ΤΕΙ ΠΕΙΡΑΙΑ | ΤΜΗΜΑ ΗΥΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Έξυπνη εξαγωγή δεδομένων από το διαδίκτυο

[Περίπτωση μεταπτυχιακών προγραμμάτων]

ΑΘΗΝΑ, ΙΑΝΟΥΑΡΙΟΣ 2014

σπουδαστής | ΚΟΒΑΤΣΗΣ ΠΕΤΡΟΣ
επιβλέπων | ΠΡΕΖΕΡΑΚΟΣ ΓΕΩΡΓΙΟΣ

Περιεχόμενα

1. Web Crawling & Μηχανές Αναζήτησης

1.1 Ψάχνοντας πληροφορίες στον Παγκόσμιο Ιστό

Σε έναν κόσμο όπου όλα κινούνται γρήγορα, που ο χρόνος σε συνδυασμό με τη γνώση αποτελεί σημαντικό παράγοντα της απόδοσης, η γρήγορη και στοχευμένη πληροφορία αποτελεί σημείο αναφοράς για την σωστή και ομαλή λειτουργία της καθημερινότητας. Η ανάγκη λοιπόν για προβολή συσσωρευμένης πληροφορίας με τρόπο ώστε να αναδεικνύονται τα σημαντικά τμήματα αυτής και να είναι ταχύτερη η επεξεργασία από τους ενδιαφερόμενους, είναι και το κλειδί για την κάλυψη του κόστους του χρόνου.

Ως απόφοιτοι τριτοβάθμιας εκπαίδευσης, η συνέχεια της μόρφωσης, καθώς αυτή αποτελεί προσωπική ευθύνη και επιλογή του καθενός, δεν θα μπορούσε να μην περάσει από τις πύλες του διαδικτύου, που στις σελίδες του υπάρχει τεράστιος όγκος δεδομένων όπου μπορεί να προσφέρει την πληροφορία και την γνώση που επιζητούμε. Η επιλογή λοιπόν του μεταπτυχιακού προγράμματος, αποτελεί διαδικασία που περνάει από πολλά φίλτρα πριν ο τελικός ενδιαφερόμενος αποφασίσει. Η επιλογή του κλάδου, το πλήθος και η κάλυψη της επιστημονικής γνώσης, η τοποθεσία του Πανεπιστημιακού ιδρύματος, ο χρόνος φοίτησης, το κόστος, σχόλια τρίτων κλπ, αποτελούν κάποια από τα φίλτρα αυτά.

Η λήψη όμως ακανόνιστης συσσωρευμένης πληροφορίας, αποτελεί παγίδα καθώς αυξάνει το χρόνο επεξεργασίας από τους ενδιαφερομένους, μια επεξεργασία όπου θα μπορούσε να γίνει σε επίπεδο μηχανής γλιτώνοντας χρόνο καθώς απελευθερώνει άλλους πόρους που μπορούν να χρησιμοποιηθούν για την σύγκριση και λήψη λοιπών αποφάσεων. Είναι σημαντικό λοιπόν να αναζητούμε δεδομένα στο διαδίκτυο, ιδίως όταν αυτά έχουν να κάνουν με θέματα επιμόρφωσης που στο μέλλον θα δημιουργήσουν κατάλληλους επιστήμονες που ίσως αναπλάσουν ή εξελίξουν αυτά που ακόμα και σήμερα θεωρούμε δεδομένα και αμετάβλητα.

1.2 Μηχανές αναζήτησης

1.2.1 Ιστορική αναδρομή

Ύστερα από την πρώτη σύλληψη της ιδέας του Διαδικτύου, από τον Αμερικανικό στρατό ως τρόπος ασφαλής και σταθερής επικοινωνίας και την υλοποίηση του από τον **Timothy John Berners-Lee** για το ερευνητικό ίδρυμα **Cern**, έως σήμερα, ο

γνωστός σε όλους **«Παγκόσμιος Ιστός»** ή **Internet,** έχει εξελιχθεί σε ένα σύστημα όχι μόνο διηπειρωτικής επικοινωνίας, αλλά και παγκόσμιας «αποθήκης» δεδομένων.

Από το 1989 με τη χρήση του Διαδικτύου από το Cern για την παρουσίαση αποτελεσμάτων σε πολλούς χρήστες, καθώς η αποστολή μηνυμάτων ηλεκτρονικού ταχυδρομείου, που αποτελούσε τότε τρόπο επικοινωνίας και ενημέρωσης αρχικά των επιστημόνων και των ενδιαφερόμενων του ιδρύματος, αποδείχτηκε αποτελεσματικότερος και απλούστερος. Όπως ήταν λογικό όμως, ο όγκος των δεδομένων άρχισε να αυξάνει με ταχύς ρυθμούς.

Η αύξηση του όγκου των δεδομένων, σήμαινε ταυτόχρονα και αύξηση του χρόνου αναζήτησης των πληροφοριών που αναζητούσε ο χρήστης. Η ιδέα, όπως φάνηκε από νωρίς, ήταν απλή. Θα έπρεπε ο όγκος των δεδομένων αυτών να είναι εύκολα προσβάσιμος και στοχευμένος για τους ενδιαφερόμενους. Άλλωστε, η «πολύ» πληροφορία χωρίς να είναι «σωστή» πληροφορία, παύει να είναι πληροφορία.

- *Μια απλή έκφραση και λύση του προβλήματος*

Για κάθε πρόβλημα όμως, συνήθως υπάρχει και η αντίστοιχη λύση. Η έννοια της αναζήτησης τότε δεν ήταν ξένη, αλλά όχι και απόλυτα ξεκάθαρη. Όταν ψάχνουμε να βρούμε έναν τηλεφωνικό αριθμό, συνήθως χρησιμοποιούμε τον τηλεφωνικό μας κατάλογο, καθώς είναι αδύνατο να θυμόμαστε όλους τους αριθμούς από μνήμη. Ο τηλεφωνικός κατάλογος είναι αντίστοιχα η λειτουργία που υλοποιεί μια μηχανή αναζήτησης στο διαδίκτυο. Ο τηλεφωνικός κατάλογος έχει δική του δομή όπου καταγράφονται τα τηλέφωνα και μπορούμε ταχύτερα να βρούμε το νούμερο που επιζητούμε. Έτσι και μια μηχανή αναζήτησης, έχει «καταγράψει» τα δεδομένα για τις σελίδες του διαδικτύου, οπότε μπορεί γρήγορα να βρει την σελίδα ή την πληροφορία που ψάχνουμε. Επίσης στους τηλεφωνικούς καταλόγους, συνήθως, έχει ανά σελίδα και τα γράμματα της αλφαβήτου, οπότε οι διαδικασίες εύρεσης αποτελέσματος μειώνουν τον χρόνο αναζήτησης. Κάτι αντίστοιχο με αυτή τη λειτουργία, υλοποιεί και μια μηχανή αναζήτησης βελτιώνοντας τους χρόνους απόκρισης των αποτελεσμάτων.

Η έννοια και λειτουργία της μηχανής αναζήτησης θα αποτελούσε σίγουρα αναπόσπαστο κομμάτι της λειτουργίας του διαδικτύου γι αυτό και αρκετά νωρίς έγινε και η παρουσίαση της πρώτης μηχανής αναζήτησης. Ενδεικτικά παρακάτω αναφέρονται κάποιες μηχανές αναζήτησης που αποτέλεσαν τον πρόδρομο για τις σημερινές σύγχρονες μηχανές.

❖ Archie

Στα τέλη του 1990 έγινε η παρουσίαση της πρώτης μηχανής αναζήτησης από τους Peter Deutsch, Alan Emtage και Bill Heelan. Ήταν ουσιαστικά ένα σύστημα καταγραφής διακομιστών FTP και των αρχείων που περιλάμβαναν.

Εικόνα 1. Archie, μηχανή αναζήτησης

❖ Veronica

Το 1992 οι Steven Foster και Fred Barrie από το πανεπιστήμιο της Νεβάδα, παρουσιάζουν την μηχανή αναζήτησης Veronica (Very Easy Rodent-Oriented Net-wide Index to Computer Archives) βασισμένη στο πρωτόκολλο Gopher σχεδιασμένο για αναζήτηση, αποστολή και μετάδοση αρχείων μέσω του διαδικτύου.

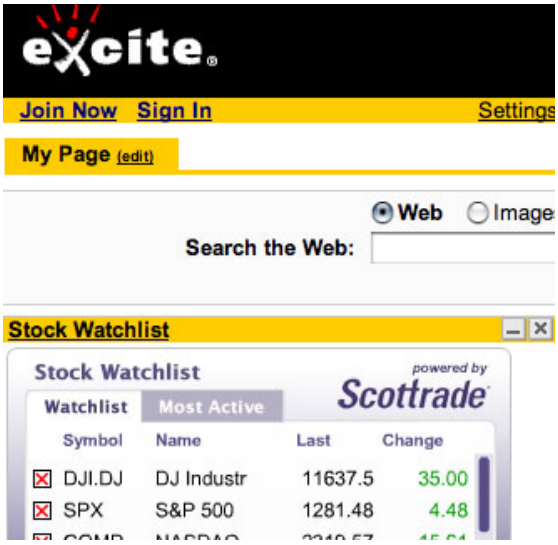
Εικόνα 2. Veronica, μηχανή αναζήτησης

❖ Jughead

Το 1993 παρουσιάζεται το Jughead από τον Rhett Jones, πανεπιστήμιο της Γιούτα. Χρησιμοποιεί επίσης το πρωτόκολλο Gopher με την διαφορά ότι κάνει αναζήτηση δεδομένων στον διακομιστή σε πραγματικό χρόνο.

❖ Excite

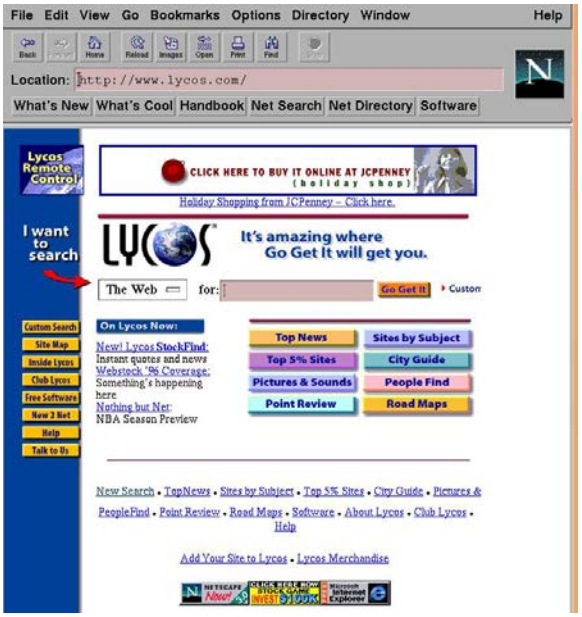
Τέλη του 1995 και φαίνεται ότι η μηχανή αναζήτησης στο διαδίκτυο δεν αποτελεί απλά μια μηχανή με συγκεκριμένη λειτουργία. Το Excite, δημιούργημα φοιτητών από το πανεπιστήμιο του Στάνφορντ, αποτελεί σημείο ορισμού για τις μοντέρνες μηχανές. Αποτελεί μία διαδικτυακή πύλη με μηχανή αναζήτησης, προσφέροντας στους χρήστες λειτουργίες ενημέρωσης ειδήσεων, καιρού, ηλεκτρονικό ταχυδρομείο, IM, προσωπική σελίδα χρήστη κλπ. Ίσως ένα προάγγελος της Google.



Εικόνα 3. Excite, μηχανή αναζήτησης

❖ Lycos

Το 1994 παρουσιάζεται το Lycos. Η μηχανή αναζήτησης προσέφερε λειτουργίες ηλεκτρονικού ταχυδρομείου, κοινωνικής δικτύωσης και σελίδες διακσέδασης.



Εικόνα 4. Lycos, μηχανή αναζήτησης

- Altavista

Το 1996 παρουσιάζεται η μηχανή Altavista όπου αποτέλεσε και βασικό πάροχο αναζήτησης της Yahoo.

Εικόνα 5. Altavista, μηχανή αναζήτησης

1.2.2 Γνωστές σύγχρονες μηχανές αναζήτησης

Από τους προδρόμους των μηχανών αναζήτησης έως σήμερα, το διαδίκτυο αποτελεί αναπόσπαστο κομμάτι της καθημερινής ζωής και ο βασικός του συνοδοιπόρος οι μηχανές αναζήτησης. Το 1998 δύο φοιτητές οι Larry Page και Sergey Brin του

πανεπιστημίου Στάνφορντ, παρουσίασαν την καινούργια μηχανή αναζήτησης που σήμερα κατέχει το μεγαλύτερο ποσοστό αναζήτησης στο διαδίκτυο, την Google. Οι δύο φοιτητές εφάρμοσαν στην αναζήτηση τους ένα νέο σύστημα αξιολόγησης σελίδων, όπου σύντομα θα αποτελούσε εμφανή καινοτομία στην ανάπτυξη αντίστοιχων εφαρμογών καθώς και εξέλιξης του ίδιου του διαδικτύου. Μάλιστα, στη δική μας εποχή, το διαδίκτυο και οι ανάπτυξη των εφαρμογών του, είναι πλέον αυτό που παραμετροποιείται ώστε να λειτουργήσει αρμονικά με τις διαθέσιμες μηχανές αναζήτησης.

Πώς άλλωστε θα μπορούσε να γίνει αλλιώς καθώς το διαδίκτυο αποτελεί πλέον χώρο πληροφόρησης, εργασίας, διασκέδασης και πολλά άλλα! Παρακάτω παρουσιάζονται σύγχρονες μηχανές αναζήτησης που χρησιμοποιούνται στον κόσμο από εκατομμύρια χρήστες.

- Google

Το 1998 οι δύο φοιτητές από το Στάνφορντ παρουσιάζουν το Google. Η καινούργια αυτή μηχανή, βασισμένη σε καινοτόμο σύστημα αξιολόγησης των σελίδων σύμφωνα με το περιεχόμενο τους, θα αποτελέσει την διασημότερη μηχανή αναζήτησης μέχρι και σήμερα. Αρχικά βασισμένη σε αλγορίθμους consumer-producer κι έπειτα mapreduce, εξασφαλίζει όχι μόνο την απόδοση και την σωστή αξιολόγηση των διαδικτυακών σελίδων, αλλά και ταχύτητα ανανέωσης των περιεχομένων της καθώς και της απόκρισης αποτελεσμάτων. Επιπλέον, εισάγει στο διαδίκτυο την έννοια του SEO(Search engine optimization), που αναφέρεται σε βέλτιστες τεχνικές ανάπτυξης της σελίδας για την υποβοήθηση της επεξεργασίας της, από την μηχανή αναζήτησης.

- Yahoo

Το 1995 παρουσιάζεται η μηχανή Yahoo(Yet Another Hierarchical Officious Oracle). Αρχικά υλοποιημένη το 1994 από τους φοιτητές του Στάνφορντ, Γιανγκ και Φίλο με ονομασία «Jerry's guide to the World Wide Web», αποτελούσε μηχανή για την ιεραρχική προβολή των διαδικτυακών σελίδων. Το 2000 η Yahoo ξεκινάει να χρησιμοποιεί την Google για την αναζήτηση στον Ιστό, ενώ τέσσερα χρόνια αργότερα εισάγει την δική της μηχανή αναζήτησης.

- Bing

Η μηχανή αναζήτησης Bing παρουσιάστηκε από την Microsoft το 2009 καθώς ανακοινώνουν με την Yahoo την κοινή τους συνεργασία. Η μηχανή Bing αντικαθιστά την μηχανή της Yahoo αφού πρώτα έχει γίνει γνώστη στον διαδικτυακό χώρο ως **Live Search**(2007), **Windows Live Search**(2006), και **MSN Search**(1998). Βασικός

πλέον ανταγωνιστής της Google, η μηχανή Bing ανακοινώνει την καινούργια υποδομή αναζήτησης και τεχνολογία αρχειοθέτησης το 2011.

- Yandex

Η μηχανή αναζήτησης Yandex παρουσιάζεται το 2010 από την ομώνυμη Ρώσικη εταιρεία και κατέχει το μεγαλύτερο ποσοστό αναζήτησης χρηστών στην Ρωσία.

- Baidu

Το 2000 η ιδρύεται η εταιρία Baidu με έδρα την Κίνα από τους Ρόμπιν Λι και Έρικ Χιου. Η ομώνυμη μηχανή αναζήτησης το 2010 καταλαμβάνει τις μισές αναζητήσεις στο διαδίκτυο στην Κίνα ενώ από το 2011 κρατάει σταθερά την πρώτη θέση στην αντίστοιχη χώρα με το μεγαλύτερο μέρισμα του ποσοστού.

- Duck Duck Go

Τον Ιούλιο του 2010 παρουσιάζεται η ιδέα της μηχανής DuckDuckGo αρχικά σαν κοινότητα όπου οι χρήστες παρουσίαζαν τα προβλήματα τους σχετικά με μηχανές αναζήτησης, επιπλέον λειτουργίες κ.α. Η μηχανή αναζήτησης χρησιμοποιεί πήγες από άλλες μηχανές αναζήτησης όπως Yahoo, Bing, Yandex και WolframAlpha καθώς και δικά της δεδομένα τα οποία παρουσιάζονται στους τελικούς χρήστες.

1.3 Τρόπος λειτουργίας των μηχανών αναζήτησης

Παρόλο το μεγάλο πλήθος των μηχανών αναζήτησης, μιας και αυξάνονται πλέον όχι μόνο για γενικούς αλλά και ειδικούς σκοπούς, η βασική αρχή της λειτουργίας τους παραμένει ίδια. Μια μηχανή αναζήτησης βασίζει την ταχύτητά απόκρισης της αλλά και την ορθότητα των αποτελεσμάτων της στα ήδη προ αποθηκευμένα δεδομένα του διαδικτύου. Άλλωστε, η αναζήτηση σε πραγματικό χρόνο των δεδομένων θα καθιστούσε ανυπόφορη την λειτουργία τους.

Πέραν λοιπόν των διαφορετικών αλγορίθμων για την εύρεση, εξαγωγή και αποθήκευση των διαδικτυακών σελίδων, μπορούμε να επικεντρωθούμε σε τρία βασικά στάδια λειτουργίας για την α. περισυλλογή τους(Crawling), β. επεξεργασία και αποθήκευση(Indexing) και γ. Προβολή των αποτελεσμάτων.

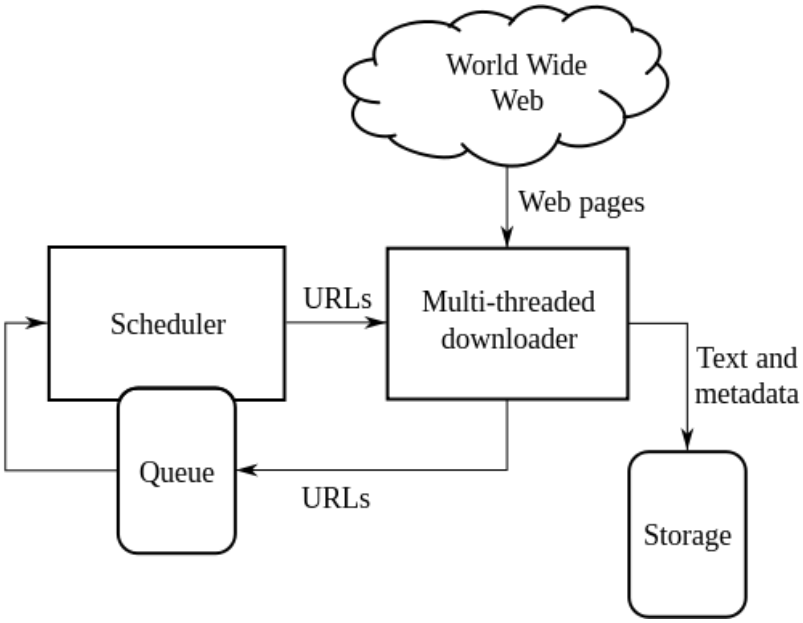
1.3.1 Web Crawling

Ο Web Crawler ή Web Spider αναφέρεται στην διαδικασία περισυλλογής των δεδομένων. Αποτελεί στην πραγματικότητα μια εφαρμογή, ή αλλιώς όπως είναι γνωστό «Internet bot», το οποίο ακολουθώντας εσωτερικές διαδικασίες επισκέπτεται διαδικτυακές διευθύνσεις ανά τακτικό ή μη, χρονικό διάστημα, για την εξαγωγή δεδομένων.

Η αρχή λειτουργίας ενός Web Crawler, βασίζεται στην αρχική τροφοδοσία του με διαδικτυακές διευθύνσεις. Με μεθόδους κλήσης HTTP GET, μπορεί να ζητήσει από τον εκάστοτε εξυπηρετητή τον HTML κώδικα, από τον οποίο μπορεί να εξάγει περεταίρω διευθύνσεις που είτε αφορούν την ίδια διαδικτυακή εφαρμογή είτε είναι διευθύνσεις που παραπέμπουν σε άλλες εφαρμογές. Το σημαντικό πρόβλημα σε αυτή την λειτουργία, είναι ο κίνδυνος ο Crawler να πέσει σε ατέρμονα βρόγχο και να εγκλωβιστεί μέσω των ανακατευθύνσεων στην ίδια εφαρμογή, ζητώντας από τον εξυπηρετητή τις ίδιες σελίδες. Για την επίλυση αυτού του προβλήματος, χρησιμοποιούνται τεχνικές κανονικοποίησης των κλήσεων στις διευθύνσεις. Τέτοιες τεχνικές περιλαμβάνουν την απαλοιφή σημείων στίξης, την μετατροπή των γραμμάτων σε κεφαλαία ή μόνο πεζά, δομή της διεύθυνσης με συμπλήρωση των πεδίων όταν αυτά λείπουν, π.χ. το http://example.gr/ θα μετατραπεί σε http://www.example.gr. Έτσι λοιπόν εξασφαλίζουμε ότι ο Crawler δεν θα εγκλωβιστεί στη ίδια εφαρμογή και δεν θα ζητήσει την ίδια σελίδα από τον εξυπηρετητή.

- *Η Αρχιτεκτονική*

Η σωστή λειτουργία ενός Web Cawler βασίζεται τόσο στην απόδοση για την σωστή ζήτηση των σελίδων από τον εξυπηρετητή όσο και στην ταχύτητα επεξεργασίας των δεδομένων αυτών. Είναι άλλωστε γνωστή στρατηγική, η κάθε εταιρία να διατηρεί μέρος ή και ολόκληρο τον αλγόριθμο λειτουργίας κρυφό ώστε να αποφθεχθούν περιπτώσεις αντιγραφής. Για την επιτυχία στην ταχύτητα ζήτησης και επεξεργασίας δεδομένων, συχνά χρησιμοποιείται παραλληλισμός ως προς την λειτουργία του ρομπότ. Όπως παρουσιάζεται στην παρακάτω εικόνα η λειτουργία έχει 3 βήματα, πρώτον το ρομπότ ζητάει μια σελίδα από τον εξυπηρετητή, δεύτερον περνάει από επεξεργασία και τυχόν καινούργιες διευθύνσεις μπαίνουν στην ουρά και τέλος επιλέγει προσεκτικά την επόμενη κλήση στον εξυπηρετητή.



Εικόνα 6. Βασική λειτουργία Web Crawler

Για την αποδοτικότερη λειτουργία του, ο Web Crawler θα πρέπει να λειτουργεί με συγκεκριμένη στρατηγική ώστε να πετύχει σωστά αποτελέσματα. Η επαναλαμβανόμενη ζήτηση ίδιων σελίδων αλλά και μία απλή αρχιτεκτονική για προσπέλαση μερικών δεκάδων σελίδων το λεπτό δεν αποτελεί βέλτιστη λειτουργία. Θα πρέπει λοιπόν ο Crawler μέσα από μία λίστα από διαθέσιμες επισκέψεις να διαλέξει την κατάλληλη η οποία και δεν έχει ξαναζητηθεί αλλά και θα αποφέρει τα αναμενόμενα αποτελέσματα.

Μοντέρνες τεχνικές πλέον προσπαθούν κα κατανοήσουν το περιεχόμενο της σελίδας από την ίδια την διεύθυνση. Αυτό αποδίδει ευελιξία και ταχύτητα καθώς περιορίζει την αναζήτηση των σελίδων μόνο σε αυτές που θα εν δυνάμει «μεταφέρουν» την σωστή πληροφορία.

Ο Crawler αποτελεί τον κινητήριο μοχλό για την μηχανή αναζήτησης καθώς είναι αυτός όπου θα περισυλλέξει τα δεδομένα από το διαδίκτυο. Η διαδικασία όμως ενημέρωσης των δεδομένων αποτελεί και το σημαντικότερο πρόβλημα της λειτουργίας μιας μηχανής αναζήτησης. Πολλές φορές απαιτούνται εβδομάδες ή ακόμα και μήνες για την ολοκλήρωση προσπέλασης ενός μόνο τμήματος του διαδικτύου. Όλο αυτό το διάστημα είναι πολύ πιθανόν οι πληροφορίες που έχουν συλλεχθεί να μην είναι έγκυρες, να έχουν επεξεργαστεί ή ακόμα και διαγραφεί.

Από την πλευρά της μηχανής αναζήτησης υπάρχει κάποιο κόστος που συνδέει την ανίχνευση και εύρεση κάποιου γεγονότος που προκύπτει από την μη εγκυρότητα του καταγεγραμμένου πόρου. Για την αξιοπιστία και την ορθότητα των αποτελεσμάτων, χρησιμοποιούνται δύο βασικές έννοιες, το **Freshness** που σαν αποτέλεσμα δυαδικής

τιμής, εκφράζει αν το αντίγραφο είναι έγκυρο συναρτήσει του χρόνου και η τιμή **Age**, που παρουσιάζει σε χρόνο την «γήρανση» του αποθηκευμένου αντίγραφου. Για την διατήρηση υψηλού επιπέδου Freshness, θα πρέπει τα δεδομένα που αποθηκεύονται να μην αλλάζουν συχνά, πρακτικά αυτό σημαίνει στατικές HTML σελίδες όπου το περιεχόμενο τους είναι κατά βάση μη δυναμικό και για την διατήρηση του όρου Age χαμηλού, η συχνή επίσκεψη, με βάση παραμέτρους χρόνου, σε σελίδες όπου το περιεχόμενό τους μεταβάλλεται πιο συχνά.

- *Για την υποβοήθηση του Web Crawler στη λήψη απόφασης του χρόνου επανεπισκεψιμότητας στη σελίδα, χρησιμοποιούνται meta tags σε επίπεδο HTML κώδικα, που εκφράζουν την συχνότητα ανανέωσης του περιεχομένου της σελίδας.*

- *Ηθικά ζητήματα Crawling*

Όπως είναι εύκολα κατανοητό, η λειτουργία εξερεύνησης του διαδικτύου από έναν Web Crawler είναι διαδικασία ταχύτερη από το απλό «σερφάρισμα» του χρήστη από σελίδα σε σελίδα για να εξάγει επιθυμητά αποτελέσματα. Η ταχύτητα των λειτουργιών αυτών πολλαπλασιάζεται όταν η μηχανή εκμεταλλεύεται τεχνικές παραλληλοποίησης, άρα και ο όγκος ζήτησης δεδομένων από τον εξυπηρετητή πολλαπλασιάζεται αντίστοιχα. Ας σκεφτούμε επίσης ότι δεν υπάρχει μόνο μία μηχανή αναζήτησης για το διαδίκτυο αλλά ο καθένας μπορεί να λειτουργεί μια για γενικούς ή ειδικούς σκοπούς.

Οι παραπάνω λειτουργίες απαιτούν πόρους, όχι τόσο από τον ίδιο τον Crawler αλλά από το ίδιο το διαδίκτυο και τους εξυπηρετητές των εκάστοτε εφαρμογών. Πρακτικά, αυτό σημαίνει φόρτος για την εφαρμογή και καθυστερημένη εξυπηρέτηση αιτήσεων σε πραγματικούς χρήστες. Τέλος, ο Web Crawler σαν εφαρμογή μπορεί να επηρεάσει την απόδοση της σελίδας, καθώς μπορεί να ζητήσει παραπάνω δεδομένα απ’ ότι μπορεί να επεξεργαστεί, δημιουργώντας έτσι αδικαιολόγητο φόρτο στον εξυπηρετητή. Αν λάβουμε υπ όψιν μας, ότι ο καθένας μπορεί να λειτουργήσει έναν Web Crawler για ιδιωτικό σκοπό, τότε ο φόρτος αυτόματα πολλαπλασιάζεται, ειδικά όταν η υλοποίηση της μηχανής δεν είναι σωστά δομημένη και κάνει άσκοπη χρήση πόρων.

Επιπλέον, σε μια εφαρμογή, υπάρχουν αρχεία που οι χρήστες δεν χρειάζεται να γνωρίζουν ότι υπάρχουν ή δεν έχουν καμία αξία δεδομένων για την αποθήκευση και διαμοιρασμό πληροφορίας. Είναι λογικό, αυτά τα αρχεία να μην προσπελάζονται από τον Crawler, καθώς θα δημιουργήσουν αδικαιολόγητο φόρτο. Για την σωστή λειτουργία των Web Crawler, υπάρχει σε κάθε εξυπηρετητή αρχείο με όνομα «robots.txt». Στο αρχείο αυτό αναγράφονται οδηγίες για το πώς πρέπει να συμπεριφέρεται ο Crawler κατά την επίσκεψη του στον διαδικτυακό τόπο. Μπορεί δηλαδή κάποιος να αποκλείσει συγκεκριμένες μηχανές ή να δηλώσει μόνο τις μηχανές που επιθυμεί να δεχτεί για αναζήτηση. Μπορεί να ορίσει περιοχές που δεν

είναι αναγκαίο ή δεν πρέπει η μηχανή να εξερευνήσει ή ακόμα, όπως πλέον υποστηρίζουν οι σύγχρονες μηχανές, να ορίσει τα χρονικά διαστήματα ζήτησης αρχείων από τον εξυπηρετητή.

- Παράδειγμα αρχείου robots.txt από τον διαδικτυακό χώρο της Google.gr

```
User-agent: *
Disallow: /search
Disallow: /sdch
Disallow: /groups
Disallow: /images
Disallow: /catalogs
Allow: /catalogs/about
Allow: /catalogs/p?
…
```

1.3.2 Αποθήκευση και Indexing

Στην προηγούμενη ενότητα έγινε αναφορά στις μηχανές Web Crawling που αποτελούν την ραχοκοκαλιά μιας μηχανής αναζήτησης, καθώς είναι οι εφαρμογές όπου θα ζητήσουν τα δεδομένα από τους εξυπηρετητές. Η ιδέα όμως της λειτουργίας μιας μηχανής αναζήτησης δεν βασίζεται στην αναζήτηση σε πραγματικό χρόνο αλλά σε αναζήτηση ήδη προ-αποθηκευμένων δεδομένων που σχετίζονται με τα πραγματικά δεδομένα του διαδικτύου. Επίσης για την πετυχημένη αναζήτηση, τόσο σε ταχύτητα όσο και σε ορθότητα δεδομένων, θα πρέπει τα δεδομένα να αποθηκευτούν σε ειδικές δομές, κατάλληλα επεξεργασμένα, με αφαιρετική έννοια, ώστε ο όγκος τους να είναι όσο το δυνατόν μικρότερος και να επιτευχτεί ταχύτερη επεξεργασία πριν την τελική τους επισκοπή.

Η αποθήκευση όπως εκφράζει και η ίδια η λέξη, αναφέρεται στην διαδικασία διατήρησης της πληροφορίας, πιθανότατα σε κάποιο τρίτο λογισμικό όπως π.χ. μια βάση δεδομένων. Κατά την αποθήκευση τα δεδομένα συνήθως διατηρούνται αυτούσια χωρίς να υποστούν κάποιο είδος επεξεργασίας. Είναι δεδομένα τα οποία η μηχανή αναζήτησης μπορεί να ανακαλέσει για προβολή ή και μετέπειτα επεξεργασία, για την εξαγωγή συμπερασμάτων σχετικά με τις σελίδες αλλά και ακόμα για αξιολόγηση και βελτιστοποίηση της λειτουργίας αλγορίθμων. Τα δεδομένα αυτά είναι συνήθως το περιεχόμενο των HTML Tags που σχετίζονται με τον τίτλο, την περιγραφή, λέξεις κλειδιά κωδικοποίηση κ.α., καθώς και μέρος ή και ολόκληρο το «καθαρό» περιεχόμενο της σελίδας ή και ολόκληρος ο HTML κώδικας.

Στις σύγχρονες μηχανές αναζήτησης, που το αποτέλεσμα αποτελεί σημαντικότερο κομμάτι από την ταχύτητα απόκρισης, συν του ότι οι αλγόριθμοι αξιολόγησης των

σελίδων σύμφωνα με το περιεχόμενο τους, καθιστούν αναγκαία την διατήρηση και επεξεργασία της πληροφορίας, πέρα από το περιεχόμενο της κάθε σελίδας, διατηρούνται δεδομένα που έχουν να κάνουν με την προέλευσή της, τους χρόνους ζήτησης και επεξεργασίας από τους εξυπηρετητές, συσχετισμός της σελίδας με άλλους διαδικτυακούς τόπους, το μέγεθος της, δεδομένα που η ίδια εφαρμογή μπορεί να δημιουργήσει ανάλογα με την σελίδα κλπ.

Πέρα από την αποθήκευση των δεδομένων όπως αναφέρθηκε παραπάνω, που συντελεί στην πλουσιότερη απόδοση πληροφορίας, είναι εξίσου αναγκαία και η ταχύτητα απόδοσης της πληροφορίας αυτής. Η αναζήτηση σε ακατέργαστα δεδομένα των βάσεων δεδομένων και λειτουργία αλγορίθμων για την εξαγωγή συμπερασμάτων σε πραγματικό χρόνο, όπως εύκολα είναι αντιληπτό, θα αποτελούσε μειονέκτημα για την γρήγορη λειτουργία της μηχανής. Για τον λόγω αυτό χρησιμοποιούνται τεχνικές «Indexing» σε μη επεξεργασμένα δεδομένα. Η διαδικασία αυτή αποτελεί κομμάτι αναγκαίας λειτουργίας για την μηχανή, καθώς κατά την ζήτηση και αποθήκευση των δεδομένων, εκτελούνται συγκεκριμένοι αλγόριθμοι και διαδικασίες για την απόδοση συμπερασμάτων, ενώ αφηρημένη αλλά και συγκεκριμένη πληροφορία επεξεργάζεται με την διαδικασία του Indexing, ώστε κατά την αναζήτηση, ο όγκος που θα πρέπει να ελεγχθεί είναι υποπολλαπλάσια μικρότερος και η πληροφορία στοχευόμενη και σύντομη.

Οι τεχνικές Indexing όπως θα αναφέρουμε και παρακάτω, έχουν σαν βασική λειτουργία την «κωδικοποίηση» λέξεων ή φράσεων, όπου με κατάλληλες διαδικασίες, βαθμολογούν την προς αναζήτηση λέξη η φράση και επιστρέφουν ιεραρχικές λίστες αποτελεσμάτων. Ένα πολύ γνωστό παράδειγμα τέτοιας εφαρμογής ανοιχτού κώδικα, είναι το *Lucene.*

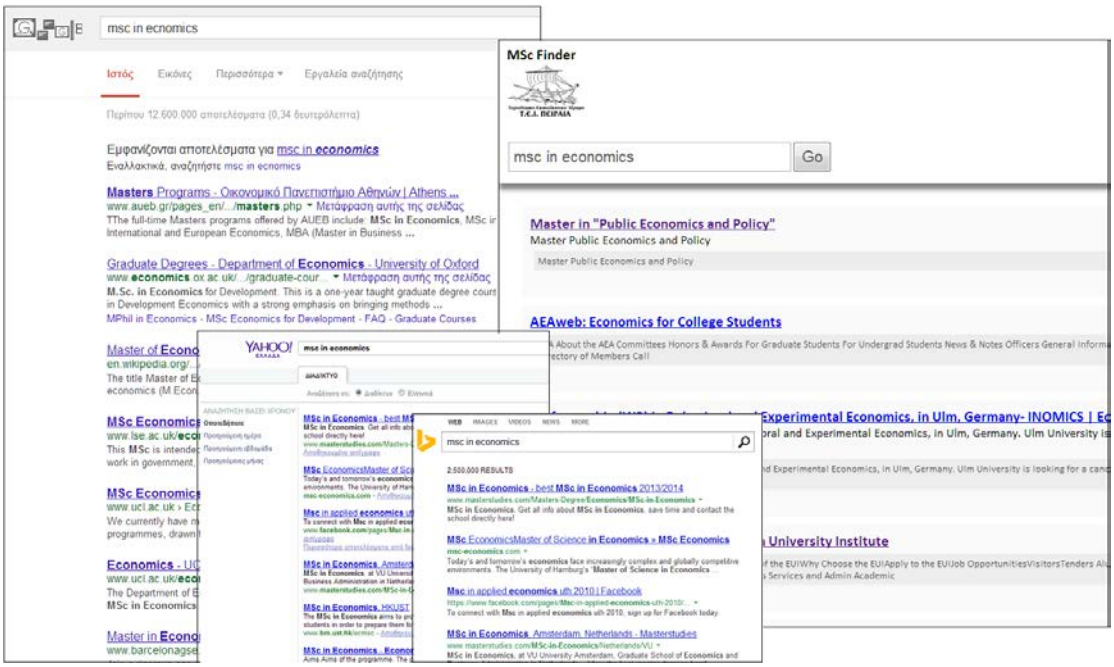
1.3.3 Αναζήτηση και Εμφάνιση αποτελεσμάτων

Το τελικό βήμα έπειτα από την εξερεύνηση, αποθήκευση και επεξεργασία των αποτελεσμάτων, είναι και η προβολή αυτών στους χρήστες του διαδικτύου. Το διαδίκτυο αποτελεί έναν ζωντανό οργανισμό ο οποίος εξελίσσεται, καινοτόμες λειτουργίες εισάγονται καθημερινά, άλλες διαδικασίες ξεχνιούνται και δεν θα ήταν λογικό οι χρήστες του να μην συμβαδίζουν με αυτό! Οι χρήστες είναι απαιτητικοί, επιζητούν ταχύτητα και ορθότητα των δεδομένων και οι Κολοσσοί όπως η Google, Microsoft και Yahoo φρόντισαν να δημιουργήσουν αυτά τα στάνταρ, που οποιαδήποτε αντίστοιχη εφαρμογή πρέπει να ακολουθεί ώστε να έχει μερίδιο επιτυχίας στο διαδίκτυο.

Ενώ η αρχική μορφή της προς αναζήτησης πρότασης αποτελούταν από σύντομες ή και μονολεκτικές φράσεις και πολλά πεδία ελέγχου σε επίπεδο HTML κώδικα (radio buttons, check boxes), πλέον οι μηχανές αναζήτησης έχουν αποκτήσει επίσης ευφυΐα και στο κομμάτι της έκφρασης των χρηστών. Πρώτη η Google καινοτόμησε με το να

«διορθώνει» τους χρήστες στα λεκτικά τους και εισήγαγε την έννοια «Μήπως εννοείται…?» ή και να συμπληρώνει αυτόματα τις λέξεις/προτάσεις. Για την υλοποίηση της συγκεκριμένης διαδικασίας, αλγόριθμοι τρέχουν σε πραγματικό χρόνο για την «εξερεύνηση» των προτάσεων αναζήτησης και δόμησης της με κατάλληλο τρόπο ώστε να επιτευχθεί αποτελεσματικότερη αναζήτηση.

Η προβολή των αποτελεσμάτων δεν θα μπορούσε να έχει άλλη πορεία από αυτή που χάραξαν οι πρώτες μηχανές αναζήτησης. Το διαδίκτυο, αποτελεί πλέον μέρος της καθημερινότητας. Η συνήθειες των ανθρώπων και η προσαρμογή τους σε καινούργιες καταστάσεις αποτελούν δύο αντικρουόμενες διαδικασίες. Για την εξασφάλιση λοιπόν της ομαλής μετάβασης σε μια καινούργια εφαρμογή, αλλά να διατηρείται η προηγούμενη γνώση των χρηστών σε αντίστοιχες λειτουργίες, εισήχθησαν στο διαδίκτυο και στην ανάπτυξη των εφαρμογών του τεχνικές UX(User Experience). Η μελέτη στην προβολή των αποτελεσμάτων αναζήτησης έδειξε ότι η απλή παρουσίαση τους σε συγκεκριμένα χρώματα εξυπηρετεί τους χρήστες να κατανοήσουν, να επεξεργαστούν και να χρησιμοποιήσουν την διδόμενη απάντηση. Παρακάτω παρουσιάζεται η μορφή των αποτελεσμάτων σε διαφορετικές μηχανές αναζήτησης. Σημαντικό είναι να παρατηρήσουμε τον τρόπο απόδοσης αυτών στον χρήστη.



Εικόνα 7. Προβολή αποτελεσμάτων αναζήτησης. a.Google, b.MScFinder, c.Yahoo, d.Bing

2. MSc Finder & Funnel Web Spider, Τεχνολογίες και Αρχιτεκτονική

Ανάμεσα στις πολλές μηχανές αναζήτησης και στους Web Crawlers, έρχεται να προστεθεί και η μηχανή που υλοποιήθηκε για τις ανάγκες της συγκεκριμένης πτυχιακής εργασίας. Όπως αναφέρεται και στο θέμα της εργασίας, για την σωστή λειτουργία της μηχανής, υπάρχουν τρία σημαντικά κομμάτια υλοποίησης. Το πρώτο αναφέρεται στο ρομπότ που είναι υπεύθυνο για την εξερεύνηση του διαδικτύου και την περισυλλογή ακατέργαστων δεδομένων από τις διαδικτυακές σελίδες, το δεύτερο κομμάτι αναφέρεται στην προβολή αυτών σε μια ξεχωριστή εφαρμογή με γραφικό περιβάλλον η οποία φιλοξενείται στο διαδίκτυο και παρουσιάζει με ευνόητο τρόπο τα δεδομένα που αναζητούν οι χρήστες και τέλος το τρίτο κομμάτι, που είναι και το σημαντικότερο, η εύρεση μέσα από πλήθος πληροφορίας την στοχευόμενη πληροφορία που αναφέρεται στα μεταπτυχιακά προγράμματα. Παρακάτω, παρουσιάζονται οι τεχνολογίες που στηρίχθηκε η εφαρμογή για την υλοποίηση της καθώς, η αρχιτεκτονική του συστήματος και διεπαφές επικοινωνίας μεταξύ των τριών τμημάτων της εφαρμογής.

2.1 Αρχιτεκτονική συστήματος

Όπως αποδεικνύεται διαχρονικά, για την σωστή δόμηση και ανάπτυξη των έργων, σε οποιοδήποτε τομέα, σημαντικό ρόλο παίζει η αρχιτεκτονική και η μελέτη υλοποίησης του. Στον ψηφιακό κόσμο επίσης, δεν θα μπορούσαν να λείπουν αυτές οι έννοιες. Με την σωστή αρχιτεκτονική και μελέτη πετυχαίνουμε το μέγιστο δυνατό αποτέλεσμα και επιπλέον αφήνουμε ανοιχτές οποιεσδήποτε άλλες δυνατότητες για την εξέλιξη των συστημάτων. Αντίστοιχες τεχνικές και χρόνος εφαρμόστηκαν για την υλοποίηση του MSc Finder αλλά και του Funnel Web Spider των δύο βασικών τμημάτων του συστήματος.

- MSc Finder

Αποτελεί την διαδικτυακή διεπαφή που είναι διαθέσιμη για την ευκολότερη αναζήτηση των αποτελεσμάτων και προβολή αυτών από τους χρήστες. Υλοποιημένη ως μια “stand alone” εφαρμογή, κάνει χρήση των αποτελεσμάτων εύρεσης του ρομπότ και των αποθηκευμένων δεδομένων στη βάση.

❖ Funnel Web Spider

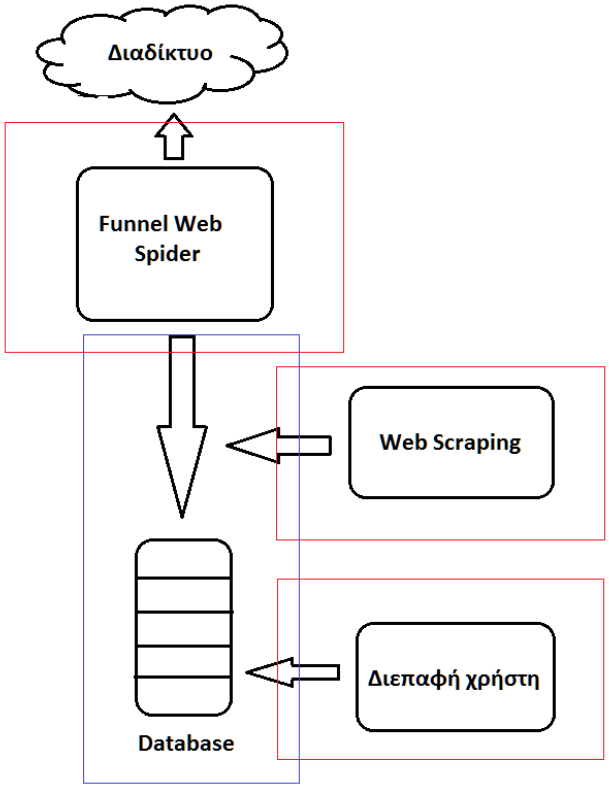
Αποτελεί το ρομπότ για την περισυλλογή των δεδομένων στο διαδίκτυο. Από διδόμενες αρχικές διευθύνσεις ή χρήση αποθηκευμένων δεδομένων στη βάση, «αναπηδάει» από σελίδα σε σελίδα και ενημερώνει ή καταγράφει καινούργια δεδομένα. Πήρε το όνομα του από την ομώνυμη Αυστραλιανή αράχνη, η οποία υφαίνει τον ιστό της σε μορφή χωνιού. Η συγκεκριμένη διαδικασία αποτελεί και μια απεικόνιση των λειτουργιών του συστήματος, καθώς έχει σαν στόχο, μέσα από πλήθος ακατέργαστων δεδομένων να βρει συγκεκριμένα δεδομένα, που στην περίπτωση μας αναφέρονται στα μεταπτυχιακά προγράμματα.

Εικόνα 8. Αυστραλιανή αράχνη Funnel Web Spider

❖ Web Scraping

Αποτελεί την λειτουργία ανάμεσα στο ρομπότ και την διαδικασία αποθήκευσης των δεδομένων για χρήση. Κατέχει την υψηλότερη «νοημοσύνη» στο σύστημα, καθώς μέσα από απλό HTML κώδικα ή κείμενο της κύριας σελίδας, μπορεί να εξάγει στοχευόμενα δεδομένα που αφορούν τον τίτλο του μεταπτυχιακού προγράμματος, τηλέφωνα και ηλεκτρονικές διευθύνσεις ταχυδρομείου που σχετίζονται με αυτό ή κάποια γενική, αλλά χρήσιμη πληροφορία, π.χ. σύντομη περιγραφή, διευθύνσεις, μαθήματα, ανακοινώσεις κλπ. Για την συγκεκριμένη διαδικασία, θα αναφερθούμε κυρίως στο 3^{ο} κεφάλαιο.

Παρακάτω παρουσιάζεται γραφικά η λειτουργία και συνδυασμός των επιμέρους τμημάτων της εφαρμογής. Η κάθε μια μπορεί να εκφραστεί και αναλυθεί ξεχωριστά από τις υπόλοιπες, πράγμα που προσθέτει ευελιξία στην ανάπτυξη και συντήρηση του κώδικα. Με κόκκινο περίγραμμα ξεχωρίζουν τα επιμέρους τμήματα ενώ με το μπλε παρουσιάζονται οι διαδικασίες που τις συνδέουν. Για τις διαδικασίες αυτές θα αναφερθούμε παρακάτω.



Εικόνα 9. Msc Finder, Funnel Web Spider και Web Scraping system

2.2 Τεχνολογίες

Πριν περίπου 200 χρόνια, η ανθρωπότητα γνώρισε την περίοδο ανάπτυξης και καλλιέργειας της ως προς τις καλές τέχνες, την επιστήμη τον ανθρωπισμό. Όχι πολύ πριν από 30 χρόνια, ένα νέο κίνημα έφερε επίσης επανάσταση και αποτέλεσε σημείο αναφοράς για την εξέλιξη της μουσικής και της ελευθερίας των ανθρώπων, ίσως μας είναι πιο γνωστό με τον όρο «τα παιδιά των λουλουδιών». Σήμερα, αρχές του 21^{ου} αιώνα, η εξελικτική ανάπτυξη της τεχνολογίας και των υπολογιστικών συστημάτων, μοιάζει να βρίσκεται στο ανοδικότερο σημείο της ενώ ταυτόχρονα υπόσχεται ότι η άνοδος αυτή θα διαρκέσει για πολλά χρόνια ακόμη.

Ο προγραμματισμός και η ανάπτυξη συστημάτων που εξυπηρετούν τον άνθρωπο, φαίνεται να κατέχει τον κυρίαρχο ρόλο και όπως είναι λογικό και ιστορικά αποδεδειγμένο, όταν η ανθρωπότητα θέλει, τότε πετυχαίνει. Αυτό λοιπόν συμβαίνει και στην επιστήμη των υπολογιστών και του λογισμικού. Καθημερινά, νέα μοντέλα ανάπτυξης λογισμικού εισάγονται στην αγορά. Νέοι κανόνες εφαρμόζονται για να διευκολύνουν την ανάπτυξη και συντήρηση των προγραμμάτων. Καινούργιες

τεχνολογικές έννοιες εισάγονται και πολλές από αυτές έρχονται για να μείνουν. Η ανάπτυξη λογισμικού πλέον αποτελεί την καινούργια παγκόσμια επανάσταση ή ακόμη και καλλιτεχνία!

Έτσι λοιπόν, όπως κάθε «μοντέρνα εφαρμογή», έτσι και η ανάπτυξη της μηχανής αναζήτησης MSc Finder, βασίστηκε στις καινοτόμες διαδικασίες που διέπουν τις σύγχρονες διαδικτυακές εφαρμογές.

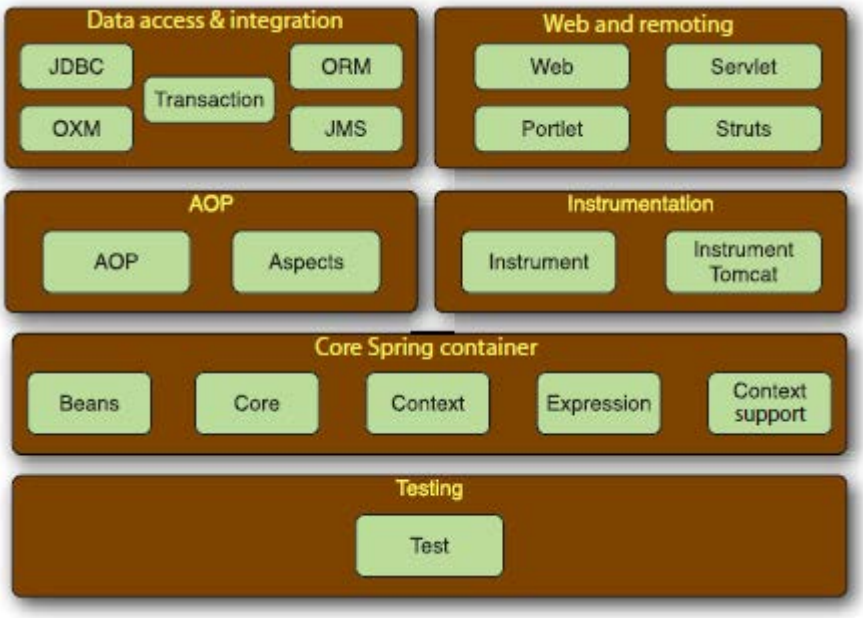
2.2.1 Spring Framework / Spring MVC

Το Spring Framework αποτελεί πλατφόρμα ανοιχτού κώδικα κάτω από την άδεια *Apache 2.0 license*, για γλώσσα προγραμματισμού Java, καθώς και μια από τις πιο διαδεδομένες πλατφόρμες για την ανάπτυξη Enterprise συστημάτων. Η πιο πρόσφατη έκδοση είναι η 3.2.4 η οποία ανακοινώθηκε τον Αύγουστο του 2013 με πολλές προσθήκες για WebSockets, υποστήριξη Java EE7 καθώς και Java SE8.

- Γιατί Java? Μια μηχανή αναζήτησης, με τα επιμέρους της τμήματα αποτελεί μια εφαρμογή η οποία μπορεί να υλοποιηθεί σε οποιαδήποτε γλώσσα προγραμματισμού. Πολλές γνωστές και αξιόπιστες μηχανές αναζήτησης και ρομπότ είναι δομημένες σε Python, PHP, Ruby και φυσικά σε Java. Λόγω της υπερβολικής μου αγάπης ως προς την τελευταία συν του ότι αποτελεί πλέον μια ώριμη γλώσσα προγραμματισμού με την μεγαλύτερη κοινότητα σε όλο τον κόσμο που συνεχώς εξελίσσεται, ήταν οι κύριοι λόγοι όπου η Java αποτέλεσε πρώτη και μοναδική επιλογή για την υλοποίηση της εφαρμογής. Η Java μπορεί να χρησιμοποιηθεί σαν γλώσσα γενικού αλλά και ειδικού σκοπού, όπως η εν λόγω εφαρμογή, με πολλές διαθέσιμες βιβλιοθήκες του πυρήνα της αλλά και από τρίτους, που μπορούν να κάνουν ευκολότερη την ανάπτυξη των εφαρμογών. Τέλος με μεγάλες και ενεργές κοινότητες στις επιμέρους πλατφόρμες (όπως Spring, Hibernate κλπ) και κάνοντας βέλτιστη χρήση της αντικειμενοστραφής έκφρασης, αποτελεί χρήσιμο εργαλείο για την ανάπτυξη μεγάλων και δομημένων Enterprise εφαρμογών.

Το Spring Framework, ύστερα από την πρώτη του έκδοση το 2002 έχει εξελιχθεί σε μία ώριμη πλατφόρμα με στοιχεία που στοχεύουν στην ασφαλή και σταθερή λειτουργία των συστημάτων. Αποτελεί στην πραγματικότητα ένα εργαλείο το οποίο βοηθάει τον προγραμματιστή να αποσπαστεί από την υλοποίηση απλών και χρονοβόρων διαδικασιών της εφαρμογής, οι οποίες βέβαια είναι απαραίτητες για την λειτουργία του συστήματος, όπως η σύνδεση με μια βάση δεδομένων, απελευθέρωση πόρων για το σύστημα, υλοποίηση και σωστή λειτουργία connection pooling, επικοινωνία και δημιουργία αντικειμένων κ.α., γλιτώνοντας απαραίτητο χρόνο, ο

οποίος μπορεί να χρησιμοποιηθεί στην ανάπτυξη του Business Logic της εφαρμογής. Επίσης εισάγει έννοιες όπως το Dependency Injection(DI) ή Inversion of Control(IoC) και Aspect Oriented Programming(AOP) όπως θα αναλυθούν και παρακάτω.

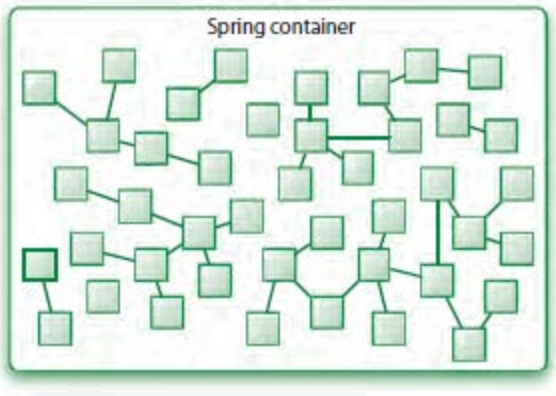


Εικόνα 10. Τα έξι επίπεδα του Spring Framework

Η βασική μονάδα υλοποίησης για το Spring είναι το *Bean.* Τα *Beans* δεν αποτελούν τίποτα παραπάνω από απλές κλάσεις με κατασκευαστή και με μεθόδους *Getters* και *Setters,* όπως αναφέρονται και στην Java με τον ορισμό *Pojo*(Plain Old Java Object). Ήδη από πολύ νωρίς το 1996 με την εμφάνιση της Java και πριν ακόμα αυτή εδραιωθεί και αποκτήσει τους ακόλουθους που έχει σήμερα, η Sun Microsystems, που αποτελεί και τον κατασκευαστή της γλώσσας, εξέδωσε το JavaBeans 1.00 Specification που αποτελούσε περιγραφή για το μοντέλο υλοποίησης των Pojo. Δύο χρόνια αργότερα, οι εφαρμογές απαιτούσαν υπηρεσίες όπως ασφάλεια, κατανεμημένη ανάπτυξη εφαρμογών, διεπαφές επικοινωνίας κ.α., οπότε και παρουσιάστηκε η έκδοση 1.0 για τα Enterprise Java Beans(EJB) to 1998. Το EJB αποτελούσε χρήσιμο εργαλείο για την ανάπτυξη εφαρμογών, καθώς αυτό-υλοποιούσε διαδικασίες για τους προγραμματιστές, αλλά πολλές φορές μετέτρεπε ακόμα και απλές εφαρμογές σε δυσνόητες και δύσκολο-συντήρητες. Αυτός ήταν και ο λόγος που τα EJB απέκτησαν πολλούς οπαδούς αλλά και ταυτόχρονα πολλούς εχθρούς.

Η βάση όμως είχε τεθεί. Τα Beans ή αλλιώς Pojos θα αποτελούσαν την βασική μονάδα υλοποίησης καθώς η απλή τους δομή συντελεί στην συντήρηση και ανάπτυξη τους. Το κλειδί ήταν απλά η απλοποίηση των διαδικασιών, το οποίο επιτυχημένα καταφέρνει το Spring Framework.

Ένας από τους βασικούς όρους σε μία εφαρμογή υλοποιημένη με το Spring Framework είναι το Spring Container. Το Spring container αποτελεί ένα κρυμμένο από τον προγραμματιστή πεδίο, το οποίο είναι βέβαια προσβάσιμο σε αυτόν με κατάλληλες διαδικασίες και μεθόδους. Είναι υπεύθυνο για την υποστήριξη και λειτουργία όλης της εφαρμογής. Το Spring container είναι πλέον υπεύθυνο για την δημιουργία των αντικειμένων, τον συσχετισμό τους, ακόμα και τον καθαρισμό τους από την μνήμη ύστερα από την εκκίνηση της εφαρμογής. Όλα τα Spring components «ζουν» εκεί και επικοινωνούν μεταξύ τους ανταλλάσοντας πληροφορίες και δεδομένα ενώ ο προγραμματιστής μπορεί να επέμβει δημιουργώντας επιπλέον στοιχεία ή καινούργιες συνάψεις μεταξύ τους.



Εικόνα 11. Spring container με τα components και πως συνδέονται μεταξύ τους.

Το πλέον διαδεδομένο κατά την υλοποίηση container είναι το Application Context που κάνει βέλτιστη χρήση του Dependency Injection. Το Spring Framework εκτελεί όλες τις διαδικασίες εκκίνησής του κατά την έναρξη του εξυπηρετητή. Για να πετύχει την αρχικοποίηση των αντικειμένων, είναι απαραίτητο να εισαχθεί αρχείο XML όπου περιέχει και τον ορισμό των Beans. Παρακάτω παρουσιάζεται ένα παράδειγμα το από το XML αρχείο του Funnel Web Spider.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:p="http://www.springframework.org/schema/p"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<bean id="domain" class="com.funnelweb.beans.Domain" lazy-init="true"
scope="prototype" />

<bean id="page" class="com.funnelweb.beans.DomainPage" lazy-init="true"
scope="prototype" />
…
```

```
<bean id="domainDAO" class="com.funnelweb.dao.DomainDAO"
parent="sessionFactoryObjectDAO" scope="singleton" />

 <bean id="domainPageDAO" class="com.funnelweb.dao.DomainPageDAO"
 parent="sessionFactoryObjectDAO" scope="singleton" />

…
</beans>
```

Από το παραπάνω παράδειγμα αρχείου ρύθμισης για το Spring Framework παρατηρούμε ότι ένα απλό Bean δηλώνεται μέσα στην κεφαλίδα <bean /> ή <bean></bean>. Παίρνει ένα χαρακτηριστικό Id το οποίο είναι και το όρισμα κλήσης για την χρήση του ή και διασύνδεσης του με άλλα Beans. Στο πεδίο «class» δηλώνεται το classpath, δηλαδή η ακριβής τοποθεσία της κλάσης που από την οποία το Spring Container θα αναζητήσει για να δημιουργήσει το στιγμιότυπο του αντικειμένου. Ένα από τα σημαντικότερα πεδία της κεφαλίδας είναι το scope. Αυτό δηλώνει στο σύστημα αν το στιγμιότυπο θα έχει αντίστοιχα χαρακτήρα «prototype» ή «singleton». Με τον όρο prototype εννοούμε ότι το Container κάθε φορά που θα ζητείται το αντικείμένο, θα δημιουργεί καινούργιο στιγμιότυπο. Είναι κάτι αντίστοιχο με την έκφραση

MyClass test = new Myclass(..);

Δηλαδή γίνεται allocate κατάλληλο μέγεθος μνήμης για την δημιουργία νέου αντικειμένου. Με τον ορισμό singleton, εννοούμε ότι κατά την κλήση του αντικειμένου, το Container θα επιστρέψει πάντα το ίδιο στιγμιότυπο. Αυτή η τεχνική είναι γνωστή για μεγάλες Enterprise εφαρμογές που πέραν από την απόδοση σε ταχύτητα, επιζητούμε και την οικονομία μνήμης. Παρακάτω παρουσιάζεται σε Java υλοποίηση η τεχνική για singleton design pattern.

```
public class MyClass {
        private MyClass instance;
        private MyClass(){}
        public static MyClass getInstance() {
                if( instance == null )
                        instance = new MyClass();
                return instance;
        }
        public void myMethod(..){…}
}
```

Κατά την χρήση αντικειμένων με singleton χαρακτήρα, είναι σημαντικό να λαμβάνουμε υπόψη μας ότι η προσπέλαση του μπορεί να γίνει από διαφορετικά αντικείμενα την ίδια στιγμή, ειδικότερα όταν το σύστημα αποτελεί ένα web application το οποίο το επισκέπτονται πολλοί χρήστες. Είναι απαραίτητο λοιπόν να λαμβάνονται όλα τα μέτρα ώστε η εφαρμογή να «κλειδώνει» και να πετυχαίνει σωστή λειτουργία σε πολλαπλές ταυτόχρονες αιτήσεις.

Στο Spring Framework, κατά την δήλωση των Bean, υπάρχουν και άλλες διαθέσιμες οδηγίες που βοηθούν τον προγραμματιστή να πετύχει την μέγιστη δυνατή απόδοση. Κάποια από αυτά όπως αναφέρονται και στο παράδειγμα είναι το «lazy-init» που αποτελεί οδηγία για τον χρόνο φόρτωσης και δημιουργίας του στιγμιότυπου. Όπως αναφέρθηκε παραπάνω, κατά την εκκίνηση του εξυπηρετητή και φόρτωση της εφαρμογής, το Spring Container θα προβεί σε διαδικασίες αρχικοποίησης των beans, με την παραπάνω οδηγία αναστέλλουμε την αρχικοποίηση του αντικειμένου έως ότου αυτό ζητηθεί για πρώτη φορά. Με αυτό τον τρόπο, μειώνουμε τον φόρτο αρχικοποίησης της εφαρμογής, άρα σημαίνει και γρηγορότερη εκκίνηση. Με την οδηγία «parent» ορίζουμε την γονική κλάση από την οποία το αντικείμενο αρχικοποιείται. Πρακτικά, αποτελεί οδηγία για την **extend** εντολή σε προγραμματιστικό επίπεδο. Το όρισμα στο parent, αποτελεί το Id κάποιου άλλου Bean. Άλλη οδηγία αποτελεί το «init» όπου σαν όρισμα δέχεται μέθοδο της κλάσεις και καλείται αμέσως μετά την αρχικοποίηση της. Αποτελεί λοιπόν καλή στρατηγική για επιπλέον αρχικοποίηση σε προγραμματιστικό επίπεδο ή διαδικασίες που θα θέλαμε το αντικείμενο μας να υλοποιήσει πριν χρησιμοποιηθεί.

- Με τον όρο Dependency Injection εννοούμε την «χαλαρή» σύνδεση μεταξύ των υλοποιημένων αντικειμένων της εφαρμογής. Οι κλάσεις που είναι και τα βασικά δομικά στοιχεία, δεν εξαρτώνται άμεσα από συγκεκριμένα στιγμιότυπα αντικειμένων, πράγμα που κάνει τον κώδικα επαναχρησιμοποιήσιμο και πιο έυκολο στην συντήρηση. Η βασική ιδέα υλοποίησης για το DI λέγεται Inversion of Control, δηλαδή μια κλάση δεν πρέπει να αυτόρυθμίζεται αλλά να ρυθμίζεται εξωτερικά. Το DI μαζί με τα «ελαφριά» για τα συστήματα Container, αποτελούν δύο από της βασικότερες ευκολίες που προσφέρει το Spring Framework.

Παρακάτω παρουσιάζεται ένα σύντομο παράδειγμα για την διαχείριση του Dependency Injection.

```
<bean id=”test” class=”com.package.Test” scope=”prototype” init=”initialize”
  p:myVal=”test value” />

<bean id=”myClass” class=”com.package.MyClass”
```

```
  p:test-ref=”test” />

<bean id=”anotherClass” class=”com.package.AnotherClass” >
        <constructor-arg name="myArg" type="java.lang.String" value="test arg"/>
</bean>
```

Αρχικά δηλώνουμε ένα Bean όπου έχει την «μαγική» παράμετρο **p**. Προγραμματιστικά αυτό σημαίνει ότι στο απλό Pojo μας, έχουμε ορίσει μια παράμετρο **myVal**, στην οποία έχουμε προσθέσει getters και setters. Το Spring Container αντιλαμβάνεται την ύπαρξη των μεθόδων αυτών και μας επιτρέπει εξωτερικά, έπειτα από την δημιουργία του στιγμιότυπου, να του προσθέτουμε τιμή στο πεδίο myVal μέσω της set μεθόδου, η οποία είναι ίση με “test value”. Αυτό πρακτικά σημαίνει ότι σε κάθε αρχικοποίηση αντικειμένου από την κλάση com.package.Test, το Spring Container θα του αναθέτει την αρχική ορισμένη τιμή.

Βέβαια το να αναθέτουμε primitive type τιμές σε μεταβλητές ίσως να μην έχει τόση μεγάλη αξία και λειτουργικότητα. Τα πράγματα όμως αποκτούν ένα μεγάλο ενδιαφέρον στο δεύτερο παράδειγμα. Η τιμή στην μέθοδο πλέον παίρνει μορφή αντικειμένου και μάλιστα στιγμιότυπου που έχουμε ορίσει μόλις λίγο πιο πάνω! Αυτό που πετύχαμε λοιπόν, είναι το στιγμιότυπο του αντικειμένου myClass, να μην είναι απλά ένα κενό αντικείμενο, αλλά ήδη «φορτωμένο» με άλλα αντικείμενα της εφαρμογής.

Τι γίνεται όταν θέλουμε να ορίσουμε στον κατασκευαστή την τιμή? Η απάντηση είναι στο τρίτο παράδειγμα. Με λίγη φαντασία και εμπειρία, εύκολα καταλαβαίνουμε ότι στην κλάση com.package.AnotherClass υπάρχει κατασκευαστής με παράμετρο myArg τύπου String, όπου κατά την αρχικοποίηση του, του αναθέτουμε την τιμή «test arg». Η διαδικασία αυτή λειτουργεί ακριβώς το ίδιο και με τιμή ανάθεσης κάποιου αντικειμένου του Spring Container.

Το Spring Framework προσθέτει επιπλέον μαγεία, καθώς η σειρά δήλωσης των Beans δεν παίζει κανέναν απολύτως ρόλο. Το Container θα υλοποιήσει μόνο του τις διαδικασίες ώστε να αρχικοποιηθούν σωστά τα δεδομένα και να μην υπάρχει κίνδυνος «άδειου» αντικειμένου.

Στην εφαρμογή μας η ζήτηση του αντικειμένου που ήδη έχει φορτωθεί στο Spring Container είναι εξίσου εύκολη. Αρχικά θα πρέπει πρώτα να φορτωθεί στο Spring το XML αρχείο με τις κατάλληλες ρυθμίσεις.

```
final ApplicationContext ac = new ClassPathXmlApplicationContext( new String[] {
        "com/funnelweb/config/application-context.xml"
});
```

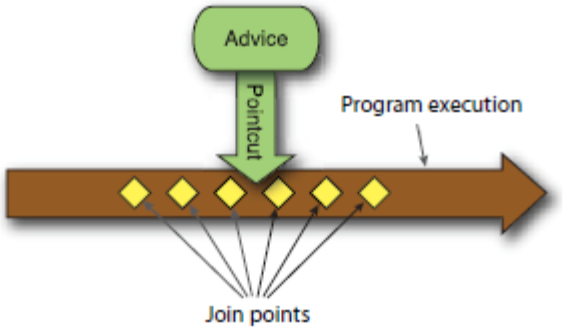
Το αντικείμενο «ac» είναι το Spring Container τύπου *ApplicationContext.* Τέλος ζητάμε από το Container κάποιο αρχικοποιημένο αντικείμενο.

```
FunnelWebSpiderGuiApp app = (FunnelWebSpiderGuiApp) ac.getBean("app");
```

Φορτώνεται λοιπόν αντικείμενο τύπου *FunnelWebSpiderGuiApp* από το Container, το οποίο έχει δηλωθεί με Id " *app* ".

Με τα παραπάνω παραδείγματα, γίνεται μια πρώτη αναφορά στο Dependency Injection και με ποιο τρόπο αυτό υλοποιείται από το Spring framework. Παρατηρούμε ότι τα αντικείμενα αρχικοποιούνται και δηλώνονται εξωτερικά σε αρχείο XML, το οποίο δίνεται σαν τροφοδοσία στο Spring Container και δημιουργείται ένα πέπλο με αντικείμενα και συσχετίσεις μεταξύ τους. Σαν καλή χρήση της παραπάνω τεχνικής, ορίζεται αναγκαία η χρήση των Intrface της Java, και αυτό διότι «χαλαρώνουμε» επιπλέον τους δεσμούς των αντικειμένων μεταξύ τους. Η ιδέα είναι απλή, όσο λιγότερες εξαρτήσεις, τόσο πιο σωστά δομημένη εφαρμογή!

- Με τον όρο Aspect Oriented Programming, εννοούμε τον επιπλέον διαμοιρασμό επιπέδου ανάπτυξης του κώδικα, από αντικειμενοστραφή έκφραση σε μικρότερες μονάδες τα Aspects ή αλλιώς «άποψη». Το AOP αποτελεί μια από τις βασικότερες δυνατότητες που προσφέρει το Spring Framework. Θα αναφέρουμε ότι δίνεται η δυνατότητα στον προγραμματιστή, να έχει τον έλεγχο του κώδικα ακόμα πιο μέσα από επίπεδο κλάσης, σε επίπεδο μεθόδου. Μπορεί δηλαδή ο προγραμματιστής να ορίσει "pointcuts" (σημαντικές περιοχές όπου θέλουμε κάτι να συμβαίνει όταν η εφαρμογή μας φτάνει στα σημεία αυτά) στον κώδικα και με κατάλληλες οδηγίες (after, before, around κλπ) να ορίσει επιπλέον διαδικασίες χωρίς να φορτώνει τις ήδη υπάρχουσες μεθόδους. Το σημαντικό στην υλοποίηση αυτή, είναι ότι μπορεί να πραγματοποιηθεί εξωτερικά σε XML αρχεία και επιπλέον για να υλοποιηθεί δεν είναι αναγκαίο να υπάρχει ο πηγαίος κώδικας!



Εικόνα 12. AOP προσθήκη λειτουργιών κατά την εκτέλεση του προγράμματος.

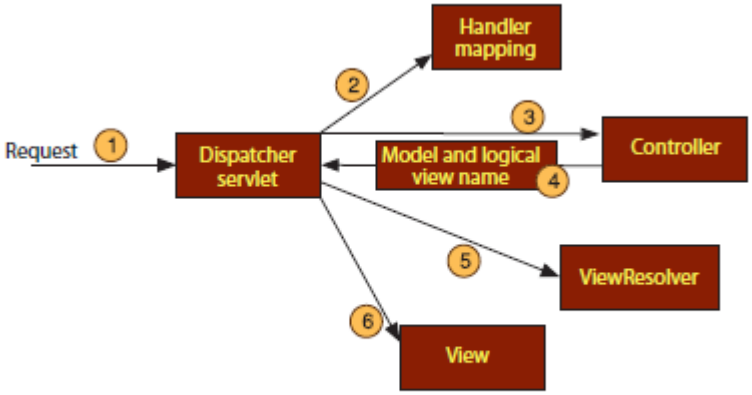
Spring MVC

Το Spring MVC αποτελεί το διαδικτυακό κομμάτι ανάπτυξης του Spring Framework. Βασισμένο στο μοντέλο Model-View-Controller (MVC) η πλατφόρμα βοηθάει τον προγραμματιστή να αναπτύξει γρήγορα και ευέλικτα διαδικτυακές εφαρμογές. Αποτελεί υποενότητα του κεντρικού πυρήνα του Spring, οπότε διατηρεί όλες τις βασικές λειτουργίες του, απαλλάσσοντας τον προγραμματιστή από βαριές και συνηθισμένες δομήσεις του συστήματος.

- Το μοντέλο MVC αναφέρεται στην λειτουργία του συστήματος. Αποτελεί στη πραγματικότητα μια θεωρητική αρχικά έννοια με άμεση όμως εφαρμογή που κατανέμει σωστά την υλοποίηση ώστε να χαλαρώνει τους δεσμούς μεταξύ λειτουργιών της εφαρμογής και το σημαντικότερο από όλα να ξεχωρίσει το επίπεδο Front, όπου συνήθως είναι η διεπαφή του χρήστη, με το επίπεδο Back, όπου συνθέτει όλες τις λειτουργίες για την περισυλλογή και δόμηση της πληροφορίας για τον χρήστη.

Με τον όρο Controller εννοούμε την διαδικασία όπου γίνεται κατανοητή η ενέργεια του χρήστη προς το σύστημα. Στο επίπεδο αυτό περισυλλέγονται χρήσιμες πληροφορίες όπου αποτελούν είσοδο για την εφαρμογή, ώστε να πράξει τις αντίστοιχες ενέργειες. Με τον όρο Model, εννοούμε στην ουσία το επίπεδο που είναι κρυμμένο από τον χρήστη, διότι δεν μπορεί και δεν πρέπει να επέμβει άμεσα και συνθέτει όλες εκείνες τις λειτουργίες που είναι απαραίτητες για την εξυπηρέτηση κάθε ενέργειας. Τέτοιες λειτουργίες μπορεί να είναι η αναζήτηση ή εγγραφή στη βάση δεδομένων, η απόδοση απάντησης από κάποιο περίπλοκο αλγόριθμο, ενημέρωση μεταβλητών και στατιστικών του συστήματος κλπ. Τέλος με τον όρο View, εννοούμε τον τρόπο που παρουσιάζεται η πληροφορία, ύστερα από

περισυλλογή και επεξεργασία, στον τελικό χρήστη. Συνήθως σε μια διαδικτυακή εφαρμογή, αυτός είναι HTML κώδικας για την περιήγηση από συνηθισμένο browser.



Εικόνα 13. Εικονική περιγραφή του μοντέλου MVC

Όπως φαίνεται και από την παραπάνω εικόνα, στην καρδιά του Spring MVC βρίσκεται ένα DispatcherServlet. Στην πραγματικότητα αυτός αποτελεί έναν διαχειριστή για τις κλήσεις που γίνονται προς το σύστημα. Αρχικά, ο χρήστης μπορεί να καλέσει μία σελίδα από τον εξυπηρετητή. Ο DispatcherServlet μέσα από το Handler mapping προωθεί την κλήση στον αντίστοιχο Controller. Ο Controller δέχεται την κλήση και συλλέγει από αυτή τυχόν δεδομένα που μπορεί να έστειλε ο χρήστης και προωθεί τα δεδομένα αυτά προς το Model / Backend για επεξεργασία. Ύστερα από την επεξεργασία, ο Controller δέχεται πίσω την πληροφορία για τον χρήστη και την αποστέλλει πίσω στον DispatcherServlet ο οποίος μέσα από τον ViewResolver επιλέγει το σωστό View και συνθέτει με αυτό την πληροφορία που έλαβε από το σύστημα ώστε να την παρουσιάσει στον χρήστη.

Το σύστημα λοιπόν, είναι δομημένο τμηματικά, ώστε το κάθε τμήμα να εκτελεί δικές του και μόνο λειτουργίες, αποφεύγοντας την στενή σύνδεσή τους, όπου συνήθως οδηγεί σε κακή υλοποίηση. Τέλος, το μοντέλο αυτό εξυπηρετεί το γεγονός ότι όλα είναι αυτόνομα, σε περίπτωση απασφαλμάτωσης, ο προγραμματιστής γνωρίζει ακριβώς που είναι το πρόβλημα και λόγω του μικρού συνήθως μεγέθους του κάθε τμήματος, η συντήρηση και επέκταση είναι εύκολη.

Παρακάτω παρουσιάζεται εν συντομία ένα μοντέλο MVC στην πράξη. Αρχικά, ορίζουμε τον DispatcherServlet στο αρχείο web.xml όπου κατέχει τις οδηγίες για τον τρόπο που θα διαχειριστεί την εφαρμογή ο κάθε εξυπηρετητής.

```
<servlet>
```

```
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>
           org.springframework.web.servlet.DispatcherServlet
       </servlet-class>
       <load-on-startup>2</load-on-startup>
</servlet>

<servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
</servlet-mapping>
```

Με την παραπάνω οδηγία ορίζουμε ότι κάθε κλήση που θα λαμβάνεται ύστερα από το root "/", θα προωθείται στον DispatcherServlet. Το σύστημα ύστερα από την έναρξη του εξυπηρετητή θα αναζητήσει στην εφαρμογή αρχείο XML με το όνομα του servlet, που από τα παραπάνω είναι ο «dispatcher» προσθέτοντας την κατάληξη «-servlet.xml». Στο αρχείο αυτό, βρίσκονται οι ρυθμίσεις για τον κεντρικό DispatcherServlet. Μπορούμε λοιπόν, όπως είδαμε και πιο πάνω να ορίσουμε Beans, όπου αποτελούν τους Controllers του συστήματος.

```
<bean name="/home" class="gr.teipir.webapp.mvc.HomeController" />
<bean name="/404" class="gr.teipir.webapp.mvc.Error404Controller" />
<bean name="/403" class="gr.teipir.webapp.mvc.Error403Controller" />
```

Η οδηγία εδώ αναφέρει στον Dispatcher ότι σε κάθε κλήση που γίνεται στο «/home», θα προωθείται η κλήση αυτή στον gr.teipir.webapp.mvc.HomeController. Το ίδιο συμβαίνει και με τις υπόλοιπες κλήσεις. Στο Controller πλέον, έχουμε τις αντίστοιχες διαδικασίες για την εξυπηρέτηση της κλήσης.

```
public class HomeController implements Controller
{
   @Override
   public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response)
throws Exception
   {
      ModelAndView mav = new ModelAndView("home");
      return mav;
   }
}
```

Όπως παρατηρούμε, ο Controller επιστρέφει πίσω ένα αντικείμενο τύπου ModelAndView, όπου έχει και την πληροφορία για το όνομα του View που θα χρησιμοποιηθεί για τον χρήστη.

Annotations

Η μαγεία του Spring Framework δεν τελειώνει εδώ. Σε πολύ μεγάλες εφαρμογές, ο όγκος των ρυθμίσεων και των οδηγιών για την πλατφόρμα αυξάνεται με ταχύς ρυθμούς. Ειδικότερα, προβλήματα όγκου εμφανίζονται όταν στην πυρήνα προσθέτουμε επιπλέον λειτουργίες όπως κάποιο ORM μοντέλο όπως θα δούμε και παρακάτω, κάποια επέκταση όπως το Spring Security κλπ. Η ρύθμιση μέσω εξωτερικών αρχείων XML αρχίζει και δημιουργεί πονοκεφάλους διότι ο έλεγχος τους πλέον γίνεται πιο δύσκολος.

Το Spring έχει κι εδώ την λύση και λέγεται **annotation-based strategy**. Δίνεται λοιπόν στον προγραμματιστή η δυνατότητα να εισάγει στο Spring Container τα επί μέρους Components χωρίς να τα έχει ορίσει σε κάποιο εξωτερικό αρχείο. Οι συγκεκριμένες τακτικές χρησιμοποιούνται στην δήλωση των Controllers. Παραπάνω είδαμε την εξής δήλωση

```
<bean name="/home" class="gr.teipir.webapp.mvc.HomeController" />
```

Πλέον, μπορούμε να δώσουμε την οδηγία στο dispatcher-servlet.xml

```
<mvc:annotation-driven/>
<context:component-scan base-package="gr.teipir.webapp.mvc" />
```

Δηλαδή τον ενημερώνουμε ότι θα χρησιμοποιήσουμε annotation based στρατηγική και θα πρέπει μέσα από το package gr.teipir.webapp.mvc να φορτώσει στο Container όλους τους Controllers με το annotation στον ορισμό της κλάσης @Controller.

```
@Controller
public class AjaxController
{
   @Autowired
   private GraduateService graduateService;


@RequestMapping(value="/getQObject",method={RequestMethod.POST,RequestM
ethod.GET},produces = "application/json; charset=utf-8")
   public @ResponseBody String handleShowQuestionRequest(HttpServletRequest
request, HttpServletResponse response) throws Exception
   {..}
..}
```

Στον παραπάνω Java κώδικα παρατηρούμε δυο επιπλέον εκφράσεις annotation. Το @Autowired που αυτόματα ψάχνει στο Spring Container και συσχετίζει την παράμετρο με την κατάλληλη τιμή, χωρίς να χρειάζεται να την κάνουμε inject(DI) από το xml αρχείο, καθώς και το @RequestMapping, όπου αποτελεί οδηγία για την κλήση που θα πραγματοποιήσει ο χρήστης στην εφαρμογή και πως θα την διαχειριστεί το σύστημα. Έχουμε λοιπόν απαλείψει σημαντικά κομμάτια κώδικα το οποία θα πρόσθεταν επιπλέον όγκο στα εξωτερικά αρχεία ρύθμισης των αντικειμένων.

2.2.2 Hibernate (ORM)

Το Hibernate αποτελεί ένα ORM σύστημα ανοιχτού κώδικα κάτω από την άδεια LGPL v2.1. Συστήθηκε το 2001 και αναπτύσσεται από την JBoss Community. Συχνά μεγάλες ή μη Enterprise εφαρμογές έχουν την ανάγκη να επικοινωνούν, να διαβάζουν και να γράφουν πληροφορίες σε βάσεις δεδομένων. Οι διαδικασίες αυτές είναι αυτοματοποιημένες και δεν αποτελούν καθαρά κομμάτι της ανάπτυξης του Business Logic. Έτσι λοιπόν, δεν υπάρχει νόημα ο προγραμματιστής να αλλοιώνει πολύτιμο χρόνο στην υλοποίηση, συντήρηση και απασφαλμάτωση τέτοιων καταστάσεων.

Ένα άλλο αναγκαίο κομμάτι όσον αφορά την επικοινωνία με τη βάση δεδομένων, είναι η πιστοποίηση ότι όλα βαίνουν σωστά. Αυτό πρακτικά σημαίνει, όταν θέλουμε να καταγράφουμε την πληροφορία, τότε αυτή πρέπει να καταγράφεται ολόκληρη και όχι τμηματικά, ειδικά όταν ενημερώνουμε πολλούς πίνακες σε ένα Transaction με τη βάση.

Επιπλέον, ας σκεφτούμε τι γίνεται όταν μια ομάδα προγραμματιστών υλοποιεί ένα μεγάλο έργο, τότε κάποιος είναι υπεύθυνος για την βάση. Όταν λοιπόν π.χ. πρέπει να προστεθεί ένα καινούργιο πεδίο ή κάποιος καινούργιος πίνακας, ή να αλλάξει κάποιο πεδίο το data type, τότε η συντήρηση αποκτά δύσκολο έργο, και γίνεται ακόμα πιο δύσκολο σε φάση ανάπτυξης της εφαρμογής και ειδικά όταν υπάρχουν πολλοί υπεύθυνοι για το ίδιο πόστο.

Τέτοιες ανάγκες, αλλά και επιπλέον διευκολύνσεις έρχεται να λύσει το Hibernate. Το Hibernate αποτελεί ένα βαρύ framework το οποίο διαχειρίζεται την βάση δεδομένων σε προγραμματιστικό επίπεδο, γενικά δεν αποτελεί καλή στρατηγική για μικρές σε όγκο εφαρμογές διότι προσθέτει φόρτο δυσανάλογο με την ίδια την εφαρμογή. Πρακτικά αποτελεί ένα στρώμα το οποίο κάθεται μεταξύ της βάσης και της κύριας εφαρμογής, αποκρύπτοντας από τον προγραμματιστή όλες εκείνες τις διαδικασίες για την επικοινωνία και ανταλλαγή δεδομένων με τη βάση. Βέβαια ο προγραμματιστής, έχει την δυνατότητα αν το επιθυμεί να μπει στο πιο κάτω επίπεδο σε περίπτωση που οι ενέργειες που θα πράξει αποτελούν κάτι πιο πολύπλοκο.

Το Hibernate αποτελεί ένα ORM(Object-Relational Mapping) μοντέλο. Η βάση δεδομένων, ανεξαρτήτου λογισμικού, δομείται και διαχειρίζεται σε προγραμματιστικό επίπεδο. Η βασική μονάδα λειτουργίας της ο πίνακας, μεταφράζεται σε κλάση για το σύστημα. Οι παράμετροι της κλάσης αποτελούν τα πεδία του πίνακα όπου με κατάλληλες οδηγίες και ανάλογα με την βάση που θα χρησιμοποιηθεί, τα Java data types μεταφράζονται σε Database data types.

Για την εισαγωγή του Hibernate Framework στην εφαρμογή, δεν χρειάζονται υπεράνθρωπες προσπάθειες! Αρκεί μόνο να εισάγουμε τις κατάλληλες βιβλιοθήκες και να ενημερώσουμε το σύστημα με κάποιο XML mapping για τις κλάσεις/πίνακες που θα χρησιμοποιήσουμε. Όπως και το Spring, λειτουργεί με βασική μονάδα τα Pojo, όπου υλοποιεί διαδικασίες Persistence Strategy για την συνδιάλεξη με την βάση, με μόνη προϋπόθεση τον κενό από ορίσματα κατασκευαστή της κλάσης.

- Με τον όρο Persistence, εννοούμε την «επίμονη» σύνδεση του αντικειμένου μας με κάποιον πίνακα της βάσης. Πρακτικά, το αντικείμενο αποτελεί τον ίδιο τον πίνακα, ενώ οποιεσδήποτε αλλαγές συμβαίνουν στο ίδιο το αντικείμενο, μπορούν με διαδικασίες να ενημερώσουν αντίστοιχα και την βάση.

Hibernate & Annotations

Για την ταχύτερη ανάπτυξη και μείωση του όγκου εξωτερικών ρυθμίσεων των αντικειμένων, το Hibernate προσφέρει την δυνατότητα υλοποίησης με annotations. Η χρήση τους γενικά αποτελεί καλή στρατηγική καθώς κάνει πιο ευέλικτο το προς ανάπτυξη σύστημα καθώς και εύκολα συμβατό με άλλες EJB ORM εφαρμογές. Παρακάτω παρουσιάζεται ένα παράδειγμα χρήσης του Hibernate με annotations.

```
import javax.persistence.*;

@Entity
@Table(name="DOMAIN")
public class Domain implements Serializable
{
   @Id
   @Column(name = "id")
   private long Id;

   @Column(name="url", nullable = false, columnDefinition="text")
   private String url;

   @Column(name = "performed", nullable = false)
   private int performed = 0;
..
```

```
}
```

Αρχικά δηλώνουμε ότι η κλάση αποτελεί ένα persistence entity με το @Entity annotation και ακριβώς αποκάτω δηλώνουμε ποιος είναι ο αντίστοιχος πίνακας όπου γίνεται το mapping με τη βάση δεδομένων. Κάθε ορισμός @Entity πρέπει να έχει ένα πεδίο πρωτεύον κλειδιού για την βάση. Αυτό δηλώνεται με το @Id annotation. Επιπλέον στην κλάση μας έχουμε τις παραμέτρους με τα δικά τους data types. Ακριβώς από πάνω τους, δηλώνουμε με το @Column το όνομα του πεδίου του πίνακα για την βάση. Κατά την δήλωση αυτή, μπορούμε να εισάγουμε επιπλέον οδηγίες, όπως το επιθυμητό όνομα του πεδίου, τον τύπο του, αν αυτός θέλουμε να είναι κάτι πιο ειδικό, αν επιτρέπεται να είναι κενός, μοναδικός κλπ.

Σε μια σωστά δομημένη βάση όμως, είναι λογικό να υπάρχουν συσχετίσεις μεταξύ των πινάκων με ορισμένα ξένα κλειδιά. Το Hibernate επιτρέπει αυτή τη λειτουργία και μάλιστα, σε προγραμματιστικό επίπεδο, φροντίζει τα ξένα κλειδιά να αντιπροσωπεύουν τα ίδια τα αντικείμενα από τους ανάλογους πίνακες. Παρακάτω δίνεται ένα τέτοιο παράδειγμα.

```
@Entity
@Table(name="DOMAIN")
public class Domain implements Serializable
{
  …
  @OneToMany(mappedBy="domain", fetch= FetchType.EAGER)
   private List<DomainPage> domainPages = new ArrayList<DomainPage>();
  …
}

@Entity
@Table(name="DOMAIN_PAGE")
public class DomainPage implements Serializable
{
  …
  @ManyToOne
   @JoinColumn(name="domain_id", nullable=false)
   @OnDelete(action= OnDeleteAction.CASCADE)
   private Domain domain;
  …

  @OneToOne(mappedBy="page", fetch= FetchType.EAGER)
   private MetaKeywords metaKeywords;
  …
}
```

Όπως βλέπουμε από τον παραπάνω κώδικα, το Hibernate συσχετίζει τους πίνακες, ή ακόμα καλύτερα τις ίδιες τις εγγραφές με στιγμιότυπα αντικειμένων. Ο προγραμματιστής δηλαδή, δεν έχει παρά μόνο να διαχειριστεί όπως η ίδια η εφαρμογή επιζητεί, τα παραγόμενα αυτά στιγμιότυπα. Αρχικά παρατηρούμε την οδηγία @OneToMany. Το Hibernate διαθέτει τέσσερις τέτοιες οδηγίες OneToOne, OneToMany, ManyToOne και ManyToMany. Πρακτικά αυτές οι οδηγίες επισημαίνουν τον τρόπο διασυνδέσεις τον πινάκων με τα ξένα κλειδιά και μεταξύ τους. Με την εντολή fetch= FetchType.EAGER δηλώνουμε ότι κατά την αρχικοποίηση του αντικειμένου, θα πρέπει ταυτόχρονα να έχουμε ενημερώσει και την λίστα με τα συσχετιζόμενα αντικείμενα της βάσης. Στη δεύτερη κλάση, σημαντικό είναι να παρατηρήσουμε την οδηγία @JoinColumn(name="domain_id", nullable=false) και @OnDelete(action= OnDeleteAction.CASCADE). Με την πρώτη δηλώνουμε το πεδίο του ξένου κλειδιού της βάσης ενώ με το δεύτερο ενημερώνουμε την βάση για καθαρισμό συσχετίσεων, άρα και εγγραφών του πίνακα, σε περίπτωση διαγραφής της εγγραφής που σχετίζεται το ξένο κλειδί.

Τέλος, ίδια εύκολη αποτελεί και η προσθήκη, ανανέωση και ζήτηση δεδομένων από την βάση καθώς συνεχίζεται η προγραμματιστική έκφραση των εγγραφών της βάσης.

Για την προσθήκη

```
public void add( Domain domain ) {
   Transaction tx = session.beginTransaction();
   session.save( domain );
   tx.commit();
   session.close();
}
```

Για την ζήτηση

```
public Domain fetch( long Id ) {
   Transaction tx = session.beginTransaction();
   Criteria criteria = session.createCriteria( Domain.class );
   Criterion c = Restrictions.eq( "id", new Long( Id ) );
   criteria.add( c );
   Domain dom = (Domain) criteria.uniqueResult();
   tx.commit();
   session.close();
   return dom;
}
```

2.2.3 PostgreSQL Database

Η PostgreSQL αποτελεί ένα ORDBMS (object-relational database management system) σύστημα ανοιχτού κώδικα κάτω από την άδεια PostgreSQL License.

Αναπτύσσεται προγραμματιστικά από την PostgreSQL Global Development Group και υποστηρίζει την πλειοψηφία των προτύπων SQL:2011. Η PostgreSQL ή απλά Postgres, είναι διαθέσιμη σε πολλές πλατφόρμες όπως Linux, Windows, Mac OS X, Solaris κ.α.

- Ένα ORD(**object-relational database**) ή ORDBMS(**object-relational database management system**) αποτελεί ένα σύστημα διαχείρισης βάσης δεδομένων αλλά με αντικειμενοστραφής έκφραση όπου κλάσεις, αντικείμενα και κληρονομικότητα είναι διαθέσιμα κατά την υλοποίηση του σχήματος της βάσης ή σε επίπεδο ερωτήσεων. Μια ORDBMS βάση δεδομένων αποτελεί τον ενδιάμεσο μεταξύ των σχεσιακών μοντέλων(RDBMS) και των OODBMS(object-oriented database management system) όπου η βάση αποτελείται από «επίμονα» αντικείμενα(όπως έχει αναφερθεί παραπάνω) υλοποιήσιμα σε μια αντικειμενοστραφείς γλώσσα με κατάλληλο σύστημα (API) υποστήριξης και επεξεργασίας δεδομένων σε προγραμματιστικό επίπεδο.

Η Postgres εξασφαλίζει την ασφάλεια εγγραφής δεδομένων μέσω του **multiversion concurrency control** (MVCC) το οποίο διαχειρίζεται την ταυτόχρονη χρήση της βάσης από πολλές συνδέσεις. Ουσιαστικά, δημιουργεί ένα στιγμιότυπο της βάσης που επιτρέπει της αλλαγές χωρίς όμως να είναι ορατές σε άλλα στιγμιότυπα μέχρι αυτές να επιβεβαιωθούν και να προωθηθούν στο σύστημα. Η διαδικασία αυτή εξαλείφει την ανάγκη δημιουργίας δικλίδων ασφαλείας κατά την συναλλαγή και πιστοποιεί ότι η βάση θα διατηρήσει τις **ACID** (atomicity, consistency, isolation, durability) αρχές και θα λειτουργεί με αποτελεσματικό τρόπο.

- Στην επιστήμη των υπολογιστών, με τον όρο ACID (*Atomicity, Consistency, Isolation, Durability*) εννοούμε την ασφαλή και σταθερή συνδιαλλαγή με την βάση δεδομένων. Η λειτουργία αυτή αποκτά περισσότερο νόημα, όταν οι συνδιαλλαγές είναι ταυτόχρονες από πολλές συνδέσεις ή χρήστες. Στο τέλος κάθε συνδιαλλαγής, η βάση και τα δεδομένα της πρέπει να είναι σε συνεπή μορφή, όπως ακριβώς βρίσκονταν για την υποστήριξη της επόμενης συνδιαλλαγής και για την διασφάλιση της ποιότητας των δεδομένων της.

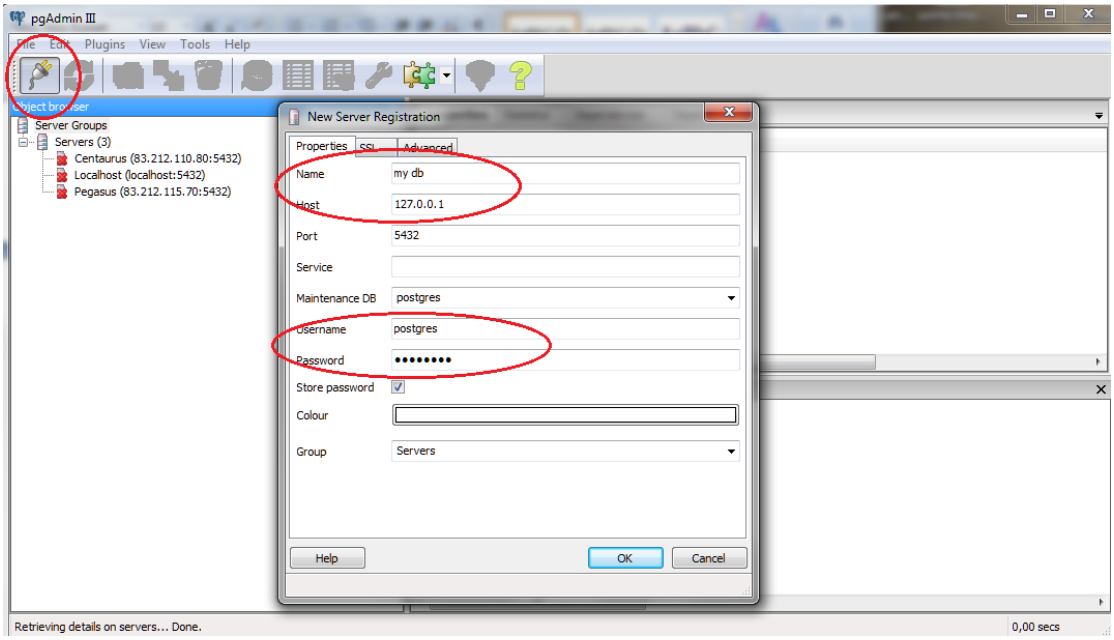
Στη PostgreSQL υπάρχουν τρείς διαθέσιμες γλώσσες για την επικοινωνία με τη βάση. Η απλή και γνωστή σε όλους SQL όπου ο χρήστης μπορεί να υλοποιήσει δικές του συναρτήσεις για την βέλτιστη και ταχύτερη απόδοση των συνδιαλλαγών, την PL/pgSQL η οποία αποτελεί αυτόνομη προγραμματιστική γλώσσα και δίνει την

δυνατότητα για μεγαλύτερο έλεγχο από την απλή SQL και τέλος καθαρή C, που επιτρέπει την φόρτωση βιβλιοθηκών. Η τελευταία, δεν αποτελεί ασφαλή γλώσσα, διότι τυχόν σφάλματα στον κώδικα, έχουν σαν αποτέλεσμα την κατάρρευση της λειτουργίας των συνδιαλλαγών.

pgAdmin

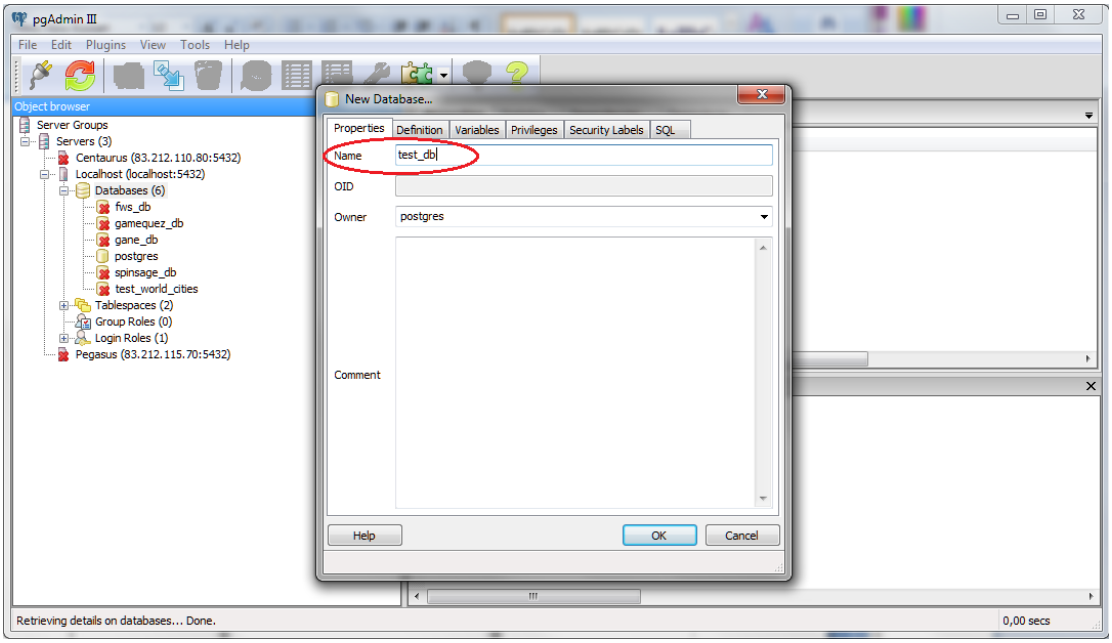
Για το στήσιμο, επεξεργασία και έλεγχος των δεδομένων της βάσης, χρησιμοποιήθηκε η εφαρμογή pgAdmin. Αποτελεί στην ουσία ένα γραφικό stand alone περιβάλλον για την διαχείριση της Postgres βάσης δεδομένων. Μπορεί να συνδεθεί τόσο σε τοπικό μηχάνημα αλλά και απομακρυσμένο, όπως και λειτουργεί για τις ανάγκες της πτυχιακής εργασίας. Παρακάτω παρουσιάζονται κάποια βήματα για την σύνδεση και δημιουργία της βάσης.

- Για την σύνδεση χρειαζόμαστε την από IP του μηχανήματος όπου βρίσκεται εγκατεστημένη η Postgres, το username και το password.



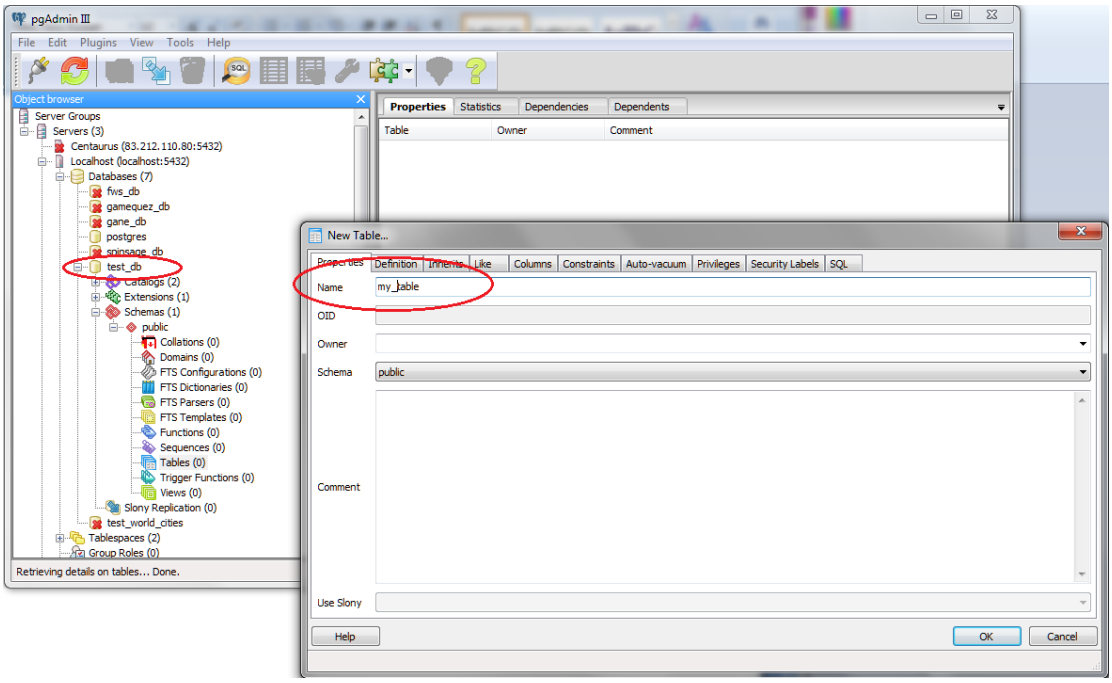
Εικόνα 14. pgAdmin, δημιουργία σύνδεσης

- Δημιουργία βάσης δεδομένων. Υπάρχει επιπλέον η δυνατότητα δημιουργίας βάση με πρότυπα άλλα σχήματα.



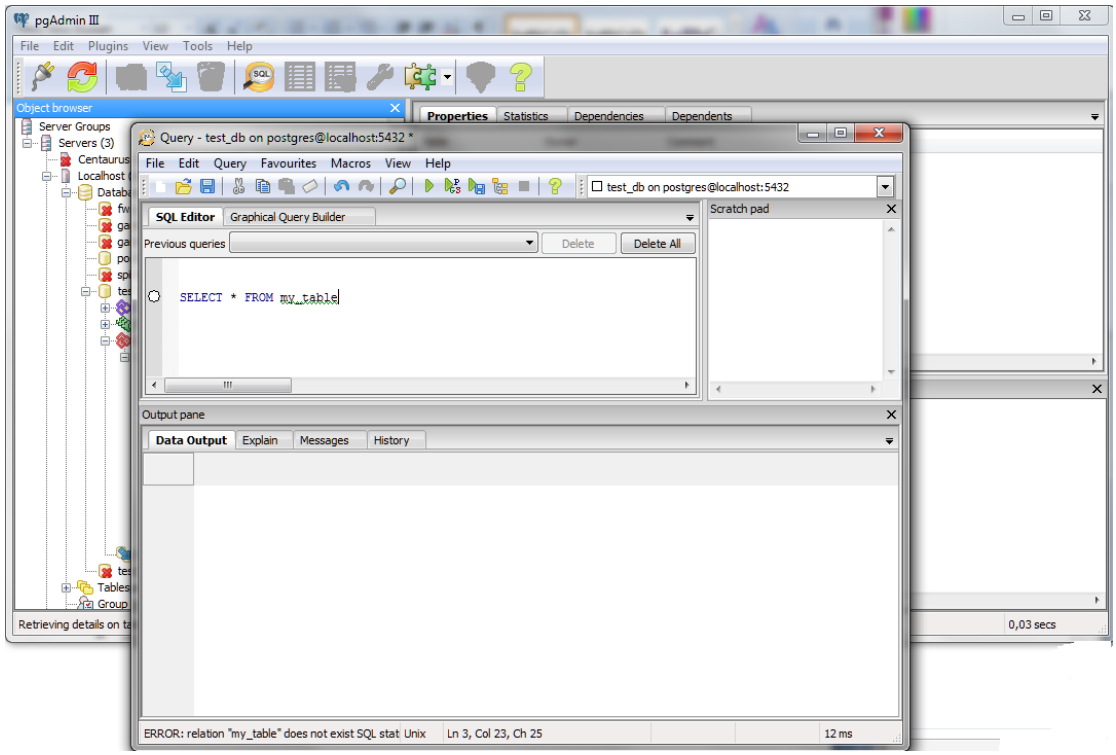
Εικόνα 15. pgAdmin, δημιουργία βάσης.

- Δημιουργία του πίνακα my_table. Δίνεται η δυνατότητα προσθήκης πεδίων, κλειδιών και συσχετίσεις με άλλους πίνακες με ξένα κλειδιά.



Εικόνα 16. pgAdmin, δημιουργία πίνακα.

- Λειτουργία και συνδιαλλαγή με απλή SQL.



Εικόνα 17. pgAdmin, εκτέλεση απλής SQL εντολής.

2.2.4 Apache Tomcat Server

Ο Apache Tomcat Server αποτελεί διαδικτυακό εξυπηρετητή ανοιχτού λογισμικού. Αναπτύσσεται και διανέμεται από την Apache Software Foundation και αποτελεί εξυπηρετητή για Java εφαρμογές καθώς υποστηρίζει πρωτόκολλα Java Servlet και JavaServer Pages. Ο Tomcat δημιουργεί κατάλληλο περιβάλλον για την υποστήριξη διαδικτυακών εφαρμογών αλλά κυρίως λειτουργεί σε βέλτιστη απόδοση, όταν εξυπηρετεί μια μοναδική εφαρμογή. Μπορεί να εγκατασταθεί σε πολλές πλατφόρμες ως μια stand alone εφαρμογή ή ως πηγαίος κώδικας για την εξυπηρέτηση της ανάπτυξης. Για πολλαπλές εφαρμογές και μεγαλύτερες σε όγκο, π.χ. ευρείες Enterprise εφαρμογές , μπορούν να χρησιμοποιηθούν και άλλοι εξυπηρετητές όπως ο JBoss Server, Jetty κ.α. Η ανάλυση του εξυπηρετητή μπορεί να διαχωριστεί σε τρία βασικά κομμάτια.

- Catalina

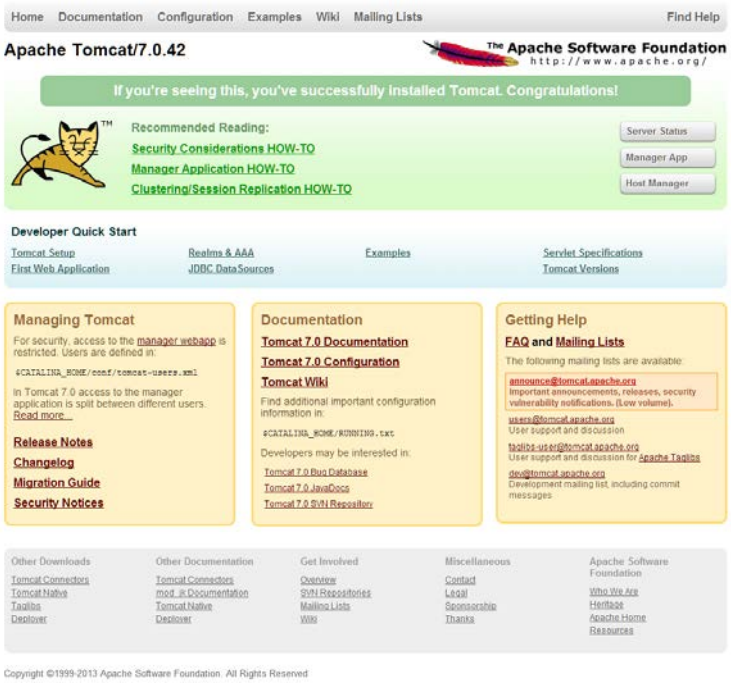
Το catalina αποτελεί τον Servlet Container του Tomcat. Υλοποιεί και υποστηρίζει πρωτόκολλα για Java Servlet και JavaServer Pages(JSP) όπως αυτά εκδόθηκαν από την Sun Microsystems.

- Coyote

Το Coyote αποτελεί την διασύνδεση του Tomcat στην εξυπηρέτηση του HTTP 1.1 πρωτοκόλλου για την υποστήριξη διαδικτυακών εφαρμογών. Υποστηρίζει την επικοινωνία σε πόρτα πρωτοκόλλου TCP και προωθεί τις κλήσεις στο κυρίως λειτουργικό σύστημα του εξυπηρετητή. Τέλος, διαχειρίζεται και είναι υπεύθυνο για την αποστολή απάντησης στον απομακρυσμένο χρήστη.

- Jasper

Αποτελεί την μηχανή διαχείρισης των JSP σελίδων του εξυπηρετητή. Ουσιαστικά, μετατρέπει τα αρχεία JSP, όπου μπορεί να έχουν και κομμάτια HTML κώδικα ή συνδυασμός αυτού με Java, σε compiled αρχεία για την αναγνώρισή τους από τον εξυπηρετητή. Είναι σημαντικό να αναφέρουμε, ότι η διαδικασία αυτή μπορεί να γίνει και σε πραγματικό χρόνο, όταν εντοπισθεί κάποια εξωτερική αλλαγή στον πηγαίο κώδικα, πράγμα που δεν συμβαίνει με τις καθαρά Java κλάσεις. Αυτός είναι και ο λόγος, που μοντέρνοι τρόποι γραφής, προσανατολίζονται κυρίως στις JSP σελίδες αντί σε καθαρή Java, διότι βοηθάει την ταχύτερη ανάπτυξη των εφαρμογών κατά τον έλεγχο και την απασφαλμάτωση (rapid development).



Εικόνα 18. Αρχική σελίδα Tomcat

3. Υλοποίηση

Η εφαρμογή που υλοποιήθηκε για την πτυχιακή εργασία αποτελείτε από τρία βασικά κομμάτια. Το πρώτο είναι το κομμάτι του «service» με ονομασία Funnel Web Spider και αποτελεί την εφαρμογή που είναι υπεύθυνη για την περισυλλογή δεδομένων από τις σελίδες του διαδικτύου. Το δεύτερο κομμάτι αποτελεί το «web interface» που στην ουσία είναι η διεπαφή σε κατάλληλη μορφή για την επικοινωνία των χρηστών από το διαδίκτυο με τα περισυλλεγμένα δεδομένα και έχει ονομασία MSc Finder. Τέλος, ένα κομμάτι το οποίο μένει κρυφό από τους χρήστες και βρίσκει λειτουργία μέσα από την διαδικασία συλλογής των δεδομένων είναι το σύστημα «web scraping» και είναι υπεύθυνο για την απόφαση της ποιότητας των δεδομένων που τελικά θα αποθηκευτούν και αργότερα θα χρησιμοποιηθούν από τους χρήστες. Παρακάτω γίνεται μια πιο εκτενής περιγραφή για τον τρόπο δόμησης και υλοποίησης των παραπάνω συνθετικών της εφαρμογής.

3.1 Υλοποίηση Service, «Funnel Web Spider»

Με τον όρο service, όπως αναφέρει και η ίδια η λέξη, εννοούμε την υπηρεσία που υλοποιήθηκε για την συλλογή των δεδομένων. Αποτελεί την καρδιά της εφαρμογής καθώς η σωστή και σταθερή λειτουργία της βασίζεται αποκλειστικά στην βέλτιστη απόδοση και λειτουργία του Funnel Web Spider. Όπως έχει αναφερθεί και παραπάνω, το service ή ρομπότ, συνθέτει έναν web crawler ή web spider όπου «πηδάει» από σελίδα σε σελίδα και υφαίνει τον Ιστό καταγράφοντας σημαντικά δεδομένα.

Σταθερότητα

Είναι λοιπόν σημαντικό, ο πυρήνας της εφαρμογής να είναι σταθερός, με τα λιγότερα δυνατόν σφάλματα, για να επιτευχθεί και το εξίσου επιθυμητό αποτέλεσμα. Για τον λόγο αυτό, η επιλογή του Spring Framework για την δόμηση της υπηρεσίας αποτέλεσε μονόδρομο. Χάρη στην διάσημη και υποστηριζόμενη από πολλές διαδικτυακές κοινότητες πλατφόρμα, πολλές λειτουργίες που θα χρειάζονταν χρόνο για να υλοποιηθούν αλλά και εξίσου απαραίτητο χρόνο για να ελεγχθεί η ορθότητα της λειτουργίας τους, υλοποιήθηκαν με τον ελάχιστο δυνατό κόπο. Η ελεγμένη σταθερότητα της πλατφόρμας, εφ όσον έχει γίνει σωστή υλοποίηση και έχουν τηρηθεί οι κανόνες για την δόμηση της εφαρμογής, προσφέρει την μέγιστη απόδοση της υπηρεσίας.

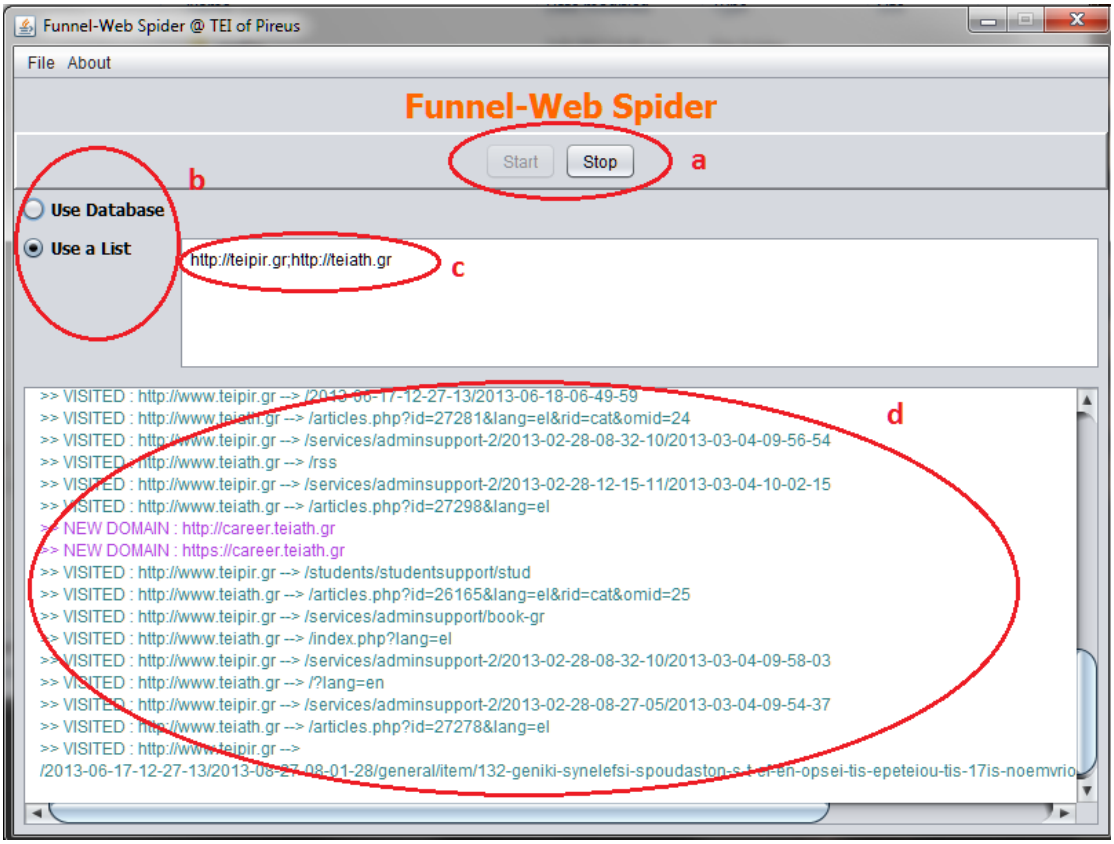
Συντήρηση και επεκτασιμότητα

Ένα από τα ζητούμενα της εφαρμογής, ήταν και το ενδεχόμενο συντήρησης αυτής, αλλά και η επεκτασιμότητα των λειτουργιών της. Ήταν λοιπόν σημαντικό, πολλά δεδομένα χρήσιμα για την λειτουργία της εφαρμογής, να ήταν διαθέσιμα σε αυτή με ένα πιο αφηρημένο τρόπο. Τέτοια δεδομένα, όπως λέξεις κλειδιά για την εύρεση και ταύτιση δεδομένων, αρχεία αρχικοποίησης για την αποφυγή συγκεκριμένων διευθύνσεων, αποφυγή προσπέλασης και καταχώρησης δεδομένων σε μορφή αρχείων, αρχικοποίηση τιμών των λειτουργιών εξωτερικά(Spring Framework) από τις κλάσεις κ.α., ομαδοποιήθηκαν σε εξωτερικά αρχεία τα οποία χρησιμοποιεί η εφαρμογή κατά την έναρξη και αρχικοποίηση της. Με τον τρόπο αυτό, πετυχαίνουμε την επεκτασιμότητα και την συντήρηση της εφαρμογής, ακόμα και χωρίς να πειράξουμε τον πηγαίο κώδικα.

Υλοποίηση

Το service, αποτελεί μια stand alone Java εφαρμογή, υλοποιημένη κάτω από την πλατφόρμα του Spring Framework, με διεπαφή για τον χρήστη βασισμένη σε Java

Swing, η οποία επικοινωνεί με βάση δεδομένων σε PostgreSQL για την συνδιάλεξη και διαμοιρασμό της πληροφορίας.



Εικόνα 19. Funnel Web Spider, service

Στην παραπάνω εικόνα παρουσιάζεται η διεπαφή του Funnel Web Spider για την διαχείριση του χρήστη. Επισημαίνονται τέσσερις βασικές περιοχές.

- a. Τα κουμπιά χειρισμού του service. Μόνο το ένα από τα δύο είναι διαθέσιμο προς επιλογή κάθε φορά. Με το κουμπί start, γίνεται εκκίνηση της διαδικασίας εφόσον είτε στο πεδίο εισαγωγής έχουμε συμπληρώσει επιθυμητές σελίδες για επίσκεψη, είτε από επιλογή χρήσης της βάσης δεδομένων. Το κουμπί stop, σταματάει την διαδικασία τω επισκέψεων, αλλά όχι ακαριαία. Αυτό σημαίνει, ότι αν στην ουρά υπάρχουν σελίδες οι οποίες βρίσκονται υπό ανάλυση, θα πρέπει πρώτα να έχει ολοκληρωθεί η διαδικασία ενημέρωσής τους. Τυχόν σελίδες που ήταν εν δυνάμει για επίσκεψη, διαγράφονται από την ουρά και τερματίζεται η λειτουργία.

- b. Η επιλογή για την χρήση είτε δεδομένων από τη βάση, ύστερα από παρελθοντικές καταχωρήσεις ή λίστα διαχωρισμένα με το σύμβολο «;» τα

οποία πληκτρολογεί ο χρήστης. Το σύστημα αυτό δίνει την δυνατότητα στον χρήστη να κάνει ενημέρωση συγκεκριμένης σελίδας οποιαδήποτε χρονική στιγμή καθώς με την επιλογή «Use database» υπάρχει μέγιστος και ελάχιστος χρόνος για την επίσκεψη και ενημέρωση παλαιότερων εγγραφών.

- c. Το πεδίο όπου ο χρήστης πληκτρολογεί τις επιθυμητές προς επίσκεψη σελίδες. Το πεδίο αυτό δεν είναι διαθέσιμο όταν η επιλογή μετακινηθεί προς χρήση της βάσης.

- d. Παρουσιάζονται χρονικά, τα δεδομένα που προσπελαύνει η υπηρεσία δίνοντας στον χρήστη και μια σύντομη εικόνα των επεξεργασμένων δεδομένων. Χρησιμοποιούνται διαφορετικά χρώματα για την σημείωση των ενεργειών.

❖ Το παραπάνω interface, όπως και ολόκληρη η εφαρμογή, υλοποιήθηκε προγραμματίστηκα σε περιβάλλον Netbeans. Το IDE (Integrated Development Environment) προσφέρει στον προγραμματιστή ειδικό περιβάλλον για την υλοποίηση εφαρμογών με γραφιστικά στοιχεία σε Java Swing. Το Netbeans αποτελεί μια πλατφόρμα ανοιχτού κώδικα, υλοποιημένη σε Java, που υποστηρίζει, εκτός από την προαναφερθείσα γλώσσα επιπλέον PHP, HTML, Css, JavasScript, C/C++ κ.α.

Consumer - Producer

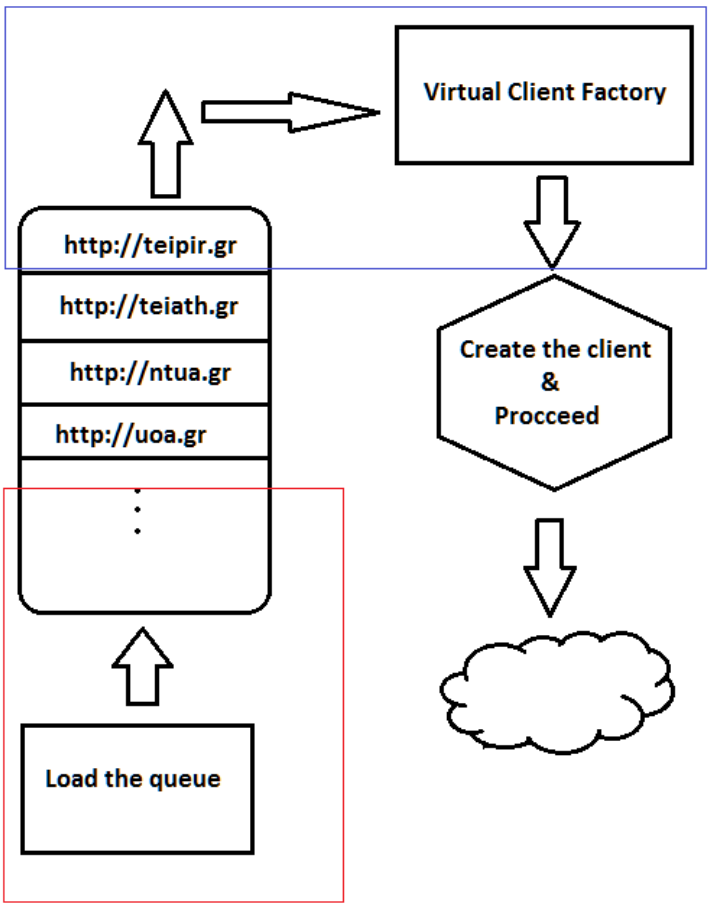
Η προγραμματιστική υλοποίηση του service βασίστηκε στον αλγόριθμο consumer-producer. Στην πραγματικότητα υπάρχουν δύο επίπεδα consumer-producer τα οποία διαχειρίζονται δυναμικά από την εφαρμογή καθώς και από τον ίδιο τον διαχειριστή. Αρχικά η εφαρμογή φροντίζει να «γεμίσει» την ουρά με δεδομένα διαδικτυακών σελίδων, οι οποίες θα προωθηθούν για επεξεργασία. Στη συνέχεια, τα δεδομένα διαχειρίζονται από το πρώτο επίπεδο consumer-producer, στο οποίο δυναμικά, ανάλογα με τις επιλογές του διαχειριστή, υλοποιεί αντικείμενα τύπου “Virtual Clients”. Οι Virtual Clients, αποτελούν ξεχωριστές θεμελιώδεις οντότητες, καθώς παίζουν τον ρόλο του «εξερευνητή» για την σελίδα.

❖ Στην επιστήμη των υπολογιστών, για την δόμηση παράλληλων συστημάτων ώστε να επιτευχθεί ταχύτητα στην επεξεργασία, χρησιμοποιείται η δομή consumer-producer. Πιο απλά αποτελείται από δύο ξεχωριστά τμήματα όπου το ένα έχει σαν διαδικασία να παράγει δεδομένα προς επεξεργασία και να τα

τοποθετεί σε μια κοινή μνήμη, ενώ το δεύτερο τμήμα την επεξεργασία των δεδομένων από την κοινή μνήμη. Σημαντικό στην υλοποίηση του προβλήματος αυτού, είναι ο παραγωγός να μην γράφει δεδομένα όταν η μνήμη είναι γεμάτη, ενώ ο καταναλωτής να μην ζητάει δεδομένα όταν η μνήμη είναι κενή. Για τον συγχρονισμό των δεδομένων, συνήθως χρησιμοποιούνται ουρές(First In First Out) και διαδικασίες αδράνειας των τμημάτων όταν αυτό επιβάλλεται.

Virtual Clients

Ο κάθε Virtual Client αποτελεί ξεχωριστό "thread" για το σύστημα, δηλαδή έπειτα από την αρχικοποίηση του, έχει την δική του «ζωή» και λειτουργία, έως ότου η διαδικασίες του σταματήσουν είτε προγραμματίστηκα, είτε λόγω έλλειψης πόρων. Οι Virtual Clients τροφοδοτούνται με μια ξεχωριστή διεύθυνση για επεξεργασία και ο αριθμός τους παραμένει σταθερός καθ' όλη την λειτουργία της εφαρμογής. Παρακάτω παρουσιάζεται γραφικά, η λειτουργία του πρώτου επιπέδου consumer-producer για την αρχικοποίηση τν Virtual Clients.



Εικόνα 20. Πρώτο επίπεδο Consumer-Producer. Virtual Clients

Όπως παρατηρούμε από την παραπάνω εικόνα, το σύστημα consumer-producer αποτελείται από δύο διαφορετικές οντότητες, έναν παραγωγό(producer) και έναν καταναλωτή. Για την σωστή όμως λειτουργία τους, υπάρχει κοινός σύνδεσμος, η ουρά. Με αυτό τον τρόπο, οι δύο αυτές οντότητες επικοινωνούν και αποφασίζουν την λειτουργία τους. Βασική προϋπόθεση για την ομαλή λειτουργία ενός τέτοιου συστήματος, είναι η μια οντότητα να εξαρτάται όσο το δυνατόν λιγότερη από την άλλη, έτσι συγκεντρώνονται αποδοτικότερα στις δικές τους λειτουργίες εξασφαλίζοντας ποιότητα και ταχύτητα στην όλη εφαρμογή.

Για την ομαλή λειτουργία του παραπάνω επιπέδου, ήταν αναγκαίο να ορισθούν κάποιοι κανόνες και παραδοχές. Το πλήθος των virtual clients ορίζεται δυναμικά από τον χρήστη, οπότε και το πλήθος της ουράς θα πρέπει να ορισθεί ανάλογα. Επιπλέον, είναι σημαντικό να έχουμε πάντα γεμάτη την ουρά ώστε να εξαλείφεται ο χρόνος απόφασης για την επόμενη επίσκεψη. Ουσιαστικά, η εφαρμογή λειτουργεί με παράλληλο τρόπο, καθώς καθ' όλη την διάρκεια επίσκεψης σε διαδικτυακές σελίδες, η εφαρμογή εκτελεί τις κατάλληλες διαδικασίες ώστε να αυξάνεται ο χρόνος απόδοσης της. Τέλος, είναι σημαντικό να λάβουμε υπ όψιν μας, ότι τα δεδομένα μπορεί να ληφθούν από την βάση δεδομένων, η οποία συνεχώς ενημερώνεται κατά την λειτουργία, έτσι, θα πρέπει ο αριθμός των επόμενων επισκέψεων στην ουρά να περιορίζεται, ειδάλλως η φόρτωση στην ουρά εκατομμυρίων σελίδων του διαδικτύου θα σήμαινε ταυτόχρονα και κατάλυση όλων των πόρων του υπολογιστή, άρα και μη σωστή λειτουργία. Μια καλή στρατηγική, είναι το πλήθος της ουράς να είναι 2*n, όπου n είναι το πλήθος των διαθέσιμων Virtual Clients.

Στο δεύτερο επίπεδο, αναφερόμαστε ουσιαστικά στο σύστημα consumer-producer μέσα στον ίδιο τον Virtual Client. Η λειτουργία του client, ξεκινώντας από μία απλή διεύθυνση π.χ. http://www.teipir.gr και αφού επισκεφθεί την πρώτη σελίδα, να δημιουργήσει μια ουρά με τις επόμενες προς επίσκεψη σελίδες από τον ίδιο εξυπηρετητή. Αν φανταστούμε ότι θα πρέπει να γίνει αρχικά μια κλήση στον απομακρυσμένο εξυπηρετητή, να λάβουμε τον HTML κώδικα για επεξεργασία, να τον αναλύσουμε και να βρούμε τις επόμενες επισκέψεις ή καινούργιες διευθύνσεις για επίσκεψη, να ενημερώσουμε σχετικά την βάση δεδομένων και πολλές άλλες διαδικασίες οι οποίες θα πρέπει να γίνουν σειριακά, τότε είναι σίγουρο ότι η εφαρμογή μας, δεν θα ήταν πολύ αποδοτική. Ένα επιπλέον επίπεδο consumer-producer μοιάζει αναγκαίο.

Η λειτουργία λοιπόν του producer αποσκοπεί στην επίσκεψη των σελίδων, στην επεξεργασία του κώδικα όπου θα εξάγει τους επόμενους προς επίσκεψη προορισμούς και στην εισαγωγή δεδομένων στην ουρά για την ενημέρωση της βάσης δεδομένων. Από την άλλη, ο καταναλωτής, λαμβάνει από τη ουρά τα δεδομένα, και εκτελεί κατάλληλες λειτουργίες ενημέρωσης της βάσης. Οι εργασίες λοιπόν μοιράζονται και παράλληλα εκτελούνται διαδικασίες μειώνοντας τον χρόνο προσπέλασης των σελίδων.

Regex Parsing & HtmlCleaner

Για την εξαγωγή δεδομένων που αναφέρονται στις νέες προς επίσκεψη σελίδες χρησιμοποιήθηκαν τεχνικές βασισμένες σε Regular Expression Matching Patterns καθώς και η ανοιχτή βιβλιοθήκη της Apache, HtmlCleaner. Όσο αφορά την δεύτερη, αποτελεί ένα δυνατό εργαλείο για την επεξεργασία και ανάλυση HTML κώδικα. Δυστυχώς, τα λάθη κατά την συγγραφή HTML κώδικα, είτε από προγραμματιστές είναι από πλατφόρμες ανάπτυξης και Συστήματα Διαχείρισης Δεδομένων (CMS), καθιστούν πολλές φορές αδύνατη την εξαγωγή συμπερασμάτων, καθώς πολλοί κανόνες παραλείπονται ή είναι εσφαλμένα δομημένοι. Αυτό θα αποτελούσε σοβαρό μειονέκτημα για μια εφαρμογή, όπως έναν Web Crawler, όπου η βασική του λειτουργία είναι η ανάλυση και επεξεργασία τέτοιων δεδομένων. Με την παραπάνω βιβλιοθήκη λοιπόν, εξασφαλίζουμε ότι σημαντικά λάθη εξαλείφονται και παράγεται ένας δομημένος HTML κώδικας, οποίος είναι πιο φιλικός για επεξεργασία.

Ένα τέτοιο απλό παράδειγμα παρουσιάζεται στον παρακάτω κώδικα

```
<p><a href=”somewhere.html”>go</p>
```

Όπως παρατηρούμε, η κεφαλίδα για το <a> δεν κλείνει σωστά. Αυτό θα μπορούσε να είναι ένα εν δυνάμει πρόβλημα για την μετέπειτα επεξεργασία. Με τον HtmlCleaner, εξασφαλίζουμε ότι ο παραπάνω κώδικας θα διορθωθεί, και πλέον θα έχουμε ένα πιο σωστά δομημένο έγγραφο για την εξαγωγή συμπερασμάτων.

Η σωστή λειτουργία του Web Crawler, βασίζεται στην καθοριστική επιλογή των σελίδων που θα πρέπει να επισκεφτούν. Η ζήτηση σελίδων που έχουν ήδη επεξεργαστεί σημαίνει ταυτόχρονα και χρονική επιβάρυνση στην ολοκλήρωση των διαδικασιών ή και ακόμα εγκλωβισμός σε ατέρμονους βρόχους. Είναι λοιπόν ζωτικής σημασίας, η ικανότητα της εφαρμογής να «καταλαβαίνει» και να διαμορφώνει τα δεδομένα σε συγκεκριμένη μορφή ώστε αυτά να προσπελαύνονται μόνο μία φορά ή όποτε αυτό είναι κανονισμένο να γίνει. Ας υποθέσουμε λοιπόν, ότι έχουμε να επεξεργαστούμε των HTML κώδικα μιας σελίδας. Τα δεδομένα αυτά, αποτελούν έναν μικρό «θησαυρό» καθώς από αυτόν μπορούμε να διαπιστώσουμε αν υπάρχουν αναφορές σε εξωτερικούς συνδέσμους ή σε υποσελίδες του τοπικού εξυπηρετητή.

Αν υποθέσουμε ότι βρισκόμαστε στην σελίδα με διεύθυνση http://example.gr, τότε είναι πιθανόν να αναγνωρίσουμε παραδείγματα όπως τα παρακάτω.

- http://example.gr

- http://www.example.gr
- http://another-exampl.gr
- http://www.another-example.gr
- page.html
- http://example.gr/page.html
- http://www.example.gr/page.html
- http://sub.example.gr

Πολύ εύκολα λοιπόν διαπιστώνουμε, ότι τα παραπάνω είναι κάποια από τα πολλά παραδείγματα που μπορεί να συναντήσουμε. Θα πρέπει λοιπόν να προχωρήσουμε σε μια κανονικοποίηση των παραπάνω δεδομένων, ώστε αυτά όταν αναφέρονται στην ίδια σελίδα, να γράφονται με συγκεκριμένο τρόπο. Για την ομοιομορφία των δεδομένων αυτών την εξαγωγή και διαμόρφωση τους, χρησιμοποιήθηκαν οι παλιές, αλλά όπως αποδεικνύονται, χρήσιμες τεχνικές Regular Expressions ή Regex. Με την τεχνική αυτή, χρησιμοποιούμε χαρακτήρες και συμβολισμούς εμφάνισής τους σε συμβολοσειρές, για την εξερεύνηση ταύτισής τους ή εξαγωγή συγκεκριμένων λέξεων ή αλληλουχίες γραμμάτων.

Τέτοια παραδείγματα είναι τα παρακάτω.

Έλεγχος για διαδικτυακή διεύθυνση

^(?:(http[s]*)://)?(www\\.)?([^/]+)

Έλεγχος αν η σελίδα αναφέρεται στον τοπικό εξυπηρετητή

1.Ελέγχουμε το πρωτόκολλο **^http[s]?://**
2. Εξάγουμε τα επιμέρους τμήματα **^(http[s]?)://(www\\.)?([^/:]+)**
3.Συνθέτουμε καινούργιο regex αντικαθιστώντας τα &1 και &2 με τα παραπάνω τμήματα **^&1://(www\\.)?&2\\/.+**
4.Ελεγχουμε αν η σελίδα αναφέρεται στον εξυπηρετητή.

Robots.txt

Το πλήθος των Web Crawler είναι μεγάλο. Ο καθένας μπορεί να δημιουργήσει έναν ή να κατεβάσει κάποιο αντίστοιχο λογισμικό δωρεάν από το διαδίκτυο. Ταυτόχρονα, αυτό σημαίνει και φόρτος στους εξυπηρετητές οι οποίοι διανέμουν τις σελίδες. Για

τον έλεγχο της χρήσης και ζήτησης σελίδων από τους απομακρυσμένους εξυπηρετητές, σε κάθε έναν, μπορεί να υπάρχει αρχείο με όνομα robots.txt στο root αρχείο κάθε εφαρμογής όπου θα δίνει πληροφορίες για τον τρόπο όπου ο κάθε Crawler θα πρέπει να «συμπεριφέρεται». Οι πληροφορίες αυτές αναφέρουν ποιες σελίδες μπορεί να εξερευνεί η μηχανή, ποιες άλλες απαγορεύεται ή δεν έχει νόημα να εξερευνήσει, τον χρόνο που θα πρέπει να περιμένει ανάμεσα στις κλήσεις ή ακόμα και αν επιτρέπεται η εξερεύνηση σε συγκεκριμένο Crawler.

Υπάρχει λοιπόν, μια άτυπη συμφωνία μεταξύ των μηχανών και των διαδικτυακών εφαρμογών όπου καθορίζουν τους κανόνες συμπεριφοράς της μηχανής για τον κάθε εξυπηρετητή, οι οποίοι ανάλογα με τις δυνατότητες εξυπηρέτησης των κλήσεων ή του φόρτου από άλλους χρήστες, διαμορφώνουν ανάλογα το αρχείο αυτό, ώστε η εφαρμογή να συνεχίσει να είναι αποδοτική στην εξυπηρέτηση της μηχανής αλλά και στις κλήσεις απομακρυσμένων χρηστών.

Βέβαια, όπως είναι εύκολα κατανοητό, η εξέταση και τήρηση του αρχείου robots, δεν αποτελεί αυτονόητη λειτουργία για όλες τις μηχανές. Αν η λειτουργία της μηχανής έχει σκοπό την βίαιη εξαγωγή πληροφοριών, τότε σίγουρα δεν θα λάβει υπ' όψιν της τις οδηγίες από το εκάστοτε αρχείο, με αποτέλεσμα να προκαλέσει προσωρινές δυσλειτουργίες στην ίδια την εφαρμογή.

Στην μηχανή Funnel Web Spider, σαφώς, η λήψη και αυστηρή ακολουθία των οδηγιών από το αρχείο robots.txt, αποτελεί βασική λειτουργία για την εξαγωγή των πληροφοριών. Ακολουθεί την πλέον σύγχρονη μορφή υλοποίησης, καθώς αντιλαμβάνεται ακόμα και τις νέες οδηγίες που έχουν εισαχθεί πρόσφατα. Για την ανάλυση του αρχείου robots, χρησιμοποιούνται τεχνικές parsing με regex όπως φαίνεται και παρακάτω.

Η μηχανή μπορεί να ρωτήσει αν επιτρέπεται η προσπέλαση στην (A)ίδια ή εάν η εφαρμογή είναι διαθέσιμη για όλες τις (B)μηχανές.

A. (?i)[^#]User-agent\\s*:\\s*Funnel-Web-Spider\\s*([^U](.*)\\s*)*
B. (?i)[^#]User-agent\\s*:\\s\\s*([^U](.*)\\s*)***

Για λόγους ανάλυσης και καλύτερης εξέτασης του αρχείου, χρησιμοποιείται η οδηγία **(?i)** και η **\\s***, οι οποίες δηλώνουν ότι δεν γίνεται διαχωρισμός σε κεφαλαία ή πεζά γράμματα και επιτρέπονται κανένα ή πολλά κενά αντίστοιχα. Αυτό γίνεται για να επεξεργασθούν ακόμα και λανθασμένες δομές στα αρχεία, παρ' όλο που η δομή τους είναι αυστηρά καθορισμένη.

Τέλος, σημαντικό είναι να αναφέρουμε μια από τις καινούργιες οδηγίες στα αρχεία robots.txt την crawl-delay η οποία αναφέρεται στην συχνότητα ζήτησης σελίδων από την μηχανή εξερεύνησης. Στην δική μας μηχανή, ο αρχικός χρόνος αναμονής είναι 3 sec, το οποίο όμως μπορεί να ενημερωθεί από εξωτερικό αρχείο ρυθμίσεων. Την

οδηγία αυτή, ακολουθούν πλέον όλες οι σύγχρονες μηχανές. Παρακάτω, παρουσιάζεται και ο τρόπος εξαγωγής της από το Funnel Web Spider.

```
(?i)[^#]Crawl\-delay\\s*:\\s+(.*?(?=\#)|(.*))
```

3.2 Υλοποίηση Web interface, «MSc Finder»

Όπως είναι εύκολα κατανοητό, ένα ρομπότ το οποίο συλλέγει δεδομένα από το διαδίκτυο και τα αποθηκεύει σε μια βάση δεδομένων, δεν θα είχε χρηστική λειτουργία αν οι ενδιαφερόμενοι χρήστες, δεν θα μπορούσαν να είχαν διαθέσιμα αυτά τα δεδομένα. Επίσης, η αποθήκευση των δεδομένων σε συγκεκριμένες μορφές θα ήταν προσβάσιμα μόνο από τους γνώστες της πληροφορικής. Μια μηχανή αναζήτησης λοιπόν θα έχανε και την αξία της αν όλα όσα περιγράφηκαν παραπάνω δεν υλοποιούνταν σε μια μορφή όπου ο καθημερινός χρήστης του διαδικτύου, γνώστης ή μη, θα έβρισκε με ευκολία και θα εξυπηρετούσαν τους σκοπούς του.

Στην ενότητα αυτή, περιγράφεται η υλοποίηση της διεπαφής του χρήστη για την εξαγωγή δεδομένων όπου έχει προ-αποθηκεύσει ο Web Crawler, στη δική μας περίπτωση ο Funnel Web Spider και πως αυτά τα δεδομένα παρουσιάζονται με ευπαρουσίαστο και κατανοητό τρόπο στο τελικό «καταναλωτή».

Μοντέλο MVC & Tiles

Η διαδικτυακή εφαρμογή που φιλοξενεί την ευρύτερη υλοποίηση ενός Crawler, του Funnel Web Spider και παρουσιάζει τα δεδομένα του, ονομάζεται MSc Finder. Όπως αυτό γίνεται αντιληπτό από τον τίτλο της, η μηχανή αναζήτησης ασχολείται με δεδομένα που αφορούν μεταπτυχιακά προγράμματα. Για τον τρόπο εξαγωγής τους, θα αναφερθούμε στην επόμενη ενότητα.

Ο βασικός πυρήνας της web εφαρμογής, είναι υλοποιημένος στην πλατφόρμα Spring MVC, το οποίο υλοποιεί ένα μοντέλο MVC(Model View Controller), όπως έχουμε ήδη αναφερθεί στην προηγούμενη ενότητα. Στην πραγματικότητα, η διεπαφή αυτή, είναι «χαζή». Με την λέξη αυτή εννοούμε ότι δεν χρειάζεται να υλοποιήσει πολύπλοκους αλγορίθμους ώστε να εξάγει συμπεράσματα και αποτελέσματα, αλλά διαχειρίζεται τα ήδη προ-επεξεργασμένα από τον Crawler, και έχει σαν κύριο σκοπό, την παρουσίαση των αποτελεσμάτων αναζήτησης στους χρήστες.

Για την λειτουργία της, χρησιμοποιούνται δύο Controllers οι οποίοι αντιστοιχούν στις αντίστοιχες σελίδες(βλέπε παρακάτω). Η δομή της σελίδας είναι βασισμένη σε «tiles» ή όπως θα λέγαμε πιο απλά, σε «κομμάτια». Η τεχνική αυτή περιγράφει μια βέλτιστη

υλοποίηση ιστοσελίδων όπου στην πραγματικότητα κάθε περιοχή της σπάει σε μικρότερα κομμάτια και μπορεί το κάθε κομμάτι να χρησιμοποιηθεί και σε άλλες σελίδες. Με τον τρόπο αυτό, πετυχαίνουμε ευελιξία και ταχύτητα στην ανάπτυξη της εφαρμογής, καθώς αποφεύγουμε να χρησιμοποιούμε τον ίδιο HTML κώδικα σε πολλά σημεία της σελίδας μας. Έτσι, όταν θέλουμε να την τροποποιήσουμε ή να την διορθώσουμε, δεν χρειάζεται να ψάχνουμε όλη την εφαρμογή μας, απλά κάνουμε τις αλλαγές σε συγκεκριμένα σημεία και αυτό έχει καθολική επιρροή.

SpEL

Σε ένα μοντέλο MVC, τέλος, είναι σημαντικό να αναφέρουμε, ότι τα δεδομένα που παράγονται από το backend, μπορούν εύκολα να «περάσουν» και να διαχειριστούν από το frontend. Παρακάτω παρουσιάζεται πώς από έναν Controller μπορούμε να περάσουμε δεδομένα στο View και τελικώς, αυτά να παρουσιάζονται στον χρήστη.

```
@RequestMapping({"/quest"})
  public ModelAndView handleLoginRequest(HttpServletRequest request, HttpServletResponse response) throws Exception
  {
     ModelAndView mav = new ModelAndView("quest");
     try
     {
        String query = request.getParameter("q");
        mav.addObject( "query", query );
        mav.addObject( "qobjects", searchService.search( query ) );
     }
     catch(Exception e)
     {
        super.write( e );
     }
     finally
     {
        return mav;
     }
  }
```

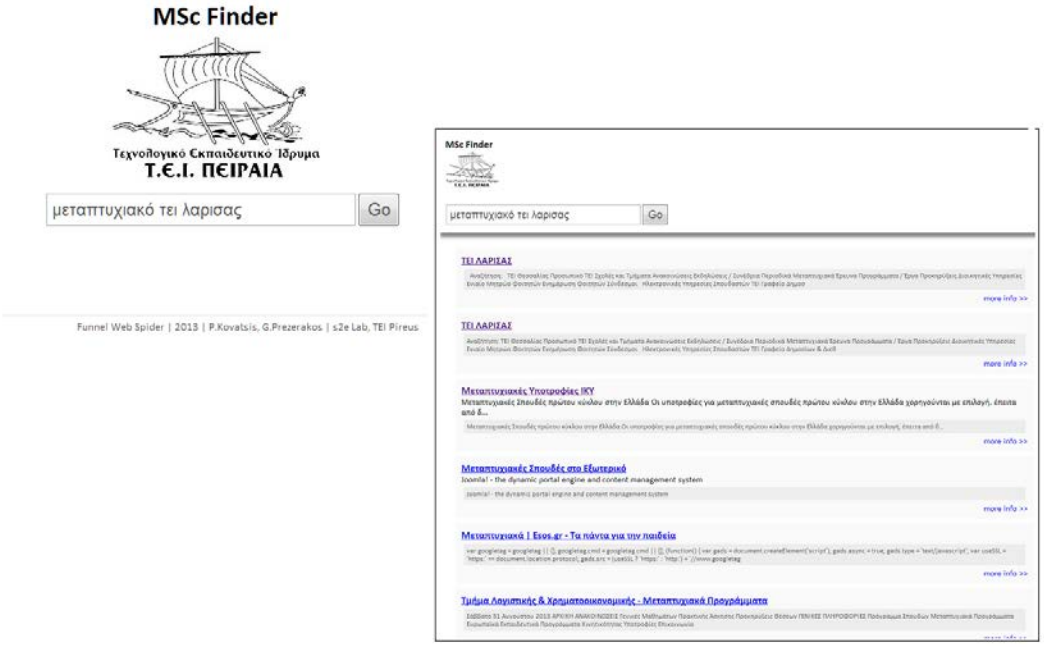
Με τον παραπάνω σύντομο κώδικα παρουσιάζουμε πως «κρύβουμε» τις λειτουργίες κάθε επιπέδου από τα υπόλοιπα επίπεδα αλλά και τον τρόπο σύνδεσης μεταξύ τους. Εδώ λοιπόν κρύβεται και η μαγεία! Όλα είναι τόσο αυτόνομα ώστε να έχουν δική τους διαχείριση αλλά ταυτόχρονα συνδέονται και να εξασφαλίσουν την ύπαρξη τους.

Από την άλλη μεριά λοιπόν μπορούμε να χρησιμοποιήσουμε SpEL(Spring Expression Language) όπου τα δεδομένα λαμβάνονται σε προσπελάσιμη μορφή ώστε να απεικονιστούμε με απλό HTML κώδικα.

```
<div>
   <c:forEach var="result" items="${qobjects}">
      <div style="margin: 10px; padding: 10px; border:1px solid #999;">
         <h3>
            <a                    href="${result.getDomain().getUrl()}${result.getPath()}"
target="_blank">
               <c:choose>
                  <c:when test="${empty result.getPageContent().getTitle()}">
                     ${result.getDomain().getUrl()}
                  </c:when>
                  <c:otherwise>
                  …
```

Οι δυο σελίδες

Η διαδικτυακή εφαρμογή αποτελείτε από δύο βασικές σελίδες, όπου για την υλοποίηση τους χρησιμοποιήθηκαν τεχνολογίες HTML, CSS και JavaScript. Παρακάτω παρουσιάζεται η πρώτη σελίδα, η οποία δίνει κάποιες βασικές πληροφορίες για την εφαρμογή, και έχει μια φόρμα ελεύθερης αναζήτησης για τα δεδομένα.



Εικόνα 21. Αρχική σελίδα και σελίδα αποτελεσμάτων

Για τις ασύγχρονες κλήσεις στην παρουσίαση των επιπλέων πληροφοριών κάθε αποτελέσματος, χρησιμοποιήθηκε η βιβλιοθήκη jQuery. Στην ουσία, αποτελεί μια επέκταση των συναρτήσεων του πυρήνα της JavaScript, για την δημιουργία εργαλείων για την ευκολότερη διαχείριση των στοιχείων μιας HTML σελίδας.

Σημαντικό είναι να αναφέρουμε, ότι η υλοποίηση σε JavaScript ακολούθησε πρότυπα αντικειμενοστρέφειας για την υλοποίηση των κατάλληλων κλάσεων και διαδικασιών σε επίπεδο Client. Τα παραπάνω, υλοποιήθηκαν με το Modular Design Pattern όπου εξασφαλίζεται η μοναδικότητα και η προστασία των κλήσεων.

Apache Lucene

Το Apache Lucene αποτελεί σύστημα ανοιχτού λογισμικού υλοποιημένο σε γλώσσα προγραμματισμού Java, κατάλληλο για την αναζήτηση δεδομένων σε μορφή κειμένου. Ιδανικό για υλοποίηση και εισαγωγή σε Enterprise συστήματα καθώς μπορεί να υποστηρίξει εφαρμογές σε πολλές γλώσσες προγραμματισμού όπως Delphi, Perl, C#, C++, Python, Ruby, και PHP. Η αρχική έκδοση ξεκίνησε το 1999 από τον Doug Cutting και το 2001 έγινε μέλος της οικογένειας ανοιχτού λογισμικού της Apache. Από το 2005 αποτελεί ένα από τα σημαντικότερα προϊόντα ανοιχτού λογισμικού της εταιρείας καθώς τώρα βρίσκετε στην έκδοση 4.4.

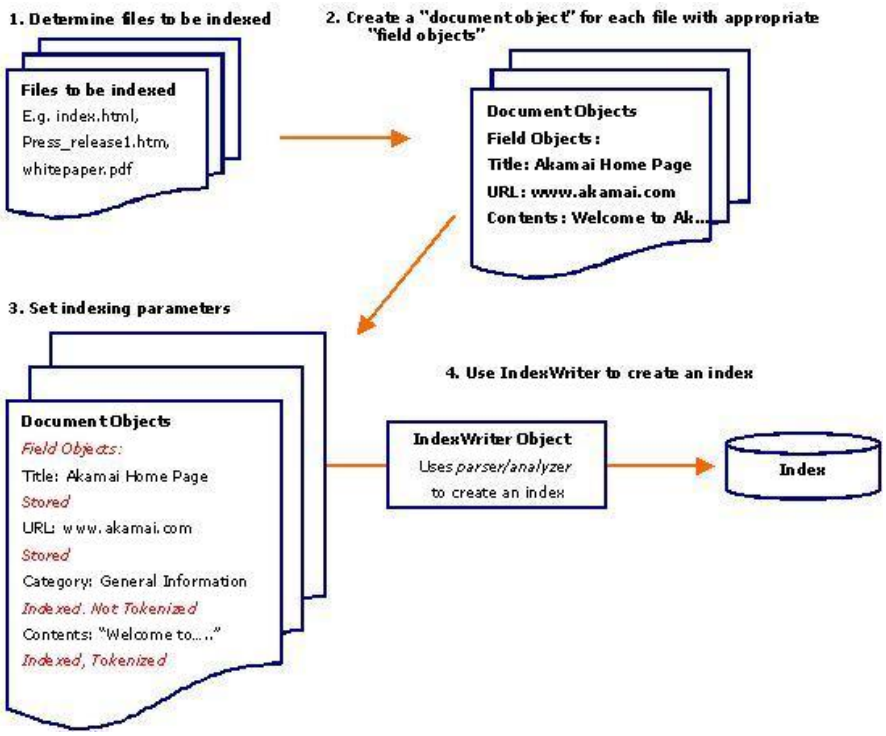
Στην ουσία το Lucene, αποτελεί πλατφόρμα για την υλοποίηση μιας μηχανής αναζήτησης σε προγραμματιστικό επίπεδο. Δεδομένα όπως κείμενα και φράσεις ομαδοποιούνται σε αρχεία, τοποθετημένα στο σύστημα αρχείων ύστερα από κατάλληλη επεξεργασία κι έπειτα προσπελαύνονται για την εύρεση στοιχείων και παρουσιάζονται ιεραρχικά με σύστημα βαθμολόγησης των αποτελεσμάτων.

Αρχειοθέτηση Στοιχείων

Ένα από τα μεγαλύτερα πλεονεκτήματα του Lucene είναι η επεξεργασία μεγάλου όγκου κειμένων και αποθήκευσής του σε συγκεκριμένη (indexing) μορφή για την γρηγορότερη αναζήτηση. Ακατέργαστες προτάσεις, όπως είναι λογικό, θα πρόσθεταν σημαντικό χρόνο στην διαδικασία της αναζήτησης, χρησιμοποιώντας όμως ευρετήρια σε λέξεις, οι οποίες τροποποιούνται για την εξοικονόμηση χώρου αλλά και του μεγέθους επεξεργασίας, μειώνουν τον χρόνο αναζήτησης σε μερικά milliseconds. Παρακάτω παρουσιάζεται ο τρόπος δόμησης των δεδομένων για το Lucene.

Η αρχή λειτουργίας του Lucene βασίζεται στην αναζήτηση και δημιουργία ευρετηρίων(Searching & Indexing). Η ταχύτητα του βασίζεται στο γεγονός ότι δεν αναζητεί λέξεις, αλλά ευρετήρια. Τα ευρετήρια έχουν μικρότερο μέγεθος και πολλές φορές είναι τόσο «παραμορφωμένα», όπου δεν μπορεί κάποιος να βρει σχετική σύνδεση του ευρετηρίου με την ίδια την λέξη. Η ιδέα αυτή μετατρέπει την σελίδο-κεντρική παρουσίαση των δεδομένων σε λεξο-κεντρική ή πιο απλά σε λέξεις κλειδιά.

Το ευρετήριο αποτελεί λοιπόν δομικό στοιχείο για το Lucene. Ένα ευρετήριο(Index) μπορεί να αποτελείτε από ένα ή περισσότερα Documents. Το Document αποτελεί μια αφηρημένη έννοια, χωρίς να προσδίδει την έννοια όπου θα είχε η ίδια η λέξη σε μετάφραση. Στην ουσία, αποτελεί ένα συνδετικό στοιχείο για ένα ή πολλαπλά δεδομένα που θέλουμε να αποθηκεύσουμε και που στη συνέχεια θα είναι διαθέσιμα για αναζήτηση.



Εικόνα 22. Η δομή του Lucene

Για κάθε Document υπάρχουν ένα ή περισσότερα δεδομένα τύπου Field. Τα Fields, αποτελούν απλές δομές με συνδυασμό κλειδιού και τιμής(key – value pair). Το κλειδί εκφράζει το προς αναζήτηση πεδίο ενώ η τιμή, το δεδομένο που θέλουμε να ορίσουμε για αναζήτηση. Το σύστημα αυτό, προσφέρει ευελιξία, καθώς όπως θα δούμε και παρακάτω, δίνει την δυνατότητα για πολύπλοκη αναζήτηση με πολλαπλά κλειδιά!

Η επιτυχημένη αποθήκευση και αναζήτηση των δεδομένων, επιτυγχάνεται με τις δύο υλοποιήσεις του πυρήνα του Lucene, τον IndexWriter και IndexSearcher. Σημαντικό και στις δύο παραπάνω διαδικασίες, είναι να ορίσουμε αφ' ενός το «σημείο» στο σύστημα αρχείων στο οποίο θέλουμε να δημιουργηθεί το περιεχόμενο των ευρετηρίων και αφ' ετέρου την σωστή εφαρμογή τους από το δικό μας σύστημα. Σημαντικό είναι, σε κάθε εφαρμογή να υπάρχει μοναδικός IndexWriter για την εγγραφή των δεδομένων, καθώς αυτό θα αποτρέψει ταυτόχρονες προσπάθειες για εγγραφή στα ίδια αρχεία. Το ίδιο δεν ισχύει για την αναζήτηση με τον IndexSearcher, καθώς το «διάβασμα μπορεί να επιτευχθεί» ταυτόχρονα από πολλούς χρήστες και μάλιστα μια τέτοια υλοποίηση θα αποδώσει μεγαλύτερη ταχύτητα.

Lucene Queries

Η λειτουργία του Lucene θα μπορούσε μακροσκοπικά να παρομοιαστεί με μια απλή βάση δεδομένων. Τα δεδομένα και στις δύο περιπτώσεις αποθηκεύονται και έπειτα, με κατάλληλες λειτουργίες ανακτούνται για επεξεργασία, προβολή κλπ. Όπως λοιπόν

σε μια βάση δεδομένων τα δεδομένα να ανακτούνται, έτσι και στο Lucene ύστερα από αναζήτηση δεδομένα επιστρέφουν πάλι στον χρήστη. Για να επιτευχθεί αυτό θα πρέπει να υλοποιηθούν Queries(ερωτήσεις) προς τα αποθηκευμένα δεδομένα σε συνδυασμό με όρους(Terms) ώστε το σύστημα να επιφέρει τα σωστά αποτελέσματα. Για να παρουσιάσουμε την αναφερόμενη υλοποίηση με απλά παραδείγματα. Ας υποθέσουμε ότι έχουμε αποθηκεύσει δεδομένα που αφορούν φοιτητές και μαθήματα σε μία σχολή καθώς και ώρες που διεξάγονται κάποια μαθήματα. Για τα παρακάτω παραδείγματα θα χρησιμοποιηθεί το ζεύγος κλειδί-τιμή που αναφέρθηκε πιο πάνω.

Για την εύρεση φοιτητών με όνομα Πέτρος θα μπορούσαμε να κάνουμε το παρακάτω

```
name: Πέτρος
```

Για την αναζήτηση συγκεκριμένου φοιτητή με ονοματεπώνυμο

```
name: “Πέτρος Κοβάτσης”
```

Παρατηρούμε ότι εκφράσεις με περισσότερες λέξεις ομαδοποιούνται με διπλά αυτάκια όπου στην ουσία υλοποιείται η πράξη AND. Αντίθετα θα μπορούσαμε να υλοποιήσουμε και μια OR

```
name:Πέτρος name: Κοβάτσης
```

Για να προσθέσουμε αναγκάστηκα την ύπαρξη πεδίου στην αναζήτηση μπορούμε να χρησιμοποιήσουμε τελεστές «+, -», δηλαδή το

```
+name: Πετρος –name: Κοβατσης
```

Εκφράζεται λεκτικά με την έννοια «θα πρέπει το όνομα να περιέχει την λέξη Πέτρος και όχι την λέξη Κοβάτσης». Σε περίπτωση μη ύπαρξης τελεστή, τότε το σύστημα συμπεριφέρεται με την έννοια χρήσης “Maybe”.

Βέβαια θα μπορούσαμε να ψάξουμε για τον φοιτητή που σίγουρα τον λένε Πέτρο και ίσως παρακολουθεί το εργαστηριακό μάθημα της Java και της Ανάπτυξης Διαδικτυακών εφαρμογών αλλά σίγουρα όχι στις 2 η ώρα για το δεύτερο μάθημα.

```
+name: Πέτρος class: Java (class: “Ανάπτυξης Διαδικτυακών Εφαρμογών” –time: 14.00)
```

Με όλα τα παραπάνω παραδείγματα, μπορούν να προστεθούν και ειδικοί χαρακτήρες όπου έχουν σημαντική σημασία στην αναζήτηση όπως το «*» όπου επιτρέπεται μόνο στο τέλος κάθε λέξης και έχει την έννοια «οτιδήποτε» ύστερα από την λέξη. Το σύμβολο «?» όπου αγνοεί συγκεκριμένη λέξη, δηλαδή η αναζήτηση «Κ?βατσης» θα επιστρέψει οποιαδήποτε λέξη έχει στο ? οποιαδήποτε άλλη λέξη. Την εμβέλεια όπου αναγράφεται μέσα σε «[]», ένα τέτοιο παράδειγμα είναι η ημερομηνία «date:[2006091 TO 20131206]». Τέλος μπορούμε να αναφέρουμε και τα Fuzzy Queries, για την αναζήτηση λέξεων με βάση την απόσταση των γραμμάτων σύμφωνα με τον αλγόριθμο Levenshtein, π.χ. το «class: Διαδικτυακών~2» θα επιστρέψει αποτελέσματα όπως Διαδικτυακών, Διαδικτυακής, Διαδικτυακές κλπ.

3.3 Υλοποίηση Web Scraping

Η λήψη ακατέργαστης πληροφορίας και παράδοσης της σε χρήστες θα πρέπει να είναι συγκεκριμένη και στοχευόμενη. Αυτή θα πρέπει να είναι η βασική προϋπόθεση για την σωστή και αποτελεσματική λειτουργία μιας μηχανής αναζήτησης. Το ίδιο λοιπόν προσπαθεί να επιτύχει και η εν λόγω εφαρμογή. Όπως αναγράφεται και στον τίτλο της εργασίας, η λέξη κλειδί «Έξυπνη» αποτελεί βασικό κριτήριο για την ανάπτυξη της εφαρμογής και εξαγωγή της πληροφορίας. Στην ενότητα αυτή θα παρουσιαστεί ο τρόπος με τον οποίο έγινε ο διαχωρισμός της σε επίπεδα σημαντικότητας.

Scraping

Με τον όρο Web Scraping (εξόρυξη πληροφορίας) εννοούμε τις τεχνικές εκείνες όπου μια μηχανή, συνήθως ένας ηλεκτρονικός υπολογιστής, μπορεί να εξάγει συγκεκριμένη πληροφορία από τις σελίδες του διαδικτύου χρησιμοποιώντας το πρωτόκολλο HTTP. Η σελίδες αυτές είναι συνήθως σε μορφή HTML και επεξεργάζονται από την μηχανή όπως θα έκανε και ένας άνθρωπος. Οι τεχνικές με τις οποίες μπορεί να επιτευχθεί η περισυλλογή πληροφορίας είναι με την παρέμβαση του ανθρώπου, όπου επιφέρει και τα καλύτερα αποτελέσματα. Ο τρόπος αυτός αποτελεί το γνωστό σε όλους «σερφάρισμα» στις διαδικτυακές σελίδες και χειροκίνητη καταγραφή των δεδομένων. Επίσης μπορεί να γίνει με την χρήση προγραμμάτων, όπου επεξεργάζονται τις σελίδες και προσπαθούν να βρουν την κατάλληλη πληροφορία. Η χρήση των μηχανών, προφανώς δεν θα επιφέρει τα ίδια σωστά αποτελέσματα όπως ένα άνθρωπος, αλλά είναι πολλαπλές φορές πιο γρήγορη.

Η χρήση των μηχανών για την υλοποίηση διαδικασιών Scraping μπορεί να διαιρεθεί σε δύο επιπλέον κομμάτια. Το πρώτο είναι πιο ασφαλές ως προς την εξαγωγή πληροφορίας, καθώς οι σελίδες όπου εξερευνούνται έχουν τέτοια μορφή υλοποίησης

όπου η μηχανή την γνωρίζει εξ αρχής. Μπορεί δηλαδή, γνωρίζοντας που βρίσκεται το κάθε στοιχείο του HTML κώδικα, να προσπελάσει και να καταγράψει συγκεκριμένη πληροφορία. Το δεύτερο κομμάτι, είναι μη ασφαλές και πιο δύσκολο στην υλοποίηση, καθώς η δομή της σελίδας δεν είναι γνωστή. Αυτό ταυτόχρονα σημαίνει ότι το σύστημα θα πρέπει να έχει κάποια μορφή νοημοσύνης ώστε να εξάγει τα δεδομένα που επιζητεί εάν αυτά υπάρχουν βέβαια.

Αν σκεφτούμε επίσης ότι, ο κάθε προγραμματιστής υλοποιεί διαφορετικά κάθε σελίδα ή ακόμα και το πλήθος των συστημάτων για αυτόματη παραγωγή σελίδων, όπου ακολουθούν δική τους διαδικασία η οποία αλλάζει ακόμα και με την έκδοση της εφαρμογής, τότε εύκολα διαπιστώνουμε ότι προστίθενται επιπλέον δυσκολίες για την σωστή απόδοση του συστήματος. Θα πρέπει λοιπόν, η εφαρμογή να μην στηρίζεται μόνο στις HTML ετικέτες αλλά και σε καθαρό εξαγόμενο κείμενο.

Για να επιτευχθούν τα παραπάνω, θα πρέπει να γίνει ανάλυση του κειμένου και προσπάθεια εύρεσης στοιχείων όπου θα οδηγήσουν στην λήψη της σωστής απόφασης. Γι τον λόγο αυτό χρησιμοποιήθηκαν λέξεις κλειδιά οι οποίες αποθηκεύονται εξωτερικά από την εφαρμογή και μπορούν να ενημερώνονται από τους διαχειριστές. Οι λέξεις κλειδιά αποτελούν τους βασικούς κόμβους για να ταυτοποιήσουμε αν σε μια σελίδα είτε σε επίπεδο κώδικά είτε σε λογικό επίπεδο κειμένου, περιέχουν δεδομένα τα οποία είναι χρήσιμα. Τις δύο αυτές τεχνικές θα τις αναλύσουμε παρακάτω.

- Μερικές από τις λέξεις κλειδιά

μεταπτυχιακό στη;
μεταπτυχιακό τίτλο στη;
μεταπτυχιακές σπουδές τμήμα;
μεταπτυχιακές σπουδές σχολή;
graduate studies on;
master program on;
master degree in;
…

Stemming

Από τις παραπάνω λέξεις κλειδιά, η εφαρμογή προσπαθεί να εντοπίσει παρουσία τους σε εξαγόμενο κείμενο. Είναι πολύ πιθανό όμως οι λέξεις αυτές να μην εμφανίζονται αυτούσιες αλλά σε άλλο βαθμό (μεταπτυχιακό - μεταπτυχιακά), άλλη κλίση (τμήμα - τμήματος) ή και συνδυασμός αυτών. Θα πρέπει λοιπόν από τις λέξεις κλειδιά να έχουμε την μεγαλύτερη δυνατή γκάμα περιπτώσεων. Για να το πετύχουμε αυτό,

χρησιμοποιήθηκαν στην εφαρμογή τεχνικές Word Stemming σε Ελληνικές αλλά και Αγγλικές λέξεις ή και συνδυασμό αυτών. Με την τεχνική αυτή, αποσπούμε από την λέξη την ρίζα της, έτσι ώστε να διατηρήσουμε την ακεραιότητα και την σημασία της, που δεν συμβαίνει με άλλες τεχνικές όπως π.χ. απαλοιφή φωνηέντων και πλέον ψάχνουμε να βρούμε εμφανίσεις της στο κείμενο. Η τεχνική Stemming υλοποιήθηκε με το Last Suffix Algorithm. Με τον αλγόριθμο αυτόν αλλά και με τους κανόνες που διέπουν την κάθε γλώσσα, μπορούμε να απαλείφοντας γράμματα από το τέλος της λέξεις να φτάσουμε σε μια «αδιαίρετη» μορφή της ή αλλιώς τη ρίζα της. Παρακάτω παρουσιάζεται η επεξεργασία της λέξης «μεταπτυχιακές» με κάποιους από τους βασικούς κανόνες της γλώσσας.

- Η λέξει τελειώνει σε «ς», οπότε αφαιρούμε το γράμμα αυτό.
- Η επόμενη λέξη είναι το «ε» το οποίο είναι φωνήεν, άρα αφαιρείται.
- Έπειτα είναι το «κ» το οποίο είναι σύμφωνο άρα η διαδικασία σταματάει εκεί.

Το αποτέλεσμα λοιπόν είναι η ρίζα «μεταπτυχιακ» όπου δίνει αποτελέσματα για

- Μεταπτυχιακό
- Μεταπτυχιακά
- Μεταπτυχιακές
- Μεταπτυχιακών
- Μεταπτυχιακοί
- Μεταπτυχιακούς

Έχουμε έτσι αυξήσει τις πιθανότητες για καταγραφή σημαντικής πληροφορίας. Η διαδικασία αυτή επαναλαμβάνεται για κάθε προς αναζήτηση λέξη κλειδί! Οι κανόνες που ακολουθήθηκαν ήταν εμπειρικοί και ανασχηματίστηκαν για την καλύτερη λειτουργία της εφαρμογής. Γενικά, τα παραπάνω στάδια εύρεσης της ρίζας, χωρίζονται σε τρία επίπεδα λέξεων από το τέλος της λέξης προς την αρχή της. Ανάλογα με το επίπεδο εμφάνισης κάθε λέξης, αλλά και της επόμενης της, ο αλγόριθμος γνωρίζει που πρέπει να σταματήσει ώστε να αποδώσει το καλύτερο αποτέλεσμα.

- Για το πρώτο επίπεδο και την Ελληνική γλώσσα, έχουμε τις παρακάτω λέξεις σε Java υλοποίηση

```
level1Suffixes.add( "α" );
level1Suffixes.add( "ά" );
level1Suffixes.add( "ό" );
level1Suffixes.add( "ο" );
level1Suffixes.add( "έ" );
level1Suffixes.add( "ε" );
```

```
level1Suffixes.add( "η" );
level1Suffixes.add( "ή" );
level1Suffixes.add( "ί" );
level1Suffixes.add( "ι" );
level1Suffixes.add( "ύ" );
level1Suffixes.add( "υ" );
level1Suffixes.add( "ν" );
level1Suffixes.add( "ς" );
```

- Για το δεύτερο επίπεδο

```
level2Suffixes.add( "ω" );
level2Suffixes.add( "ώ" );
level2Suffixes.add( "ε" );
level2Suffixes.add( "έ" );
level2Suffixes.add( "ό" );
level2Suffixes.add( "ο" );
level2Suffixes.add( "τ" );
level2Suffixes.add( "ύ" );
level2Suffixes.add( "υ" );
level2Suffixes.add( "η" );
level2Suffixes.add( "ή" );
```

- Για το τρίτο επίπεδο

```
level3Suffixes.add( "ό" );
level3Suffixes.add( "ο" );
level3Suffixes.add( "α" );
level3Suffixes.add( "ά" );
```

Τέλος, παρακάτω παρουσιάζεται ο αλγόριθμος σε Java όπου κάνει χρήση των παραπάνω για την εξαγωγή της ρίζας της λέξης.

```
StringBuilder sb = new StringBuilder();
 char clv1 = word.charAt(word.length() - (sb.length() + 1));
 if (level1Suffixes.contains(String.valueOf(clv1))) {
    sb.insert(0, String.valueOf(clv1));
    if( word.length() >= (sb.length() + 1) )
    {
       char clv2 = word.charAt(word.length() - (sb.length() + 1));
       if (level2Suffixes.contains(String.valueOf(clv2))) {
```

```
                sb.insert(0, String.valueOf(clv2));
                if(word.length() >= (sb.length() + 1))
                {
                   char clv3 = word.charAt(word.length() - (sb.length() + 1));
                   if (level3Suffixes.contains(String.valueOf(clv3))) {
                      sb.insert(0, String.valueOf(clv3));
                   }
                }
             }
          }
       }
       sw.setRoot(word.substring(0, word.length() - sb.length()));
```

Scrapping στην HTML

Με τον όρο Scraping στην HTML εννοούμε τις διαδικασίες όπου μέσα από τον κώδικα παρουσίασης προσπαθούμε να εξάγουμε πληροφορία. Η διαδικασία αυτή παίρνει προβάδισμα σε σχέση με την επεξεργασία κειμένου, καθώς η αναζήτηση δεδομένων έχει πιο στοχευόμενη μορφή αφού εξετάζει στοιχεία της σελίδας τα οποία επιφέρουν συνήθως αρκετή και σωστή πληροφορία. Για την εύρεση αυτών των δεδομένων, πραγματοποιήθηκε αρχικά έρευνα σε διαδικτυακές σελίδες Ελληνικών πανεπιστημίων αλλά και ξένων (Αγγλόφωνων), κι έπειτα εξακρίβωση των αποτελεσμάτων με τον ανθρώπινο παράγοντα. Από την έρευνα αυτή, διαπιστώθηκε ότι, οι HTML κεφαλίδες <TITLE> και <H>, μεταφέρουν πληροφορία σχετικά με τον τίτλο του μεταπτυχιακού προγράμματος. Αντίθετα, τα Meta Tags όπως Description και Keywords, παρ' όλο που αναφέρονται σε ξεχωριστές σελίδες, δεν δίνουν σημαντική πληροφορία.

Το σύστημα, βασίζεται στην εύρεση δεδομένων, εφ' όσον το περιεχόμενο της σελίδας είναι σχετικό. Για να το διαπιστώσει αυτό η εφαρμογή, χρησιμοποιεί τα δεδομένα από τις λέξεις κλειδιά. Αυτό αποτελεί και το πρώτο στάδιο για την εξαγωγή της πληροφορίας.

Εφ' όσον η σελίδα περάσει από τον πρώτο έλεγχο, συνεχίζει στην δεύτερη προς επεξεργασία διαδικασία, όπου στο HTML Scrapping αναφέρεται και ως προσαρμοσμένη αναζήτηση στα HTML nodes. Αρχικά, εντοπίζεται ο τίτλος του μεταπτυχιακού προγράμματος και καταγράφεται το HTML node όπου εντοπίστηκε. Σε περίπτωση μη εύρεσης τίτλου προγράμματος τότε η εφαρμογή συνεχίζει χωρίς καταγραφή δεδομένων, ειδάλλως κάνοντας σταδιακά βήματα προς τα πίσω βρίσκει το γονικό node του τίτλου, όπου αποτελεί και πηγή σχετικών πληροφοριών όπως τηλέφωνα, ηλεκτρονικό ταχυδρομείο και σχετικές πληροφορίες. Τέλος οι πληροφορίες αυτές καταγράφονται στη βάση δεδομένων ως μεμονωμένα τμήματα πληροφορίας και παρουσιάζονται αναλυτικά στον χρήστη.

Scrapping σε κείμενο

Το Scrapping σε κείμενο ή αλλιώς δυναμικό Scrapping αναφέρεται σε διαδικασίες όπου μέσα από καθαρό (readable) κείμενο για τον άνθρωπο μπορούμε να εντοπίσουμε την σωστή πληροφορία. Με την ενέργεια αυτή προσδίδουμε μεγαλύτερη νοημοσύνη στην εφαρμογή μας και αυξάνουμε τις πιθανότητες να εντοπίσουμε μεταπτυχιακούς τίτλους σε όλο τον κόσμο. Μαζί όμως με την αύξηση της πιθανότητας αυτής, αυξάνεται και η εμφάνιση λαθών ή μη σωστών αποτελεσμάτων κατά την αναζήτηση.

Η αναγνώριση κειμένου, αποτελεί πρόκληση ακόμα και για μεγάλες εφαρμογές. Δεν είναι μάλιστα τυχαίο όπου κολοσσοί της πληροφορικής στηρίζουν τα βασικά τους προϊόντα σε τέτοιο είδους ενέργειες. Σαν άνθρωποι λοιπόν, μπορούμε δύσκολα να υπολογίσουμε μια αριθμητική πράξη ενώ ένας απλός υπολογιστής μπορεί να την ολοκληρώσει σε μερικά milliseconds. Αντίθετα, μπορούμε πολύ εύκολα, σε κλάσματα δευτερολέπτου να αναγνωρίσουμε ένα γνωστό μας πρόσωπα σε πλήθος ανθρώπων ενώ αντίθετα ακόμα και ένας υπερυπολογιστής θα χρειασθεί αρκετά δευτερόλεπτα και ίσως στο τέλος να μην δώσει και σωστό αποτέλεσμα.

Για τους παραπάνω ευνόητους λόγους, η χρήση δυναμικής εξαγωγής επιφέρει μεν αποτελέσματα, αλλά πιθανών κάποια από αυτά να είναι λανθασμένα!

Εφ’ όσον η εφαρμογή αποφασίσει ότι η σελίδα σχετίζεται με πληροφορία που επιζητούμε, όπως αναφέρθηκε και παραπάνω, τότε «καθαρίζει» όλο τον HTML κώδικα και από αυτόν εξάγει καθαρό πλέον κείμενο. Το κείμενο αυτό επεξεργάζεται όπως θα το έκανε πιθανόν και ο ανθρώπινος εγκέφαλος σειριακά. Αρχικά εντοπίζεται το σημείο όπου βρέθηκε κάποια λέξη κλειδί. Οπτικά, αυτό θα μοιάζει με μία παράγραφο όπου θα έχει κενά και αλλαγή γραμμής από τα παραπάνω και παρακάτω της κείμενα. Μπορούμε λοιπόν με αυτό τον τρόπο να καταλάβουμε που περίπου πρέπει να επικεντρωθούμε για να βρούμε την «κρυμμένη» πληροφορία.

4. Πιστοποιητικό Αναλυτικής Βαθμολογίας (επικυρωμένο).

5. Επιστημονικές δημοσιεύσεις, διακρίσεις, αποδεικτικά επαγγελματικής ή ερευνητικής δραστηριότητας (εάν υπάρχουν).

Το παραπάνω κείμενο αποτελεί αληθινό μορφοποιημένο παράδειγμα επεξεργασίας κειμένου για εύρεση πληροφορίας. Διαβάζοντας το διαπιστώνουμε ότι τα δεδομένα μας βρίσκονται στην πρώτη παράγραφο αφού οι λέξεις κλειδιά ελέγχουν και ταυτίζουν την φράση «Προγράμματος Μεταπτυχιακών Σπουδών». Παρ' όλα αυτά, αυτό δεν σημαίνει ότι η πληροφορία μας είναι σωστή. Μπορεί να βρίσκαμε στοχευμένη πληροφορία, αλλά τελικά να μην είχε σχέση με μεταπτυχιακό πρόγραμμα. Για την επίλυση αυτού του προβλήματος, το σύστημα αποκόπτει την «κρίσιμη» παράγραφο και την στέλνει στο τελικό επίπεδο επεξεργασίας.

Στα πλαίσια του Προγράμματος Μεταπτυχιακών Σπουδών του Τμήματος Νοσηλευτικής του ΤΕΙ Λάρισας, με τίτλο «Ψυχική Υγεία» καλούνται οι ενδιαφερόμενοι υποψήφιοι μεταπτυχιακοί φοιτητές για το ακαδημαϊκό έτος 2013-2014 να υποβάλουν στη Γραμματεία του ΠΜΣ τα παρακάτω δικαιολογητικά.

Στο τρίτο επίπεδο ελέγχου, η παράγραφος εξετάζεται ως λέξεις! Έχοντας βάση μας την θέση όπου βρέθηκε η λέξη κλειδί, αποκόπτουμε από την παράγραφο την πρόταση ή την αλληλουχία λέξεων όπου πιθανόν να εκφράζουν την πληροφορία που επιζητούμε. Πηγαίνοντας διαδοχικά προς τα πίσω κι έπειτα μπροστά, μπορούμε να βρούμε βάση εμπειρικών κανόνων σχετικά με το πλήθος των λέξεων ή σημεία στίξεων του κειμένου, την τελική πληροφορία όπου είναι και ο τίτλος του μεταπτυχιακού προγράμματος.

Τέλος, η παράγραφος και οι υπόλοιπες παράγραφοι, επεξεργάζονται για την εύρεση περισσότερων δεδομένων, τηλεφώνων και e-mails. Για την εξαγωγή των παραπάνω πληροφοριών, χρησιμοποιήθηκε η βιβλιοθήκη HtmlCleaner όπου έχει αναφερθεί σε προηγούμενη ενότητα καθώς και τεχνικές Regex.

- Αποτελέσματα Scraping

Στον παρακάτω πίνακα παρουσιάζονται συγκεντρωτικά τα αποτελέσματα Scrapping της εφαρμογής.

	HTML SCRP	Logic SCRP
Τίτλος προγράμματος	340	93
Τηλέφωνο	72	14
Ηλ. Ταχυδρομείο	57	8
Σύνολο 433 αποτελέσματα	340	93

4. Βελτιστοποίηση – Το μέλλον

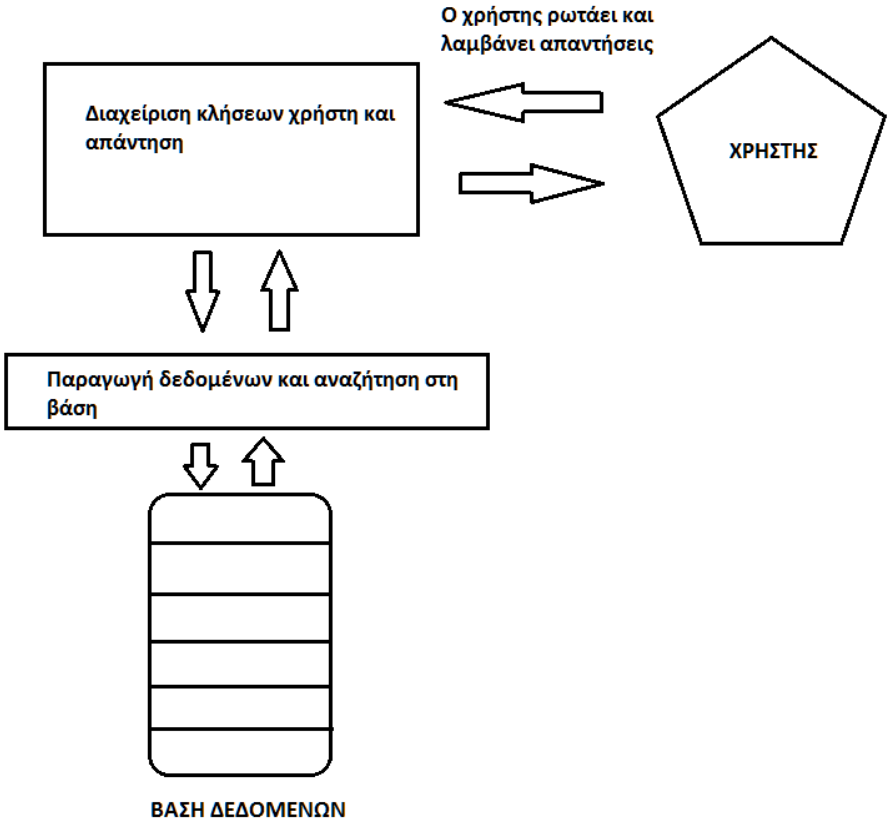
4.1 Βελτιστοποίηση ταχύτητας και αποτελεσμάτων χρήστη

Τα συμπεράσματα που μπορούν να εξαχθούν από την χρήση της εφαρμογής, αποτελούν και την πλέον κατάλληλη διαδικασία για την αξιολόγηση των αποτελεσμάτων της. Όταν ο χρήστης δέχεται σωστά αποτελέσματα τα οποία εξυπηρετούν τους σκοπούς του τότε και η μηχανή αναζήτησης θα έχει επιτύχει τον στόχο της.

Στην επιστήμη των υπολογιστών όμως, ποτέ κάτι δεν τελειώνει και πάντα υπάρχει ο χώρος για βελτιστοποίηση των εξαγόμενων δεδομένων, ειδικά όταν αυτά έχουν παραχθεί με την χρήση μηχανής και διαδικασίες προσομοίωσης των ανθρώπινων ενεργειών. Τα αποτελέσματα λοιπόν, έχουν άμεση σχέση με την αρχική εξαγωγή τους από το σύστημα. Σε επίπεδο User Interface, η βελτιστοποίηση τους αποσκοπεί στην ταχύτητα εξαγωγής τους αλλά και στην εύστοχη απόδοση τους ανάλογα με το σύστημα.

Όπως έχει παρουσιαστεί σε παραπάνω κεφάλαιο, η αναζήτηση του συστήματος βασίζεται στην πλατφόρμα της Apache Lucene. Η έκδοση που χρησιμοποιήθηκε ήταν η 3.6.2. Πλέον η πλατφόρμα διανέμεται στην έκδοση 4.4. Προφανώς μια αναδόμηση της εφαρμογής σε καινούργια έκδοση με πιθανών βελτιστοποιημένους αλγορίθμους να επιφέρει σημαντική μείωση στην ταχύτητα εύρεσης αποτελεσμάτων. Είναι σημαντικό βέβαια να επισημάνουμε ότι η πλατφόρμα του Lucene, ολοκληρώνει τις διαδικασίες της σε μερικά milliseconds όπου πρακτικά είναι μηδενικός χρόνος αντίληψης για τον άνθρωπο, αλλά θα πρέπει πάντα να σκεφτόμαστε ότι μια διαδικτυακή εφαρμογή, εξυπηρετεί εκατοντάδες και ίσως χιλιάδες ταυτόχρονους χρήστες. Αυτό σίγουρα θα απαιτούσε μια οργάνωση όπου οι ερωτήσεις θα έπρεπε να εκτελεστούν ίσως και σε παράλληλη μορφή ή και σε περιπτώσεις να υλοποιηθούν διαδικασίες caching, δηλαδή προσωρινής αποθήκευσης των αποτελεσμάτων.

Η διαδικασία με την οποία η εφαρμογή επιλέγει τα δεδομένα παρουσιάζεται στην παρακάτω εικόνα. Αρχικά ο χρήστης ρωτάει το σύστημα το οποίο με την σειρά του προωθεί κατάλληλη ερώτηση στο Lucene. Το Lucene αναζητεί στα αρχεία του και επιστρέφει κλειδιά δεδομένων της βάσης. Τέλος, για κάθε δεδομένο γίνεται ερώτηση στη βάση για να επιστραφεί το κατάλληλο αντικείμενο.



Εικόνα 23. Διαδικασία λειτουργίας Lucene

Αν φανταστούμε ότι η κάθε ερώτηση χρειάζεται κάποιο χρόνο, τότε ο χρόνος αυτός πολλαπλασιάζεται σύμφωνα με το πλήθος των δεδομένων της κάθε απάντησης! Μια έξυπνη υλοποίηση σε αυτό, θα ήταν η αποθήκευση ενεπεξέργαστων δεδομένων στο ίδιο το Lucene, όπου θα απάλειφε οποιαδήποτε επιπλέον συνδιάλεξη με την βάση! Το μοντέλο αυτό φαίνεται αρκετά όμορφο, αλλά θα πρέπει να σκεφτούμε ότι το πλήθος των αποθηκευμένων δεδομένων του Lucene αυξάνεται εκθετικά, άρα μεγαλώνει και ο χρόνος προσπέλασης και αναζήτησης της πληροφορίας. Μια καλή προσέγγιση θα ήταν ο διαμοιρασμός της πληροφορίας σε βάση και πλατφόρμα, ώστε να βρεθεί η χρυσή τομή της σωστής και γρήγορης λειτουργίας.

Τέλος, η απόδοση των δεδομένων συνδέεται άρρηκτα και με την παραγωγή τους. Βελτιωμένα συστήματα Stemming, όπου η εφαρμογή ήδη χρησιμοποιεί κάνοντας χρήση του Last Suffix Algorithm, θα μπορούσαν να επιφέρουν καλύτερη δόμηση της αρχικής πληροφορίας σε μια ακόμα πιο Generic μορφή. Αυτό θα εξυπηρετούσε τις ερωτήσεις των χρηστών, όπου θα έδιναν την δυνατότητα να έχουν και αυτά μια πιο γενική μορφή. Επιπλέον, θα μπορούσε να δημιουργηθεί σύστημα όπου από μια λέξη κλειδί να βρίσκει συνώνυμα τα οποία να βοηθούσαν στην αναζήτηση της πληροφορίας. Έτσι θα «σιγουρεύαμε» ότι ο χρήστης, ανεξαρτήτου διαλέκτου ή χρήση της γλώσσας, θα βρει την πληροφορία που επιζητεί.

Τα συμπεράσματα που προκύπτουν από τα παραπάνω, δεν ακυρώνουν τις διαδικασίες που υλοποιεί αυτή τη στιγμή η εφαρμογή αλλά σαφώς ανοίγουν τους ορίζοντες για αναδόμηση της, ώστε να επιτύχει ακόμα καλύτερα τους στόχους της.

4.2 Εισαγωγή νοημοσύνης στο Scraping

Όπως λέει και το ρητό, η αρχή είναι το ήμισυ του παντός. Δεν θα μπορούσε βέβαια να λείπει και από τον τομέα της πληροφορικής και της ανάπτυξης εφαρμογών. Για να μπορέσουμε λοιπόν να έχουμε καλύτερα αποτελέσματα στην εφαρμογή, θα πρέπει και τα δεδομένα που την τροφοδοτούν να είναι βέλτιστα. Σε εφαρμογές όμως, που η λήψη αποφάσεων για την καταγραφή των δεδομένων αποτελεί αποκλειστικό κομμάτι της μηχανής, θα πρέπει να υλοποιηθούν τέτοιες διαδικασίες ώστε η ενέργειες να είναι καθοριστικές και σωστές. Η ταχύτητα λήψης της απόφασης από τον άνθρωπο μπορεί να γίνει σε μερικά κλάσματα δευτερολέπτου, χωρίς βέβαια αυτό να σημαίνει ότι είναι και σωστή. Παρ' όλα αυτά, οι λειτουργίες όπου ο εγκέφαλος χρησιμοποιεί ώστε να καταλήξει σε κάποιο συμπέρασμα, είναι εκείνες που θα μπορούσαν να υλοποιηθούν ώστε τα δεδομένα να είναι βέλτιστα. Η δυσκολία λοιπόν, προκύπτει στο γεγονός της έκφρασης της πληροφορίας. Ο κάθε ένας μπορεί να την παρουσίαση με διαφορετικό τρόπο την έκφραση ενώ αυτή παραμένει να έχει το ίδιο νόημα.

Η λέξη κλειδί λοιπόν είναι το «νόημα». Για να καταφέρει λοιπόν η μηχανή να μην εξαρτάται μόνο από την εμφάνιση συγκεκριμένων λέξεων, την αλληλουχία τους ή την συχνότητά τους, θα μπορούσε στο μέλλον να υλοποιηθεί τεχνητό νευρωνικό σύστημα όπου με κατάλληλη εκμάθηση, θα μπορεί να επικεντρώνετε στο νόημα κάθε πρότασης και όχι στην λέξη. Αυτό θα απέδιδε μεγαλύτερη αξιοπιστία στην εξαγωγή των δεδομένων από τις σελίδες. Με τον τρόπο αυτό, διασφαλίζουμε επιπλέον, ότι τα αρχικά μας δεδομένα όπου τροφοδοτούν το σύστημα είναι πιο σωστά, άρα και ο τελικός χρήστης θα λάβει στοχευμένη πληροφορία.

Για την παραπάνω υλοποίηση μπορούν να χρησιμοποιηθούν τεχνικές NLP(Natural Language Processing). Οι τεχνικές αυτές προσομοιώνουν την ανθρώπινη αντίληψη στην επεξεργασία κειμένων χάρη στους Machine Learning αλγορίθμους, δηλαδή την επιμόρφωση της μηχανής για τον τρόπο λειτουργίας και διαχείρισης πόρων. Μπορούν δηλαδή να παραχθούν δεδομένα όπως περίληψη κειμένου, εξαγωγή ονομάτων, μετάφραση, αναγνώριση θεμάτων κλπ.

4.3 Ταχύτητα στην καταγραφή δεδομένων

Προηγουμένως έγινε αναφορά στην ταχύτητα της απόδοσης των δεδομένων από την εφαρμογή στους χρήστες. Από την άλλη, η ταχύτητα της εφαρμογής αναφέρεται και στην καταγραφή των δεδομένων από τις σελίδες. Αν σκεφτούμε ότι στο διαδίκτυο, υπάρχουν εκατομμύρια τέτοιες σελίδες, τότε θα χρειάζονταν πολλές περισσότερες από μια εφαρμογή για να καταγράψουν ένα μικρό κομμάτι του διαδικτύου. Θα πρέπει λοιπόν ο Web Crawler να αποφασίζει προσεκτικά πριν ζητήσει μια σελίδα καθώς ο χρόνος προσπέλασης της θα επιφέρει αύξηση του συνολικού χρόνου επεξεργασίας.

Πολλές από τις σελίδες είναι λογικό να μην περιέχουν την πληροφορία που επιζητούμε, πόσο μάλλον όταν αυτές είναι εξολοκλήρου άσχετες με το θέμα, όπως π.χ. μια ειδησεογραφική εφαρμογή. Εκεί ο Crawler θα επεξεργαστεί σελίδες, που όπως είναι λογικό δεν θα επιφέρουν δεδομένα. Θα μπορούσε λοιπόν να υλοποιηθεί αντίστοιχο σύστημα όπου θα εκτιμά σύμφωνα με το περιεχόμενο της σελίδας, το όνομα της ή και την διεύθυνση την σχετικότητα της σε σχέση με τα επιζητούμενα δεδομένα. Με τον τρόπο αυτό, θα αποφεύγεται η προσπέλαση σε σελίδες όπου δεν υπάρχει νόημα να επεξεργαστούν, καθώς θα μπορούσε να καταγραφεί ώστε να απορρίπτεται η προσπέλαση και σε μελλοντικές επισκέψεις.

Τέλος, η χρήση και αναγνώριση των Meta Tag σχετικά με την συχνότητα ενημέρωσης της κάθε σελίδας, θα βοηθούσε στην διατήρηση ων δεδομένων σε επίκαιρη μορφή και θα αποφεύγεται η παρουσίαση μη αξιόπιστων πληροφοριών.

Όλα τα παραπάνω, συνθέτουν κάποιες από τις διαδικασίες οι οποίες θα προσέφεραν αξιοπιστία στην λειτουργία της εφαρμογής. Η υλοποίηση του συστήματος βέβαια, πέτυχε τους βασικούς στόχους της, δηλαδή την ερευνητική υλοποίηση μιας μηχανής αναζήτησης και την αντιμετώπιση των δυσκολιών κατά την ανάπτυξή της. Η χρήση πολλών καινοτόμων εργαλείων βοήθησε αφ’ ενός στην ανάπτυξη της εφαρμογής σε μια σταθερή πλατφόρμα εξυπηρέτησης χρηστών και αφ’ ετέρου την αύξηση του επιπέδου γνώσεων του συγγραφέα και εμπειρίας του σε θέματα υλοποίησης ολοκληρωμένων Enterprise εφαρμογών.

Πηγές

[1] http://wikipedia.org/

[2] Spring in Action

[3] JBoss community

[4] Apache Foundation

[5] http://www.lucenetutorial.com/basic-concepts.html

[6] http://www.akamai.com/html/technology/how_search_works.html

[7] http://lucene.apache.org/core/2_9_4/queryparsersyntax.html

[8] http://spring.io/

[9] http://www.postgresql.org/

[10] http://lucene.apache.org/