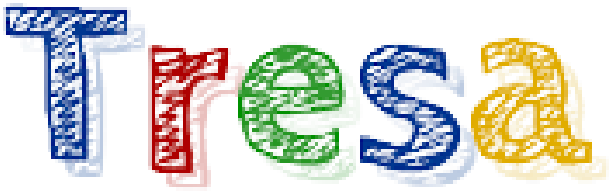
2021
2022

Μηχανή Αναζήτησης TRESA



Ιωάννης Παπαχρήστου
AM:2022201800146
Εμμανουήλ Καραπιπεράκης
AM:2022201800075

Υλοποίηση

Η υλοποίηση της εργασίας ακολούθησε την σειρά των ερωτημάτων της εκφώνησης.

Προεπεξεργασία

Καταρχάς εκκινήσαμε, με την υλοποίηση της προεπεξεργασίας των άρθρων και της αναζήτησης του χρήστη. Προκειμένου να το επιτύχουμε αυτό, χρησιμοποιήσαμε τα έτοιμα εργαλεία του Lucene. Συγκεκριμένα, πρώτη μας κίνηση ήταν η αφαίρεση των stopwords. Η υλοποίηση για αυτό ήταν σχετικά απλή. Κατά την εκκίνηση της εφαρμογής γίνεται η ευρετηρίαση(start() στο luceneTester.java) οπού ανοίγουμε επαναληπτικά τα αρχεία που υπάρχουν στο φάκελο data, αποθηκεύουμε σε 4 μεταβλητές το περιεχόμενο των πεδίων και κάνουμε την κατάλληλη προ επεξεργασία ανάλογα το πεδίο. Αν ο χρήστης κλείσει την εφαρμογή διαγράφουμε το αντεστραμμένο ευρετήριο (περιεχόμενο στο φάκελο Index). Περιληπτικά, σε έναν πίνακα τοποθετήθηκαν όλα τα stopwords και στην συνέχεια, μέσω του φίλτρου StopFilter προστέθηκαν σε ένα tokenstream, το οποίο και στην συνέχεια μέσω μιας επανάληψης μπορεί και ανακαλείται.

```
final List<String> stop_Words = Arrays.asList("a", "an", "and", "are", "as", "at", "be", "but", "by",
    "for", "if", "in", "into", "is", "it",
    "no", "not", "of", "on", "or", "such",
    "that", "the", "their", "then", "there", "these",
    "they", "this", "to", "was", "will", "with");
final CharArraySet stopSet = new CharArraySet(stop_Words, true);
tokenStream = new StopFilter(tokenStream, stopSet);



while (tokenStream.incrementToken()) {
    int startOffset = offsetAttribute.startOffset();
    int endOffset = offsetAttribute.endOffset();
    String term = charTermAttribute.toString();

    for(int i=0;i<term.length();i++)
    {
        stem.add(term.charAt(i));
    }
    stem.stem();
    term = stem.toString();
    System.out.println(term);
    query.add(new Term("body", term));
}
```

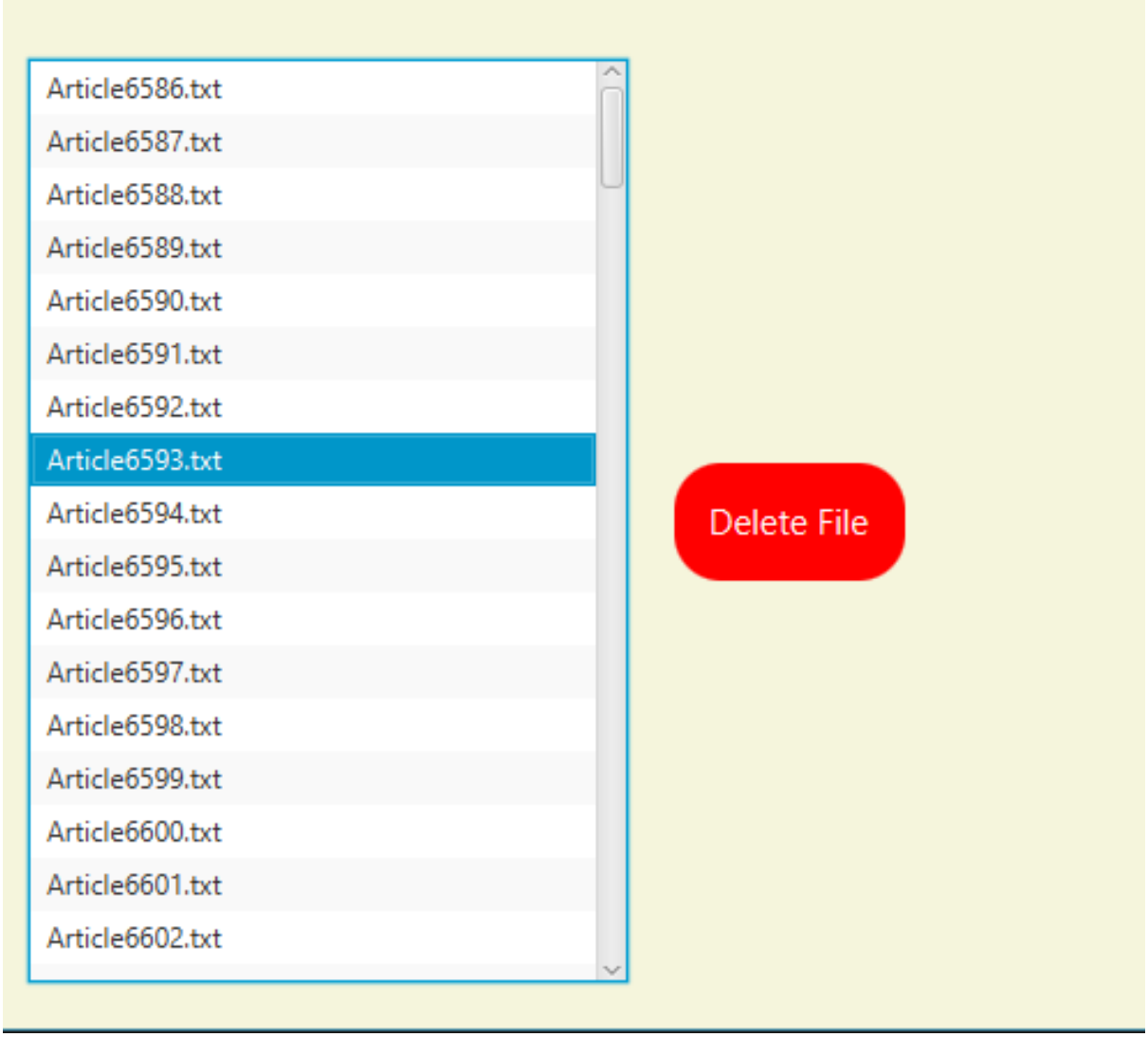
Το επόμενο βήμα της προεπεξεργασίας των κειμένων, απαιτούσε την αποκοπή των καταλήξεων των λέξεων (Stemming), δυνατότητα που διευκολύνει την μετέπειτα φάση της αναζήτησης. Η λειτουργία αυτή, επιτυγχάνεται μέσω μιας κλάσης που βρήκαμε στο διαδίκτυο και έχει όνομα PorterStemmer. Java (https://github.com/bean5/Java-Porter-Stemmer/blob/master/src/PorterStemmer.java).

Εισαγωγή/Διαγραφή αρχείων

Με την ολοκλήρωση της επεξεργασίας των κειμένων η ομάδα μας συνέχισε στο ερώτημα της εισαγωγής και διαγραφής αρχείων. Στο σημείο αυτό κρίθηκε αναγκαίο να αναπτυχθεί παράλληλα, ένα γραφικό περιβάλλον το οποίο θα διευκόλυνε στην αποσφαλμάτωση του προγράμματος και παράλληλα θα πρόσφερε στον χρήστη ένα εύκολο τρόπο χρήσης της μηχανής αναζήτησης.

Η ομάδα μας προκειμένου να φτιάξει το γραφικό περιβάλλον χρησιμοποίησε το εργαλείο maven και την γλώσσα JavaFX. Επειδή, οι γνώσεις μας δεν ήταν επαρκείς προκειμένου να φτιάξουμε ένα νέο project από το μηδέν που να τρέχει παράλληλα Lucene και JavaFX, πήραμε ένα έτοιμο project που συνδύαζε τα δύο εργαλεία από το GitHub και το αδειάσαμε. Για αυτόν τον λόγο το project μας περιέχει και ένα αρχείο με το όνομα LICENSE.

Για την εισαγωγή των αρχείων το προγραμματιστικό κομμάτι αποτελούσε κυρίως εντολές Java και JavaFX. Στο γραφικό περιβάλλον, ετοιμάσαμε έναν πίνακα (ListView), το οποίο περιέχει μέσα αρχεία που υπάρχουν ήδη στον φάκελο του project, ώστε ο χρήστης να μπορεί να βλέπει γρήγορα και εύκολα όλα τα αρχεία που τον ενδιαφέρουν. Στην περίπτωση της διαγραφής, αρκεί το πάτημα στο επιθυμητό αρχείο προς διαγραφή και μετά η οριστική αφαίρεση του από το project μέσω του κουμπιού Delete



Για την εισαγωγή των αρχείων, χρησιμοποιήσαμε την έτοιμη λειτουργία του JavaFX FileChooser, που ανοίγει ένα dialog επιλογής αρχείου. Στο πρόγραμμα, υπάρχει περιορισμός εισαγωγής μόνο .txt αρχείων. Η διαδικασία αυτή εκκινεί μετά το πάτημα του κουμπιού Insert File.

```
FileChooser fc = new FileChooser();
fc.setInitialDirectory(new File ("Data//"));
fc.getExtensionFilters().addAll(new ExtensionFilter("txt files", "*.txt"));
File selectedFile = fc.showOpenDialog(null);
```

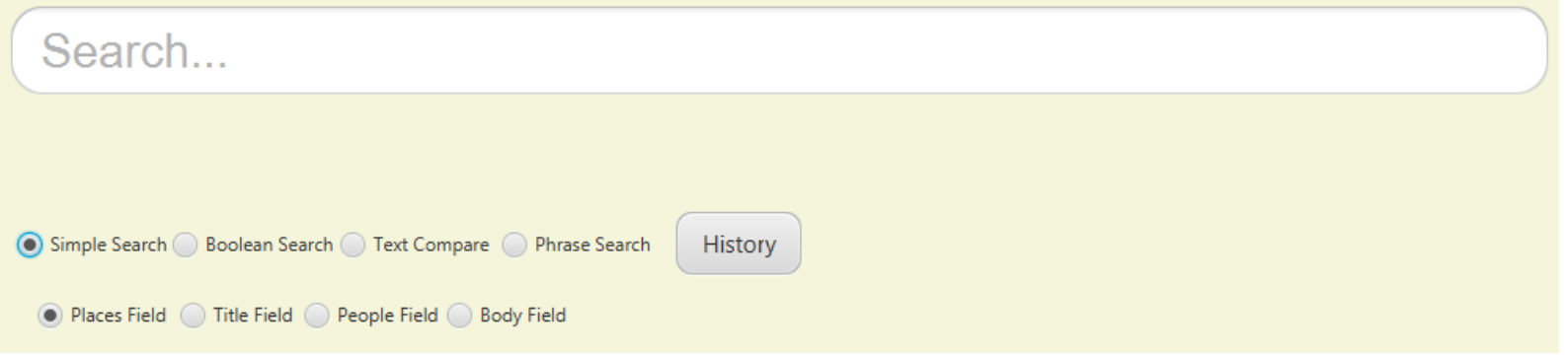
Το αρχείο μετά την εισαγωγή του εμφανίζεται στον πίνακα των αρχείων. Στην υλοποίησή μας δεν γίνεται έλεγχος ήδη υπάρχοντος αρχείου (κοινό όνομα) ή ίδιου format αρχείων (ίδιο περιεχόμενο).

Κατά την εισαγωγή ή διαγραφή κάποιου αρχείου για να ενημερώσουμε καταλληλά το αντεστραμμένο ευρετήριο διαγράφουμε το περιεχόμενο του φακέλου Index και ξανακαλούμε την μέθοδο start() για να γίνει εκ νέου ευρετηρίαση με τα ενημερωμένα αρχεία.

Έπειτα από την επεξεργασία των αρχείων, συνεχίσαμε υλοποιώντας τις αναζητήσεις που πρέπει να τελεί η μηχανή μας.

Simple Search

Εκκινώντας από το Simple Search καταρχάς, στο γραφικό μας περιβάλλον εισαγάγαμε ένα TextField εισαγωγής των query και επιλογής του κατάλληλου field.



Όταν το query εισαχθεί και επιλεγεί το κατάλληλο field, η υλοποίηση της λειτουργίας μεταβιβάζεται στο Lucene. Στο Lucene, η διαδικασία εύρεσης των αποτελεσμάτων εκκινεί με την επεξεργασία του query στην κλάση LuceneTester.java που δόθηκε από τον χρήστη ανάλογα το field στο οποίο ανήκει του query που επιθυμεί (Για παράδειγμα πέρα από την αφαίρεση των Stop Words και του Stemming, αν το πεδίο αναζήτησης είναι τίτλος, πρέπει να αφαιρεθούν πιθανές αλλαγές γραμμών ή tabs). Στην συνέχεια, το query μεταβιβάζεται στην κλάση Searcher.java, η οποία δέχεται το επιλεγμένο αναζήτησης μαζί με το query και μέσω της συνάρτησης indexSearcher.search εκτελεί την λειτουργία της αναζήτησης στα αρχεία. Στο simple search στο

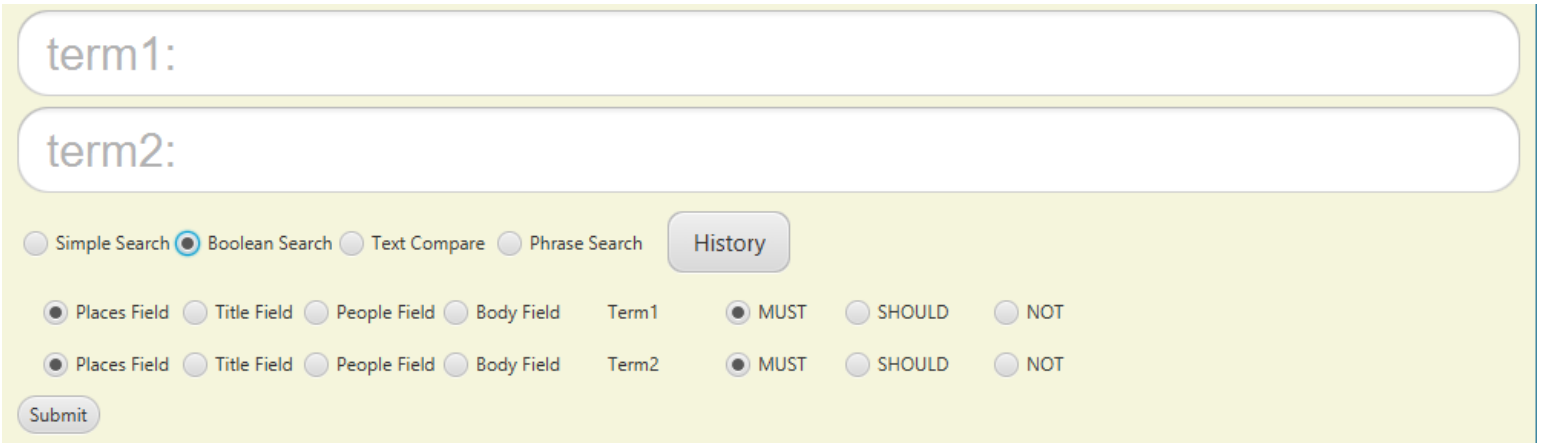
πεδίο του τίτλου, η φράση που τον περιγράφει πρέπει να δοθεί ολόκληρη προκειμένου να βρεθεί κάποιο αποτέλεσμα.

```
public TopDocs search(String searchQuery) throws IOException,ParseException
{

    query = queryParser.parse(searchQuery);
    //System.out.println(query);
    System.out.println("query: "+ query.toString());
    return indexSearcher.search(query, LuceneConstants.MAX_SEARCH);
}
```

Boolean Search

Για το Boolean Search η υλοποίηση αν και πιο πολύπλοκη, μοιάζει στην βάση της με την απλή αναζήτηση. Καταρχάς στο γραφικό περιβάλλον υπάρχουν πλέον δύο TextField και σειρές κουμπιών επιλογής πεδίων και λογικής πράξης.



Χρησιμοποιήθηκαν οι ίδιες κλάσεις με την απλή αναζήτηση, όμως αυτή τη φορά υπερφορτώσαμε τις μεθόδους αναζήτησης, ώστε να δέχονται δύο queries και fields, αλλά και την μεταξύ τους λογική πράξη. Αυτή τη φορά δημιουργήσαμε ένα boolean build query για την διαχείριση όλων αυτών των

```
BooleanQuery.Builder query = new BooleanQuery.Builder();
```

```
public TopDocs search(Builder query)throws IOException, ParseException{
    return indexSearcher.search(query.build(), LuceneConstants.MAX_SEARCH);
}
```

δεδομένων και στην συνέχεια το στείλαμε με την indexSearcher.search, όπως και στο προηγούμενο ερώτημα.

Περιγραφή λογικών πράξεων:

πρέπει οπωσδήποτε να υπάρχει το term;(MUST)

είναι επιθυμητό να υπάρχει το term;(SHOULD)

δεν πρέπει να υπάρχει το term;(NOT)

Έστω πως ψάχνουμε άρθα που μιλάνε για τράπεζες της Αμερικής:

Τότε ο χρήστης θα γράψει στο πρώτο search query "banks",θα επιλέξει το πεδίο "body" και θα επιλέξει MUST στο δεύτερο search query "Usa",θα επιλέξει το πεδίο "places" και θα επιλέξει MUST.

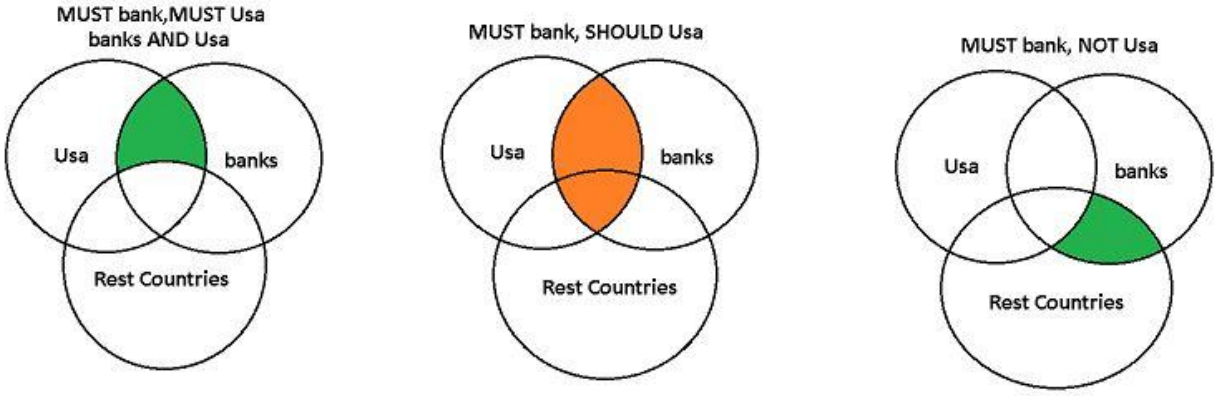
Αν ο χρήστης πατήσει submit, θα επιστρέψουν άρθρα που μιλάνε για τράπεζες και έχουν ως χωρά την Αμερική. Έστω πως υπάρχουν 11 τέτοια άρθρα.

Αν ο χρήστης επιλέξει should για το term usa, τοτε θα επιστρέψουν άρθρα που μιλάνε απαραίτητα για τράπεζες, αλλά η χωρά δεν είναι υποχρεωτικά η Αμερική, δηλαδή θα έχουμε αποτελέσματα για όλες τις χώρες. Έστω πως είναι 35

Αν ο χρήστης όμως επιλέξει NOT για τη χωρά usa τότε θα επιστρέψουν άρθρα που μιλάνε για Τράπεζες που δεν βρίσκονται όμως στην Αμερική.

Το αποτέλεσμα θα είναι 35-11=24

Ακολουθεί παράδειγμα με διάγραμμα Vein, οπού έστω πως μέσα στους κύκλους υπάρχουν διαφορά άρθρα



Phrase Search

Η υλοποίηση του Phrase Search είναι παρόμοια με τις προηγούμενες. Στο phrase μας, αφαιρούμε τα Stop Words και κάνουμε Stemming και στην συνέχεια χρησιμοποιούμε την εντολή του Lucene:

```
PhraseQuery.Builder query = new PhraseQuery.Builder();
```

Η οποία στέλνει ένα Phrase Query που φτιάξαμε με αυτές τις λέξεις, το οποίο και αναζητήσαμε στην κλάση Searcher για να εκτελέσει την αναζήτηση. Η αναζήτηση αναφέρεται αποκλειστικά στο body field.

```
public TopDocs Phsearch(org.apache.lucene.search.PhraseQuery.Builder query3)throws IOException, ParseException{
    return indexSearcher.search(query3.build(), LuceneConstants.MAX_SEARCH);
}
```

Όπως ζητείται και από την εκφώνηση, σε κάθε εκτέλεση αναζήτησης εμφανίζονται συγκεκριμένες πληροφορίες. Στο πρόγραμμα μας δημιουργήσαμε ένα dialog, το οποίο κάνει pop-up κάθε φορά που ο χρήστης στέλνει το query του. Σε αυτό το dialog υπάρχουν πίνακες με το όνομα των αρχείων , το σκορ, το περιεχόμενο των αρχείων και το κείμενο που περιβάλλει το query.

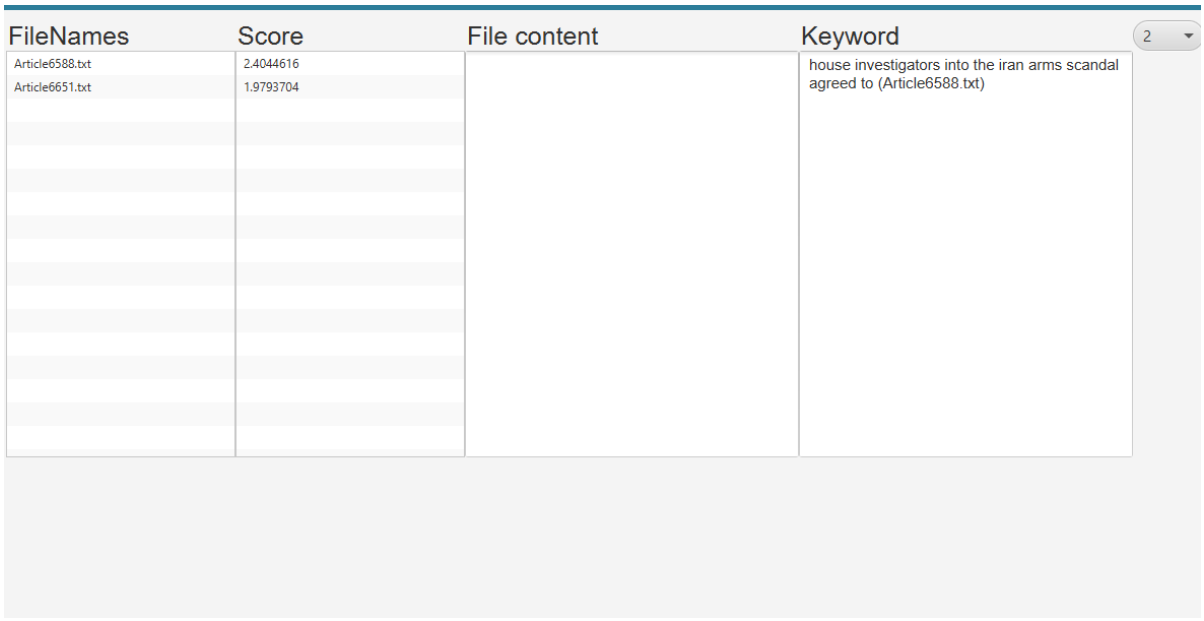
Μετά από κάθε αναζήτηση αποθηκεύαμε σε μια λίστα τα αρχεία που βρέθηκε το keyword μας και το score σχετικότητας του κάθε αρχείου, το οποίο είναι σκορ cosine similarity.

Στο τέλος της αναζήτησης εξετάζαμε το μέγεθος μιας από τις 2 λίστες (δεν έχει σημασία η σειρά). Εάν το μέγεθος της ήταν 0, σημαίνει πως δεν βρέθηκαν αποτελέσματα και εμφανίζαμε ανάλογο alert.

Εναλλακτικά καλούσαμε την κατάλληλη κλάση ανάλογα το είδος της αναζήτησης που είχε ως σκοπό την εμφάνιση των αποτελεσμάτων η οποία εμφάνιζε:

Ένα listview με τα ονόματα των αρχείων

1) Ένα παράλληλο listview με το σκορ σχετικότητας
2) Ένα κενό text area το οποίο είναι υπεύθυνο για την προβολή του περιεχομένου ενός κειμέν ου εάν γίνει κλικ σε αυτό(1ο listview)
3) Ένα text area το οποίο περιέχει ένα απόσπασμα μέσα από το κείμενο το οποίο περιέχει το keyword ή τα keywords που είχαν δοθεί προηγουμένως(μόνο την πρώτη εμφάνιση του)
4) Ένα text area το οποίο περιέχει ένα απόσπασμα μέσα από το κείμενο το οποίο περιέχει το keyword ή τα keywords που είχαν δοθεί προηγουμένως (μόνο την πρώτη εμφάνιση του).



Επίσης μέσω ενός combobox και της κλάσης results, ο χρήστης μπορεί να επιλέξει τα top κ αποτελέσματα που τον ενδιαφέρουν.

<BODY>
try to formulate a unified position
ahead of possible
future negotiations reuter
</BODY>

keyword = "position"

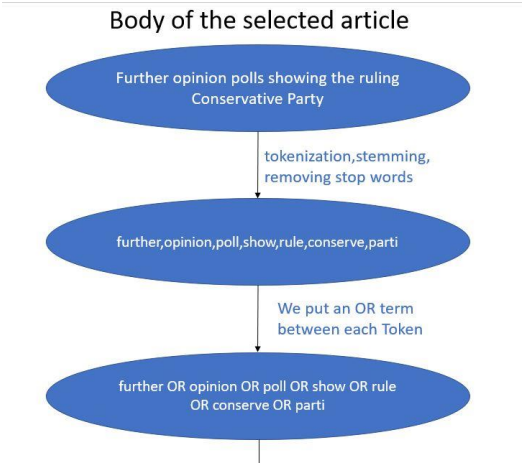
obv1 = [try, to, formulate, a, unified, position, ahead, of, possible, future, negotiations, reuter]

obv3 = [try, to, formul, a, unifi, posit, ahead, of, possibl, futur, negoti, reuter]

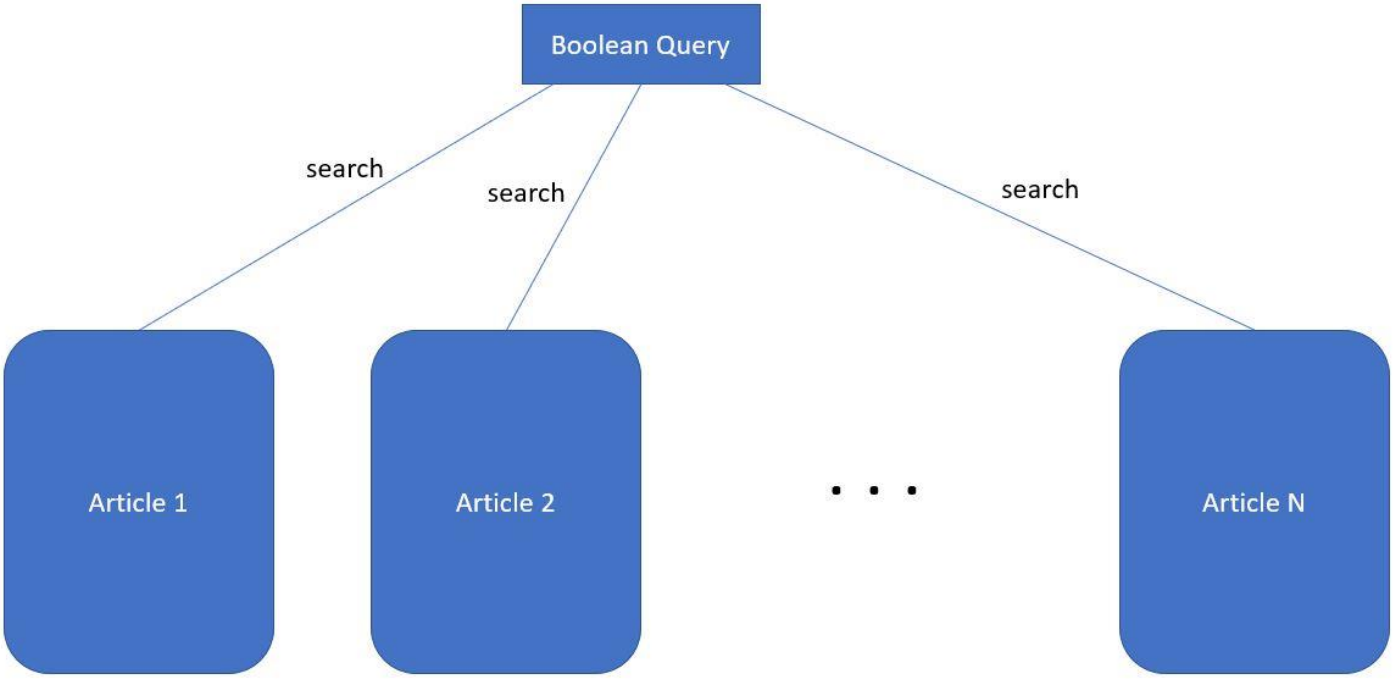
obv1[5] == keyword
ta2 = obv1[1] + obv1[2] + obv1[3] + obv1[4] + obv1[5] + obv1[6] + obv1[7] + obv1[8] + obv1[9]
ta2 = *to formulate a unified position ahead of possible future*

Text Compare

Ο χρήστης επιλεγεί ένα κείμενο και επιστρέφεται μια λίστα με τα πιο σχετικά κείμενα, ταξινομημένα. Για να το πέτυχουμε αυτό, από το κείμενο που επέλεξε ο χρήστης παίρναμε επαναληπτικά κάθε term και φτιάξαμε ένα boolean query προσθέτοντας κάθε φορά το λογικό τελεστή OR αναμεσά στα terms. Μετά κάνουμε μια αναζήτηση όπως έγινε στο boolean search και έτσι παίρνουμε τα πιο σχετικά άρθρα.



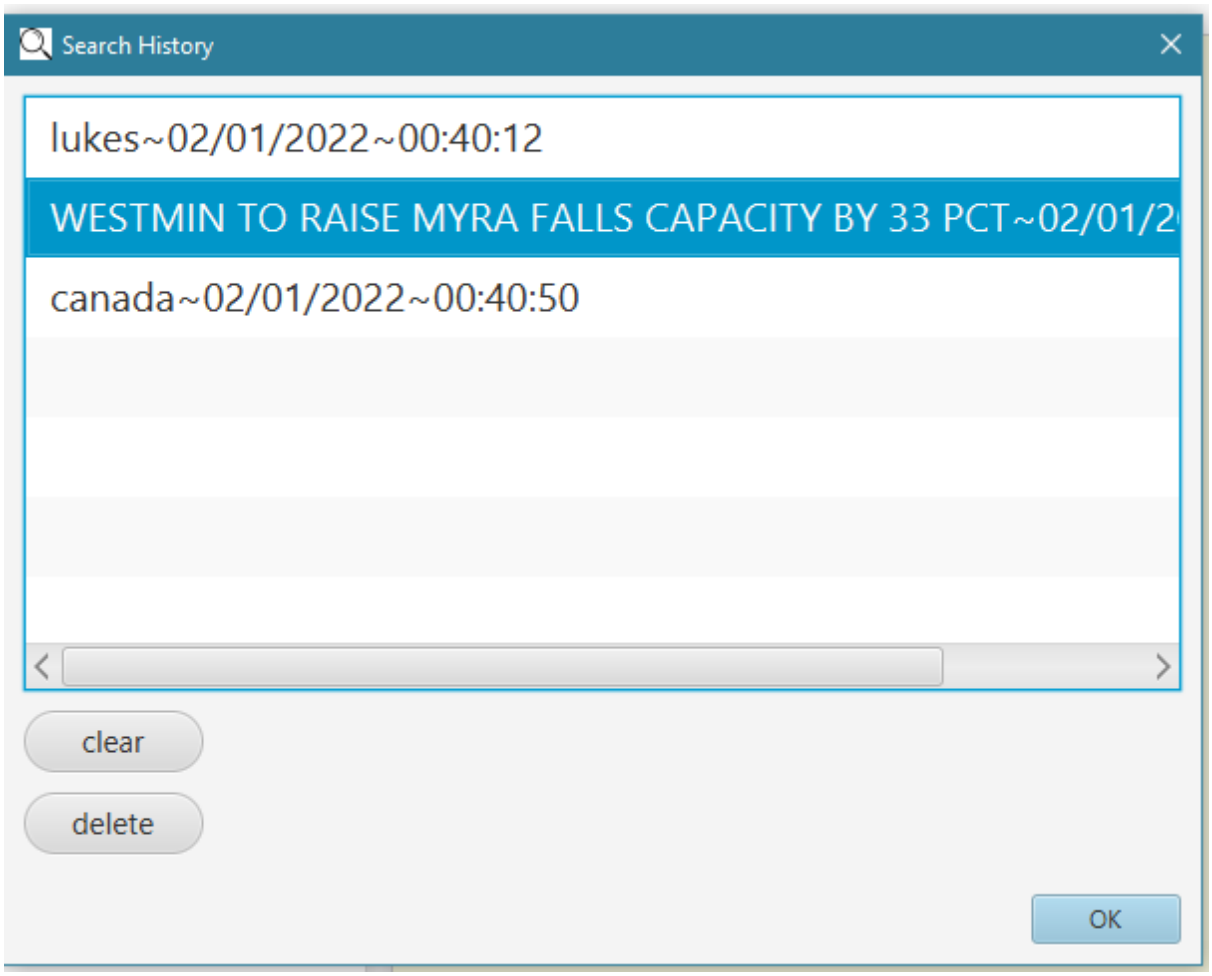
Αναζητούμε το Boolean Query που δημιουργήσαμε προηγουμένως σε όλα τα άρθρα, επιστρέφοντας το σκορ σχετικότητας, το οποίο θεωρητικά είναι ένας αριθμός που εκφράζει το πόσο σχετικό είναι το κείμενο που επιστρέφεται σε σχέση με αυτό που επέλεξε ο χρήστης.



Bonus

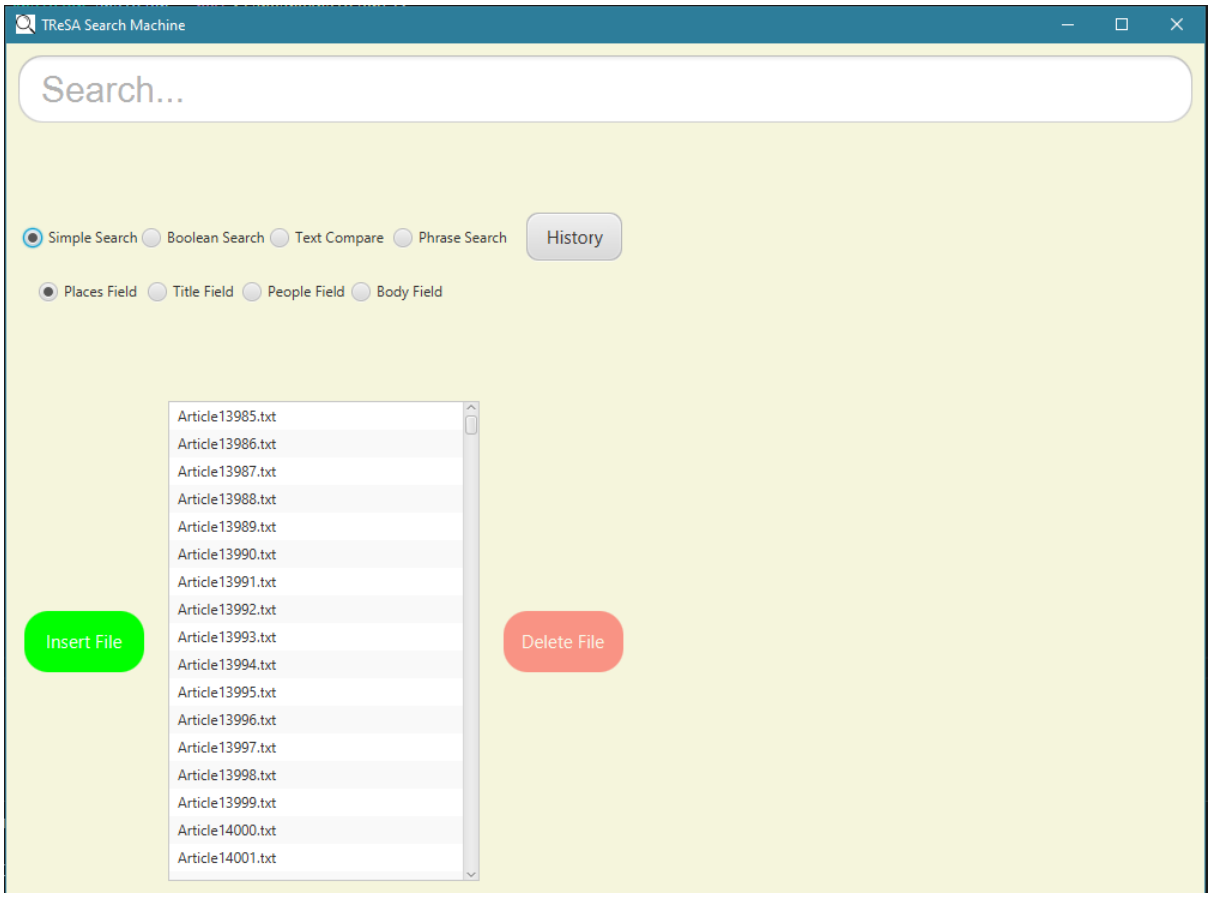
Η ομάδα μας ως υλοποίηση bonus επέλεξε να δημιουργήσει μια λειτουργία καταγραφής ιστορικού αναζήτησης. Για την υλοποίηση αυτή κατασκευάσαμε ένα αρχείο history.txt, το οποίο δέχεται το searchQuery που εισάγει ο χρήστης και καταγράφει την ημερομηνία εισαγωγής. Στην συνέχεια, μέσω ενός κουμπιού στο gui ανοίγει ένα dialog που περιέχει ένα listView , το οποίο εμφανίζει τα περιεχόμενα του αρχείου ιστορικού. Ο χρήστης έχει την δυνατότητα να επιλέξει μια ή περισσότερες γραμμές αναζήτησης προς

διαγραφή ή εναλλακτικά να σβήσει όλο το περιεχόμενο του αρχείο μέσω των κουμπιών επεξεργασίας.

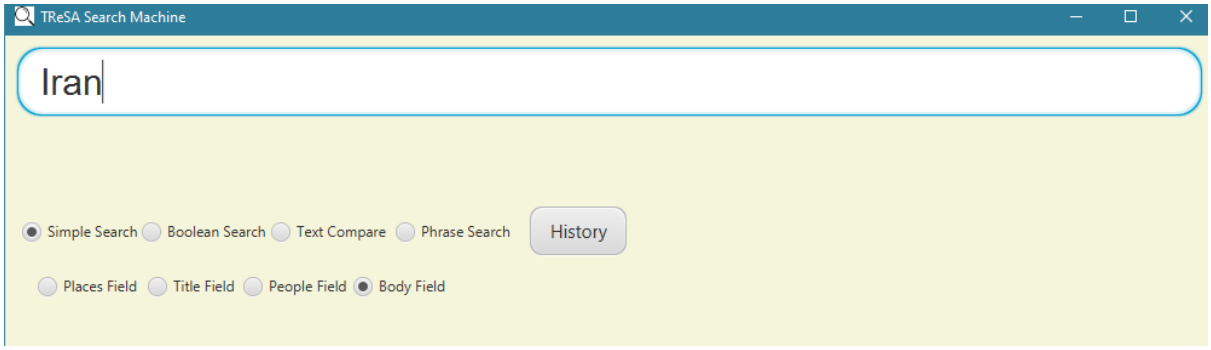


ΟΔΗΓΙΕΣ ΕΚΤΕΛΕΣΗ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

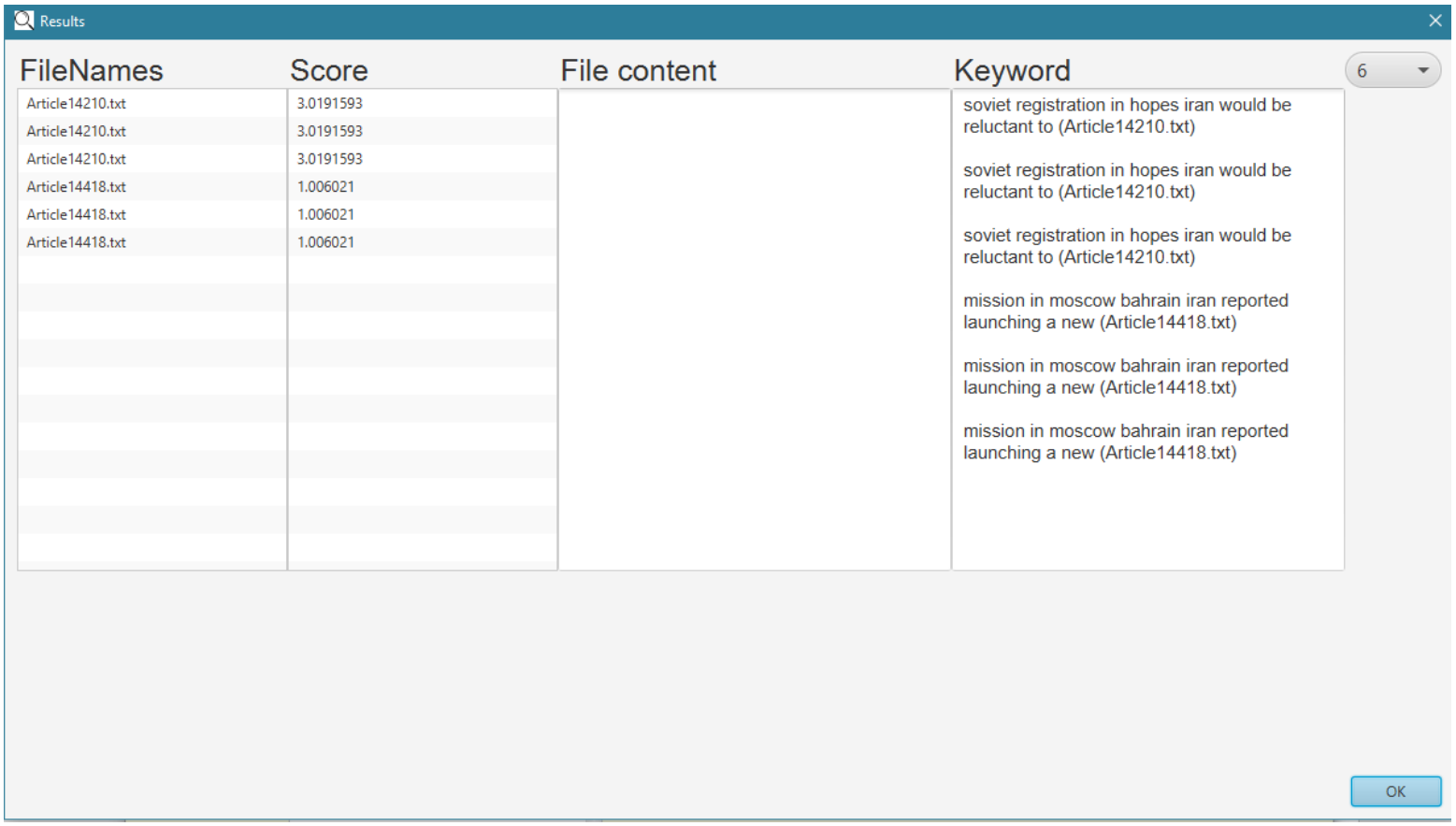
Με την έναρξη του προγράμματος εμφανίζεται το γραφικό περιβάλλον στον χρήστη.

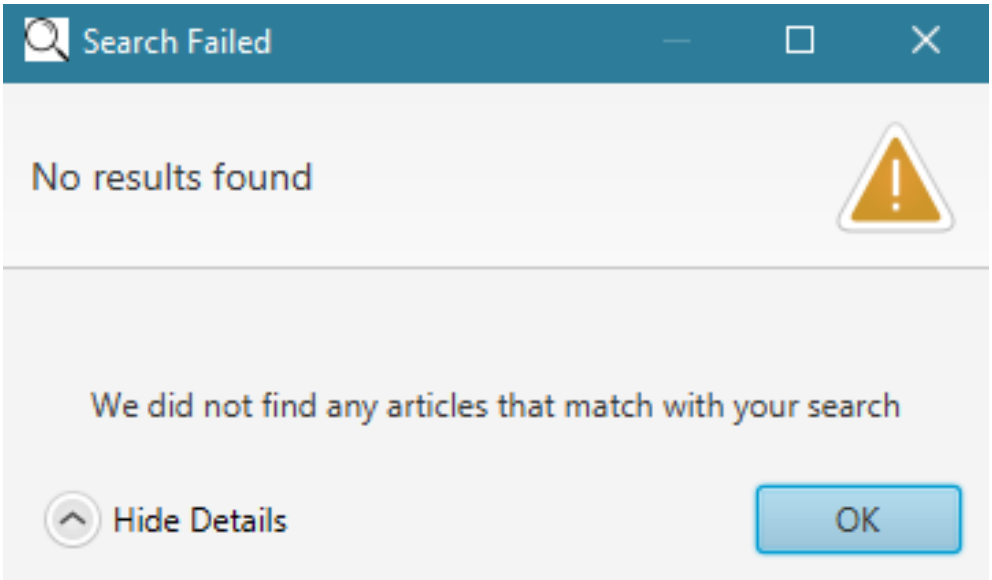


Ανάλογα την αναζήτηση που επιθυμεί να εκτελέσει, πρέπει να επιλέξει το κατάλληλο κουμπί και στην συνέχεια του πεδίο αναζήτησης.

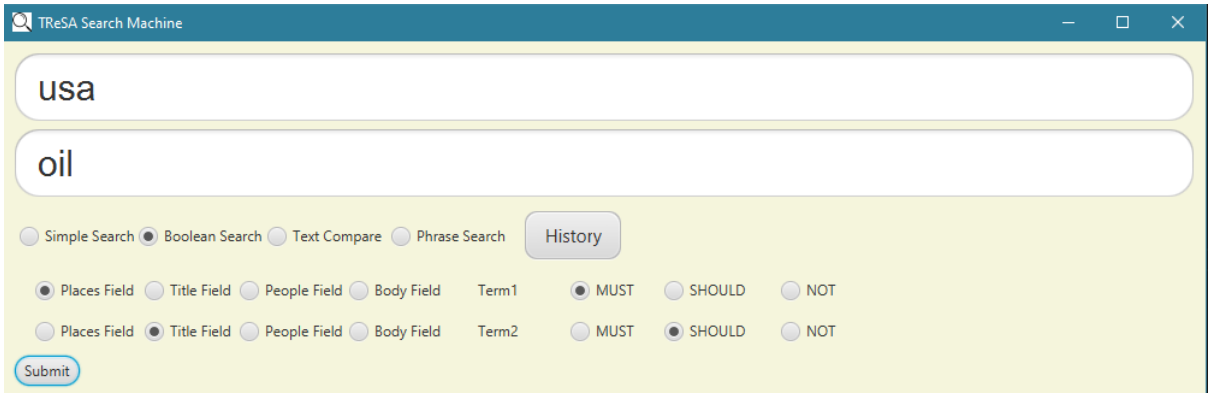


Αφού εισάγει την λέξη ή φράση που επιθυμεί να αναζητήσει, στην συνέχεια, στην απλή αναζήτηση ο χρήστης προκειμένου να στείλει το query πρέπει να πατήσει enter. Όταν αυτό πατηθεί το dialog με τις λεπτομέρειες εύρεσης εμφανίζεται αν το query ήταν επιτυχές ή ένα alert αν δεν βρέθηκε το αποτέλεσμα

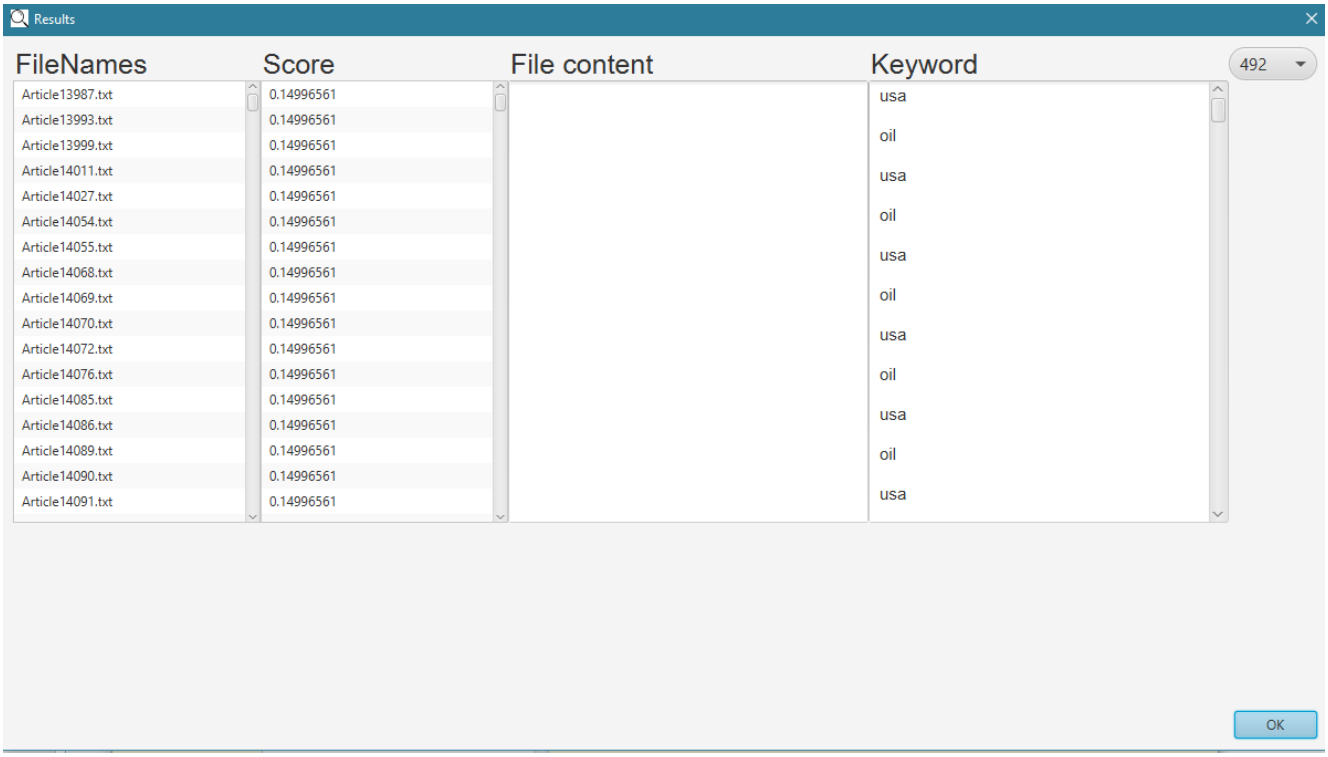




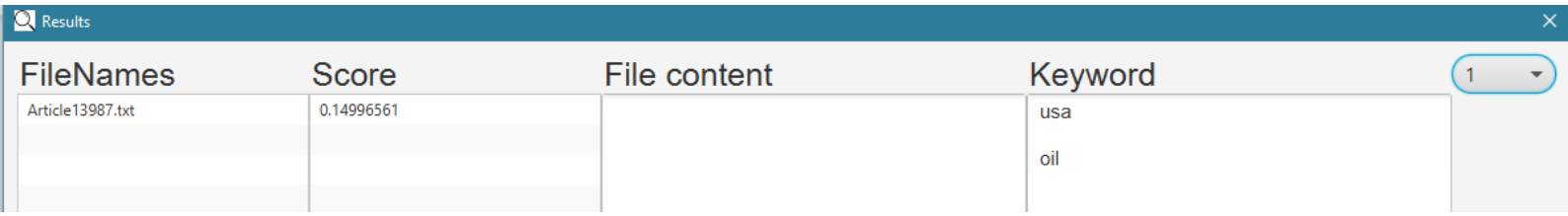
Στην αναζήτηση Boolean, ο χρήστης είναι αναγκασμένος να εισάγει δύο λέξεις που επιθυμεί να αναζητήσει, τα πεδία τους και την μεταξύ τους λογική πράξη. Επίσης αντί για enter πρέπει να πατήσει το κουμπί submit:



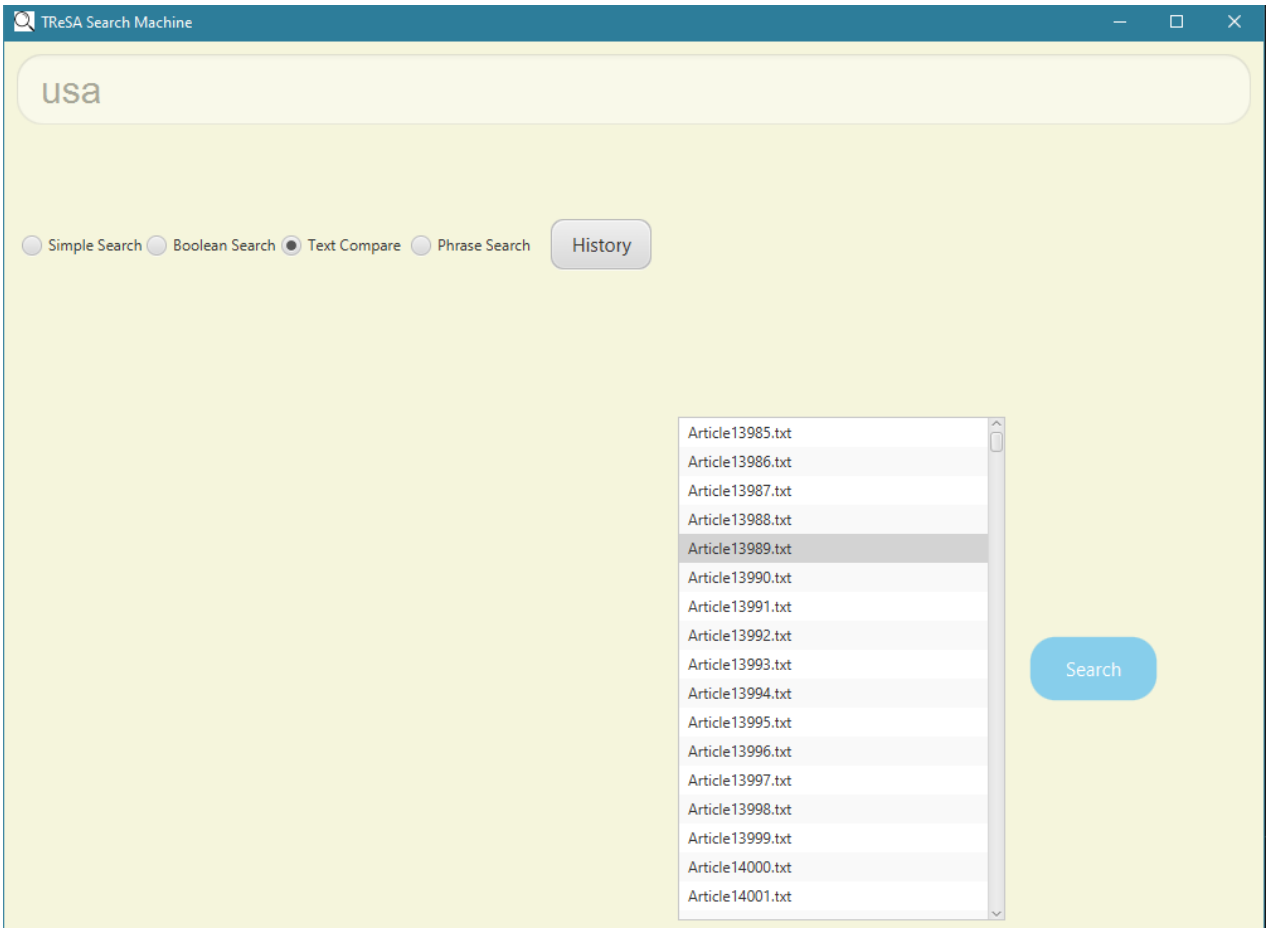
Αν η αναζήτηση πετύχει, τότε θα εμφανιστούν δεδομένα αναζήτησης και για τις δύο εισαγωγές:

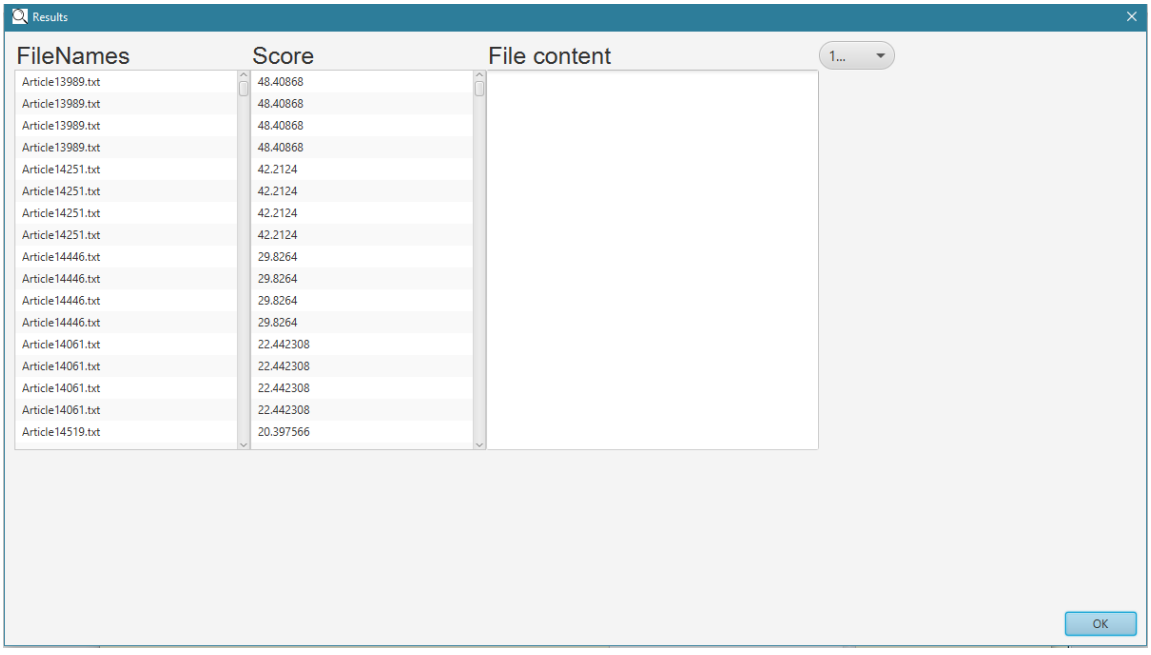


Ενδεικτικά, όπως προαναφέρθηκε στην υλοποίηση, όταν αλλάζουμε τα δεδομένα του combobox πάνω δεξιά περιορίζουμε τον αριθμό των αποτελεσμάτων που έχουν εντοπιστεί.

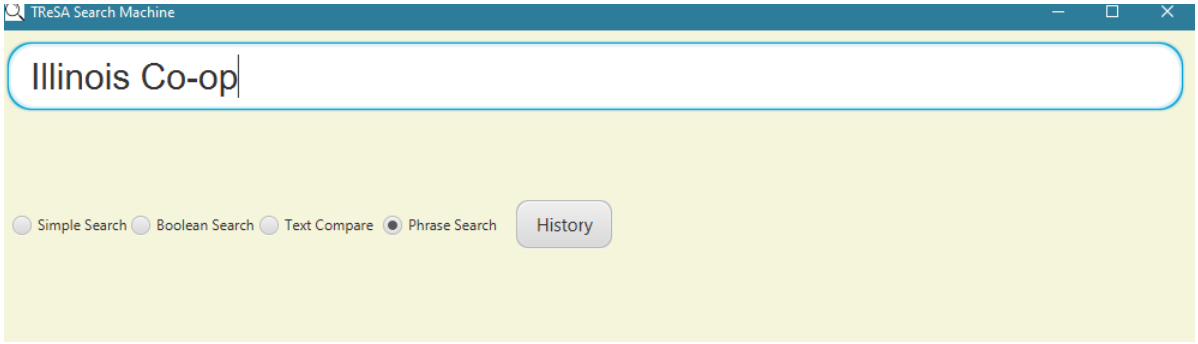


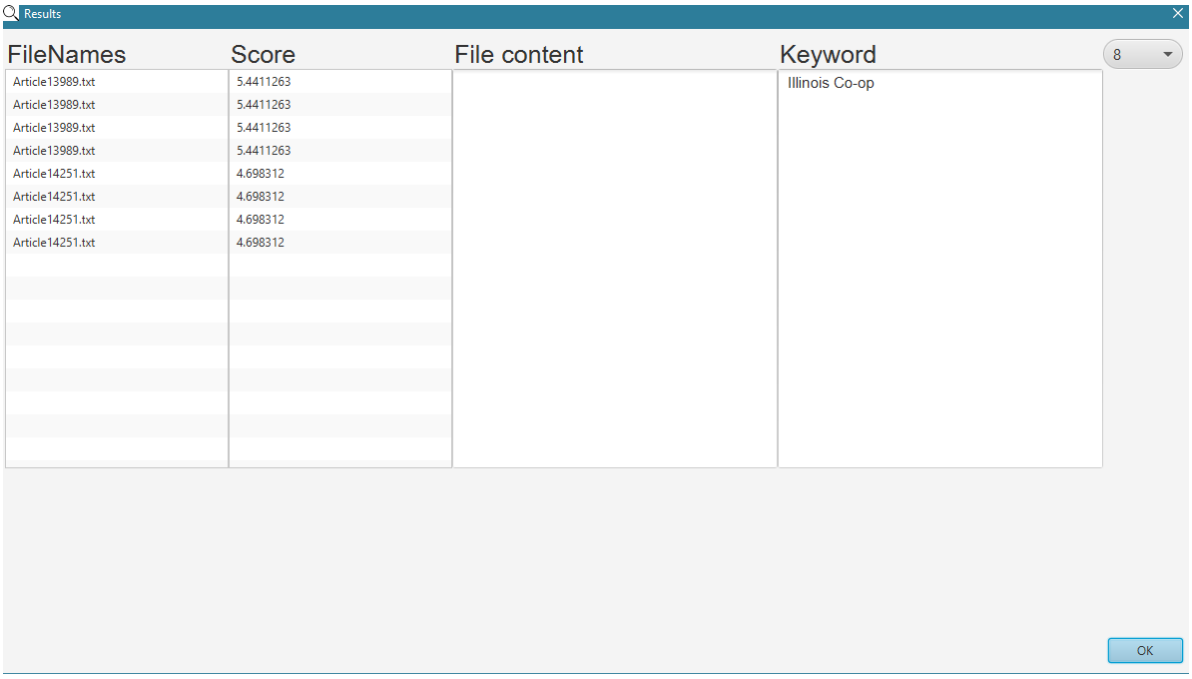
Για την Text Compare η διεπαφή του χρήστη δεν απαιτεί την συμπλήρωση ενός πεδίου εισαγωγής. Επιπλέον, καλείται να επιλέξει το αρχείο με το οποίο η λέξη θα συγκριθεί, ώστε να βρεθεί η ομοιότητά της.





Τέλος στο phrase search εισάγεται και αναζητείται μια ολόκληρη φράση πατώντας enter





Για το ιστορικό οι λειτουργίες είναι πολύ απλές. Πατώντας το κουμπί History εμφανίζεται και στην συνέχει ο χρήστης μπορεί να επιλέξει ένα ή περισσότερα στοιχεία προς διαγραφή, ή να διαγράψει όλο το περιεχόμενο.