ΤΕΙ ΘΕΣΣΑΛΟΝΙΚΗΣ - ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάπτυξη Διαδικτυακών Εφαρμογών με Πολλαπλές Στρατηγικές Αναζήτησης και χρήση του Lucece.NET

Πτυχιακή Εργασία
Τοπουζίδης Γεώργιος
Γκελενίντζε Ρομανός

Εισηγητής Καθηγητής
Μιχαήλ Σαλαμπάσης

Θεσσαλονίκη, Μάιος 2013

Περιεχόμενα

Μέρος 1^{ο} : Εισαγωγή και το αντικείμενο της πτυχιακής

Κεφάλαιο 1 : Εισαγωγή

Όπως είναι αντιληπτό πλέον στις μέρες μας η τεχνολογία της πληροφορίας παίζει έναν καθοριστικό ρόλο σε όλους τους τομείς της καθημερινής ζωής. Συνεχώς παρουσιάζονται καινούργιες τεχνολογίες και προϊόντα, τα οποία έχουν σκοπό να διευκολύνουν και να απλουστεύουν λεπτομέρειες που αφορούν τις προσωπικές, κοινωνικές και εργασιακές δραστηριότητές μας.

Το διαδίκτυο αποτελεί τον κύριο και αναπόσπαστο παράγοντα στις εξελίξεις αυτές, γιατί μέσω αυτού μεταδίδονται οι περισσότερες πληροφορίες μεταξύ δύο διαφορετικών απομακρυσμένων μονάδων που μπορεί να είναι είτε ηλεκτρονικοί υπολογιστές, είτε σύγχρονα κινητά τηλέφωνα είτε οποιεσδήποτε άλλες συσκευές μετάδοσης πληροφοριών.

Ένα αξιοσημείωτο βάρος στην μετάδοση πληροφορίας, εκτός από τις προαναφερθείς μονάδες που αποτελούν το τεχνικό μέρος , δίνεται και στο τρόπο με τον οποίο οι πληροφορίες ανταλλάσσονται μεταξύ τους. Ο τρόπος αυτός περιλαμβάνει τα διαδραστικά μέσα με τα οποία ο χρήστης άμεσα αλληλεπιδρά ώστε να μπορέσει να στείλει την κατάλληλη πληροφορία στον κατάλληλο προορισμό, ύστερα να λάβει απάντηση και ανάλογα να συνεχίσει την επικοινωνία είτε να την διακόψει εφόσον ολοκληρώθηκε ο σκοπός του.

Έτσι λοιπόν τα μέσα αυτά μπορούν να είναι είτε το λογισμικό (τα διάφορα προγράμματα εγκατεστημένα στις συσκευές) είτε ιστοσελίδες οι οποίες μπορούν να χρησιμοποιούνται ανεξαρτήτου τύπου συσκευής που έχει πρόσβαση σε αυτές. Υπάρχουν πολλές διαφορετικές ιστοσελίδες που είναι ανεπτυγμένες με πολλές διαφορετικές τεχνολογίες, και χαρακτηρίζονται επίσης από πολλά διαφορετικά χαρακτηριστικά όπως η σωστή δομή, ταχύτητα απόκρισης κ.α.

Ότι σκοπό και να εξυπηρετεί η κάθε ιστοσελίδα χρειάζεται να έχει ένα σωστό τρόπο λειτουργίας, να μην μπερδεύει τον χρήστη, να μπορεί να παρέχει όσο γίνεται δυνατόν γρηγορότερα και απλούστερα μενού πλοήγησης ώστε ο τελευταίος να βρίσκει την πληροφορία που χρειάζεται άμεσα.

1.1 Πολλαπλές μέθοδοι αναζήτησης

Συμπερασματικά αυτό που χρειάζεται ο χρήστης είναι να υπάρχει η σωστή αναζήτηση. Η αναζήτηση και αυτή από την πλευρά της όμως μπορεί να ποικίλει με πολλούς διάφορους τρόπους δηλ. η πιο δημοφιλής αναζήτηση είναι να υπάρχει μια μπάρα κειμένου συνήθως στην άνω πλευρά της ιστοσελίδας όπου μπορεί ο καθένας να γράψει αυτό που ζητά και να του εμφανιστούν ανάλογα αποτελέσματα. Κλασσικό παράδειγμα το Google.

Άλλος τρόπος είναι να υπάρχουν κάποια μενού τα οποία να περιέχουν υπομενού και τα οποία μπορεί να έχουν τα δικά τους υπομενού. Μια διαφορετική προσέγγιση είναι να υπάρχουν φωτογραφίες-μενού όταν οι επιλογές είναι λίγες ώστε να γίνεται πολύ εύκολη η πλοήγηση. Υπάρχουν πολλοί διαφορετικοί τρόποι αναζήτησης και μπορούμε να τους διακρίνουμε σε πολλές διαφορετικές ιστοσελίδες όπως στις εξής :

- www.plaisio.gr
- www.ebay.com
- www.amazon.com
- www.britannica.com
- www.jamieoliver.com

Αυτή τη στιγμή στον Ιστό απαριθμούνται αμέτρητες ιστοσελίδες με πάρα πολλούς διαφορετικούς τρόπους πλοήγησης και αναζήτησης που πλέον μερικές δεν μπορούν να χαρακτηριστούν με ακριβές τρόπο γιατί μπορούν να συνδυάζουν διαφορετικούς τρόπους στην λειτουργία τους. Το σημαντικό είναι ότι όλοι οι τρόποι αναζήτησης και πλοήγησης πρέπει να μην ξεφεύγουν από κάποια πρότυπα σχεδίασης και κανόνες λογικής γιατί οι χρήστες αν απογοητευτούν κατά την περιήγησή τους στην ιστοσελίδα υπάρχει περίπτωση να μην ενδιαφερθούν να την επισκεπτούν ξανά.

Εκτός από αυτά χρειάζεται να ειπωθεί ότι για να μπορούν οι αναζητήσεις να δουλεύουν δυναμικά, γρήγορα και αποτελεσματικά χρειάζονται μοντέρνες τεχνολογίες που παρουσιάζονται συνεχώς στην αγορά. Οι τεχνολογίες αυτές σε συνάρτηση με την τρόπο επικοινωνίας τους με μια βάση δεδομένων και τους τρόπους που διαθέτουν για την εξαγωγή και παρουσίαση της πληροφορίας, επιτρέπουν την ταχύτατη αναζήτηση και απόκριση στις αναζητήσεις των χρηστών. Μία τέτοια τεχνολογία είναι και το Lucene η οποία θα αναλυθεί λεπτομερώς στο επόμενο μέρος.

1.2 Αντικείμενο της πτυχιακής

Εισαγωγή

Στη σημερινή εποχή, στο έτος 2013 που βρισκόμαστε υπάρχουν αρκετές διαδικτυακές εφαρμογές και ιστοσελίδες που ασχολούνται με το αντικείμενο της αγοράς εργασίας. Όλες προσφέρουν χωρίς αμφιβολία περίπου τις ίδιες λειτουργίες : την αναζήτηση προσωπικού σύμφωνα με κάποια κριτήρια ή την αναζήτηση κάποιας δουλειάς. Αν και υπάρχει δυστυχώς μία ύφεση στον τομέα αυτό αυτή τη στιγμή στην Ελλάδα, η διαδικασία της εύρεσης δουλειάς και αναζήτησης προσωπικού πάντα θα είναι μια δημοφιλής διαδικασία ανάλογα βέβαια και την οικονομική κατάσταση που επικρατεί στην αγορά.

Σε γενικές γραμμές στην αγορά εργασίας έχουμε δύο βασικές περιπτώσεις όπως είπαμε. Απο τη μία πλευρά υπάρχουν πολλοί εργοδότες που ψάχνουν για

εργαζόμενους και τους αναζητούν συνήθως με κάποια συγκεκριμένα κριτήρια όπως π.χ με βάση την γεωγραφική περιοχή τους, με βάση το επίπεδο σπουδών τους κ.α. Απο την άλλη υπάρχουν και αυτοί που αναζητούν εργασία, και συχνά είτε συμπληρώνουν τα βιογραφικά τους σε κάποια διαδικτυακή εφαρμογή είτε τα στέλνουν μέσω ηλ. ταχυδρομείου ώστε να επιτευχθεί όσον το δυνατόν καλύτερη προβολή των ίδιων και να προκαλέσουν το ενδιαφέρον των εργοδοτών ώστε να αυξηθούν οι πιθανότητες τους να βρούν μία εργασία.

Δυστυχώς, πολλές φορές δημιουργούνται σε αυτήν τη κατάσταση κάποια κενά επικοινωνίας, δηλ. οι εργοδότες αναγκάζονται να ξοδέψουν πολύ χρόνο για να βρούν ακριβώς τον υπάλληλο στα μέτρα που ψάχνουν ή αυτοί που ψάχνουν για εργασία να μην μπορούν να βρούν κάποια αξιόλογη δουλειά που να ανταποκρίνεται πάνω στην προηγούμενη εμπειρία τους και να δυσκολεύονται οι εργοδότες έτσι να τους βρούνε.

1.2.1 Σκοπός

Το αντικείμενο της πτυχιακής – η εφαρμογή JobFinder είναι μια προσπάθεια που έχει στόχο να δημιουργήσει κατάλληλα φίλτρα αναζήτησης και για τις δύο παραπάνω περιπτώσεις δηλ. να κάνει πιο συγκεκριμένη και πιο εύκολη στην αναζήτηση την αγγελία που θα βάλει ο εργοδότης αλλά να κάνει και πιο έυκολα αναζητήσιμο το βιογραφικό του καθενός που αναζητεί εργασία. Το χαρακτηριστικό της εφαρμογής του JobFinder είναι ότι ο χρήστης μπορεί να συμπληρώσει το βιογραφικό του, και κάθε κομμάτι να μπορεί να αναζητηθεί ξεχωριστά ώστε να μπορέσει να βρεθεί ο ίδιος ακόμη και με την παραμικρή πληροφορία που μπορεί να σχετίζεται μ’αυτόν.

Μέρος 2ο : *LUCENE*

Κεφάλαιο 2 : Εισαγωγή

Το Lucene είναι μια δυναμική βιβλιοθήκη αναζήτησης της Java η οποία κάνει εύκολη την προσθήκη αναζήτησης σε οποιαδήποτε εφαρμογή. Τα τελευταία χρόνια το Lucene έχει γίνει εξαιρετικά δημοφιλής και αυτή τη στιγμή είναι η πιο διαδεδομένη βιβλιοθήκη ανάκτηση πληροφοριών γιατί δίνει τη δύναμη στα χαρακτηριστικά της αναζήτησης σε πολλούς ιστοχώρους και εφαρμογές. Παρόλο που είναι γραμμένη στη Java, χάρη στη δημοτικότητά της και την αποφασιστικότητα των προγραμματιστών της, αυτή τη στιγμή υπάρχουν εκδόσεις για ενσωμάτωση με άλλες γλώσσες προγραμματισμού (C/C++, C#, Ruby, Perl, Python και PHP).

Αυτό που κρύβεται πίσω από την επιτυχία του Lucene είναι η απλότητα αν και το γεγονός αυτό δεν πρέπει να ξεγελάει κανέναν γιατί στο εσωτερικό κρύβονται πολύπλοκες τεχνικές τελευταίας τεχνολογίας που δουλεύουν στο παρασκήνιο. Σημαντικό είναι ότι δεν χρειάζεται να έχει κανείς βαθιές γνώσεις όσον αφορά το τρόπο που το Lucene μπορεί να δημιουργήσει ένα ευρετήριο. Η διεπαφή προγραμματισμού εφαρμογής του (API) χρειάζεται μόνο λίγες κλάσεις για ξεκίνημα και το σημαντικότερο είναι ότι ο πυρήνας του (Jar μορφής) έχει μέγεθος μόνο ενός MB – χωρίς να έχει εξαρτήσεις.

2.1 Τι ακριβώς είναι το Lucene ;

Το Lucene είναι μια βιβλιοθήκη υψηλών επιδόσεων που μπορεί να ανακτά επεκτάσιμες πληροφορίες (IR-Information Retrieval). Το IR αναφέρεται στην διαδικασία αναζήτησης μετά-δεδομένων και γενικά της πληροφορίας μέσα στα έγγραφα. Είναι ώριμο , ανοιχτού κώδικα και δωρεάν έργο ανεπτυγμένο σε Java και βρίσκεται καταχωρημένο από το Ίδρυμα λογισμικού Apache. Επειδή είναι βιβλιοθήκη της Java δεν υποθέτει τι ακριβώς καταχωρεί στο ευρετήριό της και τι αναζητεί, πράγμα το οποίο της δίνει το πλεονέκτημα ανάμεσα σε άλλες εφαρμογές αναζήτησης. Η σχεδίαση του Lucene είναι συμπαγής και απλοϊκή κάνοντάς το εύκολο στην ενσωμάτωση των επιτραπέζιων εφαρμογών. Εκτός από τον κεντρικό του πυρήνα υπάρχουν πολλές λειτουργικές μονάδες που μπορούν να προσφέρουν και να επεκτείνουν την λειτουργικότητά του. Έχει χρησιμοποιηθεί σε πολλούς δημοφιλείς ιστότοπους και εφαρμογές όπως MySpace, LinkedIn, Apple, εγκυκλοπαίδεια Britannica ,FedEx κ.α.

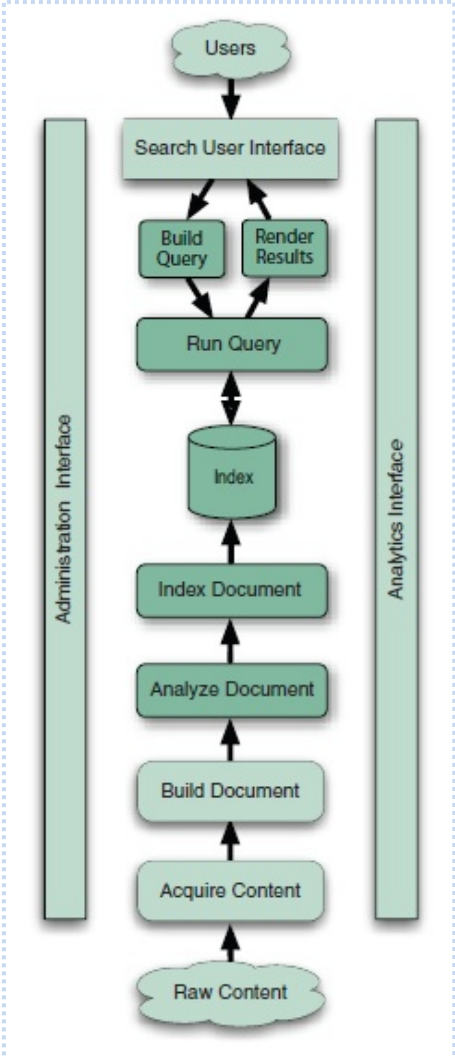
2.2 Τι μπορεί να κάνει

Πολλές φορές υπάρχει η λάθος αντίληψη ότι το Lucene είναι μία έτοιμη εφαρμογή όπως ένα πρόγραμμα αναζήτησης, πρόγραμμα ανίχνευσης του διαδικτύου ή μια μηχανή αναζήτησης ιστοχώρου. Δεν είναι κάτι από αυτά το Lucene αλλά απλά μια βιβλιοθήκη λογισμικού, μία εργαλειοθήκη και όχι μια

εφαρμογή αναζήτησης με πλήρεις χαρακτηριστικά. Αυτό που κάνει είναι να συνδέεται με την διαδικασία δημιουργίας ευρετηρίου και αναζήτησης και τα καταφέρνει πάρα πολύ καλά. Αφήνει την εφαρμογή του χρήστη να χειρίζεται συγκεκριμένους κανόνες στο πρόβλημα του τομέα της και παράλληλα κρύβει την πολυπλοκότητα της διαδικασίας δημιουργίας ευρετηρίου και αναζήτησης πίσω από μια απλή διεπαφή εφαρμογής προγραμματισμού ως προς το χρήστη. Με τη βοήθεια του Lucene οι χρήστες μπορούν να χρησιμοποιούν τις υπηρεσίες ευρετηρίου πλήρους κειμένου (full-text-search) που προσφέρουν πολλές βάσεις δεδομένων σε περιορισμένη βάση όπως επίσης να αναζητούν με σύνθετα κριτήρια όπως π.χ. (“John + Rice –eat –pudding, Apple-pie +tiger “).

Παραθέτουμε ένα σχήμα σχετικά με το ρόλο του Lucene σε μια εφαρμογή αναζήτησης:

Σύμφωνα με το σχήμα αυτό ξεκινώντας από κάτω, το πρώτο που αξίζει να ειπωθεί είναι ότι η βάση για όλες τις μηχανές αναζήτησης είναι μια διαδικασία που



Σχήμα 1: Τυπικά συστατικά μιας εφαρμογής αναζήτησης : τα επισκιασμένα μέρη δείχνουν ποια μέρη διαχερίζεται το Lucene.

ονομάζεται ***Δημιουργία Ευρετηρίου*** δηλ. η προώθηση των αρχικών δεδομένων σε ένα ευρετήριο παραπομπών υψηλών επιδόσεων ώστε να είναι εφικτή η γρήγορη αναζήτηση τους.

Για να υπάρχει μια σωστή δομή του τρόπου ανάλυσης και επεξήγησης του Lucene το κεφάλαιο θα χωριστεί σε 3 μέρη:

- Συστατικά για τη δημιουργία ευρετηρίου (Κεφ. 10)
- Συστατικά αναζήτησης (Κεφ. 11)
- Τα υπόλοιπα συστατικά μιας εφαρμογής αναζήτησης (Κεφ. 12)

Το κάθε μέρος θα περιλαμβάνει τα στοιχεία και τις διαδικασίες που χρησιμοποιούνται σε κάθε βήμα.

Κεφάλαιο 3 : Συστατικά για τη δημιουργία ευρετηρίου

Ας υποθέσουμε ότι έχουμε ένα μεγάλο αριθμό αρχείων και θέλουμε να βρούμε τα αρχεία που περιέχουν μια συγκεκριμένη λέξη η φράση. Μια προσέγγιση θα ήταν να σκανάρουμε σειριακά κάθε αρχείο για αυτή τη λέξη η φράση. Αν και αυτή η περίπτωση θα μπορούσε να δουλεύει άριστα, έχει κάποια μειονεκτήματα, το πιο προφανές από τα οποία είναι η κλιμάκωση μεγάλων συνόλων αρχείων ή μεγάλων αρχείων. Στην δική μας περίπτωση τη λύση δίνει η δημιουργία ενός ευρετηρίου με απώτερο σκοπό να κάνει αναζητήσεις ανάμεσα σε πολύ μεγάλα κείμενα πάρα πολύ γρήγορα: έτσι το ευρετήριο μετατρέπει το κείμενο σε κατάλληλη μορφή ώστε να βοηθήσει στην γρήγορη αναζήτηση, εξαλείφοντας μ'αυτό το τρόπο την διαδικασία σειριακού σκαναρίσματος. Η μετατροπή αυτή ονομάζεται ***δημιουργία ευρετηρίου*** και το προϊόν που δημιουργείται ονομάζεται ***ευρετήριο.***

Μπορούμε να φανταστούμε το ευρετήριο σαν μια δομή δεδομένων που επιτρέπει την τάχιστη τυχαία πρόσβαση σε λέξεις που βρίσκονται αποθηκευμένες σ'αυτήν. Θα μπορούσαμε να πούμε κιόλας ότι αυτή η διαδικασία μοιάζει με το ευρετήριο που υπάρχει στο τέλος ενός βιβλίου και επιτρέπει τον γρήγορο εντοπισμό σελίδων που περιγράφουν κάποιο συγκεκριμένο θέμα.

Η δημιουργία ευρετηρίου αποτελείται από μια ακολουθία μεμονωμένων διαδικασιών που περιγράφονται στη συνέχεια. Αρχικά αυτό που χρειάζεται είναι η πρόσβαση στο περιεχόμενο μέσα στο οποίο θα θέλουμε να γίνει η αναζήτηση.

3.1 Απόκτηση Περιεχομένου

Το πρώτο βήμα όπως απεικονίζεται στο Σχήμα 2.1 είναι η απόκτηση του περιεχομένου(Acquire Content). Η διαδικασία αυτή περιλαμβάνει την χρήση μιας "αράχνης" (spider, crawler) που συλλέγει το περιεχόμενο που πρέπει να εισαχθεί στο ευρετήριο. Η συλλογή αυτή μπορεί να αποτελέσει ασήμαντο γεγονός αν για παράδειγμα γίνει δημιουργία ευρετηρίου από ένα σύνολο αρχείων XML που βρίσκονται σε μια συγκεκριμένη τοποθεσία στο σύστημα αρχείων ή αν όλο το

περιεχόμενο βρίσκεται σε μια καλά-οργανωμένη βάση δεδομένων. Αντιθέτως μπορεί να αποδειχτεί και πολύ περίπλοκη και ακατάστατη διαδικασία αν το περιεχόμενο είναι σκορπισμένο σε πολλά διαφορετικά μέρη όπως διάφορες ιστοσελίδες, συστήματα αρχείων, βάσεις δεδομένων, τοπικά αρχεία XML κ.α.

Επίσης ένα πρόβλημα μπορεί να είναι και η εξουσιοδότηση στα περιεχόμενα αυτά αφού μπορεί να χρειάζονται δικαιώματα διαχειριστή ώστε να αποκτηθεί το περιεχόμενο.

Για μεγάλα σύνολα δεδομένων είναι σημαντικό να υπάρχει η λειτουργία να μπορούν να προστίθενται μόνο δεδομένα που έχουν επεξεργαστεί η προστεθεί τελευταία. Το Lucene σαν βιβλιοθήκη αναζήτησης πυρήνα δεν παρέχει καμία λειτουργία να υποστηρίζει την απόκτηση περιεχομένου. Αυτό εξαρτάται αποκλειστικά από την εφαρμογή, ή από ένα απομονωμένο μέρος κώδικα. Υπάρχουν βέβαια και "αράχνες" ανοιχτού κώδικα όπως οι παρακάτω που μπορούν να βοηθήσουν στην διαδικασία αυτή :

- Solr (http://lucene.apache.org/solr/). Προϊόν του Apache Lucene, που μπορεί να υποστηρίζει σχετικές βάσεις δεδομένων και XML τροφοδότες και την διαχείριση πλούσιων εγγράφων μέσω ενσωμάτωσης Tika.
- Nutch (http://lucene.apache.org/nutch). Και αυτό προϊόν του Apache Lucene, το οποίο έχει μία "αράχνη" υψηλής σκάλας που ανακαλύπτει το περιεχόμενο από διάφορους ιστότοπους
- Τα Grub,Heritrix, Droids και Aperture αποτελούν άλλα γνωστά εργαλεία που κινούνται σε ανάλογες γραμμές.

Αν η εφαρμογή μας έχει το περιεχόμενο διασκορπισμένο σε διάφορα σημεία, θα ήταν καλό να χρησιμοποιηθεί κάποιο από τα παραπάνω εργαλεία για πιο εύκολη εντόπιση και συλλογή περιεχομένου.

3.1.1 Απόκτηση περιεχομένου στο JobFinder

Στη δική μας εφαρμογή η διαδικασία αυτή είναι σχετικά εύκολη αφού παίρνουμε το περιεχόμενό μας από τη βάση δεδομένων :

```
public ActionResult CreateIndex()
{
    Models.Search.AddToLuceneIndex(
        _accountService.GetAllAds(),
        _accountService.GetAllCvs());
    TempData["Result"] = "Search index was created successfully!";
    return RedirectToAction("Index");
}
```

Σχήμα 2: Η μέθοδος CreateIndex που αναλαμβάνει τη διαδικασία συλλογής περιεχομένου

Όπως βλέπουμε στο παραπάνω σχήμα η μέθοδος CreateIndex() καλεί τη μέθοδο AddToLuceneIndex() προωθώντας στις παραμέτρους της δύο λίστες αντικειμένων (GetAllAds() και GetAllCvs()) οι οποίες έχουν επιστρέψει όλες τις

αγγελίες και όλα τα βιογραφικά από τη βάση δεδομένων με τη βοήθεια του στιγμιότυπου της κλάσης IGeneralService, _accountService.

Το επόμενο βήμα είναι η δημιουργία «εγγράφων» από το περιεχόμενο που έχουμε συλλέξει.

3.2 Χτίσιμο Εγγράφου

Πριν αναλυθεί η διαδικασία του χτισίματος εγγράφου είναι σημαντικό να δοθεί κάποια εξήγηση και ερμηνεία για κάποιες έννοιες που είναι βασικές για την κατανόηση της διαδικασίας αυτής. Οι πιο θεμελιώδεις μονάδες για την δημιουργία ευρετηρίου και αναζήτησης είναι οι κλάσεις ***Έγγραφο*** και ***Πεδίο*****.**

3.2.1 Έγγραφο (Document)

Η κλάση Έγγραφο αναπαριστά μια συλλογή από πεδία. Μπορούμε να το παρομοιάσουμε με ένα εικονικό έγγραφο – ένα κομμάτι δεδομένων, όπως μια ιστοσελίδα, ένα email ή ένα αρχείο κειμένου – που θα θελήσουμε να το ανακτήσουμε μελλοντικά. Τα πεδία του εγγράφου αναπαριστούν τα μετά-δεδομένα που σχετίζονται με το έγγραφο. Η αληθινή πηγή προέλευσης των δεδομένων του εγγράφου (όπως αρχεία Word, ένα έγγραφο βάσης δεδομένων ή και ένα κεφάλαιο βιβλίου) δεν έχουν καμία διαφορά στο Lucene.Αυτό που έχει σημασία είναι το κείμενο που εξαγάγουμε από τέτοια δυαδικά έγγραφα, και τα προσθέτουμε σαν τιμές πεδίου τις οποίες επεξεργάζεται το Lucene. Τα μετά-δεδομένα (όπως ο συγγραφέας, τίτλος, αντικείμενο και η ημερομηνία) προστίθενται στο ευρετήριο και αποθηκεύονται ξεχωριστά ως Πεδία του Εγγράφου.

Το Lucene δουλεύει μόνο με συμβολοσειρές ή με νούμερα, έτσι όταν έχουμε διαφορετικά αρχεία κειμένου, δημιουργούμε για το καθένα από αυτά ένα στιγμιότυπο της κλάσης Εγγράφου, δημιουργούμε τα πεδία που χρειαζόμαστε, δίνουμε όνομα σ’αυτά, συμπληρώνουμε τις τιμές τους και προσθέτουμε το έγγραφο αυτό στο ευρετήριο δυναμικά. Συμπερασματικά, σύμφωνα με τις ανάγκες του καθενός πάντα, στην εφαρμογή πρέπει να γίνει προσεκτικός σχεδιασμός για το πώς θα χτιστεί ένα έγγραφο Lucene και τα πεδία του.

3.2.2 Πεδίο(Field)

Όπως είπαμε κάθε έγγραφο περιέχει ένα η περισσότερα πεδία τα οποία έχουν ένα όνομα, μία τιμή αλλά όμως και μια πληθώρα επιλογών για τον ακριβή τρόπο στο πώς να προστεθεί η τιμή τους στο ευρετήριο του Lucene.

Ένα έγγραφο μπορεί να περιέχει πάνω από δύο πεδία με το ίδιο όνομα. Σ’αυτή τη περίπτωση οι τιμές των πεδίων επισυνάπτονται κατά τη δημιουργία του ευρετηρίου με τη σειρά που προστέθηκαν στο έγγραφο. Κατά την αναζήτηση, συμβαίνει το ίδιο αν το κείμενο από όλα τα πεδία ήταν ενωμένο και έμοιαζε με ένα μόνο πεδίο κειμένου.

3.2.3 Οι υπόλοιπες κλάσεις που καθορίζουν το χτίσιμο ευρετηρίου

3.2.3.1 IndexWriter

Η κλάση IndexWriter είναι το κεντρικό συστατικό της διαδικασίας δημιουργίας ευρετηρίου. Αυτή η κλάση δημιουργεί ένα ευρετήριο ή ανοίγει ένα υπάρχον, και προσθέτει, αφαιρεί ή ενημερώνει τα έγγραφα στο ευρετήριο. Μπορούμε να σκεφτούμε το IndexWriter σαν ένα αντικείμενο που δίνει τη δυνατότητα να γράψουμε κάτι στο ευρετήριο αλλά όχι να διαβάσουμε ή να αναζητήσουμε. Το IndexWriter χρειάζεται κάπου να αποθηκεύει το ευρετήριο και γι'αυτό το σκοπό υπάρχει η κλάση Directory.

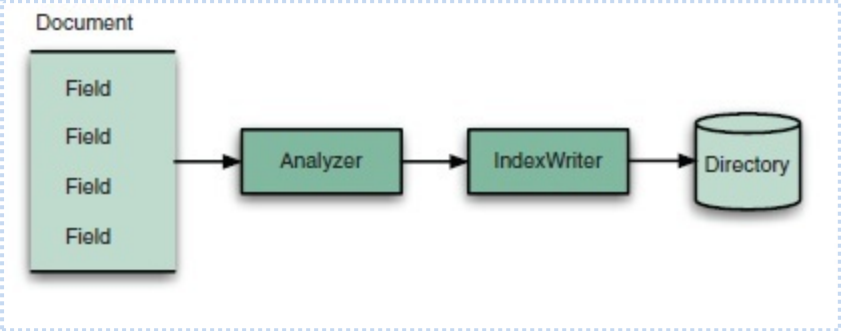
3.2.3.2 Directory

Η κλάση Directory απεικονίζει την θέση ενός ευρετηρίου του Lucene.Είναι μια αφηρημένη κλάση που επιτρέπει τις υποκλάσεις της να αποθηκεύουν το ευρετήριο όπως κρίνουν οι ίδιες. Το Lucene περιλαμβάνει διάφορες ενδιαφέρον υλοποιήσεις του Directory όπως π.χ. RamDirectory όπου τα αρχεία αποθηκεύονται στη RAM κ.α. όπως SimpleFSDirectory, NIOFSDirectory, MmapDirectory με ξεχωριστές δυνατότητες.

3.2.3.3 Analyzer

Πριν να προστεθεί το κείμενο στο ευρετήριο πρέπει να περάσει μέσα απ'τον Analyzer.Ο τελευταίος που καθορίζεται στον δομητή του IndexWriter, είναι υπεύθυνος για την εξαγωγή των στιγμιότυπων (tokens) από το κείμενο που πρέπει να προστεθούν στο ευρετήριο αγνοώντας το υπόλοιπο κείμενο. Αν το περιεχόμενο που πρέπει να προστεθεί στο ευρετήριο δεν είναι τύπου κειμένου πρέπει να μετατραπεί σ'αυτό για να μπορέσει να προστεθεί. Το Tika που είναι ένα προϊόν του Apache, μπορεί να εξαγάγει κείμενο από τους πιο συνηθισμένους τύπους εγγράφων πλούσιας-μορφής (PDF, Ms Word κ.α.). Το Analyzer είναι μια αφηρημένη κλάση, αλλά το Lucene περιλαμβάνει κάποιες υλοποιήσεις του. Κάποιες από αυτές έχουν τη δυνατότητα να αγνοούν κάποια *stop words* (συχνά χρησιμοποιούμενες λέξεις που δεν προσφέρουν τίποτα ξεχωριστό σε κάποιο κείμενο όπως , a, an, the , in και on), κάποιες να μετατρέπουν τα στιγμιότυπα σε πεζά γράμματα ώστε οι αναζητήσεις να μην έχουν διάκριση πεζών/κεφαλαίων κ.α. Οι Αναλυτές είναι ένα σημαντικό μέρος του Lucene και μπορούν να χρησιμοποιηθούν και για άλλες χρήσιμες λειτουργίες εκτός από το φιλτράρισμα εισόδου. Για έναν προγραμματιστή που ενσωματώνει το Lucene σε μια εφαρμογή, η επιλογή ενός αναλυτή είναι ένα κρίσιμο στοιχείο στην σχεδίαση της εφαρμογής. Η διαδικασία της ανάλυσης απαιτεί την ύπαρξη ενός εγγράφου που περιέχει ξεχωριστά πεδία για να μπορέσει να προστεθεί στο ευρετήριο.

Το ακόλουθο σχήμα δείχνει πώς αυτές οι κλάσεις συμμετέχουν στην διαδικασία δημιουργίας ευρετηρίου:



Σχήμα 3: Οι κλάσεις που χρησιμοποιούνται στην διαδικασία δημιουργίας ευρετηρίου εγγράφων με το Lucene

3.2.4 Η διαδικασία χτισίματος συνοπτικά

Αφού έχουμε ήδη επιλέξει το «ωμό» περιεχόμενο που πρέπει να προστεθεί στο ευρετήριο, χρειάζεται να γίνει η μετάφραση του σε έγγραφα που θα χρησιμοποιηθούν ύστερα από τις μηχανές αναζήτησης. Τυπικά το έγγραφο αποτελείται από μερικά διαχωρισμένα πεδία και τιμές, όπως ο τίτλος, σώμα, περίληψη, συγγραφέας και υπερσύνδεσμος σελίδας. Χρειάζεται προσοχή στην σχεδίαση και τον διαχωρισμό του περιεχομένου σε έγγραφα και πεδία όπως και στον υπολογισμό της τιμής για κάθε πεδίο από αυτά. Μερικές φορές αυτό είναι εύκολη υπόθεση όπως π.χ. ένα email μπορεί να αποτελεί ένα έγγραφο, αλλά άλλες φορές υπάρχουν δυσκολίες όπως π.χ. πώς να διαχειριστούμε τα συνημμένα ενός email. Δηλαδή να προσκολλήσουμε όλο το κείμενο από τα συνημμένα σε ένα και μοναδικό αρχείο, ή να κάνουμε ξεχωριστά έγγραφα για κάθε συνημμένο έτσι ώστε να συνδέονται με το αρχικό μήνυμα email.

Ύστερα από την απόφασή για τον κατάλληλο σχεδιασμό, πρέπει να ακολουθήσει η εξαγωγή κειμένου από το αρχικό περιεχόμενο για κάθε έγγραφο. Αν το περιεχόμενο είναι ήδη σε μορφή κειμένου από τη φύση του, με μια κανονική κωδικοποίηση γίνεται πολύ απλή η διαδικασία αυτή. Στη σημερινή εποχή, πολύ συχνά τα περιεχόμενα είναι σε δυαδική μορφή (PDF, Ms Office, Open Office, Adobe Flash) ή περιέχουν κάποιες μορφοποιήσεις που πρέπει να αφαιρεθούν (RDF, XML, HTML). Χρειάζεται ένα σωστό φιλτράρισμα για την εξαγωγή κειμένου από τέτοιο περιεχόμενο πριν τη δημιουργία ενός εγγράφου. Σε αυτό συνεισφέρουν και οι «αναλυτές» που μπορούν να «βγάλουν» ονόματα, μέρη, ημερομηνίες σε ξεχωριστά πεδία στο έγγραφο.

Ένα άλλο σημαντικό χαρακτηριστικό είναι ο «εμβολιασμός» με «τόνωση» κάποιων εγγράφων που φαίνονται περισσότερο η λιγότερο χρήσιμα. Για παράδειγμα θα μπορούσαμε να θέλουμε ώστε τα έγγραφα που επεξεργάστηκαν πρόσφατα να είναι πιο σημαντικά από τα παλιά έγγραφα. Μ'αυτό το τρόπο

«τονώνοντας» ένα έγγραφο στα αποτελέσματά μας μπορούμε να δώσουμε μεγαλύτερη σημασία σε αυτό απ'ότι στα άλλα έγγραφα της κατηγορίας του. Η «τόνωση» μπορεί να γίνει στατικά (ανά έγγραφο ή πεδίο) στην δημιουργία ευρετηρίου ή δυναμικά την ώρα της αναζήτησης. Αν θέλουμε να τονώσουμε ένα έγγραφο που ήδη υπάρχει χρειάζεται να το δημιουργήσουμε από την αρχή. Σχεδόν όλες οι μηχανές αναζήτησης περιλαμβάνοντας το Lucene, αυτόματα «τονώνουν» με στατικό τρόπο τα πεδία που είναι πιο σύντομα από εκείνα που είναι πιο μεγάλα. Αυτό είναι λογικό γιατί αν θελήσει κανείς να αναζητήσει μια η δύο λέξεις σε ένα έγγραφο που αποτελείται από 3-4 λέξεις θα είναι πιο πολύ σχετικό απ'ότι να αναζητήσει τις δύο λέξεις αυτές σε ένα μακρύ έγγραφο πολλών λέξεων. Γενικά ή τόνωση πρέπει να γίνεται με προσοχή γιατί υπάρχει η δυνατότητα από την κατάχρησή της να διαβρωθεί γρήγορα και καταστροφικά η εμπιστοσύνη του χρήστη.

Το Lucene προσφέρει μια διεπαφή προγραμματισμού χρήστη για την δημιουργία πεδίων και εγγράφων, αλλά δεν προσφέρει καμία λογική στο χτίσιμο ενός εγγράφου επειδή αυτό εξαρτάται αποκλειστικά από τις ανάγκες της εφαρμογής. Επίσης δεν προσφέρει φιλτράρισμα εγγράφων αν και μπορεί να γίνει αυτό με το Tika πολύ αποτελεσματικά. Στην περίπτωση που το περιεχόμενο βρίσκεται σε βάση δεδομένων, προγράμματα όπως DBSight, Hibernate Search, LuSQLm Compass κ.α. μπορούν να κάνουν εύκολη υπόθεση την δημιουργία ευρετηρίου και την αναζήτηση των πινάκων , χειρίζοντας απρόσκοπτα την Απόκτηση Περιεχομένου και το Χτίσιμο Εγγράφου.

Τα πεδία κειμένου ενός εγγράφου δεν έχουν τη δυνατότητα να εισέλθουν στο ευρετήριο αν δεν αναλυθούν πρώτα.

3.3 Ανάλυση Εγγράφου

Η *Ανάλυση* στο Lucene, είναι μια διαδικασία μετατροπής κειμένου στην πιο θεμελιώδη αναπαράσταση ευρετηρίου που είναι οι ***Όροι***. Οι όροι χρησιμοποιούνται για να καθορίσουν ποια έγγραφα αντιστοιχούν στην ερώτηση-αίτηση κατά την αναζήτηση. «Για παράδειγμα», αν προσθέταμε την συγκεκριμένη φράση στο ευρετήριο οι όροι θα έμοιαζαν με «Για» και «παράδειγμα», δηλ. σαν ξεχωριστοί όροι σε μια ακολουθία. Ο *Αναλυτής* αποτελεί μια ενθυλάκωση της διαδικασίας ανάλυσης. Ένας αναλυτής «κόβει» το κείμενο πραγματοποιώντας όποιες διαδικασίες χρειαστούν, όπως την εξαγωγή λέξεων, την αφαίρεση σημείων στίξης και τόνων, μετατροπή των γραμμάτων σε πεζά , την αφαίρεση επαναλαμβανόμενων λέξεων όπως και την μείωση των παραγώγων μιας λέξης σε μια αρχική μορφή. Η διαδικασία αυτή ονομάζεται «κοπή»(tokenization) και τα κομμάτια του κειμένου που διαιρέθηκαν από το ολόκληρο αρχικό κείμενο ονομάζονται στιγμιότυπα (tokens). Τα στιγμιότυπα συνδυασμένα με το αντίστοιχο όνομα πεδίου ονομάζονται όροι.

Ο πρωτεύον στόχος του Lucene είναι να διευκολύνει την ανάκτηση της πληροφορίας. Η έμφαση στην ανάκτηση πληροφορίας είναι σημαντική. Μπορούμε να δώσουμε πληθώρα λέξεων και το Lucene να τις εντοπίσει δυναμικά μέσα στο κείμενο. Για να καταλάβει το Lucene αυτές τις λέξεις, αναλύει το κείμενο την ώρα της αναζήτησης διαιρώντας το σε όρους. Οι όροι αυτοί αποτελούν τα βασικά στοιχεία για την αναζήτηση.

Η επιλογή ενός κατάλληλου αναλυτή αποτελεί κρίσιμη απόφαση στην ανάπτυξη της εφαρμογής, και μια μόνο επιλογή δεν μπορεί να είναι συνήθως αρκετή ή απόλυτη. Η γλώσσα είναι ένας παράγοντας, επειδή η κάθε μια έχει τα δικά της μοναδικά χαρακτηριστικά. Άλλος παράγοντας είναι «το είδος» του κειμένου που θέλουμε να αναλυθεί : υπάρχουν πολλές διαφορετικές βιομηχανίες με διαφορετική ορολογία, ακρωνύμια και συντομογραφίες που χρειάζονται προσοχή. Έτσι, μπορεί να χρειαστεί και η δημιουργία ενός τροποποιημένου αναλυτή ή και περισσότερο του ενός επειδή ένας και μόνο, όπως είπαμε και πριν συνήθως δεν φτάνει.

Στη συνέχεια θα εξερευνήσουμε πώς το Lucene χρησιμοποιεί ένα αναλυτή. Η ανάλυση πραγματοποιείται κάθε φορά που το κείμενο χρειάζεται να μετατραπεί σε όρους, πράγμα το οποίο γίνεται σε δύο πυρήνες του Lucene: κατά τη διάρκεια δημιουργίας ευρετηρίου και κατά την χρησιμοποίηση του QueryParser για την αναζήτηση. Αν δώσουμε μία επισήμανση στις λέξεις που αναζητάμε στα αποτελέσματα αναζήτησης μπορεί να χρειαστεί η ανάλυση κειμένου και σε αυτό το σημείο ακόμη. Ας δούμε τις πιο δημοφιλείς κατηγορίες αναλυτών του Lucene.

3.3.1 Αναλυτές του Lucene

Αρχικά, ας αναλύσουμε την φράση “The quick brown fox jumped over the lazy dog” χρησιμοποιώντας και τους 4 ενσωματωμένους αναλυτές του Lucene:

WhitespaceAnalyzer:

[The] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dog]

SimpleAnalyzer:

[the] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dog]

StopAnalyzer:

[quick] [brown] [fox] [jumped] [over] [lazy] [dog]

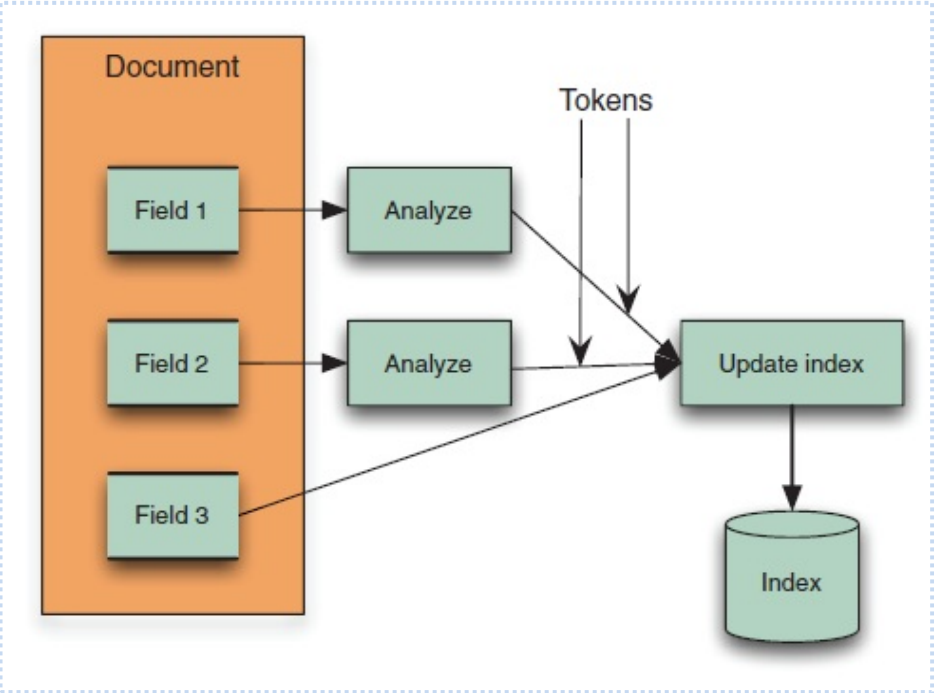
StandardAnalyzer:

[quick] [brown] [fox] [jumped] [over] [lazy] [dog]

Κάθε στιγμιότυπο φαίνεται μέσα σε αγκύλες ώστε να κάνει εμφανείς τις διαφορές. Όταν δημιουργείται το ευρετήριο, τα στιγμιότυπα που εξαγάγονται την

ώρα της ανάλυσης είναι και οι όροι που προστίθενται στο ευρετήριο. Το σημαντικότερο είναι ότι μόνο οι όροι που προστίθενται στο ευρετήριο είναι αναζητήσιμοι εκτός και αν έχουμε τις επιλογές πεδίου Field.Index.NOT_ANALYZED ή Field.Index.NOT_ANALYZED_NO_NORMS περιπτώσεις στις οποίες όλη η τιμή του πεδίου θα αποτελεί ένα και μοναδικό στιγμιότυπο που θα μπορεί να αναζητηθεί εξ ολοκλήρου.

Στο παρακάτω σχήμα φαίνεται η διαδικασία Ανάλυσης κατά την δημιουργία ευρετηρίου:



Σχήμα 4: Τα πεδία 1 και 2 αναλύονται δημιουργώντας μια ακουλουθία στιγμιοτύπων, ενώ το πεδίο 3 δεν αναλύεται έτσι ώστε η ολόκληρη τιμή του να προστεθεί στο περιεχόμενο σαν ένα και μοναδικό στιγμιότυπο

Ας δώσουμε και ένα άλλο παράδειγμα για καλύτερη κατανόηση υποθέτοντας ότι έχουμε αυτή τη φορά τη φράση “XY&Z Corporation - xyz@example.com” χρησιμοποιώντας τους ίδιους αναλυτές:

WhitespaceAnalyzer:

[XY&Z] [Corporation] [-] [xyz@example.com]

SimpleAnalyzer:

[xy] [z] [corporation] [xyz] [example] [com]

StopAnalyzer:

[xy] [z] [corporation] [xyz] [example] [com]

StandardAnalyzer:

[xy&z] [corporation] [xyz@example.com]

Σύμφωνα με τα ενδιαφέρον στοιχεία που υπάρχουν στα παραδείγματα αυτά φαίνεται πιο χαρακτηριστικά ότι τα στιγμιότυπα εξαρτώνται πολύ από τον αναλυτή. Μπορεί να προσέξει κανείς πώς αντιμετωπίζεται η λέξη “the” , το όνομα “XY&Z”, η διεύθυνση “xyz@example.com”, ο χαρακτήρας παύλα “-“ και η περίπτωση του κάθε στιγμιότυπου γενικά. Συμπερασματικά θα μπορούσαμε να πούμε τα εξής για τους αναλυτές:

- WhitespaceAnalyzer : όπως δηλώνει και το όνομά του, ξεχωρίζει το κείμενο σε στιγμιότυπα πάνω στους κενούς (whitespace) χαρακτήρες και δεν κάνει άλλη προσπάθεια για να κανονικοποιήσει τα στιγμιότυπα. Δεν κάνει τα γράμματα πεζά στα στιγμιότυπα.
- SimpleAnalyzer : πρώτα ξεχωρίζει τα στιγμιότυπα στους χαρακτήρες που δεν είναι γράμματα, ύστερα κάνει τα γράμματα στα στιγμιότυπα πεζά. Χρειάζεται προσοχή γιατί αυτός ο αναλυτής αφαιρεί τους αριθμητικούς χαρακτήρες.
- StopAnalyzer : είναι το ίδιο με τον SimpleAnalyzer απλά η διαφορά του είναι ότι αφαιρεί κοινές λέξεις. Απ’τη φύση του αφαιρεί λέξεις σύμφωνα με την Αγγλική Ορολογία όπως “the,a “ κτλ. αν και μπορούμε να τις προσθέσουμε στις εξαιρέσεις ένα θέλουμε.
- StandarAnalyzer : είναι ο πιο εξελιγμένος αναλυτής πυρήνα. Έχει κάποια λογική για να αναγνωρίζει κάποια συγκεκριμένα στιγμιότυπα, όπως ονόματα εταιριών, διευθύνσεις e-mail και ονόματα ιστότοπων. Επίσης κάνει τα γράμματα πεζά σε κάθε στιγμιότυπο και αφαιρεί λέξεις διακοπής και τα σημεία στίξης.
- Επίσης για λόγους αναφοράς υπάρχει και ο KeywordAnalyzer ο οποίος μετατρέπει όλο το κείμενο σε ένα μόνο στιγμιότυπο.

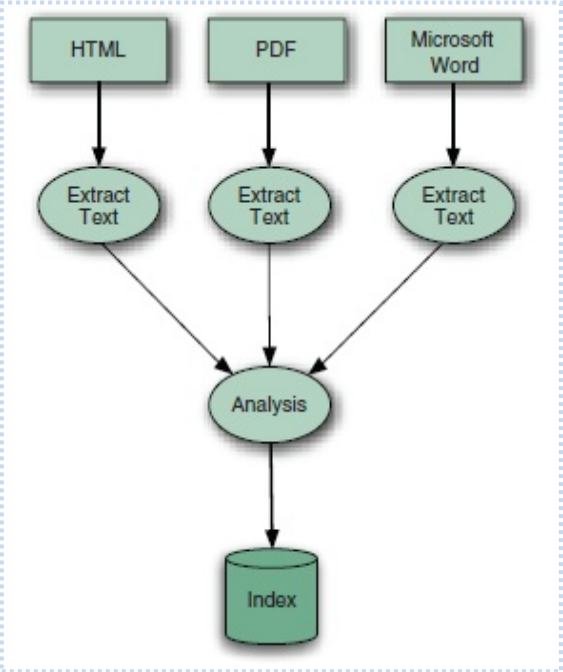
Το Lucene δεν κάνει τα αποτελέσματα της ανάλυσης ορατά στον τελικό χρήστη. Οι όροι προστίθενται στο ευρετήριο χωρίς να φαίνεται αυτό και είναι αυτοί οι οποίοι εντοπίζονται κατά την αναζήτηση. Επίσης, καθώς κάνοντας αναζητήσεις με το QueryParser, η διαδικασία ανάλυσης γίνεται στα μέρη του κειμένου της ερώτησης αίτησης, ώστε να υπάρχουν οι καλύτερες αντιστοιχίσεις.

Ανακεφαλαιώνοντας, θα μπορούσαμε να πούμε ότι το Lucene προσφέρει ένα πίνακα από ενσωματωμένους αναλυτές που δίνουν άριστο έλεγχο για την διαδικασία της ανάλυσης. Υπάρχει επίσης και η δυνατότητα να χτίσουμε τον δικό μας αναλυτή, ή να δημιουργήσουμε αυθόρμητα αλυσίδες αναλυτών ταιριάζοντας τους δημιουργούς στιγμιότυπων και τα φίλτρα στιγμιότυπων ώστε να προσαρμόσουμε στο βαθμό που θέλουμε την δημιουργία των στιγμιότυπων.

Το τελικό βήμα που ακολουθεί είναι η δημιουργία ευρετηρίου εγγράφων.

3.4 Δημιουργία Ευρετηρίου Εγγράφων

Στην διάρκεια αυτού του βήματος , το έγγραφο προστίθεται στο ευρετήριο. Το Lucene προσφέρει όλα τα απαραίτητα στοιχεία για αυτό το βήμα κάνοντας κατά κάποιον τρόπο κάτι σαν «μαγικά» κάτω από μια πολύ απλοποιημένη διεπαφή χρήστη.



Σχήμα 5: Η Δημιουργία Ευρετηρίου αποτελείται κυρίως απο 3 λειτουργίες : 1) Την εξαγωγή κειμένου από τα αρχεία πηγής 2) την ανάλυσή του 3) και την αποθήκευσή του στο ευρετήριο.

3.4.1 Η διαδικασία αυτή στην εφαρμογή του JobFinder

Στο επόμενο σχήμα φαίνεται η στατική μέθοδος Directory (_directory) όπου το _luceneDir καθορίζει το μονοπάτι για το ευρετήριο:

```
private static FSDirectory _directory//Lucene.Net.Store.FSDirectory
{
    get
    {
        if (_directoryTemp == null)
            _directoryTemp = FSDirectory.Open(new DirectoryInfo(_luceneDir));

        if (IndexWriter.IsLocked(_directoryTemp))
            IndexWriter.Unlock(_directoryTemp);

        var lockFilePath = Path.Combine(_luceneDir, "write.lock");
        if (File.Exists(lockFilePath))
            File.Delete(lockFilePath);
        return _directoryTemp;
    }
}
```

Σχήμα 5: Η στατική μέθοδος _directory καλείται έτσι όπως είναι κάνοντας κάποιους ελέγχους αρχικά

Στο παρακάτω σχήμα φαίνεται η μέθοδος AddToLuceneIndex η οποία παίρνει ως παραμέτρους λίστες αντικειμένων τύπου IEnumerable που προέρχονται από τη βάση δεδομένων και με τις εγγραφές και τις τιμές που έχουν δημιουργούμε αντίστοιχα έγγραφα και πεδία που προσθέτουμε ύστερα στο ευρετήριο.

```
public static void AddToLuceneIndex(
    IEnumerable<Ad> ads,IEnumerable<UserCv> usercvs)
{
    // init lucene
    var analyzer = new StandardAnalyzer(Version.LUCENE_29);
    using (var writer = new IndexWriter(
        _directory, analyzer, IndexWriter.MaxFieldLength.UNLIMITED))
    {
        // add data to lucene search index (replaces older entry if any)
        foreach (var userCv in usercvs)
            _addToLuceneIndexUserCvs(userCv, writer);
        foreach (var ad in ads)
            _addToLuceneIndexAd(ad, writer);

        //this is for analytic debugging information-
        //writer.SetInfoStream();

        // close handles
        analyzer.Close();
        writer.Close();
        writer.Dispose();
    }
}
```

Σχήμα 6: Η στατική μέθοδος AddUpdateLuceneIndex2 για κάθε εγγραφή που υπάρχει στη λίστα των "ads" (δηλ. Αγγελίες των εργοδοτών) καλεί την μέθοδο _addToLuceneIndexAd

Στην πρώτη γραμμή φαίνεται ότι δημιουργούμε έναν Analyzer τύπου StandardAnalyzer τον οποίο χρησιμοποιούμε με το _directory στις παραμέτρους του IndexWriter για την διαδικασία πρόσθεσης εγγραφών στο ευρετήριο. Αξίζει να σημειωθεί ότι στο τέλος κλείνουμε πρώτα τον Αναλυτή (για να αδειάσουν οι πηγές που χρησιμοποιεί ο Αναλυτής). Ύστερα κλείνουμε το IndexWriter διαπράττοντας όλες τις αλλαγές στο Ευρετήριο, κλείνοντας όλα τα σχετιζόμενα αρχεία και τέλος το απορρίπτουμε για την ορθή λειτουργία.

Η μέθοδος _addToLuceneIndexAd (επόμενο σχήμα) ελέγχει πρώτα αν μια αγγελία ήδη υπάρχει στο ευρετήριο (έλεγχος ID αγγελίας), εάν την βρει την σβήνει. Στη συνέχεια δημιουργείται ένα έγγραφο doc τύπου Document, το οποίο προσθέτει ξεχωριστά πεδία(Fields) για κάθε διαφορετική ιδιότητα και τιμή που υπάρχει στην συγκεκριμένη Αγγελία. Η μέθοδος κλείνει με τον IndexWriter να προσθέτει το έγγραφο αυτό στο ευρετήριο.

```
private static void _addToLuceneIndexAd(Ad ad, IndexWriter writer)
{
    // remove older index entry
    var searchQuery = new TermQuery(new Term("Id", ad.Id.ToString()));
    writer.DeleteDocuments(searchQuery);

    // add new index entry
    var doc = new Document();

    // add lucene fields mapped to db fields
    doc.Add(new Field("Id", ad.Id.ToString(), Field.Store.YES,
        Field.Index.NOT_ANALYZED));
    doc.Add(new Field("JobTitle", RemoveDiacritics(ad.AdJobs.JobDesc),
        Field.Store.YES, Field.Index.ANALYZED));
    doc.Add(new Field("AdvertiserId" ,ad.Advertiser.Id.ToString(),
        Field.Store.YES,Field.Index.NOT_ANALYZED));
    doc.Add(new Field("Description", RemoveDiacritics(ad.AdText),
        Field.Store.YES, Field.Index.ANALYZED));
    doc.Add(new Field("Area",RemoveDiacritics( ad.AreaName.ToLower()),
        Field.Store.YES, Field.Index.ANALYZED));
    if (ad.MunicipalityName != null)
    {
        doc.Add(new Field("Municipality",RemoveDiacritics(
            ad.MunicipalityName.ToLower()), Field.Store.YES, Field.Index.ANALYZED));
        // there is no city if there is not a municipality
        if (ad.CityName != null)
            doc.Add(new Field("City",RemoveDiacritics(ad.CityName.ToLower()),
                Field.Store.YES, Field.Index.ANALYZED));
    }

    doc.Add(new Field("type", "Ad", Field.Store.YES, Field.Index.NOT_ANALYZED));
    // add entry to index
    writer.AddDocument(doc);
}
```

Σχήμα 7: Με τη σειρά της η μέθοδος _addToLuceneIndexAd δέχεται ώς παραμέτρους μια αγγελία τύπου Ad και το IndexWriter

3.4.2 Παράμετροι πεδίου

Αναλυτικότερα όσον αφορά αυτή τη μέθοδο, κάθε πεδίο αφού δημιουργείται μπορεί να παίρνει κάποιες παραμέτρους κάποιες από τις οποίες οι πιο σημαντικές είναι:

- Ένα όνομα και όπως αναφέραμε και πριν, ένα έγγραφο μπορεί να έχει πάνω από ένα πεδίο με το ίδιο όνομα

- Τιμή για το όνομα της ιδιότητας αυτής, η οποία πρέπει να είναι πάντα σχεδόν σε μορφή συμβολοσειράς όταν προστίθεται στο έγγραφο (στο παράδειγμα μετατρέπουμε το Id τύπου Integer σε τύπου String)
- Την ιδιότητα Field.Store.*
- Την ιδιότητα Field.Index.*

3.4.2.1 Παράμετρος Field.Store

Η ιδιότητα **Field.Store** συνήθως συνοδεύεται στο τέλος είτε από κατάληξη Yes είτε από No (υπάρχει επίσης και ιδιότητα Compressed- για λόγο αναφοράς).

♦ Yes – Σημαίνει ότι η αρχική τιμή αποθηκεύεται στο ευρετήριο και μπορεί να ανακτηθεί από το IndexReader (το αντίστοιχο του IndexWriter για ανάγνωση). Αυτή η επιλογή είναι χρήσιμη όταν θέλουμε να χρησιμοποιήσουμε την προβολή των πεδίων κατά την εμφάνιση των αποτελεσμάτων (όπως π.χ. ο τίτλος ή το πρωτεύον κλειδί στη βάση δεδομένων). Καλό είναι να μην αποθηκεύουμε πολύ μεγάλα πεδία επειδή καταναλώνουν αρκετό χώρο στο ευρετήριο.

♦ No – Σημαίνει ότι δεν αποθηκεύουμε την αρχική τιμή στο ευρετήριο. Συνήθως αυτή την επιλογή χρησιμοποιούμε όταν δεν θέλουμε να προσθέσουμε στο ευρετήριο μεγάλο κείμενο που δεν έχει ανάγκη κιόλας να ανακτηθεί στην αρχική του μορφή, όπως π.χ. τα κείμενα των εγγράφων.

3.4.2.2 Παράμετρος Field.Index

Η ιδιότητα **Field.Index** προσδιορίζει για το πώς το κείμενο στο πεδίο θα είναι αναζητήσιμο μέσω του ευρετηρίου. Οι επιλογές είναι :

♦ Index.ANALYZED – Χρησιμοποιείται ο αναλυτής για να σπάσει την τιμή της ιδιότητας σε μια σειρά από ξεχωριστά στιγμιότυπα ώστε να κάνει κάθε στιγμιότυπο αναζητήσιμο. Η επιλογή αυτή είναι χρήσιμη για κανονικά πεδία με κείμενο.

♦ Index.NOT_ANALYZED – Προστίθεται κανονικά το πεδίο στο ευρετήριο, αλλά δεν αναλύεται η τιμή της συμβολοσειράς. Αντιθέτως ολόκληρη η τιμή χαρακτηρίζεται από ένα στιγμιότυπο και έτσι το κάνει προσβάσιμο. Δηλ. η επιλογή αυτή είναι χρήσιμη όταν θέλουμε να αναζητήσουμε πεδία τα οποία δεν έχουν σπάσει σε κομμάτια (στιγμιότυπα) όπως είναι ένας υπερσύνδεσμος, τα ονόματα , τηλ. νούμερα κ.α. Είναι ιδανική επιλογή για «ακριβής αντιστοίχιση»

♦ Index.NOT_ANALYZED_NO_NORMS – Είναι όπως και το Index.NOT_ANALYZED αλλά δεν αποθηκεύει κανόνες για την «τόνωση» (boost) των πεδίων κατά την αναζήτηση. Αυτό συμβάλλει ώστε να έχουμε λιγότερη χρήση της μνήμης κατά την αναζήτηση αλλά και λιγότερο χώρο στο ευρετήριο, επειδή τα μεμονωμένα στιγμιότυπα δεν χρειάζονται κανόνες εκτός και αν «τονώνονται».

♦ Index.ANALYZED_NO_NORMS – Το ίδιο όπως και προηγουμένως αλλά εφαρμόζεται ο αναλυτής.

♦ Index.NO – Η τιμή του πεδίου δεν προορίζεται για αναζήτηση.

Κεφάλαιο 4: Συστατικά Αναζήτησης

Η αναζήτηση είναι η διαδικασία της αναζήτησης λέξεων στο ευρετήριο ώστε να βρεθούν τα έγγραφα που περιέχουν τις λέξεις αυτές. Η ποιότητα μιας αναζήτησης τυπικά περιγράφεται χρησιμοποιώντας την ακρίβεια και τις μετρικές ανάκλησης. Η ανάκληση μετράει πόσο καλά το σύστημα αναζήτησης εντοπίζει τα σχετικά έγγραφα ενώ η ακρίβεια μετράει πόσο καλά το σύστημα αγνοεί τα μη σχετικά έγγραφα.

Όταν κάνουμε μια ερώτηση-αίτηση προς το ευρετήριο του Lucene, επιστρέφεται ένα στιγμιότυπο της κλάσης **TopDocs** που περιέχει ένα πίνακα **ScroreDoc** ταξινομημένο. Η ταξινόμηση από τη φύση της γίνεται σύμφωνα με το σκόρ. Το Lucene υπολογίζει ένα σκόρ (μια αριθμητική τιμή της σχετικότητας) για κάθε αρχείο που δέχεται την ερώτηση-αίτηση. Τα ScoreDocs από μόνα τους δεν είναι τα πραγματικά έγγραφα που αντιστοιχίζονται αλλά αναφορές προς τα έγγραφα αυτά μέσω του ακέραιου ID τους. Στις περισσότερες εφαρμογές που εμφανίζουν τα αποτελέσματα αναζήτησης, οι χρήστες προσπελαύνουν μόνο τα πρώτα μερικά έγγραφα, άρα δεν είναι απαραίτητο η ανάκτηση όλων των εγγράφων για όλα τα αποτελέσματα. Πρέπει να ανακτούμε μόνο τα έγγραφα που θα παρουσιαστούν στο χρήστη για τη συγκεκριμένη σελίδα. Στην πραγματικότητα, για πολύ μεγάλα ευρετήρια θα ήταν αδύνατον ή θα ήταν υπερβολική η συλλογή όλων των εγγράφων στην διαθέσιμη μνήμη υπολογιστή.

4.1 Διεπαφή αναζήτησης χρήστη (Search User Interface)

Η διεπαφή αναζήτησης χρήστη είναι αυτό που οι χρήστες βλέπουν στο περιηγητή, στην εφαρμογή υπολογιστών γραφείου ή στο κινητό όταν αλληλεπιδρούν με την εφαρμογή αναζήτησης. Η διεπαφή χρήστη είναι το πιο σημαντικό μέρος της εφαρμογής αναζήτησης, γιατί ακόμη και αν υπάρχει η πιο σπουδαία μηχανή αναζήτησης στο κόσμο πίσω από το κώδικα, στολισμένη με πολύ σπουδαίες λειτουργίες, με ένα ανόητο λάθος σε μια κατάλληλη λειτουργία μπορεί να μπερδέψει τους ασταθείς χρήστες οι οποίοι μετά μπορεί να προτιμήσουν κάποια άλλη ανταγωνιστική εφαρμογή αναζήτησης.

Η διεπαφή πρέπει να είναι απλή: δεν χρειάζεται να φαίνονται πολλές προχωρημένες επιλογές στην πρώτη σελίδα. Καλό είναι να προσφέρει κανείς ένα πανταχού παρών, διακεκριμένο κουτί αναζήτησης, που να φαίνεται παντού ώστε να μην χρειάζεται π.χ. κάποια διαδικασία δύο βημάτων (2 κλίκ) για να μπορέσει να γράψει στο κουτί αναζήτησης (αυτό είναι ένα πολύ συνηθισμένο λάθος).

Δεν πρέπει επίσης κανείς να υποτιμά την αναγκαιότητα της προβολής αποτελεσμάτων. Απλές λεπτομέρειες, όπως η χρησιμοποίηση μικρού μεγέθους γραμματοσειράς και το γέμισμα ενός αποτελέσματος με υπερβολικό κείμενο μπορούν να απογοητεύσουν την εμπειρία αναζήτησης του χρήστη. Ακόμη πρέπει να είναι σίγουρος κανείς ότι η σειρά ταξινόμησης είναι κατάλληλη ως προς τα αποτελέσματα. Ένα άλλο σημαντικό στοιχείο είναι η πλήρης διαφάνεια: αν ή

εφαρμογή αναζήτησης κάνει κάτι «ενδιαφέρον» όπως επεκτείνει την αναζήτηση για να περιλαμβάνει συνώνυμα, χρησιμοποιεί «τονώσεις» για να επηρεάσει την ταξινόμηση ή αυτόματα διορθώνει την ορθογραφία πρέπει να φαίνονται όλα αυτά πάνω από τα αποτελέσματα και να είναι εύκολο να τα απενεργοποιήσει κανείς.

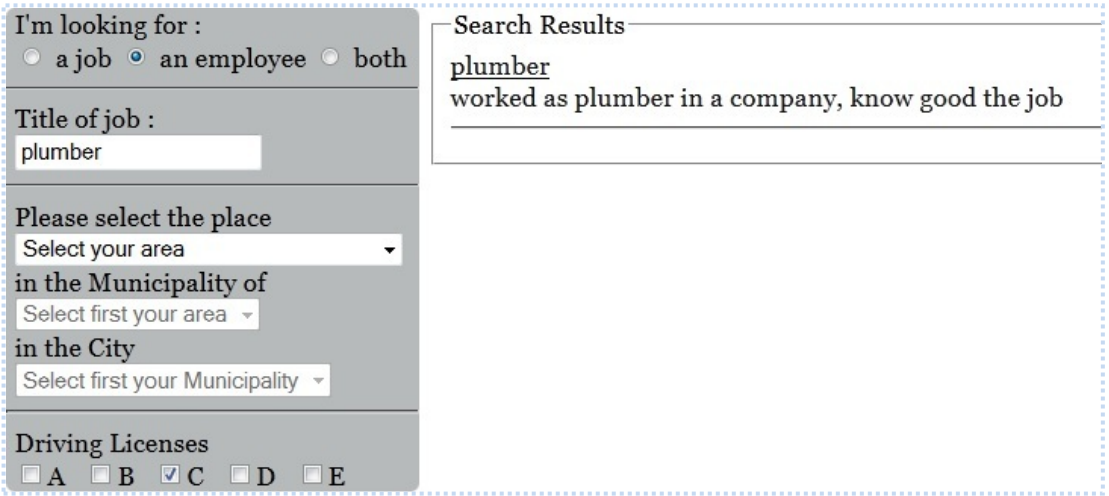
Το χειρότερο πράγμα , που συμβαίνει και αρκετά συχνά είναι, να διαβρώνεται ή εμπιστοσύνη του χρήστη στα αποτελέσματα αναζήτησης. Όταν γίνεται αυτό οι χρήστες συνήθως αποσύρονται ήσυχα και δεν υπάρχει σχεδόν ποτέ η δυνατότητα να κερδίσεις την εμπιστοσύνη τους ξανά.

Το καλύτερο είναι ο ίδιος προγραμματιστής να χρησιμοποιεί την μηχανή αναζήτησης όσο πιο εκτενώς είναι δυνατόν. Να μπορεί να αισθανθεί θετικά με τα εύστοχα σημεία που έχει πετύχει αλλά πάνω απ'όλα να διορθώσει όλα τα αρνητικά που μπορεί να βρει μπροστά του. Σχεδόν πάντα η διεπαφή αναζήτησης πρέπει να προσφέρει τον έλεγχο και τη διόρθωση της ορθογραφίας. Το Lucene έχει μια συνεισφέρον μονάδα, τον ορθογράφο που μπορεί να χρησιμοποιηθεί άνετα για το σκοπό αυτό.

Το Lucene δεν προσφέρει κάποια τυποποιημένη διεπαφή χρήστη: είναι εντελώς στις δυνατότητες του προγραμματιστή να την κατασκευάσει. Το σημαντικότερο είναι να χαρακτηρίζεται από απλότητα και φιλικότητα προς χρήση.

4.1.1 Μία διεπαφή αναζήτησης από την εφαρμογή JobFinder

Εδώ παραθέτουμε ένα παράδειγμα μιας από τις διάφορες διεπαφές αναζήτησης που υπάρχουν στην εφαρμογή μας:



Σχήμα 8: Η διεπαφή του LiveSearch στην εφαρμογή μας

Στο παραπάνω σχήμα φαίνεται η διεπαφή αναζήτησης LiveSearch στην εφαρμογή JobFinder. Από τα αριστερά διακρίνουμε το μενού επιλογών (όλο το γκρι παράθυρο) για τον χρήστη. Στην πρώτη γραμμή μπορούμε να επιλέξουμε τι ακριβώς ψάχνουμε δηλ. είτε δουλειά είτε εργαζόμενο είτε και τα δύο! Ακολουθεί ο τίτλος της δουλειάς που μας ενδιαφέρει. Ύστερα ακολουθούν οι περιοχές δηλ. με λογική σειρά (γεωγραφική περιοχή, ύστερα δήμος, ύστερα πόλη). Το τελευταίο

μενού από κάτω αφορά τα διπλώματα που θέλουμε να έχει ο εργαζόμενος που ψάχνουμε (περίπτωση μόνο για την εύρεση εργαζομένου). Από τα δεξιά υπάρχει η στήλη με τα αποτελέσματα που προκύπτουν σύμφωνα με τις επιλογές που έχουμε κάνει. Σ'αυτή τη περίπτωση φαίνεται ότι θέλουμε να βρούμε εργαζόμενο που έχει δουλέψει υδραυλικός (plumber) και να έχει κατηγορία διπλώματος C. Όταν κάνουμε κλίκ σε ένα αποτέλεσμα θα μεταφερθούμε σε άλλη σελίδα για περισσότερες πληροφορίες σχετικά με τον εργαζόμενο.

4.1.2 Οι βασικές κλάσεις μιας διεπαφής αναζήτησης

Εδώ αναφέρονται οι πιο σημαντικές κλάσεις για μια διεπαφή αναζήτησης του χρήστη:

- **IndexSearcher** : Είναι η κεντρική πύλη αναζήτησης ενός ευρετηρίου. Όλες οι αναζητήσεις προέρχονται από ένα στιγμιότυπο του IndexSearcher χρησιμοποιώντας κάποιες από τις υπερφορτωμένες μεθόδους αναζήτησης. Όπως είναι το IndexWriter για την δημιουργία ευρετηρίου έτσι είναι και το IndexSearcher για την αναζήτηση. Απαιτεί ένα στιγμιότυπο του Directory, που κρατάει το ευρετήριο το οποίο προηγουμένως δημιουργήθηκε, και προσφέρει τις μεθόδους αναζήτησης κάποιες από τις οποίες υλοποιούνται στην αφηρημένη κλάση Searcher. Στην απλούστερη περίπτωση μια μέθοδος αναζήτησης παίρνει σαν παραμέτρους ένα αντικείμενο Query και ένα int topN μέτρημα και επιστρέφει ένα αντικείμενο TopDocs. Αναλύεται λεπτομερώς στη συνέχεια.
- **Term (Όρος) :** Ο όρος είναι η βασική μονάδα αναζήτησης. Παρόμοια με το αντικείμενο Field, αποτελείται από ένα ζευγάρι συμβολοσειράς: το όνομα του πεδίου και τη τιμή του πεδίου. Χρειάζεται να τονιστεί ότι τα αντικείμενα τύπου Όρος χρησιμοποιούνται και στην διαδικασία δημιουργίας ευρετηρίου. Παρόλα αυτά οι τελευταίοι δημιουργούνται απ'τις εσωτερικές λειτουργίες του Lucene χωρίς να χρειάζεται να ανησυχούμε γι'αυτές. Κατά την αναζήτηση, μπορούμε να κατασκευάσουμε αντικείμενα τύπου όρου και να τα χρησιμοποιήσουμε μαζί με το TermQuery όπως για παράδειγμα :

```
Query q=new TermQuery(new Term("contents","lucene"));
TopDocs hits=searcher.search(q,10);
```

Ο κώδικας αυτός παροτρύνει το Lucene να αναζητήσει τα κορυφαία 10 έγγραφα που περιέχουν τη λέξη "lucene" σε ένα πεδίο που λέγεται "contents",ταξινομώντας τα έγγραφα σε φθίνουσα σχετικότητα. Επειδή το αντικείμενο TermQuery προέρχεται από την αφηρημένη γονική κλάση Query, μπορούμε να χρησιμοποιήσουμε τον τύπο Query από τα αριστερά της δήλωσης.

- **Query (και υποκλάσεις) :** Το Lucene έρχεται με έναν αριθμό από συγκεκριμένες υποκλάσεις τύπου Query. Εκτός από το TermQuery υπάρχουν και οι τύποι BooleanQuery, PhraseQuery, PrefixQuery, PhrasePrefixQuery,

TermRangeQuery, NumericRangeQuery, FilteredQuer και SpanQuery οι οποίοι θα αναλυθούν στη συνέχεια. Το Query είναι η κοινή αφηρημένη γονική κλάση τους. Περιέχει μερικές ωφέλιμες μεθόδους, οι πιο ενδιαφέρον από τις οποίες είναι η setBoost (float) η οποία τονώνει κάποιες υποαιτήσεις-ερωτήσεις περισσότερο από άλλες για διαμόρφωση ανάλογου τελικού σκοραρίσματος.

- **QueryParser :** Μετατρέπει μια έκφραση κειμένου που αιτήθηκε από κάποιον χρήστη σε ένα κανονικό αντικείμενο αίτησης-ερώτησης (Query).Αναλύεται λεπτομερώς στη συνέχεια.
- **TopDocs :** Περιέχει τα πιο υψηλά σε σκόρ έγγραφα, που έχουν επιστραφεί από το IndexSearcher.search
- **ScoreDoc :** Παρέχει πρόσβαση σε κάθε αποτέλεσμα αναζήτησης από τα TopDocs

Όταν φτιάχνει κανείς τον τρόπο αναζήτησης με το Lucene υπάρχουν δύο τρόποι για να το κάνει: είτε κατασκευάζει την αίτηση- ερώτηση με προγραμματισμό είτε χρησιμοποιεί το **QueryParser** του Lucene για να μεταφράζει το κείμενο που θα δέχεται από το χρήστη σε κατάλληλη αίτηση-ερώτηση. Η πρώτη προσέγγιση δίνει τον απόλυτο έλεγχο, ώστε η εφαρμογή να έχει οποιαδήποτε διεπαφή μέσω της οποίας θα υπάρχει αλληλεπίδραση με την δημιουργία ερωτήσεων-αιτήσεων. Η δεύτερη προσέγγιση είναι από την άλλη πλευρά πολύ εύκολη στη χρήση, και προσφέρει μία τυποποιημένη σύνταξη αναζήτησης. Και στις δύο περιπτώσεις αναλαμβάνει να εκτελέσει ύστερα την αναζήτηση ο IndexSearcher. Δίνεται περιγραφή στη συνέχεια και για τις δύο τεχνικές καθώς και το τρόπος που λειτουργούν.

4.2 Χτίσιμο μιας αίτησης-ερώτησης (Query)

Από το σημείο που ένας χρήστης αρχίσει να χρησιμοποιεί την εφαρμογή αναζήτησής μας, καθώς αλληλεπιδρά, θα εκδίδει συχνά αιτήσεις-ερωτήσεις αναζήτησης. Οι αιτήσεις αυτές μπορούν να προκύπτουν σαν αποτέλεσμα μιας HTML φόρμας ή ενός Ajax αιτήματος που υποβάλλεται από τον περιηγητή στον διακομιστή (server). Η κάθε αίτηση αυτή θα πρέπει ύστερα να μεταφράζεται σε ένα αντικείμενο Query της μηχανής αναζήτησης. Αυτό το βήμα ονομάζεται «Χτίσιμο Αίτησης - Ερώτησης (Query)».

Οι αιτήσεις-ερωτήσεις μπορούν να είναι απλές ή σύνθετες. Το Lucene προσφέρει ένα δυναμικό πακέτο, με όνομα QueryParser, για να μεταφράζει το κείμενο του χρήστη σε ένα αντικείμενο αίτησης-ερώτησης σύμφωνα με μια κοινώς αποδεκτή σύνταξη.

4.2.1 Χρησιμοποιώντας τον QueryParser

Ο QueryParser αρχικοποιείται όπως θα δούμε με τρείς παραμέτρους *:* matchVersion, field και analyzer.

QueryParser parser=new QueryParser (Version matchVersion, String field, Analyzer analyzer)

- Το **matchVersion** τύπου Version δίνει οδηγία στο Lucene για το ποια έκδοση να χρησιμοποιήσει για τις προκαθορισμένες ρυθμίσεις και αντιστοιχίσεις, ώστε να υπάρχει η συμβατότητα με παλιότερες εκδόσεις. Μερικές φορές μπορεί και να υπάρχουν όμως σφάλματα με τις παλιότερες εκδόσεις.
- Το **field** τύπου Συμβολοσειράς (String) είναι το όνομα πεδίου στο οποίο θα γίνει η αναζήτηση όλων των όρων, εκτός και αν το κείμενο προς αναζήτηση θα περιέχει ρητά ένα διαφορετικό όνομα πεδίου χρησιμοποιώντας τη σύνταξη “field:text”. Το στιγμιότυπο του QueryParser έχει μια μέθοδο **parse()** για απλούστερη χρήση:
- Το **analyzer** τύπου Analyzer καθορίζει τον αναλυτή που θα χρησιμοποιηθεί.

Εάν η έκφραση δεν μπορέσει να αναλυθεί, γίνεται εξαίρεση τύπου ParseException ,μια κατάσταση που η εφαρμογή μας θα πρέπει να μπορεί να χειριστεί ορθά. Το μήνυμα που θα εμφανίσει η εξαίρεση θα είναι μια δικαιολογημένη ένδειξη γιατί απέτυχε η ανάλυση αν και η περιγραφή αυτή ενδέχεται να είναι δύσκολα κατανοητή από τους τελικούς χρήστες.

public Query parse(String query) throws ParseException

όπου το String query είναι η έκφραση που θα αναλυθεί (parsed) όπως είναι το “+cat +dog”.

Η μέθοδος **parse()** είναι γρήγορη και απλή προς χρήση, αλλά μπορεί και να μην είναι αρκετή. Υπάρχουν αρκετές ρυθμίσεις που μπορούν να ελεγχθούν σε ένα στιγμιότυπο του QueryParser, όπως είναι ο εξ ορισμού τελεστής “OR” όταν χρησιμοποιούνται πολλαπλοί όροι. Οι ρυθμίσεις αυτές περιλαμβάνουν επίσης τοπικές ρυθμίσεις (για ανάλυση ημερομηνιών), την ελάχιστη ομοιότητα και μήκος σημείων στίξης για ασαφής αιτήσεις –ερωτήσεις καθώς και άλλες πολλές προχωρημένες ρυθμίσεις.

4.2.2 Χειρισμός βασικών εκφράσεων αιτήσεων-ερωτήσεων με το QueryParser

Ο QueryParser μεταφράζει τις εκφράσεις αιτήσεων-ερωτήσεων σε έναν από τους ενσωματωμένους τύπους αιτήσεων-ερωτήσεων του Lucene.

Παραθέτουμε κάποια παραδείγματα εκφράσεων και την μετάφρασή τους.

Έκφραση αίτησης – ερώτησης	**Τα έγγραφα που αντιστοιχίζονται σ’αυτήν…**

lucene	Περιέχουν τον όρο lucene στο προκαθορισμένο πεδίο
lucene index lucene OR index	Περιέχουν τον όρο lucene ή index , ή και τα δύο στο προκαθορισμένο πεδίο
+lucene +index lucene AND ndex	Περιέχουν και τη λέξη lucene και τη λέξη index στο προκαθορισμένο πεδίο
title : ant	Περιέχουν τον όρο ant στο πεδίο title
title : extreme-subject : sports	Περιέχουν τη λέξη extreme στο πεδίο title αλλά δεν περιέχουν τη λέξη sports στο πεδίο subject
title : “lucene in action”	Περιέχουν ακριβώς την φράση “lucene in action” στο πεδίο title
apple*	Περιέχουν όρους που ξεκινάνε με apple, όπως applepie, applecake καθώς και την ίδια λέξη apple
apple~	Περιέχουν όρους που είναι κοντά στη λέξη apple, όπως opple

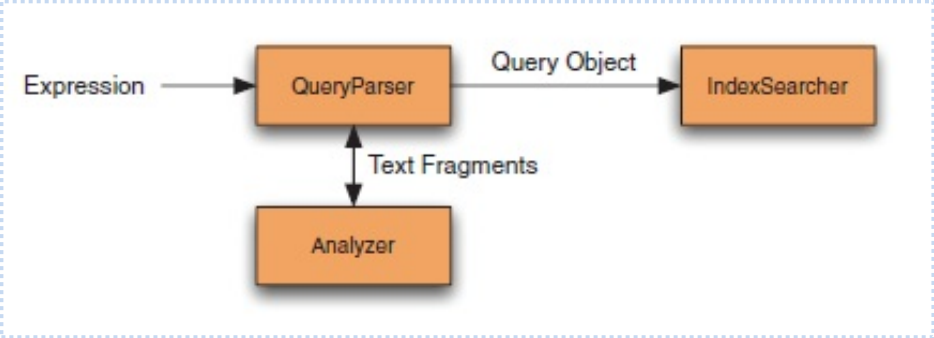
Χρειάζεται να τονιστεί ότι όποτε κάποιοι ειδικοί χαρακτήρες χρησιμοποιούνται σε μια έκφραση αίτησης-ερώτησης, πρέπει να μπορούμε να παρέχουμε ένα μηχανισμό ώστε να μπορούν οι χαρακτήρες αυτοί να χρησιμοποιηθούν έτσι όπως είναι από τη φύση τους χωρίς κάποια συγκεκριμένη σημασία. Ο QueryParser χρησιμοποιεί την κάθετο \ για την αποφυγή τέτοιων χαρακτήρων μέσα στους όρους. Οι χαρακτήρες αυτοί είναι οι εξής :

\ + - ! ( ) : ^ [ ] { } ~ * ?

Η ερώτηση μπορεί να περιλαμβάνει Boolean λειτουργίες , ερωτήσεις φράσεων ανάμεσα σε «» ή όρους μπαλαντέρ. Αν η εφαρμογή διαθέτει παραπάνω λειτουργίες στην διεπαφή χρήστη, πρέπει να υπάρχει μια λογική ώστε να μεταφράζεται αυτό σε κατάλληλη αίτηση-ερώτηση. Αυτό επιτυγχάνεται μέσω κάποιων φίλτρων.

4.2.3 Αναλύοντας μια αίτηση-ερώτηση με τον QueryParser

Οι μέθοδοι αναζήτησης του Lucene απαιτούν ένα αντικείμενο τύπου Query (αίτησης-ερώτησης). Αναλύοντας μια έκφραση αίτησης-ερώτησης είναι η πράξη της μετατροπής μιας αίτησης-ερώτησης του χρήστη όπως π.χ. “Lucene OR Analyze” σε ένα κατάλληλο αντικείμενο στιγμιότυπου αίτησης-ερώτησης. Σ’αυτήν τη περίπτωση το αντικείμενο αίτησης-ερώτησης θα ήταν ένα στιγμιότυπο του BooleanQuery με δύο προαιρετικές προτάσεις, μια για κάθε όρο. Η διαδικασία αυτή απεικονίζεται στο ακόλουθο σχήμα :



Εικόνα 9: Ο QueryParser μεταφράζει μια έκφραση κειμένου απ'τον τελικό χρήστη σε μια αυθαίρετη σύνθετη αίτηση-ερώτηση προς αναζήτηση. Επίσης διακρίνουμε ότι ο QueryParser απαιτεί έναν αναλυτή για να μπορέσει να σπάσει το κείμενο της αίτησης-ερώτησης σε όρους.

Θα μπορούσαμε να πούμε ότι οι εκφράσεις αιτήσεων-ερωτήσεων είναι παρόμοιες με τις εκφράσεις SQL που χρησιμοποιούνται για να κάνουν ερωτήσεις προς μια βάση δεδομένων ως προς το γεγονός ότι η αίτηση-ερώτηση πρέπει να αναλυθεί σε κάτι πιο χαμηλού επιπέδου ώστε ο διακομιστής της βάσης δεδομένων να μπορέσει να την καταλάβει.

Αξίζει να σημειωθεί ότι το αρχικό κείμενο μπορεί να έχει περάσει μια κανονικοποίηση σε όρους από τον αναλυτή, πράγμα το οποίο μπορεί να έχει αφαιρέσει κάποιους όρους όπως τις λέξεις διακοπής, ή να έχει κάνει τα γράμματα πεζά κ.α. Γιαυτό είναι πολύ σημαντικό ότι οι όροι που περνάνε στον IndexSearcher να είναι παρόμοιοι μ΄αυτούς που δημιουργούνται από την ανάλυση των πηγαίων εγγράφων κατά την δημιουργία ευρετηρίου. Συχνά το QueryParser του Lucene είναι αρκετό για μια εφαρμογή. Άλλες φορές όμως μπορεί να χρειαστούμε να χρησιμοποιήσουμε την έξοδο του QueryParser προσθέτοντας μετά τη δική μας λογική για να βελτιώσουμε στη συνέχεια την ερώτηση. Μερικές φορές μπορεί να θελήσουμε να τροποποιήσουμε τη σύνταξη του QueryParser, ή να τροποποιήσουμε τα στιγμιότυπα που δημιουργεί πράγμα το οποίο είναι απλό χάρη στον ανοιχτό κώδικα του Lucene.

4.2.4 Παράδειγμα QueryParser στην εφαρμογή μας

Χωρίς καμία αμφιβολία χρειάστηκε η χρήση του QueryParser και στην δική μας εφαρμογή. Τα παρακάτω σχήματα δείχνουν τον τρόπο :

```
private static Query parseQuery(string searchQuery, QueryParser parser)
{
    Query query;
    try
    {
        query = parser.Parse(searchQuery.Trim());
    }
    catch (ParseException)
    {
        query = parser.Parse(QueryParser.Escape(searchQuery.Trim()));
    }
    return query;
}
```

Σχήμα 10: Η γραμμή που περιέχει το parseQuery επεξηγείται στο επόμενο σχήμα

```
if (!(string.IsNullOrEmpty(model.JobTitle))) //i an to jobtitle den einai keno
{
    var parser2 = new MultiFieldQueryParser(Version.LUCENE_29, new[] {"JobTitle", "Description"},
                                            analyzer);
    query2 = parseQuery(model.JobTitle, parser2);
    booleanQuery.Add(query2, BooleanClause.Occur.MUST);
}
```

Σχήμα 11: Η στατική μέθοδος parseQuery

Στο πρώτο σχήμα βλέπουμε ένα σημείο κώδικα που αρχικοποιείται ο parser2 ο οποίος είναι τύπου MultifieldQueryParser ώστε να μπορέσουμε να αναζητήσουμε σε πολλά πεδία ταυτόχρονα (JobTitle, Description). Ύστερα αρχικοποιείται το query2 το οποίο δέχεται σαν τιμή, την αίτηση-ερώτηση που επιστρέφει η μέθοδος parseQuery που αναλύεται στο δεύτερο σχήμα. Η μέθοδος αυτή παίρνει σαν παραμέτρους την τιμή που θέλουμε να αναζητήσουμε (τον τίτλο της δουλειάς στην προκειμένη περίπτωση) και τον parser2 δηλ. τα πεδία που θέλουμε να αναζητήσουμε. Η αίτηση-ερώτηση που δημιουργείται πρώτα αναλύεται (parser.Parse) αφού έχουν «καθαριστεί» οι κενοί χαρακτήρες στην αρχή και στο τέλος της τιμής που θέλουμε να αναζητήσουμε. Σε περίπτωση που αποτύχει για κάποιο λόγο τότε γίνεται παρόμοια προσπάθεια στο catch που δέχεται σαν παράμετρο την εξαίρεση, και η αίτηση-ερώτηση δημιουργείται με τον ίδιο τρόπο μόνο που η τιμή πρώτα φιλτράρεται από τη μέθοδο Escape του QueryParser. Η μέθοδος αυτή επιστρέφει την συμβολοσειρά αφού έχουν αποφευχθεί οι χαρακτήρες μετά το σύμβολο \ σύμφωνα με αυτά που αναμένει ο QueryParser. Ύστερα επιστρέφεται σαν τιμή αυτή η αίτηση-ερώτηση πίσω στο query2 το οποίο προστίθεται στο booleanQuery. Το booleanQuery είναι η σύνθετη αίτηση-ερώτηση που προσθέτει υποαιτήσεις-ερωτήσεις στον εαυτό της για να γίνει η αίτηση-ερώτηση ακριβώς στα μέτρα που θέλουμε.

Για ανάκτηση αποτελεσμάτων εκτελούμε την αναζήτηση αίτησης-ερώτησης

4.3 Αναζήτηση αίτησης-ερώτησης

Η αναζήτηση αίτησης-ερώτησης είναι η διαδικασία συμβούλευσης του ευρετηρίου για την αναζήτηση και ανάκτηση των εγγράφων που αντιστοιχούν στην αίτηση-ερώτηση, ταξινομημένα με τον ανάλογο τρόπο. Αυτή η διαδικασία συμπληρώνει τις σύνθετες εσωτερικές δουλειές μιας μηχανής αναζήτησης, και το Lucene μπορεί να την διαχειριστεί για μας όπως και άλλες διαδικασίες. Το Lucene είναι εκπληκτικά επεκτάσιμο ώστε άμα θέλει κανείς να τροποποιήσει τον τρόπο που τα αποτελέσματα συλλέγονται, φιλτράρονται και τακτοποιούνται είναι πολύ απλό.

Υπάρχουν 3 κυρίως γενικές θεωρητικές μέθοδοι αναζήτησης:

- Καθαρά Δυαδικό (Boolean) Μοντέλο – Τα έγγραφα είτε ταιριάζουν ακριβώς στην ερώτηση είτε όχι ,το σκοράρισμα δεν γίνεται καθόλου. Στο μοντέλο αυτό δεν υπάρχουν μη συναφή σκόρ που σχετίζονται με ταιριαστά έγγραφα, τα οποία είναι μη ταξινομημένα.
- Μοντέλο Διανυσματικού Χώρου – Και οι ερωτήσεις και τα έγγραφα μοντελοποιούνται ως διανύσματα σε ένα υψηλά διανυσματικό χώρο, όπου κάθε μοναδικός όρος είναι ένα διάνυσμα. Η σχετικότητα ή ομοιότητα, μεταξύ μιας ερώτησης και ενός εγγράφου υπολογίζεται από το μέγεθος απόστασης διανύσματος μεταξύ των διανυσμάτων.
- Μοντέλο Πιθανοτήτων – Σ'αυτό το μοντέλο , υπολογίζουμε την πιθανότητα ένα έγγραφο να ταιριάζει καλά ως προς μια ερώτηση-αίτημα, χρησιμοποιώντας ολοκληρωτική προσέγγιση πιθανοτήτων.

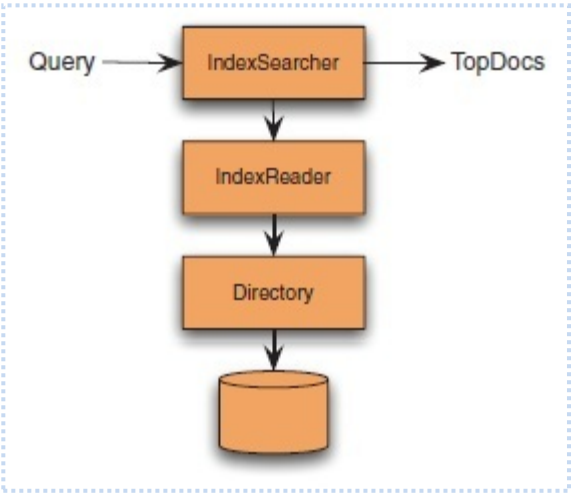
Η προσέγγιση του Lucene συνδυάζει το Διανυσματικό χώρο με καθαρά Δυαδικά μοντέλα, και προσφέρει την επιλογή ώστε να αποφασίσει κανείς ποιο μοντέλο θα ήθελε να χρησιμοποιεί σε βάση αναζήτησης-προς-αναζήτηση.

4.3.1 Χρησιμοποιώντας τον IndexSearcher

Κάνοντας αναζητήσεις με το Lucene μπορεί πραγματικά να αποτελέσει κάτι πολύ απλό. Στη περίπτωση του IndexSearcher πρώτα δημιουργούμε ένα στιγμιότυπο του, το οποίο ανοίγει ένα ευρετήριο αναζήτησης και μετά χρησιμοποιούμε τις μεθόδους αναζήτησης πάνω σε εκείνη τη κλάση που θέλουμε να κάνουμε την αναζήτηση. Η κλάση TopDocs που επιστρέφεται αναπαριστά τα πιο πετυχημένα αποτελέσματα τα οποία ύστερα παρουσιάζονται στο χρήστη.

4.3.1.1 Δημιουργώντας έναν IndexSearcher

Όπως και τα υπόλοιπα μέρη της κυρίως διεπαφής του Lucene, το IndexSearcher είναι εύκολο να χρησιμοποιηθεί. Οι κλάσεις που εμπλέκονται απεικονίζονται στο ακόλουθο σχήμα :



Σχήμα 12: Η σχέση μεταξύ των κοινών κλάσεων που χρησιμοποιούνται για αναζήτηση.

Παίρνοντας τα πράγματα με τη σειρά το πρώτο που χρειάζεται είναι, όπως και στην δημιουργία ευρετηρίου, το Directory γιατί τις περισσότερες φορές η αναζήτηση γίνεται στο ευρετήριο ενός συστήματος αρχείων.

Directory dir. = FSDirectory.open (new File("path/to/index"));

ύστερα δημιουργούμε ένα IndexReader :

IndexReader reader = IndexReader.open(dir);

και τέλος δημιουργούμε το IndexSearcher :

IndexSearcher searcher = new IndexSearcher(reader);

Το Directory παρέχει το αφηρημένο αρχείο όπως το API (Application Programming Interface). Ο IndexReader χρησιμοποιεί αυτό το API για να αλληλεπιδρά με τα αρχεία ευρετηρίου που αποθηκεύονται κατά την δημιουργία ευρετηρίου και εκθέτει το χαμηλού επιπέδου API που χρησιμοποιεί ο IndexSearcher για αναζήτηση. Το API του IndexSearcher δέχεται αντικείμενα αίτησης-ερώτησης για να κάνει αναζητήσεις και επιστρέφει αντικείμενα τύπου TopDocs προβάλλοντας έτσι τα αποτελέσματα.

Είναι αξιοσημείωτο ότι ο IndexReader κάνει όλη τη βαριά δουλειά ανοίγοντας όλα τα αρχεία ευρετηρίου και εκθέτοντας ένα χαμηλό επίπεδο ανάγνωσης API, ενώ ο IndexSearcher θα μπορούσαμε να πούμε πώς λειτουργεί περισσότερο σαν «διακοσμητικό». Επειδή είναι ακριβή εργασία το άνοιγμα του IndexReader είναι καλύτερη λύση η επαναχρησιμοποίηση ενός στιγμιότυπου για όλες τις αναζητήσεις, και το άνοιγμα να γίνεται όταν είναι εντελώς απαραίτητο.

Επίσης είναι δυνατόν να δημιουργήσουμε άμεσα έναν IndexSearcher από ένα Directory, ο οποίος θα δημιουργήσει ύστερα τον δικό του IndexReader. Με την υλοποίηση αυτή μπορεί να ανακτήσει κανείς τον διέπων IndexReader καλώντας την μέθοδο get-IndexReader, αλλά αν κλείσουμε τον searcher αυτός θα κλείσει και τον IndexReader αντίστοιχα, επειδή αυτός τον είχε ανοίξει. Ο IndexReader πάντα ψάχνει ένα στιγμιότυπο-στον-χρόνο του ευρετηρίου σαν να υπήρχε από πριν όταν είχε δημιουργηθεί ο ίδιος. Αν χρειάζεται να ψάξουμε για τις αλλαγές που έγιναν στο ευρετήριο πρέπει να ανοίξουμε ένα καινούργιο Reader. Ευτυχώς η μέθοδος IndexReader.Reopen() είναι μια πραγματοποιήσιμη λύση αποκτώντας ένα καινούργιο IndexReader που καλύπτει όλες τις αλλαγές στο ευρετήριο αλλά μοιράζεται τις πηγές με τον συγκεκριμένο Reader όταν είναι δυνατόν. Χρησιμοποιείται ως ακολούθως :

```
IndexReader newReader = reader.Reopen();
if (reader != newReader) {
reader.Close();
reader = newReader;
searcher = new IndexSearcher(reader);
}
```

Η μέθοδος Reopen επιστρέφει έναν καινούργιο Reader αν υπήρξαν αλλαγές στο ευρετήριο, περίπτωση στην οποία είναι δική μας ευθύνη να κλείσουμε τον παλιό Reader και να δημιουργήσουμε έναν καινούργιο IndexSearcher. Σε μια πραγματική πολυνηματική εφαρμογή ,όπου μπορούν να υπάρξουν πολλές ταυτόχρονες αναζητήσεις χρησιμοποιώντας ένα παλιό Reader, πρέπει να μπορούμε να προστατεύουμε αυτό το κώδικα για να είναι ασφαλής.

4.3.1.2 Πραγματοποιώντας αναζητήσεις

Από τη στιγμή που έχουμε έναν IndexSearcher, πολύ απλά καλούμε μια από τις search μεθόδους του για να μπορέσουμε να πραγματοποιήσουμε μια αναζήτηση. Η μέθοδος αναζήτησης κάνει πολύ μεγάλη δουλειά και πολύ γρήγορα στο παρασκήνιο. Επισκέφτεται κάθε μοναδικό έγγραφο που είναι υποψήφιο για την αντιστοίχιση της αναζήτησης, και αποδέχεται μόνο εκείνα τα οποία ξεπερνούν κάθε περιορισμό στην αίτηση-ερώτηση. Τέλος, συγκεντρώνει τα πιο κορυφαία αποτελέσματα και τα επιστρέφει σε εμάς.

Οι κύριες μέθοδοι αναζήτησης διαθέσιμες σε ένα στιγμιότυπο του IndexSearcher φαίνονται στο παρακάτω σχήμα. Οι πιο πολλές μέθοδοι αναζήτησης του IndexSearcher επιστρέφουν αντικείμενα τύπου TopDocs τα οποία θα αναλυθούν στη συνέχεια.

Υπογραφή μεθόδου IndexSearcher.search	**Πότε χρησιμοποιείται**
TopDocs search (Query query, int n)	Ευθείες αναζητήσεις. Η παράμετρος n δείχνει πόσα κορυφαία-σε σκόρ έγγραφα να επιστραφούν

TopDocs search(Query query, Filter filter, int n)	Οι αναζητήσεις περιορίζονται σε ένα υποσύνολο των διαθέσιμων εγγράφων, σύμφωνα με κάποιο κριτήριο φίλτρου
TopFieldDocs search(Query query, Filter filter, int n, Sort sort)	Όπως και το προηγούμενο ακριβώς αλλά επίσης υπάρχει και η ταξινόμηση ανά προσαρμοσμένο αντικείμενο.
Void search (Query query, Collector results)	Χρησιμοποιείται όταν υπάρχει προσαρμοσμένη λογική για να υλοποιηθεί σε κάθε έγγραφο που επισκεφτήκαμε, ή αν θα θέλαμε να συλλέξουμε ένα διαφορετικό υποσύνολο εγγράφων απ'ότι τα κορυφαία N σύμφωνα με τα κριτήρια ταξινόμησης
Void search (Query query, Filter filter, Collector results)	Όπως και το προηγούμενο μόνο που τα έγγραφα δέχονται μόνο αν περνούν από το κριτήριο φίλτρου

4.3.1.3 Δουλεύοντας με τα TopDocs

Όταν καλούμε την μέθοδο search, έχουμε στη διάθεσή μας ένα αντικείμενο TopDocs που μπορούμε να χρησιμοποιήσουμε για αποτελεσματική πρόσβαση στα αποτελέσματα αναζήτησης. Τυπικά , επιλέγουμε μια από τις παραπάνω μεθόδους αναζήτησης που επιστρέφουν ένα αντικείμενο TopDocs. Τα αποτελέσματα ταξινομούνται ανά σχετικότητα – η καλύτερα ανά πόσο καλά κάθε έγγραφο αντιστοιχεί σε αίτηση-ερώτηση.

Η κλάση TopDocs εκθέτει ένα μικρό αριθμό μεθόδων και χαρακτηριστικών για ανάκτηση των αποτελεσμάτων αναζήτησης. Το χαρακτηριστικό **TopDocs.totalHits** επιστρέφει τον αριθμό των εγγράφων που αντιστοιχήθηκαν. Οι αντιστοιχίσεις, εξ ορισμού, ταξινομούνται ανά φθίνουσα σε σκόρ σειρά. Το χαρακτηριστικό **TopDocs.scoreDocs** είναι ένας πίνακας που περιέχει τον επιθυμητό αριθμό κορυφαίων αποτελεσμάτων. Κάθε στιγμιότυπο του ScoreDoc έχει ένα σκόρ με αριθμό κινητής υποδιαστολής, το οποίο είναι είναι το σκόρ σχετικότητας και έναν ακέραιο doc, το οποίο είναι το ID του εγγράφου που μπορεί να χρησιμοποιηθεί για την ανάκτηση των αποθηκευμένων πεδίων για το έγγραφο αυτό καλώντας το IndexSearcher.document (doc). Τέλος, το TopDocs.**getMaxScore()** επιστρέφει το πιο κορυφαίο σκόρ ανάμεσα σε όλες τις αντιστοιχίσεις , έτσι όταν έχουμε ταξινόμηση ανά σχετικότητα αυτό θα είναι πάντα το σκόρ του πρώτου αποτελέσματος.

4.3.2 Αναζήτηση ενός όρου στο ευρετήριο

Άς δώσουμε το πρώτο και το πιο απλό παράδειγμα αναζήτησης ενός όρου στα έγγραφα ενός ευρετηρίου. Το IndexSearcher όπως είπαμε είναι η κεντρική κλάση που χρησιμοποιείται για την αναζήτηση των εγγράφων σε ένα ευρετήριο. Διαθέτει κάποιες υπερφορτωμένες μεθόδους αναζήτησης και μπορούμε να αναζητήσουμε κάποιον όρο χρησιμοποιώντας κάποια από αυτές. Ένας όρος είναι μια τιμή Συμβολοσειράς που την «ζευγαρώνουμε» με το όνομα πεδίου που την περιέχει.

Για παράδειγμα μπορούμε να δημιουργήσουμε έναν όρο έτσι:

Term t=new Term(“MoviesTitle”,”Terminator”);

΄Υστερα μια αίτηση – ερώτηση :

Query query1=new TermQuery(t);

Και το αντικείμενο TopDocs :

TopDocs docs= searcher.search(query1,10);

Λεπτομερώς, στην πρώτη σειρά δημιουργείται ένας όρος με όνομα “t” , το “MoviesTitle” προσδιορίζει το όνομα πεδίου που θέλουμε να αναζητήσουμε την λέξη “Terminator”. Η αίτηση- ερώτηση που δημιουργούμε στην δεύτερη γραμμή είναι τύπου TermQuery (υπάρχουν πολλές επιλογές στη θέση του TermQuery) που παίρνει σαν παράμετρο τον όρο. Δημιουργείται έτσι η αίτηση- ερώτηση που

```
var hits = searcher.Search
    (booleanQuery, null, hits_limit, Sort.RELEVANCE);
var results = _mapLuceneToDataList(hits.ScoreDocs, searcher);

analyzer.Close();
searcher.Close();
searcher.Dispose();
return results;
```

Σχήμα 13: Το IndexSearcher (searcher) χρησιμοποιεί την μέθοδο Search για την υποβολή της αίτησης-ερώτησης στο ευρετήριο ώστε να βρεθούν αποτελέσματα.

έχει την εξής μορφή “query1 = { MoviesTitle : Terminator }”. Η τρίτη σειρά σημαίνει ότι δημιουργούμε το “docs” αντικείμενο δηλ. documents (τα αποτελέσματα) που είναι τύπου TopDocs και αποδίδουμε το δεύτερο μέρος σαυτό. Το δεύτερο μέρος σημαίνει ότι η μέθοδος search του searcher πρέπει να βρει την αίτηση- ερώτηση με μέγιστο αριθμό πιο πετυχημένων αποτελεσμάτων 10. Το searcher είναι τύπου IndexSearcher και παίρνει για παράμετρο το Directory που αναφέραμε πιο πριν. Επίσης δεν αναφέρουμε το ScoreDocs που αναπαριστά τις επιτυχίες των αποτελεσμάτων . Αφού τελειώσουμε κλείνουμε το searcher και μετά το directory. Γενικά είναι καλύτερα αν αφήνουμε ανοιχτά αυτά τα δύο για όλες τις αιτήσεις – ερωτήσεις και να μοιραζόμαστε ένα και μόνο searcher για όλες τις αιτήσεις-ερωτήσεις που θα γίνονται μελλοντικά. Το άνοιγμα ενός καινούργιου searcher μπορεί να αποδειχτεί δαπανηρή λειτουργία επειδή θα πρέπει να φορτώνονται και να γεμίζονται εσωτερικές δομές δεδομένων απ’το ευρετήριο.

4.3.3 Παράδειγμα του IndexSearcher για αναζήτηση στο JobFinder

Στο παρακάτω σχήμα κώδικα φαίνεται ακριβώς πώς γίνεται η αναζήτηση αφού έχουμε ήδη ετοιμάσει την αίτηση-ερώτηση προς το ευρετήριό μας:

Δίνοντας λεπτομέρεια στο κώδικα χρειάζεται μια καλύτερη επεξήγηση όλων των γραμμών αυτών. Η πρώτη γραμμή δηλώνει την δημιουργία του **hits** το οποίο είναι ένα στιγμιότυπο του **TopFieldDocs** (αναφέρθηκε παραπάνω στον πίνακα)

και δέχεται τα αποτελέσματα της αναζήτησης που κάνει ο **searcher** (τύπου **IndexSearcher**) μέσω της μεθόδου **Search** που περιέχει. Η μέθοδος αυτή δέχεται στην περίπτωση αυτή 4 παραμέτρους :

- Η πρώτη παράμετρος δέχεται την πολυσύνθετη αίτηση-ερώτηση που έχουμε κατασκευάσει μέχρι τώρα (τύπου BooleanQuery)
- Η δεύτερη παίρνει κανονικά ένα φίλτρο αλλά εμείς εδώ την αφήνουμε κενή
- Η τρίτη παράμετρος δηλώνει το μέγιστο αριθμό αποτελεσμάτων που θέλουμε να βρούμε (είναι 1000 στην περίπτωσή μας γιατί όταν ξεπερνιέται αυτό ο αριθμός δημιουργείται πολύ μεγάλη επιβράδυνση στην λειτουργία)
- Η τέταρτη παράμετρος είναι η ταξινόμηση για τα αποτελέσματα ώστε να γίνει με βάση την σχετικότητα.

Στην επόμενη γραμμή αρχικοποιούμε ένα αντικείμενο results το οποίο είναι μία λίστα τύπου **IEnumerable<SearchResult>**. Το **SearchResult** είναι δική μας κλάση η οποία έχει ιδιότητες κατάλληλες να φιλοξενήσουν τις τιμές των πεδίων των εγγράφων με παρόμοιο τρόπο όπως ήταν αποθηκευμένες στο ευρετήριο. Αυτό γίνεται για να μπορέσουμε να γυρίσουμε τα αποτελέσματα με ένα εύχρηστο τρόπο προς τους χρήστες. Έτσι λοιπόν η λίστα results παίρνει τα αποτελέσματα από την μέθοδο **_mapLuceneToDataList** η οποία παίρνει σαν παραμέτρους τα **ScoreDocs** των **hits** που αναφέραμε πριν και τον **searcher** (τύπου IndexSearcher) . Η μέθοδος **_mapLuceneToDataList** φαίνεται στο επόμενο σχήμα για καλύτερη κατανόηση.

Βλέποντας το κώδικα στο σχήμα καταλαβαίνουμε ότι η ίδια η μέθοδος **_mapLuceneToDataList** λειτουργεί απλά σαν μια έμμεση μέθοδος στην

```
private static IEnumerable<SearchResult> _mapLuceneToDataList
    (IEnumerable<ScoreDoc> hits, IndexSearcher searcher)
{
    return hits.Select(hit => _mapLuceneDocumentToData(searcher.Doc(hit.Doc))).ToList();
}

//_mapLuceneDocumentToData() maps Lucene Document with search results from index to our
// class UserJobs
private static SearchResult _mapLuceneDocumentToData(Document doc)
{
    var result = new SearchResult()
        {
            Id = Convert.ToInt32(doc.Get("Id")),
            AdvertiserId = Convert.ToInt32(doc.Get("AdvertiserId")),
            Job = Regex.Replace(doc.Get("JobTitle"), @"σ\b", "ς"),
            Description = Regex.Replace(doc.Get("Description"), @"σ\b", "ς"),
            Area = doc.Get("Area"),
            Municipality = Regex.Replace(doc.Get("Municipality"), @"σ\b", "ς"),
            // ^\\b(\\w+(σ\\b)
            City = doc.Get("City"),
            Type = doc.Get("type")
        };
    if (doc.Get("type") == "UserJob")
    {
        result.YearsOfExperience = doc.Get("YearsOfExperience");
        result.DrivingLicenses = doc.Get("AdvertiserDrivingLicenses").ToUpper();
    }
    return result;
}
```


Σχήμα 14: Οι μέθοδοι _mapLuceneToDataList και _mapLuceneDocumentToData

μετατροπή δεδομένων των εγγράφων σε κατάλληλη μορφής αποτελέσματα (που λέγαμε πριν) τα οποία τα προσθέτει όλα σε μια λίστα. Λέγοντας έμμεση μέθοδος εννοούμε ότι λειτουργεί σαν βρόχος επανάληψης κλήσης της μεθόδου **_mapLuceneDocumentToData** για κάθε αποτέλεσμα στο σύνολο αποτελεσμάτων. Η τελευταία με τη σειρά της δέχεται σαν παράμετρο κάθε έγγραφο (δηλ. κάθε αποτέλεσμα)και από τις τιμές των πεδίων του εγγράφου αυτού δημιουργεί το αντικείμενο SearchResult που αναφέραμε πριν.

4.3.4 Η ποικιλία των αιτημάτων-ερωτημάτων του Lucene

Όπως είδαμε και πριν κάνοντας αιτήσεις-ερωτήσεις στο Lucene απαιτεί μια κλήση σε μία από τις μεθόδους του **search** του **IndexSearcher**, χρησιμοποιώντας ένα στιγμιότυπο του **Query**. Οι υποκλάσεις του **Query** μπορούν να δημιουργηθούν άμεσα ή ένα αντικείμενο τύπου Query μπορεί να κατασκευαστεί μέσα από την χρήση του **QueryParser**, μια πρόσοψη που μπορεί να μετατρέψει ελεύθερο κείμενο σε έναν από τους Query τύπους που θα αναλυθούν στη συνέχεια.

Είναι φανερό, χρησιμοποιώντας τον QueryParser, ο συνδυασμός μιας έκφρασης αίτησης-ερώτησης προς ανάλυση με ένα παραγόμενο Query διασύνδεσης προγραμματισμού είναι μια συνηθισμένη τεχνική για να αυξήσει, βελτιώσει ή περιορίσει κανείς μια ανθρώπινη αίτηση-ερώτηση. Για παράδειγμα μπορεί να θέλουμε να περιορίσουμε τις ελεύθερης-μορφής εκφράσεις προς ανάλυση σε ένα υποσύνολο του ευρετηρίου, όπως τα έγγραφα μιας συγκεκριμένης κατηγορίας. Με βάση τη διεπαφή χρήστη αναζήτησης, μπορεί να έχουμε ένα συλλέκτη ημερομηνιών για να μπορούμε να διαλέξουμε ένα εύρος ημερομηνιών, αναπτυσσόμενα μενού για την επιλογή κατηγορίας και ένα κουτί αναζήτησης ελεύθερης φόρμας. Η κάθε περίπτωση από αυτές μπορεί να συνυπάρξει χρησιμοποιώντας ένα συνδυασμό του QueryParser και προγραμματισμένων αιτημάτων-ερωτημάτων.

Παρουσιάζονται στη συνέχεια όλοι οι δημοφιλείς ενσωματωμένοι τύποι Query του Lucene

4.3.4.1 Αναζητώντας ανά όρο: TermQuery

Ο πιο απλός τρόπος να αναζητήσουμε έναν όρο σε ένα ευρετήριο. Ο όρος είναι το μικρότερο κομμάτι ευρετηρίου, που αποτελείται από ένα όνομα πεδίου και μια τιμή-κειμένου.

Παράδειγμα:

Term t=new Term(“contents”,”Lucene”);
Query query=new TermQuery(t);

Έτσι όλα τα έγγραφα που περιέχουν τη λέξη Lucene στο πεδίο contents θα επιστραφούν από τις αναζητήσεις. Προσοχή στους όρους ευρετηρίου γιατί η υπάρχει διάκριση πεζών-κεφαλαίων και ο αναλυτής μπορεί να έχει προσθέσει στο

ευρετήριο διαφορετικά κάποια στοιχεία. Χαρακτηριστικά, η περίπτωση του TermQuery είναι πολύ συνηθισμένη στην ανάκτηση εγγράφων από ένα στοιχείο-κλειδί.

4.3.4.2 Αναζητώντας μέσα σε ένα εύρος όρων : TermRangeQuery

Οι όροι ταξινομούνται ανά Λεξικογραφία (σύμφωνα με το String.compareTo) μέσα στο ευρετήριο, επιτρέποντας την άμεση αναζήτηση όρων κειμένου μέσα σε ένα εύρος όπως παρέχει το TermRangeQuery. Οι όροι αρχής και τέλους μπορεί είτε να περιλαμβάνονται είτε να αποκλείονται. Αν ο όρος είναι null το τέλος είναι αόριστο. Χρησιμοποιούμε την περίπτωση του TermRangeQuery π.χ. για να βρούμε όλα τα ονόματα που αρχίζουν από Τ και τελειώνουν σε Ω.

Το επόμενο παράδειγμα μας δείχνει πώς να αναζητήσουμε όλα τα βιβλία που ο τίτλος τους αρχίζει από το γράμμα “d” μέχρι “j” :

TermRangeQuery query = new TermRangeQuery(“title2”, “d”,”j”, true , true);
TopDocs matcher = searcher.search(query,100);

Οι τελευταίες δύο δυαδικές τιμές δηλώνουν αν τα σημεία αρχής και τέλους περιλαμβάνονται (true) ή εξαιρούνται (false). Δηλώσαμε true για να συμπεριληφθούν στην αναζήτηση, αλλά αν είχαμε δηλώσει false τότε δεν θα υπήρχαν αλλαγές στα αποτελέσματα επειδή δεν θα είχαμε βρει βιβλία που έχουν τον ακριβή τίτλο “d” ή “j”. Δυστυχώς χρησιμοποιώντας αυτό τον τρόπο αναζήτησης μπορεί η αναζήτηση να αποδειχτεί πάρα πολύ αργή ειδικά σε ένα πολύ μεγάλο ευρετήριο.

4.3.4.3 Αναζητώντας σε ένα εύρος αριθμών : NumericRangeQuery

Αν προσθέσαμε το πεδίο στο ευρετήριο χρησιμοποιώντας το NumericField μπορούμε αποτελεσματικά να αναζητήσουμε ένα συγκεκριμένο εύρος αριθμών. Το Lucene μεταφράζει το ζητούμενο εύρος σε ένα ισοδύναμο σύνολο από παρενθέσεις στην δομή του ευρετηρίου. Κάθε παρένθεση είναι ένας διακριτός όρος στο ευρετήριο τα έγγραφα του οποίου συνδέονται μαζί με OR. Ο αριθμός των παρενθέσεων που χρειάζεται είναι σχετικά μικρός, σημείο στο οποίο διαφέρει με την προηγούμενη περίπτωση (TermRangeQuery) δίνοντας έτσι καλύτερη απόδοση.

Ας δούμε ένα παράδειγμα βασισμένο στην ημερομηνία έκδοσης ενός βιβλίου που υπάρχει στο ευρετήριο (pubmonth). Προσθέσαμε στο ευρετήριο το πεδίο σαν ακέραιο με ακρίβεια μηνός, π.χ. Μάρτιος 2010 το οποίο προστέθηκε στο ευρετήριο σαν NumericField, με ακέραια τιμή 201003 (2010 03) :

NumericRangeQuery query = NumericRangeQuery.newIntRange(“pubmonth”,200605,200609,true,true);
TopDocs matcher = searcher.search(query,10);

Όπως και προηγουμένως στο TermRangeQuery ισχύει το ίδιο για τις δύο τελευταίες δυαδικές τιμές. Στα αποτελέσματα θα βρεθεί ένα βιβλίο που είχε εκδοθεί το Σεπτέμβριο του 2006. Εάν όμως στις δύο τελευταίες τιμές είχαμε false τότε δεν θα το έβρισκε.

4.3.4.4 Αναζητώντας σε μια συμβολοσειρά: PrefixQuery

Το PrefixQuery βρίσκει έγγραφα που περιέχουν όρους που ξεκινάνε με μια συγκεκριμένη συμβολοσειρά και αυτό μπορεί να αποδειχτεί πολύ χρήσιμο. Ο επόμενος κώδικας παρουσιάζει πως μπορούμε να κάνουμε αίτηση-ερώτηση σε μια ιεραρχική δομή αναδρομικά με απλό PrefixQuery. Τα έγγραφα περιέχουν το πεδίο category αναπαριστώντας μια ιεραρχική δομή που είναι κατάλληλη για την εύρεση εγγράφων με το PrefixQuery.

Term term = new Term("category", "/technology/computers/programming");
PrefixQuery query = new PrefixQuery (term);

Σ'αυτό το παράδειγμα αν υποθέσουμε ότι υπάρχει ένα category με όνομα "methodology" κάτω από τη διεύθυνση /technology/computers/programming τότε τα βιβλία που υπάρχουν στο category αυτό θα βρεθούν ενώ αν χρησιμοποιήσουμε το TermQuery όχι.

4.3.4.5 Συνδυάζοντας αιτήσεις-ερωτήσεις: BooleanQuery

Οι τύποι αιτήσεων-ερωτήσεων εδώ μπορούν να συνδυαστούν με σύνθετους τρόπους χρησιμοποιώντας το BooleanQuery, το οποίο είναι ένα κοντέινερ δυαδικών ρητρών. Μία ρήτρα είναι μία υποαίτηση-ερώτηση που μπορεί να είναι απαραίτητη ή προαιρετική. Αυτά τα χαρακτηριστικά επιτρέπουν για λογικούς συνδυασμούς AND, OR και NOT. Προσθέτουμε μία ρήτρα στο BooleanQuery χρησιμοποιώντας την ακόλουθη μέθοδο API :

public void add (Query query, BooleanClause.Occur occur)

όπου occur μπορεί να είναι BooleanClause.Occur.MUST, BooleanClause.Occur.SHOULD ή BooleanClause.Occur.MUST_NOT

Ένα BooleanQuery μπορεί να είναι μία ρήτρα ενός άλλου BooleanQuery επιτρέποντας την αυθαίρετη ενσωμάτωση. Το παράδειγμα που ακολουθεί δείχνει μία αίτηση-ερώτηση AND για να βρει τα πιο πρόσφατα βιβλία που έχουν θέμα "search".

1) TermQuery searchingBooks = new TermQuery (new Term ("subject","search"));
2) Query books2010 = NumericRangeQuery.newIntRange ("pubmonth",201001,201012, true,true);
3) BooleanQuery searchingBooks2010 = new BooleanQuery();
searchingBooks2010.add(searchingBooks, BooleanClause.Occur.MUST);

searchingBooks2010.add(books2010, BooleanClause.Occur.MUST);

Όπως βλέπουμε το 1) δημιουργεί την αίτηση – ερώτηση για να βρει όλα τα βιβλία που περιέχουν σαν “subject” τη λέξη “search”. Το 2) δημιουργεί την αίτηση-ερώτηση για να βρει όλα τα βιβλία που εκδόθηκαν από το πρώτο μήνα του 2010 (201001) μέχρι το τελευταίο μήνα (201012) και το 3) συνδυάζει σε μία αίτηση – ερώτηση (searchingBooks2010) τα δύο παραπάνω αιτήματα σε ένα τύπου Boolean και με τις δύο ρήτρες όμως να είναι απαραίτητες (BooleanClause.Occur.MUST).

Δεν υπάρχει κανένα πρόβλημα να ταιριάξει κανείς διαφορετικές ρήτρες μέσα σε ένα BooleanQuery, αρκεί να υπάρχει σε κάθε μία το BooleanClause.Occur. Έτσι μπορούμε να δημιουργήσουμε πολύ δυναμικές αιτήσεις-ερωτήσεις. Ο μέγιστος αριθμός των BooleanQuerys που μπορούμε να δημιουργήσουμε είναι εξ ορισμού 1024. Αυτός ο περιορισμός ισχύει για να μην ξεπεραστούν κάποια όρια και γίνει ζημιά στην απόδοση. Πάντως εάν θελήσει κανείς μπορεί να αυξήσει τον αριθμό αυτό από τη μέθοδο setMaxClauseCount (int) αλλά με προσοχή.

Παρακάτω υπάρχει ένα μέρος από το δικό μας κώδικα όσον αφορά το BooleanQuery :

```
using (var searcher = new IndexSearcher(_directory, true))
{
    var hits_limit = 1000; //after 1000 hits it becomes very very slow
    var booleanQuery = new BooleanQuery();
    var analyzer = new StandardAnalyzer(Version.LUCENE_29);
    Query query, query1;
    //   var reader = IndexReader.Open(_directory, false);

    if (!(string.IsNullOrEmpty(model.JobTitle))) //an to jobtitle den einai keno i null
    {
        var parser = new MultiFieldQueryParser(
            Version.LUCENE_29, new[] { "JobTitle", "Description" },analyzer);
        query = parseQuery(RemoveDiacritics(model.JobTitle.ToLower()), parser);
        booleanQuery.Add(query, BooleanClause.Occur.MUST);
    }
}
```

Σχήμα 15: Παράδειγμα της εφαρμογής μας που χρησιμοποιεί BooleanQuery

Όπως βλέπουμε στο παράδειγμα ύστερα από τις πρώτες γραμμές που καθορίζουν κάποια στοιχειώδη πράγματα σαν την δήλωση του Analyzer και την αρχικοποίησή του, γίνεται ένας έλεγχος εάν ο τίτλος της δουλειάς (model.jobTitle) είναι κενός η null. Αφού διαπιστώσουμε ότι δεν είναι, χρησιμοποιούμε έναν **MultiFieldQueryParser** ο οποίος συνδυάζει τα πεδία των εγγράφων στα οποία θα γίνουν αναζητήσεις : ***JobTitle*** και ***Description.*** Ύστερα, δημιουργείται η αίτηση-ερώτηση από την ανάλυση του τίτλου σύμφωνα με τα προαναφερθέντα πεδία. Η τελευταία γραμμή δηλώνει ότι το booleanQuery προσθέτει την αίτηση-ερώτηση στον εαυτό του και το BooleanClause.Occur.MUST σημαίνει ότι οπωσδήποτε πρέπει να βρεθεί η συγκεκριμένη αίτηση-ερώτηση.

4.3.4.6 Αναζητώντας με φράση : PhraseQuery

Ένα ευρετήριο διαθέτει εξ ορισμού πληροφορίες θέσεων των όρων, εφόσον δεν έχουν δημιουργηθεί καθαρά δυαδικά πεδία (Boolean) με την επιλογή omitTermFreqAndPositions. Το PhraseQuery χρησιμοποιεί την πληροφορία αυτή για να εντοπίσει έγγραφα όπου οι όροι βρίσκονται σε μια συγκεκριμένη απόσταση ο ένας από τον άλλον. Για παράδειγμα, σκεφτείτε ένα πεδίο που περιέχει την φράση “the quick brown fox jumped over the lazy dog”. Χωρίς να γνωρίζουμε όλη τη πρόταση, μπορούμε να βρούμε το έγγραφο αυτό αναζητώντας στα πεδία του που να έχουν τους όρους quick και fox το ένα κοντά στο άλλο. Ένα TermQuery μπορεί να κάνει τη δουλειά αυτή αναζητώντας κάποια από αυτές τις λέξεις, αλλά στην περίπτωση αυτή θέλουμε μόνο έγγραφα που έχουν φράσεις όπου οι λέξεις βρίσκονται η μία δίπλα στην άλλη (quick fox) ή να έχουν μία λέξη ανάμεσα (quick [κάτι] fox).

Η μέγιστη επιτρεπόμενη διαφορά απόστασης μεταξύ όρων ονομάζεται slop. Η απόσταση είναι ο αριθμός των κινήσεων θέσης των όρων που μπορούν να ανακατασκευάσουν μία φράση σε σειρά.

Φράσεις πολλαπλών όρων – Multi-term phrases

Το PhraseQuery υποστηρίζει τις φράσεις πολλαπλών όρων. Ανεξαρτήτου αριθμού των όρων που χρησιμοποιούνται για μια φράση, ο slop παράγοντας είναι ο μέγιστος αριθμός συνολικών κινήσεων που επιτρέπουν τους όρους να μπουν σε μια σειρά.

4.3.4.7 Αναζητώντας με χαρακτήρες μπαλαντέρ : WildcardQuery

Οι αιτήσεις-ερωτήσεις με χαρακτήρες μπαλαντέρ επιτρέπουν την αναζήτηση όρων που έχουν κάποια μέρη που λείπουν αλλά παρόλα αυτά μπορούν να βρεθούν. Οι δύο πιο συνηθισμένοι χαρακτήρες που χρησιμοποιούνται είναι **“*”** για μηδέν ή περισσότερους χαρακτήρες και **“?”** για ένα ή κανένα χαρακτήρα. Για παράδειγμα μπορούμε να βρούμε με όρο “?ild*” τους όρους “wild,mild,mildew” αλλά όχι και τον όρο “child”.

4.3.4.8 Αναζητώντας για παρόμοιους όρους: FuzzyQuery

Το FuzzyQuery του Lucene βρίσκει όρους παρόμοιους με έναν συγκεκριμένο όρο. Ο αλγόριθμος “απόστασης Levenshtein” ορίζει πόσο παρόμοιοι είναι οι όροι σε ένα ευρετήριο σύμφωνα με τον όρο που δόθηκε. Η “επεξεργασία απόστασης” είναι ένας άλλος όρος για την απόσταση Levenshtein. Είναι ένα μέγεθος ομοιότητας μεταξύ δύο συμβολοσειρών, όπου η απόσταση μετριέται σαν αριθμός διαγραφής, εισαγωγής και αντικατάστασης χαρακτήρων που χρειάζονται για να μετατρέψουν μια συμβολοσειρά σε μια άλλη. Για παράδειγμα η επεξεργασία απόστασης μεταξύ του “three” και “tree” είναι 1, επειδή μόνο ένας χαρακτήρας διαγραφής χρειάζεται.

4.3.4.9 Βρίσκοντας όλα τα έγγραφα: MatchAllDocsQuery

Το MatchAllDocsQuery όπως δηλώνει και το όνομα απλά βρίσκει κάθε έγγραφο στο ευρετήριο. Εξ ορισμού, αναθέτει ένα σταθερό σκόρ, την τόνωση δηλ. για κάθε αίτηση-ερώτηση (1.0 προκαθορισμένο) σε όλα τα έγγραφα που ταιριάζουν στην εύρεση. Εάν χρησιμοποιεί κανείς το MatchAllDocsQuery σαν την κύρια αίτηση-ερώτηση, είναι καλύτερα να ταξινομεί ανά πεδίο διαφορετικό του εξ ορισμού πεδίου σχετικότητας.

4.4 Αποτύπωση Αποτελεσμάτων

Αφού αποκτάμε, όπως είπαμε, το σύνολο εγγράφων που ταιριάζουν στην αίτηση-ερώτηση ταξινομημένα σε σωστή σειρά, πρέπει να τα αποτυπώνουμε με ένα αισθητικό και αναλώσιμο τρόπο προς τους χρήστες. Η διεπαφή χρήστη πρέπει επίσης να προσφέρει ένα καθαρό μονοπάτι για τις επόμενες αναζητήσεις ή ενέργειες, όπως να κάνει κλίκ στην επόμενη σελίδα, ή να βελτιώσει την αναζήτηση, ή να βρει έγγραφα παρόμοια με ένα αποτέλεσμα, ώστε ο χρήστης να μην φτάνει ποτέ σε ένα τέλος.

4.4.1 Σελιδοποίηση στα αποτελέσματα

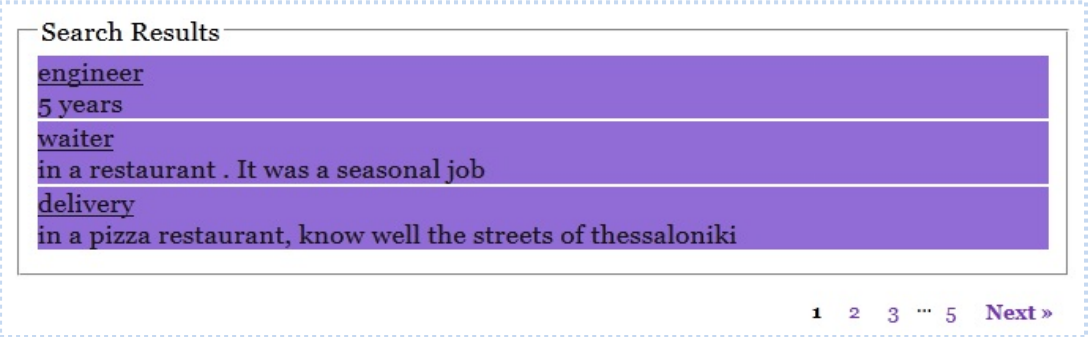
Η παρουσίαση αποτελεσμάτων αναζήτησης στους τελικούς χρήστες πολύ συχνά περιλαμβάνει μόνο τα πρώτα 10 ή 20 πιο σχετικά έγγραφα. Η σελιδοποίηση ανάμεσα στα ScoreDocs είναι μια συχνή απαίτηση, και αν διαπιστωθεί η συχνή χρήση της σελιδοποίησης από τους χρήστες πρέπει να γίνει και επανασχεδιασμός κατά ένα τρόπο στη σχεδίαση. Αν και ο χρήστης συνήθως βρίσκει τα αποτελέσματα που θέλει στην πρώτη σελίδα πάντα είναι καλό να υπάρχει η σελιδοποίηση για τυπικούς λόγους. Υπάρχουν δύο τρόποι για να γίνει αυτό:

- Κάνοντας συλλογή πολλαπλών σελίδων που ανταποκρίνονται τα αποτελέσματά τους με τις αρχικές επιλογές αναζήτησης και να κρατήσουμε τα στιγμιότυπα αποτελεσμάτων ScoreDocs και IndexSearcher διαθέσιμα καθώς ο χρήστης πλοηγείται στα αποτελέσματα αναζήτησης.
- Κάνοντας κάθε φορά την αίτηση-ερώτηση όταν ο χρήστης θέλει να πλοηγηθεί σε άλλη σελίδα.

Η δεύτερη επιλογή αποτελεί συχνά και την καλύτερη λύση. Αυτό συμβαίνει γιατί δεν χρειάζεται η αποθήκευση της κατάστασης ανά χρήστη, πράγμα το οποίο σε μια web εφαρμογή μπορεί να αποδειχτεί πολύ δαπανηρή διαδικασία, ειδικά με ένα μεγάλο αριθμό χρηστών. Μπορεί η δεύτερη επιλογή να φαίνεται σαν σπατάλη, αλλά η απίστευτη ταχύτητα του Lucene την αποζημιώνει. Επίσης, χάρη στην προσωρινή αποθήκευση εισόδων/εξόδων στα σύγχρονα λειτουργικά συστήματα η διαδικασία αυτή πρέπει να είναι τυπικά γρήγορη επειδή τα αναγκαία bit από το δίσκο ήδη θα πρέπει να βρίσκονται προσωρινά αποθηκευμένα στη RAM.

Συνήθως οι χρήστες δεν κάνουν κλίκ πέρα από την πρώτη σελίδα αποτελεσμάτων έτσι και αλλιώς.

Η διαδικασία της επανάληψης αίτησης – ερώτησης περιλαμβάνει την επανάληψη της αρχικής αναζήτησης με μεγαλύτερο αριθμό ζητούμενων



Σχήμα 16: Αποτύπωση αποτελεσμάτων στην διεπαφή DetailedSearch

αντιστοιχίσεων και τα αποτελέσματα εμφανίζονται στην αρχή της επιθυμητής σελίδας. Το πώς η αρχική αίτηση-ερώτηση μπορεί να κρατηθεί χωρίς να χαθεί εξαρτάται από την αρχιτεκτονική της εφαρμογής μας. Σε μια εφαρμογή διαδικτύου όπου ο χρήστης πληκτρολογεί μια έκφραση η οποία αναλύεται με το QueryParser, η αρχική έκφραση θα μπορούσε να γίνει μέρος των συνδέσμων για την πλοήγηση στις σελίδες και να ξανά-αναλύεται για κάθε αίτημα ή αλλιώς η έκφραση να κρατιέται ζωντανή σε ένα κρυφό πεδίο HTML ή σαν cookie.

4.4.2 Αποτύπωση αποτελεσμάτων και σελιδοποίηση στο JobFinder

Στο παρακάτω σχήμα μπορούμε να δούμε πώς προβάλλουμε τα αποτελέσματά μας και την σελιδοποίηση που υπάρχει κάτω δεξιά :

Αναλύοντας παρεταίρω το σχήμα πάνω βλέπουμε ότι η προβολή των αποτελεσμάτων γίνεται μέσα σε ένα πλαίσιο στο οποίο χωράνε 3 αγγελίες (καθορίσαμε 3 για το συγκεκριμένο παράδειγμα). Πατώντας σε κάθε αγγελία μεταφερόμαστε στην ανάλογη σελίδα για να δούμε περισσότερες λεπτομέρειες. Κάτω δεξιά φαίνεται ότι είμαστε στη πρώτη σελίδα. Βλέπουμε τις επόμενες δύο σελίδες που μπορούμε να πάμε άμεσα καθώς και την τελευταία. Επίσης υπάρχει και κουμπάκι Next που μπορεί να μας πηγαίνει ακριβώς στην επόμενη σελίδα από την οποία βρισκόμαστε. Αυτό είναι ένα χαρακτηριστικό της σελιδοποίησης που δουλεύει με την διαδικασία αίτησης-ερώτησης που αναφέραμε πριν.

Κεφάλαιο 5 : Τα υπόλοιπα συστατικά της εφαρμογής αναζήτησης

Χρειάζεται ακόμη λίγα να προστεθούν για το τι χρειάζεται μία πλήρως λειτουργική μηχανή αναζήτησης, ειδικά όταν τρέχει σε ένα δικτυακό τόπο. Πρώτον, χρειάζεται διαχείριση, ώστε να παρακολουθείται η κατάσταση της εφαρμογής, η ρύθμιση διάφορων συστατικών και το ξεκίνημα και η παύση των διακομιστών. Επίσης χρειάζονται να συμπεριληφθούν οι αναλύσεις, για να επιτρέπουν την

διαφορετική προβολή ως προς τον τρόπο που οι χρήστες χρησιμοποιούν την αναζήτηση, δίνοντας μαυτό τον τρόπο συμπεράσματα για το τι δουλεύει και τι δεν δουλεύει. Τέλος, για μεγάλες εφαρμογές αναζήτησης, η κλιμάκωση είναι ένα πολύ σημαντικό στοιχείο ώστε η εφαρμογή να μπορεί να διαχειρίζεται όλο και πιο μεγάλα σε μέγεθος περιεχόμενα και όλο και περισσότερες ταυτόχρονες αναζητήσεις.

Την διεπαφή διαχείρισης μπορούμε να δούμε αριστερά στο Σχήμα 1.

5.1 Διεπαφή διαχείρισης

Μια μοντέρνα μηχανή αναζήτησης είναι ένα περίπλοκο κομμάτι λογισμικού και έχει πολλές λειτουργίες που χρειάζονται κατάλληλες ρυθμίσεις. Αν χρησιμοποιείτε μια αράχνη για την ανακάλυψη του περιεχομένου, η διεπαφή διαχείρισης πρέπει να επιτρέπει να καθορίζετε τις αρχικές διευθύνσεις υπερσυνδέσμων, να δημιουργείτε κανόνες για να κάνετε σαφές ποιες διευθύνσεις η αράχνη πρέπει να επισκέπτεται η ποιούς τύπους εγγράφων πρέπει να φορτώνει κ.α. Επίσης η έναρξη και η παύση λειτουργίας των διακομιστών, η καταστροφή των αρχείων καταγραφής, ο έλεγχος της υγείας του συστήματος, η δημιουργία και η ανάκτηση από αντίγραφα ασφαλείας είναι όλα παραδείγματα για το τι μπορεί να προσφέρει η διεπαφή διαχείρισης.

Το Lucene έχει ένα πλήθος από επιλογές ρυθμίσεων που θα μπορούσε να αποκαλύπτει μια διεπαφή διαχείρισης. Κατά τη διάρκεια δημιουργίας του ευρετηρίου μπορεί να χρειάζεται ο συντονισμός του μεγέθου του buffer της RAM, ή του αριθμού των τμημάτων που πρέπει να συγχωνευτούν, κάθε πότε να διαπράττονται οι αλλαγές ή πότε να γίνονται βελτιστοποιήσεις και οι διαγραφές από το ευρετήριο.

Η αναζήτηση επίσης έχει σημαντικές επιλογές διαχείρισης, όπως πόσο συχνά πρέπει να ανοίγεται ο «αναγνώστης». Μπορεί επίσης να θέλει κανείς να παρακολουθεί κάποιες πληροφορίες όπως π.χ. την αποτυχία κάποιων εγγράφων να προστεθούν στο ευρετήριο, ή τις ερωτήσεις που οδήγησαν σε εξαιρέσεις στο πρόγραμμα, πράγματα για τα οποία η διεπαφή προγραμματισμού της διαχείρισης θα μπορούσε να δώσει πληροφορίες.

Πολλές εφαρμογές αναζήτησης , όπως οι επιτρεπέζιου τύπου , δεν χρειάζονται τέτοιο συστατικό ενώ μια πλήρης επιχειρησιακή εφαρμογή μπορεί να έχει μία περίπλοκη διεπαφή διαχείρισης. Συχνά ή διεπαφή είναι βασισμένη στον Ιστό, αλλά μπορεί να αποτελείται και από κάποια επιπρόσθετα εργαλεία γραμμής εντολών. Δεξιά στο Σχήμα 1 είναι η διεπαφή ανάλυσης.

5.2 Διεπαφή ανάλυσης

Η διεπαφή ανάλυσης, είναι συνήθως μία διεπαφή χρήστη βασισμένη στον Ιστό και πολλές φορές τρέχει κάτω από ένα ξεχωριστό διακομιστή φιλοξενώντας μια

μηχανή αναφοράς. Οι αναφορές που κάνει είναι σημαντικές: μπορεί να αποκτήσει κανείς αρκετή νοημοσύνη για τους χρήστες όπως γιατί αγοράζουν ή δεν αγοράζουν τα πράγματα από το δικό σας ιστότοπο, κοιτώντας απλά για πατέντες στα αρχεία καταγραφής των αναζητήσεων. Αν δουλεύει κανείς ένα ιστότοπο ηλεκτρονικού εμπορίου, υπάρχουν πολύ δυνατά εργαλεία που μπορούν να συνεισφέρουν ώστε να γίνουν βελτιώσεις στην αίσθηση εμπειρίας αγοράς που κάνουν οι χρήστες. Αυτά τα εργαλεία μπορούν να βοηθήσουν στη διαφάνεια το πώς οι χρήστες αναζητούν, να δείξουν ποιες αναζητήσεις απέτυχαν να δημιουργήσουν επιτυχή αποτελέσματα, σε ποια αποτελέσματα οι χρήστες έκαναν κλίκ και πόσο συχνά μια αγορά ακολουθήθηκε ή όχι από μια αναζήτηση.

Οι μετρικές του Lucene που μπορούν να τρέφουν την διεπαφή αναλυτικών περιλαμβάνουν:

- Πόσο συχνά και τι διαφορετικές ερωτήσεις υπάρχουν (ενός όρου, μιας φράσης, δυαδικές ερωτήσεις κτλ.)
- Ερωτήσεις που έχουν μικρή σχετικότητα
- Ερωτήσεις που ο χρήστης δεν πάτησε πουθενά κλίκ στα αποτελέσματα
- Κάθε πότε οι χρήστες ταξινομούν ανά συγκεκριμένο πεδίο αντί ανά σχετικότητα.
- Ανάλυση του χρόνου αναζήτησης του Lucene

Επίσης μπορεί να θέλει κανείς να δει και τις μετρικές δημιουργίας ευρετηρίου, όπως τα έγγραφα που προστίθενται στο ευρετήριο το δευτερόλεπτο ή το μέγεθος σε byte των εγγράφων αυτών.

Το Lucene, παρόλα αυτά δεν προσφέρει αναλυτικά εργαλεία. Αν η εφαρμογή αναζήτησης βρίσκεται στον Ιστό, το Google Analytics είναι ένας γρήγορος τρόπος για την δημιουργία μιας διεπαφής ανάλυσης. Το τελευταίο θέμα που αναφέρεται είναι η κλιμάκωση.

5.3 Κλιμάκωση

Μία ιδιαίτερα περίπλοκη περιοχή της εφαρμογής αναζήτησης είναι η κλιμάκωση. Η πλειοψηφία των εφαρμογών αναζήτησης δεν έχουν αρκετό περιεχόμενο ή τόσο ταυτόχρονη κίνηση για να απαιτούν κλιμάκωση πέρα του ενός υπολογιστή. Η δημιουργία καταλόγου και η ικανότητα διαβίβασης δεδομένων της αναζήτησης με το Lucene επιτρέπουν ένα αρκετά μεγάλο περιεχόμενο σε έναν υπολογιστή. Παρόλα αυτά υπάρχουν εφαρμογές που μπορεί να χρειάζεται να λειτουργούν σε δύο παράλληλα παρόμοιους υπολογιστές για να διαβεβαιώνουν ότι δεν θα υπάρχει ένα σημείο κάμψης σε περίπτωση αποτυχίας υλικού. Αυτή η προσέγγιση επιτρέπει να μπορεί κανείς να απομονώσει κάποιο υπολογιστή από τη παραγωγή για να κάνει συντήρηση και αναβάθμιση χωρίς να επηρεάσει τις αναζητήσεις που πραγματοποιούνται.

Υπάρχουν δύο διαστάσεις για να κλιμακωθούν: Καθαρή ποσότητα του περιεχομένου, και καθαρή ικανότητα διαβίβασης δεδομένων ερωτημάτων-αιτήσεων. Αν υπάρχει ένα υπερβολικά μεγάλο περιεχόμενο πρέπει να χωριστεί σε κομμάτια, ώστε ο κάθε υπολογιστής να κάνει αναζητήσεις σε ένα κομμάτι. Ένας διακομιστής προσκηνίου στέλνει μία αίτηση-ερώτηση σε όλα τα κομμάτια, και μετά συγχωνεύει τα αποτελέσματα σε ένα μοναδικό σύνολο αποτελεσμάτων. Αν αντιθέτως υπάρχει μεγάλη διαβίβαση δεδομένων την ώρα που υπάρχει αυξημένη επισκεψιμότητα, τότε πρέπει να διανεμηθεί το ίδιο ευρετήριο και να αντιγραφεί σε άλλους υπολογιστές. Ένας ζυγός φόρτου προσκηνίου στέλνει κάθε εισερχόμενη αίτηση-ερώτηση στον λιγότερο φορτισμένο υπολογιστή στο παρασκήνιο. Αν χρειάζεται κανείς και τις δύο διαστάσεις κλιμάκωσης, όπως μια μηχανή αναζήτησης Ιστού, θα ταιριάξει και τις δύο πρακτικές.

Υπάρχουν αρκετές πολυπλοκότητες για να χτιστεί τέτοια αρχιτεκτονική. Χρειάζεται κανείς έναν αξιόπιστο τρόπο αντιγραφής του ευρετηρίου αναζήτησης μεταξύ υπολογιστών. Αν ένας υπολογιστής δεν λειτουργεί, προγραμματισμένα ή μη, χρειάζεται ένας τρόπος να ενημερωθεί μέχρι να μπει ξανά στη «παραγωγή». Η ανάκτηση από λάθος σε μία κατανεμημένη σύνθεση μπορεί να είναι περίπλοκο γεγονός. Τέλος, σημαντικές λειτουργίες όπως η ορθογραφία και επισήμανση όπως και ο τρόπος που υπολογίζεται το βάρος των όρων επηρεάζονται από την κατανεμημένη αρχιτεκτονική.

Το Lucene δεν προσφέρει άμεσα δυνατότητες για κλιμάκωση. Αλλά το **Solr** και το **Nutch** υποστηρίζουν την κατανομή ευρετηρίου και την αντιγραφή του. Επίσης υπάρχουν τα εργαλεία **Katta** και **Elastic Search** που είναι ανοιχτού κώδικα και βασίζονται πάνω στο Lucene. Καλό θα ήταν πριν να χτίσει κάποιος την δική του εκδοχή να κοιτάξει πρώτα στις λύσεις αυτές.

5.4 Βελτιστοποίηση (Optimization)

Όταν προσθέτουμε έγγραφα στο ευρετήριο ειδικά όταν είναι πολλά ή χρησιμοποιούμε πολλαπλές συνόδους με το IndexWriter, αναμφισβήτητα θα δημιουργηθεί ένα ευρετήριο το οποίο έχει πολλά ξεχωριστά τμήματα. Κατά την αναζήτηση όμως το Lucene θα πρέπει να αναζητάει κάθε τμήμα ξεχωριστά στο ευρετήριο και να συνδυάζει τα αποτελέσματα. Αν και αυτό δουλεύει άψογα, οι εφαρμογές που χειρίζονται τεράστια ευρετήρια θα δούν μεγάλη βελτίωση στην απόδοση χρησιμοποιώντας την Βελτιστοποίηση στα ευρετήριά τους. Αυτό γίνεται γιατί με τη διαδικασία αυτή πολλά τμήματα ενώνονται σε ένα η μερικά. Ένα βελτιστοποιημένο ευρετήριο επίσης ξοδεύει λιγότερους περιγραφείς αρχείων κατά την αναζήτηση. Αξίζει να σημειωθεί ότι **η Βελτιστοποίηση βελτιώνει τον χρόνο αναζήτησης αλλά όχι και τον χρόνο δημιουργίας ευρετηρίου.**

5.5.1 Μέθοδοι Βελτιστοποίησης

Είναι πολύ πιθανόν να έχουμε εξαιρετική διαμεταγωγή αναζήτησης και χωρίς Βελτιστοποίηση, έτσι πρέπει να είναι σίγουρος κανείς αν πραγματικά χρειάζεται

την Βελτιστοποίηση. Το IndexWriter περιλαμβάνει 4 μεθόδους για να βελτιστοποιεί :

- Optimize () : Μειώνει το ευρετήριο σε ένα και μοναδικό τμήμα, χωρίς να επιστρέφει αν δεν τελειώσει πρώτα η διαδικασία.
- Optimize (int maxNumSegments) : Επίσης γνωστή σαν Βελτιστοποίηση κατά μέρος, μειώνει το ευρετήριο σε maxNumSegments τμήματα το πολύ. Επειδή η συγχώνευση σε ένα μόνο τμήμα είναι και η πιο δαπανηρή, βελτιστοποιώντας σε 5 ας πούμε τμήματα θα είναι πολύ πιο γρήγορο απ'ότι να βελτιστοποιήσουμε σε ένα μόνο αλλά με αντίκτυπο λίγο πιο αργούς χρόνους αναζήτησης.
- Optimize (boolean doWait) : Είναι ακριβώς όπως η πρώτη περίπτωση, μόνο που το doWait εάν είναι false τότε η κλήση επιστρέφει αμέσως την ώρα που οι αναγκαίες συγχωνεύσεις γίνονται στο παρασκήνιο.
- Optimize (int maxNumSegments, boolean doWait) είναι ακριβώς ο συνδυασμός των δύο προηγούμενων μαζί.

5.5.2 Συμβουλές για Βελτιστοποίηση

Χρειάζεται προσοχή στην χρησιμοποίηση της Βελτιστοποίησης επειδή καταναλώνει σημαντική ποσότητα της κεντρικής μονάδας επεξεργασίας και των πόρων εισόδου / εξόδου. Είναι μια τιμή που πληρώνει κανείς μια φορά για να έχει γρηγορότερη αναζήτηση. Εάν ενημερώνει κανείς το ευρετήριό του σπάνια, και γίνονται πολλές αναζητήσεις σ'αυτό μεταξύ ενημερώσεων τότε είναι μια διαδικασία που αξίζει. Εάν ένας υπολογιστής κάνει και την δημιουργία ευρετηρίου και την αναζήτηση σ'αυτό, χρειάζεται να γίνεται η βελτιστοποίηση σε μια χρονική στιγμή που δεν θα υπάρχει κυκλοφορία για να μην υπάρχουν συγκρούσεις με τις αναζητήσεις.

Ένας άλλος παράγοντας που αξίζει προσοχή είναι η χρησιμοποίηση αρκετού προσωρινού χώρου αποθήκευσης στο σκληρό δίσκο κατά την Βελτιστοποίηση. Επειδή το Lucene πρέπει να συγχωνεύει κάποια τμήματα μαζί, η συγχώνευση χρησιμοποιεί την προσωρινή αποθήκευση για να κρατάει τα προσωρινά αρχεία για το καινούργιο τμήμα τα οποία δεν μπορούν να αφαιρεθούν εάν δεν δημιουργηθεί πρώτα η δημιουργία του τμήματος. Η διαδικασία αυτή μπορεί και να αυξήσει το μέγεθος του ευρετηρίου σχεδόν τριπλάσια το αρχικό μέγεθός του. Όταν τελειώσει όμως η διαδικασία και καλέσουμε τη μέθοδο commit() , η χρησιμοποίηση του δίσκου θα πέσει σε χαμηλότερο επίπεδο από του αρχικού.

Μέρος 3ο : Άλλες τεχνολογίες που χρησιμοποιήθηκαν στο JobFinder

Κεφάλαιο 6 : Το πρότυπο ASP.NET MVC

Εισαγωγή – μια ματιά στο παρελθόν

Γυρνώντας περίπου στα μέσα του 1990, οι προγραμματιστές δημιουργούσαν εκείνο τον καιρό διαδραστικές ιστοσελίδες χρησιμοποιώντας προγράμματα που έτρεχαν πάνω στο διακομιστή. Τα προγράμματα αυτά γνωστά ως CGI (Common Gateway Interface)που ήταν η κοινώς αποδεκτή τεχνολογία εκείνο το καιρό, δέχονταν μια αίτηση διαδικτύου και ήταν υπεύθυνα για την δημιουργία μιας HTML απάντησης. Η προτυποποίηση δεν ήταν σταθερή και τα προγράμματα ήταν δύσκολο να γραφτούν, να δοκιμαστούν και να εντοπιστούν τα σφάλματά τους. Στα τέλη του 1990, η Microsoft μετά από σύντομο πέρασμα με τα HTX πρότυπα και τις IDC διασυνδέσεις, παρουσίασε το ASP (Active Server Pages) και έφερε την προτυποποίηση στις διαδικτυακές εφαρμογές. Η σελίδα του διακομιστή ήταν ουσιαστικά μια μίξη HTML εγγράφου και δυναμικής δέσμης ενεργειών(scripting). Αν και αυτό ήταν ένα μεγάλο βήμα σε σύγκριση με τις εναλλακτικές επιλογές που υπήρχαν, οι σελίδες του διακομιστή άρχισαν να γίνονται υπερβολικά μεγάλες σε μέγεθος και αυτός ο συνδυασμός κώδικα και σήμανσης στο τέλος ήταν σχεδόν ακατανόητος.

Στις αρχές του 2002 η Microsoft δημοσίευσε την πρώτη έκδοση του πλαισίου .NET (NET Framework) που είχε τεράστια διαφορά από τον κόσμο του ASP προγραμματισμού.

Το Μοντέλο-Προβολή-Ελεγκτής έχει ρίζες από την κοινότητα προγραμματισμού Smalltalk από το ’70 , ενώ απέκτησε και αρκετή φήμη από την Ruby on Rails το 2003.

6.1 Τα μέρη του MVC

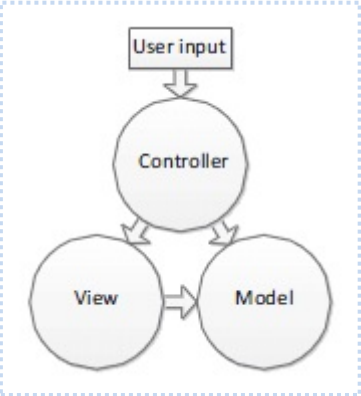
Τρία είναι τα μέρη που χαρακτηρίζουν το MVC:

- **Μοντέλο (Model)** – Ο χώρος που είναι χτισμένες οι κλάσεις του λογισμικού. Αν κάποιος δημιουργούσε ένα blog , τα μοντέλα θα ήταν *Δημοσίευση* και *Σχόλια.* Σε κάποια περιβάλλοντα, ο όρος μοντέλο μπορεί να αναφέρεται σε ένα συγκεκριμένο μοντέλο προβολής.
- **Προβολή (View)** – Η οπτική αναπαράσταση του μοντέλου, περιλαμβάνοντας κάποιο πλαίσιο. Είναι συνήθως το αποτέλεσμα της σήμανσης

που το πλαίσιο επεξεργάζεται προς τον περιηγητή, όπως είναι η HTML αναπαράσταση του blog δημοσιεύματος.

- **Ελεγκτής (Controller)** – Ο συντονιστής που γεφυρώνει το μοντέλο με την προβολή. Ο ελεγκτής είναι υπεύθυνος για την διαδικασία εισαγωγής δεδομένων, τη συμπεριφορά πάνω στο μοντέλο και τις αποφάσεις κάποιων αντιδράσεων όπως να παρουσιάζει μια Προβολή ή να ανακατευθύνει το περιεχόμενο προς άλλη σελίδα. Σύμφωνα με τα προηγούμενα παραδείγματα ένας ελεγκτής μπορεί να ψάξει τα πιο πρόσφατα σχόλια που αφορούν ένα Δημοσίευμα και να τα μεταφέρει στην Προβολή για να γίνει κανονικά η προβολή τους στο χρήστη.

6.2 Τι ακριβώς είναι το ASP.NET MVC?



Σχήμα 17: Τα μέρη του προτύπου MVC. Ο Ελεγκτής δέχεται την είσοδο του χρήστη, κατασκευάζει το κατάλληλο Μοντέλο, και το προωθεί στην Προβολή. Και ο Ελεγκτής και η Προβολή έχουν εξάρτηση ώς προς το Μοντέλο, το οποίο όμως είναι αδιάφορο ώς προς τους άλλους

Το πλαίσιο ASP.NET MVC της Microsoft είναι το πιο νέο της πλαίσιο προγραμματισμού για την δημιουργία διαδικτυακών εφαρμογών. Το πλαίσιο ASP. NET MVC σχεδιάστηκε από το μηδέν ώστε να ευκολύνει ακόμη περισσότερο την κατασκευή καλού, ποιοτικού και σωστά-δομημένου λογισμικού.

Δημιουργήθηκε για να υποστηρίζει την ανάπτυξη λογισμικού που βασίζεται σε κάποια πρότυπα σχεδίασης. Με λίγα λόγια να ευκολύνει την υλοποίηση σχεδιαστικών κανόνων και προτύπων λογισμικού στη διαδικασία ανάπτυξης των διαδικτυακών εφαρμογών και να υποστηρίζει τις δοκιμές μονάδων (Unit Tests). Γενικά οι διαδικτυακές εφαρμογές που είναι γραμμένες σύμφωνα μ'αυτό το πλαίσιο θεωρούνται ότι είναι δοκιμασμένες σε υψηλό βαθμό.

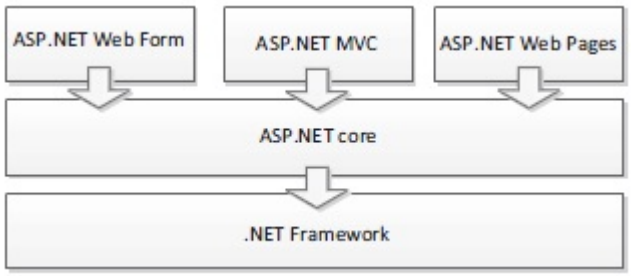
Γενικά, το πλαίσιο MVC παρουσιάστηκε σαν εναλλακτική λύση για τα **Web Forms**, και έκτοτε από το 2007 που ήταν η πρώτη κοινώς προβολή του, έχει γίνει πολύ δημοφιλής. Αξίζει να σημειωθεί ότι πολλές διαδικτυακές εφαρμογές μεγάλου μεγέθους έχουν υλοποιηθεί σύμφωνα με το πλαίσιο αυτό. Από την αρχή του μετράμε 4 εκδόσεις, η 3η που κυκλοφόρησε στο τέλος του Ιανουαρίου 2011 και είναι η πρώτη έκδοση του ASP.NET MVC που είναι εξαρτημένη από το .NET 4. Το

ASP .NET MVC 4 δουλεύει με το .NET 4 όσο καλά και με το .NET 4.5 το οποίο κυκλοφόρησε πρόσφατα.

6.3 Τι σημαίνει το MVC ?

Το MVC σημαίνει Model-View-Controller δηλαδή Μοντέλο-Προβολής-Ελεγκτή και είναι πολύ δημοφιλές σχεδιαστικό πρότυπο στο χώρο του Διαδικτυακού Προγραμματισμού.

Όπως είπαμε επειδή είναι εναλλακτική λύση των Web Forms, το ASP .NET MVC έχει μια διαφορετική προσέγγιση όσον αφορά τη δομή των διαδικτυακών εφαρμογών. Αυτό σημαίνει ότι δεν επεξεργάζεται κανείς τις ASPX σελίδες, τους διάφορους ελέγχους, τα postbacks, τις view καταστάσεις και περίπλοκες περιπτώσεις κύκλους ζωής. Αντιθέτως εργάζεται κανείς με τους ελεγκτές, τις δράσεις και τις προβολές. Παρ'ολα αυτά υπάρχουν και κοινά σημεία που μπορούν να χρησιμοποιηθούν όπως οι χειριστές HTTP και οι λειτουργικές μονάδες HTTP και αν υπάρχει ανάγκη μπορεί να γίνει μέχρι και συνδυασμός του MVC μαζί με τα Web Forms σε μία εφαρμογή. Όπως φαίνεται στο επόμενο σχήμα το MVC και τα Web Forms βρίσκονται στο ίδιο επίπεδο στη πλατφόρμα της ASP .NET.



Σχήμα 18: Η σχέση μεταξύ διάφορων .NET τεχνολογιών.

Η τεχνολογία "Web Pages" κυκλοφόρησε παράλληλα με το ASP .NET 3 και αποτελεί μια 3η εναλλακτική λύση σχετικά με τις άλλες δύο. Χαρακτηρίζεται κυρίως από μια απλότητα και έχει ώς σκοπό να βρει αποδέκτες νέους προγραμματιστές που θέλουν να μάθουν τη πλατφόρμα του ASP .NET. Για την τεχνολογία αυτή η Microsoft ανέπτυξες ξεχωριστό IDE, με τό όνομα WebMatrix και είναι πολύ πιο απλοποιημένο από το Visual Studio.

6.4 Ο ρόλος του MVC στο ASP.NET Framework

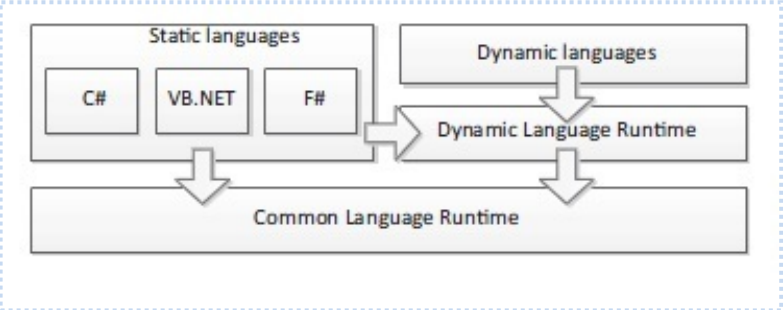
Το .NET αποτέλεσε τεράστιο άλμα για τους προγραμματιστές που ήταν εξοικειωμένοι με τις δυναμικές δέσμες ενεργειών του κλασσικού ASP. Το .NET παρουσίασε καινούργιες γλώσσες προγραμματισμού που μπορούσαν να μεταγλωττιστούν στην ίδια «μεσάζων γλώσσα» ώστε να τρέξουν στο CLR

(Common Language Runtime) του .NET . Αρχικά σ'αυτές τις γλώσσες περιλαμβάνονταν η C#, Visual Basic.NET και J#.

Με το καιρό οι γλώσσες που ήταν διαθέσιμες για το CLR εξελίχθηκαν. Με την τελευταία έκδοση του .NET Framework 4, οι διαθέσιμες γλώσσες που ήταν διαθέσιμες ήταν οι εξής.

- C# 4
- VB.NET 10
- F#

Επιπρόσθετα το .NET 4 περιλαμβάνει μια νέα ***Δυναμική Γλώσσα σε πραγματικό χρόνο εκτέλεσης*** (DLR- Dynamic Language Runtime) η οποία επιτρέπει δυναμικές γλώσσες προγραμματισμού να τρέχουν πάνω από το CLR.



Σχήμα 19: Οι γλώσσες .NET πάνω στο CLR

Το πλαίσιο της Microsoft για την ανάπτυξη εφαρμογών λογισμικού – όλων των ειδών εφαρμογών όπως για επιτραπέζιους Η/Υ, για το Διαδίκτυο και για εφαρμογές κονσόλας – είναι το γνωστό πλαίσιο της ***.NET (NET Framework)****. Το πλαίσιο αυτό αποτελείται από μια πληθώρα δεκάδων χιλιάδων κλάσεων, τις οποίες μπορεί να χρησιμοποιήσει κανείς κατά την δημιουργία οτιδήποτε λογισμικού. Για παράδειγμα , το πλαίσιο της .NET περιλαμβάνει κλάσεις που δουλεύουν με το σύστημα αρχείων κατά την πρόσβαση σε μια βάση δεδομένων χρησιμοποιώντας κανονικές παραστάσεις (regular expressions) και παραγωγή εικόνων.*

Το πλαίσιο ASP. NET περιέχει κλάσεις που δημιουργήθηκαν με ειδικό σκοπό την υποστήριξη διαδικτυακών εφαρμογών , τέτοιες κλάσεις για παράδειγμα όπως για το caching της σελίδας, την αυθεντικοποίηση και την εξουσιοδότηση χρηστών κ.α.

6.5 Τα πλεονεκτήματα του MVC έναντι Web Forms

Το MVC υπερέχει έναντι των Web Forms σε κάποια στοιχεία, τα οποία το κάνουν καλύτερη επιλογή για τη δημιουργία νέων εφαρμογών στη πλατφόρμα της .NET.

Πιο κοντά στο πρωτόκολλο

Όσο τα Web Forms προσπαθούν να κρύψουν την ανιθαγενή φύση του HTTP το MVC δεν το κάνει. Ασπάζοντας το πρότυπο MVC και αντιστοιχίζοντας μια αίτηση HTTP σε μία κλήση μεθόδου, το MVC προσφέρει μια αίσθηση ανάπτυξης λογισμικού που δεν συγκρίνεται με καμία άλλη από το προσκήνιο της ανάπτυξης λογισμικού. Το μοντέλο είναι επίσης έντονα απλοποιημένο : χάνονται πλέον οι περίπλοκοι κύκλοι ζωής γεγονότων των Web Forms και οι αφαιρέσεις πάνω στο πρωτόκολλο HTTP ελαχιστοποιούνται σε μεγάλο βαθμό.

Διαχωρισμός ανησυχιών

Καθώς τα Web Forms συνδέουν σφιχτά την διεπαφή χρήστη με το κώδικα που βρίσκεται στο παρασκήνιο, το MVC προωθεί τη σχεδίαση όπου η διεπαφή χρήστη (η Προβολή) κρατείται ξεχωριστά από το κώδικα που την ελέγχει (μέσω του Ελεγκτή). Όταν υπάρχει σωστή υλοποίηση, σημαίνει ότι οι εφαρμογές μπορούν να χρησιμοποιηθούν ευκολότερα από τους προγραμματιστές αλλά και να συντηρηθούν, δηλ. μια αλλαγή στον ελεγκτή δεν σημαίνει οπωσδήποτε ότι πρέπει να γίνει αλλαγή και στην διεπαφή χρήστη.

Ελεγξιμότητα

Διαχωρίζοντας την λογική της εφαρμογής από την διεπαφή χρήστη, το MVC κάνει ευκολότερο τον ξεχωριστό έλεγχο διαφορετικών συστατικών. Οι κλάσεις Ελεγκτών μπορούν να δοκιμαστούν χωρίς να χρειάζεται να δοκιμαστεί κάτι στην διεπαφή χρήστη. Αντιθέτως με τα Wev Forms ,οι MVC Ελεγκτές δεν έχουν απευθείας εξάρτηση από την ανήθικα αδοκίμαστη κλάση HttpContext αλλά αντιθέτως βασίζονται σε μια αφαιρετική έννοια η οποία κάνει ευκολότερο το γράψιμο αυτοποιημένων δοκιμών μονάδων (Unit tests).

6.6 Τι υπάρχει καινούργιο στο ASP .NET MVC 3 ή 4 (περιληπτικά)

- Η μηχανή προβολής Razor
- Διαχείριση πακέτων με το NuGet
- Βελτιωμένη επεκτασιμότητα
- Σφαιρικά φίλτρα ενεργειών
- Δυναμικά χαρακτηριστικά γλωσσών
- Caching στην έξοδο μερικού περιεχομένου μιας ιστοσελίδας
- Βελτιώσεις Ajax
- Ενίσχυση στις υποδομές επικύρωσης
- Πρότυπα για κινητά
- Διαδικτυακή διεπαφή προγραμματισμού εφαρμογής (Web API(Application Programming Interface))

6.7 Διαφορά υλοποίησης μεταξύ Razor και Web Forms

Αξίζει να σημειωθεί η διαφορά όσον αφορά τη σύνταξη Razor και την σύνταξη σε Web Forms δίνοντας ένα παράδειγμα για την υπεροχή του πρώτου.

Το παράδειγμα δείχνει μια απλή σελίδα που κατασκευάζει μια λίστα από ονόματα προϊόντων.

Σε **Web Forms** μορφή (στην Προβολή) :

```
<%@ Page Language="C#"
Inherits="System.Web.Mvc.ViewPage<Product[]>" %>
<ul>
<% foreach(var product in Model) { %>
<li><%: product.Name %></li>
<% } %>
</ul>
```

Σε **Razor** μορφή (στην Προβολή) :

```
@model Product[]
<ul>
@foreach(var product in Model) {
<li>@product.Name</li>
}
</ul>
```

Όπως φαίνεται το αποτέλεσμα σε σύνταξη Razor μορφής είναι πιο απλό ,καθαρό καθώς και το πιο σημαντικό ότι γλιτώνουμε παραπάνω χαρακτήρες στη σήμανση (@ αντί για <% και %>).

Κεφάλαιο 7 : C# - C Sharp

Εισαγωγή

Η C# ή C Sharp είναι μια πολλαπλό-παραδειγματική γλώσσα προγραμματισμού η οποία περιλαμβάνει δυναμική δακτυλογράφηση, επιτακτικές, δηλωτικές, λειτουργικές, γενικές, αντικειμενοστραφείς και συστατικοστρεφείς προγραμματιστικές πειθαρχίες.

Αναπτύχθηκε από την Microsoft μέσα στο περιβάλλον της .NET και αργότερα καθιερώθηκε σαν ISO στάνταρτ από την Ecma. Η C# είναι μια από τις γλώσσες προγραμματισμού που σχεδιάστηκαν για το CLR (Common Language Infrastructure – Κοινή γλώσσα υποδομής).

7.1 Το όνομα C#

Το όνομα “C Sharp” εμπνεύστηκε από τη μουσική σημειογραφία όπου το sharp υποδεικνύει ότι ή νότα πρέπει να είναι ένα ημίτονο ψηλότερα στην ένταση του τόνου. Αυτό είναι παρόμοιο με τη γλώσσα C++ όπου το “++” υποδεικνύει ότι η μεταβλητή πρέπει να προσαυξηθεί κατά 1.

Η κατάληξη # έχει χρησιμοποιηθεί και από άλλες .NET γλώσσες που είναι παραλλαγές των υπαρχόντων γλωσσών, περιλαμβάνοντας τις :

- J# (.NET γλώσσα επίσης σχεδιασμένη από τη Microsoft που προέρχεται από την Java 1.1)
- A# (απ’την γλώσσα Ada)
- F# (Λειτουργική γλώσσα προγραμματισμού

7.2 Σκοπός του Σχεδιασμού της

- Έχει προορισμό να θεωρείται μια απλή, μοντέρνα, αντικειμενοστραφής γλώσσα γενικών-σκοπών. Η πιο πρόσφατη έκδοση είναι η 5.0 που κυκλοφόρησε τον Αύγουστο του 2012.
- Η γλώσσα και οι υλοποιήσεις της, πρέπει να υποστηρίζουν κάποιους κανόνες ανάπτυξης λογισμικού όπως έλεγχο ισχυρού τύπου(strong type checking), έλεγχο ορίων πίνακα, προσπάθεια εντοπισμού και χρησιμοποίησης μεταβλητών που δεν έχουν οριστεί και αυτόματη συλλογή “σκουπιδιών”. Η ευρωστία λογισμικού, ανθεκτικότητα και παραγωγικότητα του προγραμματιστή είναι απαραίτητη.
- Υπάρχει σκοπός η γλώσσα να χρησιμοποιείται στην ανάπτυξη συστατικών λογισμικού που αναπτύσσονται σε κατανεμημένα περιβάλλοντα.
- Η μεταφερσιμότητα πηγαίου κώδικα είναι τόσο σημαντική, όσο και η μεταφερσιμότητα του προγραμματιστή, ειδικά για τους προγραμματιστές που είναι ήδη εξοικειωμένοι με την C και την C++.
- Υποστήριξη για διεθνοποίηση είναι πολύ σημαντική
- Η C# έχει προορισμό να είναι κατάλληλη για ανάπτυξη εφαρμογών και για συστήματα που φιλοξενούνται στο διαδίκτυο και για ενσωματωμένα συστήματα αρχίζοντας από πολύ μεγάλα τα οποία χρησιμοποιούν έξυπνα λειτουργικά συστήματα μέχρι πολύ μικρά τα οποία έχουν συγκεκριμένες λειτουργίες.
- Παρόλο που οι εφαρμογές σε C# σκοπεύουν να είναι οικονομικές όσον αφορά τη μνήμη και τις απαιτήσεις ρεύματος που απαιτούντα κατά την επεξεργασία, η γλώσσα δεν είχε σκοπό να ανταγωνιστεί σε επιδόσεις και μέγεθος τη C ή την γλώσσα ASSEMBLY.

7.3 Ιδιαίτερα Χαρακτηριστικά

Κάποια ιδιαίτερα χαρακτηριστικά που την ξεχωρίζουν από την C και C++ (και την Java) είναι:

- Δεν έχει global μεταβλητές ή λειτουργίες. Όλες οι μέθοδοι και μέλη δηλώνονται μέσα στις κλάσεις. Τα στατικά μέλη των δημόσιων κλάσεων μπορούν να αντικαταστήσουν τις global μεταβλητές και λειτουργίες.
- Οι τοπικές μεταβλητές δεν μπορούν να επισκιάζουν τις μεταβλητές σε ένα κλειστό μπλόκ, έτσι όπως στη C και στην C++.
- Η C# υποστηρίζει ένα αυστηρό τύπο δεδομένων Boolean, “bool”. Οι δηλώσεις που παίρνουν συνθήκες όπως το “while” και “if” απαιτούν την έκφραση ενός τύπου που υλοποιεί τον χειριστή “true” όπως είναι ο boolean. Καθώς η C++ επίσης έχει ένα τύπο boolean, μπορεί ελεύθερα να μετατραπεί σε ή από ακέραιο τύπο και εκφράσεις τύπου “if (a)” απαιτούν μόνο το a να μπορεί να μετατραπεί σε τύπου bool, επιτρέποντας το a να είναι η ακέραιος ή δείκτης. Η C# δεν επιτρέπει αυτό το “ακέραιος μπορεί να είναι σωστό η λάθος” γεγονός, γιατί θέλει αναγκαστικά να εξασφαλίζει τους προγραμματιστές να χρησιμοποιούν εκφράσεις που επιστρέφουν ακριβώς τύπο bool, πράγμα που μπορεί να ξεπεράσει τα κοινά λάθη προγραμματισμού στη C ή την C++ όπως το “if(a = b)” δηλ. αντί για το “==”.
- Στη C# οι δείκτες διευθύνσεων μνήμης(memory address pointers) μπορούν να χρησιμοποιηθούν μέσα σε μπλόκ ειδικά σταμπαρισμένα με τη λέξη “μη ασφαλής (unsafe)”, και τα προγράμματα με κώδικα μη-ασφαλής χρειάζονται συγκεκριμένες αρμοδιότητες για να εκτελεστούν. Η πρόσβαση σε αντικείμενα γίνεται περισσότερο μέσω ασφαλών αναφορών σε αντικείμενα, και πάντα δείχνει σε ένα “ζωντανό” αντικείμενο ή έχει τιμή null. Είναι εφικτό να αποκτήσει μια αναφορά κανείς σε ένα “νεκρό” αντικείμενο (αυτό που έχει μαζευτεί στα σκουπίδια (garbage collected)) ή σε ένα τυχαίο μπλόκ της μνήμης. Ένας μη ασφαλής δείκτης μπορεί να δείξει σε ένα στιγμιότυπο τύπου-τιμής, πίνακα, συμβολοσειράς ή σε ένα μπλόκ μνήμης που βρίσκεται σε μια στοίβα. Ο κώδικας που δεν είναι σταμπαρισμένος ώς μη-ασφαλής μπορεί να αποθηκεύσει ή χειριστεί δείκτες μέσω του τύπου System.IntPtr αλλά δεν μπορεί να διακόψει την αναφορά σ’αυτούς.
- Η διαχειρίζουσα μνήμη δεν μπορεί να απελευθερωθεί ρητά. Αντιθέτως μαζεύεται αυτομάτως στα σκουπίδια. Το αυτόματο συμμάζεμα των σκουπιδιών (garbage collection) βοηθά τον προγραμματιστή να μην έχει την ευθύνη να απελευθερώνει μνήμη για κάτι το οποίο πλέον δεν χρειάζεται, αποτρέποντας έτσι την διαρροή μνήμης που είναι γνωστό φαινόμενο.
- Εκτός από την “try…catch” κατασκευή για το χειρισμό εξαιρέσεων η C# διαθέτει και “try…finally” ώστε να εγγυάται την εκτέλεση του κώδικα στο μπλοκ finally είτε γίνει εξαίρεση είτε όχι.
- Πολλαπλή κληρονομικότητα δεν υποστηρίζεται, αν και μια κλάση μπορεί να υλοποιήσει οσεσδήποτε διεπαφές (interfaces). Αυτή ήταν μια απόφαση σχεδιασμού από τον ηγέτη-αρχιτέκτονα της γλώσσας για να αποφευχθεί η πολυπλοκότητα και να απλοποιηθούν οι αρχιτεκτονικές απαιτήσεις μέσω CLI.
- Η C# όπως και η C++ αλλά εκτός της Java υποστηρίζει υπερφόρτωση χειριστών.
- Η C# είναι πιο πολύ ασφαλές-τύπου από την C++.

7.4 Παράδειγμα απλούστατου προγράμματος

Παρακάτω προσφέρεται ένα απλούστατο πρόγραμμα σε C# , το γνωστό παράδειγμα εκτύπωσης στην οθόνη “HELLOWORLD”.

```
using System;
class Program
{
   static void Main()
   {
      Console.WriteLine("Hello world!");
   }
}
```

Κεφάλαιο 8 : Microsoft Visual Studio

8.1 Τί είναι το Visual Studio

Το Visual Studio της Microsoft είναι ένα ενοποιημένο περιβάλλον ανάπτυξης εφαρμογών της εταιρίας Microsoft. Χρησιμοποιείται για την ανάπτυξη εφαρμογών γραφικών διεπαφής χρήστη και κονσόλας, αλλά και εφαρμογές – φόρμες των Windows, σελίδες διαδικτύου, εφαρμογές Web, υπηρεσίες Web και σε εγγενή και σε διαχειριζόμενο κώδικα για όλες τις πλατφόρμες υποστηριζόμενες από τα Microsoft Windows, κινητά που χρησιμοποιούν Windows, Windows CE, πλαίσιο .NET, συμπαγές πλαίσιο .NET και το Microsoft Silverlight.

Το Visual Studio περιλαμβάνει ένα πρόγραμμα επεξεργασίας κώδικα που υποστηρίζει το IntelliSense και τον επανασχεδιασμό κώδικα. Ο ενσωματωμένος αποσφαλματωτής δουλεύει και ώς απασφαλματωτής επιπέδου – πηγής και επιπέδου μηχανής. Άλλα ενσωματωμένα εργαλεία είναι τα εξής : ένας σχεδιαστής για φόρμες που μπορεί να κατασκευάζει GUI εφαρμογές (Graphical User Interface), διαδικτυακός σχεδιαστής, σχεδιαστής κλάσεων και σχεδιαστής σχήματος βάσης δεδομένων. Το Visual Studio δέχεται πρόσθετα που επεκτείνουν την λειτουργικότητα σχεδόν σε κάθε επίπεδο – περιλαμβάνοντας την υποστήριξη για συστήματα ελέγχου – πηγής (π.χ. το Subversion και το Visual SourceSafe) προσθέτοντας νέα σετ εργαλείων όπως οι διορθωτές κειμένου και οι οπτικοί σχεδιαστές για συγκεκριμένες γλώσσες ή σετ εργαλείων που είναι προορισμένα για άλλους κύκλους ζωής ανάπτυξης λογισμικού (όπως το Team Foundation Server).

Επίσης το Visual Studio υποστηρίζει διάφορες γλώσσες προγραμματισμού οι οποίες επιτρέπουν στον διορθωτή κειμένου και στον απασφαλματωτή να υποστηρίζουν (ανάλογα το βαθμό) σχεδόν οποιαδήποτε γλώσσα προγραμματισμού, εάν βέβαια ήδη υπάρχει μια υπηρεσία για τη συγκεκριμένη γλώσσα. Οι ενσωματωμένες γλώσσες περιλαμβάνουν τις C/C++(μέσω Visual

C++), VB.NET (μέσω Visual Basic .NET), C#(μέσω Visual C#) και F#. Υποστήριξη για άλλες γλώσσες όπως M,Python, Ruby υπάρχει εάν υπάρχουν διαθέσιμες κάποιες υπηρεσίες γλωσσών και να έχουν εγκατασταθεί ξεχωριστά. Εκτός απ'αυτά υποστηρίζονται τα XML/XSLT,HTML /XHTML, JavaScript και CSS. Μεμονωμένες εκδόσεις του Visual Studio για συγκεκριμένες γλώσσες επίσης υπάρχουν αλλά προσφέρουν πιο περιορισμένες υπηρεσίες στον χρήστη : Microsoft Visual Basic, Visual J#, Visual C# και Visual C++.

Η Microsoft προσφέρει δωρεάν τις "Express" εκδόσεις του Visual Studio 2010 με τα συστατικά Visual Basic, Visual C#, Visual C++ και Visual Web Developer. Τα Visual Studio 2012,2010,2008 και 2005 Επαγγελματικές Εκδόσεις μαζί με συγκεκριμένες εκδόσεις γλώσσας (Visual Basic , C++,C#, J#) του Visual Studio 2010 είναι διαθέσιμα στους φοιτητές/μαθητές για λήψη μέσω του προγράμματος DreamSpark της Microsoft.

8.2 Χαρακτηριστικά

8.2.1 Διορθωτής κειμένου

Το Visual Studio όπως οποιοδήποτε άλλο Ολοκληρωμένο Περιβάλλον Ανάπτυξης (IDE – Integrated Development Environment) περιλαμβάνει τον διορθωτή κειμένου που υποστηρίζει την επισήμανση σύνταξης και ολοκλήρωση του κώδικα χρησιμοποιώντας το IntelliSense όχι μόνο για μεταβλητές, λειτουργίες και μεθόδους αλλά και σχήματα γλώσσας όπως βρόχους και αιτήσεις. Το IntelliSense υποστηρίζεται για τις συμπεριλαμβανόμενες γλώσσες, όπως και για το XML, τα CSS και JavaScript κατά την ανάπτυξη διαδικτυακών τόπων και εφαρμογών. Κατά το προγραμματισμό κώδικα μπορεί να προβάλλεται μία λίστα προτάσεων για αυτόματη συμπλήρωση του κώδικα. Στο Visual Studio 2008 η λίστα αυτή έχει τη δυνατότητα να φαίνεται σαν ημιδιαφανής ώστε να φαίνεται ο κώδικας από πίσω. Ο διορθωτής κώδικα χρησιμοποιείται για όλες τις υποστηριζόμενες γλώσσες.

Ο διορθωτής κειμένου του Visual Studio επίσης υποστηρίζει την τοποθέτηση σελιδοδεικτών στο κώδικα για γρήγορη πλοήγηση. Άλλα βοηθήματα πλοήγησης περιλαμβάνουν τα μπλοκ κώδικα να μπορούν να συσσωρεύονται σε μια γραμμή και την προοδευτική αναζήτηση με κανονικό κείμενο και με κανονικές εκφράσεις. Ο διορθωτής κειμένου υποστηρίζει επίσης πολλές λειτουργίες όπως λειτουργία επανασχεδιασμού κώδικα συμπεριλαμβάνοντας αναδιάταξη παραμέτρων, μετονομασία μεθόδων και μεταβλητών, εξαγωγή διεπαφής και ενθυλάκωση μελών κλάσης μέσα σε ιδιότητες κ.α.

8.2.2 Αποσφαλματωτής

Το Visual Studio περιλαμβάνει ένα αποσφαλματωτή και εκτός από τις λειτουργίες που αναφέραμε προηγουμένως μπορεί να επισυνάπτεται στις διεργασίες που εκτελούνται εκείνη τη στιγμή να τις παρακολουθεί και να τις

αποσφαλματώνει. Εάν ο πηγαίος κώδικας για την διεργασία που εκτελείται εκείνη τη στιγμή είναι διαθέσιμος, ο κώδικας μπορεί να είναι ορατός καθώς εκτελείται η διεργασία. Εάν δεν είναι διαθέσιμος τότε μπορεί να απεικονίζεται η ανακατασκευή του κώδικα. Επίσης υποστηρίζονται πολυνηματικές εφαρμογές και το γεγονός ότι ο απασφαλματωτής μπορεί να ρυθμιστεί να ενεργοποιείται όταν μια εφαρμογή έξω από το περιβάλλον του Visual Studio αντιμετωπίζει κάποιο σφάλμα.

Εκτός από αυτά ο αποσφαλματωτής μπορεί να καθορίζει σημεία διακοπής του προγράμματος σε συγκεκριμένα σημεία και να παρακολουθεί τις τιμές των μεταβλητών και τις διαδικασίες εκτέλεσης. Μια σημαντική λειτουργία υπάρχει που λέγεται “Edit and Continue” δηλ. “κάνε επεξεργασία και συνέχισε” η οποία επιτρέπει την επεξεργασία του κώδικα καθώς αποσφαλματώνεται αλλά δυστυχώς υποστηρίζεται μόνο στα συστήματα των 32 bit (όχι όμως των 64bit).Επίσης, υπάρχουν κάποια χρήσιμα στοιχεία όπως π.χ. κατά την αποσφαλμάτωση με μια κίνηση του δείκτη του ποντικιού πάνω σε μια μεταβλητή μπορεί κανείς να δει την τιμή που έχει εκείνη τη στιγμή.

8.2.3 Οπτικοί σχεδιαστές ανάπτυξης εφαρμογών

Το Visual Studio περιλαμβάνει μια σειρά από οπτικούς σχεδιαστές για να βοηθούν στην ανάπτυξη των εφαρμογών. Τα εργαλεία αυτά περιλαμβάνουν (Ονομαστικά):

- Σχεδιαστή για φόρμες Windows
- WPF Σχεδιαστή (για Θεμελιώδη παρουσίαση σε Windows)
- Δικτυακός σχεδιαστής/τρόπος ανάπτυξης
- Σχεδιαστή κλάσεων
- Σχεδιαστή δεδομένων
- Σχεδιαστή αντιστοίχισης

8.2.4 Εκδόσεις

Το Visual Studio διατίθεται στις παρακάτω εκδόσεις:

- Visual Studio Express
- Visual Studio Professional
- Visual Studio Premium
- Visual Studio Ultimate
- Visual Studio Test Professional

Κεφάλαιο 9 : jQuery

9.1 Τι είναι?

Το jQuery είναι μια βιβλιοθήκη της JavaScript για πληθώρα προγραμμάτων περιήγησης και σχεδιάστηκε για την απλοποίηση της εκτέλεσης δέσμης ενεργειών

HTML στην πλευρά του “πελάτη”. Κυκλοφόρησε το 2006 απ’τον προγραμματιστή John Resig. Αυτή τη στιγμή χρησιμοποιείται πάνω από το 55% από τους 10.000 πιο δημοφιλής διαδικτυακούς τόπους σε επισκεψιμότητα και θεωρείται η πιο δημοφιλής βιβλιοθήκη JavaScript.

Το jQuery είναι δωρεάν λογισμικό ανοιχτού κώδικα. Η σύνταξη είναι σχεδιασμένη ώστε να είναι ευκολότερη η περιήγηση ενός εγγράφου, επιλογή στοιχείων του εγγράφου αντικειμένων μοντέλου (DOM – Domain Object Model),την δημιουργία κινούμενων εικόνων, χειρισμό γεγονότων και ανάπτυξη εφαρμογών AJAX (Asynchronous JavaScript and XML). Το jQuery προσφέρει επίσης τη δυνατότητα στους προγραμματιστές να δημιουργούν πρόσθετα πάνω στη βιβλιοθήκη της JavaScript. Αυτό τους δίνει την δυνατότητα να δημιουργούν αφαιρέσεις για χαμηλού επιπέδου αλληλεπίδραση, κινούμενες εικόνες, προχωρημένα εφέ και υψηλού επιπέδου θέματα γραφικών στοιχείων. Με κατάλληλη προσέγγιση το jQuery μπορεί να βοηθήσει στην δημιουργία δυναμικών ιστοσελίδων και διαδικτυακών εφαρμογών.

Η Microsoft και η Nokia έχουν ανακοινώσει σχέδια να ενσωματώσουν το jQuery στις πλατφόρμες τους. Η Microsoft το ασπάζεται μέσα στο Visual Studio για χρήση με τα πλαίσια ASP.NET AJAX και ASP.NET MVC ενώ η Nokia το έχει ενσωματώσει στην διαδικτυακή εκτελέσιμη πλατφόρμα θεμάτων της.

9.2 Χαρακτηριστικά

Το jQuery περιλαμβάνει τα εξής χαρακτηριστικά :

- Επιλογή στοιχείων DOM χρησιμοποιώντας το λογισμικό ανοιχτού κώδικα Sizzle που δουλεύει σε πολλούς διαφορετικούς περιηγητές και μπορεί να κάνει επιλογή μηχανής - αποτελεί παρακλάδι του προϊόντος jQuery.
- Διέλευση μέσα στο DOM και τροποποίηση (υποστηρίζοντας τα CSS 1-3)
- Με βάση τους CSS επιλογείς, χειρισμός του DOM που χρησιμοποιεί ονόματα και ιδιότητες των κομβικών στοιχείων (id και class) σαν κριτήρια για την δημιουργία επιλογέων.
- Γεγονότα
- Εφέ και κινούμενες εικόνες
- AJAX
- Επεκτασιμότητα μέσω προσθέτων
- Παροχές – όπως πληροφορίες για το μεσολαβητή του χρήστη, αναγνώριση χαρακτηριστικών.
- Μέθοδοι συμβατότητας που είναι διαθέσιμοι σε μοντέρνους περιηγητές αλλά να μπορούν να υποστηρίζουν και παλιούς π.χ. οι λειτουργίες inArray() και each()
- Υποστήριξη για πολλούς περιηγητές.

9.3 Ένα απλό παράδειγμα

$("div.test").add("p.quote").addClass("blue").sideDown("slow");

Το πρώτο που βλέπουμε είναι το σύμβολο $ το οποίο είναι το χαρακτηριστικό σύμβολο μεθόδου για το jQuery αντικείμενο. Τυπικά η πρόσβαση και ο χειρισμός πολλαπλών κόμβων του DOM ξεκινάει μ'αυτό το σύμβολο.

Ύστερα ακολουθεί μια συμβολοσειρά που λέει βρες όλα τα div που έχουν κλάση "test",βρες και όλες τις παραγράφους(p) που έχουν κλάση "quote" κάνε δηλ. μια διασύνδεση με αυτά τα στοιχεία, και σε κάθε ένα από αυτά πρόσθεσε την κλάση "blue" και μετά αύξησε το ύψος τους αργά με ένα κινούμενο τρόπο.

Επίσης για να μπορούν να γίνονται δοκιμές jQuery σε ένα αυτοματοποιημένο πλαίσιο υπάρχει το Qunit με το οποίο μπορεί κανείς να κάνει ελέγχους στο κώδικά του, στα πρόσθετα αλλά επίσης και σε κώδικα της JavaScript. Μπορεί επίσης να κάνει ελέγχους στην πλευρά του διακομιστή (server).

Κεφάλαιο 10 : MySQL

10.1 Τι είναι και που χρησιμοποιείται

Η MySQL είναι το πιο χρησιμοποιημένο, ανοικτού κώδικα σύστημα διαχείρισης σχεσιακής βάσης δεδομένων(RDBMS) παγκοσμίως και από το 2008 μπορεί να δουλεύει σαν εξυπηρετητής προσφέροντας πολλαπλή-πρόσβαση χρηστών για πληθώρα βάσεων δεδομένων.

Η MySQL είναι μια δημοφιλής επιλογή βάσης δεδομένων για χρήση σε διαδικτυακές εφαρμογές και είναι το κεντρικό συστατικό για την ευρέως χρησιμοποιούμενη στοίβα λογισμικού LAMP (ανοιχτού κώδικα διαδικτυακή εφαρμογή). Το LAMP είναι ακρωνύμιο για "Linux, Apache,MySQL,Perl/PHP/Python". Πολλά δωρεάν και ανοιχτού κώδικα έργα χρησιμοποιούν σαν πλήρες σύστημα διαχείρισης βάσης δεδομένων την MySQL.

Για εμπορική χρήση υπάρχουν κάποιες εκδόσεις που ανέρχονται σε κόστος, και προσφέρουν πρόσθετη λειτουργικότητα. Οι εφαρμογές που χρησιμοποιούν MySQL βάσεις δεδομένων περιλαμβάνουν : TYPO3, Joomla, WordPress, phpBB, MyBB, Drupal κ.α. Η MySQL επίσης χρησιμοποιείται σε πολλά υψηλού-προφίλ μεγάλης-σκάλας προϊόντα παγκοσμίου ιστού, όπως Wikipedia, Google (όχι για αναζήτηση), Facebook, Twitter, Flickr, Nokia.com και YouTube.

Κανονικά δεν συνοδεύεται από καμία γραφική διεπαφή χρήστη για την διαχείριση των MySQL βάσεων δεδομένων ή την διαχείριση δεδομένων μέσα στις βάσεις δεδομένων. Οι χρήστες μπορούν να χρησιμοποιούν τα ενσωματωμένα εργαλεία γραμμής εντολών ή να χρησιμοποιούν "Front-End" λογισμικό και διαδικτυακές εφαρμογές που δημιουργούν και διαχειρίζονται βάσεις MySQL,

σχεδιάζουν σχήματα βάσεων δεδομένων, ελέγχουν καταστάσεις και δουλεύουν με έγγραφα δεδομένων. Το επίσημο σετ από “front-end” εργαλεία της MySQL είναι το **MySQL Workbench** που αναπτύσσεται από την Oracle, και είναι ελεύθερα διαθέσιμο για χρήση.

Το MySQL Workbench αντικαθιστά το προηγούμενο πακέτο λογισμικού, MySQL GUI Tools. Παρόμοιο με άλλα πακέτα τρίτων κατασκευαστών, αλλά γνωστό για την εγκυρότητά του, το MySQL Workbench αφήνει τους χρήστες να διαχειρίζονται τον σχεδιασμό & τη μοντελοποίηση βάσεων δεδομένων, ανάπτυξη SQL (αντικαθιστώντας το MySQL Query Browser) και την διαχείριση βάσεων δεδομένων (αντικαθιστώντας το MySQL Administrator).

Είναι διαθέσιμο σε δύο εκδόσεις, την απλή, δωρεάν και ανοιχτού κώδικα Community Edition της οποίας μπορεί να γίνει η λήψη από την σελίδα της MySQL,και την Standard Edition η οποία επεκτείνει και βελτιώνει τα χαρακτηριστικά της πρώτης έκδοσης. Υπάρχουν και άλλα εργαλεία που μπορούν να συνεργαστούν με την MySQL και να επιτρέπουν στους χρήστες να δουλεύουν οπτικά με τα σχήματα βάσεων και δεδομένων όπως Adminer, DaDaBIK, DBEdit, HeidiSQL, Navicat, OpenOffice, SQLBuddy.

Η MySQL μπορεί να δουλέψει σε πολλές διαφορετικές πλατφόρμες συστημάτων όπως: Microsoft Windows, Linux, Max OS X, OpenSolaris, Solaris, AIX, BSDi κ.α. Η MySQL είναι γραμμένη σε C και C++ και ο αναλυτής SQL της σε “yacc” και σε ένα “χειροποίητο” αναλυτή. Πολλές γλώσσες προγραμματισμού με συγκεκριμένες διεπαφές εφαρμογής προγραμματισμού (API) περιλαμβάνουν βιβλιοθήκες για πρόσβαση σε βάσεις δεδομένων της MySQL. Αυτές περιλαμβάνουν το MySQL Connector/Net για ενσωμάτωση στο Visual Studio(γλώσσες όπως η C# και Visual Basic χρησιμοποιούνται περισσότερο) και το JDBC της Java. Επιπλέον μια διεπαφή ανοικτής σύνδεσης βάσης δεδομένων (ODBC) που λέγεται MyODBC, επιτρέπει πρόσθετες γλώσσες προγραμματισμού που υποστηρίζουν την διεπαφή τύπου ODBC να μπορούν να επικοινωνούν με την βάση δεδομένων MySQL, τέτοια παραδείγματα όπως το ASP ή το ColdFusion.

10.2 Κάποια σημαντικά χαρακτηριστικά

Ανάμεσα από τα πολλά χαρακτηριστικά της MySQL είναι :

- Υποστήριξη πολλών πλατφόρμων
- Αποθηκευμένες λειτουργίες
- Σχήματα πληροφοριών
- Προβολές που ανανεώνονται
- Ανεξάρτητες μηχανές αποθήκευσης (το MyISAM για ταχύτητα ανάγνωσης, InnoDB για συναλλαγές και ακεραιότητα αναφορών, το MySQL Archive για αποθήκευση ιστορικών δεδομένων σε μικρό χώρο)
- Υποστήριξη SSL

- Caching των αιτήσεων προς τη βάση
- Πλήρους κειμένου δημιουργία ευρετηρίου και αναζήτηση χρησιμοποιώντας την μηχανή MyISAM.
- Υποστήριξη του Unicode
- Πολλαπλές μηχανές αποθήκευσης επιτρέποντας την επιλογή πιο κατάλληλης για κάθε πίνακα στην εφαρμογή (στην MySQL 5.1 οι μηχανές αποθήκευσης μπορούν να φορτωθούν δυναμικά στον χρόνο εκτέλεσης)

Κεφάλαιο 11 : Nhibernate

Εισαγωγή

Πολλές φορές όταν δημιουργείται κάποια καινούργια εφαρμογή(project) όσο απλή και να είναι, χρειάζεται ένα είδους αποθήκευσης. Μερικές φορές αυτό μπορεί να συνοδεύεται από μια συλλογή από αξίες σε ένα XML αρχείο ή ένα ζευγάρι κλειδιού-αξίας σε ένα αρχείο.

Άλλες φορές όμως χρειάζεται να έχουμε πρόσβαση σε μεγάλους όγκους δεδομένων που βρίσκονται σε πολλαπλούς πίνακες βάσεων δεδομένων. Σε κάθε περίπτωση χρειάζεται αρκετά συχνά η εύρεση μεθόδου μέσω της οποίας θα μπορούμε να αντλούμε και να αποθηκεύουμε τα δεδομένα που έχουμε σκοπό να προσπελαύνουμε. Εδώ παρουσιάζεται το Nhibernate.

11.1 Τι είναι το Nhibernate

Πολύ απλά, το Nhibernate είναι ένα πλαίσιο που μας επιτρέπει να “μιλάμε” με μια σχετική βάση δεδομένων με αντικειμενοστραφές τρόπο. Μπορούμε να αποθηκεύουμε (η πολύ συχνά λέμε “επιμένουμε”) αντικείμενα σε μια βάση δεδομένων και να τα φορτώνουμε από εκεί αργότερα. Το Nhibernate “αυτόματα-Μάγιά” μεταφράζει την αντικειμενοστραφής γλώσσα μας σε μια γλώσσα που την καταλαβαίνει μια βάση δεδομένων. Μ’αυτό το τρόπο το Nhibernate παράγει τις απαραίτητες εντολές σε SQL ώστε να μπορούμε να εισάγουμε, ενημερώνουμε, σβήνουμε και φορτώνουμε δεδομένα.

11.2 Με μια ματιά

Εάν χρησιμοποιούμε το Nhibernate, τότε δεν χρειάζεται ποτέ να γράφουμε κώδικα που να χειρίζεται την δυσκολία αναντιστοιχίας μεταξύ του τρόπου που προγραμματίζουμε εφαρμογές σε .NET και του τρόπου λειτουργίας μιας βάσης δεδομένων. Το Nhibernate αναλαμβάνει τη διαδικασία αυτή.

Όσον αφορά πώς προκύπτει η δυσκολία αναντιστοιχίας λεπτομερέστερα ας υποθέσουμε ότι έχουμε μια εφαρμογή που είναι γραμμένη σε .NET και είναι αντικειμενοστραφής. Έχουμε να αντιμετωπίσουμε αντικείμενα που περιέχουν

δεδομένα και λογική. Πολύ συχνά βλέπουμε ότι ένα αντικείμενο επικοινωνεί με άλλα αντικείμενα μέσω μηνυμάτων ή γεγονότων ενώ από την άλλη πλευρά, μια σχετική βάση δεδομένων σχετίζεται με μεγάλες ποσότητες δεδομένων. Οι σχετικές βάσεις δεδομένων παίζουν πολύ σπουδαίο ρόλο όταν πρέπει να χειριστούμε αυτές τις ποσότητες δεδομένων. Ωστόσο σε μια σχετική βάση δεδομένων δεν υπάρχει λογική στην έννοια των αντικειμένων. Η λογική και τα δεδομένα ζουν ξεχωριστά.

Με ένα πιο τυπικό τρόπο μπορούμε να πούμε ότι το Nhibernate είναι μια σχετική αντιστοίχιση αντικειμένων (OR – Object Relational Mapper) δηλ. ένα εργαλείο η ένα πλαίσιο. Το Nhibernate είναι η εκδοχή του Hibernate της Java. Είναι σχεδιασμένο να αλληλεπιδρά με τις βάσεις δεδομένων RDBMS (Relational Data Base Management Systems) μέσα από το κώδικα μιας εφαρμογής το οποίο το ενσωματώνει στο συγκεκριμένο αντικειμενοστραφές σχεδιασμό της.

Το Nhibernate δεν είναι το μοναδικό OR πλαίσιο για τον .NET αλλά μάλλον είναι το πιο ώριμο με τα πιο πλούσια χαρακτηριστικά από τα υπόλοιπα που υπάρχουν. Κάποια από τα υπόλοιπα που υπάρχουν αυτή τη στιγμή στην αγορά είναι : Το Entity Framework της Microsoft, το LLBLGen Pro, Subsonic και Genome. Το σήμα του Nhibernate είναι το εξής:

11.3 Γιατί να το χρησιμοποιήσει κανείς?

Στο παρελθόν γράφονταν πολύ προσεκτικά χειρωνακτικά στρώματα πρόσβασης δεδομένων(Data access layers). Ο χρόνος που ξοδευόταν ήταν 50%

Σχήμα 20: Το λογότυπο του Nhibernate

η και περισσότερος του συνολικού για να υλοποιηθεί και να συντηρηθεί το στρώμα αυτό. Δεν ήταν και πολύ σπουδαία διαδικασία όμως γιατί περιλάμβανε το γράψιμο αρκετού επαναλαμβανόμενου κώδικα. Για κάθε ένα και μοναδικό τύπο αντικειμένου που χρειαζόταν να καταχωρηθεί στη βάση δεδομένων, έπρεπε να γραφτεί και παρόμοιος κώδικας. Έπρεπε να γράφεται κώδικας για κάθε καινούργιο αντικείμενο που εισαγόταν στη βάση δεδομένων, που ενημερωνόταν και που διαγραφόταν. Ύστερα έπρεπε και πάλι να γράφεται κώδικας ώστε να διαβαστεί ένα μοναδικό αντικείμενο η μια λίστα αντικειμένων στη βάση δεδομένων. Το λιγότερο που χρειαζόταν ήταν το γράψιμο 6 μεθόδων ανά τύπο αντικειμένου.

Κάπως δημιουργήθηκε η εντύπωση ότι η διαδικασία αυτή δεν προσέδιδε αξία η κύρος στην επιχειρησιακή εφαρμογή που δημιουργούταν. Δεν έκανε την εφαρμογή συγκεκριμένη καθώς ήταν κώδικας υποδομής μόνο. Για κάθε μια

εφαρμογή που αναπτυσσόταν χρειαζόταν παρόμοιος κώδικας, είτε ήταν σύστημα παραγγελίας για ηλεκτρονικό κατάστημα είτε σύστημα διαχείρισης για ένα αεροδρόμιο είτε ένα σύστημα συναλλαγής για μια τράπεζα. Όλες οι εφαρμογές χρειάζονταν ένα παρόμοιο στρώμα πρόσβασης δεδομένων.

Συνεπώς αφού αυτή η διαδικασία είχε καταντήσει “βαρετή” γράφοντας τον ίδιο κώδικα ξανά και ξανά άρχισαν να γίνονται οι πρώτες σκέψεις για την αυτοματοποίησή της. Η πρώτη προσέγγιση ήταν κάποιοι παραγωγοί κώδικα οι οποίοι βασισμένοι πάνω σε κάποιες συγκεκριμένες ρυθμίσεις και πρότυπα μπορούσαν να γράφουν το κώδικα για το στρώμα αυτό. Με τη προσέγγιση αυτή λύθηκαν αρκετά προβλήματα και πλέον μπορούσαν οι προγραμματιστές να επικεντρώνονται περισσότερο σε πιο σημαντικά πράγματα όπως στο γράψιμο κώδικα, τη συντήρηση της επιχειρηματικής λογικής και της διεπαφής χρήστη, πράγματα τα οποία προσέδιναν πιο καλή και μοναδική αξία στο συνολικό προϊόν. Έτσι δημιουργήθηκε και το Nhibernate για να λύσει τα προβλήματα που υπήρχαν αποτελώντας την αρχή για τα πλαίσια OR.

Σήμερα η πρόσβαση σε σχεσιακά δεδομένα από μια αντικειμενοστραφής εφαρμογή αποτελεί ένα λυμένο πρόβλημα. Δεν υπάρχει λόγος να δημιουργηθεί κάτι καινούργιο αφού πλέον ο κώδικας πρόσβασης δεδομένων είναι περίπου ίδιος για τα προβλήματα ενός πεδίου ορισμού. Αυτό ισχύει και για καινούργιες τεχνολογίες όπως οι μεγάλοι πίνακες που χρησιμοποιούνται στη Google και αλλού.

Κάθε εφαρμογή που χειρίζεται δεδομένα πλέον, γράφει ή διαβάζει τα δεδομένα αυτά από ή προς το μέσο αποθήκευσης με τον ίδιο τρόπο. Γι’αυτό πλέον να γράψει κανείς ένα κώδικα διαχείρισης στρώματος δεδομένων είναι χάσιμο χρόνου. Όπως είπε κάποιος κάνοντας τέτοιο πράγμα είναι “σαν να κλέβεις χρήματα από τον ίδιο σου τον πελάτη”.

Τέλος, το Nhibernate υπάρχει εδώ και μερικά χρόνια, και πολλές εταιρείες το έχουν χρησιμοποιήσει σε πολλά διαφορετικά έργα ανάπτυξης λογισμικού. Έχει ανταπεξέλθει πάρα πολύ καλά σε βαριές και δύσκολες καταστάσεις που χρειάζονταν μεταφορά τεράστιου όγκου δεδομένων.

11.4 Τι είναι η Αντιστοίχιση (Mapping)

Αρχικά θα ήταν χρήσιμο να ειπωθεί τι είναι η αντιστοίχιση (mapping). Γνωρίζουμε ότι ένα μοντέλο κώδικα είναι αντικειμενοστραφές ενώ το σχήμα της βάσης δεδομένων είναι σχεσιακό. Το πρώτο συνεργάζεται με μεμονωμένα αντικείμενα, ενώ το δεύτερο δουλεύει με ποσότητες δεδομένων. Υπάρχει σαφώς μια αναντιστοιχία μεταξύ των δύο αυτών προσεγγίσεων. Για να καλυφθεί αυτό το κενό μεταξύ τους χρησιμοποιείται η αντιστοίχιση η οποία καθορίζει πώς τα δεδομένα που ζουν στο μοντέλο, στα αντικείμενα της και τις ιδιότητές τους βρίσκουν το τρόπο σύνδεσης με τους πίνακες της βάσης δεδομένων και τα πεδία τους.

Κατά το ξεκίνημα μιας εφαρμογής πληροφορικής, θέλουμε να καθορίσουμε το μοντέλο πρώτα και να το ακολουθήσει ύστερα η βάση δεδομένων και όχι αντίστροφα. Αν κάνουμε το σχήμα της βάσης δεδομένων να ανταποκρίνεται σωστά στο μοντέλο, η αντιστοίχιση γίνεται εύκολα και απλά. Μπορούμε να χρησιμοποιήσουμε κάποιες λογικές συνθήκες για τα περισσότερα μέρη των αντιστοιχίσεων ώστε να μην χρειάζεται να καθορίσουμε ρητά πολλές λεπτομέρειες ή ακόμη και να αφήσουμε το πλαίσιο να δημιουργήσει αυτόματα την αντιστοίχιση για εμάς.

Κύριοι τύποι αντιστοιχίσεων (mapping) όταν χρησιμοποιείται το Nhibernate ώς OR πλαίσιο :

- Fluent αντιστοίχιση
- Με βάση το XML
- Με βάση χαρακτηριστικών
- Αυτόματη αντιστοίχιση (αλλιώς Συνθηκών)

Θα δοθεί έμφαση στο πρώτο καθώς αυτό είναι που έχει χρησιμοποιηθεί στο JobFinder.

11.5 FLUENT MAPPING

Στη περίπτωση αυτή η αντιστοίχιση του πεδίου οντοτήτων με τους πίνακες της βάσης δεδομένων γίνεται με έναν ασφαλή-τύπο χρησιμοποιώντας το Fluent API, το οποίο κάνει το κώδικα πολύ ευανάγνωστο και κατανοητό. Η αντιστοίχιση γίνεται σε ξεχωριστή κλάση και έτσι δεν “μολύνει” την οντότητα. Το πλαίσιο που μας δίνει την δυνατότητα αυτή να κάνουμε την αντιστοίχιση με ευχέρεια (fluently), χρησιμοποιεί εκφραστικά δέντρα και στατική αντανάκλαση για να μας προσφέρει την ασφάλεια-τύπων, όπου δεν χρειάζεται να χρησιμοποιήσουμε μαγικές συμβολοσειρές.

11.5.1 Ένα απλό παράδειγμα

Για παράδειγμα θα αναφέρουμε πώς μπορούμε να αντιστοιχίσουμε μια οντότητα Product χρησιμοποιώντας την Fluent αντιστοίχιση. Δημιουργούμε μια κλάση αρχικά (συνήθως με το όνομα της κλάσης που θέλουμε να αντιστοιχίσουμε + τη κατάληξη Map) ProductMap η οποία κληρονομεί από την ClassMap<T> όπου T το όνομα της κλάσης που αντιστοιχίζουμε.

Public class ProductMap : ClassMap<Product>

{..}

Στο δομητή της κλάσης, καθορίζουμε τις λεπτομέρειες της αντιστοίχισης. Τα βασικά χαρακτηριστικά αντιστοιχίζονται χρησιμοποιώντας την μέθοδο Map και δίνοντας ώς παράμετρο, έκφραση τύπου lambda (συνήθως τη καταλαβαίνουμε από το σύμβολο “=>”), καθορίζοντας το χαρακτηριστικό της κλάσης που

αντιστοιχίζουμε. Για το χαρακτηριστικό Name της κλάσης Product θα γράφαμε το ακόλουθο:

```
Public ProductMap()
{
   Map(x=>x.Name);
}
```

Ο κώδικας αυτός είναι ο δομητής της κλάσης ProductMap και αντιστοιχίζει το χαρακτηριστικό Name στο πεδίο Name του πίνακα Product της βάσης δεδομένων. Εάν θέλουμε να προσθέσουμε κάποιες λεπτομερείς πληροφορίες μπορούμε να προσθέσουμε π.χ. να μην μένει κενό η μέγιστο μήκος χαρακτήρων να μην είναι πάνω από 50 :

```
Map(x=>x.Name)
.Not.Nullable()
.Length(50);
```

11.5.2 Πλεονεκτήματα και μειονεκτήματα

Τα πλεονεκτήματα της αντιστοίχισης Fluent είναι:

♦ Ο κώδικας που χρησιμοποιείται για να καθορίσει την αντιστοίχιση είναι πολύ ευανάγνωστος και κατανοητός.

♦ Η αντιστοίχιση κρατείται ξεχωριστά από την οντότητα, και έτσι δεν μολύνει την οντότητα.

♦ Η αντιστοίχιση είναι ασφαλές-τύπου. Δεν χρησιμοποιούνται μαγικές συμβολοσειρές.

♦ Οι πιθανές επιλογές και ρυθμίσεις για κάθε στοιχείο είναι ανακαλύψιμες καθώς η Fluent αντιστοίχιση υποστηρίζεται από το IntelliSense.

Τα μειονεκτήματα :

♦ Πρέπει να καθορίζουμε μια κλάση αντιστοίχισης για κάθε οντότητα και αντικείμενο αξίας.

♦ Κάποιες “εξωτικές” αντιστοιχίσεις δεν υποστηρίζονται, αλλά η κατάσταση καλυτερεύει με κάθε νέα έκδοση του Fluent Nhibernate.

♦ Καθορίζοντας τις αντιστοιχήσεις με το Fluent τρόπο έχει επιβάρυνση στο όταν αρχίζει μια εφαρμογή αυτές θα πρέπει να μεταφραστούν σε αντιστοιχήσεις XML, οι οποίες μετά θα αναλυθούν και χρησιμοποιηθούν από το Nhibernate. Αυτό το βήμα δεν είναι απαραίτητο όταν οι αντιστοιχίσεις γίνονται κατευθείαν σε XML αρχεία.

♦ Όταν το όνομα του πίνακα της βάσης διαφέρει από το όνομα της οντότητας ή το όνομα πεδίου από το όνομα του χαρακτηριστικού της οντότητας, τότε πρέπει να χρησιμοποιήσουμε μαγικές συμβολοσειρές.

Κεφάλαιο 12 : Dependency Injection

12.1 Τι ακριβώς είναι

Το Dependency Injection είναι μια πατέντα σχεδίασης λογισμικού που επιτρέπει την επιλογή ενός συστατικού να γίνεται στο χρόνο εκτέλεσης και όχι στο χρόνο μεταγλώττισης. Αυτό μπορεί να χρησιμοποιηθεί σαν ένας απλός τρόπος για τη δυναμική φόρτωση προσθέτων ή την επιλογή τεχνητών αντικειμένων σε περιβάλλοντα δοκιμής, σε σύγκριση με αληθινά αντικείμενα σε περιβάλλοντα παραγωγής. Αυτή η πατέντα σχεδίασης λογισμικού εμβάλλει (injects) τα εξαρτώμενα στοιχεία (αντικείμενα ή αξίες κτλ.) στον προορισμό αυτόματα, γνωρίζοντας απλά την απαίτηση του προορισμού. Υπάρχει και μια άλλη πατέντα , με το όνομα “dependency lookup” η οποία είναι μια κανονική διαδικασία και θεωρείται η αντίστροφη διαδικασία του “dependency injection”.

12.2 Τι ακριβώς κάνει

Το dependency injection αναμειγνύει τουλάχιστον 3 στοιχεία:

- έναν εξαρτώμενο καταναλωτή
- μία δήλωση των εξαρτήσεων ενός συστατικού, γνωστών ώς συμβάσεις διεπαφής
- ένα έμβολο (πολλές φορές αναφερόμενο ώς πάροχο ή δοχείο) που δημιουργεί στιγμιότυπα κλάσεων οι οποίες υλοποιούν την τρέχουσα διεπαφή εξάρτησης, μόλις απαιτηθεί.

Το εξαρτώμενο αντικείμενο περιγράφει από ποιό συστατικό λογισμικού εξαρτάται για να κάνει τη δουλειά του. Το έμβολο αποφασίζει ποιές συμπαγείς κλάσεις ανταποκρίνονται στις απαιτήσεις του εξαρτώμενου αντικειμένου, και τις προσφέρει σ’αυτό.

Στο συμβατικό προγραμματισμό λογισμικού, το εξαρτώμενο αντικείμενο αποφασίζει για το εαυτό του για το ποιές συμπαγείς κλάσεις θα χρησιμοποιήσει. Στη πατέντα του dependency injection, η απόφαση εξουσιοδοτείται στο έμβολο το οποίο μπορεί να διαλέξει την αντικατάσταση διάφορων υλοποιήσεων (συμπαγών κλάσεων μιας διεπαφής εξαρτώμενης σύμβασης) σε εκτελέσιμο χρόνο αντί σε χρόνο μεταγλώττισης.

Η δυνατότητα να παίρνει αποφάσεις στο χρόνο-εκτέλεσης και όχι στον χρόνο μεταγλώττισης είναι το βασικότερο πλεονέκτημα του dependency injection. Πολλαπλές, διαφορετικές υλοποιήσεις ενός και μοναδικού συστατικού μπορούν να δημιουργούνται στο χρόνο εκτέλεσης και να εμβάλλονται (injected) στο ίδιο κώδικα ελέγχου. Ο κώδικας ελέγχου ύστερα μπορεί να ελέγχει κάθε διαφορετικό συστατικό λογισμικού χωρίς να γνωρίζει ότι αυτό που είχε περαστεί στο συστατικό αυτό, είναι υλοποιημένο διαφορετικά.

Ο έλεγχος συστατικών μιας μονάδας σε μεγάλα συστήματα λογισμικού είναι δύσκολος, επειδή τα συστατικά όταν ελέγχονται συχνά απαιτούν την παρουσία μιας αρκετά δυνατής αρχιτεκτονικής και συναρμολόγησης ώστε να μπορέσουν να λειτουργήσουν, γεγονός το οποίο αποτελεί ένα κίνητρο για τη χρήση του dependency injection. Η τεχνική λοιπόν αυτή απλοποιεί την διαδικασία να φέρνει στο προσκήνιο ένα στιγμιότυπο ενός απομονωμένου συστατικού που να λειτουργεί. Επειδή τα συστατικά δηλώνουν τις εξαρτήσεις τους, ένας έλεγχος μπορεί αυτόματα να επικαλέσει μόνο εκείνες τις εξαρτήσεις που χρειάζονται να ελεγχθούν.

Χωρίς το dependency injection, το συστατικό καταναλωτής που χρειάζεται μια συγκεκριμένη υπηρεσία για να ολοκληρώσει μια εργασία, πρέπει να δημιουργεί ένα στιγμιότυπο της συγκεκριμένης κλάσης που υλοποιεί την εξαρτώμενη επαφή.

Όταν χρησιμοποιούμε το dependency injection, το συστατικό καταναλωτής καθορίζει την συμφωνία υπηρεσίας μέσω διεπαφής (interface), και το συστατικό που εμβάλλει επιλέγει μια υλοποίηση στη θέση του εξαρτώμενου συστατικού.

Πιο σύνθετες υλοποιήσεις, όπως Spring, Google Guice και Microsoft Managed Extensibiliy Framework (MEF) αυτοματοποιούν αυτή τη διαδικασία. Αυτά τα πλαίσια αναγνωρίζουν τις παραμέτρους του δομητή ή τα χαρακτηριστικά πάνω στα αντικείμενα που δημιουργούνται ώς αιτήσεις για εξαρτώμενα αντικείμενα. Ύστερα, αυτόματα εμβάλλουν τις παραμέτρους του δομητή ή καθορίζουν τα χαρακτηριστικά με προ-κατασκευασμένα στιγμιότυπα των εξαρτήσεων σαν μέρος της διαδικασίας δημιουργίας του εξαρτώμενου αντικειμένου. Ο πελάτης κάνει αίτηση στο σύστημα του dependency injection για υλοποίηση μιας συγκεκριμένης διεπαφής: το σύστημα δημιουργεί το αντικείμενο και αυτόματα το γεμίζει με τις εξαρτήσεις όπως χρειάζεται.

12.3 Πλεονεκτήματα

Ένα πλεονέκτημα χρησιμοποιώντας το dependency injection είναι η μείωση του επαναλαμβανόμενου κώδικα στα αντικείμενα της εφαρμογής αφού όλη τη δουλειά της αρχικοποίησης ή του καθορισμού των εξαρτήσεων τη χειρίζεται το συστατικό του προμηθευτή. Άλλο πλεονέκτημα είναι ότι προσφέρει ευκαμψία ρυθμίσεων επειδή οι εναλλακτικές υλοποιήσεις μιας υπηρεσίας μπορούν να χρησιμοποιηθούν χωρίς επαναγλωττισμένο κώδικα. Αυτό είναι χρήσιμο στον έλεγχο μονάδας (unit testing), επειδή είναι εύκολο να εμβάλλει μια ψεύτικη υλοποίηση της υπηρεσίας μέσα στο αντικείμενο που δοκιμάζεται, αλλάζοντας το αρχείο ρυθμίσεων ή να παρακάμπτει τις καταχωρήσεις των συστατικών στον χρόνο εκτέλεσης. Εκτός από αυτά, το dependency injection διευκολύνει το γράψιμο κώδικα ελέγχου.

Τύποι (σύμφωνα με τον Martin Fowler) με τους οποίους ένα αντικείμενο μπορεί να πάρει αναφορά σε μια εξωτερική ενότητα, σύμφωνα με την πατέντα που χρησιμοποιείται για να προσφέρει την εξάρτηση:

- Τύπος 1 ή “interface injection”, στην οποία η εξαγόμενη ενότητα προσφέρει μια διεπαφή την οποία οι χρήστες της πρέπει να υλοποιήσουν ώστε να λάβουν τις εξαρτήσεις κατά τον χρόνο εκτέλεσης.
- Τύπος 2 ή “setter injection”, στην οποία η εξαρτώμενη ενότητα εκθέτει μία μέθοδο ρυθμιστή την οποία το πλαίσιο χρησιμοποιεί για να εμβάλλει την εξάρτηση.
- Τύπος 3 ή “constructor injection”, στην οποία οι εξαρτήσεις προσφέρονται μέσα από το δομητή της κλάσης.

Είναι δυνατόν για άλλα πλαίσια να έχουν και άλλους τύπους εμβολής, εκτός από τους παραπάνω.

12.4 NINJECT

12.4.1 Τί είναι το Ninject?

Το Ninject είναι πάρα πολύ γρήγορο, ελαφρύ έμβολο εξάρτησης (dependency injector) για .NET εφαρμογές. Βοηθάει κάποιον να χωρίσει μια εφαρμογή σε μια συλλογή από χαλαρά-συνδεδεμένες, υψηλής-σύζευξης κομμάτια, και μετά να τα κολλήσει μαζί με ένα ευλύγιστο τρόπο. Χρησιμοποιώντας το Ninject για την υποστήριξη της αρχιτεκτονικής ενός λογισμικού, ο κώδικας γίνεται ευκολότερος να γραφτεί, επαναχρησιμοποιηθεί, δοκιμαστεί και να αλλαχτεί.

12.4.2 Χαρακτηριστικά του Ninject

Στοιχεία που χαρακτηρίζουν το Ninject:

1. “Επικεντρωμένο”. Πάρα πολλά έργα κατηγορίας DI (Dependency Injection) θυσιάζουν την χρηστικότητα για κάποια χαρακτηριστικά τα οποία συνήθως δεν είναι απαραίτητα. Κάθε φορά που ένα χαρακτηριστικό προστίθεται στο Ninject, η ωφέλεια του αποτελεί επιβάρυνση στην πολυπλοκότητα που προσθέτει στην καθημερινή χρήση. Ο σκοπός της δημιουργίας του Ninject είναι να χρειάζεται μια όσο τον δυνατόν γίνεται λιγότερο γνώση. Το Ninject έχει πολλά προχωρημένα χαρακτηριστικά, αλλά η κατανόησή τους δεν είναι απαραίτητα για την χρήση των απλών χαρακτηριστικών.
2. “Λείο”. Το φούσκωμα του πλαισίου είναι μια μεγάλη ανησυχία για κάποια έργα, γι’αυτό και η λειτουργία του πυρήνα του Ninject βρίσκεται σε μια συναρμολόγηση αρχείου χωρίς εξαρτήσεις έξω απ’την βασική κλάση της βιβλιοθήκης .NET.
3. “Γρήγορο”. Το Ninject εκμεταλλεύεται τον ελαφρύ κώδικα των γενικών χαρακτηριστικών στην έκδοση 2.0 του CLR. Αυτό μπορεί να έχει ώς αποτέλεσμα 8-50 φορές στη βελτίωση απόδοσης πολλών καταστάσεων.
4. “Ακριβής”. Το Ninject βοηθά τους προγραμματιστές να τακτοποιούν τα πράγματα σωστά από την πρώτη φορά.

Μέρος 4ο : Η εφαρμογή JobFinder

Κεφάλαιο 13 : Τρόπος λειτουργίας του JobFinder

13.1 Αρχική σελίδα

Η αρχική σελίδα του JobFinder περιλαμβάνει αρχικά στο πάνω μέρος την στάνταρ αναζήτηση και για τις δύο περιπτώσεις (εργοδότη και αυτούς που αναζητούν εργασία). Συγκεκριμένα υπάρχουν δύο πεδία αναζήτησης, το πρώτο είναι με βάση το επάγγελμα και το δεύτερο είναι με βάση την γεωγραφική περιοχή. Η αναζήτηση δουλεύει είτε συμπληρώνοντας κάποιο από τα δύο πεδία είτε και τα δύο για πιο συγκεκριμένα αποτελέσματα. Τα αποτελέσματα εμφανίζουν και τις αγγελίες εργοδοτών που ψάχνουν για κάποιο υπάλληλο και αυτούς που ψάχνουν για εργασία έχοντας στην εμπειρία τους το συγκεκριμένο επάγγελμα. Αν δεν συμπληρωθεί το επάγγελμα τότε εμφανίζονται οι αγγελίες με κριτήριο την γεωγραφική περιοχή. Υπάρχει ανάλυση στη συνέχεια για την αναζήτηση αυτή.

Πάνω από την αναζήτηση διακρίνονται δύο κουμπιά : πατώντας το πρώτο με το όνομα **CVs** μπορούμε να δούμε τις τελευταίες δουλειές από τα βιογραφικά που προστέθηκαν ενώ πατώντας το δεύτερο με το όνομα **Jobs** μπορούμε να δούμε τις τελευταίες αγγελίες που προστέθηκαν.

Στα δεξιά αυτών των κουμπιών έχουμε τις υπόλοιπες τρείς επιλογές αναζήτησης : **Detailed Search** , **Live Search** και **Faceted Search.** Και οι τρείς αναζητήσεις θα εξηγηθούν λεπτομερώς στη συνέχεια.

Δεξιά από τις αναζητήσεις αυτές βλέπουμε δύο επιλογές εάν ο χρήστης δεν έχει ταυτοποιηθεί. Η μία είναι **“Join Us”** την οποία πατώντας ο χρήστης μεταφέρεται στην σελίδα της δημιουργίας καινούργιου λογαριασμού. Η δεύτερη επιλογή είναι **“Log In”** πατώντας στην οποία ο χρήστης μεταφέρεται στο μενού ταυτοποίησης εισαγωγής του χρήστη.

13.2 Προβολή τελευταίων καταχωρίσεων

Όπως είπαμε αφορά τα δύο κουμπιά που αναφέραμε προηγουμένως : Το **Cvs** για προβολή τελευταίων δουλειών των βιογραφικών που καταχωρήθηκαν και το **Jobs** για την προβολή των τελευταίων αγγελιών. Και στις δύο περιπτώσεις έχουμε μία λίστα αποτελεσμάτων που φαίνεται σαν ένας πίνακας εγγραφών όπου στο CV υπάρχουν 3 πεδία για κάθε εγγραφή και στο Jobs 4.

Τα πεδία στο CV είναι τα εξής :

- *Είδος υπαλλήλου* : ο τίτλος επαγγέλματος στο βιογραφικό του χρήστη
- *Προϋπηρεσία* : πόσα χρόνια την έχει κάνει τη συγκεκριμένη δουλειά
- *Περιοχή* : όπου μπορούμε να δούμε την περιοχή που διαμένει αυτός που έχει κάνει την δουλειά αυτή.

Παρομοίως τα πεδία στο Jobs είναι :

- *Είδος υπαλλήλου* : ο τίτλος επαγγέλματος που ζητείται να έχει ο υποψήφιος
- *Πού* : σε ποιό μέρος ζητείται
- *Από* : ποιά εταιρία έχει δημοσιεύσει αυτή την αγγελία
- *Πότε* : πότε καταχωρήθηκε η αγγελία αυτή

Κάνοντας κλίκ σε κάποια εγγραφή από αυτές μεταφερόμαστε σε άλλη σελίδα. Η σελίδα που θα μεταφερθούμε εξαρτάται βέβαια από την εγγραφή στην οποία κάναμε κλίκ δηλ. άμα είναι τύπου Cv ή Jobs. Για να δούμε την κάθε περίπτωση ξεχωριστά:

- Εάν πατήσουμε πάνω σε μια **δουλειά** ενός βιογραφικού μεταφερόμαστε στη σελίδα που μπορούμε να δούμε πληροφορίες για τον χρήστη αυτόν που έχει κάνει τη δουλειά αυτή. Μπορούμε να δούμε τι άλλες δουλειές έχει κάνει και κάποιες πληροφορίες που αφορά τον ίδιο.
- Εάν πατήσουμε σε μια **αγγελία** που έχει δημοσιευτεί, εδώ υπάρχουν δύο ενδιαφέρον περιπτώσεις. Εάν η αγγελία αυτή έχει δημοσιευτεί από τον εργοδότη ο οποίος βρίσκεται ήδη ταυτοποιημένος στην σελίδα τότε μεταφέρεται στο μενού επεξεργασίας της αγγελίας ώστε αν θέλει κάτι μπορεί να αλλάξει σ'αυτήν. Μπορεί ακόμη να τη σβήσει αν θελήσει. Σε κάθε άλλη περίπτωση μεταφερόμαστε σε άλλη σελίδα όπου έχουμε τη δυνατότητα να προβάλουμε μόνο τις πληροφορίες της αγγελίας.

13.3 Εγγραφή καινούργιου χρήστη

Η εγγραφή ενός καινούργιου χρήστη προϋποθέτει την επιλογή του είτε να είναι Εργοδότης (Employer) είτε να είναι κάποιος που ψάχνει για δουλειά (Job Seeker). Ανάλογα με την επιλογή του διαμορφώνονται στη συνέχεια και τα ανάλογα μενού στα οποία θα συμπληρώνει τα κατάλληλα πεδία που χρειάζονται.

13.3.1 Αναζητώντας εργασία (Job Seeker)

Έχοντας κάνει εγγραφή σαν κάποιος που ψάχνει για εργασία και αφού έχει ταυτοποιηθεί από το σύστημα θα εμφανιστούν πάνω δεξιά στο μενού κάποια στοιχεία και σύνδεσμοι. Το πρώτο από πάνω αριστερά είναι ένα μήνυμα καλωσορίσματος ("Welcome + όνομα χρήστη"). Ακολουθεί ο υπερσύνδεσμος **"my profile"** όπου ο χρήστης μεταφέρεται σε άλλη σελίδα όπου χρειάζεται να βάλει τα προσωπικά του στοιχεία. Ακριβώς κάτω από τα αριστερά εμφανίζεται ο υπερσύνδεσμος **"my curriculum vitae"** ο οποίος οδηγεί σε άλλη σελίδα εκεί που συμπληρώνεται το βιογραφικό του χρήστη. Και οι δύο σελίδες θα αναλυθούν πιο αναλυτικά στη συνέχεια. Εδώ είναι σημαντικό να πούμε ότι όταν έναν νέος χρήστης κάνει εγγραφή και θελήσει να συμπληρώσει το βιογραφικό του, εάν δεν έχει συμπληρώσει τα στοιχεία του πρώτα θα μεταφερθεί στην σελίδα αυτή όπου θα του εμφανιστεί το μήνυμα ότι πρέπει πρώτα να συμπληρώσει τα προσωπικά του στοιχεία πριν να μπορέσει να συμπληρώσει το βιογραφικό. Ακριβώς από τα

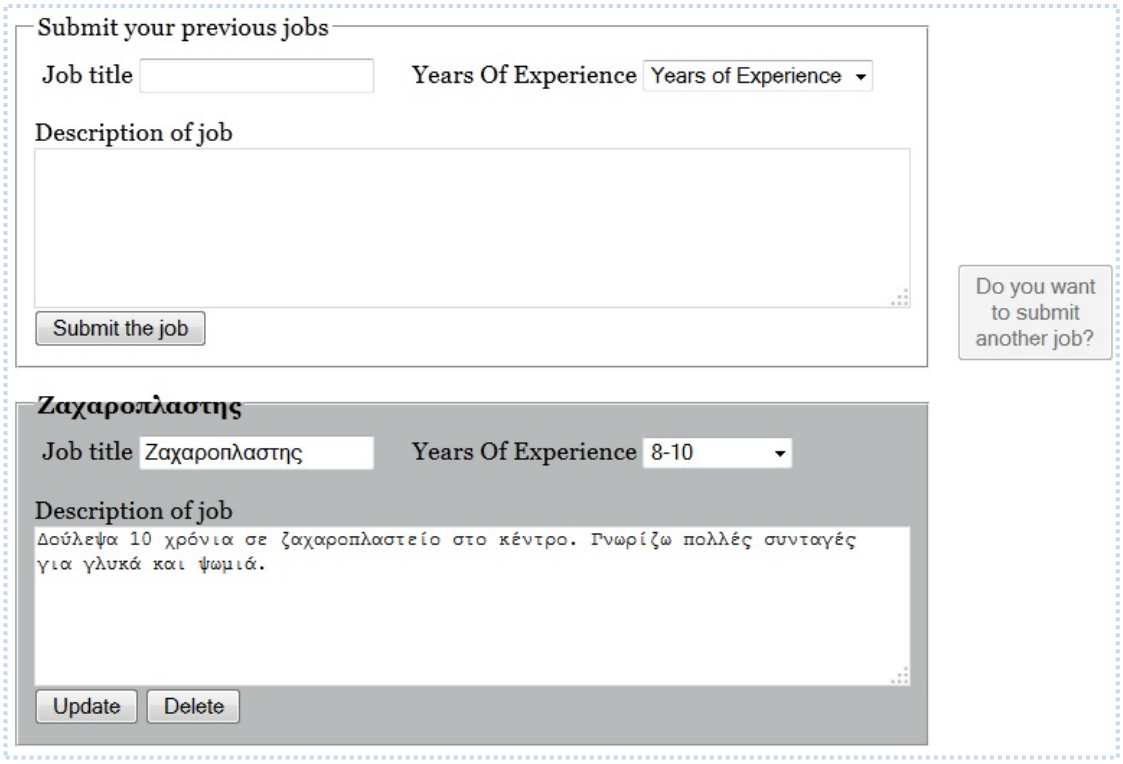
δεξιά υπάρχει ο σύνδεσμος **“log out”** σε περίπτωση που θέλει να αποσυνδεθεί από το σύστημα ο χρήστης.

13.3.2 Αναζητώντας υπαλλήλους

Έχοντας κάνει κάποιος εγγραφή σαν εργοδότης αφού ταυτοποιηθεί από το σύστημα πάνω δεξιά θα εμφανιστούν κάποια διαφορετικά στοιχεία. Όπως ακριβώς και στην προηγούμενη περίπτωση το πρώτο από πάνω αριστερά θα είναι ένα μήνυμα καλωσορίσματος + το όνομα του εργοδότη και από τα δεξιά ο υπερσύνδεσμος **“my profile”** που οδηγεί το χρήστη στη σελίδα που συμπληρώνει τα στοιχεία του. Στην ακριβώς από κάτω γραμμή υπάρχουν οι υπερσύνδεσμοι **“my job posts**”, “post a job” και φυσικά το **“log out”** για αποσύνδεση. Από την στιγμή που θα κάνει ο χρήστης εγγραφή αν προσπαθήσει να πατήσει στο υπερσύνδεσμο **“post a job”** θα μεταφερθεί στο μενού του προφίλ όπου θα του εμφανιστεί μήνυμα ότι δεν μπορεί να δημιουργήσει αγγελία για δουλειά πριν συμπληρώσει τα προσωπικά του στοιχεία. Εδώ τα στοιχεία που συμπληρώνει ο χρήστης είναι λίγο διαφορετικά από την περίπτωση του χρήστη που θα ψάξει για αγγελίες. Εκτός από τα βασικά στοιχεία, υπάρχει ένα μικρό παραθυράκι που εισάγονται τα στοιχεία του ατόμου που θα είναι υπεύθυνος για την επικοινωνία όσον αφορά τις αγγελίες.

13.4 Βιογραφικό του χρήστη (Curriculum Vitae)

Έχοντας κάνει την εγγραφή σαν κάποιος που αναζητά εργασία και έχοντας συμπληρώσει τα προσωπικά στοιχεία υπάρχει πλέον η δυνατότητα να συμπληρώσει κάποιος το βιογραφικό του. Πατώντας στον υπερσύνδεσμο **"Curriculum Vitae"** μεταφέρεται στη σελίδα συμπλήρωσης του βιογραφικού. Υπάρχουν δύο διαφορετικές επιλογές που μπορεί να κάνει κανείς εδώ αλλά αντιθέτως καλύτερο είναι να κάνει και τις δύο. Η πρώτη είναι να ανεβάσει κάποιος στη βάση δεδομένων το δικό του βιογραφικό σε αρχείο **".doc"** ή **".pdf".** Η δεύτερη επιλογή είναι να συμπληρώσει τα στοιχεία του στα πεδία της φόρμας. Στο πρώτο μενού μπορεί να βάλει τις γλώσσες που γνωρίζει. Ύστερα κάποιες ειδικές δυνατότητες και μετά τα διπλώματα οδήγησης που μπορεί να διαθέτει. Το πιο ενδιαφέρον στοιχείο είναι ο τρόπος που συμπληρώνει κάποιος τις προηγούμενες δουλειές που έχει κάνει. Το επόμενο σχήμα δείχνει πώς μπορεί να γίνει αυτό:



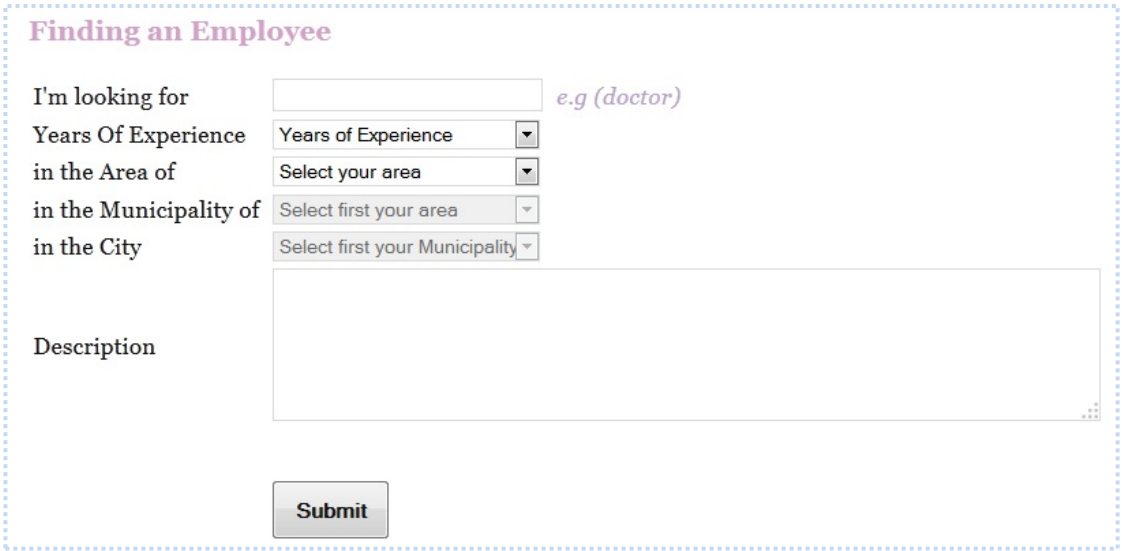
Σχήμα 21: Συμπλήρωση των προηγούμενων δουλειών

Όπως φαίνεται στο Σχήμα 22 υπάρχουν δύο κουτιά. Το πρώτο από πάνω είναι το κουτί που εμφανίζεται όταν θέλουμε να δηλώσουμε μια δουλειά που έχουμε κάνει στο παρελθόν. Εισάγουμε το τίτλο της δουλειάς (εδώ πληκτρολογώντας δύο γράμματα εμφανίζονται όλες οι επιλογές που υπάρχουν ήδη στη βάση μας με ονόματα δουλειών) , ύστερα επιλέγουμε τα χρόνια προϋπηρεσίας και μετά συμπληρώνουμε μια περιγραφή σχετικά με τη δουλειά αυτή. Μόλις συμπληρώσουμε τη δουλειά και πατήσουμε το κουμπί **"Submit the job"** εμφανίζεται ένα μήνυμα από τα δεξιά ότι έγινε επιτυχημένη καταχώριση. Έπειτα το κουμπί από τα δεξιά **"Do you want to submit another job"** γίνεται ενεργό και

μόλις το πατήσουμε η δουλειά που τώρα καταχωρήθηκε πηγαίνει ακριβώς από κάτω και εμφανίζεται ακριβώς όπως ο ζαχαροπλάστης μέσα σε γκρι πλαίσιο. Πλέον μπορούμε να κάνουμε κάποιες αλλαγές και να πατήσουμε **"Update"** είτε μπορούμε να σβήσουμε τη συγκεκριμένη δουλειά πατώντας **"Delete"** όπου θα εμφανιστεί ένα παράθυρο επιβεβαίωσης για τη διαγραφή. Αν κάποιος θέλει να κάνει κατάχρηση της χρησιμοποίησης αυτής και να γεμίσει τη βάση υπάρχει ένα όριο – 30 δουλειές- που αν φτάσει κανείς σ'αυτό υπάρχει μια προειδοποίηση.

13.5 Αγγελίες του εργοδότη

Αφού συμπληρώσει τα στοιχεία του κανείς, μετά έχει τη δυνατότητα να μπορεί να προσθέτει αγγελίες. Κάνοντας κλίκ στον υπερσύνδεσμο **"post a job"** μεταφέρεται σε άλλη σελίδα όπου μπορεί να δημιουργήσει μια καινούργια αγγελία. Παρακάτω υπάρχει το σχήμα προτύπου μιας καινούργιας αγγελίας:



Σχήμα 22: Πρότυπο δημοσίευσης μιάς καινούργια αγγελίας

Βλέπουμε ότι το πρώτο πεδίο από πάνω είναι για τη συμπλήρωση του τίτλου της δουλειάς : π.χ. Ηλεκτρολόγος. Από κάτω ακριβώς είναι το πεδίο που καθορίζει πόσα χρόνια χρειάζεται να έχει προϋπηρεσία για την δουλειά αυτή. Κάτω είναι το πεδίο που καθορίζει σε ποια περιοχή ο ενδιαφερόμενος θα πρέπει να εργαστεί, ακολουθεί η επιλογή του δήμου και πιο κάτω η επιλογή πόλης/περιοχής/ τοποθεσίας. Τουλάχιστον η περιοχή πρέπει να επιλεγεί αλλιώς θα εμφανιστεί το ανάλογο μήνυμα για την επιλογή. Το τελευταίο πεδίο είναι μια περιγραφή όπου ο εργοδότης μπορεί να δώσει κάποιες λεπτομέρειες σχετικά με τη δουλειά αυτή.

Κάνοντας κλίκ στον υπερσύνδεσμο **"my job posts"** μεταφέρεται κανείς σε άλλη σελίδα όπου μπορεί να δει μέσα από ένα πίνακα όλες τις αγγελίες που έχει βάλει. Κάθε αγγελία έχει τρία πεδία : τον τίτλο της δουλειάς του υπαλλήλου, το μέρος που ζητείται και την ημερομηνία δημοσίευσής της . Αν πατήσει κανείς πάνω σε κάποια εγγραφή δηλ. σε κάποια από αυτά τα πεδία θα μεταφερθεί σε άλλη

σελίδα όπου θα μπορεί να κάνει όποιες αλλαγές θέλει σχετικά με την αγγελία αυτή ή να την σβήσει κιόλας.

Κεφάλαιο 14 : Μέθοδοι Αναζήτησης με το Lucene

Η εφαρμογή του JobFinder προσφέρει στο σύνολο 4 μεθόδους αναζήτησης. Οι περισσότερες έχουν κάποια κοινά χαρακτηριστικά σαφώς αλλά υπάρχουν κάποιες σημαντικές διαφορές.

14.1 Ενσωματωμένη μέθοδος αναζήτησης (Central Search)

Η ενσωματωμένη μέθοδος αναζήτησης βρίσκεται καρφιτσωμένη πάνω στη επάνω σταθερή μπάρα της εφαρμογής. Οποιαδήποτε στιγμή και οπουδήποτε να μετακινηθούμε μπορούμε να τη χρησιμοποιήσουμε για να κάνουμε άμεση αναζήτηση. Αποτελείται από τις δύο μπάρες δηλ. τον τίτλο της δουλειάς και το όνομα της τοποθεσίας.

14.1.1 Τίτλος της δουλειάς

Μόλις αρχίσουμε να πληκτρολογούμε μέσα στο πεδίο τίτλου ύστερα από δύο γράμματα θα εμφανιστεί μια λίστα ακριβώς από κάτω με διαθέσιμους τίτλους της δουλειάς που περιέχουν τα γράμματά αυτά. Μπορούμε να επιλέξουμε κάποιο από τα επαγγέλματα που θα εμφανιστούν και αυτόματα αυτό θα μπει στο πεδίο. Εάν θέλουμε να βάλουμε περισσότερα επαγγέλματα για αναζήτηση, μπορούμε να χρησιμοποιήσουμε το κόμμα για τον διαχωρισμό των τίτλων. Αξίζει να σημειωθεί εδώ ότι δεν έχει σημασία πώς ακριβώς θα αναζητήσουμε ένα τίτλο δουλειάς, π.χ. είτε βάλουμε «ηλεκτρολόγος» είτε «ΗλεκΤρολΟγοΣ» είτε «Ηλέκτρόλογός» το αποτέλεσμα θα είναι ακριβώς το ίδιο. Η διευκόλυνση αυτή έγινε για ευνόητους λόγους.

Εάν πληκτρολογήσουμε μόνο στο πεδίο τίτλου χωρίς να βάλουμε τίποτα στο πεδίο της τοποθεσίας δεν θα αναζητηθεί καμία τοποθεσία. Τα πεδία των εγγράφων του Lucene μέσα από τα οποία θα γίνει η αναζήτηση είναι ο τίτλος δουλειάς και η περιγραφή που δίνει ο καθένας για τη δουλειά όσον αφορά τα **βιογραφικά** και αντίστοιχα ο τίτλος της δουλειάς και η περιγραφή που έδωσε ένας εργοδότης σε μια **αγγελία**. Δηλαδή, εάν συμπλήρωσε κάποιος τη λέξη «Ηλεκτρονικός» θα αναζητηθούν έγγραφα που περιέχουν τη λέξη ηλεκτρονικός στο πεδίο «τίτλος δουλειάς» και στο πεδίο «περιγραφή». Και οι προηγούμενες δουλειές του καθενός και οι αγγελίες των εργοδοτών χαρακτηρίζονται από ίδιο όνομα πεδίου για καλύτερη αναζήτηση.

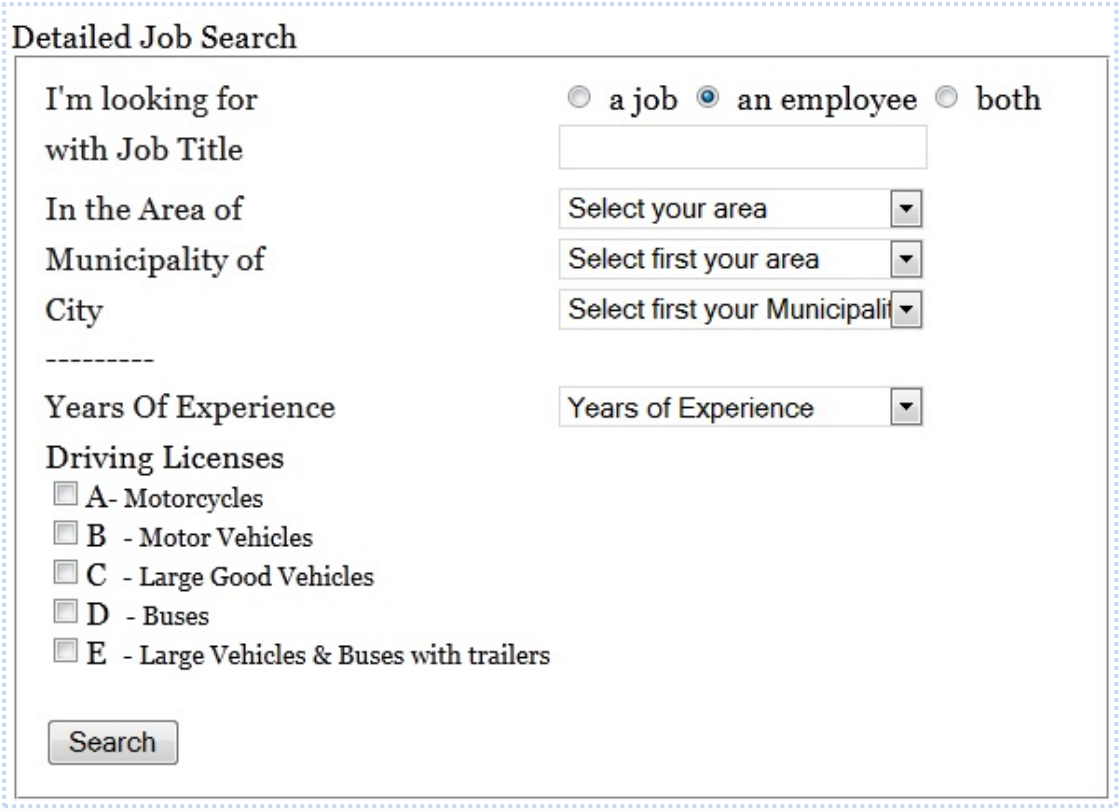
14.1.2 Όνομα τοποθεσίας

Όπως και στο πεδίο τίτλου δουλειάς έτσι και εδώ ύστερα από δύο γράμματα που θα πληκτρολογήσουμε εμφανίζεται από κάτω το μενού με τις επιλογές τοποθεσιών σύμφωνα με τα γράμματά μας. Και εδώ ισχύει το ίδιο όπου μπορούμε να πληκτρολογήσουμε παραπάνω από μια τοποθεσία βάζοντας απλά κόμμα

μεταξύ τοποθεσιών. Εάν συμπληρωθούν και τα δύο πεδία τότε θα μας δώσουν τους συνδυασμούς από τις δουλειές και τις τοποθεσίες που θα πληκτρολογήσουμε.

14.2 Λεπτομερής Αναζήτηση (Detailed Search)

Στο επόμενο σχήμα φαίνεται η διεπαφή χρήστη της λεπτομερής αναζήτησης:

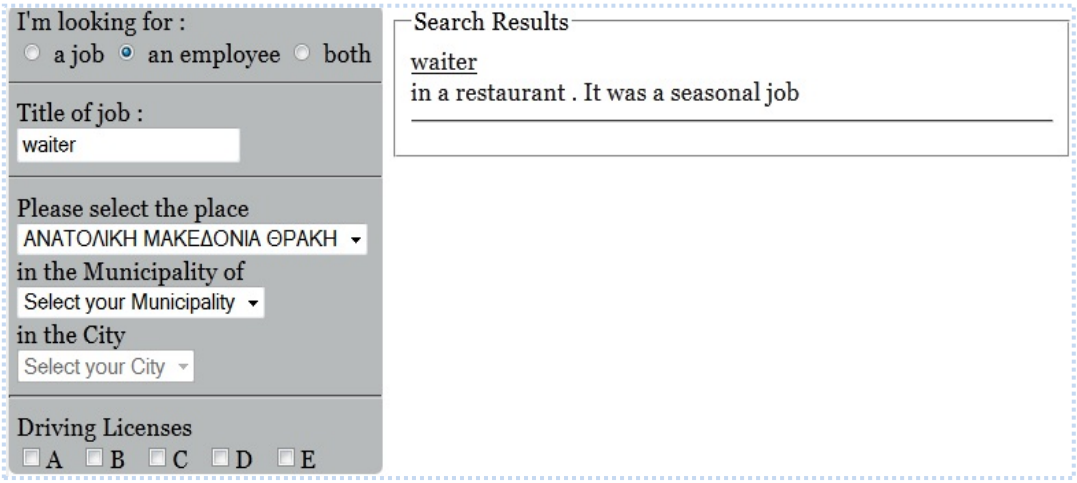


Σχήμα 23: Η διεπαφή λεπτομερής αναζήτησης

Όπως βλέπουμε στο σχήμα η διεπαφή είναι πολύ απλή. Η πρώτη γραμμή καθορίζει τι ψάχνουμε ακριβώς δηλ. είτε δουλειά, είτε εργαζόμενο είτε και τα δύο. Το αμέσως επόμενο πεδίο καθορίζει το **«τίτλο δουλειάς»**. Λειτουργεί ακριβώς έτσι όπως και το πεδίο της ενσωματωμένης αναζήτησης. Ύστερα ακολουθεί η **«περιοχή»** που επιλέγουμε. Αν διαλέξουμε κάποια περιοχή πλέον έχουμε τη δυνατότητα να επιλέξουμε και το **«δήμο»**. Αν διαλέξουμε δήμο μπορούμε να επιλέξουμε και την **«πόλη / περιοχή / τοποθεσία»**. Μετά ακριβώς υπάρχει το πεδίο **«χρόνια προϋπηρεσίας»** όπου μπορεί να διαλέξει ένας εργοδότης τα χρόνια υπηρεσίας που θέλει ο υπάλληλος να έχει για τη συγκεκριμένη δουλειά. Το τελευταίο που βλέπουμε είναι τα **«διπλώματα»** που θα θελήσει ο εργοδότης να έχει ένας υπάλληλος. Η επιλογή αυτή είναι διαθέσιμη μόνο όταν γίνεται αναζήτηση για υπάλληλο.

14.3 Ζωντανή Αναζήτηση (Live Search)

Μπορούμε να δούμε την ζωντανή αναζήτηση στο επόμενο σχήμα:

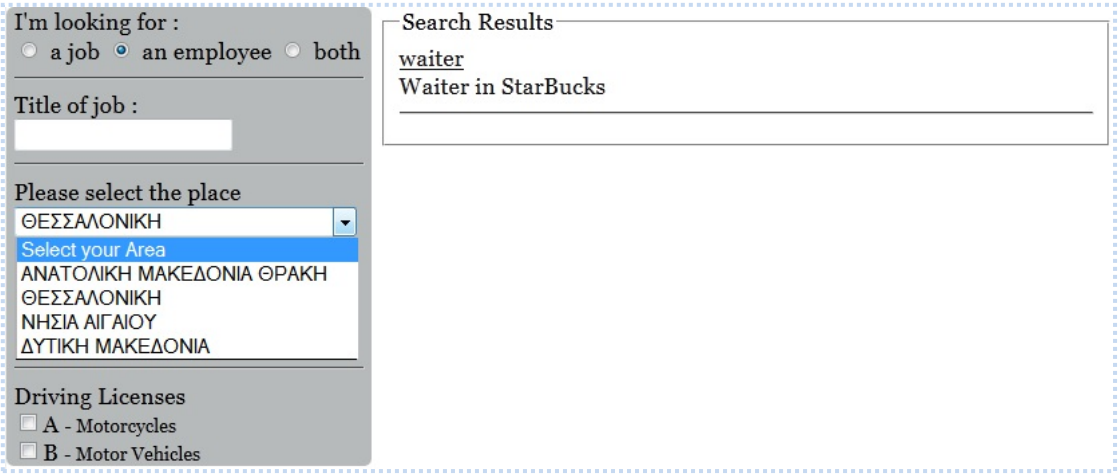


Σχήμα 24: Διεπαφή ζωντανής αναζήτησης

Όπως φαίνεται η διεπαφή αυτή διαφέρει από την λεπτομερή αναζήτηση. Ο σχεδιασμός της είναι πιο μινιμαλιστικός και η σημαντική διαφορά είναι ότι ανάλογα με τις ρυθμίσεις που κάνουμε εμφανίζονται ακριβώς από τα δεξιά τα ανάλογα αποτελέσματα. Από τα αριστερά βλέπουμε την διεπαφή η οποία έχει παρόμοια πεδία με την λεπτομερή αναζήτηση. Αυτό που χαρακτηρίζει αυτή την διεπαφή είναι ότι πλέον ο χρήστης δεν πατάει σε κάποιο κουμπί για να στείλει τα δεδομένα. Υπάρχει μια ζωντανή αλληλεπίδραση με το Lucene σε κάθε αλλαγή και κάθε επιλογή που κάνουμε. Επίσης όσον αφορά το κώδικα η σημαντικότερη διαφορά είναι ότι εδώ τα δεδομένα αποστέλλονται και παραλαμβάνονται μαζί με τα αποτελέσματα μέσω αιτήσεων AJAX με τη βοήθεια του jQuery, ενώ με την λεπτομερή αναζήτηση θα αποστέλλονταν με αιτήσεις POST του πρωτοκόλλου HTTP και θα γινόταν αλλαγή και φόρτωση καινούργιας σελίδας.

14.4 Πολύπλευρη Αναζήτηση (Faceted Search)

Η πολύπλευρη αναζήτηση είναι η πιο δυναμική αναζήτηση της εφαρμογής μας. Συνδυάζει όλα τα χαρακτηριστικά της ζωντανής αναζήτησης μόνο που εδώ δεν υπάρχουν στιγμές που επιστρέφονται μηδενικά αποτελέσματα. Αυτό γίνεται γιατί οι επιλογές που μας εμφανίζονται στα πεδία, όλες περιλαμβάνουν έστω και κάποια εγγραφή. Έτσι η δυνατότητα αναζήτησης γίνεται ακόμη πιο ευχάριστη υπόθεση. Παρακάτω βλέπουμε την πολύπλευρη αναζήτηση:



Σχήμα 25: Πολύπλευρη αναζήτηση

Χαρακτηριστικά φαίνεται από το σχήμα ότι π.χ. οι περιοχές που μπορούμε να επιλέξουμε είναι 4 και όχι όλες γιατί μόνο σ'αυτές τις περιοχές υπάρχουν εργαζόμενοι. Επίσης φαίνεται ότι αφού επιλέξαμε Θεσσαλονίκη για περιοχή υπάρχουν μόνο 2 διπλώματα τα οποία διαθέτουν αυτοί που μένουν στη περιοχή αυτή (A,B).

ΒΙΒΛΙΟΓΡΑΦΙΑ

Βιβλία:

1. McCandless, M. et al. (2010), Lucene in Action , second edition, Manning
2. Freeman, A. and Sanderson, S. (2011), Pro ASP.NET MVC 3 Framework ,3rd Edition, Apress.
3. Galloway, J. et al., (2011) , Professional ASP.NET MVC 3 ,Wrox
4. Otero, C. (2012), Professional jQuery, Wrox
5. Schenker, G. and Cure, A. (2011) , Nhibernate 3 Beginners Guide, Packt Publishing
6. Tobin, H. et al., (2009), Nhibernate in Action, Manning
7. Nagel, C. et al., (2010), C# 4 and .NET 4, Wrox

Αναφορές στον Ιστό:

1. http://lucene.apache.org
2. http://en.wikipedia.org/wiki/ASP.NET_MVC_Framework
3. http://en.wikipedia.org/wiki/C_Sharp_%28programming_language%29
4. http://en.wikipedia.org/wiki/Microsoft_Visual_Studio
5. http://en.wikipedia.org/wiki/JQuery
6. http://en.wikipedia.org/wiki/Mysql
7. http://en.wikipedia.org/wiki/NHibernate
8. http://nhforge.org/
9. http://en.wikipedia.org/wiki/Dependency_injection
10. http://www.ninject.org/
11. http://en.wikipedia.org/wiki/Lucene