωθεί. Αυτό το προγραμματιστικό μοντέλο έχει τις ρίζες του στα συστήματα καταμερισμού χρόνου (*time sharing systems*), όπου η Κεντρική Μονάδα Επεξεργασίας ή ΚΜΕ (*CPU*) εκτελεί πολλές εργασίες εναλλάσσοντας τες με μεγάλη συχνότητα [*SGG*]. Αντιπροσωπεύοντας κάθε χρήστη με μια διεργασία, σε αυτά τα συστήματα ένας χρήστης πρέπει να περιμένει να τελειώσει μια λειτουργία Ε/Ε μέχρι να αποφασίσει την επόμενη λειτουργία που θέλει να εκτελέσει. Σκεφτείτε την άσχημη εμπειρία ενός χρήστη που δεν μπορεί να αλληλεπιδράσει με ένα σύστημα καθώς περιμένει μια λειτουργία Ε/Ε να ολοκληρωθεί (πχ. να φορτώσει ένα βίντεο).



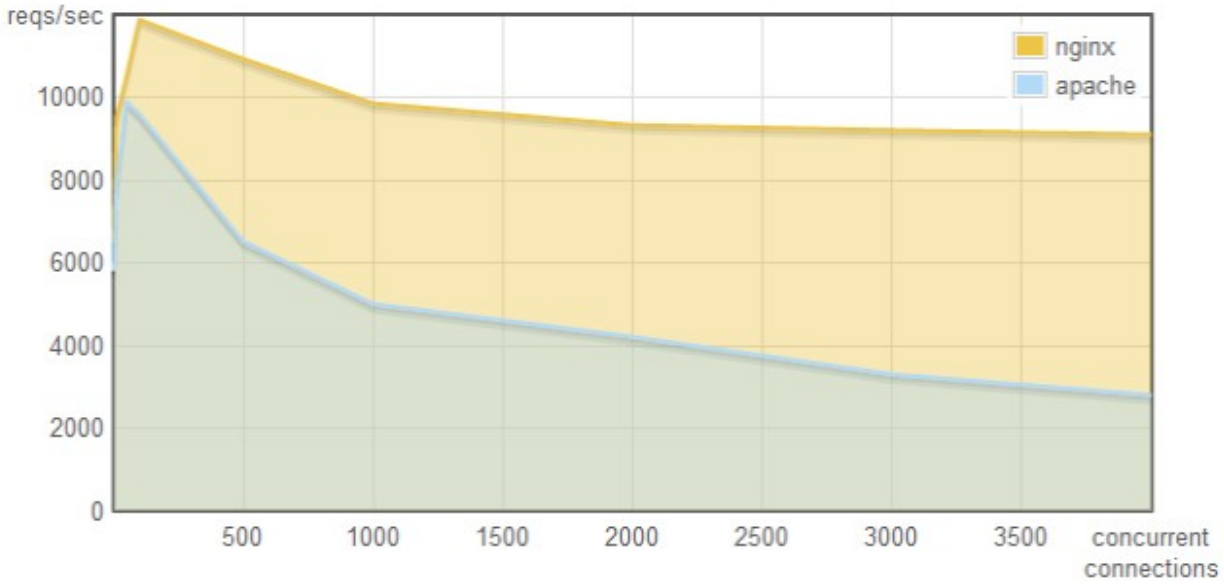
Σχήμα *5.1* – Το πολυνηματικό μοντέλο ενός *PHP* εξυπηρετητή

Μια εναλλακτική σε αυτό το προγραμματιστικό μοντέλο είναι ο πολυνηματισμός (*multi-threading*). Νήμα (*thread*) είναι ένα είδος ελαφριάς διεργασίας που μπορεί να χρησιμοποιήσει την ΚΜΕ και μοιράζεται την μνήμη με κάθε άλλο νήμα μέσα σε μια διεργασία [*SGG*]. Τα νήματα προστέθηκαν στο παραπάνω μοντέλο για να μπορέσουν να ξεμπλοκάρουν μια διαδικασία από μια λειτουργία Ε/Ε. Όσο ένα νήμα περιμένει μια λειτουργία Ε/Ε να ολοκληρωθεί, ένα άλλο νήμα της ίδιας διεργασίας μπορεί να χρησιμοποιήσει την ΚΜΕ. Μόλις η λειτουργία Ε/Ε ολοκληρωθεί, το πρώτο νήμα ξυπνάει και το δεύτερο σταματάει. Σε συστήματα με πολλές ΚΜΕ διάφορα νήματα μπορούν να εκτελούνται ταυτόχρονα. Στο σχήμα 5.1 φαίνεται ένας εξυπηρετητής βασισμένος σε νήματα˙ για κάθε εισερχόμενο αίτημα ο εξυπηρετητής δημιουργεί και ένα καινούργιο νήμα.

Ο πολυνηματισμός λύνει το πρόβλημα της απόδοσης και των μεγάλων νεκρών διαστημάτων της πρώτης υλοποίησης αλλά απαιτεί μεγάλο φόρτο εργασίας από τον

προγραμματιστή. Ο προγραμματιστής θα πρέπει να χρησιμοποιεί διάφορες λύσεις συγχρονισμού (*synchronization primitives*), όπως κλειδώματα (*locks*) και σημαφόρους (*semaphores*), για να αποτρέψει την ταυτόχρονη πρόσβαση δύο νημάτων στο ίδιο σημείο της μνήμης [*SGG*]. Με αυτόν τον τρόπο αναγκάζεται να προβλέψει κάθε πιθανή σειρά που τα διάφορα νήματα θα εκτελεστούν ώστε να αποτρέψει πιθανά προβλήματα. Τέτοιου είδους λάθη (*bugs*) συχνά συμβαίνουν τυχαία, και είναι πολύ δύσκολο να εντοπιστούν.

Πέρα από την δυσκολία προγραμματισμού του, το μοντέλο βασισμένο στα νήματα εξακολουθεί να μην κλιμακώνει (*scale*) καλά καθώς ο αριθμός των χρηστών αυξάνει. Η διαχείριση όλων αυτών των νημάτων είναι ένα μεγάλο βάρος για το λειτουργικό σύστημα, τόσο λόγω των απαιτήσεων μνήμης όσο και λόγω της εναλλαγής περιβάλλοντος λειτουργίας (*context switch*). Η εναλλαγή της ΚΜΕ ανάμεσα στα νήματα απαιτεί την αποθήκευση της κατάστασης του τρέχοντος νήματος και την επαναφορά της κατάστασης του προς εκτέλεσης νήματος. Αυτό είναι γνωστό ως εναλλαγή περιβάλλοντος λειτουργίας. Λόγω αυτών των περιορισμών η απόδοση ενός τέτοιου συστήματος αρχίζει να φθίνει μετά από κάποιον αριθμό νημάτων.

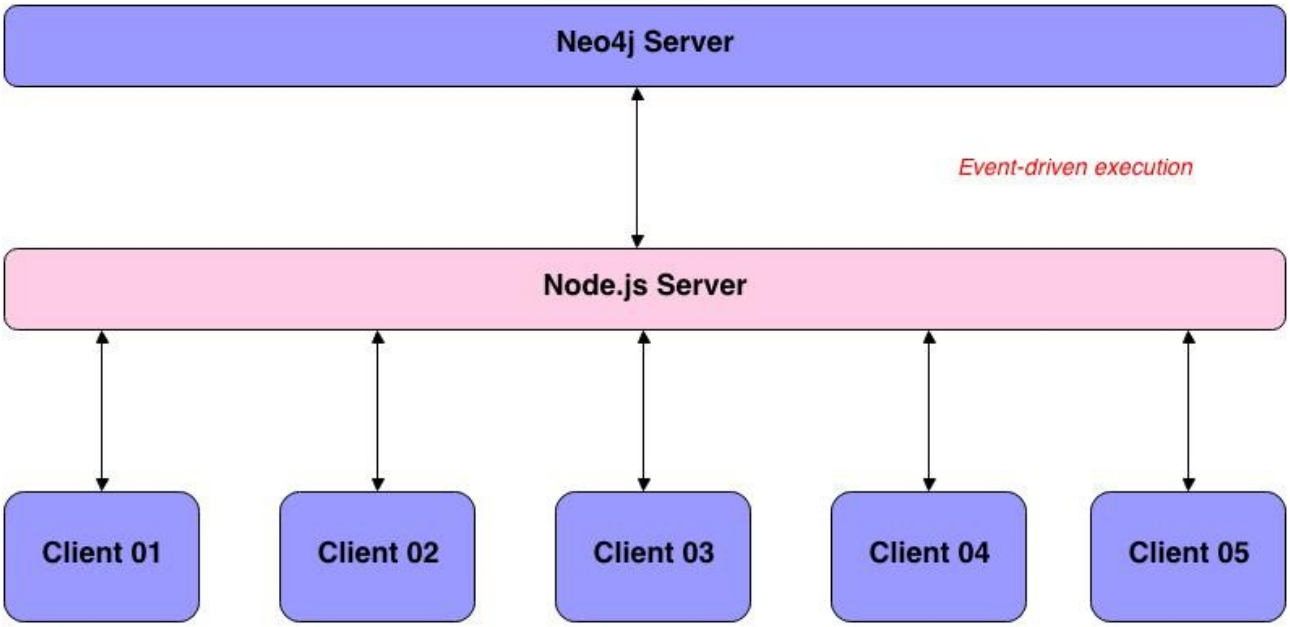


Σχήμα *5.2 [NGAP]* – Σύγκριση των εξυπηρετητών *NGINX* και *Apache*

Η σύγκριση του *Apache* με τον *NGINX* [*NGAP*] φανερώνει την επίδραση της εναλλαγής περιβάλλοντος λειτουργίας στην απόδοση ενός τέτοιου συστήματος. Ο *NGINX* είναι ένας *HTTP* εξυπηρετητής (*HTTP server*), όπως ο *Apache*. Αντί όμως να χρησιμοποιεί την πολυνηματική προσέγγιση με λειτουργίες Ε/Ε με αναμονή (blocking I/O), χρησιμοποιεί ασύγχρονη Ε/Ε (όπως και το *Node*). Όπως φαίνεται στο σχήμα 5.2, ο *NGINX* μπορεί να διαχειριστεί περισσότερα αιτήματα από τον *Apache* για ίδιο αριθμό συνδέσεων. Αυτό μεταφράζεται σε πολύ καλύτερη απόδοση καθώς για αριθμό συνδέσεων που πλησιάζουν τις 3500 ο *NGINX* μπορεί να εξυπηρετήσει τα αιτήματα 3 φορές γρηγορότερα.

Πέρα από τους περιορισμούς στην ταχύτητα εξυπηρέτησης αιτημάτων, που οφείλονται στην εναλλαγή περιβάλλοντος λειτουργίας, το πολυνηματικό μοντέλο αντιμετωπίζει και περιορισμούς μνήμης. Υποθέτοντας για παράδειγμα ότι κάθε νήμα μπορεί δυνητικά να

δεσμεύσει *2MB* μνήμης, και τρέχει σε ένα σύστημα με *8GB RAM*, μας θέτει ένα άνω όριο 4000 ταυτόχρονων συνδέσεων, χωρίς να υπολογίζουμε το κόστος εναλλαγής περιβάλλοντος λειτουργίας. Το *Node* ακολουθώντας μια τελείως διαφορετική προσέγγιση, αποφεύγει όλους αυτούς τους περιορισμούς δημιουργώντας μόνο ένα νήμα το οποίο χειρίζεται όλες τις λειτουργίες Ε/Ε. Αυτό φαίνεται στο σχήμα 5.3. Με αυτόν τον τρόπο πετυχαίνει επίπεδα κλιμάκωσης (*scalability*) που μπορεί να φτάσουν τις ένα εκατομμύριο ταυτόχρονες συνδέσεις [*BLC*].



Σχήμα *5.3* – Το μονονηματικό, οδηγούμενο από τα γεγονότα μοντέλο του *Node.js* εξυπηρετητή

5.1.3 Το Βασισμένο στα Γεγονότα Μοντέλο

Ο οδηγούμενος από τα γεγονότα (*event-driven*) προγραμματισμός είναι ένα προγραμματιστικό μοντέλο όπου η ροή της εκτέλεσης "οδηγείται" από τα γεγονότα. Ο χειρισμός των γεγονότων αυτών γίνεται από τους χειριστές συμβάντος (*event handlers*) ή τις επιστρεφόμενες κλήσεις γεγονότων (*event callbacks*). Μια επιστρεφόμενη κλήση είναι μια συνάρτηση η οποία καλείται όταν συμβαίνει ένα γεγονός. Στα πλαίσια του εξυπηρετητή το γεγονός αυτό είναι συνήθως η ολοκλήρωση μιας λειτουργίας Ε/Ε.

Στο παρακάτω παράδειγμα, φαίνεται ένα ερώτημα (*query*) στην *Neo4j* γραμμένο σε *PHP*. Η *PHP* είναι μια γλώσσα προγραμματισμού για την πλευρά του εξυπηρετητή που χρησιμοποιεί λειτουργίες Ε/Ε με αναμονή (*blocking I/O*).

```
$result = query('MATCH (n) RETURN n AS total');
print_r($result);
```

Ο παραπάνω κώδικας εκτελεί μια λειτουργία Ε/Ε. Η εκτέλεση σταματάει και συνεχίζεται

μόλις ολοκληρωθεί αυτή η λειτουργία. Αυτό μπορεί να πάρει από δευτερόλεπτα μέχρι λεπτά ανάλογα με την καθυστέρηση αυτής της λειτουργίας (*latency of the I/O operation*).

Αντίθετα, στα οδηγούμενα από τα γεγονότα συστήματα αυτό το ερώτημα θα γραφόταν όπως φαίνεται παρακάτω:

```
query_finish = function (result){
     console.log(result);
}
query('MATCH (n) RETURN n AS total', query_finish);
```

Οι παραπάνω γραμμές κώδικα μπορεί να φαίνονται παράξενες σε έναν αναγνώστη που δεν είναι εξοικειωμένος με την *JavaScript*. Εδώ ορίζουμε τι θα συμβεί μόλις ολοκληρωθεί το ερώτημα και το αποθηκεύουμε στην συνάρτηση "*query_finish*". Μετά περνάμε αυτήν την συνάρτηση σαν όρισμα στο ερώτημα μας. Όταν αυτό ολοκληρωθεί, θα καλέσει την συνάρτηση "*query_finish*" με παράμετρο το αποτέλεσμα, αντί απλά να επιστρέψει το αποτέλεσμα. Με αυτόν τον τρόπο μπορούμε να στρέψουμε την ροή εκτέλεσης σε άλλα πράγματα αντί απλά να περιμένουμε την λειτουργία E/E να ολοκληρωθεί. Μόλις ολοκληρωθεί θα ενημερωθούμε μέσω της συνάρτησης επιστρεφόμενης κλήσης (*callback function*), εδώ την "*query_finish*".

Αυτός ο τρόπος προγραμματισμού όπου οι λειτουργίες E/E δεν επιστρέφουν απλά μια τιμή αλλά συνάρτηση ονομάζεται προγραμματισμός οδηγούμενος από τα γεγονότα ή ασύγχρονος προγραμματισμός. Με αυτόν τον τρόπο πολλές λειτουργίες E/E μπορούν να συμβαίνουν παράλληλα, και κάθε φορά η συνάρτηση επιστρεφόμενης κλήσης μας ενημερώνει για την ολοκλήρωση της αντίστοιχης λειτουργίας.

Η *JavaScript* λόγω του τρόπου που αντιμετωπίζει τις συναρτήσεις, ταιριάζει φυσικά σε αυτό το είδος προγραμματισμού. Στην ενότητα 5.1.1 αναφέρθηκε ότι η *JavaScript* υποστηρίζει το μοτίβο κλεισίματος (*closure pattern*) καθώς και το πέρασμα συναρτήσεων σαν παραμέτρους σε άλλη συνάρτηση (*first-class functions*). Λόγω αυτών των χαρακτηριστικών, ο ορισμός συναρτήσεων επιστρεφόμενης κλήσης είναι μια πολύ εύκολη διαδικασία στην *JavaScript*.

Ο οδηγούμενος από τα γεγονότα προγραμματισμός συνοδεύεται από έναν βρόχο γεγονότων (*event loop*). Ο βρόχος γεγονότων είναι μια κατασκευή που είναι υπεύθυνη για 2 πράγματα σε κάθε επανάληψη: την ανίχνευση γεγονότων (*event detection*) και την ενεργοποίηση του αντίστοιχου χειριστή συμβάντος (*event handler triggering*). Ο βρόχος γεγονότων λοιπόν, ελέγχει σε κάθε επανάληψη ποια λειτουργία E/E έχει ολοκληρωθεί και καλεί την αντίστοιχη συνάρτηση επιστρεφόμενης κλήσης. Ο βρόχος γεγονότων είναι απλά ένα νήμα μέσα σε μια διεργασία και συνεπώς έχει τα εξής 2 χαρακτηριστικά [*PT*]:

- Κάθε χρονική στιγμή τρέχει το πολύ ένας διαχειριστής γεγονότος.
- Κάθε διαχειριστής γεγονότος τρέχει χωρίς διακοπή μέχρι να ολοκληρωθεί.

5.1.4 Ο Διαχειριστής Πακέτων του Node – NPM

Ο διαχειριστής πακέτων του *Node* (*Node Package Manager* ή *NPM*) μας δίνει την δυνατότητα να κατεβάσουμε, να εγκαταστήσουμε και να χρησιμοποιήσουμε στην εφαρμογή μας πακέτα (*modules*) τρίτων. Στα πρώτα στάδια της ζωής του *Node*, ο διαχειριστής πακέτων απαιτούσε ξεχωριστή εγκατάσταση. Από την έκδοση 0.6.0 του *Node*, η εγκατάσταση του διαχειριστή γίνεται αυτόματα μαζί με την εγκατάσταση της πλατφόρμας.

Με την εγκατάσταση του *Node*, είμαστε σε θέση να χρησιμοποιήσουμε κάποια ενσωματωμένα πακέτα (*built-in modules*), τα οποία μας παρέχουν κάποιες διεπαφές εφαρμογών χαμηλού επιπέδου (*low-level APIs*) ώστε να μπορούμε να υλοποιήσουμε κάποιες βασικές λειτουργίες που χρειάζονται οι εφαρμογές. Αυτά τα πακέτα είναι γνωστά ως ο πυρήνας του *Node* (*Node core*) [*CHHR*]. Στα πλαίσια όμως σύνθετων εφαρμογών είναι σχεδόν απαραίτητο να συμπεριλάβουμε πακέτα τρίτων.

Ο διαχειριστής πακέτων του *Node* διατηρεί μια κεντρική αποθήκη όλων των πακέτων που οι προγραμματιστές δημοσιοποιούν και μας παρέχει ένα εργαλείο γραμμής-εντολών (*command-line tool*) ώστε να μπορούμε να κατεβάζουμε, να εγκαθιστούμε και να διαχειριζόμαστε αυτά τα πακέτα. Η αναζήτηση τέτοιων πακέτων μπορεί να γίνει τόσο μέσω της γραμμής εντολών (εντολή *search*) όσο και μέσω της σελίδας www.npmjs.org. Σε αυτήν την σελίδα μπορούμε να δούμε και διάφορα στατιστικά στοιχεία, όπως πόσα πακέτα υπάρχουν συνολικά, από ποια πακέτα εξαρτώνται περισσότερο άλλα πακέτα και ποια πακέτα δημοσιοποιήθηκαν πιο πρόσφατα.

Εγκατάσταση Πακέτων

Αφού βρούμε το πακέτο που θέλουμε, το επόμενο βήμα είναι να το εγκαταστήσουμε. Αυτό γίνεται με δύο τρόπους χρησιμοποιώντας τον διαχειριστή πακέτων: ολικά (*globally*) ή τοπικά (*locally*).

Η ολική λειτουργία (*global mode*), είναι κατάλληλη για την εγκατάσταση πακέτων που θέλουμε να τα έχουμε διαθέσιμα σε όλες τις εφαρμογές μας. Για να εγκαταστήσουμε ένα πακέτο ολικά εκτελούμε την εντολή:

```
$ npm install -g package_name
```

Μετά την ολοκλήρωση της εγκατάστασης το πακέτο θα αποθηκευτεί στον εξής κατάλογο (*directory*) αν χρησιμοποιούμε *linux*:

```
/usr/local/lib/node_modules/
```

Για να χρησιμοποιήσουμε ένα πακέτο στην εφαρμογή μας εκτελούμε την εντολή:

```
require('package_name');
```

Με αυτόν τον τρόπο το *Node* θα πάει και θα φέρει το πακέτο από τον κατάλογο που αναφέραμε προηγουμένως ˙ εκτός αν έχουμε εγκαταστήσει το ίδιο πακέτο και τοπικά, οπότε το *Node* προτιμάει το τοπικό πακέτο.

Η τοπική λειτουργία (*local mode*) είναι η προκαθορισμένη λειτουργία του διαχειριστή πακέτων. Εδώ η λειτουργία περιορίζεται στον κατάλογο (*directory*) που βρισκόμαστε και δεν μπορούμε να πειράξουμε τις ολικές ρυθμίσεις και να κάνουμε αλλαγές σε ολόκληρο το σύστημα. Με αυτόν τον τρόπο μπορούμε να εγκαταστήσουμε πακέτα από τα οποία εξαρτάται η εφαρμογή μας, χωρίς να επηρεάσουμε άλλες εφαρμογές του συστήματος μας. Η εντολή εγκατάστασης είναι ίδια με την ολική λειτουργία χωρίς την σημαία `-g`. Με την εγκατάσταση το πακέτο θα αποθηκευτεί στον φάκελο `node_modules` κάτω από τον κατάλογο που βρισκόμαστε.

Πολλές φορές ένα πακέτο που θέλουμε να εγκαταστήσουμε εξαρτάται από άλλα πακέτα τα οποία δεν έχουμε εγκατεστημένα. Ο διαχειριστής πακέτων εγκαθιστά τόσο το πακέτο που ζητάμε μέσω της εντολής `npm install`, όσο και όλα τα πακέτα από τα οποία αυτό εξαρτάται. Έστω για παράδειγμα ότι θέλουμε να εγκαταστήσουμε το πακέτο Α που εξαρτάται από τα πακέτα Β και Γ. Το Node θα εγκαταστήσει τα πακέτα Β και Γ και θα τα βάλει στον κατάλογο `./node_modules/A/node_modules`.

Στα πλαίσια της εφαρμογής μας χρειαστήκαμε και εγκαταστήσαμε τοπικά τα εξής πακέτα:

1. ***Socket.IO*** : Το *Socket.IO*[4] είναι αναμφισβήτητα το πιο γνωστό πακέτο της κοινότητας του *Node*. Μας επιτρέπει να αναπτύξουμε διαδικτυακές εφαρμογές πραγματικού χρόνου, απαλλάσσοντας μας από τους περιορισμούς του *HTTP* πρωτοκόλλου. Ουσιαστικά δημιουργεί ένα κανάλι επικοινωνίας διπλής κατεύθυνσης μεταξύ του εξυπηρετούμενου (*client*) και του εξυπηρετητή (*server*). Χρησιμοποιεί το πρωτόκολλο διαδικτυακών υποδοχών (*WebSocket protocol*), αλλά και άλλες τεχνικές (όπως η *long polling*) για φυλλομετρητές που δεν υποστηρίζουν αυτό το πρωτόκολλο. Το *Socket.IO* μας παρέχει επίσης, μέσω μιας βιβλιοθήκης, μια διεπαφή για να αναπτύξουμε την εφαρμογή μας με παρόμοιο τρόπο και στην πλευρά του εξυπηρετούμενου.
2. ***neo4j*** : Το *neo4j* [*GNN*] μας παρέχει μια διεπαφή εφαρμογών που λειτουργεί σαν περιτύλιγμα (*wrapper*) στην *REST* διεπαφή της βάσης δεδομένων γράφου *Neo4j*. Κάποια χαρακτηριστικά της *Neo4j*, όπως οι ετικέτες (*labels*), δεν υποστηρίζονται από την διεπαφή παρά μόνο μέσω της γλώσσας ερωτημάτων *Cypher*.

5.1.5 Ο Εξυπηρετητής

Το *Node* κάνει την δημιουργία διαφόρων τύπων εξυπηρετητών (*servers*) μια πολύ εύκολη διαδικασία. Στο *Node* ο εξυπηρετητής και η εφαρμογή είναι το ίδιο. Αυτό μπορεί να μοιάζει παράξενο σε κάποιον που έχει συνηθίσει να έχει έναν εξυπηρετητή πάνω στον οποίο τρέχει την εφαρμογή του (όπως ένας *Apache HTTP* εξυπηρετητής τρέχει μια *PHP* εφαρμογή).

Στα πλαίσια της εφαρμογής μας θα δημιουργήσουμε έναν *HTTP* εξυπηρετητή, ο οποίος θα εξυπηρετεί ένα στατικό *HTML* αρχείο. Αρχικά πρέπει να φορτώσουμε το αρχείο μας

4 http://socket.io

από τον δίσκο. Αυτό γίνεται χρησιμοποιώντας το ενσωματωμένο πακέτο *'fs'*. Αυτό το πακέτο μας παρέχει μια διεπαφή (*API*) ώστε να μπορούμε να αλληλεπιδράσουμε με το σύστημα αρχείων (*file system*).

Στην παρακάτω υλοποίηση παρατηρούμε έντονα τον οδηγούμενο από τα γεγονότα (*event-driven*) τρόπο προγραμματισμού. Η συνάρτηση επιστρεφόμενης κλήσης (*callback function*) θα εκτελεστεί μόλις το αρχείο *client.html* έχει φορτωθεί, και περνάει τα περιεχόμενα του στην μεταβλητή *page*.

```
var fs = require('fs');
var page;

fs.readFile("client.html", 'utf-8', function (error, data){
     if (error){
          console.error('Error Reading html file', error);
     }else{
          page=data;
     }
});
```

Στην συνέχεια πρέπει να φτιάξουμε τον εξυπηρετητή μας. Για αυτόν τον σκοπό θα χρησιμοποιήσουμε το ενσωματωμένο πακέτο *'http'* του *Node*. Αυτό το πακέτο μας παρέχει μια διεπαφή για λειτουργίες *HTTP* τόσο στον εξυπηρετητή και όσο στον εξυπηρετούμενο.

```
var http = require('http');

var app = http.createServer(function (request, response) {
     response.writeHead(200, {'Content-Type': 'text/html'});
     //200 stands for OK http status code
     response.write(page);
     response.end();
});
app.listen(3000, function(){
     console.log('Server running at http://127.0.0.1:3000');
});
//Server listens to 3000 port for requests on
```

Για τον *HTTP* εξυπηρετητή, η συνάρτηση επιστρεφόμενης κλήσης περνάει σαν παράμετρος στην μέθοδο *createServer* και ορίζει, πως κάθε εισερχόμενο *HTTP* αίτημα (*HTTP request*) πρέπει να εξυπηρετηθεί. Αυτή η συνάρτηση δέχεται δύο παραμέτρους: την *request* και την *response*. Όταν εκτελεστεί η συνάρτηση, δηλαδή όταν μας έρθει ένα *HTTP* αίτημα, η συνάρτηση θα εκτελεστεί και θα δοθούν τιμές στις παραμέτρους. Με αυτόν τον τρόπο μπορούμε να μάθουμε λεπτομέρειες για το αίτημα (παράμετρος *request*) και να απαντήσουμε σε αυτό (παράμετρος *response*). Στον παραπάνω κώδικα χειριζόμαστε όλα τα εισερχόμενα *HTTP* αιτήματα με τον ίδιο τρόπο ˙ στέλνουμε ως

απάντηση το αρχείο *client.html* (μεταβλητή *page* από το προηγούμενο παράδειγμα).

Αφού δημιουργήσαμε τον εξυπηρετητή μας, πρέπει να τον ξεκινήσουμε. Για αυτόν τον λόγο τον βάζουμε να "ακούει" την *TCP/IP* πύλη (*port*) 3000. Η επιλογή της πύλης 3000 είναι αυθαίρετη. Κάθε πύλη πάνω από την 1024 θα λειτουργούσε. Σε μερικά λειτουργικά συστήματα όπως το *linux*, χρειαζόμαστε ειδικά δικαιώματα για να χρησιμοποιήσουμε πύλες κάτω από την 1024 [*CHHR*].

Με τον εξυπηρετητή να τρέχει πλέον, αν επισκεφτούμε με έναν φυλλομετρητή την http://127.0.0.1:3000 θα δούμε σαν απάντηση την σελίδα *client.html*. Με ελάχιστες γραμμές *JavaScript* λοιπόν, και το *Node* εγκατεστημένο, καταφέραμε να στήσουμε έναν *HTTP* εξυπηρετητή που εξυπηρετεί ένα *HTML* αρχείο.

Το τελευταίο κομμάτι που μας απασχόλησε, όσον αφορά τον εξυπηρετητή, είναι ο τρόπος επικοινωνίας του με τον εξυπηρετούμενο. Το *HTTP* πρωτόκολλο δεν είναι ιδανικό για εφαρμογές πραγματικού χρόνου επειδή είναι ένα πρωτόκολλο χωρίς πληροφορίες κατάστασης (*stateless protocol*). Αυτό σημαίνει ότι ενώ ο εξυπηρετούμενος μπορεί να στέλνει *HTTP* αιτήματα στον εξυπηρετητή και ο εξυπηρετητής να απαντάει σε αυτά, το ανάποδο δεν γίνεται. Αν ένα γεγονός συμβεί στον εξυπηρετητή, αυτός δεν έχει κάποιο τρόπο να ενημερώσει τον εξυπηρετούμενο για την αλλαγή της κατάστασης του σε πραγματικό χρόνο. Πρέπει να περιμένει να λάβει ένα αίτημα από τον εξυπηρετούμενο ώστε να ενημερώσει για την αλλαγή.

Για να λυθεί αυτό το πρόβλημα ο εξυπηρετούμενος πρέπει να χρησιμοποιεί περιοδικές ερωτήσεις (*polling*). Πρέπει δηλαδή να ρωτάει συνεχώς, για παράδειγμα κάθε δευτερόλεπτο ή και λιγότερο, τον εξυπηρετητή για την κατάστασή του. Με αυτόν τον τρόπο μπορούμε να εξομοιώσουμε επικοινωνία πραγματικού χρόνου. Αυτή η λύση δεν είναι καθόλου αποδοτική καθώς ο εξυπηρετητής πρέπει να χειριστεί πολλά εισερχόμενα αιτήματα, ακόμα και αν δεν έχει συμβεί κάποια αλλαγή στην κατάσταση του. Μια πιο αποδοτική λύση είναι η τεχνική μακράς περιόδευσης (*long polling)*. Σε αυτήν την τεχνική ο εξυπηρετούμενος στέλνει μόνο ένα αίτημα στον εξυπηρετητή και ο τελευταίος διατηρεί την σύνδεση ανοικτή μέχρι να παρουσιαστεί κάποια αλλαγή στην κατάσταση του.

Αυτοί οι περιορισμοί παρακίνησαν την ανάπτυξη του πρωτοκόλλου διαδικτυακών υποδοχών (*WebSocket protocol*). Αυτό ορίζει έναν τρόπο αμφίδρομης επικοινωνίας μεταξύ του εξυπηρετητή και του εξυπηρετούμενου, χωρίς την επιβάρυνση (πχ. οι κεφαλίδες *HTTP* στέλνονται με κάθε αίτημα) και το συνεχές άνοιγμα και κλείσιμο συνδέσεων του *HTTP* πρωτοκόλλου. Στην εφαρμογή μας για την υλοποίηση του πρωτοκόλλου διαδικτυακών υποδοχών θα χρησιμοποιήσουμε το *Socket.IO* (http://socket.io) πακέτο. Αυτό δεν είναι ενσωματωμένο με το *Node* και θα το εγκαταστήσουμε χρησιμοποιώντας τον διαχειριστή πακέτων του *Node* (*NPM*). Το *Node* μπορεί ταυτόχρονα να εξυπηρετήσει τόσο *HTTP* αιτήματα, όσο και συνδέσεις διαδικτυακών υποδοχών (*WebSocket connections*) χρησιμοποιώντας μόνο μια *TCP/IP* πύλη.

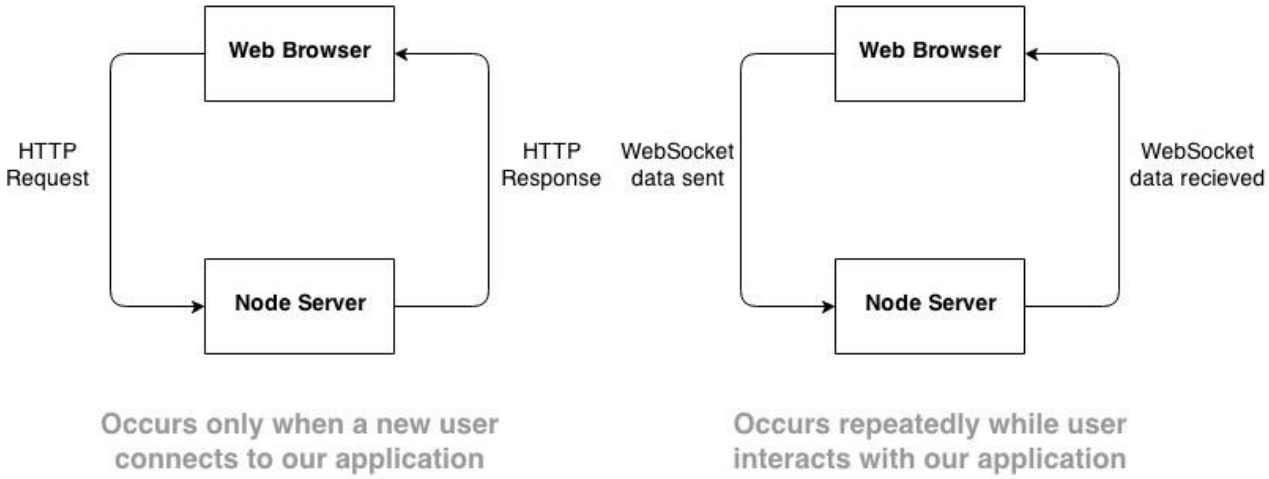
Αρχικά "ακούμε" την πύλη 3000 για συνδέσεις διαδικτυακών υποδοχών. Είναι η ίδια πύλη που "ακούμε" και για *HTTP* αιτήματα. Κάθε φορά που συνδέεται ένας εξυπηρετούμενος η συνάρτηση επιστρεφόμενης κλήσης εκτελείται, και ο εξυπηρετητής είναι σε θέση να δεχτεί μηνύματα από τον εξυπηρετούμενο, και τυπώνει το περιεχόμενο

αυτών των μηνυμάτων.

```
var io = require('socket.io').listen(app);

io.sockets.on('connection', function (socket) {
     socket.on('my event', function(content) {
          console.log(content);
     });
});
```

Το όνομα του μηνύματος επιλέγεται αυθαίρετα (εδώ "*my_event*") και δεν φέρει κανένα απολύτως νόημα για το *Socket.IO*. Είναι ουσιαστικά μια ταμπέλα (*label*) που δίνουμε στο μήνυμα, ώστε να ξεχωρίζουμε διαφορετικά είδη μηνυμάτων μεταξύ τους. Απλά πρέπει να προσέξουμε ότι το μήνυμα πρέπει να έχει την ίδια ταμπέλα και στην πλευρά του εξυπηρετούμενου.



Σχήμα 5.4 - Χειρισμός *HTTP* και διαδικτυακών υποδοχών στα πλαίσια μιας εφαρμογής

Στο σχήμα 5.4, φαίνεται η ανάπτυξη μιας εφαρμογής που χρησιμοποιεί τόσο *HTTP* όσο και συνδέσεις διαδικτυακών υποδοχών (*WebSocket connections*). Αρχικά ο εξυπηρετούμενος μέσω μιας *HTTP* αίτησης λαμβάνει το περιεχόμενο του *HTML* αρχείου, ώστε να στηθούν τα πράγματα στον φυλλομετρητή του χρήστη. Στην συνέχεια πάλι ο εξυπηρετούμενος ανοίγει μια μόνιμη και αμφίδρομη σύνδεση διαδικτυακών υποδοχών με τον εξυπηρετητή. Με αυτόν τον τρόπο πετυχαίνουμε επικοινωνία πραγματικού χρόνου μεταξύ εξυπηρετούμενου και εξυπηρετητή.

5.2 Η πλευρά του Εξυπηρετούμενου

Σε αυτό το σημείο είμαστε σε θέση να εμφανίσουμε τα δεδομένα στον χρήστη, μέσω ενός διαδικτυακού γραφικού περιβάλλοντος. Για την δημιουργία αυτού του περιβάλλοντος χρησιμοποιήσαμε διάφορες τεχνολογίες και βιβλιοθήκες, που μας

παρέχουν εργαλεία για την οπτικοποίηση της πληροφορίας.

5.2.1 Τεχνολογίες

Οι γλώσσες που χρησιμοποιήθηκαν για την οπτική διεπαφή με τον χρήστη είναι οι ακόλουθες :

- ***HTML*** : Η *HTML* (ακρωνύμιο του *HyperText Markup Language* ή Γλώσσα Σήμανσης Υπερκειμένου) είναι μια γλώσσα που χρησιμοποιείται για την περιγραφή της δομής των ιστοσελίδων. Η περιγραφή αυτή της δομής, γίνεται με χρήση σήμανσης (*markup*). Ο κύριος τρόπος που επιτυγχάνεται η σήμανση στην *HTML* είναι τα στοιχεία *HTML* (*HTML elements*). Τα στοιχεία *HTML* αποτελούνται από τις ετικέτες (*tags*) εκκίνησης και τερματισμού ˙ οι λέξεις ανάμεσα στις γωνιώδης παρενθέσεις (*angle-brackets* ή "< >"), τις ιδιότητες μέσα στην ετικέτα εκκίνησης και το κείμενο ή γραφικό περιεχόμενο μεταξύ των δύο ετικετών. Οι ετικέτες είναι λοιπόν οι κρυφές λέξεις (δεν εμφανίζονται στον χρήστη), που ορίζουν πως θα σχηματιστεί και θα εμφανιστεί το περιεχόμενο των σελίδων. Οι φυλλομετρητές (*browsers*) μπορούν να διαβάσουν αρχεία αυτής της γλώσσας (έχουν κατάληξη "*.html"*), και να τα καταστήσουν οπτικά ή ακουστικά προσιτά στον χρήστη.
- ***CSS*** : Η *CSS* (ακρωνύμιο του *Cascading Style Sheets* ή Διαδοχικά Φύλλα Στυλ) είναι μια γλώσσα που χρησιμοποιείται για την περιγραφή της παρουσίασης των ιστοσελίδων. Ουσιαστικά ελέγχει την εμφάνιση ενός εγγράφου που έχει γραφτεί σε μια γλώσσα σήμανσης (όπως η *HTML*), μέσω της στιλιστικής του ανάπτυξης, επεμβαίνοντας στο χρώμα, στο σχέδιο και τις γραμματοσειρές του. Επίσης μας προσφέρει την δυνατότητα να προσαρμόσουμε το παρουσιαστικό μιας ιστοσελίδας στους διάφορους τύπους συσκευών όπως για παράδειγμα μεγάλες ή μικρές οθόνες και εκτυπωτές. Ο διαχωρισμός της *CSS* από την *HTML* μας διευκολύνει αφενός στην συντήρηση των ιστοσελίδων και αφετέρου στο να είναι προσιτό το ίδιο περιεχόμενο σε διαφορετικά περιβάλλοντα.
- ***JavaScript*** : Η γλώσσα *JavaScript* λόγω της ραγδαίας ανάπτυξης του διαδικτύου, εκπλήρωσε το "γράψε το μια φορά, τρέξε το παντού" (*write once, run anywhere*) όνειρο της *Java* στις αρχές τις δεκαετίας του 1990. Η γλώσσα *JavaScript* είναι μια δυναμική γλώσσα σεναρίων που υποστηρίζει αντικειμενοστραφή, προστακτικό και συναρτησιακό προγραμματισμό. Χρησιμοποιείται κυρίως για προγραμματισμό στην πλευρά του εξυπηρετούμενου (*client-side programming*) καθώς τρέχει πάνω σε φυλλομετρητές (*browsers*). Επιτρέπει την επικοινωνία μιας ιστοσελίδας με τον χρήστη μέσω της ασύγχρονης ανταλλαγής δεδομένων και της δημιουργίας δυναμικού περιεχομένου χωρίς να χρειάζεται η διαμεσολάβησή του εξυπηρετητή (*server*). Τα βασικά χαρακτηριστικά της γλώσσας (το μοτίβο κλεισίματος - *closure pattern* και τις συναρτήσεις ως αντικείμενα πρώτης τάξης – *first-class functions*) τα είδαμε στην ενότητα 5.1.1, καθώς στα πλαίσια της εφαρμογής μας την χρησιμοποιήσαμε τόσο στην πλευρά του εξυπηρετούμενου όσο και του εξυπηρετητή.

- ***JQuery*** : Η *JQuery* είναι η δημοφιλέστερη βιβλιοθήκη της *JavaScript*, φτιαγμένη για να λειτουργεί σε όλους τους γνωστούς φυλλομετρητές (*browsers*). Σχεδιάστηκε για να απλοποιήσει την υλοποίηση σεναρίων στην πλευρά του πελάτη. Παρέχει μεθόδους στους προγραμματιστές ώστε να κάνει ευκολότερη και γρηγορότερη την δημιουργία ενός δυναμικού περιβάλλοντος καθώς και την επικοινωνία μεταξύ εξυπηρετούμενου – εξυπηρετητή (*client - server*).

Όλες οι παραπάνω τεχνολογίες που χρησιμοποιήσαμε στα πλαίσια της εφαρμογής μας, τρέχουν κανονικά σε όλους τους γνωστούς σύγχρονους φυλλομετρητές. Αρκεί λοιπόν ο χρήστης να ανοίξει κάποιον από αυτούς όπως ο *Google Chrome* ή ο *Mozilla Firefox*, για να μπορέσει να τρέξει την εφαρμογή.

5.2.2 Βιβλιοθήκες

Για την την εύκολη και αποτελεσματική παρουσίαση των πληροφοριών στον χρήστη χρησιμοποιήσαμε διάφορα μέσα οπτικοποίησης, όπως χάρτες, γραφήματα και σύννεφα λέξεων (*tag clouds*). Για την διευκόλυνσή μας στον σκοπό αυτό χρησιμοποιήσαμε τις εξής βιβλιοθήκες :

- ***OpenLayers*** : Η *OpenLayers*[5] είναι μια βιβλιοθήκη ανοιχτού κώδικα γραμμένη σε *JavaScript* που μας επιτρέπει να εμφανίζουμε δεδομένα χάρτη σε φυλλομετρητές. Οι χάρτες αυτοί είναι διαδραστικοί καθώς επιτρέπουν στον χρήστη να κάνει πολύ περισσότερα πράγματα από το να κοιτάει απλά την δημιουργία του προγραμματιστή. Για να πετύχει η βιβλιοθήκη αυτήν την διαδραστικότητα παρέχει στους χρήστες ένα σύνολο εργαλείων ελέγχου όπως δυνατότητα περιήγησης, εστίασης και άλλα [*DLA*]. Ένα τελευταίο χαρακτηριστικό είναι η εκφραστικότητα καθώς δίνει την δυνατότητα "πάνω" από τον χάρτη να εφαρμόσουμε πολλά στρώματα (*layers*) γραφικών αντικειμένων (εικόνες, πολύγωνα και άλλα).
- ***HeatmapLayer*** : Η *HeatmapLayer* [*HML*] είναι μια βιβλιοθήκη που μας παρέχει την δυνατότητα εφαρμογής ενός στρώματος (*layer*) χάρτη θερμότητας (*heatmap*) "πάνω" από τον χάρτη της *OpenLayers*. Χάρτης Θερμότητας είναι μια γραφική αναπαράσταση δεδομένων, που χρησιμοποιεί χρώματα για να δείξει το μέγεθος της δραστηριότητας σε κάποιο σημείο. Συνήθως τα σκοτεινά χρώματα συμβολίζουν χαμηλά επίπεδα δραστηριότητας ενώ τα φωτεινά υψηλά επίπεδα δραστηριότητας. Στην εφαρμογή μας το πράσινο συμβολίζει χαμηλό αριθμό τιτιβισμάτων (*tweets*) σε μια περιοχή, ενώ το κόκκινο υψηλό αριθμό.
- ***Sigma*** : Η *Sigma* ή *Sigma.js*[6] είναι μια βιβλιοθήκη *JavaScript* που μας επιτρέπει να σχεδιάσουμε γράφους σε ιστοσελίδες. Επίσης μας παρέχει διάφορες πρόσθετες βιβλιοθήκες που δίνουν την δυνατότητα ο χρήστης να αλληλεπιδράσει

5 openlayers.org
6 sigmajs.org

με τον γράφο (πχ. *animation,* εστίαση). Στα πλαίσια της εφαρμογής μας, την χρησιμοποιήσαμε για να αναπαραστήσουμε την βάση δεδομένων μας (στο κεφάλαιο 3 είδαμε ότι η βάση δεδομένων μας αποθηκεύει τα δεδομένα σε μορφή γράφου).

- ***Raphaël*** : Η *Raphaël*[7] είναι μια βιβλιοθήκη *JavaScript* που μας επιτρέπει να εμφανίσουμε γεωμετρικά στοιχεία που εκφράζονται από μια μαθηματική έκφραση (σημεία, γραμμές, καμπύλες, πολύγωνα και άλλα), ως εικόνες στην οθόνη. Στα πλαίσια της εφαρμογής μας την χρησιμοποιήσαμε για να εμφανίσουμε έναν περιστρεφόμενο κύκλο (*spinner*) στην οθόνη. Με αυτόν τον τρόπο ενημερώνουμε τον χρήστη ότι τα δεδομένα δεν είναι διαθέσιμα για οπτικοποίηση και πρέπει να περιμένει.

Σχήμα 5.5 – Ένας περιστρεφόμενος κύκλος χρησιμοποιώντας την βιβλιοθήκη *Raphaël*

- ***Morris.js*** : Η *Morris.js*[8] είναι μια βιβλιοθήκη που μας επιτρέπει να εμφανίσουμε διαγράμματα σε ιστοσελίδες. Για να το πετύχει αυτό χρησιμοποιεί την βιβλιοθήκη *Raphaël,* αλλά μας προσφέρει μεθόδους που κάνουν πολύ εύκολη την διαδικασία. Υποστηρίζει τέσσερις τύπους διαγραμμάτων : γραφήματα γραμμών (*line charts*), γραφήματα περιοχής (*area charts*), γραφήματα ράβδου (*bar charts*) και γραφήματα πίτας (*donut charts*).
- ***Goat 1000*** : Η *Goat 1000*[9] είναι μια βιβλιοθήκη που μας επιτρέπει να εμφανίσουμε σύννεφα λέξεων (*tag clouds*) στην ιστοσελίδα μας. Μας προσφέρει αρκετές δυνατότητες όπως εναλλαγή χρώματος στις λέξεις ανάλογα με το βάρος, και κίνηση των λέξεων με διάφορες ταχύτητες.

7 raphaeljs.com
8 morrisjs.github.io/morris.js
9 www.goat1000.com

6 Η Εφαρμογή μας

6.1 Αρχιτεκτονική

Στα πλαίσια της εφαρμογής μας χρειαζόμαστε μεγάλο όγκο δεδομένων πραγματικού χρόνου. Για τον λόγο αυτό χρησιμοποιήσαμε το *Streaming API* του *Twitter,* ώστε να ξεπεράσουμε τους περιορισμούς ορίου χρήσης (*rate limits*) των υπόλοιπων διεπαφών.

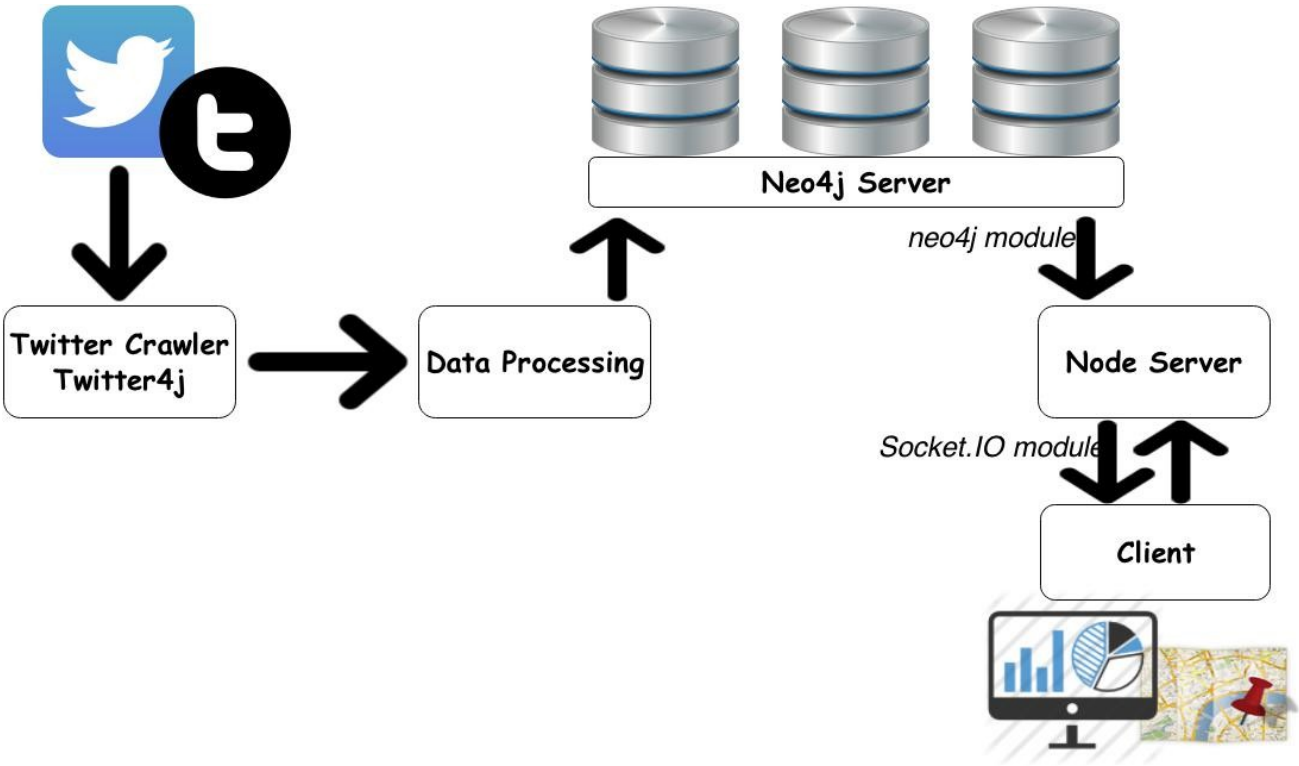
Η αρχική ιδέα για την αρχιτεκτονική της εφαρμογής μας ήταν η δημιουργία μιας μόνιμης σύνδεσης μεταξύ του *Twitter* και του *Node* εξυπηρετητή (*Node server*) μας, μέσω της οποίας θα προμηθεύουμε την εφαρμογή μας τα δεδομένα που χρειάζεται. Με αυτόν τον τρόπο θα χρησιμοποιούσαμε τον εξυπηρετητή μας, τόσο για την επικοινωνία και την παρουσίαση των δεδομένων στον εξυπηρετούμενο (*client*), όσο και για την συλλογή των δεδομένων από το *Twitter*. Μια τέτοια υλοποίηση απαιτεί μεγάλο υπολογιστικό φόρτο στην πλευρά του εξυπηρετητή. Επίσης να σημειώσουμε εδώ ότι η χρησιμοποίηση του *Streaming API* καθώς και του *REST API* μέσω του *Node* είναι μια πολύπλοκη διαδικασία, καθώς όσα πακέτα είναι διαθέσιμα για αυτόν τον σκοπό στον διαχειριστή πακέτων του *Node* (*ntwitter*, *twitter-node*) δεν μας παρέχουν την ποικιλία των μεθόδων που μας παρέχουν βιβλιοθήκες άλλων προγραμματιστικών περιβαλλόντων.

Στα πλαίσια της αρχιτεκτονικής της εφαρμογής μας αποφασίσαμε να διαχωρίσουμε αυτές τις δύο λειτουργίες (την συλλογή δεδομένων από την παρουσίαση τους). Έτσι η συλλογή των δεδομένων και η επεξεργασία τους γίνεται μέσω μιας *Java* εφαρμογής, που τρέχει ανεξάρτητα από τον εξυπηρετητή μας στο παρασκήνιο. Στην συνέχεια αυτά τα δεδομένα αποθηκεύονται στην βάση δεδομένων από όπου και τα παρουσιάζει στον χρήστη ο εξυπηρετητής μας. Με αυτόν τον τρόπο ο εξυπηρετητής μας ασχολείται αποκλειστικά με την βασική του λειτουργία: την εξυπηρέτηση της σελίδας μας. Για την χρησιμοποίηση του *Streaming API* και του *REST API* χρησιμοποιήσαμε την *Twitter4j*, μια *Java* βιβλιοθήκη που μας παρέχει έτοιμες μεθόδους για πρόσβαση στα δεδομένα του *Twitter*.

Η συλλογή αυτών των δεδομένων, ακολουθείται από την επεξεργασία τους ώστε να είναι σε μορφή κατάλληλη για την αποθήκευση τους στην βάση δεδομένων. Για την αποθήκευσή των δεδομένων στην βάση χρησιμοποιούμε την *REST* διεπαφή που αυτή μας παρέχει. Αυτό το πετυχαίνουμε μέσω της βιβλιοθήκης *Java REST Bindings*, η οποία περιτυλίγει τις *REST* κλήσεις πίσω από την διεπαφή *GraphDatabaseService* που χρησιμοποιείται στην ενσωματωμένη λειτουργία (*embedded mode*).

Για την ανάσυρση των δεδομένων από την βάση μέσω του *Node* εξυπηρετητή μας χρησιμοποιούμε το πακέτο *neo4j*. Το συγκεκριμένο πακέτο λειτουργεί και αυτό (όπως και η βιβλιοθήκη *Java REST Bindings*) σαν περιτύλιγμα (*wrapper*) στην *REST* διεπαφή

της βάσης. Αφού ανασύρει τα δεδομένα ο εξυπηρετητής μας τα παρουσιάζει σε πραγματικό χρόνο στον χρήστη. Για την επίτευξη της επικοινωνίας πραγματικού χρόνου μεταξύ εξυπηρετητή και εξυπηρετούμενο χρησιμοποιούμε το πρωτόκολλο διαδικτυακών υποδοχών (*WebSocket protocol*) μέσω του πακέτου *Socket.IO*. Στο σχήμα 6.1 φαίνεται συνοπτικά η αρχιτεκτονική της εφαρμογής μας.



Σχήμα *6.1* – Επισκόπηση της αρχιτεκτονικής της εφαρμογής μας

6.2 Συλλογή και Επεξεργασία Δεδομένων

Για την συλλογή δεδομένων πραγματικού χρόνου από το *Twitter* χρησιμοποιούμε το *Filter stream* του *Streaming API*. Ως κριτήριο αναζήτησης θέτουμε το περιβάλλον πλαίσιο (*bounding box*) μέσα στο οποίο γίνεται ένα τιτίβισμα (*tweet*). Στα πλαίσια της εφαρμογής μας χρησιμοποιήσαμε τα εξής περιβάλλοντα πλαίσια (σε μορφή συντεταγμένων {δύση, νότος, ανατολή, βοράς}):

- {`37.85, 23.34, 38.16, 23.91`} – Αθήνα
- {`51.2225, -0.6485, 51.7884, 0.2991`} – Λονδίνο
- {`52.32794, 4.78959, 52.42268, 4.99971`} – Άμστερνταμ
- {`40.52861, -74.32982, 40.92823, -73.72557`} – Νέα Υόρκη
- {`23.36881, 28.51182, 31.13415, 33.03819`} - Αίγυπτος

Κατά την συλλογή των δεδομένων από αυτές τις περιοχές, και πριν τα αποθηκεύσουμε στην βάση, περνάμε τα δεδομένα από μια επεξεργασία ώστε να διευκολύνουμε στη συνέχεια την αναζήτηση τους και την παρουσίαση τους.

Για κάθε εισερχόμενο τιτίβισμα βρίσκουμε το χρήστη–δημιουργό του. Αυτός είναι πιθανόν να υπάρχει ήδη στην βάση, είτε επειδή έχουμε ήδη συλλέξει κάποιο άλλο του τιτίβισμα στο παρελθόν, είτε επειδή είναι ακόλουθος (*follower*) κάποιου υπάρχοντος χρήστη. Αν δεν υπάρχει τον δημιουργούμε.

Στην συνέχεια επικεντρωνόμαστε στο κείμενο του τιτιβίσματος. Αρχικά, βρίσκουμε όλα τα *hashtags* που αυτό περιέχει. *Hashtag* είναι η λέξη του κειμένου ενός τιτιβίσματος που ακολουθεί τον ειδικό χαρακτήρα "#", και αποτελεί μια ένδειξη για το θέμα στο οποίο αναφέρεται το τιτίβισμα. Στην συνέχεια, αναζητούμε στο τιτίβισμα μεταδεδομένα που πιθανόν να περιέχει. Τέτοια μπορεί να είναι κάποια φωτογραφία, ή κάποιος σύνδεσμος (*link*) σε κάποια ιστοσελίδα.

Το τελευταίο κομμάτι που αφορά το κείμενο του τιτιβίσματος είναι η ανάλυση συναισθημάτων (*sentiment analysis*) ή αλλιώς εξόρυξη γνώμης-άποψης (*opinion mining*). Για τον σκοπό αυτό χρησιμοποιούμε μια έτοιμη βιβλιοθήκη (*AlchemyAPI*) η οποία μας δίνει την δυνατότητα να χρησιμοποιήσουμε πολλά εργαλεία επεξεργασίας φυσικής γλώσσας (*Natural Language Processing – NLP*) μέσω μιας διεπαφής προγραμματισμού εφαρμογών (API). Στα πλαίσια της εφαρμογής μας θα χρησιμοποιήσουμε ανάλυση συναισθημάτων στα κείμενα των τιτιβισμάτων με σκοπό να αποφανθούμε για την πολικότητα του κάθε τιτιβίσματος (θετικό, ουδέτερο ή αρνητικό).

Άλλες πληροφορίες που μας ενδιαφέρουν είναι η ώρα δημιουργίας (*timestamp*) ενός τιτιβίσματος, καθώς μας βοηθάει να οργανώσουμε τα τιτιβίσματα στον χρόνο, και η ακριβής τοποθεσία του σε μορφή συντεταγμένων γεωμετρικού μήκους (*longitude*) και πλάτους (*latitude*), καθώς μας βοηθάει να οργανώσουμε τα τιτιβίσματα στον χώρο.

Πάνω στις γεωμετρικές συντεταγμένες ενός τιτιβίσματος (ζευγάρι τιμών {γεωμετρικό μήκος, γεωμετρικό πλάτος}) δημιουργούμε ένα χωρικό ευρετήριο ("`tweetWKT`"). Με αυτόν τον τρόπο διευκολύνεται και επιταχύνεται η αναζήτηση τιτιβισμάτων με βάση την γεωγραφική περιοχή όπου αυτά έγιναν.

Στα πλαίσια της πληρότητας της εφαρμογής μας, αποφασίσαμε να δημιουργήσουμε το γράφο ακολούθων (*followers graph*) για τους χρήστες που έχουν δημοσιεύσει κάποιο τιτίβισμα. Αυτή η διαδικασία τρέχει στο παρασκήνιο και είναι ανεξάρτητη από την διαδικασία συλλογής τιτιβισμάτων. Μάλιστα συνίσταται να τρέχει στο παρασκήνιο ακόμα και όταν δεν συλλέγουμε τιτιβίσματα, προκειμένου να καταφέρουμε να δημιουργήσουμε ένα σχετικά "πλήρη" γράφο ακολούθων. Δηλαδή έναν γράφο όπου δεν θα υπάρχουν χρήστες-δημιουργοί τιτιβίσματος για τους οποίους δεν θα γνωρίζουμε τους ακόλουθους τους. Για την υλοποίηση της συγκεκριμένης διαδικασίας χρησιμοποιήσαμε το *REST API*.

6.2.1 Περιγραφή Κλάσεων

Προκειμένου να καταφέρουμε να συλλέξουμε και να επεξεργαστούμε τα δεδομένα δημιουργήσαμε μια *Java* εφαρμογή που τρέχει στο παρασκήνιο και αποτελείται από 6 κλάσεις, οι οποίες και περιγράφονται στην συνέχεια της παρούσας ενότητας. Για την κατανόηση των διασυνδέσεων μεταξύ των κλάσεων το σχήμα 6.2 δείχνει το αντίστοιχο διάγραμμα κλάσεων.

1. TwitterStreamGeo

Η συγκεκριμένη κλάση είναι υπεύθυνη αρχικά για την ομαλή έναρξη της εφαρμογής. Ορίζει τα περιβάλλοντα πλαίσια (*boundary boxes*) για την κάθε περιοχή, ξεκινάει την σύνδεση με την βάση δεδομένων καθώς και την έναρξη του νήματος (με όνομα "`Background Thread`") που είναι υπεύθυνο για την δημιουργία του γράφου των ακολούθων (*followers graph*). Στην συνέχεια συλλέγει σε πραγματικό χρόνο τα τιτιβίσματα (*tweets*) και τα αποθηκεύει στην βάση. Η μόνη της μέθοδος είναι η `showDialogBox` η οποία είναι υπεύθυνη για την δημιουργία ενός γραφικού περιβάλλοντος ώστε να μπορούμε να χειριζόμαστε την λειτουργία της διαδικασίας (ξεκίνημα, σταμάτημα κτλ.)**.**

2. DbFunctions

Η συγκεκριμένη κλάση χειρίζεται την επικοινωνία της εφαρμογής με την βάση δεδομένων. Πρόκειται για ένα μοναδικό αντικείμενο (*singleton*), από την άποψη ότι δεν μπορούμε να δημιουργήσουμε περισσότερα από ένα αντικείμενα αυτού του τύπου. Με αυτόν τον τρόπο εξασφαλίζουμε μια και μοναδική σύνδεση μεταξύ της εφαρμογής μας και της βάσης δεδομένων. Αποτελείται από τις εξής μεθόδους:

- *getInstance* : Μέθοδος που δημιουργεί που επιστρέφει το μοναδικό αντικείμενο του *DbFunctions*.
- *createDb* : Αρχικοποιεί την σύνδεση με την βάση δεδομένων.
- *createIndex* : Αρχικοποιεί το χωρικό ευρετήριο πάνω στην ιδιότητα "`wkt`" των κόμβων *Tweet*.
- *setLastNode* : Μέθοδος που επιστρέφει το τελευταίο χρονικά τιτίβισμα. Χρήσιμο για την δημιουργία μιας χρονικής αλυσίδας με τιτιβίσματα. Καλείται κατά την αρχικοποίηση της σύνδεσης με την βάση. Στην συνέχεια γνωρίζουμε ποιο είναι το τελευταίο χρονικά τιτίβισμα, καθώς τα τιτιβίσματα έρχονται σε πραγματικό χρόνου (*real time*) και συνεπώς ταξινομημένα ως προς αυτόν.
- *insertDb* : Μέθοδος που χρησιμοποιείται για να εισάγουμε στην βάση δεδομένων μας ένα νέο τιτίβισμα μαζί με τα χαρακτηριστικά του (τοποθεσία, χρήστης-δημιουργός, χρόνος, κείμενο κτλ.). Φροντίζει τόσο για την ομαλή ένταξη του τιτιβίσματος στο χωρικό ευρετήριο, όσο και για την επεξεργασία του κειμένου του τιτιβίσματος προκειμένου να το εισάγει στην βάση.

- *getHashTagNode* : Μέθοδος που ανακτά από την βάση το ζητούμενο *Hashtag*, ή το δημιουργεί αν αυτό δεν υπάρχει.
- *getUserNode* : Μέθοδος που ανακτά από την βάση δεδομένων το ζητούμενο *User*, ή τον δημιουργεί μαζί με όλα τα χαρακτηριστικά του.
- *getFollowingUser* : Μέθοδος που επιστρέφει τον χρήστη του οποίου ψάχνουμε τους ακόλουθους.
- *setFollowerUser* : Μέθοδος που χρησιμοποιείται για την εισαγωγή στην βάση δεδομένων μας του ακόλουθου του χρήστη που επιστρέφει η μέθοδος *getFollowingUser*.
- *inactiveFollowingUser* : Θέτει κάποιον χρήστη ως μη ενεργό.
- *addHashTagsRelationships* : Μέθοδος που χρησιμοποιείται για την εισαγωγή στην βάση δεδομένων μας πληροφορίας που αφορά την συσχέτιση μεταξύ *hashtags* που εμφανίζονται στο ίδιο τιτίβισμα.
- *addLabel* : Προσθέτει ετικέτα σε κάποιον κόμβο.

3. FollowersBackgroundThread

Η συγκεκριμένη κλάση χειρίζεται το παρασκηνιακό νήμα (με όνομα "`Background Thread`") που είναι υπεύθυνο για την δημιουργία του γράφου ακολούθων (*followers graph*). Αποτελείται από τις εξής μεθόδους :

- *findFollowers* : Μέθοδος που βρίσκει τους ακόλουθους κάποιου χρήστη με βάση το μοναδικό *id* αυτού.
- *checkLimit* : Μέθοδος που ελέγχει ώστε να μην υπερβούμε τον περιορισμό ορίου χρήσης (*rate limit*). Το όριο είναι 15 ερωτήματα ανά 15 λεπτά. Σε περίπτωση που φτάσουμε το όριο το νήμα "πέφτει" για ύπνο.
- *goSleep* : Στέλνει για ύπνο το παρασκηνιακό νήμα για όσο χρόνο ακριβώς απαιτείται από τον περιορισμό ορίου χρήσης (λιγότερο σίγουρα από 15 λεπτά).

4. TwitterAPIs

Η συγκεκριμένη κλάση χειρίζεται τις διεπαφές του *Twitter*. Πρόκειται για ένα μοναδικό αντικείμενο (*singleton*), από την άποψη ότι δεν μπορούμε να δημιουργήσουμε περισσότερα από ένα αντικείμενα αυτού του τύπου. Με αυτόν τον τρόπο εξασφαλίζουμε ότι δεν θα χρησιμοποιήσουμε περισσότερες από μια φορές τις διεπαφές του *Twitter* με τα ίδια διαπιστευτήρια (*credentials*) κάτι που θα οδηγήσει σε διακοπή της εφαρμογής μας. Αποτελείται από τις εξής μεθόδους :

- *getInstance* : Μέθοδος που δημιουργεί που επιστρέφει το μοναδικό αντικείμενο του *TwitterAPIs*.
- *getStream* : Επιστρέφει το *Streaming API*.

- *getREST* : Επιστρέφει το *REST API*, προκειμένου να χρησιμοποιηθεί από το παρασκηνιακό νήμα.

5. Text

Η συγκεκριμένη κλάση χειρίζεται το κείμενο των τιτιβισμάτων προκειμένου να απομονώσει τα διάφορα *hashtags* ή μεταδεδομένα, όπως συνδέσμους σε άλλες ιστοσελίδες και εικόνες, που συνδέονται άμεσα με το τιτίβισμα. Για κάθε τιτίβισμα δημιουργείται ένα καινούργιο αντικείμενο αυτού του τύπου. Αποτελείται από τις εξής μεθόδους :

- *setURL* : Μέθοδος που υπολογίζει τα μεταδεδομένα που συνδέονται με το κείμενο του τιτιβίσματος.
- *setHashtagList* : Μέθοδος που υπολογίζει την λίστα με όλα τα *hashtags* που πιθανόν υπάρχουν στο κείμενο του τιτιβίσματος.
- *getURL* : Μέθοδος που παρέχει τα μεταδεδομένα που συνδέονται με το κείμενο του τιτιβίσματος.
- *getHashtagList* : Μέθοδος που παρέχει την λίστα με όλα τα *hashtags* που πιθανόν υπάρχουν στο κείμενο του τιτιβίσματος.

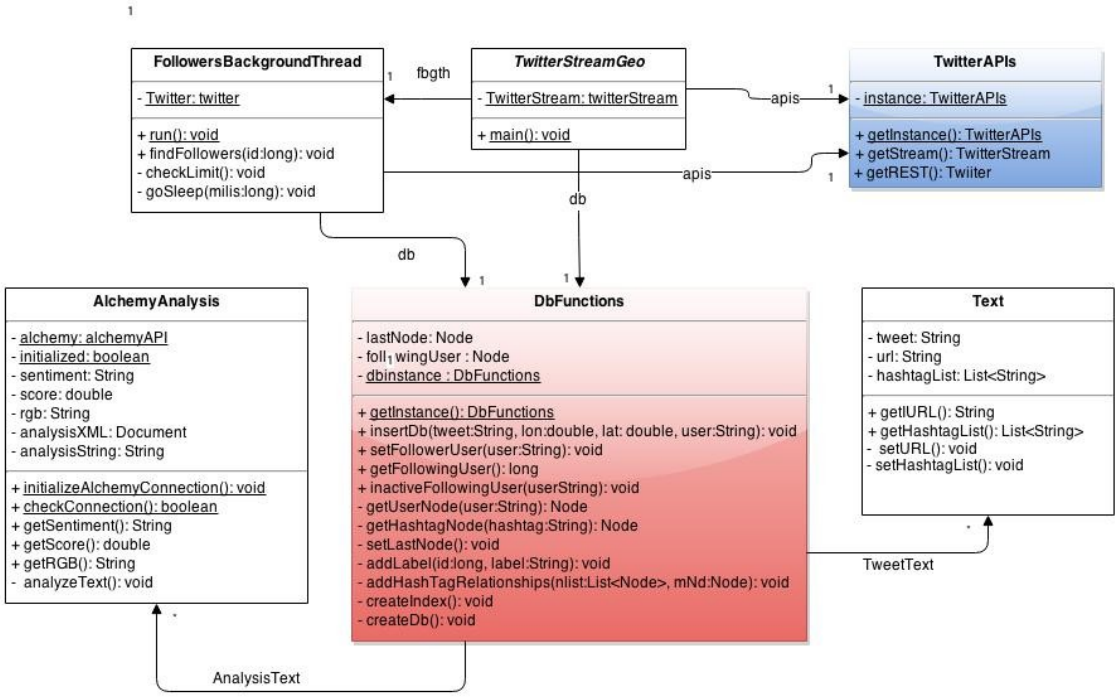
6. AlchemyAnalysis

Η συγκεκριμένη κλάση είναι υπεύθυνη για την ανάλυση συναισθημάτων (*sentiment analysis*) του κειμένου ενός τιτιβίσματος, προκειμένου να αποφασιστεί η πολικότητα του περιεχομένου του (θετικό, αρνητικό ή ουδέτερο). Για κάθε τιτίβισμα δημιουργούμε ένα καινούργιο αντικείμενο αυτής της κλάσης. Αποτελείται από τις εξής μεθόδους :

- *initializeAlchemyConnection* : Αυτή η μέθοδος χρησιμοποιείται για να συνδεθούμε με το *AlchemyAPI*. Καλείται στην αρχή της εφαρμογής ώστε να δηλώσουμε τα διαπιστευτήρια μας στο *AlchemyAPI* και να ξεκινήσουμε να στέλνουμε αιτήσεις (*requests*).
- *checkConnection* : Αυτή η μέθοδος χρησιμοποιείται για να ελέγξει την σύνδεση μας στο *AlchemyAPI*. Σε περίπτωση που υπάρχει κάποια σύνδεση ανοιχτή δεν μας επιτρέπει να δημιουργήσουμε καινούργια σύνδεση με τα ίδια διαπιστευτήρια.
- *analyzeText* : Μέθοδος που είναι υπεύθυνη για την διεξαγωγή της ανάλυσης συναισθημάτων στο κείμενο του τιτίβισμα.
- *getSentiment* : Μέθοδος που παρέχει το αποτέλεσμα της ανάλυσης συναισθημάτων σε μορφή κειμένου – πολικότητας.
- *getScore* : Παρέχει το αποτέλεσμα της ανάλυσης συναισθημάτων σε αριθμητική μορφή από το -1(*negative*) έως το 1(positive).
- *scoreToRGB* : Μέθοδος που μετατρέπει το αριθμητικό αποτέλεσμα της

ανάλυσης συναισθημάτων σε πρότυπο χρώματος *RGB* δεκαεξαδικής μορφής. Για παράδειγμα το κόκκινο παίρνει την τιμή "`#ff0000`". Έτσι οι αρνητικές τιμές (κοντά στο -1) μετατρέπονται σε κόκκινο, οι ουδέτερες (κοντά στο 0) σε μπλε ενώ οι θετικές (κοντά στο 1) σε πράσινο χρώμα.

- *getRGB* : Μέθοδος που παρέχει το το αποτέλεσμα της ανάλυσης συναισθημάτων σε πρότυπο χρώματος *RGB* δεκαεξαδικής μορφής.



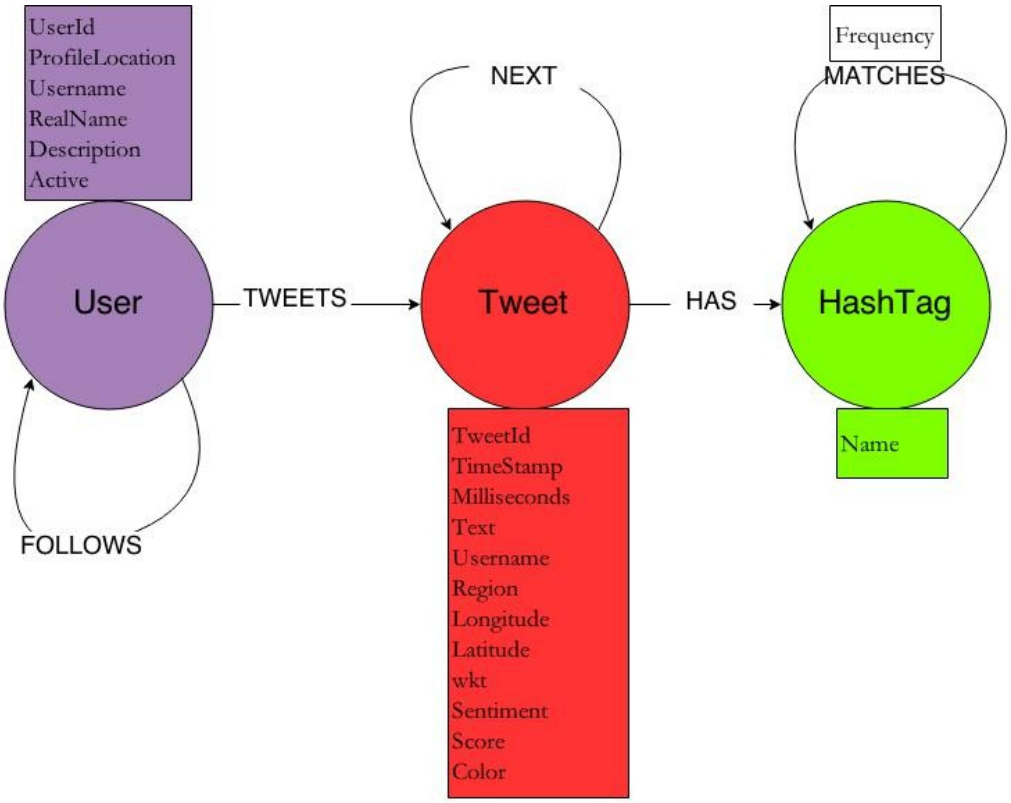
Σχήμα 6.2 – Διάγραμμα κλάσεων για την εφαρμογή συλλογής και επεξεργασίας δεδομένων

6.3 Σχήμα της Βάσης

Στην προηγούμενη ενότητα είδαμε τον τρόπο με τον οποίο συλλέγουμε και επεξεργαζόμαστε τα δεδομένα. Αφού τα επεξεργαστούμε πρέπει να τα αποθηκεύσουμε σε μια βάση δεδομένων, ώστε να είμαστε σε θέση να τα ανασύρουμε ανά πάσα στιγμή και να τα παρουσιάσουμε στον χρήστη. Για τον σκοπό αυτόν χρησιμοποιήσαμε την *Neo4j*.

Η *Neo4j* είναι μια βάση δεδομένων γράφου με ιδιότητες (*property graph database*). Αυτό σημαίνει ότι περιέχει κόμβους και σχέσεις μεταξύ αυτών. Τόσο οι κόμβοι όσο και οι σχέσεις μπορούν να έχουν ταμπέλες (*labels*), και ιδιότητες (*properties*) ως ζευγάρια "*key-value*".

Στο σχήμα 6.3 φαίνεται το σχήμα δεδομένων της βάσης που χρησιμοποιούμε. Μέσα σε κύκλο βρίσκονται τα διαφορετικά ήδη κόμβων ή οι διαφορετικές οντότητες. Με βελάκια φαίνονται οι σχέσεις μεταξύ των κόμβων ενώ σε ορθογώνια κουτιά φαίνονται οι ιδιότητες.



Σχήμα *6.3* – Το σχήμα της βάσης δεδομένων μας

Θα εξηγήσουμε το παραπάνω σχήμα αναλυτικά. Οι ετικέτες κόμβων ή οι οντότητες της βάσης δεδομένων μας είναι οι εξής :

1. ***User*** : Στην οντότητα *User* ανήκουν όλοι οι χρήστες του *Twitter*. Για κάθε καινούργιο τιτίβισμα, αναγνωρίζουμε τον χρήστη–δημιουργό του και τον προσθέτουμε στην βάση ως χρήστη. Κάθε κόμβος *User* έχει τις εξής ιδιότητες (τις αναγράφουμε ως όνομα ιδιότητας και σε παρένθεση τον τύπο της):
   - *UserId* (*long*) : Το μοναδικό χαρακτηριστικό κάθε χρήστη του *Twitter*.
   - *ProfileLocation* (*string*) : Η περιοχή που έχει θέση ο χρήστης ως τοποθεσία του προφίλ του. Μπορεί να μην υπάρχει τέτοια πληροφορία (ο χρήστης να μην έχει θέσει τίποτα) ή να μην έχει νόημα η περιοχή που έχει βάλει ο χρήστης σαν τοποθεσία του προφίλ του. Σε περίπτωση που ο χρήστης δεν έχει θέσει τοποθεσία προφίλ η συγκεκριμένη ιδιότητα παίρνει την τιμή "`Username has not set a Profile Location`". Δεν πρέπει να την μπερδεύουμε με την τοποθεσία του τιτιβίσματος.
   - *Username* (*string*) : Το όνομα του χρήστη. Είναι αυτό με το οποίο μπορούν να του απευθυνθούν οι άλλοι χρήστες (μετά τον ειδικό χαρακτήρα "@"), και αυτό που εμφανίζεται όταν δημοσιοποιεί ένα τιτίβισμα.
   - *RealName* (*string*) : Το πραγματικό όνομα του χρήστη. Μπορεί να ταυτίζεται ή και όχι με το *Username*.
   - *Description* (*string*) : Η σύντομη περιγραφή ενός χρήστη για τον εαυτό του.

Μπορεί να μην υπάρχει τέτοια πληροφορία, δηλαδή ο χρήστης να μην έχει θέσει τίποτα, οπότε αυτή η ιδιότητα δεν παίρνει κάποια τιμή.

- *Active* (*string*) : Μας πληροφορεί για το αν ο χρήστης είναι ενεργός ή όχι. Κάθε χρήστης που δημοσιοποιεί ένα τιτίβισμα είναι ενεργός˙ η συγκεκριμένη ιδιότητα παίρνει την τιμή "YES". Σε μια μεταγενέστερη χρονική στιγμή μπορεί να αναζητήσουμε τους ακόλουθους (*followers*) του χρήστη, ώστε να δημιουργήσουμε τον γράφο ακολούθων (*followers graph*). Ο χρήστης μπορεί να μην είναι πλέον ενεργός, είτε επειδή έχει διαγράψει τον λογαριασμό του (τιμή "NO") ,είτε επειδή έχει κάνει τον λογαριασμό του προστατευόμενο οπότε και δεν έχουμε δικαιοδοσία να λάβουμε πληροφορίες για το συγκεκριμένο χρήστη (τιμή "PROTECTED").

2. ***Tweet*** : Στην οντότητα *Tweet* ανήκουν όλα τα τιτιβίσματα που έχουμε συλλέξει. Κάθε τιτίβισμα έχει υποχρεωτικά κάποιον χρήστη-δημιουργό, και μια ακριβή τοποθεσία από όπου προέρχεται. Η τοποθεσία είναι της μορφής ζευγαριού γεωμετρικού μήκους και πλάτους. Κάθε κόμβος *Tweet* έχει τις εξής ιδιότητες :

   - *TweetId* (*long*) : Το μοναδικό χαρακτηριστικό κάθε τιτιβίσματος που γίνεται στο *Twitter.*
   - *TimeStamp* (*string*) : Η ακριβής ημερομηνία και ώρα στην οποία δημοσιοποιήθηκε ένα τιτίβισμα. Είναι αποθηκευμένο ως συμβολοσειρά (*string*) της μορφής "*ddd MMM dd HH:mm:ss zzz yyyy*", για παράδειγμα "*Sun Jul 27 21:43:50 EEST 2014*".
   - *Milliseconds* (*long*) : Ο αριθμός των χιλιοστών του δευτερολέπτου ανάμεσα στην ημερομηνία της ιδιότητας *TimeStamp* και της 01/01/1970 στις 12 τα μεσάνυχτα.
   - *Tweet* (*string*) : Το κείμενο του τιτιβίσματος.
   - *Username* (*string*) : Το μοναδικό όνομα του χρήστη-δημιουργού του τιτιβίσματος. Η συγκεκριμένη πληροφορία υπάρχει ήδη στην βάση αφού, όπως θα δούμε στην συνέχεια, οι κόμβοι *User* και *Tweet* σχετίζονται. Συνεπώς μπορεί να ανακτηθεί μέσω ενός ερωτήματος (*query*) σε αυτήν. Επειδή όμως μαζί με την εμφάνιση κάθε τιτιβίσματος εμφανίζουμε και το όνομα του δημιουργού του, αποφασίσαμε να αποθηκεύσουμε ξανά την πληροφορία αυτή ώστε να αποφύγουμε την εκτέλεση επιπλέον ερωτημάτων.
   - *Region* (*string*) : Η περιοχή από όπου προέρχεται το τιτίβισμα. Μπορεί να πάρει τιμές μόνο τις περιοχές από όπου μαζεύουμε τιτιβίσματα, δηλαδή μια εκ των *Athens*, *London*, *Amsterdam*, *NewYork* και *Egypt*. Την χρησιμοποιούμε για να ελέγξουμε την σωστή λειτουργία του χωρικού ευρετηρίου μας.
   - *Longitude* (*double*) : Το γεωγραφικό μήκος της ακριβής τοποθεσίας του τιτιβίσματος.
   - *Latitude* (double) : Το γεωγραφικό πλάτος της ακριβής τοποθεσίας του τιτιβίσματος.
   - *wkt* (*string*) : Πρόκειται για το ζεύγος των τιμών του γεωγραφικού μήκους και

πλάτους της ακριβής τοποθεσίας του τιτιβίσματος. Είναι της μορφής "`POINT(γεωγραφικό_μήκος,γεωγραφικό_πλάτος)`". Για παράδειγμα ένα τιτίβισμα που γίνεται στο σημείο με συντεταγμένες 37.946729 γεωγραφικό μήκος και 23.664910 γεωγραφικό πλάτος, η τιμή της ιδιότητας *wkt* είναι "*POINT*(37.946729,23.664910)". Η συγκεκριμένη ιδιότητα είναι χρήσιμη για την δημιουργία του χωρικού ευρετηρίου πάνω στους κόμβους *Tweet*.

- *Sentiment* (*string*) : Πρόκειται για το αποτέλεσμα της συναισθηματικής ανάλυσης (*sentiment analysis*) που έγινε στο τιτίβισμα. Μπορεί να πάρει τις τιμές "`Positive`", "`Negative`", "`Neutral`", "`No available Sentiment – unsupported language`", "`No available Sentiment – daily limit exceeded`" και "`No sentiment returned`". Η σημασία κάθε μία από τις δυνατές τιμές είναι προφανής.
- *Score* (*double*) : Το αποτέλεσμα της συναισθηματικής ανάλυσης που έγινε στο τιτίβισμα αλλά σε αριθμητική μορφή. Παίρνει τιμές ανάμεσα στο -1 και το +1. Τιμές κοντά στο -1 σημαίνουν ότι το τιτίβισμα έχει αρνητικό περιεχόμενο, ενώ αντίθετα πλησιάζοντας το +1 το περιεχόμενο γίνεται όλο και θετικότερο. Τιμή 0 έχουμε για τα τιτιβίσματα που έχουν ουδέτερο περιεχόμενο και για αυτά που για κάποιον λόγο δεν μπόρεσε να γίνει η συναισθηματική ανάλυση.
- *Color* (*string*) : Η συγκεκριμένη ιδιότητα εκφράζει το χρώμα που θα πάρει κάθε τιτίβισμα κατά την εμφάνιση του, ανάλογα με την συναισθηματική ανάλυση που του έγινε (βάση της ιδιότητας *Score*). Οι τιμές που παίρνει η συγκεκριμένη είναι σε δεκαεξαδική μορφή στο πρότυπο χρώματος *RGB* (Red Green Blue). Για παράδειγμα ένα κόκκινο τιτίβισμα θα πάρει την τιμή "`#ff0000`". Όσο πιο αρνητικό είναι το περιεχόμενο του τιτιβίσματος (πλησιάζει το -1), τόσο πιο έντονο κόκκινο είναι το χρώμα που θα πάρει (πλησιάζει το "`#ff0000`"). Αντίθετα, όσο πιο θετικό είναι το περιεχόμενο, τόσο πιο έντονο πράσινο το αντίστοιχο χρώμα (η τιμή του πλησιάζει το "`#00ff00`"). Τέλος οι τιμές 0 θα πάρουν μπλε χρώμα ("`#0000ff`").

Να τονίσουμε εδώ ότι προκειμένου να βελτιώσουμε την απόδοση των χωρικών ερωτημάτων πάνω στους κόμβους *Tweet*, δημιουργήσαμε το χωρικό ευρετήριο (*spatial index*) *tweetWKT*.

3. ***HashTag*** : Σε αυτήν την οντότητα ανήκουν όλες οι λέξεις οι οποίες έχουν χρησιμοποιηθεί ως *hashtag* σε κάποιο τιτίβισμα. Κάθε κόμβος *HashTag* περιέχεται υποχρεωτικά σε ένα τουλάχιστον τιτίβισμα. Η μόνη ιδιότητα που έχουν οι κόμβοι αυτής της οντότητας είναι η *Name* (*string*). Η ιδιότητα αυτή το όνομα του συγκεκριμένου *hashtag*.

Οι ετικέτες σχέσεων ή τα είδη σχέσεων της βάσης δεδομένων μας είναι οι εξής:

1. ***TWEETS*** : Η συγκεκριμένη σχέση συνδέει έναν χρήστη με ένα τιτίβισμα, δηλαδή έναν κόμβο *User* με έναν κόμβο *Tweet*. Ο κόμβος αρχής είναι ο χρήστης και ο κόμβος πέρατος το τιτίβισμα. Ουσιαστικά έχει κατεύθυνση από τον χρήστη στο τιτίβισμα. Κάθε κόμβος *Tweet* έχει ακριβώς μια εισερχόμενη σχέση αυτού

του τύπου. Αντίθετα ένας κόμβος *User* μπορεί να έχει περισσότερες από μια εξερχόμενες τέτοιες σχέσεις. Η σχέση *TWEETS* δεν περιέχει κάποια ιδιότητα.

2. ***HAS*** : Η σχέση *HAS* συνδέει ένα τιτίβισμα με ένα *hashtag* (έναν κόμβο *Tweet* με έναν κόμβο *HashTag*). Η κατεύθυνσή της σχέσης είναι από το τιτίβισμα προς το *hashtag*. Κάθε κόμβος *Tweet* μπορεί να έχει από καμία μέχρι πολλές τέτοιες εξερχόμενες σχέσεις. Οι κόμβοι *HashTag* έχουν τουλάχιστον μια τέτοια εισερχόμενη σχέση. Η σχέση *HAS* δεν περιέχει κάποια ιδιότητα.

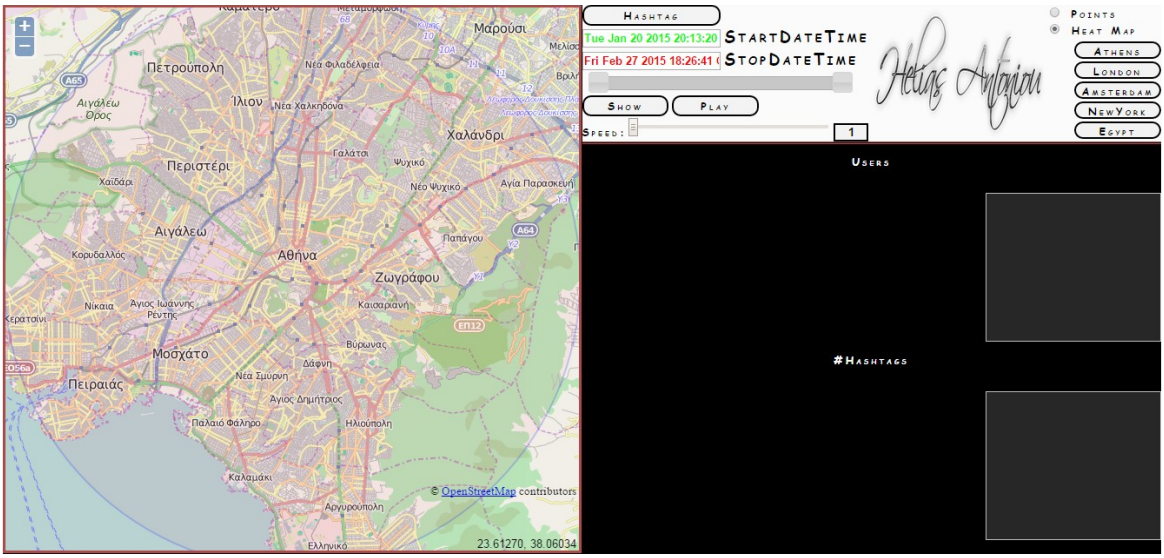
3. ***MATCHES*** : Η σχέση *MATCHES* συνδέει δύο κόμβους *HashTag* μεταξύ τους. Μια τέτοια σχέση δημιουργείται όταν δύο *hashtag* εμφανίζονται στο ίδιο τιτίβισμα. Θα θέλαμε η συγκεκριμένη σχέση να μην έχει κατεύθυνση. Στην *Neo4j* (και σε κάθε άλλη βάση δεδομένων γράφου με ιδιότητες) όμως κάτι τέτοιο δεν είναι εφικτό˙ κάθε σχέση έχει υποχρεωτικά έναν κόμβο αρχής και έναν κόμβο πέρατος και συνεπώς κατεύθυνση. Για να άρουμε αυτόν τον περιορισμό θα μπορούσαμε να κάνουμε την σχέση διπλής κατεύθυνσης δημιουργώντας δύο σχέσεις αντί για μία. Έτσι κάθε κόμβος *HashTag* θα ήταν κόμβος αρχής στην μια σχέση και κόμβος πέρατος στην άλλη. Αυτή η υλοποίηση δεν είναι αποδοτική καθώς απαιτεί την δημιουργία μεγάλου αριθμού σχέσεων χωρίς καμία πληροφορία [*GH*]. Εκμεταλλευόμενοι το γεγονός ότι η *Neo4j* μας επιτρέπει να κάνουμε ερωτήματα (*queries*) στην βάση χωρίς να ενδιαφερόμαστε για την κατεύθυνση μιας σχέσης, αποφασίσαμε να δημιουργήσουμε μια μόνο σχέση με τυχαία κατεύθυνση. Δηλαδή επιλέγουμε τυχαία έναν από τους δύο κόμβους ως κόμβο αρχής και τον άλλο ως κόμβο πέρατος. Η σχέση *MATCHES* έχει μία ιδιότητα : *Frequency* (*integer*). Αυτή η ιδιότητα είναι μια μετρική για το πόσες φορές τα δύο *hashtags* έχουν βρεθεί στο ίδιο τιτίβισμα. Μεγάλη τιμή αυτής της ιδιότητας σημαίνει στενή εννοιολογική σχέση μεταξύ των δύο *hashtag*. Για παράδειγμα η σχέση *MATCHES* για τα hashtags "*wc2014*" και "*brazil*" είχε πολύ μεγάλη τιμή Frequency κατά την διάρκεια του παγκοσμίου κυπέλλου ποδοσφαίρου στην Βραζιλία το 2014. Αυτό σημαίνει ότι πάρα πολλά τιτιβίσματα περιείχαν και τα δύο αυτά *hashtags*.

4. ***NEXT*** : Η σχέση *NEXT* συνδέει δύο κόμβους *Tweet* μεταξύ τους. Ουσιαστικά δημιουργεί την αλυσίδα των τιτιβισμάτων στον χρόνο. Η κατεύθυνση της σχέσης εδώ έχει μεγάλη σημασία. Ο κόμβος αρχής είναι το τιτίβισμα που προηγείται χρονικά (έχει μικρότερη τιμή στην ιδιότητα *Milliseconds*). Κάθε κόμβος *Tweet* έχει ακριβώς μία εισερχόμενη και μία εξερχόμενη τέτοια σχέση. Εξαιρείται ο πρώτος χρονικά κόμβος *Tweet* της βάσης καθώς έχει μόνο εξερχόμενη σχέση, και ο τελευταίος κόμβος που έχει μόνο εισερχόμενη. Η σχέση *NEXT* δεν έχει κάποια ιδιότητα.

5. ***FOLLOWS*** : Η σχέση *FOLLOWS* συνδέει δύο κόμβους *User* μεταξύ τους. Ουσιαστικά είναι υπεύθυνη για την δημιουργία του γράφου ακολούθων (*followers graph*) στην βάση δεδομένων μας. Η κατεύθυνση της σχέσης έχει σημασία. Ο κόμβος αρχής της σχέσης δηλώνει τον χρήστη ο οποίος ακολουθεί κάποιον άλλο χρήστη. Αντίθετα ο κόμβος πέρατος της σχέσης δηλώνει τον χρήστη ο οποίος ακολουθείται. Κάθε κόμβος *User* μπορεί να έχει καμία ή περισσότερες τόσο εισερχόμενες όσο και εξερχόμενες σχέσεις *FOLLOWS*.

7 Παρουσίαση

Στο σημείο αυτό είμαστε σε θέση να περιγράψουμε τις βασικές λειτουργίες που μπορεί να εκτελέσει ένας χρήστης μέσω της εφαρμογής μας. Αρχικά, εκκινώντας την εφαρμογή εμφανίζεται η αρχική σελίδα της (σχήμα 7.1). Στο αριστερό μέρος αυτής υπάρχει ένας χάρτης (*openlayers*), που εμφανίζει την περιοχή της Αθήνας (μία από τις πέντε περιοχές ενδιαφέροντος). Η πόλη της Αθήνας επιλέχθηκε τυχαία, και όπως θα δούμε στη συνέχεια, η εναλλαγή μεταξύ των περιοχών ενδιαφέροντος είναι μια πολύ εύκολη διαδικασία. Στο δεξί μέρος της σελίδα υπάρχουν διάφορες επιλογές αναζήτησης (πάνω δεξιά) καθώς και εργαλεία οπτικοποίησης των δεδομένων μας (κάτω δεξιά), όπως σύννεφα λέξεων (*tag clouds*) και γραφήματα πίτας (*donut charts*).



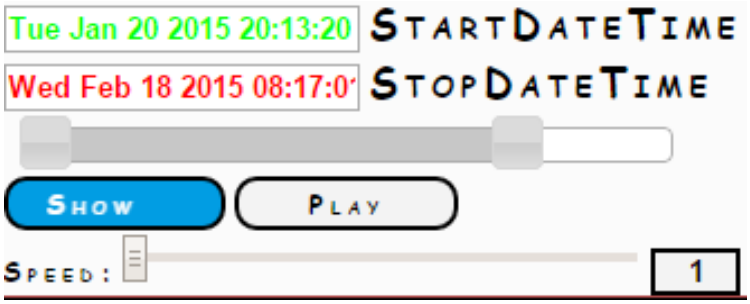
Σχήμα *7.1* – Αρχική σελίδα της εφαρμογής μας

Να σημειώσουμε εδώ ότι η εφαρμογή ξεκινάει να φέρνει τιτιβίσματα (*tweets*) σε πραγματικό χρόνο (*real time*) και τα εμφανίζει στο χάρτη αλλά και στα γραφήματα οπτικοποίησης, χωρίς ο χρήστης να επιλέξει κάτι. Αυτό γίνεται επειδή η εμφάνιση δεδομένων πραγματικού χρόνου είναι η προκαθορισμένη και βασική λειτουργία της εφαρμογής μας.

Πέρα όμως από τα δεδομένα πραγματικού χρόνου, η εφαρμογή δίνει στο χρήστη τη δυνατότητα να τοποθετήσει την αναζήτηση του σε δεδομένα του παρελθόντος. Στο σχήμα 7.2 φαίνονται οι επιλογές αναζήτησης για δεδομένα του παρελθόντος.

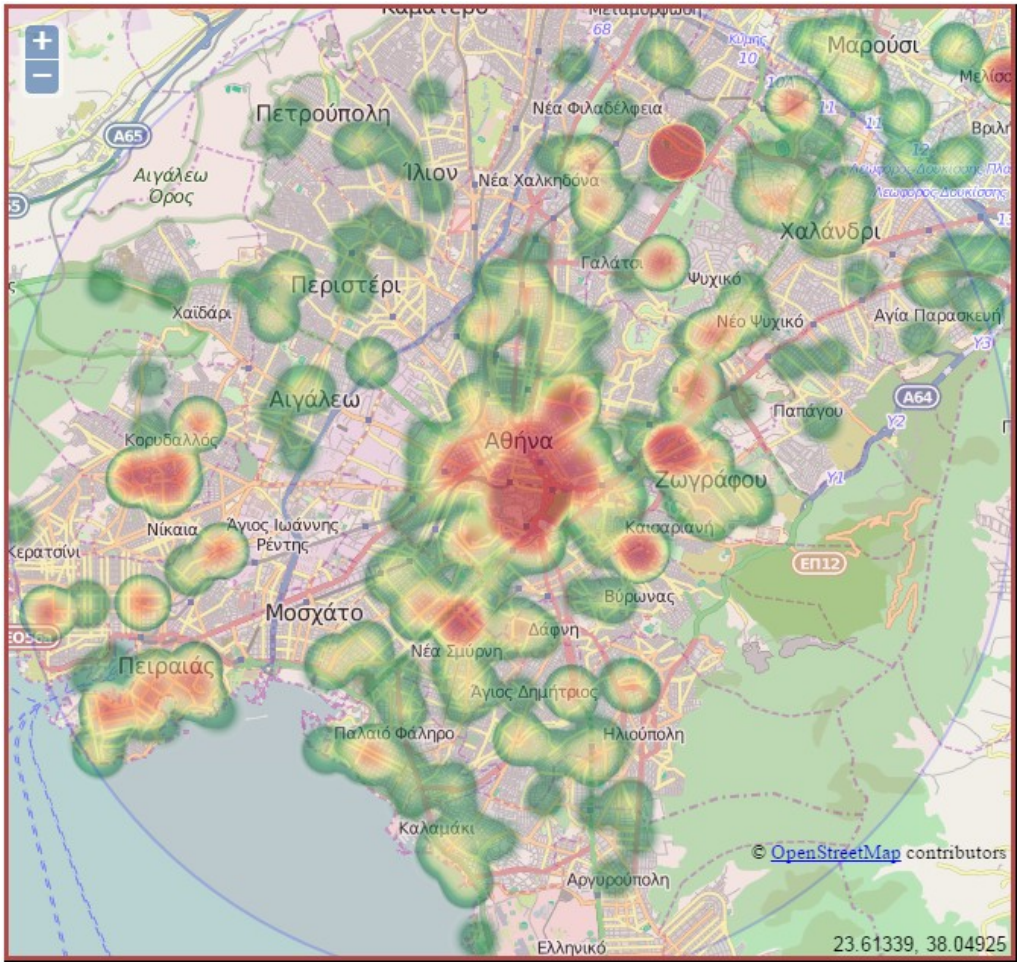
Συγκεκριμένα με την χρήση ενός εργαλείου ολίσθησης (*slider*) δίνεται η δυνατότητα στο χρήστη να επιλέξει ένα συγκεκριμένο χρονικό διάστημα στο παρελθόν για να κάνει την αναζήτηση του. Για παράδειγμα, έστω ότι ο χρήστης θέλει να βρει όλα τα τιτιβίσματα

που έγιναν στο χρονικό διάστημα από τις 20 Ιανουαρίου 2015 (20:13:20) έως τις 18 Φεβρουαρίου 2015 (08:17:01). Παρατηρούμε ότι μετακινώντας τα δύο άκρα του εργαλείου ολίσθησης ενημερώνονται αυτόματα οι τιμές των `StartDateTime` και `StopDateTime`. Στο σχήμα 7.2 φαίνονται οι αντίστοιχες επιλογές αναζήτησης.



Σχήμα 7.2 – Επιλογές αναζήτησης δεδομένων που τοποθετούνται χρονικά στο παρελθόν

Με την επιλογή `SHOW`, εμφανίζονται στον χάρτη όλα τα τιτιβίσματα που έγιναν στο συγκεκριμένο χρονικό διάστημα για την συγκεκριμένη περιοχή ενδιαφέροντος (εδώ η πόλη της Αθήνας). Στο σχήμα 7.3 φαίνονται τα αποτελέσματα σε μορφή χάρτη θερμότητας (*heatmap*).

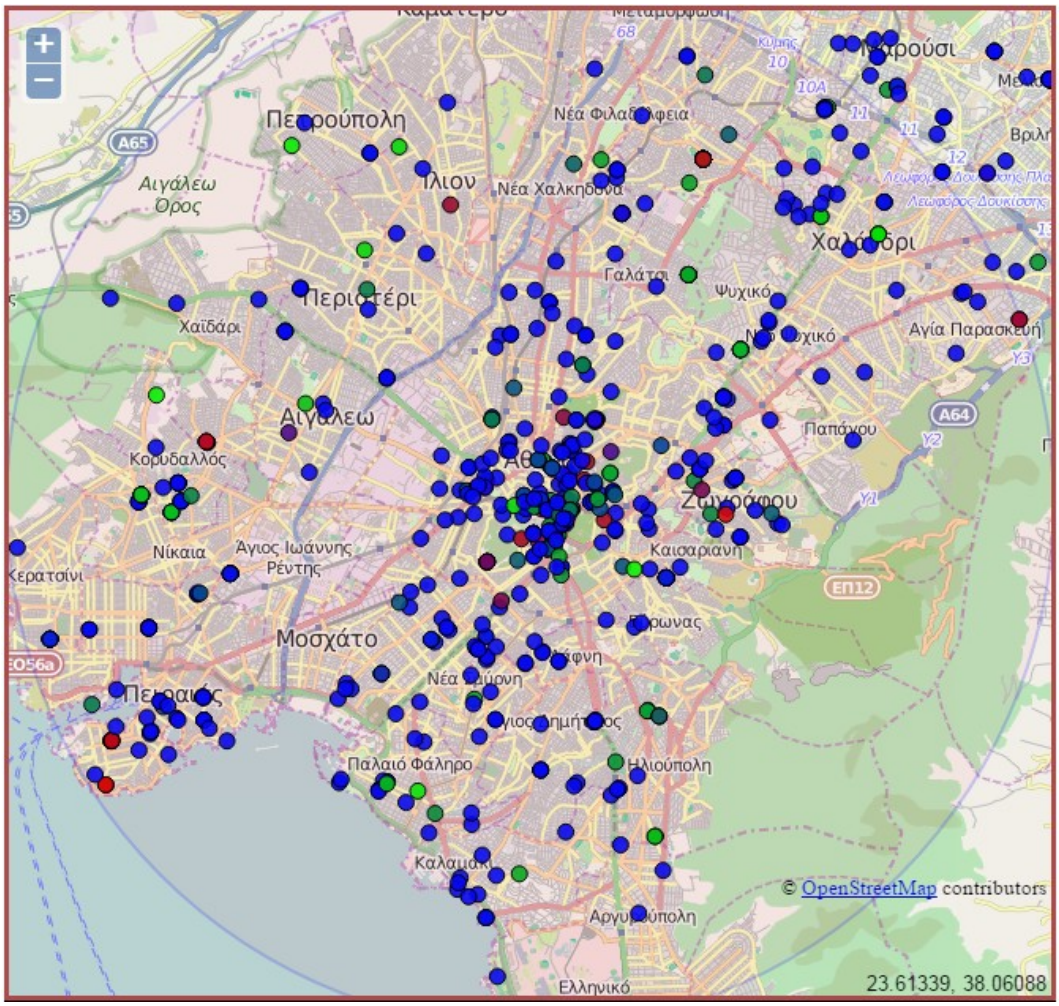


Σχήμα 7.3 – *Heatmap* για αποτελέσματα αναζήτησης τιτιβισμάτων του παρελθόντος

Ο χάρτης θερμότητας μας δίνει πληροφορία για την διασπορά των τιτιβισμάτων στο χώρο. Κόκκινο χρώμα αντιπροσωπεύει μεγάλη πυκνότητα τιτιβισμάτων σε μια περιοχή, ενώ αντίθετα η απουσία χρώματος σημαίνει και απουσία τιτιβισμάτων.

Σχήμα *7.4* – Επιλογές παρουσίασης τιτιβισμάτων στο χάρτη

Η εναλλαγή της παρουσίασης μεταξύ χάρτη θερμότητας και χάρτη σημείων, όπου το κάθε τιτίβισμα είναι ένα σημείο στο χάρτη, γίνεται μέσω της σχετικής επιλογής (σχήμα 7.4). Στο σχήμα 7.5 φαίνονται τα αποτελέσματα της προηγούμενης αναζήτησης σε μορφή χάρτη σημείων. Ουσιαστικά, ο χάρτης θερμότητας του σχήματος 7.3 και ο χάρτης σημείων του σχήματος 7.5 αποτελούν το αποτέλεσμα της ίδιας ακριβώς αναζήτησης (αυτής του σχήματος 7.2), εκφρασμένης σε διαφορετική μορφή.



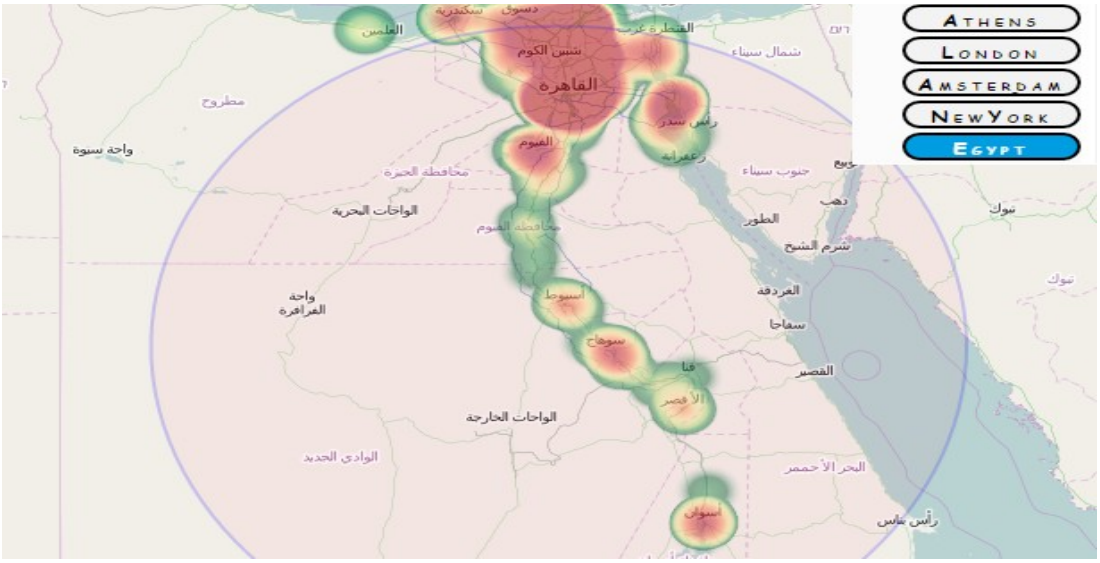
Σχήμα *7.5* – Χάρτης σημείων για αποτελέσματα αναζήτησης τιτιβισμάτων του παρελθόντος

Στον χάρτη σημείων γίνεται φανερή η ανάλυση συναισθημάτων που έχει γίνει στα κείμενα των τιτιβισμάτων (*sentiment analysis*). Το χρώμα του κάθε σημείου αντιπροσωπεύει την πολικότητα (θετικό, αρνητικό ή ουδέτερο) στο κείμενο του αντίστοιχου τιτιβίσματος. Έντονα κόκκινο χρώμα αντιπροσωπεύει το αρνητικό συναίσθημα, ενώ έντονα πράσινο χρώμα το θετικό συναίσθημα. Τέλος, το μπλε χρώμα αντιπροσωπεύει είτε την απουσία συναισθήματος από το κείμενο του τιτιβίσματος

(ουδέτερο) είτε τα τιτιβίσματα που δεν μπόρεσε να γίνει η ανάλυση. Ο μεγάλος αριθμός μπλε τιτιβισμάτων οφείλεται στο γεγονός ότι το *AlchemyAPI* δεν υποστηρίζει την ελληνική γλώσσα (η γλώσσα των περισσοτέρων τιτιβισμάτων για την πόλη της Αθήνας) και συνεπώς για αυτά δεν μπόρεσε να γίνει ανάλυση. Βάζοντας το δρομέα του ποντικιού πάνω σε κάποιο τιτίβισμα ανοίγει ένα παράθυρο με περαιτέρω πληροφορίες για το τιτίβισμα αυτό (κείμενο, χρήστης, ακριβής χρονική στιγμή δημιουργίας κτλ). Ένα τέτοιο παράδειγμα φαίνεται στο σχήμα 7.6.

Σχήμα *7.6* – Πληροφορίες για κάποιο τιτίβισμα της επιλογής μας

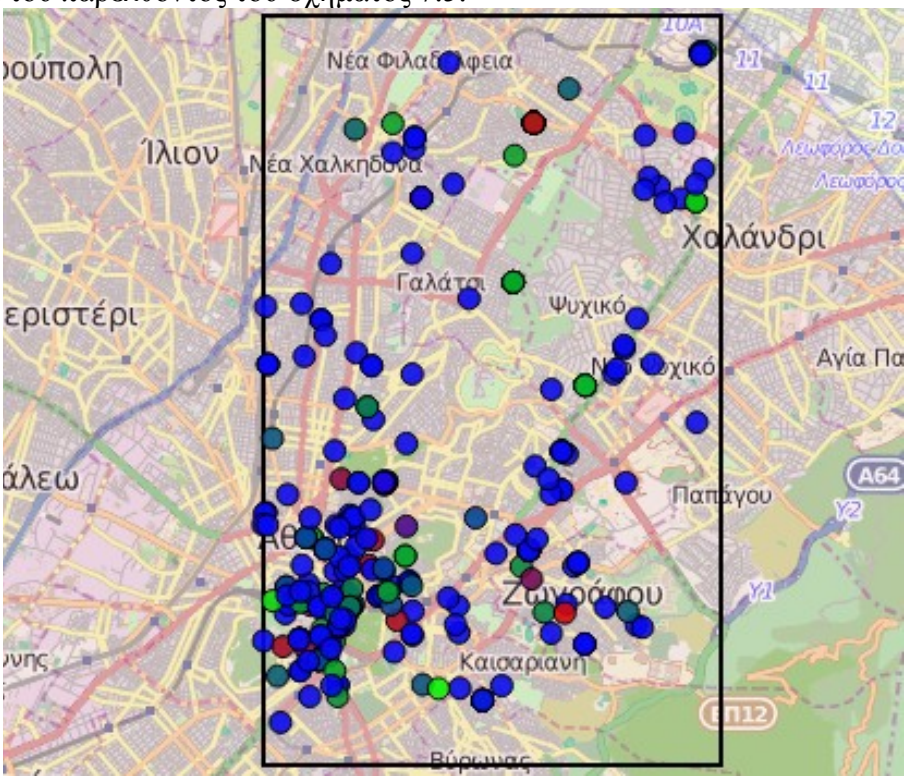
Πέρα από τη δυνατότητα να εμφανίσει μαζικά δεδομένα που έγιναν σε ένα συγκεκριμένο χρονικό διάστημα στο παρελθόν, η εφαρμογή δίνει στο χρήστη τη δυνατότητα να προσομοιώσει τη χρονική ροή δημιουργίας των δεδομένων αυτών. Με την επιλογή `PLAY` (σχήμα 7.2) εμφανίζονται στο χάρτη τα τιτιβίσματα που έγιναν σε μία περιοχή ενδιαφέροντος, ξεκινώντας από μια χρονική στιγμή στο παρελθόν (στο παράδειγμα του σχήματος 7.2 στις 20 Ιανουαρίου 2015) και προσομοιώνοντας ροή πραγματικού χρόνου. Δηλαδή, κάθε τιτίβισμα εμφανίζεται στον χάρτη με βάση τη χρονική στιγμή δημιουργίας του. Με την επιλογή `SPEED` (σχήμα 7.2) μπορούμε να αλλάξουμε τον ρυθμό εμφάνισης των δεδομένων. Η προκαθορισμένη τιμή είναι `1`, ενώ η μέγιστη είναι `20`. Η επιλογή `20` σημαίνει ότι τα δεδομένα εμφανίζονται 20 φορές ταχύτερα σε σχέση με τον πραγματικό ρυθμό δημιουργίας τους.



Σχήμα *7.7* – Επιλογή της Αιγύπτου ως περιοχή ενδιαφέροντος

Πέρα όμως από τη δυνατότητα αναζήτησης δεδομένων στο παρελθόν, η εφαρμογή δίνει στο χρήστη τη δυνατότητα να τοποθετήσει την αναζήτηση του στο χώρο. Στο σχήμα 7.7 φαίνεται ο τρόπος με τον οποίο ο χρήστης μπορεί να μετακινηθεί ανάμεσα στις 5 περιοχές ενδιαφέροντος.

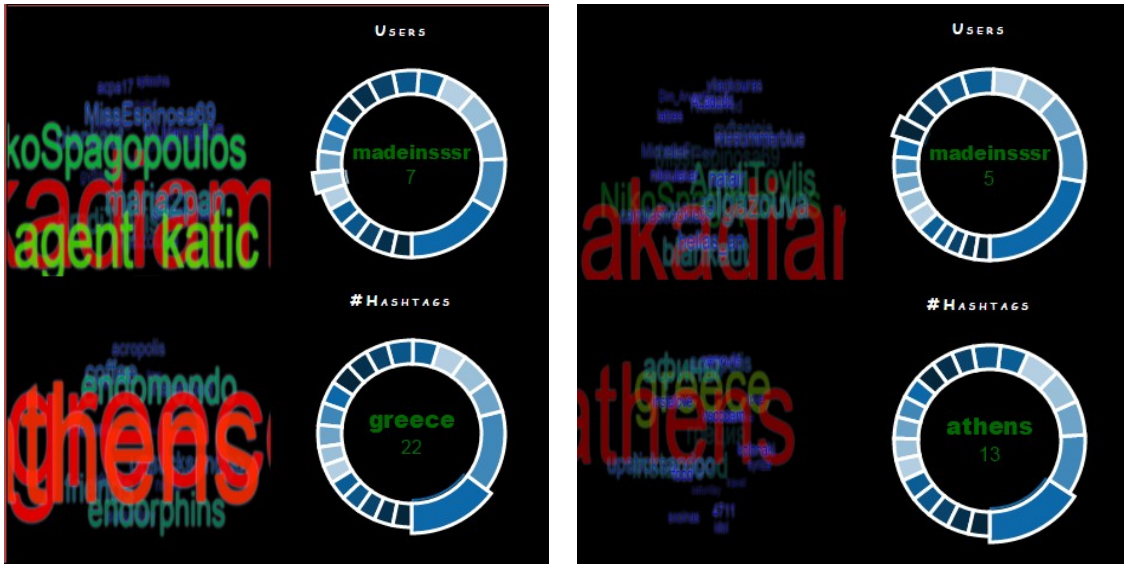
Μπορούμε όμως να περιορίσουμε την αναζήτηση μας ακόμα περισσότερο, επιλέγοντας ως περιοχή ενδιαφέροντος ένα τμήμα από την αρχική περιοχή (Αθήνα, Λονδίνο, Άμστερνταμ, Νέα Υόρκη και Αίγυπτος). Αυτό γίνεται στο χάρτη κρατώντας πατημένο το αριστερό *Shift* και κάνοντας κλικ και σύρσιμο με το ποντίκι στην περιοχή που επιθυμούμε. Να σημειώσουμε εδώ ότι οι αναζητήσεις στο χώρο και στο χρόνο μπορούν να συνδυαστούν. Μπορεί δηλαδή ο χρήστης να επιλέξει οποιοδήποτε χρονικό παράθυρο και ταυτόχρονα να διαλέξει το περιβάλλον πλαίσιο στο οποίο επιθυμεί να επικεντρωθεί. Στο σχήμα 7.8 παρατηρούμε τον περιορισμό της περιοχής ενδιαφέροντος για τα δεδομένα του παρελθόντος του σχήματος 7.5.



Σχήμα *7.8* – Περιορισμός της περιοχής ενδιαφέροντος για την πόλη της Αθήνας

Ταυτόχρονα και σε πλήρη αρμονία με την εμφάνιση των δεδομένων στο χάρτη η εφαρμογή εμφανίζει ένα σύννεφο λέξεων (*tag cloud*) και ένα γράφημα πίτας (*donut chart*), τόσο με τους πιο ενεργούς χρήστες (χρήστες με τα περισσότερα τιτιβίσματα) όσο και με τα πιο δημοφιλή *hashtags* (*hashtags* που εμφανίζονται περισσότερες φορές στα τιτιβίσματα). Η εμφάνιση των χρηστών και των *hashtags* αντιστοιχεί στα τιτιβίσματα που φαίνονται στο χάρτη. Είναι εναρμονισμένη δηλαδή, τόσο με τους χρονικούς αλλά και τους χωρικούς περιορισμούς που έχει θέσει ο χρήστης. Στο σχήμα 7.9 βλέπουμε τα σύννεφα λέξεων και τα αντίστοιχα γραφήματα πίτας που αντιστοιχούν σε τιτιβίσματα του παρελθόντος για την πόλη της Αθήνας. Στα αριστερά οι πληροφορίες αφορούν ολόκληρη την πόλη της Αθήνας (χάρτης του σχήματος 7.5), ενώ στα δεξιά οι

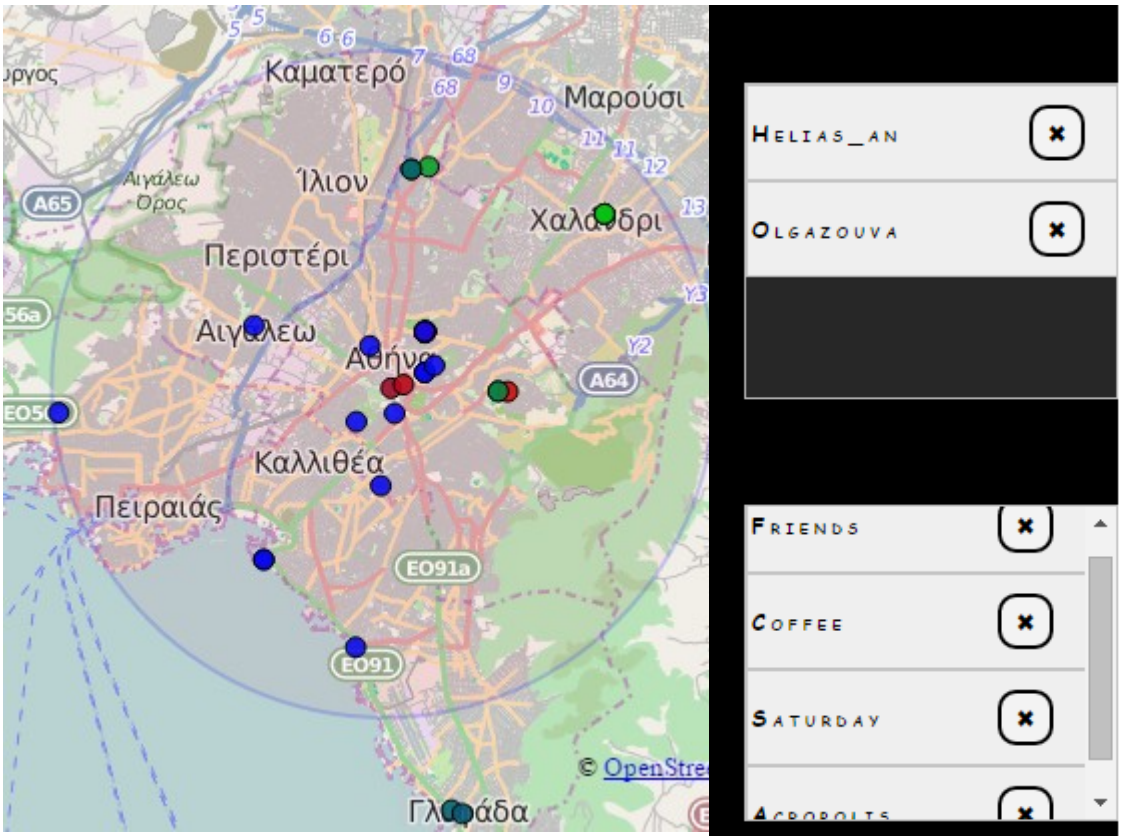
πληροφορίες αφορούν την περιορισμένη περιοχή ενδιαφέροντος του σχήματος 7.8. Παρατηρούμε ότι στη πρώτη περίπτωση ο χρήστης με όνομα "`madeinsssr`" έχει δημιουργήσει 7 τιτιβίσματα, ενώ στη δεύτερη περίπτωση έχει δημιουργήσει 5. Δηλαδή, δύο από τα τιτιβίσματα του συγκεκριμένου χρήστη βρίσκονται έξω απο την περιορισμένη περιοχή ενδιαφέροντος που φαίνεται στο χάρτη του σχήματος 7.8, και συνεπώς δεν συμπεριλαμβάνονται στα γραφήματα του δεξιού μέρους του σχήματος 7.9. Αντίστοιχα, για ολόκληρη τη περιοχή της Αθήνας το δημοφιλέστερο *hashtag* είναι το "`#greece`" (συχνότητα 22), ενώ όταν περιορίσουμε την περιοχή ενδιαφέροντος στο κέντρο της πόλης αυτό είναι το "`#athens`" (συχνότητα 13).



Σχήμα *7.9 - Tag Clouds* και *donut charts* για τους πλέον ενεργούς χρήστες και *hashtags* που αντιστοιχούν στα σχήματα *7.5* και *7.8*

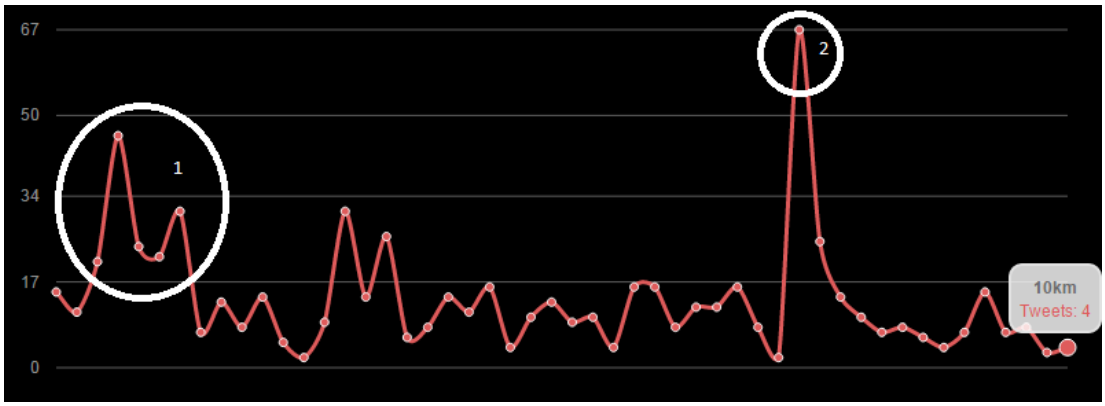
Τέλος ο χρήστης μπορεί να περιορίσει την αναζήτηση του ακόμα περισσότερο με βάση τόσο το *hashtag* όσο και το χρήστη που επιθυμεί. Δηλαδή μπορεί να επιλέξει να εμφανίσει όλα τα τιτιβίσματα που περιέχουν κάποιο συγκεκριμένο *hashtag*, όσο και αυτά που έχουν κάποιο συγκεκριμένο χρήστη-δημιουργό. Μπορεί να γίνει και συνδυασμός των δύο παραπάνω κατηγοριών. Σε μια τέτοια περίπτωση, όπου δηλαδή ο χρήστης επιλέγει να εμφανίσει τιτιβίσματα που αντιστοιχούν σε περισσότερους από έναν χρήστες ή *hashtags,* η εμφάνιση των αποτελεσμάτων αποτελεί το λογικό Ή (*logical OR*) των επιμέρους αναζητήσεων. Στο σχήμα 7.10 φαίνεται η επιλογή για εμφάνιση δύο χρηστών (`helias_an` και `olgazouva`) και τεσσάρων *hashtags* (`#friends`, `#coffee`, `#saturday` και `#acropolis`).

Αυτές είναι οι βασικές επιλογές αναζήτησης τιτιβισμάτων στην εφαρμογή μας. Μπορούμε δηλαδή να αναζητήσουμε τιτιβίσματα με βάση τη χρονική στιγμή και την ακριβή τοποθεσία δημιουργίας τους, με βάση το χρήστη-δημιουργό τους και με βάση τα *hashtags* που αυτά περιέχουν.



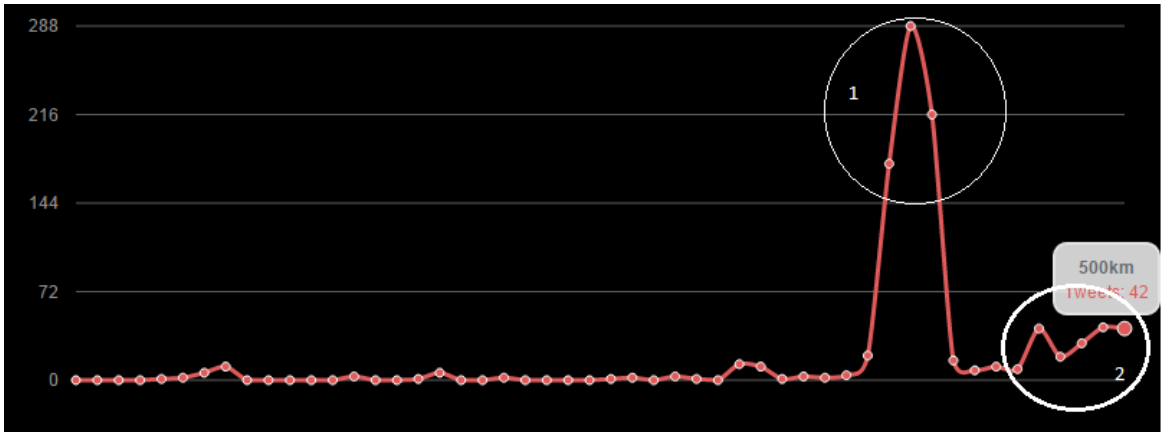
Σχήμα *7.10* – Αναζήτηση με βάση το χρήστη-δημιουργό και το *hashtag*

Στην συνέχεια η εφαρμογή παρουσιάζει στο χρήστη διάφορες ενδιαφέρουσες πληροφορίες μέσω γραφημάτων. Στο γράφημα του σχήματος 7.11 φαίνεται η κατανομή των τιτιβισμάτων βάση της απόστασης τους από το κέντρο της Αθήνας, σε ακτίνα έως 10 χιλιομέτρων. Παρατηρούμε ότι μεγάλος αριθμός τιτιβισμάτων προέρχονται από το κέντρο της πόλης (κύκλος 1) κάτι που είναι σε συμφωνία με τον χάρτη θερμότητας του σχήματος 7.3. Σε απόσταση 7.4 χιλιομέτρων από το κέντρο της πόλης παρατηρούμε μια απότομη αύξηση των τιτιβισμάτων (κύκλος 2). Αυτή οφείλεται σε κάποιον μεμονωμένο χρήστη που έχει δημιουργήσει μεγάλο αριθμό τιτιβισμάτων από ένα συγκεκριμένο μέρος. Αυτός ο χρήστης είναι το έντονο κόκκινο χρώμα που παρατηρούμε στο χάρτη θερμότητας στην περιοχή της Νέας Φιλαδέλφειας.



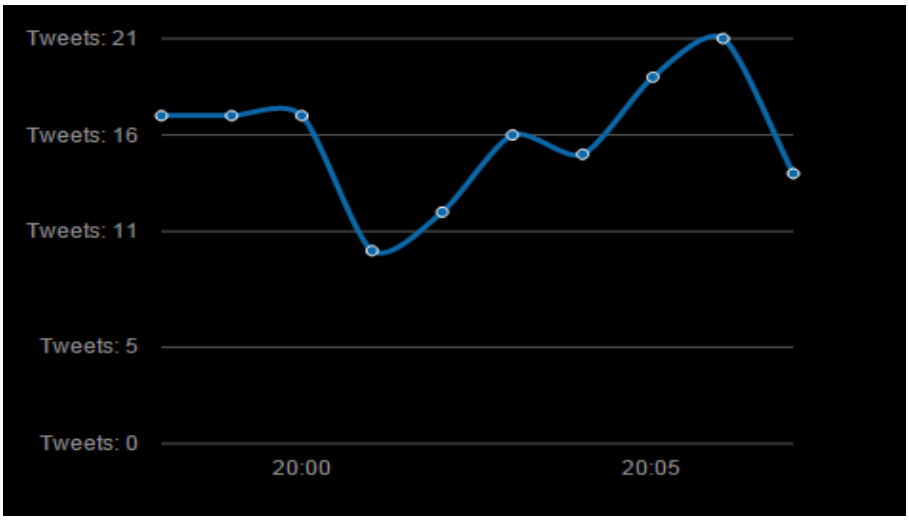
Σχήμα *7.11* – Κατανομή τιτιβισμάτων βάση της απόστασης τους από το κέντρο της Αθήνας

Στο σχήμα 7.12 παρατηρούμε την αντίστοιχη κατανομή για την περιοχή ενδιαφέροντος της Αιγύπτου, σε ακτίνα έως 500 χιλιομέτρων. Εδώ παρατηρούμε μια απότομη αύξηση των τιτιβισμάτων κοντά στα 400 χιλιόμετρα (κύκλος 1). Αυτή οφείλεται στο γεγονός ότι μεγάλο μέρος του πληθυσμού και συνεπώς τιτιβισμάτων προέρχονται από την περιοχή του Καΐρου (περίπου 700 τιτιβίσματα), ενώ το υπόλοιπο τμήμα της συγκεκριμένης περιοχής είναι καθόλου ή ελάχιστα κατοικημένο λόγω των ερήμων. Επίσης κοντά στα 500 χιλιόμετρα παρατηρούμε μια ανοδική τάση (κύκλος 2) που οφείλεται στη συγκέντρωση τιτιβισμάτων στις παραθαλάσσιες περιοχές και κυρίως στις εκβολές του Νείλου. Όλες αυτές οι παρατηρήσεις είναι σε αρμονία με το χάρτη θερμότητας του σχήματος 7.7.



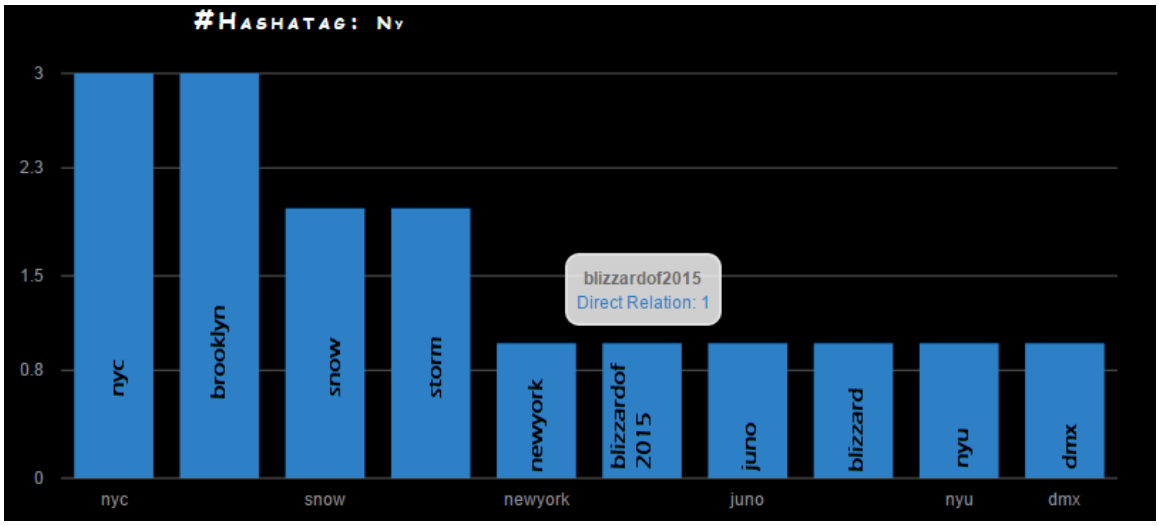
Σχήμα *7.12* – Κατανομή τιτιβισμάτων για την περιοχή της Αιγύπτου

Στο γράφημα του σχήματος 7.13 φαίνεται η κατανομή των τιτιβισμάτων στο χρόνο για την περιοχή ενδιαφέροντος (στο σχήμα το Λονδίνο). Ουσιαστικά πρόκειται για ένα γράφημα που μας δείχνει το πόσο ενεργοί (συχνότητα που δημιουργούν τιτιβίσματα) είναι οι χρήστες της επιλεγμένης περιοχής τα τελευταία 10 λεπτά. Το συγκεκριμένο γράφημα ανανεώνεται αυτόματα κάθε μισό λεπτό, κρατώντας τον χρήστη συνεχώς ενήμερο για την δραστηριότητα των χρηστών μιας περιοχής.

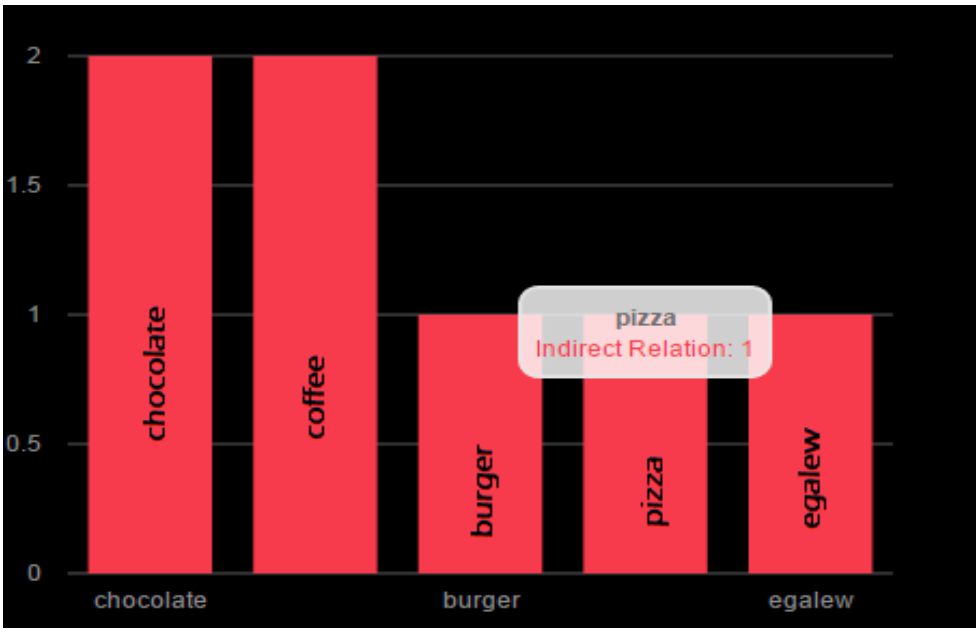


Σχήμα *7.13* – Κατανομή τιτιβισμάτων στο χρόνο για την περιοχή του Λονδίνου

Μια ενδιαφέρουσα πηγή πληροφορίας αφορά τη σχέση μεταξύ των *hashtags* των τιτιβισμάτων. Για παράδειγμα, *hashtags* που εμφανίζονται στο ίδιο τιτίβισμα με μεγάλη συχνότητα ή *hashtags* τα οποία αν και δεν εμφανίζονται στο ίδιο τιτίβισμα, εμφανίζονται σε διαφορετικά τιτιβίσματα του ίδιου χρήστη κρύβουν κάποιας μορφής σχέση. Στο σχήμα 7.14 φαίνονται τα *hashtags* τα οποία εμφανίζονται με μεγαλύτερη συχνότητα στο ίδιο τιτίβισμα με το *hashtag* "#ny" (αποτελεί ακρωνύμιο για την πόλη της Νέας Υόρκης – *New York*). Παρατηρούμε ότι πολλά από τα *hashtags* αυτά (`#snow`, `#storm`, `#blizzard`, `#blizzardof2015`) αφορούν τα έντονα καιρικά φαινόμενα που έπληξαν την πόλη της Νέας Υόρκης το χειμώνα του 2015.



Σχήμα *7.14 - Hashtags* που εμφανίζονται συχνότερα στο ίδιο τιτίβισμα με το *hashtag "ny"*



Σχήμα *7.15 - Hashtags* που εμφανίζονται συχνότερα σε διαφορετικά τιτιβίσματα κάποιου χρήστη με το *hashtag "saturday"*

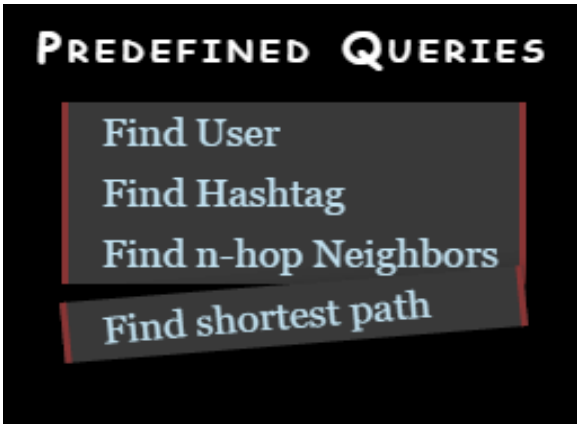
Στο σχήμα 7.15 φαίνονται τα *hashtags* τα οποία αν και δεν εμφανίζονται στο ίδιο

τιτίβισμα, εμφανίζονται σε διαφορετικά τιτιβίσματα του ίδιου χρήστη με το *hashtag* "`#saturday`". Εδώ αναμένουμε μια όχι τόσο άμεση σχέση μεταξύ των *hashtags*, τουλάχιστον όχι τόσο άμεση όσο με τα *hashtags* της προηγούμενης περίπτωσης (αυτά που εμφανίζονται στο ίδιο τιτίβισμα).

Κάθε φορά που ένας χρήστης αλλάζει περιοχή ενδιαφέροντος η εφαρμογή φορτώνει ένα στιγμιότυπο της βάσης δεδομένων με τιτιβίσματα της επιλεγμένης περιοχής. Στο σχήμα 7.16 φαίνεται ένα στιγμιότυπο της βάσης δεδομένων που περιλαμβάνει χρήστες, τιτιβίσματα και *hashtags*. Να σημειώσουμε εδώ ότι είναι απαραίτητο να περιορίσουμε το αποτέλεσμα του ερωτήματος σε ένα τμήμα της βάσης το πολύ 200 κόμβων. Χωρίς αυτόν τον περιορισμό (δηλαδή αν η εφαρμογή εμφάνιζε ολόκληρη τη βάση δεδομένων) το αποτέλεσμα θα ήταν δυσανάγνωστο για τον χρήστη, καθώς θα περιλάμβανε πληθώρα κόμβων και σχέσεων.

Σχήμα *7.16* – Ένα στιγμιότυπο της βάσης δεδομένων

Πέρα από την εμφάνιση ενός στιγμιότυπου, η εφαρμογή δίνει στο χρήστη τη δυνατότητα να αλληλεπιδράσει με την βάση δεδομένων μέσω απευθείας ερωτημάτων (*queries*) σε αυτήν. Τα ερωτήματα πρέπει να είναι στη γλώσσα ερωτημάτων *Cypher*. Ο χρήστης μπορεί να επιλέξει να εκτελέσει ένα ερώτημα της αρεσκείας του σε περίπτωση που είναι εξοικειωμένος με τη γλώσσα *Cypher* ή να επιλέξει κάποιο από τα προεπιλεγμένα ερωτήματα του σχήματος 7.17.

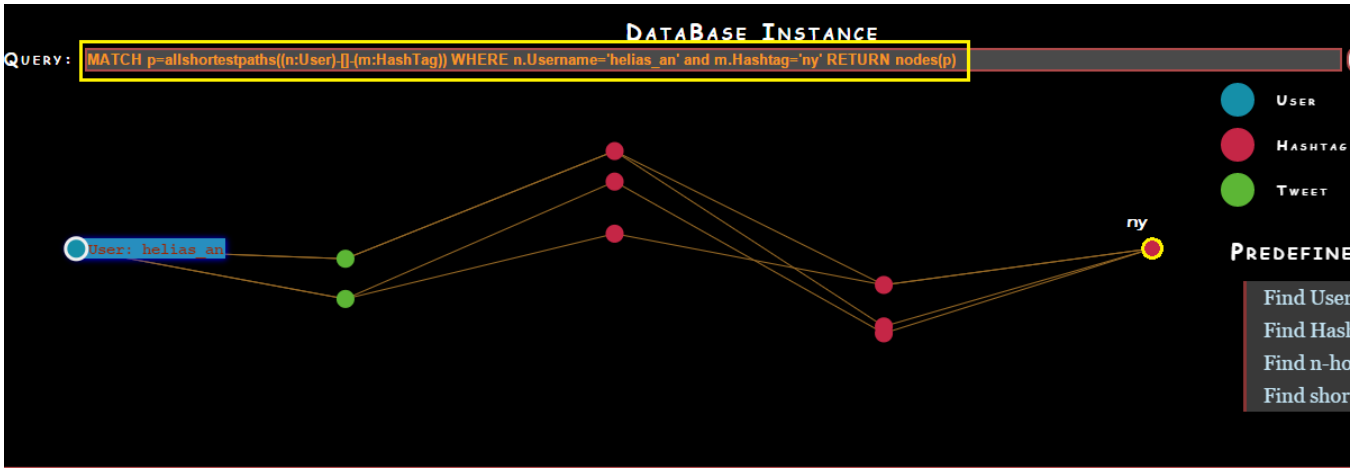


Σχήμα *7.17* – Τα προεπιλεγμένα ερωτήματα που είναι διαθέσιμα μέσω της εφαρμογής

Σε περίπτωση επιλογής κάποιου από τα προεπιλεγμένα ερωτήματα, αυτό εμφανίζεται έτοιμο στη γραμμή ερωτημάτων (κίτρινο πλαίσιο στο σχήμα 7.18). Ο χρήστης μπορεί να

αλλάξει τα κριτήρια αναζήτησης του ερωτήματος κατά βούληση και στη συνέχεια να εκτελέσει το ερώτημα μέσω του πλήκτρου `CALCULATE`.

Στο σχήμα 7.18 φαίνεται το αποτέλεσμα ενός ερωτήματος που υπολογίζει το συντομότερο μονοπάτι ανάμεσα στο χρήστη "`helias_an`" και το *hashtag* "`#ny`". Στο κίτρινο πλαίσιο στη γραμμή ερωτημάτων φαίνεται το ερώτημα που εκτελείται. Σε περίπτωση ύπαρξης περισσοτέρων του ενός συντομότερων μονοπατιών επιστρέφονται όλα.



Σχήμα *7.18*- Συντομότερα μονοπάτια ανάμεσα στον *user "helias_an"* και στο *hashtag "ny"*

Αυτές είναι οι βασικές δυνατότητες που παρέχει η εφαρμογή στους χρήστες της. Είδαμε λοιπόν, τον τρόπο με τον οποίο ο χρήστης μπορεί μέσω του χάρτη να ενημερώνεται σε πραγματικό χρόνο για τη ροή δημιουργίας χωρικών δεδομένων στο δίκτυο του *Twitter*. Μέσα από τα σύννεφα λέξεων και τα γραφήματα πίτας ο χρήστης ενημερώνεται ταυτόχρονα για τα δημοφιλέστερα *hashtags* καθώς και τους πλέον ενεργούς χρήστες.

Πέρα όμως από την παρουσίαση δεδομένων πραγματικού χρόνου, η εφαρμογή παρέχει στους χρήστες ένα εργαλείο αναζήτησης για δεδομένα του παρελθόντος. Η αναζήτηση δεδομένων με αυτόν τον τρόπο είναι αποδοτική καθώς εκμεταλλεύεται την ύπαρξη ενός χωρικού ευρετηρίου στα δεδομένα, και ταυτόχρονα φιλική στο χρήστη καθώς καθιστά εύκολη την επιλογή των χρονικών στιγμών αρχής και πέρατος της αναζήτησης όσο και τα χωρικά όρια αυτής. Τέτοιου είδους αναζήτηση θα μπορούσε να γίνει με τη χρήση του *Search API* (ενότητα 2.4.2). Σε αυτή τη περίπτωση θα χάναμε σημαντικά σε απόδοση λόγω τόσο των περιορισμών ορίου χρήσης (*rate limits*) όσο και των συνεχών αιτήσεων *HTTP* (*HTTP requests*) προς τη συγκεκριμένη διεπαφή.

Τέλος η εφαρμογή παρέχει ένα εργαλείο αλληλεπίδρασης με τη βάση δεδομένων. Με αυτό τον τρόπο ο χρήστης μπορεί να εξοικειωθεί με τη γλώσσα ερωτημάτων *Cypher*, καθώς και να κατανοήσει το υποκείμενο μοντέλο και τον τρόπο που συνδέονται τα δεδομένα μεταξύ τους σε μια βάση δεδομένων γράφου (*graph database*) όπως η *Neo4j*.

8 Επίλογος

Αντιλαμβανόμενοι την έκρηξη των δεδομένων που παρατηρείται στις μέρες μας και την αξία της ανάπτυξης εφαρμογών για την επεξεργασία και ανάλυση αυτών, στην εργασία αυτή παρουσιάσαμε ένα σύστημα για την διαχείριση των πληροφοριών που προέρχονται από την ιστοσελίδα κοινωνικής δικτύωσης του *Twitter*. Το σύστημα μας αποθηκεύει τιτιβίσματα (*tweets*) σε μια μη-σχεσιακή (*NoSQL*) βάση δεδομένων γράφου (*graph database*), αφού πρώτα τα αναλύσει τόσο ως προς το γεωγραφικό τους χαρακτηρισμό (*geo-tagging*) όσο και ως προς διάφορα γλωσσικά χαρακτηριστικά μέσω κάποιας μορφής γλωσσικής ανάλυσης (*lingual analysis*). Τελικά παρουσιάζει τα δεδομένα αυτά στον χρήστη με την βοήθεια χάρτη, γραφημάτων, σύννεφα λέξεων (*tag clouds*) και άλλων.

Μέσα από την εφαρμογή μας μπορούμε να αλληλεπιδράσουμε απευθείας με την βάση δεδομένων μέσω της γλώσσας ερωτημάτων *Cypher*. Δίνεται λοιπόν η δυνατότητα στον χρήστη, να ανασύρει από την βάση δεδομένων οποιαδήποτε πληροφορία θέλει μέσω του κατάλληλου ερωτήματος. Φυσικά για να γίνει αυτό πρέπει ο χρήστης να έχει μια γνώση τόσο της γλώσσας ερωτημάτων Cypher όσο και του σχήματος της βάση δεδομένων μας.

Η εφαρμογή μας έχει αρκετά περιθώρια βελτίωσης. Δεν πρέπει να ξεχνάμε τους περιορισμούς ορίου χρήσης (*rate limits*) που μας θέτει η διεπαφή προγραμματισμού εφαρμογών του *Twitter*, με συνέπεια ο γράφος των ακολούθων (*followers graph*) να σχηματίζεται με αργούς ρυθμούς. Ένας άλλος περιορισμός οφείλεται στο εργαλείο επεξεργασίας φυσικής γλώσσας (*Natural Language Processing – NLP)* που χρησιμοποιήσαμε. Το *AlchemyAPI* δεν υποστηρίζει όλες τις γλώσσες και μας θέτει ένα ανώτατο όριο 1000 κλήσεων ανά ημέρα. Αποτέλεσμα αυτού είναι να μην μπορούμε να εφαρμόσουμε ανάλυση συναισθημάτων (*sentiment analysis*) σε ένα μεγάλο αριθμό τιτιβισμάτων.

Μελλοντικά η εφαρμογή μας θα μπορούσε σίγουρα να επεκταθεί, τόσο στην κατεύθυνση μιας πληρέστερης και πιο στοχευμένης σε κείμενα που προέρχονται από το *Twitter* (μικρό μήκος κειμένου, χρήση συντακτικά λανθασμένων προτάσεων, συντμήσεων λέξεων και εκφράσεων συναισθήματος) ανάλυσης συναισθημάτων, όσο και στην κατεύθυνση του συνδυασμού της γεοχωρικής πληροφορίας των τιτιβισμάτων με την ανάλυση συναισθημάτων αυτών. Θα μπορούσαμε για παράδειγμα, να δημιουργήσουμε κάποιας μορφής πρόταση φίλων ή ακολούθων (*friend or followers suggestion*), που θα γίνεται λαμβάνοντας υπόψη όχι μόνο τα θέματα των τιτιβισμάτων των χρηστών αλλά και την γεωγραφική περιοχή που αυτά έγιναν.

Μια ενδιαφέρουσα επέκταση της εφαρμογής μας είναι η κατεύθυνση του τρόπου που διαχέονται τα δεδομένα στο χώρο, μέσω των αναδημοσιεύσεων (*retweets*). Στην ενότητα 2.1 είδαμε ότι το *Twitter* δεν επιτρέπει στους χρήστες να κάνουν αναδημοσίευση που να

συνοδεύεται από πληροφορία τοποθεσίας. Μια τέτοια πληροφορία μπορεί να γίνει γνωστή έμμεσα, μέσα από τα υπόλοιπα τιτιβίσματα ενός χρήστη. Αν για παράδειγμα ένας χρήστης δημοσιεύσει ένα τιτίβισμα που συνοδεύεται από γεωγραφική πληροφορία, και αμέσως μετά κάνει μια αναδημοσίευση, μπορούμε να συμπεράνουμε με σχετική ασφάλεια τη γεωγραφική θέση αυτής της αναδημοσίευσης.

9 Βιβλιογραφία

[S] statista.com/topics/1145/internet-usage-worldwide

[MSGL] Seth A. Myers, Aneesh Sharma, Pankaj Gupta, Jimmy Lin, "Information Network or Social Network? The Structure of the Twitter Follow Graph", April 2014.

[FBC] newsroom.fb.com/company-info/

[ATC] about.twitter.com/company

[ML] Seth A. Myers, Jure Leskovec, "The Bursty Dynamics of the Twitter Information Network", 11 March 2014.

[PPF] Evangelos Papalexakis, Konstantinos Pelechrinis, Christos Faloutsos, "Spotting Misbehaviors in Location-based Social Networks using Tensors", April 2014.

[KSH] Ricardo Kawase, Patrick Siehndel, Eelco Herder, "Haters Gonna Hate: Job-Related Offenses in Twitter", 2014.

[ZGWS] Emilio Zagheni, Venkata Rama Kiran Garimella, Ingmar Weber, Bogdan State, "Inferring International and Internal Migration Patterns from Twitter Data", 2014.

[MPL] Fred Morstatter, Jürgen Pfeffer, Huan Liu, "When is it Biased? Assessing the Representativeness of Twitter's Streaming API", January 2014.

[MPLC] Fred Morstatter, Jürgen Pfeffer, Huan Liu, Kathleen M. Carley, "Is the Sample Good Enough? Comparing Data from Twitter's Streaming API with Twitter's Firehose", 2013.

[FM] firstmonday.org/ojs/index.php/fm/article/view/4366/3654

[HOA] hueniverse.com/oauth/

[JL] Nitin Jindal, Bing Liu, "Opinion Spam and Analysis", 2008.

[L] Bing Liu, "Sentiment Analysis and Opinion Mining", 22 April 2012.

[GSCDMEHYFS] Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, Noah A. Smith, "Part-of-Speech Tagging for Twitter: Annotation, Features and Experiments", 2011.

[RWE] Ian Robinson, Jim Webber, Emil Eifrem, "Graph Databases", O'Reilly Media, June 2013.

[PVW] Jonas Partner, Aleksa Vukotic, Nicki Watt, "Neo4j in Action", 28 October 2014, Early Access : June 2012.

[SKS] Abraham Silberschatz, Henry F. Korth, S. Sudarshan, "Database System Concepts

- 6th Edition", McGraw-Hill January 28 2010.

[G] Antonin Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching", 1984.

[MNPT] Yiannis Manolopoulos, Alexandros Nanopoulos, Apostolos N. Papadopoulos, Yiannis Theodoridis, "R-Trees: Theory and Applications", Springer 2006.

[WRT] wiki/R-tree

[SGG] Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, "Operating System Concepts – 7th Edition", Wiley, 2005.

[NGAP] blog.webfaction.com/2008/12/a-little-holiday-present-10000-reqssec-with-nginx-2

[BLC] blog.caustik.com/2012/08/19/node-js-w1m-concurrent-connections

[PT] Pedro Teixeira, "Professional Node.js: Building JavaScript Based Scalable Software", Wiley, October 23 2012.

[CHHR] Mike Cantelon, Marc Harter, T. J. Holowaychuk, Nathan Rajlich, "Node.js in Action", Manning, October 2013.

[GNN] github.com/thingdom/node-neo4j

[DLA] Alessio Di Lorenzo, Giovanni Allegri, "Instant OpenLayers Starter", Packt Publishing, April 2013.

[HML] github.com/hoehrmann/openlayers-heatmap

[GH] graphaware.com/neo4j/2013/10/11/neo4j-bidirectional-relationships.html