Πανεπιστήμιο Ιωαννίνων
Τμήμα Μηχ. Η/Υ και Πληροφορικής

ΜΥΕ003
Ανάκτηση Πληροφορίας
Εαρινό εξάμηνο 2022-2023

Τελική φάση project

ΓΑΖΟΣ ΔΗΜΗΤΡΙΟΣ
ΑΜ: 4035

Ομάδα: Γάζος Δημήτριος ΑΜ: 4035

GitHub link: https://github.com/Dimigaz/MYE003_project2023_phase1_4035

Το project αυτό εκπονήθηκε ως *MAVEN PROJECT* σε *Eclipse IDE*, *Java 18* και για το γραφικό περιβάλλον χρησιμοποιήθηκε το *Eclipse WindowBuilder* σε *java swing*. Η έκδοση της *Lucene* είναι 9.5.0.

Ανάλυση Project

- ### Συλλογή εγγράφων (corpus)

Στην εργασία μας αρχικά πρέπει να συλλέξουμε τη συλλογή των εγγράφων μας. Τα έγγραφα αυτά αφορούν δεδομένα σχετικά με τραγούδια.
Για την συλλογή των εγγράφων αυτών, χρησιμοποιήσαμε μια από τις δοθέντες συλλογές που μας δίνονται από την διδάσκοντα μέσω του Kaggle:

Spotify Million Song Dataset (link: https://www.kaggle.com/datasets/notshrirang/spotify-million-song-dataset?select=spotify_millsongdata.csv)

Για την καλύτερη διαχείριση των δεδομένων μας, μετατρέψαμε το αρχείο αυτό κρατώντας τα απαραίτητα πεδία: δηλαδή το όνομα του καλλιτέχνη(artist), τον τίτλο του τραγουδιού(title) και τους στίχους(lyrics).
Επίσης αφαιρέσαμε πολλά από τα έγγραφα για να είναι πιο εύκολη η διαχείριση της συλλογής μας. Το αρχείο που περιέχει τα έγγραφά μας είναι σε αρχείο μορφής CSV. Στην τελική παραδοτέα φάση παραδίδουμε 1000 τυχαία κομμάτια από την προαναφερθέντα συλλογή μαζί με το αρχείο που παραδώσαμε στην πρώτη φάση.

- ### Σχεδιασμός

Εισαγωγή: Στόχος του προγράμματος μας είναι η υλοποίηση μιας μηχανής αναζήτησης σχετικά με τραγούδια και η εύρεση πληροφοριών για αυτά, όπως για παράδειγμα ο καλλιτέχνης που ερμηνεύει ένα συγκεκριμένο τραγούδι ή τους στίχους από ένα κομμάτι.

Ανάλυση κειμένου και κατασκευή ευρετηρίου: Αρχικά, το πρόγραμμά μας πρέπει να διαβάσει το CSV αρχείο με τη συλλογή μας. Έτσι δημιουργήσαμε την *class csvConvert* όπου μετατρέπουμε το CSV σε TXT format με διαχωριστικό στοιχείο ανάμεσα στα πεδία το “,,” για να μην υπάρχει σύγχυση αν κάποιο πεδίο περιέχει κόμμα. Για την ευκολότερη κατασκευή της κλάσης αυτής χρησιμοποιήθηκε η βιβλιοθήκη *<com.opencsv>*.

Πριν ξεκινήσει το πρόγραμμα μας, ελέγχουμε αν υπάρχει κάποιο index από προηγούμενη χρήση και αν υπάρχει το διαγράφουμε με χρήση της βιβλιοθήκης <commons.io> ώστε να μην υπάρχει σύγχυση στα αποτελέσματα σε περίπτωση που έχουμε ξανατρέξει το πρόγραμμά μας.

Επιλέγουμε το CSV αρχείο που θέλουμε να εισάγουμε στο πρόγραμμα αλλάζοντας το *String Path.* Τα CSV αρχεία που διαθέτουμε και τηρούν τις προϋποθέσεις του προγράμματός μας βρίσκονται στο φάκελο *"corpus".*
Εφόσον έχουμε επιτυχής δημιουργία του TXT αρχείου, δημιουργούμε έναν Indexer μέσω της κλάσης *class Indexing*. Αρχικά ορίζουμε τον Analyzer που θα χρησιμοποιήσουμε για το κατάλληλο tokenization, δηλαδή διαίρεση τους σε *tokens* (ακολουθία από χαρακτήρες). Χρησιμοποιούμε τον *Standard Analyzer* ο οποίος χωρίζει το κείμενο με βάση το κενό σύμβολο, μετατρέπει όλους τους χαρακτήρες σε lowercase, αφαιρεί "stop words" κλπ.
Δημιουργούμε ένα νέο directory στο δίσκο μας με το όνομα *"testindex"* το οποίο θα περιέχει το ευρετήριο index μας με τα documents που θα παράγουμε. Επίσης δημιουργούμε και το indexWriter (iwriter) στον οποίο θα προσθέσουμε τα έγγραφα documents που θα συλλέξουμε.

```
Analyzer analyzer = new StandardAnalyzer();

//create indexWriter
Directory directory = FSDirectory.open(Paths.get(IndexDir));
IndexWriterConfig config = new IndexWriterConfig(analyzer);
IndexWriter iwriter = new IndexWriter(directory, config);
```

Διαβάζουμε το TXT αρχείο, και ανά μία γραμμή τη φορά, διαχωρίζουμε μέσω ",," για να ορίσουμε τα κατάλληλα πεδία. Τα πεδία αυτά είναι με την σειρά:

1. artist
2. title
3. lyrics

Όλα μας τα δεδομένα είναι γραμμένα στην Αγγλική γλώσσα όποτε εφαρμόζουμε τους κανόνες ομαδοποιήσης και σύνταξης γι΄ αυτή.
Δημιουργούμε την μέθοδο *getDocument* με τα δεδομένα που συλλέξαμε. Με τη χρήση της *class LuceneConstants* ορίζουμε τα fields του document κατάλληλα. Εφόσον θέλουμε να μπορούμε να αναζητήσουμε keywords σε όλα τα πεδία, τότε επιλέγουμε το *textField* όπου αναφέρεται σε πεδίο το οποίο είναι indexed, tokenized και ορίζουμε να αποθηκεύονται. Προσθέτουμε τα πεδία στο document και επιστρέφουμε στην προηγούμενη συνάρτηση. Το προσθέτουμε στο iwriter μας και συνεχίζουμε τη διαδικασία μέχρι να συλλέξουμε όλα τα έγγραφα (όσες και οι γραμμές του αρχείου μας).

```
//Create Fields
Field ArtistField = new TextField(LuceneConstants.ARTIST,artist,Field.Store.YES);
Field TitleField = new TextField(LuceneConstants.TITLE,title,Field.Store.YES);
Field LyricsField = new TextField(LuceneConstants.LYRICS,lyrics,Field.Store.YES);

//Add fields to document
documents.add(ArtistField);
documents.add(TitleField);
documents.add(LyricsField);
```

```
for (String line : lines) {
    if (line.length() > 0)
    {
        //assign the fields
        f=line.split(",,");
        f_artist = f[0].trim();
        f_title = f[1].trim();
        f_lyrics = f[2].trim();

        Document doc = getDocument(f_artist,f_title,f_lyrics);
        iwriter.addDocument(doc);
        num++;
```

Αναζήτηση: Το σύστημα μας υποστηρίζει αναζήτηση εγγράφων με χρήση λέξεων κλειδιών σε οποιοδήποτε πεδίο επιλέξει ο χρήστης. Υπεύθυνη για την αναζήτηση των εγγράφων είναι η *class Searching* η οποία καλείται από το γραφικό περιβάλλον *class GUI_Panel*. Με χρήση της Java swing κατασκευάσαμε ένα interface όπου από εκεί θα αλληλεπιδρά ο χρήστης εισάγοντας τα δεδομένα και της εμφάνισης των κατάλληλων αποτελεσμάτων.
Ο χρήστης διαλέγει το MODE δηλαδή σε ποιο από τα τρία πεδία θέλει να εκτελέσει αναζήτηση. Για την καλύτερη διευκόλυνση του, ο χρήστης το ορίζει με τη επιλογή του κατάλληλου κουμπιού. Έπειτα στο παρακάτω πεδίο εισάγει τα keywords. Το πρόγραμμα εφόσον ο χρήστης πατήσει το πλήκτρο “Search” καλεί την κλάση με τη συνάρτηση Search με ορίσματα τα στοιχεία που όρισε πριν ο χρήστης.

Ορίζουμε ξανά Standard Analyzer και ανοίγουμε το ευρετήριο index που δημιουργήσαμε προηγουμένως μέσω της DirectoryReader (ireader) για να έχουμε πρόσβαση στο index μας. Έπειτα δημιουργούμε την κύρια indexSearcher(isearcher) η οποία διαβάζει τα indexed field μας κατά της περίοδο της αναζήτησης.

Το πρόγραμμά μας επιτρέπει και στον χρήστη να κάνει κάποιο λάθος κατά την διάρκεια της εισαγωγής των λέξεων κλειδιών, όπως για παράδειγμα να ξεχάσει ένα γράμμα ή να μπερδέψει τη σειρά των γραμμάτων. Για να το επιτύχουμε αυτό χρησιμοποιούμε Fuzzy Query. Η ενεργοποίηση του για να μπορεί να επιτρέπει και στον parser να χρησιμοποιεί τον Analyzer γίνεται προσθέτοντας το σύμβολο “~” στο τέλος του String.

Κάνουμε χρήση της *QueryParser* για την ανάλυση των όρων της αναζήτησης και τους μετατρέπουμε ως Query. Χρησιμοποιούμε τον Standard Analyzer για να αναλύσουμε το κείμενο της ερώτησης.
Με χρήση της isearcher που ορίσαμε προηγουμένως, ψάχνουμε στο ευρετήριο και συλλέγουμε τα έγγραφα τα οποία έχουν τουλάχιστον score μεγαλύτερο του μηδενός, δηλαδή τουλάχιστον ένα match. Αυτά τα αποθηκεύουμε στο πίνακα hits. Τρέχουμε τον πίνακα αυτό και παίρνουμε κατάλληλα τα δεδομένα και τα τοποθετούμε ανά field στο array shades το οποίο επιστρέφει η συνάρτησή μας.

Τα αποτελέσματα αυτά προκύπτουν βάση *scoring* για το πόσο συναφή είναι σε αυτό που αναζητά ο χρήστης.

```
Analyzer analyzer = new StandardAnalyzer();

//Create IndexSearcher
Directory indexDirectory = FSDirectory.open(Paths.get("testindex"));

DirectoryReader ireader = DirectoryReader.open(indexDirectory);

IndexSearcher isearcher = new IndexSearcher(ireader);

//Fuzzy Parser
QueryParser queryParser = new QueryParser(mode,analyzer); //"fieldname"
//queryParser.setFuzzyMinSim(3f);
queryParser.setFuzzyPrefixLength(3);

Query query = queryParser.parse(user_text);

//collect hits
TopDocs results = isearcher.search(query, ireader.maxDoc(), Sort.RELEVANCE);
ScoreDoc[] hits = results.scoreDocs;
```

Ο πίνακας shades που θα επιστρέψουμε από τη συνάρτηση θα περιέχει τα δεδομένα και από τα 3 fields για κάθε έγγραφο που περιέχει κάποιο από τα keywords που εισήγαγε ο χρήστης. Το γραφικό περιβάλλον δέχεται τον πίνακα αυτόν και τον προβάλει κατάλληλα *(περισσότερες λεπτομέρειες στην παρουσίαση αποτελεσμάτων).*

```
//place in array shades
String[][] shades = new String[hits.length][3];
for (int i = 0; i < hits.length; i++)
{
    Document hitDoc = isearcher.doc(hits[i].doc);
    //System.out.print("Artist name: " + hitDoc.get(LuceneConstants.ARTIST) + "|");
    shades[i][0] = hitDoc.get(LuceneConstants.ARTIST);
    //System.out.print("Title name: " + hitDoc.get(LuceneConstants.TITLE) + "|");
    shades[i][1] = hitDoc.get(LuceneConstants.TITLE);
    //System.out.println("Song lyrics: " + hitDoc.get(LuceneConstants.LYRICS));
    shades[i][2] = hitDoc.get(LuceneConstants.LYRICS);


}
```

Παρουσίαση αποτελεσμάτων: Πρόκειται για την αλληλεπίδραση του χρήστη με το πρόγραμμα μας. Στην αναζήτηση εγγράφων προαναφερθήκαμε στο interface UI που δημιουργήσαμε και πως ο χρήστης κάνει εισαγωγή τον δεδομένων. Τα αποτελέσματα της αναζήτησης αυτής παρουσιάζονται σε μορφή πίνακα 10 θέσεων ανά σελίδα. Αποφασίσαμε πως παρόλο που ο χρήστης επιλέγει σε πιο πεδίο θέλει να εκτελέσει την αναζήτηση, να εμφανίζουμε όλα τα πεδία του εγγράφου.

Αν τα έγγραφα που κάνουν match είναι περισσότερα από 10, τότε ο χρήστης με τη χρήση τον κουμπιών NEXT-PREV εμφανίζει τα επόμενα 10 ή τα προηγούμενα αντίστοιχα. Με κατάλληλα αλληλοκλεδώματα του κουμπιού επιτυγχάνουμε όταν ο χρήστης βρίσκεται στα πρώτα 10 να κλειδώνει το κουμπί PREV και όταν φτάσει στα τελευταία 10 το κουμπί NEXT. Έτσι το πρόγραμμα δείχνει στον χρήστη μια συνολική εικόνα γα το πλήθος των εγγράφων που τον ενδιαφέρουν.
(Υλοποίηση στην *class GUI_Panel*).

Μια ακόμα λειτουργία του προγράμματός μας είναι να τονίζονται οι όροι που κάνουν match ώστε να γίνεται ορατό στον χρήστη γιατί εμφανίστηκε αυτό το έγγραφο. Επίσης κάνει σημαντικά ευκολότερη τη εντόπιση της αναζήτησης στο κείμενο. Για να το πετύχουμε αυτό πριν ολοκληρωθεί το Search, χρησιμοποιούμε έναν HTML formatter και κάνουμε highlight το κείμενο της Query. Τρέχουμε τον πίνακα με τα hits και δημιουργούμε κατάλληλα TextFragments, τα οποία περιέχουν πληροφορία για το κομμάτι του εγγράφου (field) που μας ενδιαφέρει μαζί με το highlight που υλοποιήσαμε προηγουμένως. Έτσι τρέχουμε τον array shades και αντικαθιστούμε στην κατάλληλη θέση με το frag που δημιουργήσαμε. Εφόσον χρησιμοποιήσαμε HTML για να μαρκάρουμε ως bolt το Query, προσθέτουμε και τα κατάλληλα <html> και </html> ώστε να αναγνωρίζει και να τονίζει τα κατάλληλα σημεία.
(Υλοποίηση στην *class Searching*).

Με διπλό κλικ στη θέση του πίνακα, εμφανίζεται ένα νέο παράθυρο το οποίο προβάλει καλύτερα τα περιεχόμενα του κάθε πεδίου, έτσι ώστε ο χρήστης να μπορεί να διαβάσει ευκολότερα τα δεδομένα (πχ τους στίχους από το τραγούδι που έψαχνε). Υπεύθυνη για το παράθυρο αυτό είναι η *class GUI_song_info*. Τα highlighted σημεία από την αναζήτηση παραμένουν και εδώ.

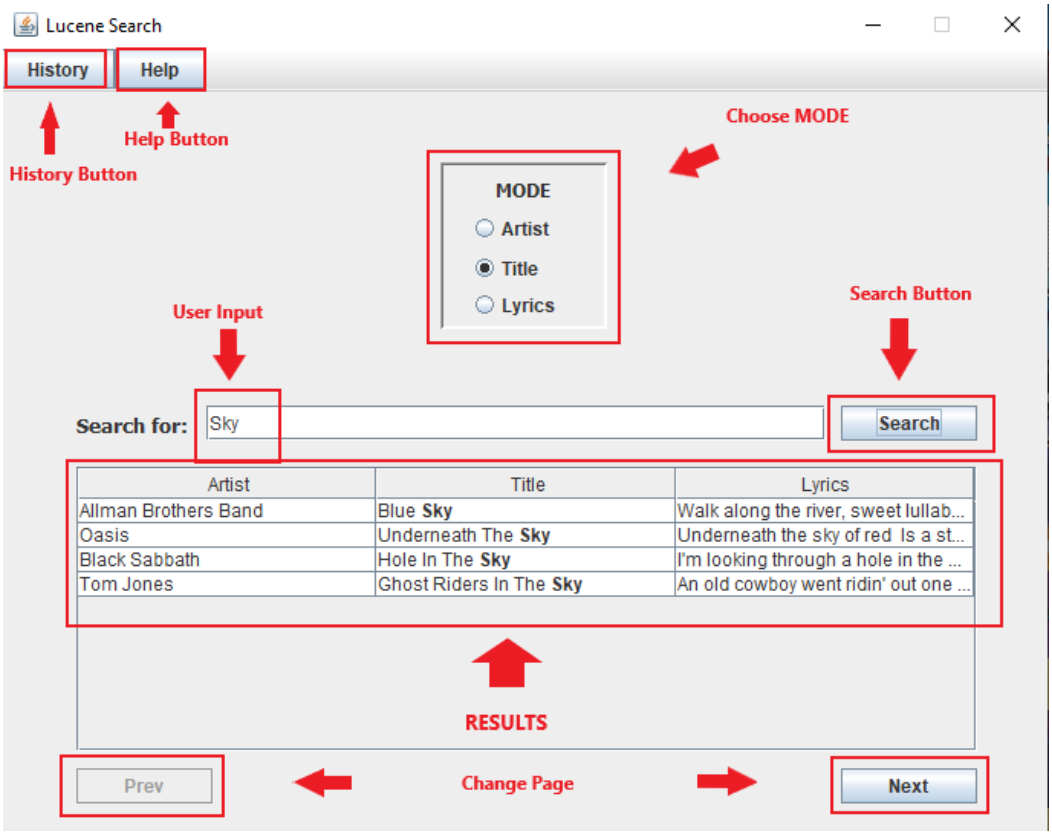
Το πρόγραμμά μας κρατάει και το ιστορικό των keywords που εισάγει ο χρήστης στο σύστημα. Στο interface υπάρχει το κουμπί “History” το οποίο ανοίγει ένα νέο παράθυρο. Στο παράθυρο αυτό περιέχονται όλες οι προηγούμενες αναζητήσεις που εκτέλεσε ο χρήστης. Κάθε φορά που τρέχουμε μια αναζήτηση, το Query προστίθεται στην λίστα αυτή. Το ιστορικό διατηρείται μόνο κατά την διάρκεια μια εκτέλεσης του προγράμματος. Αν ο χρήστης κλείσει το παράθυρο, το ιστορικό παύει να ισχύει. Υπεύθυνη για το παράθυρο αυτό είναι η *class GUI_History*, η οποία καλείται από το γραφικό μας περιβάλλον.
Μια λειτουργία που μας προσφέρει η διατήρηση του ιστορικού είναι η πρόταση ερωτήματος στο χρήστη με τη μορφή auto complete. Αν το ερώτημα που προσπαθεί να εισάγει ο χρήστης μοιάζει με κάποιο προηγούμενο ερώτημα που βρίσκεται στο ιστορικό, τότε το πρόγραμμα προτείνει στον χρήστη την προηγούμενη αναζήτηση (Όπως το Google εμφανίζει το ιστορικό αν η αναζήτηση

μοιάζει). Για να το υλοποιήσουμε αυτό χρειαστήκαμε μια εξωτερική βιβλιοθήκη <org.swinglabs.swingx> <swingx-Autocomplete> και η υλοποίηση βρίσκεται στην class GUI_Panel.

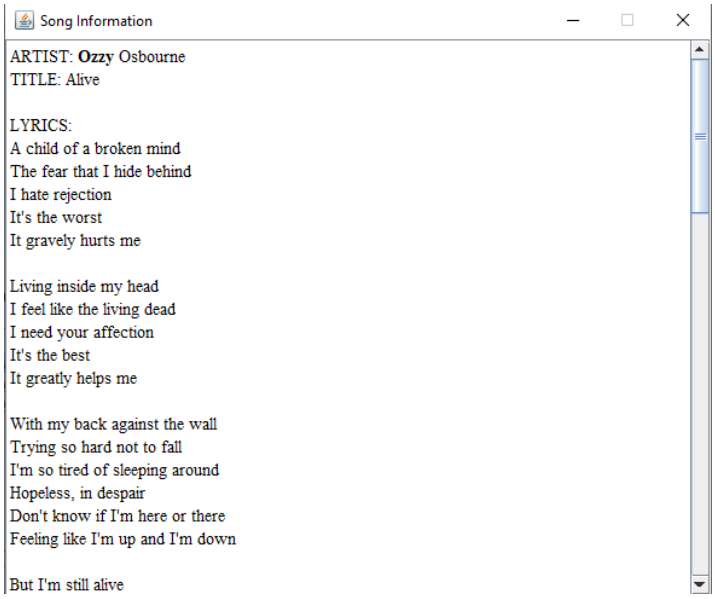
Επίσης κατασκευάσαμε το παράθυρο "Help" το οποίο παρέχει πληροφορίες στο χρήστη για το πρόγραμμα και πως να το χρησιμοποιήσει εξηγώντας το interface, αλλά και πληροφορία σχετικά με το πρόγραμμα και σε ποια βιβλιοθήκη βασίζεται. Υπεύθυνη για το παράθυρο αυτό είναι η *class GUI_Help*, η οποία καλείται από το γραφικό μας περιβάλλον.

Για τη δημιουργία του interface, όπως τοποθέτηση κουμπιών στο χώρο, μέγεθος πλαισίου κτλπ χρησιμοποιήσαμε το WindowBuilder IDE στο Eclipse σε υλοποίηση Java Swing. Τα αρχεία με τις κλάσεις που ξεκινάνε από *GUI* υλοποιούν συνολικά το γραφικό μας περιβάλλον.

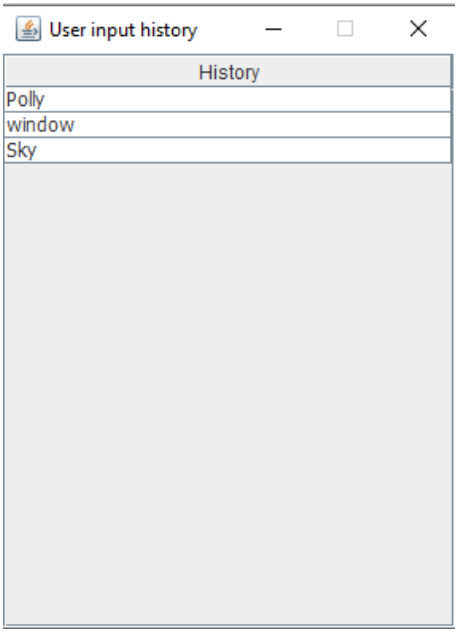
- Παρουσίαση Προγράμματος



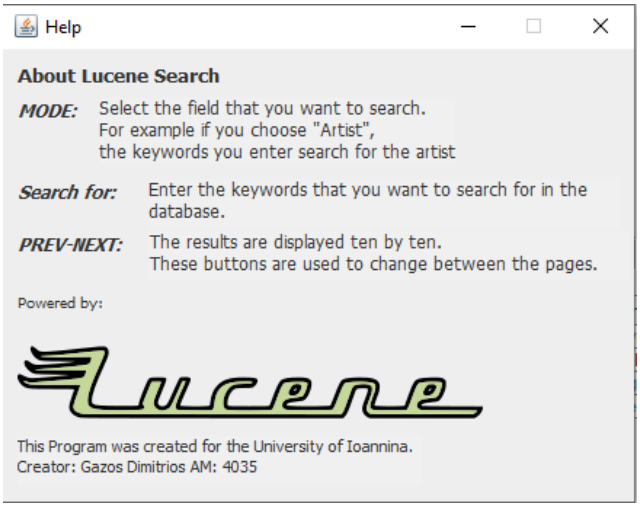
(Το main Panel του interface μας)



(παράθυρο που δείχνει τα στοιχεία του τραγουδιού που επιλέξαμε)



(Ιστορικό)



(παράθυρο πληροφοριών)

***Πράγματα που δεν καταφέραμε να υλοποιήσουμε**

Δεν καταφέραμε να κάνουμε grouping τα αποτελέσματα της αναζήτησης. Παρόλο που ψάξαμε αρκετά και προσπαθήσαμε μέσω fieldType να ορίσουμε τα ορίσματα ως *SORTED,* μας εμφάνιζε μήνυμα λάθους πώς τα πεδία που επιστρέφαμε ήταν κενά παρόλο που βεβαιωθήκαμε πως δεν ήταν. Λόγω έλλειψης χρόνου δεν προλάβαμε να το διορθώσουμε αν και δοκιμάσαμε αρκετούς τρόπους. Ενδεικτικά στον κωδικά μας έχουμε αφήσει σε μορφή σχολίου την προσπάθεια μας για grouping όπως και τα fieldType που προσπαθήσαμε να κάνουμε Sort.

.