Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**ΥΠΗΡΕΣΙΑ ΠΡΟΤΑΣΕΩΝ ΓΙΑ ΤΕΚΜΗΡΙΑ ΜΕ ΕΠΙΚΑΙΡΟ ΘΕΜΑ ΣΕ ΜΙΑ ΒΙΒΛΙΟΘΗΚΗ**
Ονοματεπώνυμο Φοιτητή	**Απόστολος Καραλής**
Πατρώνυμο	**Άγγελος**
Αριθμός Μητρώου	**ΜΠΣΠ13043**
Επιβλέπων	**Χρήστος Δουληγέρης, Καθηγητής**
Συνεπιβλέπων	**Ιωάννης Παπαδάκης, Επίκουρος Καθηγητής**

Ημερομηνία Παράδοσης **Νοέμβριος 2015**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Χρήστος Δουληγέρης Καθηγητής	Δημήτριος Βέργαδος Επίκουρος Καθηγητής	Ιωάννης Παπαδάκης Επίκουρος Καθηγητής

Περιεχόμενα

ΥΠΗΡΕΣΙΑ ΠΡΟΤΑΣΕΩΝ ΓΙΑ ΤΕΚΜΗΡΙΑ ΜΕ ΕΠΙΚΑΙΡΟ ΘΕΜΑ ΣΕ ΜΙΑ ΒΙΒΛΙΟΘΗΚΗ

Περίληψη

Μία από τις βασικές ευθύνες των βιβλιοθηκών είναι η παροχή επίκαιρων πληροφοριών στους χρήστες τους. Παραδοσιακά, αυτό επιτυγχάνεται μέσω της αξιοποίησης στατιστικών μεθόδων, όπως η καταμέτρηση των τεκμηρίων που διανέμονται στους χρήστες και η ανάλυση του αρχείου καταγραφής των αναζητήσεων των χρηστών στον ηλεκτρονικό κατάλογο της βιβλιοθήκης και σε άλλες ηλεκτρονικές υπηρεσίες της. Ωστόσο, αυτές οι μέθοδοι εστιάζουν στην ανάλυση της αλληλεπίδρασης με τους χρήστες σε ένα ορισμένο χρονικό διάστημα παρέχοντας ως εκ τούτου γνώση για το παρελθόν. Στον αντίποδα, τα μέσα κοινωνικής δικτύωσης φαίνονται ικανά να παράσχουν επίκαιρες πληροφορίες με έναν πιο άμεσο τρόπο. Στην κατεύθυνση αυτή, η παρούσα διατριβή επικεντρώνεται στην ανάπτυξη μιας πειραματικής δικτυακής υπηρεσίας ικανής να ενσωματώσει τα Twitter hashtags στον ηλεκτρονικό κατάλογο μίας βιβλιοθήκης.

Λέξεις κλειδιά: Βιβλιοθήκη, Twitter, hashtags, μέσα κοινωνικής δικτύωσης

Abstract

One of the basic responsibilities of libraries to their users is the provision of timely information. Traditionally, such a goal is achieved through the employment of statistical methods such as the measurement of the items' circulation and through the log file analysis capturing the searches users perform against the OPAC and other online library services. However, such methods focus on the analysis of user transactions over a certain period of time and therefore provide insight into the past. On the other hand, social media seem capable of providing timely information in a more direct way. In this paper, an experimental web service is deployed capable of integrating popular Twitter hashtags with the library's OPAC.

Keywords: Library, Twitter, hashtags, social media

1. Εισαγωγή

Μία από τις βασικές ευθύνες των βιβλιοθηκών είναι η παροχή επίκαιρων η/και δημοφιλών πληροφοριών στους χρήστες τους. Παραδοσιακά, αυτό επιτυγχάνεται μέσω της αξιοποίησης στατιστικών μεθόδων, όπως η καταμέτρηση των τεκμηρίων που διανέμονται στους χρήστες [25, 26] και η ανάλυση του αρχείου καταγραφής των αναζητήσεων των χρηστών στον ηλεκτρονικό κατάλογο της βιβλιοθήκης (Online Public Access Catalog - OPAC) και σε άλλες ηλεκτρονικές υπηρεσίες της [27].

Αυτές οι μέθοδοι εστιάζουν στην ανάλυση της αλληλεπίδρασης με τους χρήστες σε ένα ορισμένο χρονικό διάστημα παρέχοντας ως εκ τούτου γνώση για το παρελθόν. Αυτό συνεπάγεται ότι η διαδικασία λήψης αποφάσεων της βιβλιοθήκης αγνοεί τις τρέχουσες τάσεις καθώς αυτές δεν έχουν ακόμα αντικατοπτριστεί στις αναζητήσεις των χρηστών. Επιπλέον, καθίσταται δύσκολη η αναγνώριση της περιστασιακής δημοτικότητας ορισμένων τεκμηρίων που σχετίζονται με θέματα που έχουν μεγαλύτερη ζήτηση σε συγκεκριμένες χρονικές περιόδους [28]. Θα πρέπει ακόμα να ληφθεί υπόψη το γεγονός ότι με την επικράτηση του Web, οι χρήστες των μηχανών αναζήτησης τείνουν να επηρεάζονται από την απλότητα της διαδικασίας αναζήτησης και ανάκτησης [29], που τους παρέχει σχετικές και ενημερωμένες πληροφορίες σε εύθετο χρόνο. Συνεπακόλουθα, οι χρήστες περιμένουν από τις βιβλιοθήκες να λειτουργούν με ένα παρόμοιο τρόπο και να λαμβάνουν έγκαιρα τα αποτελέσματα που σχετίζονται με τις ανάγκες και τις απαιτήσεις τους [30].

Σε μια προσπάθεια να ξεπεραστούν τα παραπάνω θέματα, οι βιβλιοθήκες πρέπει να αναπτύξουν νέες μεθόδους για την αντιμετώπιση των ζητημάτων της δημοτικότητας και της επικαιρότητας των πληροφοριών. Στην κατεύθυνση αυτή, τα μέσα κοινωνικής δικτύωσης δίνουν τη δυνατότητα στις βιβλιοθήκες να εστιάσουν στους χρήστες τους και τις απαιτήσεις τους [31] με έναν πιο άμεσο τρόπο [32, 33, 34, 35].

Κινούμενοι προς αυτήν την κατεύθυνση, στην παρούσα διατριβή παρουσιάζουμε την ανάπτυξη μίας πειραματικής δικτυακής υπηρεσίας ικανής να αναγνωρίσει τα επίκαιρα ή/και δημοφιλή θέματα μιας βιβλιοθήκης μέσα από τα hashtags του Twitter. Κατόπιν, το σύνολο των επίκαιρων και δημοφιλών θεμάτων επεκτείνεται με τη βοήθεια κατάλληλων τεχνολογιών του Σημασιολογικού Ιστού. Για κάθε ένα από αυτά τα θέματα αναζητούνται σχετικά βιβλία στον ηλεκτρονικό κατάλογο μίας βιβλιοθήκης, τα οποία εν συνεχεία προτείνονται στους χρήστες. Με τον τρόπο αυτό, παρουσιάζονται στους χρήστες, σε εύθετο χρόνο, βιβλία τα οποία είναι σχετικά με επίκαιρα ή/και δημοφιλή θέματα.

Το υπόλοιπο της διατριβής περιλαμβάνει 6 ενότητες, ξεκινώντας από την ενότητα 3. Καθεμία από τις ενότητες 3 έως 6 παρουσιάζει τις υπηρεσίες τρίτων (third-party services) που αξιοποιούνται από την εφαρμογή μας. Οι ενότητες 7 και 8 περιέχουν κατά αντιστοιχία την περιγραφή και την αξιολόγηση της εφαρμογής μας. Στην ενότητα 8 παρουσιάζονται τα γενικά συμπεράσματα που προέκυψαν από την ανάπτυξη και εκτέλεση της εφαρμογής μας. Τέλος, στο παράρτημα Α καταγράφεται ο κώδικας της εφαρμογής.

2. Προγραμματιστική διεπαφή αναζήτησης Twitter

Το πολύ γνωστό μέσο κοινωνικής δικτύωσης Twitter [1], προσφέρει ένα σύνολο από διεπαφές προγραμματισμού εφαρμογών (Application Programming Interfaces – APIs), μέσω των οποίων δίδεται η δυνατότητα σε μια εφαρμογή να αποκτήσει πρόσβαση στα ιδιωτικά δεδομένα ενός χρήστη, όπως για παράδειγμα στις ιδιωτικές του συνομιλίες, ή να αντλήσει πληροφορίες από τα δεδομένα που είναι δημοσίως διαθέσιμα στο Twitter, με χαρακτηριστικό παράδειγμα την εύρεση των πιο δημοφιλών θεμάτων στο Twitter.

Μία από αυτές τις διεπαφές είναι η Search [2, 3] που αποτελεί μέρος της έκδοσης 1.1 των RESTful APIs του Twitter. Η εν λόγω διεπαφή επιτρέπει τη διενέργεια αναζητήσεων για την εύρεση των πιο πρόσφατων ή δημοφιλών tweets που πληρούν τα κριτήρια της εκάστοτε αναζήτησης. Ο όρος tweets αναφέρεται στα μηνύματα που δημοσιεύονται στο Twitter.

2.1. Αυθεντικοποίηση εφαρμογής-πελάτη

Η αυθεντικοποίηση των εφαρμογών-πελατών (δηλαδή η επιβεβαίωση της ταυτότητας τους) αποτελεί προαπαιτούμενο για τη χρήση των APIs του Twitter. Στην έκδοση 1.1 των APIs του Twitter προσφέρονται δύο τρόποι αυθεντικοποίησης:

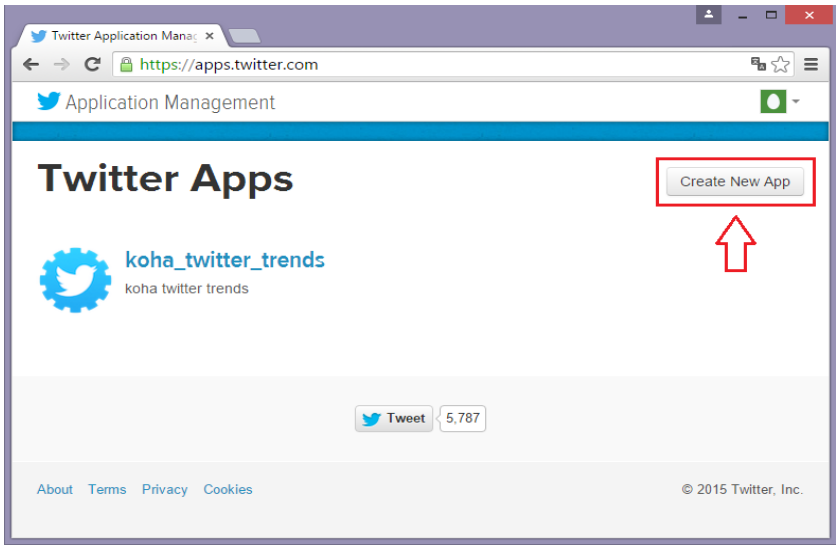
- Αυθεντικοποίηση χρήστη εφαρμογής (Application-user authentication): βασίζεται στο πρωτόκολλο OAuth 1.0A [4] και επιτρέπει σε μια εφαρμογή να ενεργεί για λογαριασμό ενός χρήστη του Twitter μετά την παραχώρηση της απαιτούμενης άδειας από αυτόν. Η χρησιμοποίηση του συγκεκριμένου τρόπου αυθεντικοποίησης είναι απαραίτητη στις περιπτώσεις χρήσης των APIs όπου είτε επιχειρείται η τροποποίηση στοιχείων στο λογαριασμό ενός χρήστη είτε ζητείται πρόσβαση σε δεδομένα ενός χρήστη που δεν είναι δημοσίως διαθέσιμα.
- Αυθεντικοποίηση της ιδίας εφαρμογής (Application-only authentication): αξιοποιεί το πρωτόκολλο OAuth 2 [5] και ειδικότερα την τεχνική «Client Credentials Grant (Έκδοση Διαπιστευτηρίων Πελάτη)» αυτού. Η προκείμενη μέθοδος δίδει τη δυνατότητα σε μια εφαρμογή να ενεργήσει για τον εαυτό της, παρέχοντάς της κατά αυτόν τον τρόπο πρόσβαση μόνο στα δεδομένα του Twitter που είναι διαθέσιμα στο ευρύ κοινό.

Το RESTFul Search API, καθότι προσφέρει τη δυνατότητα αναζητήσεων tweets τα οποία είναι διαθέσιμα στο ευρύ κοινό, δεν απαιτεί το περιβάλλον ενός αυθεντικοποιημένου χρήστη. Ως εκ τούτου, για τη χρήση του συγκεκριμένου API, αρκεί η αυθεντικοποίηση της ιδίας εφαρμογής (Application-only authentication), η οποία είναι σαφώς απλούστερη από την αυθεντικοποίηση χρήστη εφαρμογής (Application-user authentication) και προσφέρει υψηλότερο όριο χρήσης.

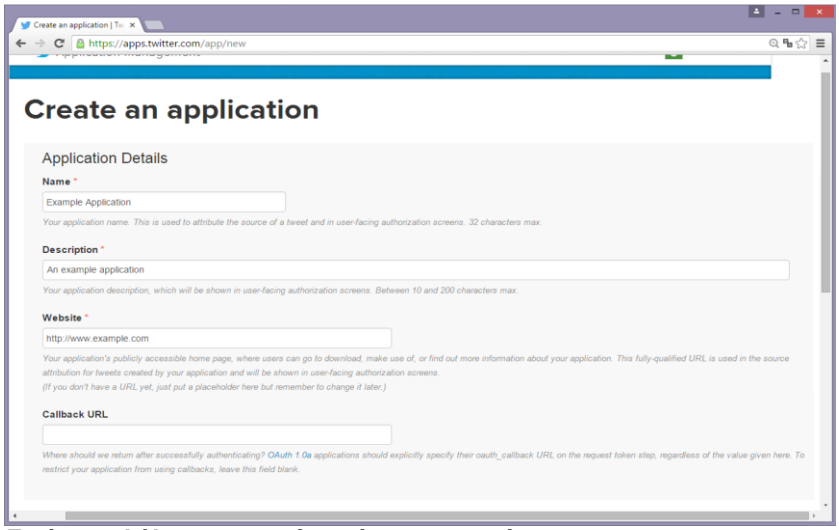
2.1.1. Δήλωση εφαρμογής

Η αυθεντικοποίηση απαιτεί εξ ορισμού την κατάθεση διαπιστευτηρίων από την οντότητα που επιδιώκει να αυθεντικοποιηθεί. Στο Twitter για την απόκτηση των διαπιστευτηρίων μιας εφαρμογής απαιτείται η δήλωσή της, μέσω ενός ενεργού και συνδεδεμένου λογαριασμού, στην ιστοσελίδα https://apps.Twitter.com/app. Αξίζει να σημειωθεί ότι σε αυτήν την ιστοσελίδα, πέραν της δυνατότητας δήλωσης νέων εφαρμογών, παρουσιάζονται όλες οι εφαρμογές που έχουν μέχρι στιγμής δηλωθεί από τον τρέχοντα κάθε φορά λογαριασμό Twitter.

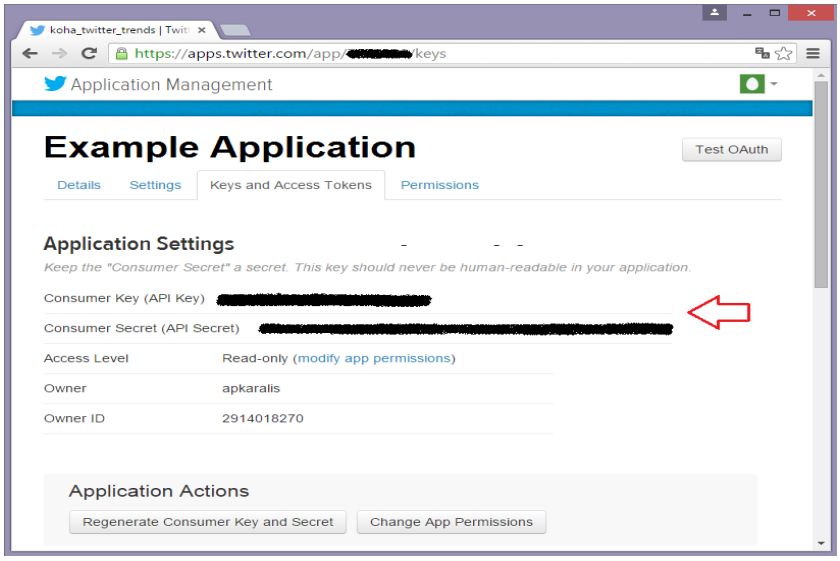
Στη συνέχεια παρουσιάζεται ένα παράδειγμα δήλωσης μιας εφαρμογής, καθώς επίσης τα διαπιστευτήρια αυτής.



Εικόνα 1. Άνοιγμα φόρμας δήλωσης νέας εφαρμογής



Εικόνα 2. Δήλωση στοιχείων νέας εφαρμογής



Εικόνα 3. Διαπιστευτήρια εφαρμογής

2.1.2. Αυθεντικοποίηση της ιδίας εφαρμογής

Η επαλήθευση της ταυτότητας μιας εφαρμογής πραγματοποιείται με την προσκόμιση ενός ειδικού διακριτικού πρόσβασης (access Token) από αυτήν. Η διαδικασία που πρέπει να ακολουθηθεί για την απόκτηση του εν λόγω διακριτικού αποτελείται από τις δύο ακόλουθες φάσεις:

1. Κωδικοποίηση κλειδιού (consumer key) και μυστικού πελάτη (consumer secret)
2. Αίτημα λήψης διακριτικού πρόσβασης

Στην πρώτη φάση λαμβάνει χώρα ο ακόλουθος αλγόριθμος:

1. Το κλειδί πελάτη (consumer key) και το μυστικό πελάτη (secret) που έχουν δοθεί στην εφαρμογή, κωδικοποιούνται σύμφωνα με το πρότυπο των URL (Uniform Resource LocaTors), όπως αυτό περιγράφεται στο RFC1738[6]. Σημειώνεται ότι, τη στιγμή της συγγραφής του παρόντος κειμένου, η μορφή του κλειδιού και του μυστικού του πελάτη είναι τέτοια που η προαναφερόμενη κωδικοποίηση τους δεν τα αλλάζει. Ωστόσο, αυτό το βήμα θα πρέπει να εκτελείται ενόψει πιθανής αλλαγής στη μορφή αυτών των τιμών.
2. Δημιουργείται μία συμβολοσειρά αποτελούμενη κατά σειρά από το κωδικοποιημένο κλειδί πελάτη, τον χαρακτήρα ":" και το κωδικοποιημένο μυστικό πελάτη.
3. Η συμβολοσειρά που προέκυψε από το προηγούμενο βήμα κωδικοποιείται βάσει του σχήματος base64[37].

Πίνακας 1. Παράδειγμα εκτέλεσης αλγορίθμου κωδικοποίησης consumer key και consumer secret

Consumer key	xvz1evFS4wEEPTGEFPHBog	
Consumer secret	L8qq9PZyRg6ieKGEKhZolGC0vJWLw8iEJ88DRdyOg	
Βήμα	Αποτέλεσμα	
1ο	κωδικοποιημένο consumer key	xvz1evFS4wEEPTGEFPHBog
	κωδικοποιημένο consumer secret	L8qq9PZyRg6ieKGEKhZolGC0vJWLw8iEJ88DRdyOg
2ο	xvz1evFS4wEEPTGEFPHBog:L8qq9PZyRg6ieKGEKhZolGC0vJWLw8iEJ88DRdyOg	
3ο	eHZ6MWV2RlM0d0VFUFRHRUZQSEJvZzpMOHFxOVBaeVJnNmllS0dFS2hab2xHQzB2SldMdzhpRUo4OERSZHlPZw==	

Στη δεύτερη φάση, αποστέλλεται μία αίτηση HTTP POST στη διεύθυνση URL https://api.twitter.com/oauth2/Token, η οποία:

- Περιλαμβάνει υποχρεωτικά την επικεφαλίδα Authorization με τιμή *Basic <αποτέλεσμα βήματος 3 αλγορίθμου πρώτης φάσης>.*
- Περιέχει απαραιτήτως την επικεφαλίδα Content-Type με τιμή *application/x-www-form-urlencoded;charset=UTF-8*.
- Το σώμα (body) της αίτησης πρέπει να είναι *grant_type=client_credentials.*

Η απάντηση HTTP που λαμβάνεται, στην περίπτωση που η αίτηση είχε μορφοποιηθεί σωστά, περιέχει ως σώμα ένα JSON αντικείμενο αποτελούμενο από τα εξής πεδία:

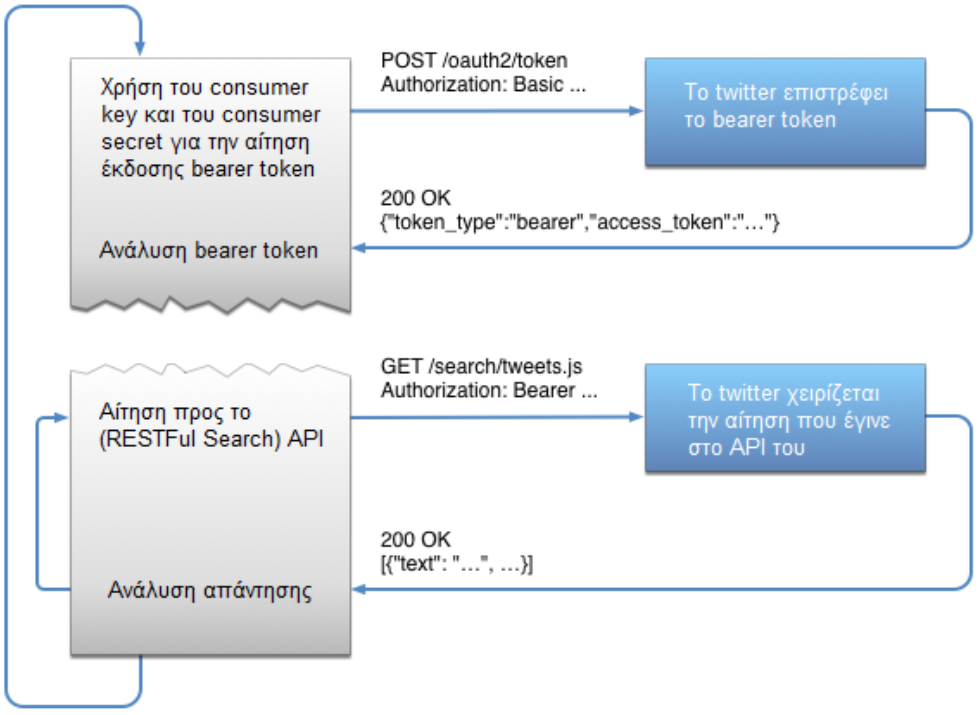
- **token_type**: ο τύπος του διακριτικού πρόσβασης ο οποίος θα πρέπει να έχει την τιμή bearer.
- **access_Token**: το διακριτικό πρόσβασης, το οποίο καθώς είναι τύπου bearer ονομάζεται bearer Token.

```
POST /oauth2/token HTTP/1.1
Host: api.twitter.com
User-Agent: My Twitter App v1.0.23
Authorization: Basic eHZ6MWV2RlM0d0VFUFRHRUZQSEJvZzpMOHFxOVBaeVJnNmllS0dFS2hab2xHQzB2SldMdzhpRUo4OERSZHlPZw==Content-Type: application/x-www-form-urlencoded;charset=UTF-8
Content-Length: 29
Accept-Encoding: gzip

grant_type=client_credentials
```

Εικόνα 4. Παράδειγμα αίτησης HTTP για την έκδοση bearer token

```
HTTP/1.1 200 OK
Status: 200 OK
Content-Type: application/json; charset=utf-8
...
Content-Encoding: gzip
Content-Length: 140

{"token_type":"bearer","access_token":"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA%2FAAAAAAAAAAAAAAAAAAA%3DAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"}
```

Εικόνα 5. Παράδειγμα φυσιολογικής απάντησης HTTP σε ένα αίτημα έκδοσης bearer token



Εικόνα 6. Σχεδιάγραμμα χρήσης Application-only authentication στην κλήση ενός API

Όταν μια εφαρμογή καλεί ένα API, το οποίο υποστηρίζει τον παραπάνω τρόπο αυθεντικοποίησης, μπορεί να χρησιμοποιήσει το bearer Token για να αποδείξει την ταυτότητα της. Τούτο επιτυγχάνεται με την εισαγωγή της επικεφαλίδας Authorization με τιμή Bearer <bearer_Token> στην αίτηση HTTP. Στην εικόνα 6 παρουσιάζεται η διαδικασία που λαμβάνει χώρα για τη χρήση του προαναφερόμενου τρόπου αυθεντικοποίησης κατά την κλήση του RESTFul Search API. του Twitter. Αντίστοιχη διαδικασία ακολουθείται για τη χρήση του συνόλου των API του Twitter.

Κρίνεται σκόπιμο να σημειωθεί ότι ένα bearer Token έχει περιορισμένη διάρκεια ζωής, κατά τη διάρκεια της οποίας οι νέες αιτήσεις για την έκδοση bearer Token επιστρέφουν το ισχύον bearer Token. Παράλληλα, το Twitter παρέχει τη δυνατότητα τόσο ανάγνωσης του υπολειπόμενου χρόνου ζωής ενός εκδοθέντος bearer Token όσο και της ρητής ακύρωσης αυτού.

2.2. Υποβολή ερωτημάτων

Η αποστολή των ερωτημάτων αναζήτησης στο RESTFul Search API, πραγματοποιείται στη διεύθυνση URL https://api.twitter.com/1.1/search/tweets.json με τη χρήση της μεθόδου GET. Στο παρακάτω πίνακα, περιγράφονται η μόνη υποχρεωτική παράμετρος q και κάποιες ενδεικτικές προαιρετικές παράμετροι για τις αιτήσεις προς το RESTFul Search API.

Πίνακας 2. Ενδεικτικές παράμετροι για τις αιτήσεις προς το RESTFul Search API

Παράμετρος	Περιγραφή
q	Αντιπροσωπεύει το ερώτημα της αναζήτησης το οποίο αφού αρχικώς οριστεί ως μία συμβολοσειρά από UTF8 χαρακτήρες, πρέπει κατόπιν να κωδικοποιηθεί σύμφωνα με το πρότυπο των URL. Το ερώτημα της αναζήτησης δεν πρέπει να ξεπερνά τους 500 χαρακτήρες, συμπεριλαμβανομένων των τελεστών. Παραδείγματα (πριν την κωδικοποίηση): ▪ "watching now" → αναζήτηση των tweets που περιέχουν και τις δύο λέξεις, watching και now. ▪ "happy our" → αναζήτηση των tweets που περιλαμβάνουν τη φράση happy our.
result_type	Προσδιορίζει τον τύπο των αποτελεσμάτων της αναζήτησης. Η εξ ορισμού τιμή είναι "mixed", ενώ οι δυνατές τιμές είναι οι ακόλουθες: ▪ recent: να γίνει λήψη μόνο των πιο πρόσφατων αποτελεσμάτων. ▪ popular: τα αποτελέσματα να περιλαμβάνουν μόνο τα πιο δημοφιλή. ▪ mixed: να ληφθούν τόσο τα πιο δημοφιλή όσο και τα πιο πρόσφατα αποτελέσματα.
count	Ορίζει το πλήθος των αποτελεσμάτων που θα ληφθούν. Η μέγιστη δυνατή τιμή είναι 100, ενώ η εξ ορισμού 15. Παράδειγμα: 80
since_id	Δηλώνει ρητώς πως τα λαμβανόμενα αποτελέσματα πρέπει να έχουν αναγνωριστικό (ID) μεγαλύτερο από αυτό που έχει δοθεί ως τιμή σε αυτήν την παράμετρο. Όσο μεγαλύτερο είναι το αναγνωριστικό τόσο πιο πρόσφατο είναι το αποτέλεσμα και το αντίθετο. Επειδή υπάρχει όριο στον αριθμό των tweets που είναι προσπελάσιμα μέσω του προκείμενου API, αν το πλήθος των tweets που είναι μεγαλύτερα από το since_id ξεπερνάει αυτό το όριο, τότε ως since_id θέτεται το παλαιότερο δυνατό, ώστε να μην ξεπερνιέται αυτό το όριο. Παράδειγμα: 12345
max_id	Υπαγορεύει ότι τα αποτελέσματα απαιτείται να διαθέτουν αναγνωριστικό (ID) μικρότερο ή ίσο από το οριζόμενο σε αυτήν την παράμετρο. Παράδειγμα: 54321

```
GET
https://api.twitter.com/1.1/search/tweets.json?q=%23freebandnames&since_id=2401269984051000&
max_id=250126199840518145&result_type=mixed&count=4
Authorization: Bearer
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA%2FAAAAAAAAAAAAAAAAAAAA%3DAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

Εικόνα 7. Παράδειγμα αίτησης προς το RESTFul Search API

2.3. Λήψη αποτελεσμάτων

Τα αποτελέσματα των ερωτημάτων λαμβάνονται σε JSON[38] μορφή. Ειδικότερα, για κάθε ερώτημα επιστρέφεται ένα αντικείμενο JSON που περιλαμβάνει τη λίστα των αποτελεσμάτων, τα οποία καλούνται statuses και αντιπροσωπεύουν τα tweets που πληρούν τα κριτήρια της αναζήτησης.

Εικόνα 8. Παράδειγμα απάντησης JSON (σε δενδρική αναπαράσταση) από το RESTful Search API

2.4. Όρια χρήσης

Ο αριθμός των αιτήσεων που δύνανται να σταλθούν στο RESTFul Search API είναι περιορισμένος και εξαρτάται από τον τρόπο αυθεντικοποίησης. Τα όρια χρήσης τα οποία ορίζονται για χρονικά παράθυρα των 15 λεπτών παρουσιάζονται παρακάτω.

Πίνακας 3. Όριο αιτήσεων για το RESTful Search API ανάλογα με τον τρόπο αυθεντικοποίησης

Τρόπος Αυθεντικοποίησης	Όριο αιτήσεων / χρονικό παράθυρο 15 λεπτών
Application-user authentication	180
Application-only authentication	450

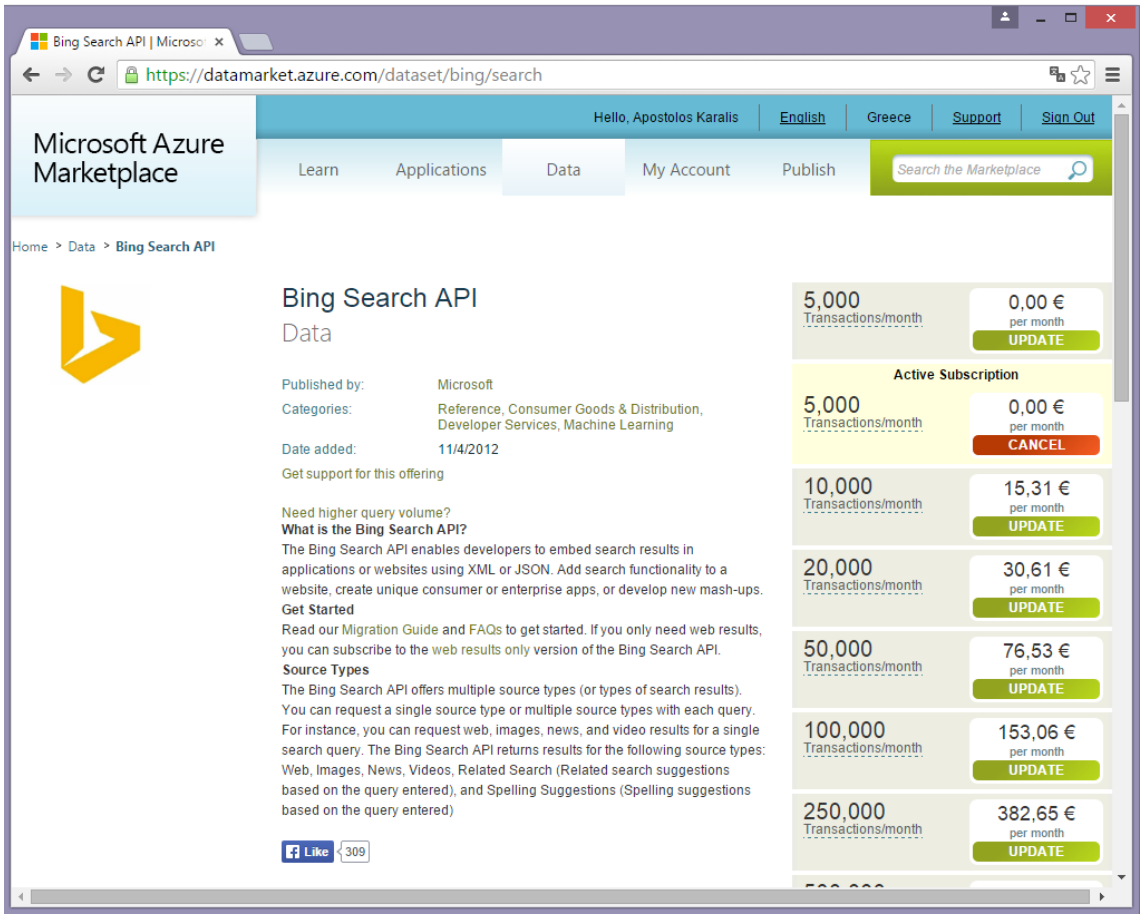
3. Ορθογραφικές προτάσεις Bing

Η εταιρεία Microsoft παρέχει μέσω του υπολογιστικού νέφους (cloud) Microsoft Azure, ένα API για την πρόσβαση στις πληροφορίες που συλλέγονται από τη μηχανή αναζήτησης της Bing. Το συγκεκριμένο API ονομάζεται Bing Search API Marketplace [8, 9, 10] και υπακούει στο πρωτόκολλο ODATA [11], δηλαδή σε μία τυποποιημένη μορφή της RESTFul αρχιτεκτονικής.

Ανάμεσα στις δυνατότητες του Bing Search API περιλαμβάνεται η παροχή ορθογραφικών προτάσεων (Spelling Suggestions), τις οποίες μπορεί να αξιοποιήσει μια εφαρμογή και ως προτάσεις τύπου "μήπως εννοούσατε (Did you Mean)".

3.1. Αυθεντικοποίηση

Κλήσεις στο Bing Search API επιτρέπονται μόνο σε ενεργούς συνδρομητές. Για να γίνει κάποιος συνδρομητής, θα πρέπει να μεταβεί στην ιστοσελίδα https://datamarket.azure.com/dataset/bing/search και αφού αυθεντικοποιηθεί, να επιλέξει το οικονομικό πακέτο που τον ενδιαφέρει.



Εικόνα 9. Ιστοσελίδα εγγραφής συνδρομητή στο Bing Search API

Όπως είναι φυσικό, η πρόσβαση των συνδρομητών στο Bing Search API είναι εφικτή μόνο μετά από την επιβεβαίωση της ταυτότητας τους. Για αυτό τον σκοπό, χρησιμοποιείται ο βασικός τρόπος αυθεντικοποίησης (Basic Authentication Scheme) που έχει οριστεί για τις αιτήσεις HTTP στο RFC2617[36]. Αναλυτικότερα, σε κάθε HTTP κλήση προς το API, εισάγεται η επικεφαλίδα Authorization με τιμή Basic <basic_credentials>.

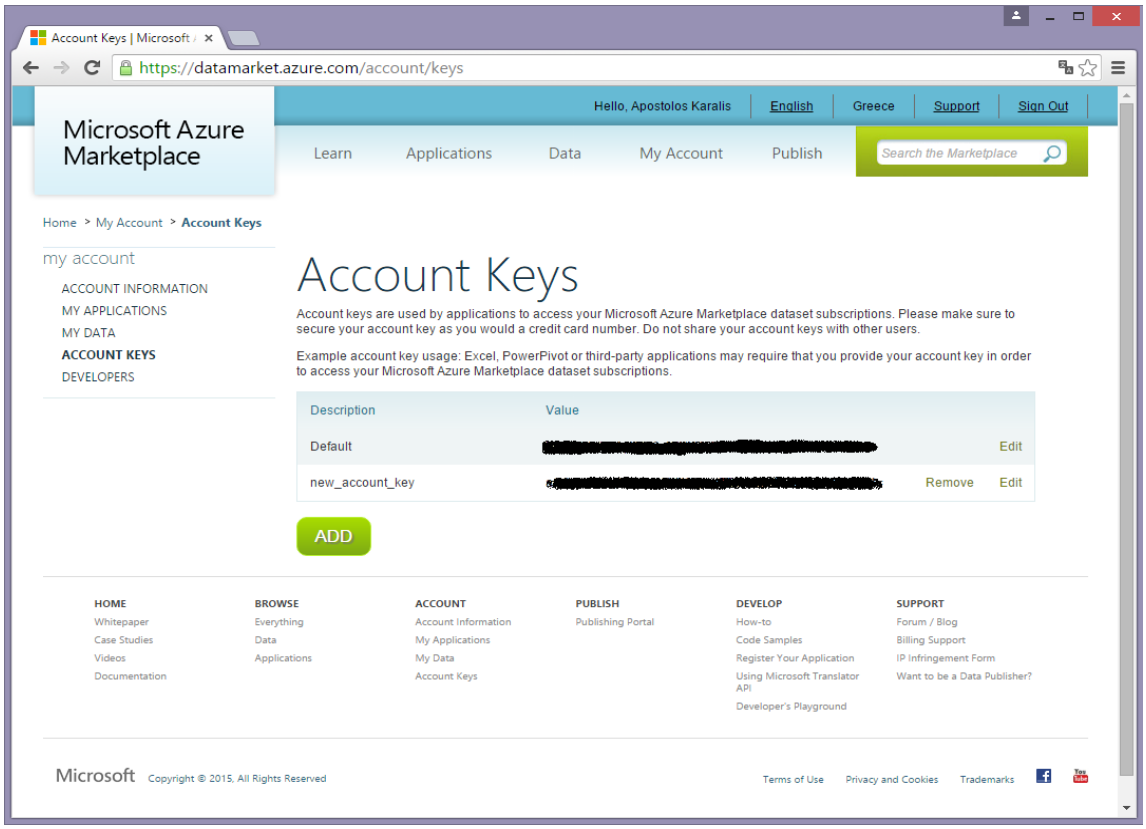
Στην περίπτωση του Bing Search API, τα basic_credentials παράγονται με τη βοήθεια ενός από τα κλειδιά λογαριασμού (account keys) που διαθέτει ο χρήστης, σύμφωνα με τον ακόλουθο αλγόριθμο:

1. Δημιουργείται μία συμβολοσειρά αποτελούμενη κατά σειρά από το account key, τον χαρακτήρα ":" και ξανά το account key.
2. Η συμβολοσειρά του προηγούμενου βήματος κωδικοποιείται βάσει του σχήματος base64. Το αποτέλεσμα που προκύπτει αποτελεί τα basic_credentials.

Πίνακας 4. Παράδειγμα εκτέλεσης αλγορίθμου παραγωγής basic_credentials για το Bing Search API

Account key	aO6yhzOmgQ5MhWSiH8ZpodFb7essNiJR4Kk3PCuFr4=
Βήμα	Αποτέλεσμα
1ο	aO6yhzOmgQ5MhWSiH8ZpodFb7essNiJR4Kk3PCuFr4=:aO6yhzOmgQ5MhWSiH8ZpodFb7essNiJR4Kk3PCuFr4=
2ο	YU82eWh6T21nUTVNaFdTaUg4WnBvZEZiN2Vzc05pSlI0S2szUEN1RnI0PQ==

Είναι αξιοπρόσεκτο ότι κάθε χρήστης δύναται να διαθέτει περισσότερα του ενός account keys. Σε κάθε χρήστη υπάρχει ένα εξ ορισμού (default) account key το οποίο δημιουργείται κατά την κατασκευή του λογαριασμού του χρήστη και συνδέεται μόνιμα με αυτόν. Για λόγους ασφαλείας, προτείνεται στους χρήστες να μη χρησιμοποιούν το default account key. Η δημιουργία ενός νέου account key ή διαγραφή ενός υπάρχοντος (πλην του default) επιτελείται μέσω της υποενότητας ACCOUNT KEYS του Microsoft Azure Marketplace, που φαίνεται στην εικόνα 10.



Εικόνα 10. Υποενότητα «Account Keys» για τους χρήστες του Microsoft Azure Marketplace

3.2. Υποβολή ερωτημάτων

Η υπηρεσία των ορθογραφικών προτάσεων του Bing Search API παρέχεται στη διεύθυνση URL https://api.datamarket.azure.com/Bing/Search/SpellingSuggestion μέσω της μεθόδου GET. Κάθε αίτηση πρέπει να συνοδεύεται από την παράμετρο Query, η οποία συνιστά και τη μόνη υποχρεωτική παράμετρο. Στον πίνακα 5 περιγράφεται η συγκεκριμένη παράμετρος, καθώς επίσης τρεις παράμετροι που πηγάζουν από το πρωτόκολλο ODATA.

Πίνακας 5. Ενδεικτικές παράμετροι της υπηρεσίας Spelling Suggestions του Bing Search API

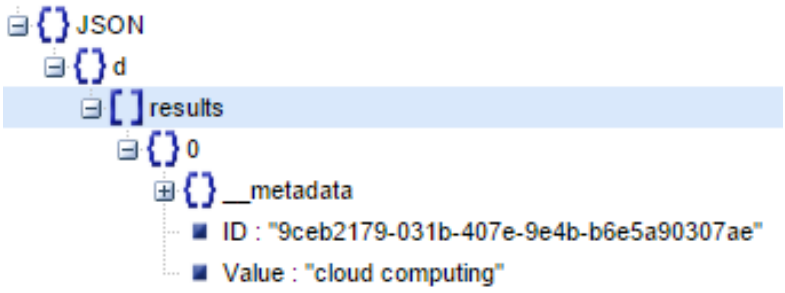
Παράμετροι	Περιγραφή
Query	Εμπεριέχει το ερώτημα που αποτελείται από μία ή περισσότερες λέξεις για τις οποίες ζητείται η παροχή ορθογραφικών προτάσεων. Το ερώτημα πρέπει να περικλείεται από μονά εισαγωγικά ('). Τόσα τα μονά εισαγωγικά όσο και οι μη λατινικοί χαρακτήρες θα πρέπει να κωδικοποιηθούν σύμφωνα με το πρότυπο των URL. Παράδειγμα (πριν την κωδικοποίηση): 'clud computing'
$Top	Καθορίζει τον μέγιστο αριθμό των αποτελεσμάτων που θα επιστραφούν. Παράδειγμα: 2
$skip	Αντιπροσωπεύει τον αριθμό των πρώτων κατά σειρά αποτελεσμάτων που θα παρακαμφθούν, δηλαδή δεν θα συμπεριληφθούν στην ODATA απάντηση. Εξ ορισμού, η παράμετρος αυτή έχει τιμή 0. Παράδειγμα: 5
$format	Ορίζει τη μορφή της ODATA απάντησης. Οι υποστηριζόμενες μορφές είναι η JSON και η ATom (για την XML) που είναι η εξ ορισμού τιμή.

3.3. Λήψη αποτελεσμάτων

Κάθε ένα από τα αποτελέσματα που λαμβάνονται, περιέχει τα εξής κύρια στοιχεία:

- ID: ένα αναγνωριστικό που προσδιορίζει μοναδικά το αποτέλεσμα ανάμεσα στα υπόλοιπα
- Value: η ορθογραφική πρόταση

Στην ακόλουθη εικόνα φαίνονται τα αποτελέσματα που λήφθηκαν για την αίτηση της εικόνας 11. Τα αποτελέσματα, όπως άλλωστε ζητήθηκε, αποτυπώθηκαν με μορφή JSON. Ειδικότερα, εμπεριέχονται εντός του αντικειμένου d το οποίο αντιπροσωπεύει το λεγόμενο έγγραφο υπηρεσίας (Service Document) στην ορολογία του πρωτοκόλλου ODATA.



Εικόνα 11. Απάντηση JSON (σε δενδρική αναπαράσταση) στην αίτηση της εικόνας 11

3.4. Όρια χρήσης

Όπως προαναφέραμε, η πρόσβαση στο Bing Search API είναι εφικτή μόνο για τους ενεργούς συνδρομητές. Προσφέρεται πλήθος οικονομικών πακέτων, όπως φαίνεται και στην εικόνα 9, ανάλογα με τον μέγιστο αριθμό μηνιαίων συνδιαλλαγών (transactions) με το Bing Search API που επιθυμεί να πραγματοποιήσει ο χρήστης. Μεταξύ αυτών των πακέτων, υπάρχει ένα με μηδενική οικονομική συνδρομή το οποίο επιτρέπει την εκτέλεση 5000 transactions ανά μήνα. Επισημαίνεται, για την αποφυγή σύγχυσης, ότι κάθε ερώτημα που υποβάλλεται στο Bing Search API συνιστά μια δοσοληψία (transaction).

4. Αναζήτηση σχετικών όρων και κατηγοριών στην DBPedia

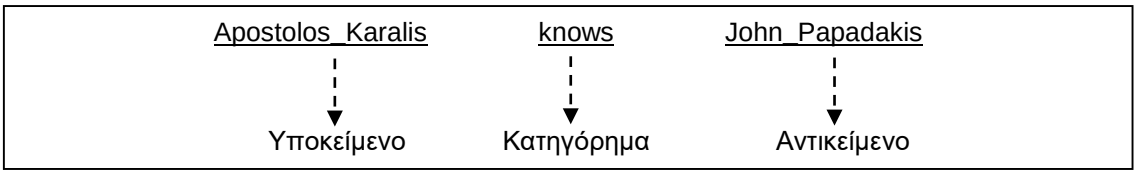
Η DBPedia αποτελεί μία πολυπληθή κοινότητα που προσπαθεί να εξάγει δομημένες πληροφορίες (structured information) από τη Wikipedia και να παρέχει αυτές τις πληροφορίες μέσω του Ιστού (Web) [12]. Επιπρόσθετα, η DBPedia προσφέρει ένα σημείο πρόσβασης για την εκτέλεση ερωτημάτων SPARQL επί αυτών των πληροφοριών. Δύο παραδείγματα τέτοιων ερωτημάτων αποτελούν η αναζήτηση των όρων και των κατηγοριών που σχετίζονται με μία λέξη ή φράση.

Για να είναι σε θέση κάποιος να διατυπώσει και να κατανοήσει ένα SPARQL ερώτημα, θα πρέπει προηγουμένως να έχει καταλάβει τις βασικές έννοιες του Σημασιολογικού Ιστού (Semantic Web), στις οποίες θα αναφερθούμε εν συντομία στη συνέχεια.

4.1. Βασικές έννοιες Σημασιολογικού Ιστού

Ο Σημασιολογικός Ιστός [13, 14, 15, 16] αντιμετωπίζεται ως μία προέκταση του Παγκοσμίου Ιστού, όπου το πλήθος των αδόμητων πληροφοριών κωδικοποιούνται από τους παρόχους περιεχομένου (content providers) σε σημασιολογικές δομές δεδομένων.

Το βασικό μοντέλο οργάνωσης των δεδομένων του Σημασιολογικού Ιστού αποτελούν οι τριάδες (triples), μέσω των οποίων συντελείται ο συσχετισμός των δεδομένων. Πρακτικά, σε κάθε τριάδα πραγματοποιείται απόδοση τιμής σε μία συγκεκριμένη ιδιότητα μίας οντότητας. Ως εκ τούτου, η γενική μορφή των τριάδων είναι: (οντότητα, ιδιότητα, τιμή). Ωστόσο, στο σημασιολογικό ιστό, η γενική μορφή των τριάδων αποδίδεται με όρους μαθηματικής λογικής ως (υποκείμενο, κατηγόρημα, αντικείμενο).



Εικόνα 12. Παράδειγμα τριάδας (υποκείμενο, κατηγόρημα, αντικείμενο)

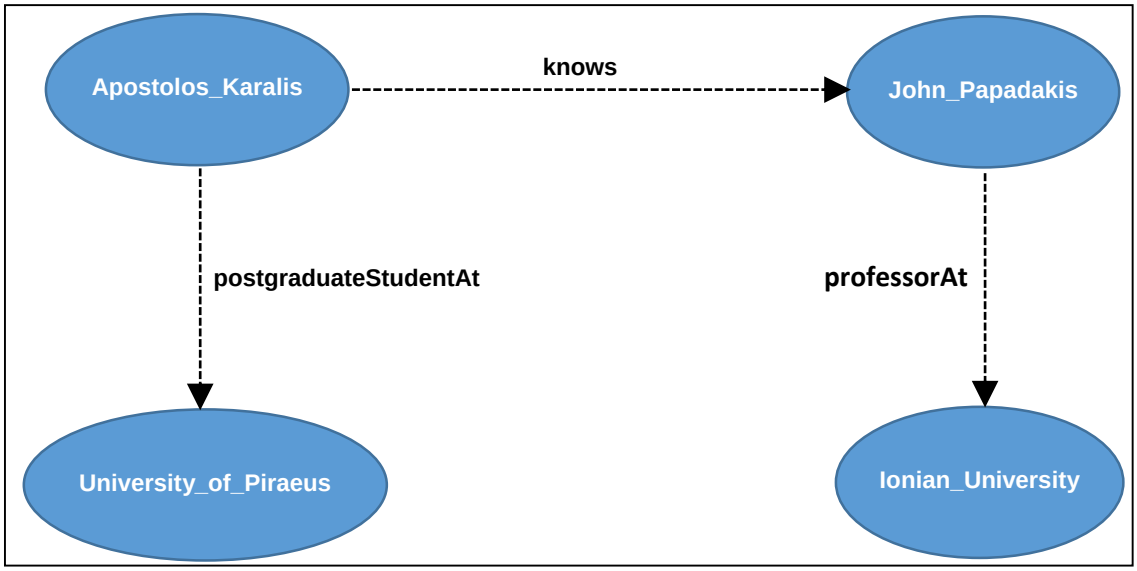
Το υποκείμενο μιας τριάδας αντιπροσωπεύει πάντα μία οντότητα, όπως λόγου χάριν ένα άτομο (Person) ή μια περιοχή (Place). Από την άλλη πλευρά, τα αντικείμενα είναι είτε οντότητες που πιθανώς να συνιστούν υποκείμενα σε άλλες τριάδες είτε λεκτικές τιμές (literal values) όπως συμβολοσειρές (strings) και αριθμοί.

Πολλαπλές τριάδες δύνανται να συνδεθούν μεταξύ τους διαμέσου της χρήσης των ίδιων υποκειμένων ή αντικειμένων, δημιουργώντας μία αλυσίδα από σχέσεις που μπορεί να αναπαρασταθεί από ένα κατευθυνόμενο γράφημα (directed graph). Αν επιπρόσθετα της τριάδας της εικόνας 12, θεωρήσουμε δύο τριάδες (John_Papadakis, professorAt, Ionian_University) και (AposTolos_Karalis, postgraduateStudentAt, University_of_Piraeus) τότε προκύπτει το κατευθυνόμενο γράφημα της εικόνας 13.

Βασική προϋπόθεση για τη δημιουργία συνδέσεων μεταξύ των τριάδων, αποτελεί η χρήση μοναδικών ονομάτων-αναγνωριστικών για τα υποκείμενα και αντικείμενα. Στα τετριμμένα παραδείγματα τριάδων που παρουσιάσαμε πρωτύτερα, θεωρήσαμε ότι τα αναγνωριστικά των υποκειμένων και αντικειμένων είναι μοναδικά. Ωστόσο, στην πραγματικότητα δεν ισχύει κάτι τέτοιο, καθότι λόγου χάριν υπάρχουν περισσότερα τους ενός άτομα με ονοματεπώνυμο AposTolos Karalis.

Παρόμοια, τα κατηγορήματα πρέπει να προσδιορίζονται με μοναδικό τρόπο. Για να γίνει αντιληπτή αυτή αναγκαιότητα, αρκεί να σκεφτούμε την περίπτωση δύο ξεχωριστών δομών τριάδων, όπου η πρώτη ασχολείται με ταινίες και η δεύτερη με αγροτικά προϊόντα, ενώ και οι δυο χρησιμοποιούν το κατηγόρημα producerOf. Στην πρώτη περίπτωση, σε κάθε τριάδα που εμπεριέχεται αυτό το κατηγόρημα, αναμένεται ως υποκείμενο μία ταινία και ως αντικείμενο ο παραγωγός ή κάποιος από τους παραγωγούς αυτής της ταινίας. Αντίθετα, στη δεύτερη περίπτωση, το υποκείμενο θα πρέπει να είναι μία οντότητα που παράγει αγροτικά προϊόντα και το αντικείμενο κάποιο αγροτικό προϊόν (π.χ. τομάτες).

Το πρόβλημα της αμφισημίας στα αναγνωριστικά ονόματα αντιμετωπίζεται αποτελεσματικά από το πρότυπο RDF.



Εικόνα 13. Παράδειγμα κατευθυνόμενου γράφου αναπαράστασης αλυσίδας τριάδων

4.1.1. Resource Data Framework (RDF)

Η RDF (Resource Data Framework) συνιστά μία πρότυπη γλώσσα του W3C για την περιγραφή μοντέλων δεδομένων (data models) ως σύνολα από τριάδες. Όπως προαναφέραμε, κάθε τριάδα απαρτίζεται από ένα υποκείμενο, ένα κατηγόρημα και ένα αντικείμενο. Το RDF προσθέτει αρκετές σημαντικές έννοιες που προσδίδουν πολύ μεγαλύτερη ακριβολογία και ευρωστία σε αυτά τα μοντέλα.

Καταρχάς, το RDF καθιερώνει τη χρήση των URI (Uniform Resource Identifiers) ώστε να επιλύσει το πρόβλημα της αμφισημίας των οντοτήτων. Ειδικότερα, κάθε οντότητα (υποκείμενο ή αντικείμενο μίας τριάδας) ή κατηγόρημα μπορεί να θεωρηθεί πόρος (resource) και να αναπαρασταθεί με ένα URI. Όσον αφορά τις λεκτικές τιμές των αντικειμένων, η εξάλειψη της αμφισημίας πραγματοποιείται με τη συνοδεία της τιμής από τον τύπο της. Έτσι, το λεκτικό “1”^^xsd:string, το οποίο δηλώνει μία συμβολοσειρά που αντιστοιχεί στον χαρακτήρα 1, διαφέρει από το “1”^^xsd:integer, που αντιστοιχεί στον ακέραιο αριθμό 1.

Η RDF υιοθετεί αλλά δεν επιβάλλει μία σειρά τύπων δεδομένων από το προϋπάρχον πρότυπο XML Schema. Το πρότυπο XML Schema χρησιμοποιεί τον χώρο διευθύνσεων (namespace) http://www.w3.org/2001/XMLSchema# (συντομογραφικά, xsd:) για να δηλώσει τύπους δεδομένων όπως xsd:string, xsd:boolean, xsd:integer κ.λ.π. Αξίζει να σημειωθεί πως στη περίπτωση που μία λεκτική τιμή είναι τύπου xsd:string, η δήλωση του τύπου μπορεί να παραληφθεί.

Υπάρχουν διάφορες μορφές για την αναπαράσταση των τριάδων RDF. Η μορφή Turtle [17] είναι μία από τις πιο σύγχρονες και κατανοητές στον άνθρωπο. Σε αυτή τη μορφή, οι τριάδες χωρίζονται μεταξύ τους με την τελεία, ενώ τα συστατικά της τριάδας διαχωρίζονται με τον κενό χαρακτήρα. Ένα παράδειγμα αποτύπωσης μίας τριάδας σε αυτή τη μορφή δίνεται στην ακόλουθη εικόνα. Το προκείμενο παράδειγμα επαναδιατυπώνεται στην εικόνα 15 με τη βοήθεια των προθεμάτων (prefixes) που υποστηρίζει η Turtle.

```
<http://netlab.cs.unipi.gr/Apostolos_Karalis> <http://xmlns.com/foaf/0.1/knows> <http://tab.ionio.gr/Jonh_Papadakis>.
```

Εικόνα 14. Παράδειγμα τριάδας σε μορφή Turtle

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
<http://netlab.cs.unipi.gr/Apostolos_Karalis> foaf:knows <http://tab.ionio.gr/Jonh_Papadakis>.
```

Εικόνα 15. Παράδειγμα τριάδας σε μορφή Turtle με χρήση προθεμάτων

4.1.2. SPARQL

Η γλώσσα SPARQL [18] χρησιμοποιείται για τη διατύπωση ερωτημάτων σε ποικίλες πηγές δεδομένων (data sources), όταν τα δεδομένα αποθηκεύονται φυσικώς ως τριάδες RDF ή μπορούν να αναπαρασταθούν σε αυτή τη μορφή με τη βοήθεια ενδιάμεσου λογισμικού (middleware) .

Ως προς τη σύνταξη, η SPARQL μοιάζει με τη γλώσσα ερωτημάτων SQL που αξιοποιείται στις σχεσιακές βάσεις δεδομένων. Ένα ενδεικτικό ερώτημα SPARQL για την προβολή (SELECT) συγκεκριμένων στοιχείων μίας υποθετικής δομής RDF, φαίνεται στην εικόνα 16. Στο εν λόγω ερώτημα, ζητείται η εύρεση όλων των οντοτήτων που συνιστούν ταυτόχρονα υποκείμενα στη πρώτη και αντικείμενα στη δεύτερη τριάδα του τμήματος WHERE. Γίνεται εύκολα αντιληπτό πως αυτές οι οντότητες αντιπροσωπεύονται στο ερώτημα από τη μεταβλητή ?entities. Το όνομα κάθε μεταβλητής οφείλει να ξεκινάει με τον χαρακτήρα ?. Παράλληλα, σύμφωνα πάντα με τις προδιαγραφές της SPARQL, για την αναπαράσταση των τριάδων χρησιμοποιείται η μορφή Turtle.

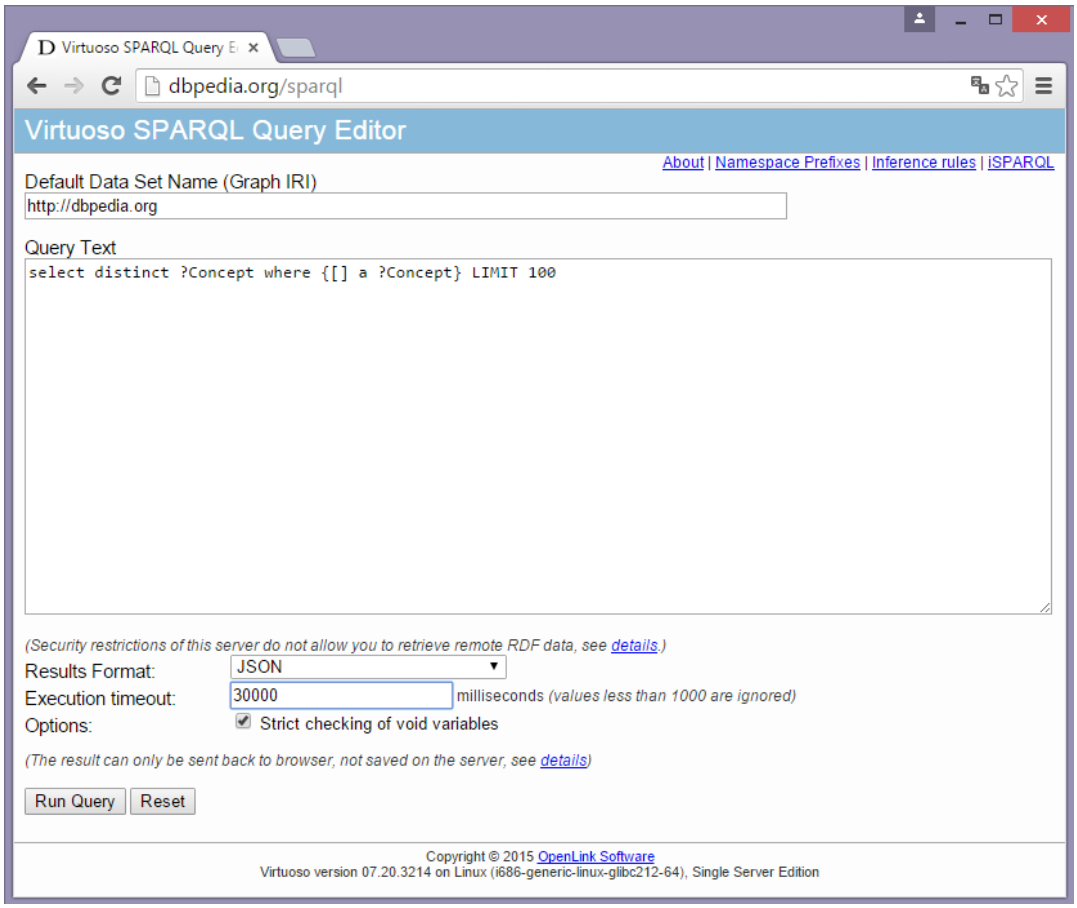
```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?entities WHERE {
<http://netlab.cs.unipi.gr/Apostolos_Karalis> foaf:knows ?entities.
?entities foaf:workInfoHomepage <http://thalassa.ionio.gr/staff/papadakis/homepage.html>
}
```

Εικόνα 16. Παράδειγμα ερωτήματος προβολής στη γλώσσα SPARQL

4.2. Το τελικό σημείο SPARQL της DBPedia

Η DBPedia παρέχει μία γραφική διεπαφή για την επιτέλεση ερωτημάτων SPARQL, δηλαδή ένα τελικό σημείο SPARQL (SPARQL Endpoint), το οποίο είναι διαθέσιμο στο ευρύ κοινό μέσω της διεύθυνσης http://dbpedia.org/sparql. Τα αποτελέσματα των ερωτημάτων μπορούν να ληφθούν σε διάφορες μορφές μεταξύ των οποίων η XML, η JSON και η CSV.

Εκείνο που έχει ιδιαίτερη σημασία να τονιστεί είναι πως κατά την ανάπτυξη της εφαρμογής μας, η οποία υποβάλλει ερωτήματα SPARQL στη DBPedia, αντιμετωπίσαμε πρόβλημα με τον μέγιστο ρυθμό υποβολής ερωτημάτων που δύναται να εξυπηρετήσει η DBPedia. Όταν λοιπόν εκτελούσαμε συνεχόμενα ερωτήματα, συνέβαινε πολύ συχνά κάποια από τα ερωτήματα να απορρίπτονται με τον κωδικό σφάλματος στην απάντηση HTTP να είναι το 503. Για να ελαχιστοποιήσουμε αυτό το πρόβλημα, υπολογίσαμε πειραματικά έναν ρυθμό υποβολής που συνήθως δεν δημιουργεί πρόβλημα.



Εικόνα 17. Το SPARQL Endpoint της DBPedia

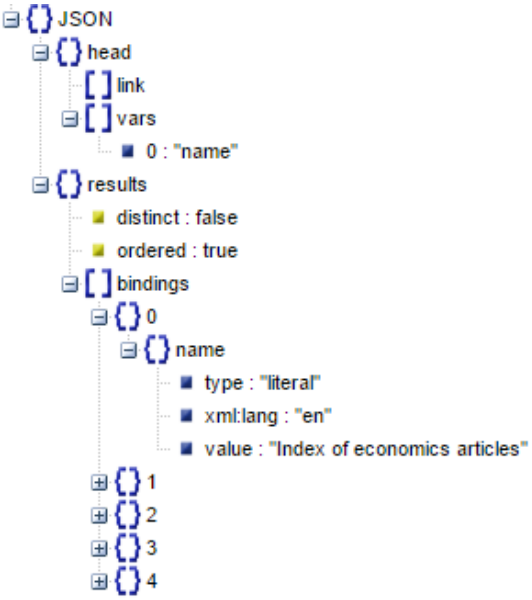
Επιστρέφοντας στο λόγο σύνταξης αυτής της ενότητας, δηλαδή στην αναζήτηση σχετικών όρων και κατηγοριών στην DBPedia, παραθέτουμε τα σχετικά ερωτήματα μαζί με τις αντίστοιχες απαντήσεις σε μορφή JSON. Η αναζήτηση πραγματοποιείται ενδεικτικά για τους όρους και τις κατηγορίες που σχετίζονται με τον όρο Economics. Φυσικά, μπορούμε να αντικαταστήσουμε τον όρο αυτόν με οποιοδήποτε άλλον.

```
prefix dc: <http://purl.org/dc/terms/>
SELECT distinct ?name WHERE {
    ?categories rdfs:label "Economics"@en .
    ?relatedterms dc:subject ?categories .
    ?relatedterms rdfs:label ?name .
 FILTER(langMatches(lang(?name), 'EN'))
} LIMIT 5
```

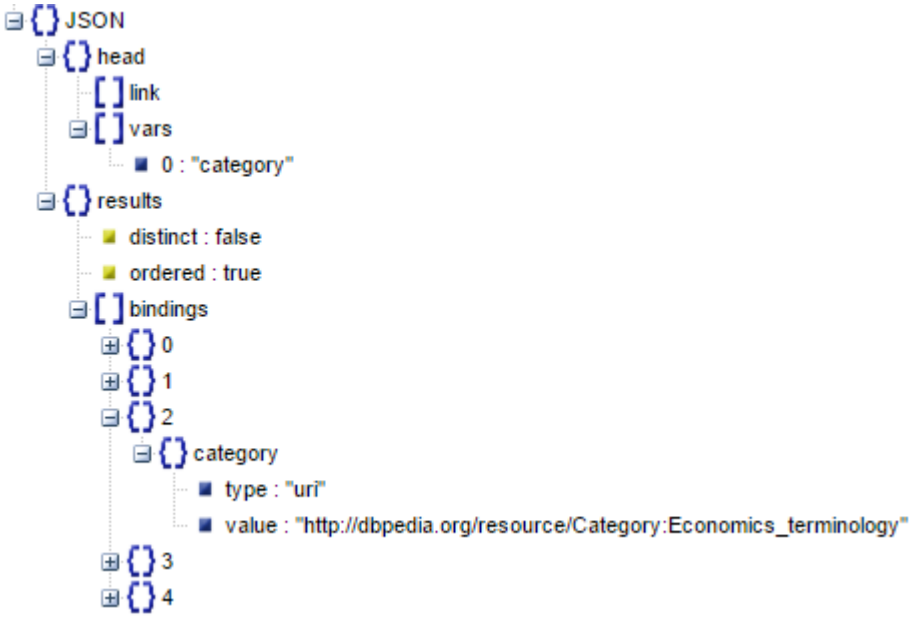
Εικόνα 18. Ερώτημα στο SPARQL Endpoint της DBPedia για την εύρεση όρων σχετικών με τον αγγλικό όρο Economics

```
prefix dc: <http://purl.org/dc/terms/>
select distinct ?category where {
        <http://dbpedia.org/resource/Economics> dc:subject ?category .
} LIMIT 5
```

Εικόνα 19. Ερώτημα στο SPARQL Endpoint της DBPedia για την εύρεση κατηγοριών σχετικών με τον αγγλικό όρο Economics



Εικόνα 20. Δομή της απάντησης JSON στο ερώτημα αναζήτησης σχετικών όρων



Εικόνα 21. Δομή των της απάντησης JSON στο ερώτημα αναζήτησης σχετικών κατηγοριών

Στην επόμενη ενότητα θα αναφερθούμε στο πληροφοριακό σύστημα βιβλιοθηκών Koha, το οποίο αποτελεί το υπολογιστικό περιβάλλον που φιλοξενεί την υπηρεσία που σχεδιάσαμε.

5. Αναζήτηση βιβλίων στο Koha

Το Koha [19] αποτελεί ένα ολοκληρωμένο Web σύστημα διαχείρισης βιβλιοθηκών. Μεταξύ άλλων παρέχει δυνατότητες καταλογογράφησης και αναζήτησης τεκμηρίων.

Στην ενότητα αυτή, θα περιγράψουμε καταρχάς τη χρήση της έκδοσης 1.1 του πρωτοκόλλου SRU [20] για την εύρεση του αριθμού των βιβλίων που υπάρχουν στη βιβλιοθήκη που χρησιμοποιεί το Koha και που σχετίζονται με ένα δοσμένο συγγραφέα, θέμα ή τίτλο. Κατόπιν, για τις περιπτώσεις που βρίσκονται σχετικά βιβλία, θα καταγράψουμε το HTTP URL μονοπάτι μέσω του οποίου μπορούμε να λάβουμε την ιστοσελίδα με τη λίστα αυτών των βιβλίων. Θα υποθέσουμε ότι το σύστημα Koha στο οποίο θα υποβληθούν τα ερωτήματα έχει αριθμό έκδοσης 3.20 και όνομα www.example.com, ενώ «ακούει» για αιτήσεις SRU στην πόρτα 9999.

Μία αίτηση SRU συνιστά ουσιαστικά ένα HTTP URL όπου το τμήμα του ερωτήματος (query) του έχει συγκεκριμένη μορφή, που για την έκδοση 1.1 περιγράφεται αναλυτικά στο [20]. Στο Koha, για να λάβουμε το πλήθος των βιβλίων που σχετίζονται με έναν συγγραφέα, θέμα ή τίτλο χρησιμοποιούμε αντίστοιχα τα εξής ερωτήματα SRU:

- http://www.example.com:9999/biblios?version=1.1&operation=searchRetrieve&query=***Author*%3D**Tanenbaum,
- http://www.example.com:9999/biblios?version=1.1&operation=searchRetrieve&query=***Subject%3DJava*** και
- http://www.example.com:9999/biblios?version=1.1&operation=searchRetrieve&query=***Title%3DNetworks***.

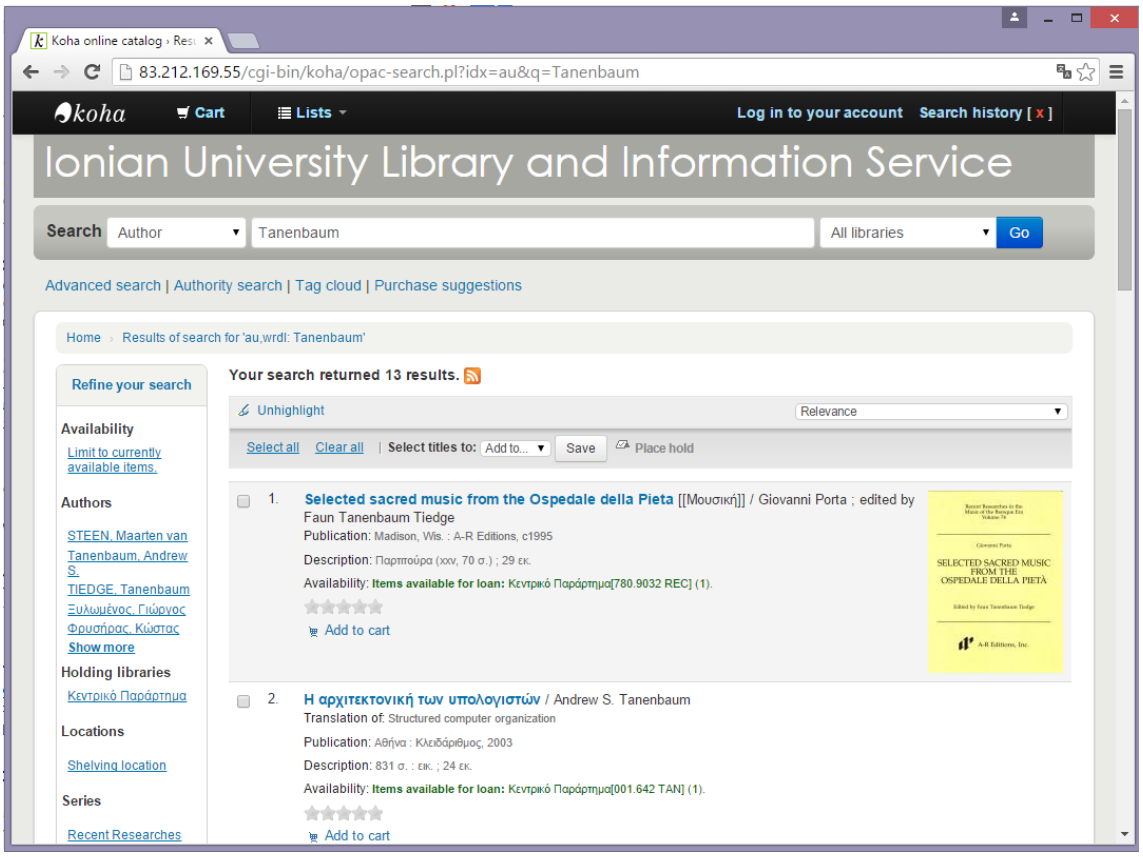
Προφανώς, οι τιμές που δώσαμε στα SRU ερωτήματα για τον συγγραφέα, το θέμα και τον τίτλο είναι ενδεικτικές. Οι απαντήσεις που θα λάβουμε έχουν μορφή XML και δομή που φαίνεται στη κατοπινή εικόνα.

```
<zs:searchRetrieveResponse
   xmlns:zs="http://www.loc.gov/zing/srw/" >
   <zs:version>1.1</zs:version>
   <zs:numberOfRecords>13</zs:numberOfRecords>
</zs:searchRetrieveResponse>
```

Εικόνα 22. Παράδειγμα απάντησης του Koha σε ένα αίτημα SRU για την εύρεση του πλήθους των βιβλίων που σχετίζονται με έναν όρο

Για την εμφάνιση της λίστας των βιβλίων που συνδέονται με έναν συγγραφέα, θέμα ή τίτλο, θα πρέπει να ζητούμε από το Koha κατά αντιστοιχία τις παρακάτω σελίδες:

- http://www.example.com/cgi-bin/koha/opac-search.pl?idx=***au***&q=**Tanenbaum**
- http://www.example.com/cgi-bin/koha/opac-search.pl?idx=***su***&q=**Java**
- http://www.example.com/cgi-bin/koha/opac-search.pl?idx=***ti***&q=**Networks**



Εικόνα 23. Παράδειγμα λίστας βιβλίων στο Koha σχετικών με έναν συγγραφέα

6. Περιγραφή εφαρμογής

Υπό μία αφαιρετική σκοπιά, η πειραματική εφαρμογή που αναπτύξαμε, λαμβάνει επίκαιρα ή/και δημοφιλή tweets και αξιοποιώντας τα hashtags αυτών, προτείνει σχετικά βιβλία. Στη γλώσσα του Twitter, ένα hashtag είναι μία ειδική σύντομη συμβολοσειρά που τυπικά οφείλει να εκφράζει το θέμα ενός Tweet.

Εικόνα 24. Αφαιρετική προσέγγιση της εφαρμογής

Το Twitter καθιστά δυνατή την άμεση λήψη των πιο επίκαιρων ή/και δημοφιλών hashtags που αφορούν είτε μία συγκεκριμένη γεωγραφική περιοχή είτε όλη την οικουμένη. Ωστόσο, στην αρχική υλοποίηση της εφαρμογής μας, η οποία χρησιμοποιούσε την εν λόγω υπηρεσία που περιγράφεται στο [21], φάνηκε ότι τα hashtags είναι πολύ γενικά ή δύσκολα να ερμηνευτούν.

Στη κατεύθυνση αντιμετώπισης αυτού του προβλήματος, στραφήκαμε στο RESTFul Search API του Twitter, όπου πλέον αιτούμαστε τη λήψη μόνο εκείνων των Tweets που αφορούν έναν συγκεκριμένο πληροφοριακό χώρο (π.χ. τον προγραμματισμό-programming). Με τη συγκεκριμένη μέθοδο, πετύχαμε σημαντική μείωση της πιθανότητας εσφαλμένης ερμηνείας των hashtags ενός Tweet.

Ένα δεύτερο πρόβλημα προκύπτει από τη μεγάλη ελευθερία στη σύνταξη ενός hashtag, που συχνά οδηγεί στην παραβίαση των κανόνων της φυσικής γλώσσας (π.χ. αγγλικής) στην οποία θεωρητικά συντάσσεται. Για παράδειγμα, όταν ένα hashtag περιέχει περισσότερες από μία λέξεις, αυτές τυπικά δεν χωρίζονται με τον κενό χαρακτήρα. Όπως είναι φυσικό, απαιτείται η χρήση κάποιου φίλτρου «Did you mean» το οποίο θα μετατρέπει τα hashtags σε μία ορθογραφική και ορθοσυντακτική μορφή, την οποία χάριν ευκολίας θα την ονομάσουμε καθαρή τιμή του hashtag. Στην εφαρμογή μας, το ρόλο αυτού του φίλτρου αναλαμβάνει η υπηρεσία Spelling Suggestion του Bing.

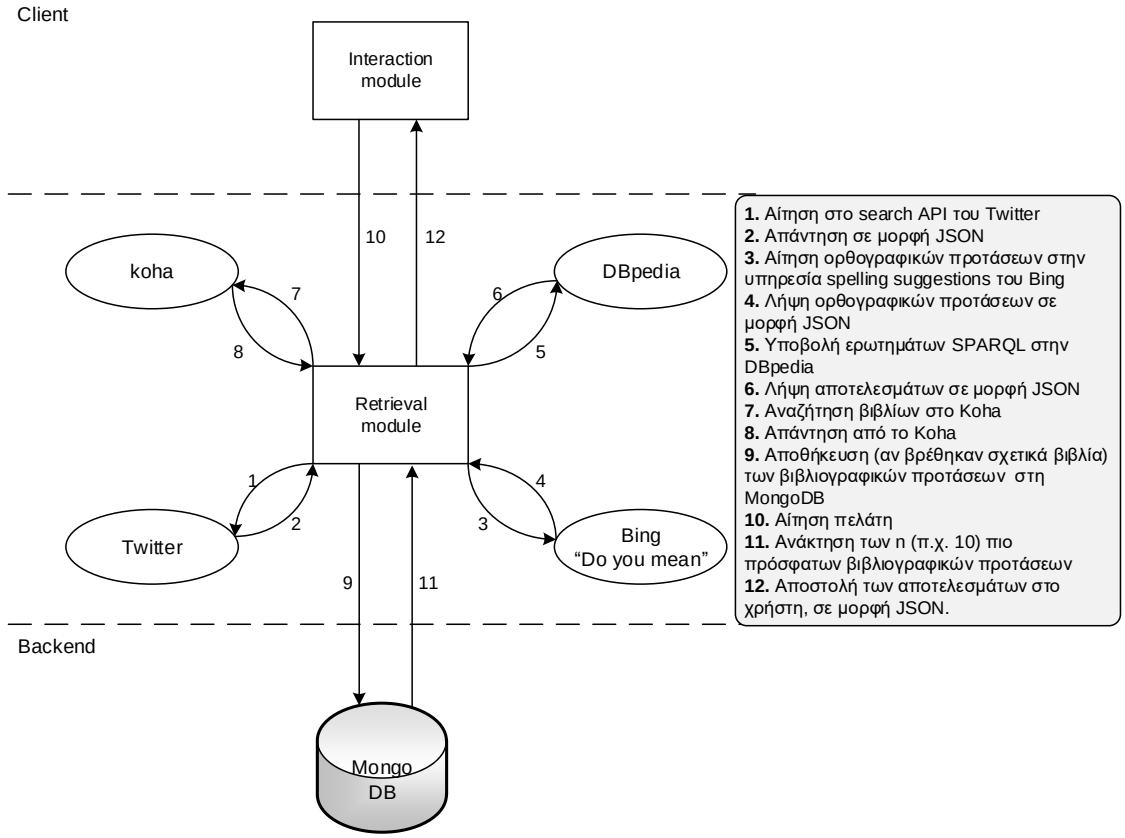
Ύστερα, η εφαρμογή μας, χρησιμοποιεί την καθαρή τιμή ενός hashtag τόσο για την αναζήτηση των σχετικών βιβλίων στον ηλεκτρονικό κατάλογο ενός συστήματος Koha όσο για την εύρεση, μέσω του SPARQL Endpoint της DBPedia, των όρων και των κατηγοριών που σχετίζονται με αυτήν την τιμή. Οι συναφείς όροι και κατηγορίες αξιοποιούνται και αυτές για την εύρεση επιπλέον βιβλίων, αυξάνοντας την αποτελεσματικότητα της εφαρμογής.

Η αναζήτηση στο Koha πραγματοποιείται σε δύο στάδια όπως αυτά περιγράφονται στην ενότητα 5. Στο σημείο αυτό, κρίνεται επιβεβλημένο να σημειωθεί ότι γνωρίζοντας το πληροφοριακό χώρο (π.χ. programming) από τον οποίο προέρχεται ένα hashtag, στις αναζητήσεις βιβλίων που πραγματοποιούνται εισάγεται το αντίστοιχο πρόθεμα (π.χ. programming), ώστε να επιτευχθεί μεγαλύτερη ακρίβεια (precision) στη σχετικότητα των επιστρεφόμενων βιβλίων με τον όρο της αναζήτησης.

Τα ευρεθέντα αποτελέσματα, δηλαδή οι προτάσεις επίκαιρων/δημοφιλών βιβλίων, αποθηκεύονται σε μία βάση δεδομένων και ανακτώνται από τους χρήστες του συστήματος Koha μέσω μίας παρεχόμενης δικτυακής υπηρεσίας (Web Service).

6.1. Αρχιτεκτονική εφαρμογής

Η εφαρμογή μας υλοποιεί αρχιτεκτονική τριών επιπέδων (3-tier) στην οποία ο πελάτης (client) είναι ένα τμήμα Javascript εμφωλευμένο στον html κώδικα της αρχικής σελίδας του Koha, ο εξυπηρετητής που προσφέρει τις βιβλιογραφικές προτάσεις έχει κατασκευαστεί με τη χρήση του NodeJS [22], ενώ επιλέχτηκε ως βάση η MongoDB [23]. Στο παρακάτω σχήμα παρουσιάζεται αναλυτικά η αρχιτεκτονική του συστήματος μας.



Εικόνα 25. Η αρχιτεκτονική της εφαρμογής

6.2. Δομή κώδικα

Θα αναφερθούμε αποκλειστικά στον κώδικα του εξυπηρετητή καθότι το κομμάτι του πελάτη είναι τετριμμένο. Όπως προαναφέραμε, ο εξυπηρετητής υλοποιήθηκε με τη χρησιμοποίηση του NodeJS. Το NodeJS είναι ένα λογισμικό για την ανάπτυξη εφαρμογών με τη χρήση της γλώσσας σεναρίων JavaScript. Δημιουργήθηκε πάνω στη μηχανή εκτέλεσης κώδικα JavaScript του φυλλομετρητή Chrome με στόχο την εύκολη ανάπτυξη γρήγορων και κλιμακωτών δικτυακών εφαρμογών. Το NodeJS υποστηρίζει το ασύγχρονο μοντέλο προγραμματισμού, επιτρέποντας την επιτέλεση των λειτουργιών εισόδου/εξόδου δίχως να μπλοκάρεται άσκοπα η εκτέλεση της εκάστοτε εφαρμογής.

Στο NodeJS η οργάνωση του κώδικα γίνεται σε τμήματα (modules). Ένα module προσφέρει συνήθως μία προγραμματιστική διεπαφή μέσω της οποίας είναι εφικτή η αξιοποίηση της λειτουργικότητας που αυτό προσφέρει.

Ο κώδικας του εξυπηρετητή μας ακολουθεί την οργάνωση σε modules. Ειδικότερα, συντίθεται από τα ακόλουθα βασικά modules:

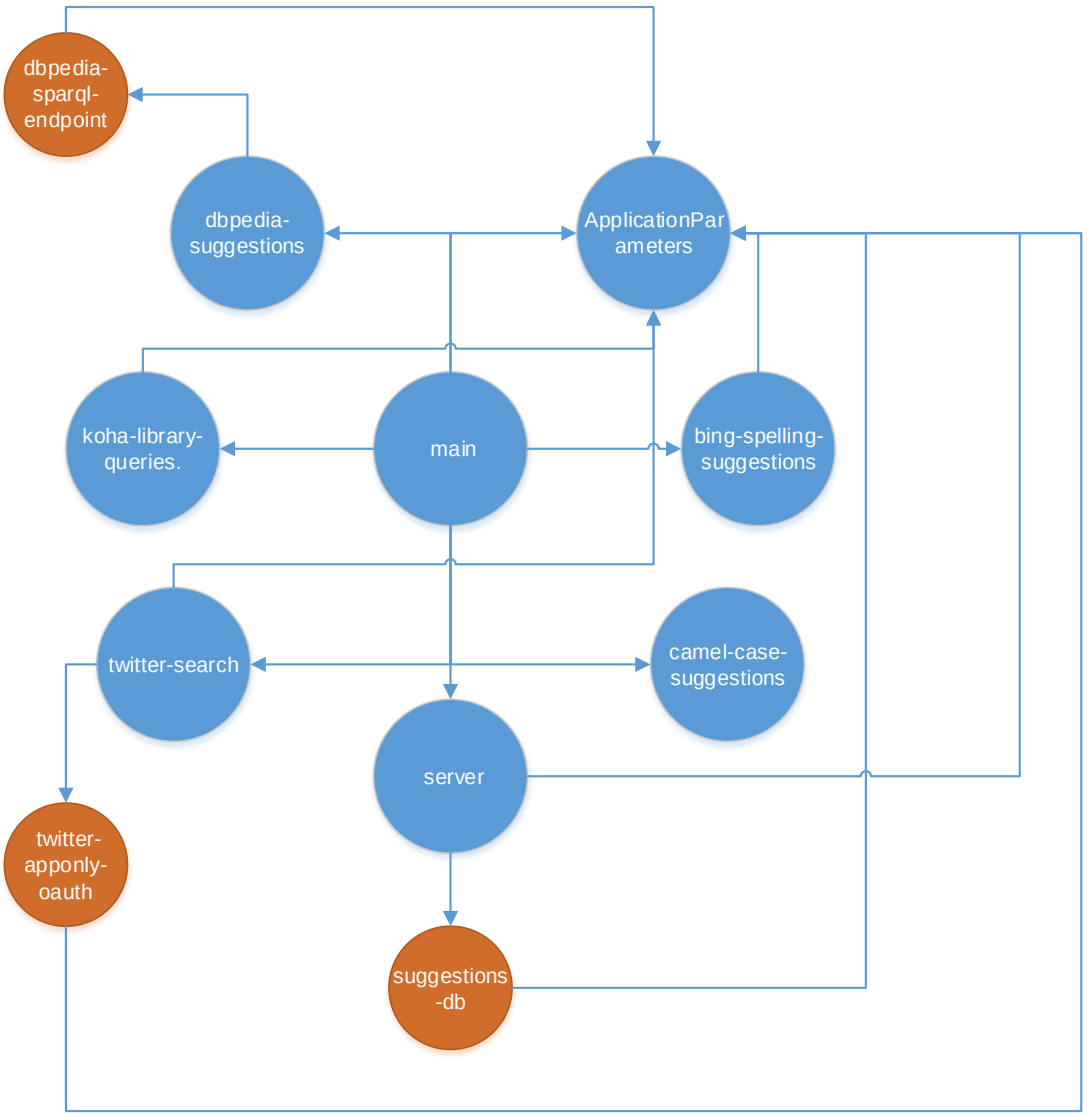
- **ApplicationParameters.js**: περιέχει τις παραμέτρους και τις τιμές τους που απαιτούνται από την εφαρμογή. Για παράδειγμα, περιλαμβάνει το consumer_key και το consumer_secret που είναι απαραίτητα για την αλληλεπίδραση με το RESTFul Search API του Twitter.
- **bing-spelling-suggestions.js**: παρέχει τη δυνατότητα υποβολής ερωτημάτων στην υπηρεσία Spelling Suggestions του Bing.
- **capitalization-suggestions.js**: αποτελείται από μία συνάρτηση που υπολογίζει όλες εκείνες τις μορφές μιας φράσης στις οποίες μία ή περισσότερες από τις λέξεις που την

αποτελούν έχει κεφαλαίο το πρώτο της γράμμα. Επιπλέον, επιτρέπεται η επιλόγη του διαχωριστικού των λέξεων που θα χρησιμοποιηθεί στις προαναφερόμενες μορφές μιας φράσης. Το συγκεκριμένο module, αξιοποιείται κατά την υποβολή των ερωτημάτων SPARQL στην DBPedia, όπου για παράδειγμα η οντότητα που αναπαριστά το “Cloud computing” βρίσκεται στο URI dbpedia.org/resource/Cloud_computing αλλά όχι στο dbpedia.org/resource/cloud_Computing. Ουσιαστικά, η εφαρμογή μας δοκιμάζει όλες τις παραγόμενες περιπτώσεις.

- **dbpedia-suggestions.js**: επιτρέπει την αναζήτηση στη σημασιολογική δομή δεδομένων της DBpedia, όρων και κατηγοριών που σχετίζονται με κάποιον δοσμένο όρο.
- **koha-library-queries.js**: υλοποιεί τον μηχανισμό των δύο φάσεων για την αναζήτηση βιβλίων στο Koha.
- **main.js**: είναι το κύριο τμήμα της εφαρμογής από το οποίο ξεκινάει η εκτέλεση του κώδικα.
- **twitter-search.js**: προσφέρει τη διεπαφή για την επικοινωνία με το RESTFul Search API του Twitter.
- **server.js**: αποτελεί το module που αλληλοεπιδρά με τους πελάτες. Για λόγους απόδοσης, η εφαρμογή μας δημιουργεί καινούργια διεργασία για την εκτέλεση αυτού το module.

Τα δευτερεύοντα modules που δημιουργήθηκαν για την υποστήριξη της λειτουργίας του εξυπηρετητή είναι τα ακόλουθα:

- **dbpedia-sparql-endpoint.js**: υποστηρίζει την υποβολή ερωτημάτων στο SPARQL Endpoint της DBPedia.
- **suggestions-db.js**: συνιστά το module αλληλεπίδρασης με τη βάση δεδομένων, επιτρέποντας την άμεση αποθήκευση και ανάκτηση βιβλιογραφικών προτάσεων.
- **twitter-apponly-oauth.js**: προσφέρει την υποδομή για την υποστήριξη του τρόπου αυθεντικοποίησης ιδίας εφαρμογής (Application-only Authentication) του Twitter.



Εικόνα 26. Τα βασικά (μπλε) και δευτερεύοντα (πορτοκαλί) modules και οι μεταξύ τους εξαρτήσεις

6.3. Σχήμα αποθήκευσης και ανάκτησης

Η αποθήκευση των βιβλιογραφικών προτάσεων πραγματοποιείται σε μία βάση MongoDB. Η MongoDB είναι μία μη σχεσιακή (NoSQL) βάση δεδομένων, όπου τα δεδομένα οργανώνονται σε συλλογές (Collections). Μία συλλογή αποτελεί ένα σύνολο από έγγραφα (documents) στα οποία αποθηκεύονται τα δεδομένα με BSON [24] (Binary JSON) μορφή. Οι συλλογές είναι αντίστοιχες των πινάκων (tables) των σχεσιακών βάσεων δεδομένων, ενώ οι εγγραφές (records) που περιέχονται ακολουθούν τη μορφή των JSON objects, με τη διαφορά ότι αποθηκεύονται με δυαδική (binary) μορφή και όχι με μορφή κειμένου (text).

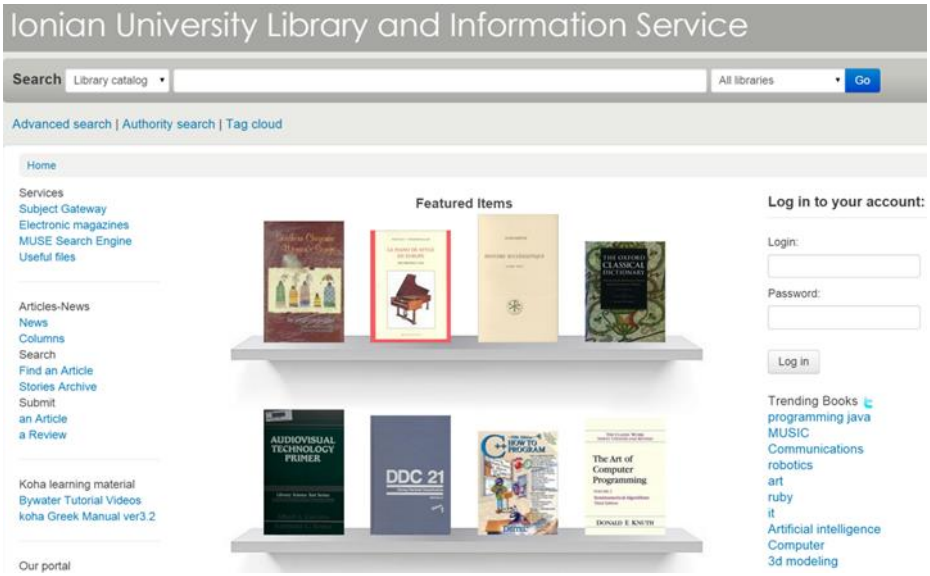
Για τις ανάγκες της εφαρμογής μας, δημιουργήσαμε τη συλλογή trends στην οποία εμπεριέχεται το έγγραφο AssetSuggestions στο οποίο αποθηκεύονται οι βιβλιογραφικές προτάσεις. Στο ακόλουθο σχήμα φαίνεται ένα παράδειγμα εγγραφής από το οποίο μπορεί να γίνει εύκολα αντιληπτή η γενική μορφή των εγγραφών. Ειδικότερα, κάθε εγγραφή αποτελείται από τα εξής πεδία:

- **_id**: αποτελεί ένα μοναδικό αναγνωριστικό που λειτουργεί ως κλειδί για την εγγραφή. Παράγεται από αυτόματα από τη MongoDB.
- **keyword**: περιέχει τον όρο για τον οποίο βρέθηκαν βιβλία στο Koha.
- **keyword_types**: συνιστά έναν πίνακα που περιγράφει υπό ποια σκοπιά ή ποιες σκοπιές έγινε αναζήτηση βάσει του συγκεκριμένου keyword. Οι δυνατές τιμές που μπορεί να περιέχει ο πίνακας είναι title, author και subject για να δηλωθεί ότι έγινε αναζήτηση βιβλίων τον οποίο ο τίτλος, ο συγγραφέας και το θέμα αντίστοιχα, ταιριάζουν με το keyword.
- **url**: είναι το URL της σελίδας με τη λίστα των ευρεθέντων βιβλίων.
- **hashtag**: είναι το hashtag από το οποίο παρήχθη τελικά η συγκεκριμένη βιβλιογραφική πρόταση.
- **timestamp**: χρονοσφραγίδα δημιουργίας της εγγραφής.

```
{
    "_id": ObjectId("556b73e5aecdc82d0b0e709d"),
    "keyword": "programming python",
    "keyword_types": [
        "title"
    ],
    "url": "http://83.212.169.55:80/cgi-bin/koha/opac-search.pl?idx=ti&q=programming%20python",
    "hashtag": "python",
    "timestamp": ISODate("2015-05-31T20:49:41.964Z")
}
```

Εικόνα 27. Παράδειγμα εγγραφής (record) στο έγγραφο (document) AssetSuggestions

Κατά την αποστολή των βιβλιογραφικών προτάσεων σε έναν πελάτη, τα πεδία _id και timestamp δεν περιλαμβάνονται στην απάντηση που εννοείται πως ακολουθεί το μορφότυπο JSON. Οι βιβλογραφικές προτάσεις τοποθετούνται εντός ενός πίνακα JSON με το όνομα suggestions.



Εικόνα 28. Στιγμιότυπο (κάτω δεξιά) βιβλιογραφικών προτάσεων - Trending Books

7. Αξιολόγηση εφαρμογής

Η διαδικασία αξιολόγησης που επιλέξαμε, στοχεύει στον προσδιορισμό του βαθμού στον οποίο τα δημοφιλή tweets επηρεάζουν τους χρήστες των βιβλιοθηκών στην επιλογή τεκμηρίων. Για τον σκοπό αυτό, αποφασίσαμε να συγκρίνουμε τον αριθμό των τεκμηρίων που επιλέχθηκαν μέσω της εφαρμογής μας σε σχέση με το πλήθος των τεκμηρίων που διαλέχθηκαν με τη βοήθεια των υπολοίπων υπηρεσιών αναζήτησης του ηλεκτρονικού καταλόγου της βιβλιοθήκης. Ένας άλλος στόχος της αξιολόγησης ήταν να διερευνηθεί κατά πόσο η εφαρμογή μας βοηθά τους χρήστες στην επιλογή τεκμηρίων της βιβλιοθήκης που σχετίζονται με επίκαιρα ή/και δημοφιλή θέματα. Στην κατεύθυνση αυτή, πραγματοποιήσαμε το ακόλουθο πείραμα.

7.1. Το πείραμα

Το πείραμα έλαβε χώρα στο πλαίσιο του μαθήματος «Τεχνολογίες Διαδικτύου» του μεταπτυχιακού προγράμματος «Πληροφορική» του Πανεπιστημίου Πειραιώς. Πιο συγκεκριμένα, 10 μεταπτυχιακοί φοιτητές, 4 γυναίκες και 6 άντρες, συμμετείχαν στο πείραμα. Οι ηλικίες τους κυμαίνονταν από 25 έως 45 και κατά μέσο όρο όλοι είχαν κάποια εμπειρία με τα μέσα κοινωνικής δικτύωσης και τους ηλεκτρονικούς καταλόγους των βιβλιοθηκών. Σύμφωνα με το πείραμα, ζητήθηκε από τους φοιτητές να επισκεφτούν την αρχική σελίδα της βιβλιοθήκης μέσα ένα χρονικό διάστημα 3 ημερών (12-15 Μαΐου 2015) και να επιλέξουν 5 τεκμήρια που βρήκαν ελκύστηκα στο θεματικό χώρο του μαθήματος. Κατόπιν, τους ζητήθηκε να αποστείλουν ένα μήνυμα ηλεκτρονικού ταχυδρομείου με τις επιλογές τους.

Όπως φαίνεται στην εικόνα 28, το γραφικό περιβάλλον της αρχικής σελίδας του ηλεκτρονικού καταλόγου της βιβλιοθήκης αντιμετωπίζει ξεχωριστά καθεμία από τις υπηρεσίες αναζήτησης που προσφέρει, συμπεριλαμβανομένης της υπηρεσίας μας. Ειδικότερα, η αρχική σελίδα παρέχει πρόσβαση στις ακόλουθες υπηρεσίες: α) γενική επιλογή αναζήτησης β) εξειδικευμένη επιλογή αναζήτησης, γ) επιλεγμένα τεκμήρια (featured items) δ) η προτεινόμενη υπηρεσία μας. Οι συνολικές αλληλεπιδράσεις των χρηστών με το σύστημα καταγράφηκαν στα αρχεία καταγραφής (log files) του συστήματος.

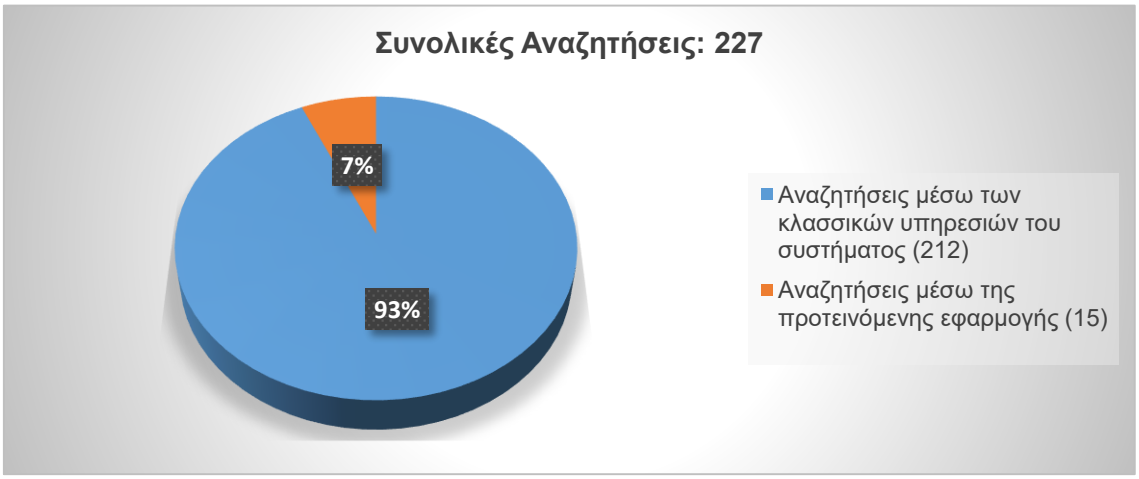
7.2. Αποτελέσματα

Στο πέρας του πειράματος, οι συμμετέχοντες είχαν επιλέξει 40 μοναδικά τεκμήρια (βάσει των μηνυμάτων ηλεκτρονικού ταχυδρομείου που λήφθηκαν) στη θεματική περιοχή του μαθήματος.

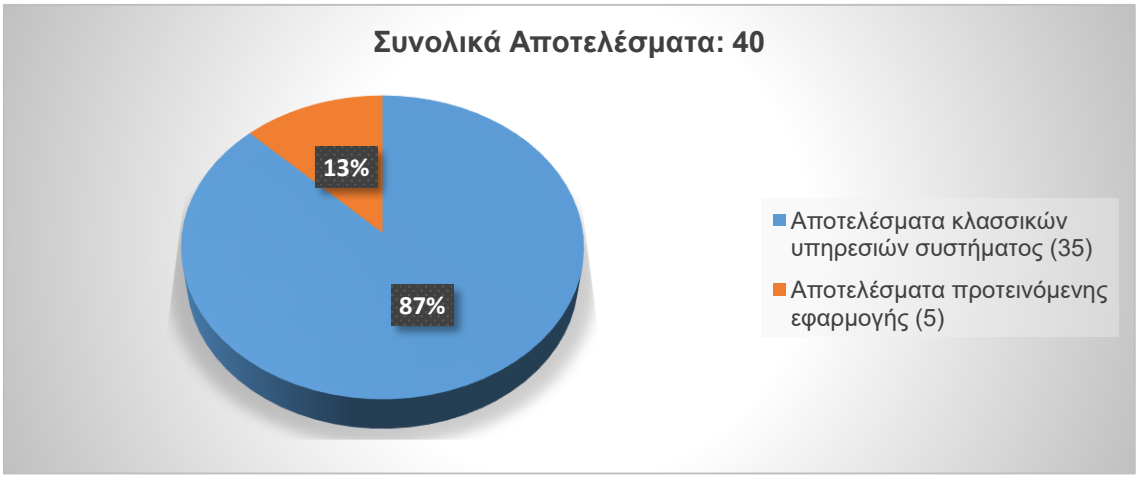
Σύμφωνα με το αρχείο καταγραφής του συστήματος, ελήφθησαν 227 αιτήσεις HTTP GET για τις διάφορες υπηρεσίες της αρχικής σελίδας. Οι αιτήσεις αυτές οδήγησαν είτε σε μεμονωμένα τεκμήρια (31 αιτήσεις) είτε σε λίστες τεκμηρίων (196 αιτήσεις). Ως εκ τούτου, συνάγεται το συμπέρασμα ότι οι συμμετέχοντες αξιολόγησαν αυτά τα στοιχεία/λίστες και επέλεξαν τα 40 αντικείμενα που τους προσέλκυσαν περισσότερο.

Όπως φαίνεται στην εικόνα 29, οι 15 από τις 227 αιτήσεις προήλθαν από την υπηρεσία που προσφέρει η εφαρμογή μας και τα υπόλοιπα 212 από τις άλλες υπηρεσίες του συστήματος. Επιπλέον, 5 από τα 40 επιλεγμένα τεκμήρια βρέθηκαν στα αποτελέσματα αναζήτησης των 15 αιτημάτων που προήλθαν από την υπηρεσία μας. Συνεπακόλουθα, 35 από τα 40 τεκμήρια διαλέχθηκαν από τις υπόλοιπες υπηρεσίες αναζήτησης του συστήματος.

Από μία οπτική γωνία, ο αντίκτυπος της προτεινόμενης εφαρμογής είναι μάλλον χαμηλός (15 από τις 227 αιτήσεις προήλθαν από την προτεινόμενη υπηρεσία). Ωστόσο, μία περαιτέρω ανάλυση των δεδομένων δείχνει ότι η επίδραση των προτάσεων της προτεινόμενης υπηρεσίας ήταν σημαντικά υψηλότερη από αυτήν των υπόλοιπων υπηρεσιών αναζήτησης του συστήματος. Πιο συγκεκριμένα, οι συμμετέχοντες επέλεξαν 5 από τα 15 τεκμήρια που προσφέρθηκαν μέσω της λίστας που τους παρείχε η υπηρεσία μας. Στον αντίποδα, οι συμμετέχοντες επέλεξαν 35 τεκμήρια από τα αποτελέσματα 212 αναζητήσεων που επιτέλεσαν μέσω των υπολοίπων υπηρεσιών αναζήτησης του συστήματος.

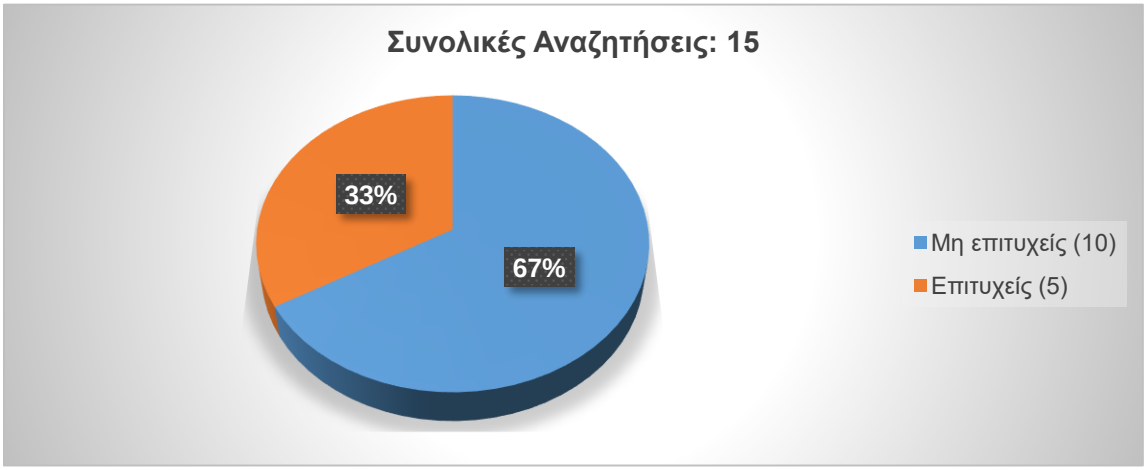


Εικόνα 29. Αναζητήσεις που πραγματοποιήθηκαν στο σύστημα



Εικόνα 30. Αποτελέσματα κλασσικών υπηρεσιών συστήματος και προτεινόμενης εφαρμογής

Εικόνα 31. Αποτελέσματα κλασσικών υπηρεσιών συστήματος



Εικόνα 32. Αποτελέσματα προτεινόμενης εφαρμογής

7.3. Χρήσιμες παρατηρήσεις

Ένα συμπέρασμα το οποίο μπορούμε να εξάγουμε με ασφάλεια από το παραπάνω πείραμα είναι η δυνατότητα να ενσωματώσουμε σε μία βιβλιοθήκη, πληροφορίες που προέρχονται από μέσα κοινωνικής δικτύωσης.

Λαμβάνοντας υπόψιν τους περιορισμούς χρήσης των υπηρεσιών τρίτων (third party services) που αξιοποιούνται από την εφαρμογή μας, εκτελέσαμε πειραματικά την εφαρμογή για ένα διάστημα 13.5 ωρών κατά το οποίο η ανανέωση των βιβλιογραφικών προτάσεων πραγματοποιούνταν κάθε 1.5 ώρα. Μέσα σε αυτό το διάστημα, καταγράψαμε 720 μοναδικά hashtags. Η εφαρμογή κατάφερε να εντοπίσει σχετικά τεκμήρια, στον ηλεκτρονικό κατάλογο της βιβλιοθήκης, για τα 35 από αυτά. Επομένως, ακόμα και εντός μίας συγκεκριμένης πληροφοριακής περιοχής (π.χ. πληροφορικής) είναι φανερό ότι η πιθανότητα αντιστοίχισης ενός hashtag σε ένα τεκμήριο είναι σχετικά μικρή. Αυτό μπορεί να οφείλεται στο γεγονός πως το λεξιλόγιο των tweets διαφέρει σημαντικά από τη γλώσσα ευρετηρίασης που χρησιμοποιείται από τη βιβλιοθήκη.

Ωστόσο, αξίζει να σημειωθεί ότι 29 από τα 35 hashtags αντιστοιχήθηκαν με τη βοήθεια της DBPedia. Αυτό καταδεικνύει πως η σημασιολογική ανάλυση που προσφέρεται από την DBPedia βελτιώνει σε μεγάλο βαθμό την αποτελεσματικότητα της εφαρμογής μας.

8. Συμπεράσματα – Σύνοψη

Στην παρούσα διατριβή παρουσιάσαμε μία εφαρμογή ικανή να παρέχει στον ηλεκτρονικό κατάλογο μίας βιβλιοθήκης, επίκαιρες ή/και δημοφιλείς βιβλιογραφικές προτάσεις, μέσω της αξιοποίησης αντίστοιχα των επίκαιρων ή/και δημοφιλών hashtags του Twitter. Η εγκατάσταση της εφαρμογής πραγματοποιήθηκε στον ηλεκτρονικό κατάλογο Koha του Ιόνιου Πανεπιστημίου.

Η αξιολόγηση της εφαρμογής κατέδειξε ότι οι ηλεκτρονικοί κατάλογοι των βιβλιοθηκών μπορούν να ωφεληθούν από την αξιοποίηση των επίκαιρων ή/και δημοφιλών πληροφοριών που παρέχουν τα μέσα κοινωνικής δικτύωσης, όπως το Twitter. Φάνηκε επίσης ότι η χρήση του σημασιολογικού ιστού μπορεί να διαδραματίσει καθοριστικό ρόλο στη βελτίωση της συνολικής απόδοσης τέτοιου είδους υπηρεσιών, που βασίζονται σε πληροφορίες εκτός βιβλιοθήκης.

9. Αναφορές

[1] Twitter: https://Twitter.com (τελευταία πρόσβαση 22/10/2015).

[2] Αναφορά στο RESTful Search API του Twitter. Διαθέσιμη στην ιστοσελίδα https://dev.Twitter.com/rest/public/search (τελευταία πρόσβαση 22/10/2015).

[3] Αναφορά στο RESTful Search API του Twitter. Διαθέσιμη στην ιστοσελίδα https://dev.Twitter.com/rest/reference/get/search/tweets (τελευταία πρόσβαση 22/10/2015).

[4] Hammer-Lahav, E., Ed., "The OAuth 1.0 ProTocol", RFC 5849, DOI 10.17487/RFC5849, Απρίλιος 2010. Διαθέσιμο στην ιστοσελίδα http://www.rfc-ediTor.org/info/rfc5849 (τελευταία πρόσβαση 22/10/2015).

[5] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, Οκτώβριος 2012. Διαθέσιμο στην ιστοσελίδα http://www.rfc-ediTor.org/info/rfc6749 (τελευταία πρόσβαση 22/10/2015).

[6] Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource LocaTors (URL)", RFC 1738, DOI 10.17487/RFC1738, Δεκέμβριος 1994. Διαθέσιμο στην ιστοσελίδα http://www.rfc-ediTor.org/info/rfc1738 (τελευταία πρόσβαση 22/10/2015).

[7] Microsoft Azure: https://azure.microsoft.com (τελευταία πρόσβαση 31/10/2015).

[8] Bing, "Migrating To the Bing Search API", Σεπτέμβριος 2012. Διαθέσιμο στην ιστοσελίδα http://go.microsoft.com/fwlink/?LinkID=272625&clcid=0x409 (τελευταία πρόσβαση 31/10/2015).

[9] Bing, "Bing Search API – Quick Start and Code Samples", Οκτώβριος 2012. Διαθέσιμο στην ιστοσελίδα http://go.microsoft.com/fwlink/?LinkID=272626&clcid=0x409 (τελευταία πρόσβαση 31/10/2015).

[10] Bing, "Schema Tabular Documentation for the Bing Search API", Ιανουάριος 2013. Διαθέσιμο στην ιστοσελίδα http://go.microsoft.com/fwlink/?LinkID=272627&clcid=0x409 (τελευταία πρόσβαση 31/10/2015).

[11] Αναφορά στο πρωτόκολλο ODATA. Διαθέσιμη στην ιστοσελίδα http://www.odata.org (τελευταία πρόσβαση 31/10/2015).

[12] Αναφορά στη DBPedia. Διαθέσιμη στην ιστοσελίδα http://wiki.dbpedia.org/about (τελευταία πρόσβαση 23/11/2015).

[13] John Davies, Frank van Harmelen, Dieter Fensel, "Towards the Semantic Web: OnTology-driven Knowledge Management", John Wiley & Sons, Inc., New York, NY, 2002.

[14] Toby Segaran, Colin Evans, Jamie Taylor, Segaran Toby, Evans Colin, Taylor Jamie, "Programming the Semantic Web", O'Reilly Media, Inc., 2009.

[15] John Hebeler, Matthew Fisher, Ryan Blace, Andrew Perez-Lopez, "Semantic Web Programming", Wiley Publishing, 2009.

[16] Θεόδωρος Ανδρόνικος, Ιωάννης Παπαδάκης, Μιχαήλ Στεφανιδάκης, "Ανοικτά συνδεδεμένα δεδομένα και εφαρμογές, μια πρακτική προσέγγιση στον Σημασιολογικό Ιστό" (προς δημοσίευση).

[17] W3C, "RDF 1.1 Turtle, Φεβρουάριος 2014". Διαθέσιμο στην ιστοσελίδα http://www.w3.org/TR/turtle/ (τελευταία πρόσβαση 27/11/2015).

[18] W3C, “SPARQL Query Language for RDF”, Ιανουάριος 2008. Διαθέσιμο στην ιστοσελίδα http://www.w3.org/TR/rdf-sparql-query/ (τελευταία πρόσβαση 27/11/2015).

[19] Koha Library Software: https://koha-community.org/ (τελευταία πρόσβαση 27/11/2015).

[20] The Library of Congress, “SRU 1.1.”. Διαθέσιμο στην ιστοσελίδα http://www.loc.gov/standards/sru/sru-1-1.html (τελευταία πρόσβαση 27/11/2015).

[21] Αναφορά στην υπηρεσία λήψης των trending Topics του Twitter. Διαθέσιμη στην ιστοσελίδα https://dev.Twitter.com/rest/reference/get/trends/place (τελευταία πρόσβαση 27/11/2015).

[22 [NodeJS: https://nodejs.org/en/ (τελευταία πρόσβαση 27/11/2015).

[23] MongoDB: https://www.mongodb.org/ (τελευταία πρόσβαση 27/11/2015).

[24] BSON: http://bsonspec.org/ (τελευταία πρόσβαση 27/11/2015).

[25]. Agee, J. (2005), “Collection evaluation: a foundation for collection development. Collection Building”, 24 (3), pp. 92-95.

[26]. Knievel, J. E., Wicht, H. & Silipigni Connaway, L (2006)., “Use of Circulation Statistics and Interlibrary Loan Data in Collection Management”, College & Research Libraries, 67 (1), pp. 35-49.

[27] Covey, D. T. (2002), “Usage and usability assessment: Library practices and concerns”, Digital Library Federation, CLIR, WashingTon, DC.

[28] Smith, I. M. (1999), “What do we know about public library use?”, Aslib Proceedings, 51 (9), pp. 302-314.

[29] Haigh, S. (2005), “Library Catalogue Users Are Influenced by Trends in Web Searching Search Strategies. Evidence Based Library and Information Practice”, 1 (3), pp. 77-79.

[30] 27. Kumar, S. (2011). Effect of web searching on the OPAC: a comparison of selected university libraries. Library Hi Tech News, 28 (6), pp. 14 – 21.

[31] Anderson, P. (2007), “What is web 2.0? Ideas, technologies and implications for education”, JISC Technology and Standards Watch. Διαθέσιμο στην ιστοσελίδα: http://www.jisc.ac.uk/media/documents/techwatch/tsw0701b.pdf (τελευταία πρόσβαση: 25/05/2015).

[32] Gerolimos, M. & Konsta, R. (2011), “Services for Academic Libraries in New Era. D-Lib Magazine”, 17 (7/8).

[33] Walia, P.K. & Gupta, M. (2012), “Application of web 2.0 Tools by national libraries. Webology”, 9 (2).

[34] Linh, N.C. (2008), “A survey of the application of Web 2.0 in Australian university librar-ies, Library Hi Tech, 26 (4), pp. 630-653.

[35] Mahmood, K. & Richardson, J.V. Jr. (2013), “Impact of Web 2.0 technologies on academic libraries: a survey of ARL libraries”, The Electronic Library, 31 (4), pp. 508-520.

[36] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, DOI 10.17487/RFC2617, Ιούνιος 1999. Διαθέσιμο στην ιστοσελίδα https://www.rfc-editor.org/rfc/pdfrfc/rfc2617.txt.pdf (τελευταία Πρόσβαση: 28/05/2015).

[37] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, Οκτώβριος 2006. Διαθέσιμο στην ιστοσελίδα http://www.rfc-editor.org/info/rfc4648(τελευταία Πρόσβαση: 28/05/2015).

[38] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, Μάρτιος 2014. Διαθέσιμο στην ιστοσελία http://www.rfc-editor.org/info/rfc7159 .

10. Παράρτημα Α: Κώδικας εφαρμογής

Στην ενότητα αυτή καταγράφεται ο κώδικας κάθε module της εφαρμογής καθώς και το αρχείο package.json που δίνει σημαντικές πληροφορίες για την εφαρμογή, όπως ποιο το είναι κύριο αρχείο κώδικα από το οποία ξεκινά η εκτέλεση της εφαρμογής και ποιες είναι οι εξαρτήσεις της από εξωτερικά modules.

ApplicationParameters.js

```
var ApplicationParameters = {};
ApplicationParameters.TwitterCredentials = {
    CONSUMER_KEY: "xvz1evFS4wEEPTGEFPHBog",
    CONSUMER_SECRET: "L8qq9PZyRg6ieKGEKhZolGC0vJWLw8iEJ88DRdyOg"
};

ApplicationParameters.AzureCredentials = {
    ACCOUNT_KEY: "aO6yAhSJ2a94gqetJD8qddiHfZpfd2d7eki4TFJJk3PCuFr4="
};

ApplicationParameters.KOHA_HOST = "www.example.com";
ApplicationParameters.KOHA_HTTP_PORT = 80;
ApplicationParameters.KOHA_SRU_PORT = 9999;

ApplicationParameters.MONGODB_HOST = "www.example.com";
ApplicationParameters.MONGODB_PORT = 27017;
ApplicationParameters.MONGODBCredentials = {
    username: "user",
    password: "pass"
};
ApplicationParameters.MONGODB_SCHEMA = "trends";

ApplicationParameters.SERVER_HOST = "www.example.com";
ApplicationParameters.SERVER_PORT = 20000;
ApplicationParameters.SUGGESTIONS_LIMIT = 10;
ApplicationParameters.SUGGESTIONS_REFRESH_TIMEOUT = 5400000; //milliseconds

ApplicationParameters.DBPEDIA_TIMEOUT = 10000; //  milliseconds (values less
//than 1000 are ignored)

module.exports = ApplicationParameters;
```

bing-spelling-suggestions.js

```
var ApplicationParameters = require("./ApplicationParameters.js");
var https = require('https');

function getSpellingSuggestions(phrase, suggestions_limit, callback) {
    var basic_auth_key = new Buffer(
        ApplicationParameters.AzureCredentials.ACCOUNT_KEY + ":" +
        ApplicationParameters.AzureCredentials.ACCOUNT_KEY
    ).toString("base64");

    var options = {
        hostname: "api.datamarket.azure.com",
        port: 443,
        path: "/Bing/Search/SpellingSuggestions?Query=%27" +
        encodeURIComponent(phrase) + "%27&$format=json&$top=" +
        suggestions_limit,
        method: "POST",
        headers: {
```

```
            "Authorization": "Basic " + basic_auth_key
        }
    };

    var request = https.request(options, function(response) {
        var response_body = "";

        if (response.statusCode != 200) {
           callback(new Error("The status code of the response is not 200(OK). " +
                "[StatusCode:" + response.statusCode + "]"));
        }
        else {
            response.on("data", function(chunk) {
                response_body += chunk;
            });

            response.on("end", function() {
                try {
                    var results = JSON.parse(response_body).d.results;
                    var suggestions = [];
                    var i;
                    for (i = 0; i < results.length; i++) {
                        suggestions[i] = results[i].Value;
                    }
                    callback(null, suggestions);

                }
                catch (error) {
                    callback(error);
                }
            });
        }
    });

    request.on("error", function(error) {
        callback(error);
    });
    request.end();
}

exports.getSpellingSuggestions = getSpellingSuggestions;
```

capitalization-suggestions.js

```
function getCapitalizationSuggestions(phrase, separator) {
    if (!separator) {
        separator = "";
    }

    var tokens = phrase.split(" ");
    var results = [];
    var first_character, i, j, res_size;

    if (tokens.length != 0) {
        first_character = tokens[0].charAt(0);

        if (isNaN(parseInt(first_character))) {
            results.push(first_character.toUpperCase() +
tokens[0].substring(1));
            results.push(first_character.toLowerCase() +
tokens[0].substring(1));
        }
```

```
        else {
            results.push(tokens[0]);
        }

        for (i = 1; i < tokens.length; i++) {
            res_size = results.length;
            first_character = tokens[i].charAt(0);

            for (j = 0; j < res_size; j++) {

                if (isNaN(parseInt(first_character))) {
                    results.push(results[j] + separator +
first_character.toLowerCase() + tokens[i].substring(1));
                    results[j] = results[j] + separator +
first_character.toUpperCase() + tokens[i].substring(1);
                }
                else {
                    results[j] = results[j] + separator + tokens[i];
                }
            }
        }
    }

    return results;
}

exports.getCapitalizationSuggestions = getCapitalizationSuggestions;
```

dbpedia-suggestion.js

```
var dbpedia = require("./dbpedia-sparql-endpoint.js");

function getCategories(phrase, callback) {
    var query =
        "prefix dc: <http://purl.org/dc/terms/>" +
        "select distinct ?category where {" +
        "{<http://dbpedia.org/resource/" + phrase + "> dc:subject ?category
.}} LIMIT 1000";

    dbpedia.sparqlQuery(query, function(error, results) {
        var categories = [];
        var i;
        if (error) {
            callback(error);
        }
        else {
            for (i = 0; i < results.length; i++) {
                categories[i] =
                    results[i].category.value.substring(
                        "http://dbpedia.org/resource/Category:".length
                    ).replace(/_/g, " ");
            }
            callback(null, categories);
        }
    });
}

function getRelatedTerms(phrase, callback) {
    var query =
        "prefix dc: <http://purl.org/dc/terms/>" +
        "SELECT distinct ?name WHERE { ?categories rdfs:label '" + phrase +
"'@en ." +
```

```
        "?relatedterms dc:subject ?categories ." +
        "?relatedterms rdfs:label ?name ." +
        "FILTER(langMatches(lang(?name), 'EN'))} LIMIT 1000";

    dbpedia.sparqlQuery(query, function(error, results) {
        var relatedterms = [];
        var i;
        if (error) {
            callback(error);
        }
        else {
            for (i = 0; i < results.length; i++) {
                relatedterms[i] = results[i].name.value;
            }
            callback(null, relatedterms);
        }
    });
}

module.exports.getCategories = getCategories;
module.exports.getRelatedTerms = getRelatedTerms;
```

koha-library-queries.js

```
var ApplicationParameters = require("./ApplicationParameters.js");
var http = require('http');

function executeSRURequest(encoded_searchpart, callback) {
    var options = {
        hostname: ApplicationParameters.KOHA_HOST,
        port: ApplicationParameters.KOHA_SRU_PORT,
        path: "/biblios?" + encoded_searchpart,
        method: "GET"
    };

    var request = http.request(options, function(response) {
        var response_body = "";

        if (response.statusCode != 200) {
            callback(new Error("The status code of the response is not
200(OK). " +
                "[StatusCode:" + response.statusCode + "]"));
        }
        else {
            response.on("data", function(chunk) {
                response_body += chunk;
            });

            response.on("end", function() {
                callback(null, response_body);
            });
        }
    });

    request.on("error", function(error) {
        callback(error);
    });
    request.end();
}

function searchForBooks(keyword, keyword_types, callback) {
```

```
    var sru_req_base_searchpart =
"version=1.1&operation=searchRetrieve&query=";
    var book_list_url = "http://" + ApplicationParameters.KOHA_HOST +
        ":" + ApplicationParameters.KOHA_HTTP_PORT +
        "/cgi-bin/koha/opac-search.pl?";

    var sru_req_searchpart, book_list_url_suffix, i, encoded_keyword;
    var search_canceled = false;
    var author_flag = false,
        subject_flag = false,
        title_flag = false;
    var async_search_counter = keyword_types.length;
    var useful_keyword_types = [];

    var invalidKeywordTypeCombination = function() {
        search_canceled = true;
        callback(new Error("Invalid keyword type combination"));
    };

    for (i = 0; i < keyword_types.length; i++) {
        encoded_keyword = keyword;

        if (keyword.indexOf("\"") != 0 || keyword.lastIndexOf("\"") !=
keyword.length - 1) {
            encoded_keyword = "\"" + keyword.split(" ").join("\" \"") + "\"";
        }

        encoded_keyword = encodeURIComponent(encoded_keyword);

        switch (keyword_types[i].toLowerCase()) {
            case "author":
                if (author_flag) {
                    invalidKeywordTypeCombination();
                    return;
                }
                sru_req_searchpart = sru_req_base_searchpart + "Author%3D" +
encoded_keyword;
                book_list_url_suffix = "idx=au&q=" +
encodeURIComponent(keyword);
                author_flag = true;
                break;
            case "subject":
                if (subject_flag) {
                    invalidKeywordTypeCombination();
                    return;
                }
                sru_req_searchpart = sru_req_base_searchpart + "Subject%3D" +
encoded_keyword;
                book_list_url_suffix = "idx=su&q=" +
encodeURIComponent(keyword);
                subject_flag = true;
                break;
            case "title":
                if (title_flag) {
                    invalidKeywordTypeCombination();
                    return;
                }
                sru_req_searchpart = sru_req_base_searchpart + "Title%3D" +
encoded_keyword;
                book_list_url_suffix = "idx=ti&q=" +
encodeURIComponent(keyword);
                title_flag = true;
                break;
```

```
            default:
                invalidKeywordTypeCombination();
                return;
        }

        (function() {
            var keyword_type = keyword_types[i];
            var book_list_url_suffix_ = book_list_url_suffix;
            var start_tag, end_tag, number_of_records;
            executeSRURequest(sru_req_searchpart, function(error,
sru_response) {
                if (!search_canceled) {

                    if (error) {
                        search_canceled = true;
                        callback(error);
                    }
                    else {
                        start_tag = "<zs:numberOfRecords>";
                        end_tag = "</zs:numberOfRecords>";
                        number_of_records =
parseInt(sru_response.substring(sru_response.indexOf(start_tag) +
start_tag.length, sru_response.indexOf(end_tag)));

                        if (isNaN(number_of_records)) {
                            search_canceled = true;
                            callback(new Error("Unexpected SRU Response"));
                        }
                        else {
                            if (number_of_records > 0) {

                                if (useful_keyword_types.length > 0) {
                                    book_list_url = book_list_url + "&op=or&"
+ book_list_url_suffix_;
                                }
                                else {
                                    book_list_url = book_list_url +
book_list_url_suffix_;
                                }

                                useful_keyword_types.push(keyword_type);
                            }

                            if (--async_search_counter == 0) {
                                callback(null, {
                                    "keyword": keyword,
                                    "keyword_types": useful_keyword_types,
                                    "url": (useful_keyword_types.length == 0 ?
"" : book_list_url)
                                });
                            }
                        }
                    }
                }
            });
        })();
    }
}
exports.searchForBooks = searchForBooks;
```

main.js

```
var ApplicationParameters = require("./ApplicationParameters.js");
var bing_ss = require("./bing-spelling-suggestions.js");
var capitalization_cases = require("./capitalization-suggestions.js");
var dbpedia_suggestions = require("./dbpedia-suggestions.js");
var koha_library_queries = require("./koha-library-queries.js");
var twitter_search = require("./twitter-search.js");
var search_target_en = "programming";
var suggestions_db = require("./suggestions-db.js");

function logError(error) {
    console.error(error.stack);
}

function makeBookSuggestions() {

    var makeSuggestionsBasedOnThisTrend = function(trend) {

        var storeSuggestion = function(error, suggestion) {
            if (error) {
                logError(error);
            }
            else {
                if (suggestion.url != "") {
                    suggestion.hashtag = trend;
                    suggestion.timestamp = new Date();
                    suggestions_db.addSuggestions([suggestion],
function(error) {
                        if (error) {
                            logError(error);
                        }
                    });
                }
            }
        };

        var capitalization_cases =
capitalization_cases.getCapitalizationSuggestions(trend, '_');
        var i, j;

        for (i = 0; i < capitalization_cases.length; i++) {
            dbpedia_suggestions.getCategories(
                capitalization_cases[i],
                function(error, categories) {
                    if (error) {
                        logError(error);
                    }
                    else {
                        for (j = 0; j < categories.length; j++) {

koha_library_queries.searchForBooks(search_target_en + " " + categories[j],
["title", "subject"], storeSuggestion);
                        }
                    }
                });

            dbpedia_suggestions.getRelatedTerms(
                capitalization_cases[i],
                function(error, related_terms) {

                    if (error) {
                        logError(error);
```

```
                    }
                    else {
                        for (j = 0; j < related_terms.length; j++) {

koha_library_queries.searchForBooks(search_target_en + " " + related_terms[j],
["title", "subject"], storeSuggestion);
                        }
                    }
                });
        }

        koha_library_queries.searchForBooks(search_target_en + " " + trend,
["title", "subject"], storeSuggestion);

    };

    var since_id = 0;
    twitter_search.search({
        q: search_target_en,
        count: 100,
        lang: "en",
        result_type: "mixed",
        "since_id": since_id
    }, function(error, response) {

        var i, j, tweet, hashtags;

        for (i = 0; i < response.statuses.length; i++) {
            tweet = response.statuses[i];

            if (!tweet.entities) {
                continue;
            }

            hashtags = tweet.entities.hashtags;

            for (j = 0; j < hashtags.length; j++) {
                (function() {
                    var hashtag = hashtags[j].text;
                    bing_ss.getSpellingSuggestions(hashtag, 1, function(error,
suggestions) {
                        if (error) {
                            logError(error);
                        }
                        else if (suggestions.length != 0) {
                            makeSuggestionsBasedOnThisTrend(suggestions[0]);
                        }
                        else {
                            makeSuggestionsBasedOnThisTrend(hashtag);
                        }
                    });
                })();
            }
        }

        since_id = response.search_metadata.max_id_str;
    });
}

(function main() {
    require("child_process").fork("./server.js");
    makeBookSuggestions();
    setInterval(function() {
```

```
        makeBookSuggestions();
    }, ApplicationParameters.SUGGESTIONS_REFRESH_TIMEOUT);
})();
```

twitter-search.js

```
var ApplicationParameters = require("./ApplicationParameters.js");
var bearer_token = null;
var twitter_oauth_module;

function search(params, callback) {
    var param_keys = Object.keys(params);
    var i;

    if (bearer_token == null) {
        twitter_oauth_module = require("./twitter-apponly-oauth.js");
        twitter_oauth_module.getBearerToken(
            ApplicationParameters.TwitterCredentials.CONSUMER_KEY,
            ApplicationParameters.TwitterCredentials.CONSUMER_SECRET,
            function(error, btoken) {
                if (error) {
                    callback(error);
                }
                else {
                    bearer_token = btoken;
                    search(params, callback);
                }
            });
    }
    else {
        var https = require("https");
        var options = {
            hostname: "api.twitter.com",
            port: 443,
            method: "GET",
            headers: {
                "Authorization": "Bearer " + bearer_token,
                "Connection": "close"
            }
        };

        options.path = "/1.1/search/tweets.json?";

        for (i = 0; i < param_keys.length; i++) {
            options.path += param_keys[i] + "=" +
encodeURIComponent(params[param_keys[i]]) + "&";
        }

        options.path = options.path.substring(0, options.path.length - 1);

        var request = https.request(options, function(response) {
            var response_body = "";
            if (response.statusCode != 200) {
                callback(new Error("The status code of the response is not
200(OK). " +
                    "[StatusCode:" + response.statusCode + "]"));
            }
            else {
                response.on("data", function(chunk) {
                    response_body += chunk;
                });
```

```
                response.on("end", function() {
                    try {
                        callback(null, JSON.parse(response_body));
                    }
                    catch (error) {
                        callback(error);
                    }
                });
            }
        });

        request.on("error", function(error) {
            callback(error);
        });

        request.end();
    }
}

exports.search = search;
```

server.js

```
var ApplicationParameters = require("./ApplicationParameters.js");
var suggestions_db = require("./suggestions-db.js");
var http = require("http");

process.on("disconnect", function() {
    process.exit(0);
});

http.createServer(function(request, response) {
    if (request.method === "GET" || request.method === "POST") {

        suggestions_db.getSuggestions(ApplicationParameters.SUGGESTIONS_LIMIT,
function(error, suggestions) {
            if (error) {
                response.writeHead(500);
            }
            else {
                response.writeHead(200, {
                    "Content-Type": "application/json",
                    "Access-Control-Allow-Origin": "*"
                });

                response.write(JSON.stringify({
                    "suggestions": suggestions
                }));
            }
            response.end();
        });

    }
    else if (request.method === "HEAD") {
        response.writeHead(200);
        response.end();
    }
    else {
        response.writeHead(405);
        response.end();
    }
```

```
}).listen(ApplicationParameters.SERVER_PORT,
ApplicationParameters.SERVER_HOST);
```

dbpedia-sparql-endpoint.js

```
var ApplicationParameters = require("./ApplicationParameters.js");
var http = require("http");
var queue = [];

function processNextRequest() {
    var requestHandler;
    if (queue.length != 0) {
        requestHandler = queue[0];
        setTimeout(function() {
            requestHandler();
        }, 50);
    }
}

function sparqlQuery(query, callback) {

    queue.push(function() {

        var options = {
            hostname: "dbpedia.org",
            port: 80,
            path: "/sparql?default-graph-uri=http%3A%2F%2Fdbpedia.org&query="
+ encodeURI(query) +
            "&format=application%2Fsparql-results%2Bjson&timeout=" +
ApplicationParameters.DBPEDIA_TIMEOUT,

            method: "GET",
            headers: {
                "Connection: ": "Keep-Alive"
            }
        };

        var query_finish = function() {
            queue.shift();
            processNextRequest();
        };

        var request = http.request(options, function(response) {
            var response_body = "";

            if (response.statusCode != 200) {
                query_finish();
                callback(new Error("The status code of the response is not
200(OK). " +
                    "[StatusCode:" + response.statusCode + "]" + " " +
query));
            }
            else {
                response.on("data", function(chunk) {
                    response_body += chunk;
                });

                response.on("end", function() {
                    query_finish();
                    try {
                        var json_results =
JSON.parse(response_body).results.bindings;
```

```
                        callback(null, json_results);
                    }
                    catch (error) {
                        callback(error);
                    }
                });
            }
        });

        request.on("error", function(error) {
            query_finish();
            callback(error);
        });
        request.end();
    });

    if (queue.length == 1) processNextRequest();
}

exports.sparqlQuery = sparqlQuery;
```

suggestions-db.js

```
var ApplicationParameters = require("./ApplicationParameters.js");
var mongo_client = require('mongodb').MongoClient;
var mongodb_connection_url = "mongodb://" +
    ApplicationParameters.MONGODBCredentials.username + ":" +
    ApplicationParameters.MONGODBCredentials.password + "@" +
    ApplicationParameters.MONGODB_HOST + ":" +
    ApplicationParameters.MONGODB_PORT + "/" +
    ApplicationParameters.MONGODB_SCHEMA + "?autoReconnect=true";

var connection_options = {
    "db": {
        bufferMaxEntries: 0
    }
};

var db = null;

function addSuggestions(suggestions, callback) {
    var collection;
    if (!db) {
        mongo_client.connect(mongodb_connection_url, connection_options,
            function(error, database) {
                if (error) {
                    callback(error);
                }
                else {
                    db = database;
                    addSuggestions(suggestions, callback);
                }
            });
    }
    else {
        collection = db.collection("AssetSuggestions");
        collection.insert(suggestions, function(error, result) {
            callback(error);
        });
    }
}
```

```
function getSuggestions(limit, callback) {
    var collection;
    if (!db) {
        mongo_client.connect(mongodb_connection_url, connection_options,
            function(error, database) {
                if (error) {
                    callback(error);
                }
                else {
                    db = database;
                    getSuggestions(limit, callback);
                }
            });
    }
    else {
        collection = db.collection("AssetSuggestions");
        collection.aggregate([{
            $sort: {
                "timestamp": 1
            }
        }, {
            $group: {
                _id: {
                    keyword: "$keyword"
                },
                doc: {
                    $first: {
                        keyword: "$keyword",
                        keyword_types: "$keyword_types",
                        url: "$url",
                        hashtag: "$hashtag"
                    }
                }
            }

        }, {
            $limit: limit
        }, {
            $project: {
                _id: 0,
                "keyword": "$doc.keyword",
                "keyword_types": "$doc.keyword_types",
                "url": "$doc.url",
                "hashtag": "$doc.hashtag"
            }
        }], function(error, documents) {
            if (error) {
                callback(error);
            }
            else {
                callback(null, documents);
            }
        });
    }
}

module.exports.addSuggestions = addSuggestions;
module.exports.getSuggestions = getSuggestions;
```

twitter-apponly-oauth.js

```
function getBearerToken(consumer_key, consumer_secret, callback) {

    var bearer_token_credentials =
        new Buffer(encodeURIComponent(consumer_key) + ":" +
            encodeURIComponent(consumer_secret)).toString("base64");

    var https = require("https");
    var options = {
        hostname: "api.twitter.com",
        port: 443,
        path: "/oauth2/token",
        method: "POST",
        headers: {
            "Authorization": "Basic " + bearer_token_credentials,
            "Content-Type": "application/x-www-form-
urlencoded;charset=UTF-8",
            "Connection": "close"
        }
    };

    var request = https.request(options, function(response) {
        var response_body = "";

        if (response.statusCode != 200) {
            callback(new Error("The status code of the response is not
200(OK). " +
                "[StatusCode:" + response.statusCode + "]"));
        }
        else {
            response.on("data", function(chunk) {
                response_body += chunk;
            });

            response.on("end", function() {
                try {
                    var token_type =
JSON.parse(response_body).token_type;
                    if (token_type != "bearer") {
                        throw new Error("Token type is not set to
bearer. " +
                            "[TokenType:" + token_type + "]");
                    }

                    var bearer_token =
JSON.parse(response_body).access_token;
                    callback(null, bearer_token);
                }
                catch (error) {
                    callback(error);
                }
            });
        }
    });

    request.on("error", function(error) {
```

```
        callback(error);
    });

    request.write("grant_type=client_credentials");
    request.end();
}

module.exports.getBearerToken = getBearerToken;
```

package.json

```
{
    "name": "koha_hashtags",
    "version": "1.0.1",
    "description": "A service capable of integrating popular Twitter
hashtags with the library's OPAC",
    "main": "main.js",
    "author": "AposTolos Karalis <akaralis@hotmail.com>",
    "dependencies": {
    "mongodb": ">=2.0.16"
     }
}
```