

ΙΟΝΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ



Πτυχιακή Εργασία

Web Εφαρμογή για την Απεικόνιση και Επεξεργασία Δεδομένων
Βιβλιοθηκών

Γραμμένος Πασχάλης

A.M. Π2012021

Επιβλέπων: Μιχαήλ Στεφανιδάκης

Σεπτέμβριος 2021

Επιβλέπων

Μιχαήλ Στεφανιδάκης, Αναπληρωτής Καθηγητής, Ιόνιο Πανεπιστήμιο

Τριμελής Επιτροπή

Μιχαήλ Στεφανιδάκης, Αναπληρωτής Καθηγητής, Ιόνιο Πανεπιστήμιο

Κωνσταντίνος Οικονόμου, Καθηγητής, Ιόνιο Πανεπιστήμιο

Θεόδωρος Ανδρόνικος, Αναπληρωτής Καθηγητής, Ιόνιο Πανεπιστήμιο

Αυτή η προσπάθεια αφιερώνεται στην οικογένεια μου, τους φίλους μου
αλλά και στον επιβλέποντα καθηγητή μου που με στήριξαν και με
βοήθησαν σε όλη αυτή τη διαδρομή

Περίληψη

Τα συστήματα ηλεκτρονικού καταλόγου (Integrated Library Systems – ILS) που χρησιμοποιούνται στις βιβλιοθήκες, είναι προγράμματα με πολλές λειτουργίες τα οποία έχουν κάποια μειονεκτήματα. Συνήθως είναι προγράμματα μεγάλης κλίμακας, με σημαντικές υπολογιστικές απαιτήσεις, μη ευέλικτα και πολλές φορές μη φιλικά προς τον χρήστη. Για την επίλυση ενός υποσυνόλου των προβλημάτων αυτών των συστημάτων, στην παρούσα εργασία αναπτύσσεται μια web εφαρμογή που πραγματεύεται μια από τις λειτουργίες των ILS συστημάτων, αυτή της αναζήτησης και της εμφάνισης των εγγράφων ενός καταλόγου. Στόχος της εφαρμογής είναι με τη βοήθεια σύγχρονων τεχνολογιών να είναι ευέλικτη και ταυτόχρονα φιλική προς τον χρήστη και τους υπολογιστικούς πόρους του συστήματος. Τέλος, συμπεραίνεται ότι με την χρήση των κατάλληλων τεχνολογιών, μια τέτοια εφαρμογή μπορεί να πετύχει τους αναφερόμενους στόχους.

Λέξεις κλειδιά: συστήματα βιβλιοθήκης, προγράμματα ILS, web εφαρμογή

Abstract

The ILS systems, which are used in libraries, constitute multi-module programs that include a set of limitations. Usually, they are large-scale programs, with significant computational requirements, inflexible and often not user-friendly. In order to tackle a portion of the problems of those systems, in this thesis we develop a web application that deals with a core function of ILS systems: searching and displaying the documents of a catalogue. The goal of the application in development is, with the help of modern technologies, to be flexible and at the same time friendly both to the user and to the system's computational resources. Finally, it is concluded that by employing appropriate technologies, such an application can achieve the objectives set.

Keywords: library systems, ILS programs, web application

Περιεχόμενα

1.	Εισαγωγή.....	1
2.	Ανάλυση τεχνολογιών	2
2.1	Elasticsearch.....	2
2.2	Flask.....	3
2.3	JSON	4
2.4	MARC.....	5
2.5	JavaScript	8
2.6	CSS	11
2.7	Git.....	12
3.	Ανάλυση εφαρμογής.....	15
3.1	Αρχιτεκτονική του Back-end	15
3.2	Αρχιτεκτονική του Front-end.....	41
4.	Εγκατάσταση εφαρμογής.....	56
5.	Σύνοψη – Προτάσεις για το μέλλον	63
	Βιβλιογραφία-Δικτυογραφία	64

1. Εισαγωγή

Οι βιβλιοθήκες βασίζονται στα συστήματα διαχείρισης βιβλιοθηκών για να διαχειρίζονται τις συλλογές από αρχεία και βιβλία καθώς και τις σχέσεις με τα μέλη τους. Αυτά τα συστήματα ονομάζονται **ILS** (**I**ntegrated **L**ibrary **S**ystems) ή **ILSaS** (**I**ntegrated **L**ibrary **S**ystems **a**s a **S**ervice) τα οποία διαθέτουν αρκετές λειτουργίες οι οποίες είναι απαραίτητες σε μια βιβλιοθήκη, όπως για παράδειγμα λειτουργία καταλογογράφησης, αναζήτηση βιβλίων στους καταλόγους αλλά και είσοδος ή έξοδος δεδομένων στο πρόγραμμα, στοιχεία κινήσεων, δηλαδή οι δανεισμοί των βιβλίων στα μέλη της βιβλιοθήκης, αναφορές και ιστορικό ενός χρήστη, αρχείο δημοσιευμάτων, περιοδικά ή εφημερίδες. Αυτά τα συστήματα έχουν το μειονέκτημα ότι είναι μεγάλα και «βαριά» προγράμματα λόγω των πολλών λειτουργιών που διαθέτουν, γεγονός που τα κάνει να είναι μη ευέλικτα, δύσχρηστα και μη φιλικά προς τον χρήστη αλλά και προς τους υπολογιστικούς πόρους του συστήματος που έχουν εγκατασταθεί. Γι' αυτόν τον λόγο δημιουργήσαμε μια web εφαρμογή η οποία θα κληθεί να λύσει κάποια από τα προβλήματα των συστημάτων που χρησιμοποιούνται στις βιβλιοθήκες, όπως για παράδειγμα το μέγεθος και την ευελιξία της εφαρμογής αλλά και την δυσκολία στην εγκατάσταση. Η εφαρμογή αναπτύχθηκε με μια από τις λειτουργίες των παραπάνω συστημάτων, αυτή της αναζήτησης και εμφάνισης των καταλόγων στον χρήστη. Πιο συγκεκριμένα η εφαρμογή δίνει την δυνατότητα στον χρήστη να κάνει αναζήτηση στους καταλόγους της βιβλιοθήκης με τρεις διαφορετικούς τρόπους, εμφανίζει τα έγγραφα με όλα τα στοιχεία που περιλαμβάνει το καθένα, ώστε ο χρήστης να βρει αυτό που επιθυμεί εύκολα και γρήγορα και με όλα τα απαραίτητα στοιχεία βάση του προτύπου καταγραφής των δεδομένων που χρησιμοποιείται. Στόχος της εφαρμογής είναι με τη χρήση σύγχρονων τεχνολογιών να είναι φιλική προς τον χρήστη μέσα από μια όμορφη διεπαφή (interface) και κυρίως να είναι «ελαφριά» ώστε να είναι ευχάριστη η εμπειρία και χωρίς προβλήματα. Στα παρακάτω κεφάλαια θα αναφερθούν αλλά και θα αναλυθούν οι τεχνολογίες που χρησιμοποιήθηκαν ώστε να αναπτυχθεί αυτή η εφαρμογή, θα αναλυθούν τα πιο σημαντικά σημεία στον κώδικα αλλά θα εξηγηθεί και ο τρόπος που εγκαταστάθηκε η εφαρμογή. Τέλος θα αναφερθούν κάποιες προτάσεις επέκτασης της εφαρμογής στο μέλλον αλλά και τα συμπεράσματα που βγήκαν μετά από την ανάπτυξη της εφαρμογής.

2. Ανάλυση τεχνολογιών

Ο στόχος αυτού του project είναι η δημιουργία μιας Web εφαρμογής για την απεικόνιση και την επεξεργασία δεδομένων βιβλιοθηκών. Σε αυτό το κεφάλαιο θα αναλυθούν οι τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη αυτής της εφαρμογής, διάφορες πληροφορίες για τις τεχνολογίες αυτές αλλά και η χρησιμότητα τους. Αναφορικά, οι τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη του project είναι το Elasticsearch, το Flask, το JSON, η JavaScript, η CSS, το Git αλλά και το πρότυπο MARC για τα δεδομένα των βιβλιοθηκών.

2.1 Elasticsearch

Το **Elasticsearch**¹ είναι μια δωρεάν και ανοιχτή, κατανεμημένη μηχανή αναζήτησης και ανάλυσης για όλους τους τύπους δεδομένων. Μερικά παραδείγματα τύπων δεδομένων είναι τα παρακάτω:

- Δεδομένα κειμένων
- Αριθμητικά
- Γεωχωρικά
- Δομημένα
- Μη δομημένα

Είναι χτισμένο στο Apache **Lucene**² (το οποίο είναι μια βιβλιοθήκη μηχανών αναζήτησης κειμένου υψηλής απόδοσης γραμμένη σε Java) και κυκλοφόρησε το 2010 για πρώτη φορά από την Elasticsearch N.V. (γνωστή πλέον ως Elastic). Το Elasticsearch είναι γνωστό για την κατανεμημένη φύση του, την ταχύτητα και την επεκτασιμότητα του.

Η ταχύτητα και η επεκτασιμότητα του Elasticsearch αλλά και η ικανότητα του να ευρετηριάζει πολλούς τύπους περιεχομένου κάνει το εργαλείο αυτό να είναι χρήσιμο σε διάφορες περιπτώσεις χρήσης, όπως για παράδειγμα:

- Αναζήτηση σε εφαρμογές
- Αναζήτηση σε ιστότοπους
- Εταιρική αναζήτηση
- Αναλυτικά στοιχεία με καταγραφές
- Διάφορες μετρήσεις
- Παρακολούθηση απόδοσης της εφαρμογής
- Ανάλυση και οπτικοποίηση γεωχωρικών δεδομένων
- Στοιχεία ανάλυσης ασφάλειας
- Επιχειρηματική ανάλυση

Τα ακατέργαστα δεδομένα εισέρχονται στο Elasticsearch από ποικίλες πηγές, για παράδειγμα αρχεία καταγραφής, διάφορες μετρήσεις του συστήματος αλλά και από

¹<https://www.elastic.co>

²<https://lucene.apache.org/>

διαδικτυακές εφαρμογές. Η απορρόφηση των δεδομένων (data ingestion) είναι η διαδικασία με την οποία αναλύονται, ομαλοποιούνται αλλά και εμπλουτίζονται τα ακατέργαστα δεδομένα προτού ευρετηριαστούν στο Elasticsearch. Αφού τα δεδομένα ευρετηριαστούν, οι χρήστες μπορούν να εκτελέσουν απλά ή σύνθετα queries (ερωτήματα) στα δεδομένα και να χρησιμοποιήσουν τα αποτελέσματα για να ανακτήσουν απλές ή περίπλοκες περιλήψεις των δεδομένων τους.

Μια συλλογή εγγράφων στο Elasticsearch που σχετίζονται μεταξύ τους, ονομάζεται ευρετήριο. Το Elasticsearch αποθηκεύει τα δεδομένα ως έγγραφα JSON. Σε κάθε έγγραφο συσχετίζεται ένα σύνολο κλειδιών (ονόματα από πεδία ή ιδιότητες) με τις αντίστοιχες τιμές τους (συμβολοσειρές, αριθμοί, Boolean δεδομένα, ημερομηνίες, πίνακες με τιμές, γεωγραφικές τοποθεσίες και διάφοροι τύποι δεδομένων).

Το Elasticsearch χρησιμοποιεί το ανεστραμμένο ευρετήριο ως δομή δεδομένων, το οποίο είναι σχεδιασμένο ώστε να επιτρέπει γρήγορες αναζητήσεις κειμένου. Το ανεστραμμένο ευρετήριο καταγράφει κάθε μοναδική λέξη που εμφανίζεται σε κάθε έγγραφο και προσδιορίζει όλα τα έγγραφα στα οποία εμφανίζεται η κάθε λέξη.

Το Elasticsearch αποθηκεύει έγγραφα και δημιουργεί ένα ανεστραμμένο ευρετήριο για να κάνει τα δεδομένα των εγγράφων να είναι διαθέσιμα για αναζήτηση, σχεδόν σε πραγματικό χρόνο, κατά τη διάρκεια της διαδικασίας ευρετηρίασης. Η δημιουργία ευρετηρίου ξεκινά με το API του ευρετηρίου, μέσω του οποίου μπορεί να προστεθεί ή να ενημερωθεί ένα έγγραφο JSON σε ένα συγκεκριμένο ευρετήριο.

2.2 Flask

Το **Flask**³ είναι μια ενότητα (module) της Python, ουσιαστικά ένα Web Framework το οποίο επιτρέπει να αναπτυχθούν εύκολα διαδικτυακές εφαρμογές. Ο μικρός και εύκολος στην επέκταση πυρήνας του, κατατάσσει το Flask ουσιαστικά στα micro-frameworks τα οποία δεν περιλαμβάνουν ORM (**O**bject **R**elational **M**anager) ή σχετικές λειτουργίες. Το framework αυτό έχει διάφορα χαρακτηριστικά όπως για παράδειγμα δρομολόγηση URL (URL routing) αλλά και template engine, δηλαδή μηχανή δημιουργίας πρότυπων διαδικτυακών σελίδων.

Ένα Web Framework αντιπροσωπεύει μια συλλογή από βιβλιοθήκες και ενότητες οι οποίες βοηθούν τους developers να αναπτύξουν μια εφαρμογή ιστού χωρίς να ανησυχούν για low-level λεπτομέρειες, όπως για παράδειγμα πρωτόκολλα, διαχείριση νημάτων κλπ.

Με λίγα λόγια το Flask είναι μια βιβλιοθήκη από εργαλεία, η οποία είναι γραμμένη σε Python. Αναπτύχθηκε από τον Armin Ronacher, ο οποίος ηγήθηκε μιας διεθνούς ομάδας με λάτρεις της γλώσσας προγραμματισμού Python η οποία ονομάστηκε Pocco.

Το Flask εξαρτάται από τη μηχανή προτύπου (template) Jinja2⁴ αλλά και το WSGI εργαλείο Werkzeug⁵ (και τα δυο είναι project της ομάδας Pocco). Το Werkzeug είναι μια ολοκληρωμένη βιβλιοθήκη για web applications. Ξεκίνησε ως μια απλή συλλογή διαφόρων

³<https://flask.palletsprojects.com>

⁴<https://jinja.palletsprojects.com/en/3.0.x/intro/>

⁵<https://werkzeug.palletsprojects.com/en/2.0.x/>

βοηθητικών προγραμμάτων για εφαρμογές και έχει καταλήξει ως μια από τις πιο προηγμένες βιβλιοθήκες. Το WSGI⁶ (**W**eb **S**erver **G**ateway **I**nterface) είναι μια προδιαγραφή που περιγράφει τον τρόπο με τον οποίο ένας διακομιστής ιστού επικοινωνεί με εφαρμογές ιστού (web applications) αλλά και τον τρόπο με τον οποίο οι εφαρμογές ιστού μπορούν να συνδέονται μεταξύ τους για την επεξεργασία ενός αιτήματος.

Αυτό που κάνει το Flask να ξεχωρίζει από άλλα Python Frameworks, όπως για παράδειγμα το Django, είναι το ότι το Flask είναι πολύ κοντά στην απόφια Python. Είναι εύκολο να ξεκινήσει η δημιουργία μιας εφαρμογής χωρίς να χρειαστεί πολύ χρόνος για να μάθει κάποιος τις διαφορές ανάμεσα στο Framework και την native Python. Εκτός από αυτό, είναι πολύ σαφές Framework, κάτι το οποίο κάνει την ανάγνωση του κώδικα πιο εύκολη.

Για παράδειγμα, για να δημιουργηθεί μια «Hello World» εφαρμογή χρειάζονται μόλις μερικές γραμμές κώδικα:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World!'

if __name__ == '__main__':
    app.run()
```

2.3 JSON

Το JSON (**J**ava**S**cript **O**bject **N**otation), το οποίο εκδόθηκε από τον Douglas Crockford, είναι ένα ελαφρύ πρότυπο ανταλλαγής δεδομένων, το οποίο είναι εύκολο για τους ανθρώπους να διαβάσουν αλλά και να γράψουν σε αυτή τη μορφή. Επιπροσθέτως είναι εύκολο και για τις μηχανές να το παράγουν (generate) αλλά και να το αναλύσουν (parse). Βασίζεται σε ένα υποσύνολο τις γλώσσας προγραμματισμού JavaScript, Standard ECMA-262 Έκδοση 3^η – Δεκέμβριος 1999. Είναι ένα πρότυπο κειμένου το οποίο είναι εντελώς ανεξάρτητο από γλώσσες προγραμματισμού αλλά χρησιμοποιεί πρακτικές οι οποίες είναι γνωστές στους προγραμματιστές της C – οικογένειας γλωσσών προγραμματισμού, δηλαδή C, C++, C# αλλά και οι Java, JavaScript, Perl, Python και πολλές άλλες. Αυτές οι ιδιότητες της JSON την κάνουν ένα ιδανικό πρότυπο ανταλλαγής δεδομένων.

Το JSON είναι χτισμένο σε δύο δομές:

- Ένα σύνολο από ζευγάρια με ονόματα και τιμές. Στις γλώσσες προγραμματισμού αυτό εκλαμβάνεται ως ένα object, καταχώριση, λεξικό, πίνακα hash, δομή ή λίστα κλειδιών.
- Μια ταξινομημένη λίστα τιμών. Στις περισσότερες γλώσσες προγραμματισμού, εκλαμβάνεται ως ένας πίνακας (array), διάνυσμα, λίστα ή ακολουθία.

⁶<https://wsgi.readthedocs.io/en/latest/what.html>

Αυτές είναι οι καθολικές δομές δεδομένων. Όλες οι μοντέρνες γλώσσες προγραμματισμού τις υποστηρίζουν με κάποιον τρόπο.

Στο JSON υπάρχουν οι παρακάτω μορφές:

- Ένα αντικείμενο (object) το οποίο είναι ένα σύνολο από ζευγάρια με ονόματα και τιμές, χωρίς τάξη. Το αντικείμενο ξεκινάει αλλά και τελειώνει με άγκιστρο «{ }». Κάθε όνομα ακολουθείται από άνω-κάτω τελεία και το κάθε ζευγάρι ονόματος/τιμής χωρίζεται από τα υπόλοιπα με κόμμα.
- Ένας πίνακας είναι ένα σύνολο από τιμές σε σειρά. Οι πίνακες ξεκινάνε και τελειώνουν με αγκύλη «[]». Οι τιμές στους πίνακες χωρίζονται μεταξύ τους με κόμμα.

Οι τιμές μπορεί να έχουν την μορφή συμβολοσειράς (string), μέσα σε διπλά quotes «" "» ή και αριθμού. Μπορεί η τιμή να είναι True ή False ή ακόμα και Null. Επίσης μπορεί να έχουν την μορφή αντικειμένου (object) ή πίνακα.

Ένα string είναι μία συλλογή από Unicode χαρακτήρες, οι οποίοι βρίσκονται μέσα σε διπλά quotes. Ένας χαρακτήρας αντιπροσωπεύεται ως ένας μονός χαρακτήρας string, το οποίο μοιάζει πολύ με ένα string στην C ή στην Java. Οι αριθμοί δεν υποστηρίζουν οκταδικά και δεκαεξαδικά συστήματα.

Ένα παράδειγμα δεδομένων σε JSON είναι το παρακάτω:

```
{
  "firstName": "Dimitris",
  "lastName": "Papadopoulos",
  "isDeveloper": true,
  "pets": [
    {
      "name": "Jack",
      "type": "Dog"
    }
  ]
}
```

2.4 MARC

Το πρότυπο MARC (ακρώνυμο του **M**achine **R**eadable **C**ataloging), είναι ένας σύντομος και βολικός τρόπος για την εκχώρηση ετικετών σε κάθε μέρος μιας εγγραφής καταλόγου ώστε να μπορεί να τη χειριστεί ένας υπολογιστής. Η αρχική του μορφή αναπτύχθηκε το 1965 σε πιλοτική μορφή, γνωστή ως MARC I. Οι παράλληλες εξελίξεις που έγιναν το επόμενο διάστημα οδήγησαν την Αγγλο-Αμερικάνικη συνεργασία στο έργο MARC II που ξεκίνησε το 1968, το οποίο ήταν καθοριστικό στον ορισμό της έννοιας του MARC ως μορφή επικοινωνίας. Το MARC αναλυτικά σημαίνει:

Machine-readable: σημαίνει ότι ένας υπολογιστής μπορεί να διαβάσει και να ερμηνεύσει τα δεδομένα στην εγγραφή του καταλόγου.

Cataloging Record: ουσιαστικά εγγραφή καταλόγου, είναι μια βιβλιογραφική εγγραφή ή οι πληροφορίες που εμφανίζονται παραδοσιακά σε μια κάρτα καταλόγου. Η εγγραφή περιλαμβάνει

- Περιγραφή του αντικειμένου
- Κύρια καταχώριση και προστιθέμενες καταχωρίσεις
- Θεματικές επικεφαλίδες και
- Τον ταξινομικό αριθμό.

Περιγραφή του αντικειμένου

Οι βιβλιοθηκονόμοι ακολουθούν συγκεκριμένους κανόνες, συγκεκριμένα τους Αγγλο-Αμερικάνικους Κανόνες Καταλογογράφησης 2 (AACR2), για να συνθέσουν τη βιβλιογραφική περιγραφή ενός αντικειμένου βιβλιοθήκης. Η περιγραφή αυτή εμφανίζεται στις ενότητες μιας καρτέλας. Περιλαμβάνει τον τίτλο, τη δήλωση ευθύνης, την έκδοση, τις λεπτομέρειες του υλικού, τις πληροφορίες δημοσίευσης, τη φυσική περιγραφή αλλά και σημειώσεις.

Κύρια καταχώριση και προστιθέμενες καταχωρίσεις

Οι Αγγλο-Αμερικάνικοι κανόνες καταλογογράφησης περιλαμβάνουν επίσης κάποιους κανόνες για τον προσδιορισμό των σημείων πρόσβασης στην εγγραφή αλλά και τη μορφή που πρέπει να λάβουν αυτά τα σημεία πρόσβασης. Με λίγα λόγια, τα σημεία πρόσβασης είναι τα σημεία ανάκτησης στον κατάλογο της βιβλιοθήκης όπου μπορεί κάποιος να αναζητήσει το αντικείμενο.

Με άλλα λόγια, οι κανόνες AACR2 χρησιμοποιούνται για να απαντήσουν ερωτήσεις όπως:

- *Για αυτό το βιβλίο, θα πρέπει να υπάρχουν καταχωρίσεις στον κατάλογο για περισσότερους από ένα συγγραφείς ή περισσότερους από έναν τίτλους;*
- *Πως πρέπει να γραφτεί το όνομα του συγγραφέα;*
- *Είναι αυτό το στοιχείο «κύρια καταχώριση τίτλου»;*

Θεματικές επικεφαλίδες

Οι θεματικές επικεφαλίδες χρησιμοποιούν την Sears List of Subject Headings και την Library of Congress Subject headings ή κάποιον άλλο κατάλογο τυπικών θεματικών επικεφαλίδων για να επιλέξουν τα θέματα στα οποία θα αναφέρεται το αντικείμενο. Η χρήση μιας εγκεκριμένης λίστας ή καταλόγου είναι σημαντική για τη συνέπεια, για να διασφαλιστεί ότι όλα τα αντικείμενα σε ένα συγκεκριμένο θέμα βρίσκονται στην ίδια επικεφαλίδα και συνεπώς στην ίδια θέση στον κατάλογο.

Για παράδειγμα, η λίστα τίτλων θέματος υποδεικνύει ότι σε όλα τα βιβλία σχετικά με τις γάτες πρέπει να εκχωρηθεί το θέμα CATS. Με αυτόν τον τρόπο, όλα τα βιβλία για αυτό το θέμα θα είναι σε συγκεκριμένο μέρος στον κατάλογο για να τα βρει κάποιος.

Ταξινομικός αριθμός

Οι βιβλιοθηκονόμοι χρησιμοποιούν το Dewey Decimal Classification, το Library of Congress Classification ή κάποιο άλλο ταξινομικό σύστημα για να επιλέξουν τον αριθμό ταξινόμησης για ένα στοιχείο. Ο σκοπός αυτού του αριθμού είναι να τοποθετηθούν τα στοιχεία με το ίδιο θέμα στο ίδιο ράφι της βιβλιοθήκης. Τα περισσότερα αντικείμενα υποδιαιρούνται αλφαβητικά από τον συγγραφέα για καλύτερη ταξινόμηση. Το δεύτερο μέρος του αριθμού ταξινόμησης αντιπροσωπεύει συνήθως το όνομα του συγγραφέα, βοηθώντας λοιπόν αυτή την αλφαβητική υποδιαίρεση.

Εκδόσεις MARC

Από την αρχή που δημιουργήθηκε αυτό το πρότυπο μέχρι και σήμερα έχουν αναπτυχθεί αρκετές εκδόσεις του. Αναφορικά κάποιες από αυτές είναι οι παρακάτω:

- UKMARC
- INTERMARC
- USMARC
- UNIMARC
- MARC 21

Όροι της MARC και η εξήγησή τους

Στην MARC υπάρχουν κάποιες «πινακίδες» οι οποίες χρησιμοποιούνται για την επισήμανση δεδομένων. Τα ονόματα αυτών των πινακίδων είναι: πεδίο (field), ετικέτα (tag), δείκτης (indicator), υποπεδίο (subfield), κωδικός υποπεδίου (subfield code) και προσδιοριστής περιεχομένου.

Field

Κάθε βιβλιογραφική εγγραφή χωρίζεται σε πεδία. Υπάρχει ξεχωριστό πεδίο για τον συγγραφέα, τις πληροφορίες τίτλου και ούτω καθεξής. Αυτά τα πεδία υποδιαιρούνται σε ένα ή περισσότερα υποπεδία. Επειδή τα ονόματα των πεδίων είναι πολύ μεγάλα για να βρίσκονται σε κάθε εγγραφή, αντιπροσωπεύονται από τριψήφιες ετικέτες (tags).

Tag

Κάθε πεδίο σχετίζεται με έναν τριψήφιο αριθμό που ονομάζεται ετικέτα ή αλλιώς tag. Μια ετικέτα προσδιορίζει το πεδίο, το είδος των δεδομένων δηλαδή που ακολουθεί.

Μερικά tag που χρησιμοποιούνται συχνά στην MARC 21 είναι:

- 010 tag – επισημαίνει τον αριθμό ελέγχου της βιβλιοθήκης του κογκρέσου (LCCN)
- 020 tag – επισημαίνει τον διεθνή πρότυπο αριθμό βιβλίου (ISBN)
- 100 tag – επισημαίνει ένα όνομα ως κύρια καταχώρηση (συγγραφέα πχ)
- 250 tag – επισημαίνει την έκδοση
- 260 tag – επισημαίνει τις πληροφορίες της δημοσίευσης κλπ.

Ένα παράδειγμα ενός field είναι το παρακάτω

100 1_ \$a Pirsig, Robert M.

Indicators

Δυο θέσεις χαρακτήρων ακολουθούν το κάθε tag (όπως φαίνεται και στο παράδειγμα). Μια ή και οι δυο θέσεις μπορούν να χρησιμοποιηθούν ως δείκτες. Σε κάποια πεδία μόνο η πρώτη ή μόνο η δεύτερη θέση θα χρησιμοποιηθούν. Σε κάποια άλλα πεδία υπάρχει πιθανότητα να χρησιμοποιηθούν και οι δυο θέσεις ή και καμία. Όταν μια θέση δείκτη δεν χρησιμοποιείται και αφήνεται κενή, ονομάζεται «απροσδιόριστος» δείκτης. Για να ξεχωρίσει η κενή θέση του δείκτη χρησιμοποιείται στη θέση του κενού ο χαρακτήρας της κάτω παύλας (_), ή της δίεσης (#).

Κάθε δείκτης είναι ένας αριθμός από το 0 έως το 9. Υπάρχει πιθανότητα ο δείκτης να είναι και γράμμα αλλά είναι πολύ σπάνιο.

Subfields

Τα πιο πολλά πεδία περιέχουν αρκετά κομμάτια από δεδομένα. Κάθε τύπος δεδομένου μέσα στο πεδίο ονομάζεται υποπεδίο (subfield) και το κάθε υποπεδίο έχει και ένα κωδικό. Τα πεδία 001 μέχρι 009 δεν περιέχουν υποπεδία.

Ο κωδικός υποπεδίου είναι ένα μικρό γράμμα (σε μερικές περιπτώσεις αριθμός) που προηγείται από έναν οριοθέτη, ο οποίος χρησιμοποιείται για να διαχωρίσει τα υποπεδία μεταξύ τους. Ο οριοθέτης είναι ένα σύμβολο, για παράδειγμα η κάτω παύλα (_) ή το δολάριο (\$) κλπ.

2.5 JavaScript

Η JavaScript είναι μια γλώσσα προγραμματισμού ή καλύτερα μια scripting γλώσσα η οποία φτιάχτηκε για να «ζωντανέψει τις ιστοσελίδες». Τα προγράμματα που φτιάχνονται σε αυτή τη γλώσσα ονομάζονται scripts. Μπορούν να γραφτούν στο HTML αρχείο και να τρέξουν αυτόματα όταν φορτώνει η ιστοσελίδα. Τα scripts παρέχονται και εκτελούνται ως ένα απλό κείμενο, δηλαδή δεν χρειάζονται κάποια ειδική προετοιμασία ή compile για να τρέξουν. Ως

εκ τούτου είναι εντελώς διαφορετική από τη γλώσσα προγραμματισμού Java που υπάρχει στο όνομα της.

Γιατί ονομάστηκε JavaScript

Όταν η JavaScript φτιάχτηκε, είχε αρχικά το όνομα «LiveScript». Η Java όμως ήταν πολύ δημοφιλής εκείνη την εποχή, έτσι αποφασίστηκε ότι άμα τοποθετηθεί στο κοινό ως «μικρός αδερφός» της Java θα βοηθήσει πολύ.

Όπως εξελίχθηκε όμως, η JavaScript έγινε εντελώς ανεξάρτητη γλώσσα που ονομάστηκε ECMAScript και τώρα δεν έχει καμία σχέση με την Java.

Η JavaScript δεν είναι η μοναδική scripting γλώσσα, υπάρχουν και άλλες όπως η VBScript και η Perl. Ο κύριος λόγος για την επιλογή της JavaScript είναι η ευρεία χρήση και η διαθεσιμότητα. Όλα τα δημοφιλή προγράμματα περιήγησης (πχ Firefox, Chrome κλπ.) την υποστηρίζουν. Ως εκ τούτου, βγαίνει το αποτέλεσμα ότι τα περισσότερα άτομα που περιηγούνται στο διαδίκτυο, έχουν εγκατεστημένη μια έκδοση της JavaScript αν και υπάρχει η επιλογή από το πρόγραμμα περιήγησης να απενεργοποιηθεί. Το καλύτερο με την JavaScript είναι ότι κάποιος προγραμματιστής μπορεί να πετύχει τον σκοπό του χρησιμοποιώντας αυτή τη γλώσσα, χωρίς να ξέρει πολλά πράγματα γι' αυτή. Επειδή όμως είναι μια γλώσσα με μεγάλη δύναμη, είναι ακόμα καλύτερα όταν ξέρεις τι κάνεις και πώς να τη χρησιμοποιήσεις σωστά.

Η JavaScript αυτές τις ημέρες δεν μπορεί να εκτελεστεί μόνο στη μεριά του προγράμματος περιήγησης, αλλά και στη μεριά του server αλλά και σε κάθε συσκευή που έχει ένα ειδικό πρόγραμμα που ονομάζεται μηχανή JavaScript. Οι μηχανές έχουν διαφορετικές κωδικές ονομασίες, για παράδειγμα:

- V8 – στο Chrome και στο Opera
- SpiderMonkey – στο Firefox
- Υπάρχουν και άλλες ονομασίες όπως «Chakra» για τον Internet Explorer, «ChakraCore» για τον Microsoft Edge, «Nitro» και «SquirrelFish» στο Safari κλπ.

Πως δουλεύουν αυτές οι μηχανές

1. Η μηχανή διαβάζει το script.
2. Στη συνέχεια μετατρέπει (κάνει compile) το script σε γλώσσα μηχανής.
3. Τέλος, ο κώδικας εκτελείται αρκετά γρήγορα.

Τι μπορεί να κάνει η JavaScript

Η μοντέρνα JavaScript είναι μια «ασφαλής» γλώσσα προγραμματισμού. Δεν παρέχει χαμηλού επιπέδου πρόσβαση στην μνήμη ή στην CPU. Οι δυνατότητές της εξαρτώνται από το περιβάλλον που εκτελείται. Για παράδειγμα η Node.js (πλατφόρμα ανάπτυξης λογισμικού, κυρίως διακομιστών, χτισμένη σε JavaScript περιβάλλον) υποστηρίζει λειτουργίες που

επιτρέπουν στην JavaScript του server-side να διαβάζει ή να γράφει αρχεία στον διακομιστή, να πραγματοποιεί αιτήματα δικτύου κλπ.

Η JavaScript που εκτελείται στο client-side επιτρέπει:

- Να προστεθεί νέος HTML κώδικας στην ιστοσελίδα, να τροποποιηθεί ήδη υπάρχον περιεχόμενο, στυλ κλπ.
- Αντίδραση στις ενέργειες των χρηστών, για παράδειγμα στο πάτημα του κλικ του ποντικιού, στις κινήσεις του δείκτη του ποντικιού κλπ.
- Αποστολή αιτημάτων μέσω δικτύου σε απομακρυσμένους server, λήψη ή και αποστολή αρχείων (πχ AJAX τεχνολογία)
- Λήψη και ορισμό των cookies, εμφάνιση μηνυμάτων κλπ.
- Αποθήκευση των δεδομένων στο client-side κομμάτι, με λίγα λόγια local storage των δεδομένων.

Τι δεν μπορεί να κάνει η JavaScript

Η δυνατότητες της JavaScript στο client-side, στο πρόγραμμα περιήγησης δηλαδή είναι περιορισμένες για λόγους ασφάλειας του χρήστη. Ο στόχος είναι η πρόληψη μιας «κακόβουλης» ιστοσελίδας από το να έχει πρόσβαση σε ιδιωτικές πληροφορίες ή να βλάψει τα δεδομένα του χρήστη.

Παραδείγματα τέτοιων περιορισμών περιλαμβάνουν:

- Η JavaScript που εκτελείται στο client-side δεν μπορεί να διαβάσει ή να γράψει σε αρχεία που βρίσκονται στον σκληρό δίσκο, ούτε να αντιγράψει ή να εκτελέσει προγράμματα. Δεν έχει άμεση πρόσβαση σε λειτουργίες του λειτουργικού συστήματος. Τα μοντέρνα προγράμματα περιήγησης της επιτρέπουν να δουλεύει με αρχεία, αλλά η πρόσβαση είναι περιορισμένη και παρέχεται μόνο εάν ο χρήστης επιτρέψει ορισμένες ενέργειες, όπως να σύρουμε ένα αρχείο στο παράθυρο του περιηγητή μας ή να το επιλέξουμε μέσα από το <input> tag. Υπάρχουν τρόποι για αλληλεπίδραση με την κάμερα ή το μικρόφωνο αλλά και άλλες συσκευές, αλλά χρειάζεται η αποκλειστική άδεια από τον χρήστη. Αυτό έχει ως αποτέλεσμα να αποτραπεί η κρυφή ενεργοποίηση της κάμερας από την JavaScript.
- Οι διαφορετικές καρτέλες ή παράθυρα του περιηγητή μας, δεν «γνωρίζονται» μεταξύ τους. Κάποιες φορές συμβαίνει, όταν για παράδειγμα ένα παράθυρο χρησιμοποιεί JavaScript για να ανοίξει ένα άλλο. Αλλά και σε αυτή τη περίπτωση η μια σελίδα δεν έχει πρόσβαση στην άλλη.
- Η JavaScript μπορεί εύκολα να επικοινωνήσει μέσω του δικτύου με τον διακομιστή που προέρχεται η σελίδα. Η δυνατότητα της όμως να λαμβάνει δεδομένα από διαφορετικές ιστοσελίδες είναι περιορισμένη. Αν και γίνεται, απαιτεί συμφωνία (που εκφράζεται σε κεφαλίδες HTTP) από την απομακρυσμένη πλευρά. Για ακόμα μια φορά είναι ένας περιορισμός για λόγους ασφάλειας.

Τι κάνει την JavaScript μοναδική

Τρία είναι τα στοιχεία που την κάνουν μοναδική:

- Πλήρης ενσωμάτωση με HTML / CSS.
- Τα απλά πράγματα γίνονται απλά και εύκολα.
- Η υποστήριξη από όλα τα μεγάλα προγράμματα περιήγησης και το γεγονός ότι είναι ενεργοποιημένη ως προεπιλογή.

Η σύνταξη της JavaScript δεν ταιριάζει σε όλους. Ο κάθε άνθρωπος επιθυμεί διαφορετικές λειτουργίες. Αυτό είναι αναμενόμενο μιας και τα project και οι απαιτήσεις είναι διαφορετικά για όλους. Τελευταία έχει εμφανιστεί μια πληθώρα από γλώσσες οι οποίες μετατρέπονται σε JavaScript πριν εκτελεστούν στον περιηγητή. Μερικά παραδείγματα, αναφορικά, είναι τα παρακάτω:

- CoffeeScript
- TypeScript
- Flow
- Dart
- Brython
- Kotlin

2.6 CSS

Η CSS (Cascading Style Sheets) είναι μια γλώσσα η οποία επιτρέπει να δημιουργούνται όμορφες ιστοσελίδες. Δημοσιεύτηκε για πρώτη φορά από το W3C (World Wide Web Consortium) το 1995. Μέχρι το 1996 είχε γίνει πολύ διαδεδομένη, όσο και η HTML. Στην ουσία δίνει τη δυνατότητα επεξεργασίας των στοιχείων που περιέχει μια ιστοσελίδα, της θέσης των αντικειμένων, του χρώματος της γραμματοσειράς και του background και πολλά άλλα, με λίγα λόγια ορίζεται το πώς παρουσιάζονται τα έγγραφα στον χρήστη. Προφανώς, η CSS είναι άχρηστη χωρίς ένα έγγραφο, μιας και δεν θα έχει περιεχόμενο να εμφανίσει. Ένα έγγραφο είναι συνήθως δομημένο με τη χρήση μια γλώσσα σήμανσης όπως η HTML (HyperText Markup Language) αλλά και άλλες γλώσσες σήμανσης όπως SVG ή XML.

Η CSS μπορεί να χρησιμοποιηθεί για βασικό στυλ κειμένου ενός εγγράφου, για παράδειγμα αλλαγή του χρώματος και του μεγέθους ενός κειμένου. Μπορεί να χρησιμοποιηθεί για τη δημιουργία διάταξης των στοιχείων ενός εγγράφου, για παράδειγμα την μετατροπή μιας απλής στήλης που περιέχει ένα κείμενο σε μια κύρια περιοχή για το κείμενο και μιας πλευρικής στήλης για σχετικές πληροφορίες. Επίσης μπορεί να χρησιμοποιηθεί για εφέ όπως για παράδειγμα δημιουργία εφέ εισόδου του κειμένου στο έγγραφο.

Είναι μια γλώσσα που βασίζεται σε κανόνες, τους οποίους τους θέτει ο developer, ορίζοντας διάφορες ομάδες που περιέχουν συγκεκριμένο στυλ εμφάνισης που θα πρέπει να εφαρμοστούν σε συγκεκριμένα στοιχεία του κειμένου ή ομάδες στοιχείων μιας ιστοσελίδας. Για παράδειγμα «Θέλω τα στοιχεία της κύριας επικεφαλίδας της σελίδας μου να είναι

μέγεθος 16 pixel και να είναι κόκκινο χρώμα». Αυτός ο κανόνας στην CSS θα έπρεπε να γραφτεί:

```
h1 {  
  color: red;  
  font-size: 16px;  
}
```

Ο κανόνας ανοίγει με τον «selector» ο οποίος επιλέγει το HTML στοιχείο που θέλουμε να επεξεργαστούμε την εμφάνιση του (στη συγκεκριμένη περίπτωση το <h1>). Στη συνέχεια υπάρχει ζευγάρι από αγκύλες μέσα στις οποίες γράφουμε μια ή παραπάνω δηλώσεις, οι οποίες παίρνουν τη μορφή ζευγαριού ιδιότητας – αξίας. Κάθε ζευγάρι καθορίζει μια ιδιότητα του στοιχείου που επιλέγουμε και στην συνέχεια ορίζουμε την τιμή που επιθυμούμε να έχει. Στο παραπάνω παράδειγμα επεξεργαστήκαμε την ιδιότητα «color», δηλαδή του χρώματος της γραμματοσειράς. Η ιδιότητα αυτή παίρνει διάφορες τιμές από χρώματα ώστε να μπορεί κάποιος να το διαμορφώσει όπως ακριβώς επιθυμεί. Επίσης έχουμε την ιδιότητα «font-size» η οποία καθορίζει το μέγεθος της γραμματοσειράς και παίρνει τιμές με ποικίλες μονάδες μεγέθους.

Ο τρόπος που δουλεύει η CSS είναι απλός. Όταν ένα πρόγραμμα περιήγησης μας εμφανίζει ένα έγγραφο, πρέπει να συνδυάσει το περιεχόμενο του εγγράφου με τις πληροφορίες του στυλ. Επεξεργάζεται το έγγραφο σε διάφορα στάδια τα οποία είναι τα παρακάτω (κάθε πρόγραμμα περιήγησης θα χειριστεί με διαφορετικό τρόπο τη διαδικασία).

Η παρακάτω είναι μια απλοποιημένη έκδοση του τι συμβαίνει:

- Το πρόγραμμα περιήγησης φορτώνει το HTML
- Μετατρέπει το HTML σε DOM (**D**ocument **O**bject **M**odel). Το DOM αντιπροσωπεύει το έγγραφο στη μνήμη του υπολογιστή.
- Το πρόγραμμα περιήγησης ανακτά τους περισσότερους πόρους που συνδέονται με το HTML έγγραφο, όπως για παράδειγμα ενσωματωμένες εικόνες ή βίντεο αλλά και το CSS αρχείο που έχουμε φτιάξει εμείς.
- Το πρόγραμμα περιήγησης αναλύει το ληφθέν CSS και ταξινομεί τους κανόνες που έχουν οριστεί, στα διαφορετικά στοιχεία (classes, ID κλπ.)
- Το αποτέλεσμα τοποθετείται στη δομή αφού εφαρμοστούν οι κανόνες.
- Το οπτικό αποτέλεσμα εμφανίζεται στην οθόνη μας.

2.7 Git

Το Git είναι ένα παράδειγμα ενός καταμεμημένου συστήματος ελέγχου έκδοσης (DVCS), το οποίο χρησιμοποιείται συνήθως για την ανάπτυξη λογισμικού ανοιχτού κώδικα αλλά και εμπορικού. Τα συστήματα αυτά επιτρέπουν την πλήρη πρόσβαση σε κάθε αρχείο αλλά και κλάδο ενός project και επιτρέπει σε κάθε χρήστη την πρόσβαση σε ένα πλήρες και αυτόνομο ιστορικό όλων των αλλαγών στον κώδικα. Το Git δεν χρειάζεται συνεχή σύνδεση με ένα

κεντρικό αποθετήριο και αυτό είναι που το έκανε ιδιαίτερα δημοφιλές. Οι προγραμματιστές – χρήστες μπορούν να εργαστούν στο project μαζί με τα υπόλοιπα μέλη της ομάδας τους και να συνεργαστούν ασύγχρονα από οποιαδήποτε ζώνη ώρας.

Χωρίς το σύστημα αυτό, τα μέλη μιας προγραμματιστικής ομάδας υπάρχει πιθανότητα να προβούν σε περιττές εργασίες, καθυστέρηση στο χρονοδιάγραμμα που έχει τεθεί για το project αλλά και σε πολλαπλά αντίγραφα ενός έργου. Για να εξαλειφθεί η περιττή εργασία, το Git δίνει τη δυνατότητα σε κάθε άτομο που συνεισφέρει σε ένα project να δει μια ενοποιημένη όψη ενός έργου αλλά και να δει τις εργασίες που βρίσκονται ήδη σε εξέλιξη. Η δυνατότητα να δει κάποιος το ιστορικό αλλαγών στον κώδικα, βοηθά τα μέλη της προγραμματιστικής ομάδας του project να δουν ποιος έκανε τις αλλαγές αυτές και πώς συμβάλλει κάθε μέλος της στην ανάπτυξη του έργου. Αυτό βοηθάει να παραμείνουν προσηλωμένοι στον στόχο ενώ εργάζονται ανεξάρτητα.

Σύμφωνα με μια έρευνα προγραμματιστών στο Stack Overflow, περισσότερο από το 70% των προγραμματιστών χρησιμοποιούν το Git στην δουλεία τους, το οποίο το καθιστά ως το πιο χρησιμοποιημένο σύστημα ελέγχου στον κόσμο.

Ένα αποθετήριο (repository) ή ένα Git project, περιλαμβάνει όλη τη συλλογή αρχείων και φακέλων που σχετίζονται με ένα έργο, μαζί με το ιστορικό αναθεωρήσεων κάθε αρχείου. Το ιστορικό των αρχείων εμφανίζεται ως ένα στιγμιότυπο στο χρόνο που ονομάζεται commit, τα οποία commit μπορούν να οργανωθούν σε πολλές γραμμές ανάπτυξης που ονομάζονται branches. Επειδή το Git είναι ένα κατανεμημένο σύστημα ελέγχου έκδοσης, τα αποθετήρια είναι αυτόνομες μονάδες και όποιος έχει στη διάθεση του ένα αντίγραφο του αποθετηρίου, έχει στη διάθεση του ολόκληρο τον κώδικα και το ιστορικό του. Με την χρήση της γραμμής εντολών ή άλλων διεπαφών εύκολων στη χρήση, ένα αποθετήριο git επιτρέπει επίσης: κλωνοποίηση του αποθετηρίου, αλληλεπίδραση με το ιστορικό του project, δημιουργία branches, συγχώνευση των branches αλλά και σύγκριση των αλλαγών σε διάφορες εκδόσεις του κώδικα στο project.

Δουλεύοντας με ένα αποθετήριο, η ανάπτυξη του έργου διατηρείται οργανωμένη και προστατευμένη. Οι προγραμματιστές που δουλεύουν σε αυτό το έργο, ενθαρρύνονται να βρουν και να διορθώσουν σφάλματα στον κώδικα, ή να δημιουργήσουν νέες δυνατότητες στο έργο χωρίς να φοβούνται ότι θα επηρεαστεί το βασικό κομμάτι της ανάπτυξης. Το Git το διευκολύνει αυτό το εγχείρημα μέσω της χρήσης θεματικών branches, τα οποία είναι ελαφροί δείκτες στο ιστορικό των commit.

Παρακάτω θα αναλυθούν κάποιες βασικές εντολές στο Git:

- **git init:** αρχικοποιεί ένα καινούργιο αποθετήριο. Προσθέτει έναν κρυφό υποφάκελο στον υπάρχοντα κατάλογο, που φιλοξενεί την εσωτερική δομή δεδομένων που απαιτείται για τον έλεγχο της έκδοσης.
- **git clone:** δημιουργεί έναν τοπικό κλώνο ενός project που υπάρχει ήδη. Ο κλώνος περιλαμβάνει όλα τα αρχεία, τα branches αλλά και το ιστορικό του project.
- **git add:** δημιουργεί ένα στάδιο στο project. Είναι το πρώτο μέρος αυτής της διαδικασίας δυο βημάτων στο project. Τυχόν αλλαγές που υπάρχουν θα γίνουν μέρος του επόμενου βήματος ώστε να γίνουν μέρος του ιστορικού του project. Η διαδικασία της σταδιοποίησης (staging) και του commit δίνει τον πλήρη έλεγχο του ιστορικού του έργου στους προγραμματιστές, χωρίς να αλλάζουν τον τρόπο που προγραμματίζουν και λειτουργούν.

- **git commit:** αποθηκεύει το στιγμιότυπο της εφαρμογής με τις αλλαγές που έχουν πραγματοποιηθεί και ολοκληρώνει τη διαδικασία παρακολούθησης των αλλαγών που γίνονται. Με λίγα λόγια, το commit λειτουργεί όπως η λήψη μιας φωτογραφίας. Ό,τι έχει γίνει git add, θα γίνει μέρος του στιγμιότυπου με το git commit.
- **git status:** εμφανίζει την κατάσταση των αλλαγών ως untracked, modified ή staged.
- **git branch:** εμφανίζει τα branch στα οποία γίνονται αναβαθμίσεις ή προσθήκες.
- **git merge:** συγχωνεύει τα κομμάτια ανάπτυξης μαζί. Αυτή η εντολή χρησιμοποιείται συνήθως για των συνδυασμό όλων των αλλαγών που έχουν γίνει σε δυο διαφορετικά branches. Για παράδειγμα, ένας προγραμματιστής θα κάνει συγχώνευση όταν θέλει να συνδυάσει τις αλλαγές που έκανε με το κύριο κομμάτι της ανάπτυξης του project.
- **git pull:** ενημερώνει το τοπικό μέρος της ανάπτυξης του project με τις ενημερώσεις που έχουν γίνει στο απομακρυσμένο μέρος του. Οι προγραμματιστές χρησιμοποιούν αυτή την εντολή εάν ένα άλλο μέλος της ομάδας έχει κάνει ένα commit σε ένα branch απομακρυσμένα και υπάρχει περίπτωση να έχει αντίκτυπο στη δική του τοπική έκδοση του project.
- **git push:** ενημερώνει το απομακρυσμένο αποθετήριο με ό,τι commit έχουν γίνει τοπικά σε ένα branch.

3. Ανάλυση εφαρμογής

Σε αυτό το κεφάλαιο θα αναλυθεί η αρχιτεκτονική της εφαρμογής που αναπτύχθηκε. Συγκεκριμένα η αρχιτεκτονική της περιέχει τόσο το back-end κομμάτι όσο και το front-end. Πιο αναλυτικά θα εξηγηθεί η λειτουργία των σημαντικών σημείων στον κώδικα, καθώς και ο αντίκτυπος του, ώστε να επιτευχθεί το επιθυμητό αποτέλεσμα της εφαρμογής. Στόχος του project ήταν, η εφαρμογή που δημιουργήθηκε να είναι πιο «ελαφριά» και πιο εύχρηστη από τις μεγάλες και «βαριές» που χρησιμοποιούν οι βιβλιοθηκονόμοι για να δουν τα δεδομένα μιας βιβλιοθήκης αλλά και να τα επεξεργαστούν. Ένας επιπλέον στόχος ήταν να είναι εύχρηστη αλλά και με όμορφο περιβάλλον το οποίο θα είναι φιλικό προς τον χρήστη.

Τα δεδομένα που εμφανίζονται στον χρήστη έχουν επεξεργαστεί και έχουν χωριστεί σε ομάδες που ονομάσαμε datagroup. Μέσα σε αυτές τις ομάδες υπάρχουν δύο τύποι δεδομένων. Αρχικά είναι οι ομάδες που φτιάξαμε για τα tag, τις οποίες τις ονομάσαμε taggroup και τα κυρίως δεδομένα τα οποία εμφανίζουμε στον χρήστη, τα οποία τα ονομάσαμε values.

3.1 Αρχιτεκτονική του Back-end

Το Back-end είναι η πλευρά του server σε μια εφαρμογή ή μια ιστοσελίδα. Αποθηκεύει και τακτοποιεί δεδομένα και διασφαλίζει ότι όλα στην πλευρά του χρήστη ενός ιστότοπου ή μιας web εφαρμογής λειτουργούν καλά. Ουσιαστικά είναι το τμήμα μιας εφαρμογής με το οποίο ο χρήστης δεν έχει αλληλεπίδραση, οπότε δεν έρχεται σε άμεση επαφή μαζί του. Στη συγκεκριμένη εφαρμογή, το back-end είναι υπεύθυνο αρχικά για την επεξεργασία των δεδομένων που έχουν γίνει indexed στο Elasticsearch, ώστε να μπορέσει στην συνέχεια να τα προβάλλει μέσω του front-end στον χρήστη. Εκτός από αυτό, στο back-end ορίζονται τα routes της εφαρμογής, οι διαδρομές δηλαδή με τις οποίες θα μπορέσει να δει το αποτέλεσμα του κώδικα στο πρόγραμμα περιήγησης ο χρήστης.

Αρχικά δημιουργήθηκε το route για το homepage, την αρχική σελίδα δηλαδή η οποία βλέπει ο χρήστης όταν θα ανοίγει την web εφαρμογή. Ο ρόλος του server σε αυτό το στάδιο είναι να εμφανίσει τα κατάλληλα δεδομένα ώστε να επιλέξει ο χρήστης τι θέλει ακριβώς να αναζητήσει καθώς και να «χτίσει» την σελίδα που θα δει κάποιος όταν θα ανοίξει την εφαρμογή.

```
@form_bp.route('/')
def index_page():

    warning = request.args.get("warning")
    no_result = request.args.get("no_result")
    data_list = select_list()

    return render_template("index_page.html", data_list =
data_list, warning=warning, no_result=no_result)
```

- **warning:** Λογική μεταβλητή που παίρνει την τιμή της με την μέθοδο GET, μέσα από το URL από άλλη συνάρτηση σε άλλο route.
- **no_result:** Λογική μεταβλητή που παίρνει την τιμή της με την μέθοδο GET.
- **data_list:** Μια λίστα με όλα τα ονόματα των taggroup που θα χρειαστεί να παρουσιάσουμε στον χρήστη σε μια φόρμα select.

Ο παραπάνω κώδικας πετυχαίνει αυτό ακριβώς που εξηγήθηκε πιο πάνω. Αρχικά δηλώνεται το route το οποίο λέει στο Flask το URL που θα ενεργοποιήσει τη συνάρτηση `index_page`, σε αυτή τη περίπτωση η διαδρομή που δηλώνεται είναι το `homepage` της εφαρμογής. Στη συνάρτηση αρχικά έχουν δηλωθεί δυο μεταβλητές οι οποίες περνιούνται στην εφαρμογή, τις τιμές των οποίων τις κάνει request με τη μέθοδο GET, ώστε να μπορέσει στο front-end να εμφανίσει το κατάλληλο μήνυμα σε περίπτωση που ο χρήστης δεν έχει επιλέξει δεδομένα για αναζήτηση και πατήσει το κουμπί Submit (θα αναλυθεί περαιτέρω στο υποκεφάλαιο με το front-end). Στην συνέχεια έχει δηλωθεί μια μεταβλητή, η `data_list` με την οποία καλούμε μια συνάρτηση, την `select_list()`.

```
def select_list():
    output_list = []

    s = Search(using=es, index="lib-index", doc_type="lib-
data").extra(size=0)

    s.aggs.bucket('NestedAgg', 'nested',
path='datagroup').bucket('TaggroupAgg', 'terms',
field='datagroup.taggroup', size = 99999)

    response = s.execute()

    for k in response.aggregations.NestedAgg.TaggroupAgg.buckets:
        output_list.append(k['key'])

    output_list.sort()

    return output_list
```

Ο ρόλος της συνάρτησης `select_list()` είναι να βρει τις ονομασίες στα group με τα tag που έχουμε κάνει index με το Elasticsearch έτσι ώστε να τα χρησιμοποιήσουμε για να φτιάξουμε με δυναμικό τρόπο ένα μενού με dropdown λίστα με όλες τις ονομασίες που θα έχει βρει η συνάρτηση. Πιο αναλυτικά, με τη βοήθεια των aggregations στο elasticsearch αναζητούνται τα taggroups. Αρχικά στην ξεκινάμε την αναζήτηση, ορίζοντας το index και τον τύπο των εγγράφων που αναζητούνται. Ο ρόλος της extra μεταβλητής `size = 0` είναι ώστε να μην επιστραφούν όλα τα έγγραφα κατά την εκτέλεση της αναζήτησης. Στη συνέχεια ορίζουμε το κυρίως σώμα του aggregation καθώς και τον τύπο του, το οποίο στην περίπτωση μας είναι το

bucket μιας και θέλουμε να μας ομαδοποιήσει τα δεδομένα των taggroup. Μέσα στο bucket ορίζουμε τη διαδρομή για τα δεδομένα που αναζητούμε καθώς είναι εμφωλευμένα, ορίζοντας και την μεταβλητή size = 99999, ένα μεγάλο νούμερο δηλαδή για να σιγουρευτούμε ότι θα μας επιστραφούν όλα τα δεδομένα που επιθυμούμε μιας και το default στην περίπτωση που δεν ορίσουμε εμείς ένα νούμερο είναι το 10.

```
{'took': 8, 'timed_out': False, '_shards': {'total': 5, 'successful': 5, 'skipped': 0, 'failed': 0},  
'hits': {'total': 50, 'max_score': 0.0, 'hits': []}, 'aggregations': {'NestedAgg': {'doc_count': 284,  
'TaggroupAgg': {'doc_count_error_upper_bound': 0, 'sum_other_doc_count': 0, 'buckets':  
[{'key': 'index800', 'doc_count': 124}, {'key': 'index100', 'doc_count': 84}, {'key': 'index200',  
'doc_count': 50}, {'key': 'index400', 'doc_count': 14}, {'key': 'index900', 'doc_count': 8}, {'key':  
'index300', 'doc_count': 2}, {'key': 'index600', 'doc_count': 1}, {'key': 'index700', 'doc_count':  
1}]}}}}
```

Στο αποτέλεσμα του ερωτήματος που έχουμε κάνει στο Elasticsearch χρησιμοποιώντας aggregation υπάρχουν αρκετές πληροφορίες, όπως για παράδειγμα τα συνολικά έγγραφα που έχουν γίνει index στο elasticsearch, όπως επίσης και τα έγγραφα που περιέχουν στοιχεία τα οποία ταιριάζουν με τη διαδρομή που ορίσαμε. Παρατηρούμε επίσης ότι τα «hits» είναι κενά όπως ακριβώς θέλαμε, μιας και δεν χρειαζόμασταν όλα τα δεδομένα αλλά μόνο τα taggroup τα οποία βρίσκονται μέσα στα «buckets».

Στη συνέχεια, στον κώδικα, ορίζεται μια κενή λίστα οι οποία θα χρησιμοποιηθεί κατά την επεξεργασία των δεδομένων του ερωτήματος που κάναμε.

```
output_list = []
```

Συγκεκριμένα στο output_list θα υπάρχει η τελική λίστα που θα επιστρέψει η συνάρτηση.

```
for k in response.aggregations.NestedAgg.TaggroupAgg.buckets:  
    output_list.append(k['key'])  
  
output_list.sort()  
  
return output_list
```

Για να το καταφέρουμε αυτό χρειάζεται με μια επανάληψη for η οποία θα βρίσκει όλα τα δεδομένα από τα taggroup τα οποία υπάρχουν στα buckets και θα τα εκχωρεί μέσα στη λίστα

που δημιουργήσαμε. Στη συνέχεια ταξινομούμε τη λίστα `output_list` και την επιστρέφουμε πίσω στο `data_list`.

Το αποτέλεσμα που επιστρέφει η συνάρτηση `select_list` είναι το παρακάτω:

```
['index800', 'index100', 'index200', 'index400', 'index900', 'index300', 'index600', 'index700']  
-----  
['index100', 'index200', 'index300', 'index400', 'index600', 'index700', 'index800', 'index900']
```

Η πάνω λίστα είναι χωρίς να έχει ταξινομηθεί το `output_list`, ενώ το κάτω είναι η ταξινομημένη λίστα την οποία και επιστρέφει η συνάρτηση.

Στη συνέχεια, το τελευταίο πράγμα που κάνει η συνάρτηση `index_page` είναι να «χτίσει» (`render`) την σελίδα την οποία θα δει ο χρήστης. Με την βοήθεια του εργαλείου `render_template`, κάνει `render` την HTML σελίδα που φτιάχτηκε και επιστρέφει κάθε μεταβλητή που είναι απαραίτητη στο front-end.

Στην αρχική σελίδα της εφαρμογής, ο χρήστης έχει την επιλογή να κάνει την αναζήτηση που επιθυμεί. Όπως θα δούμε και στο front-end υπάρχει τρόπος να επιλέξεις από ένα μενού το `group` των `tag` ώστε να γίνει η αναζήτηση με βάση αυτό, να πληκτρολογήσουμε το στοιχείο το οποίο ψάχνουμε ώστε να γίνει αναζήτηση σε όλα τα έγγραφα με βάση αυτό, αλλά υπάρχει και ο τρόπος να γίνει αναζήτηση με συνδυασμό και των δυο τρόπων. Να ψάξουμε δηλαδή για παράδειγμα, το στοιχείο A στο `group` με τα `tag` 200. Όλο αυτό το back-end το καταφέρνει με τον παρακάτω τρόπο.

```
@table_bp.route('/search/<int:page>', methods = ["POST", "GET"])  
def form_input(page):
```

Αρχικά πρέπει να φτιαχτεί μια συνάρτηση, στη συγκεκριμένη περίπτωση το όνομα της είναι `form_input` με τη βοήθεια της οποίας θα κάνουμε όλες αυτές τις επεξεργασίες, αλλά και η διαδρομή στην οποία θα εμφανίζονται τα αποτελέσματα αυτής της συνάρτησης στον χρήστη. Στη διαδρομή έχει προστεθεί και μια μεταβλητή, η οποία είναι ακέραιος αριθμός ώστε να μπορούμε να εμφανίσουμε τα δεδομένα χωρισμένα σε σελίδες για ευκολία στον χρήστη. Για παράδειγμα αν είμαστε στην 2^η σελίδα, η διαδρομή θα γίνει «/search/2». Στην συνέχεια δηλώνονται οι μέθοδοι με τις οποίες λαμβάνουμε τα δεδομένα που εισάγει ο χρήστης. Αρχικά έχει δηλωθεί η μέθοδος `POST` ώστε να λαμβάνονται τα δεδομένα που εισάγει ο χρήστης στην φόρμα της αρχικής σελίδας, αλλά και η μέθοδος `GET` ώστε να λαμβάνονται ξανά τα δεδομένα, αυτή τη φορά μέσω του URL, κατά την αλλαγή σελίδας.

```
tag_input = request.values.get('tag_input')  
value_input = request.values.get('value_input')
```

- **tag_input:** Μεταβλητή με τιμή το taggroup.
- **value_input:** Μεταβλητή με τιμή το στοιχείο που επιθυμεί να ψάξει ο χρήστης.

Τα δεδομένα αυτά εκχωρούνται σε δυο μεταβλητές, η tag_input στην οποία εκχωρείται η ομάδα με τα tag που έχει επιλεγεί από τον χρήστη και η value_input στην οποία εκχωρείται η τιμή που έχει πληκτρολογηθεί από τον χρήστη και επιθυμούμε να ψάξουμε.

```
PER_PAGE = 5  
OFFSET = (page - 1) * PER_PAGE
```

Για να γίνει η αναζήτηση θα πρέπει να ληφθεί υπόψιν το πόσα έγγραφα θα εμφανιστούν στον χρήστη σε κάθε σελίδα αλλά και να υπάρξει ένας τρόπος ώστε να επιλέγουμε τα σωστά έγγραφα κάθε φορά κατά την αλλαγή της σελίδας. Η σταθερά PER_PAGE είναι ένας ακέραιος αριθμός στον οποίο εκχωρείται ο αριθμός των εγγράφων που θα εμφανιστεί σε κάθε σελίδα. Η μεταβλητή OFFSET είναι υπεύθυνη για να βρίσκει η εφαρμογή τα σωστά έγγραφα κάθε φορά, η τιμή της οποίας υπολογίζεται από την παρακάτω πράξη.

(page - 1) * PER_PAGE

Αν για παράδειγμα η σελίδα είναι η 1^η τότε το OFFSET θα πάρει την τιμή 0, οπότε θα ξεκινήσει η αναζήτηση από το 1^ο έγγραφο (στο elasticsearch το 1^ο έγγραφο είναι το 0). Σε περίπτωση όμως που η σελίδα είναι η 2^η τότε το OFFSET θα πάρει την τιμή 5 οπότε θα ξεκινήσει στην 2^η σελίδα να μας δείχνει το 6^ο έγγραφο από αυτά που έχει βρει στην αναζήτηση η εφαρμογή.

Για να πραγματοποιηθεί η αναζήτηση σωστά σε κάθε πιθανή επιλογή δεδομένων προς αναζήτηση που μπορεί να έκανε ο χρήστης, θα πρέπει να γίνουν κάποιοι έλεγχοι με if ώστε να μπορεί η εφαρμογή να εκτελεί το σωστό ερώτημα κάθε φορά.

```

        if tag_input and not value_input:
            res = es.search(index="lib-index", from_= OFFSET,
size= PER_PAGE, body={"query": {"nested":{"path":"datagroup",
"query":{"match": {"datagroup.taggroup": tag_input}}}}})

        elif value_input and tag_input is None:
            res = es.search(index="lib-index", from_= OFFSET,
size= PER_PAGE, body={"query": {"nested":{"path":"datagroup",
"query":{"match": {"datagroup.value": value_input}}}}})

        elif value_input and tag_input:
            res = es.search(index="lib-index", from_= OFFSET,
size= PER_PAGE, body={"query":{"nested":{"path":"datagroup",
"query":{"bool":{"must":[{"match": {"datagroup.value":
value_input}}, {"match": {"datagroup.taggroup": tag_input}}]}}}}})

        elif not value_input and tag_input is None:
            warning = True

        return redirect(url_for('form_bp.index_page',
warning=warning))

```

Για να μπορέσουμε να βρούμε τα έγγραφα τα οποία ψάχνουμε θα πρέπει να δημιουργήσουμε ένα query ώστε να τα αναζητήσει στο elasticsearch. Θα πρέπει όμως να ληφθούν υπόψιν όλες οι πιθανές ενέργειες που μπορεί να κάνει κατά την αναζήτηση ένας χρήστης. Αυτές οι ενέργειες είναι:

- Να ψάξει μόνο με βάση τις ομάδες των tag (taggroups).
- Να ψάξει μόνο με βάση το στοιχείο που θέλει να αναζητήσει σε ένα έγγραφο, σε όποια ομάδα από tag και αν βρίσκεται.
- Να ψάξει για ένα στοιχείο σε μια συγκεκριμένη ομάδα με tag.

```

        if tag_input and not value_input:
            res = es.search(index="lib-index", from_= OFFSET,
size= PER_PAGE, body={"query": {"nested":{"path":"datagroup",
"query":{"match": {"datagroup.taggroup": tag_input}}}}})

```

Το παραπάνω κομμάτι κώδικα είναι υπεύθυνο να κάνει την αναζήτηση με βάση τις ομάδες των tag. Θα πρέπει αρχικά να γίνει έλεγχος, ένα if δηλαδή με την ακόλουθη συνθήκη: αν η μεταβλητή tag_input έχει κάποια τιμή και η μεταβλητή value_input δεν έχει καμία, δεν έχει πληκτρολογήσει δηλαδή τίποτα ο χρήστης. Στην περίπτωση που επιβεβαιώνεται η συνθήκη, τότε πραγματοποιείται μια αναζήτηση στο elasticsearch για να βρει τα έγγραφα τα οποία επιθυμούμε. Όπως στο προηγούμενο ερώτημα, έτσι και εδώ θα πρέπει να δηλώσουμε το όνομα που έχει το index, το from_ είναι υπεύθυνο για να ορίσει το όριο των εγγράφων που θα επιστρέψει το query, με λίγα λόγια βρίσκει το πρώτο έγγραφο που θα επιστρέψει η αναζήτηση, το OFFSET ουσιαστικά που εξηγήθηκε πιο πάνω και το size είναι ο συνολικός αριθμός των εγγράφων που θα επιστραφούν, το PER_PAGE ουσιαστικά που έχει οριστεί πιο

πάνω. Στην συνέχεια στο body γράφουμε το ερώτημα που θέλουμε να κάνει η εφαρμογή στο elasticsearch, το οποίο προσπαθεί να ταιριάξει το taggroup των εγγράφων με το tag που έχει επιλέξει ο χρήστης πιο πριν και εκχωρήθηκε στο tag_input και τα αποτελέσματα αυτού του ερωτήματος τα εκχωρούμε στη μεταβλητή res. Για παράδειγμα αν αναζητήσουμε για το taggroup 600 το αποτέλεσμα του query θα έχει την παρακάτω μορφή και θα εκχωρηθεί ολόκληρο στην μεταβλητή res.

```
{'_shards': {'failed': 0, 'total': 5, 'successful': 5, 'skipped': 0}, 'timed_out': False, 'hits': {'total': 1, 'max_score': 4.1163235, 'hits': [{'_index': 'lib-index', '_id': 'AXmqW0E66ukx1HPjvl9Y', '_score': 4.1163235, '_type': 'lib-data', '_source': {'datagroup': [{'taggroup': 'index100', 'value': '__ $a 20050519agrey010305 ga'}, {'taggroup': 'index100', 'value': '__ $a AACR2 2nd rev. ed.'}, {'taggroup': 'index200', 'value': '_0 $a Πλάτων $f 427-347 π.Χ. '}, {'taggroup': 'index400', 'value': '_1 $a Πλατωνική φιλοσοφία '}, {'taggroup': 'index400', 'value': '_1 $a Πλατωνισμός , Αρχαίος '}, {'taggroup': 'index400', 'value': '_1 $a Plato '}, {'taggroup': 'index400', 'value': '_1 $a Platon '}, {'taggroup': 'index600', 'value': '__ $a 184 $c Platonic philosophy $v 23η έκδ. '}, {'taggroup': 'index700', 'value': '_1 $a Plato $2 lcna $3 http://id.loc.gov/authorities/names/n79139459 '}, {'taggroup': 'index800', 'value': '_0 $a GR $b Ionian University $c 20100614 '}, {'taggroup': 'index800', 'value': '__ $a Republic, 1993 $b t.p. (Plato) '}, {'taggroup': 'index800', 'value': '__ $a Oeuvres completes, 1997- $b t.p. (Platon) '}, {'taggroup': 'index800', 'value': '__ $a E.B.E. '}, {'taggroup': 'index800', 'value': '__ $a L.C. '}, {'taggroup': 'index800', 'value': '__ $a Παγκόσμιο βιογραφικό λεξικό, 1990-1991 $b τ.8 (Πλάτων· 427 - 347 π.Χ., Αθήνα· φιλόσοφος, ο κορυφαίος της ανθρωπότητας) '}, {'taggroup': 'index800', 'value': '4_ $u http://viaf.org/viaf/108159964 $x viaf '}, {'taggroup': 'index900', 'value': '__ $a Aslanidi $z 2010-06 '}], 'record': [{'data': 'ionioa10001644', 'tag': 1}, {'data': 'ionio', 'tag': 3}, {'data': '20050519203344.0', 'tag': 5}, {'subfields': [{'code': 'z', 'data': '(ionio)ionioa10001644'}, {'tag': 35, 'ind1': ' ', 'ind2': ' '}, {'subfields': [{'code': 'a', 'data': '20050519agrey010305 ga'}, {'tag': 100, 'ind1': ' ', 'ind2': ' '}, {'subfields': [{'code': 'a', 'data': 'AACR2 2nd rev. ed.'}, {'tag': 152, 'ind1': ' ', 'ind2': ' '}, {'subfields': [{'code': 'a', 'data': 'Πλάτων'}, {'code': 'f', 'data': '427-347 π.Χ.'}, {'tag': 200, 'ind1': ' ', 'ind2': '0'}, {'subfields': [{'code': 'a', 'data': 'Πλατωνική φιλοσοφία'}, {'tag': 400, 'ind1': ' ', 'ind2': '1'}, {'subfields': [{'code': 'a', 'data': 'Πλατωνισμός , Αρχαίος'}, {'tag': 400, 'ind1': ' ', 'ind2': '1'}, {'subfields': [{'code': 'a', 'data': 'Plato'}, {'tag': 400, 'ind1': ' ', 'ind2': '1'}, {'subfields': [{'code': 'a', 'data': 'Platon'}, {'tag': 400, 'ind1': ' ', 'ind2': '1'}, {'subfields': [{'code': 'a', 'data': '184'}, {'code': 'c', 'data': 'Platonic philosophy'}, {'code': 'v', 'data': '23η έκδ.'}, {'tag': 676, 'ind1': ' ', 'ind2': ' '}, {'subfields': [{'code': 'a', 'data': 'Plato'}, {'code': '2', 'data': 'lcna'}, {'code': '3', 'data': 'http://id.loc.gov/authorities/names/n79139459'}, {'tag': 700, 'ind1': ' ', 'ind2': '1'}, {'subfields': [{'code': 'a', 'data': 'GR'}, {'code': 'b', 'data': 'Ionian University'}, {'code': 'c', 'data': '20100614'}, {'tag': 801, 'ind1': ' ', 'ind2': '0'}, {'subfields': [{'code': 'a', 'data': 'Republic, 1993'}, {'code': 'b', 'data': 't.p. (Plato)'}], 'tag': 810, 'ind1': ' ', 'ind2': ' '}, {'subfields': [{'code': 'a', 'data': 'Oeuvres completes, 1997-'}, {'code': 'b', 'data': 't.p. (Platon)'}], 'tag': 810, 'ind1': ' ', 'ind2': ' '}, {'subfields': [{'code': 'a', 'data': 'E.B.E.'}, {'tag': 810, 'ind1': ' ', 'ind2': ' '}, {'subfields': [{'code': 'a', 'data': 'L.C.'}, {'tag': 810, 'ind1': ' ', 'ind2': ' '}, {'subfields': [{'code': 'a', 'data': 'Παγκόσμιο βιογραφικό λεξικό, 1990-1991'}, {'code': 'b', 'data': 'τ.8 (Πλάτων· 427 - 347 π.Χ., Αθήνα· φιλόσοφος, ο κορυφαίος της ανθρωπότητας)'}], 'tag': 810, 'ind1': ' ', 'ind2': ' '}, {'subfields': [{'code': 'u', 'data': 'http://viaf.org/viaf/108159964'}, {'code': 'x', 'data': 'viaf'}], 'tag': 856, 'ind1': '4', 'ind2': ' '}, {'subfields': [{'code': 'a', 'data': 'Aslanidi'}, {'code': 'z', 'data': '2010-06'}], 'tag': 970, 'ind1': ' ', 'ind2': ' '}]}}], 'took': 3}
```

Με λίγα λόγια βρίσκει όλα τα έγγραφα που έχουν κάποιο στοιχείο στην συγκεκριμένη ομάδα με tags.

Στη δεύτερη περίπτωση που ο χρήστης θελήσει να κάνει μια αναζήτηση με βάση μιας τιμής που θα πληκτρολογήσει στο κατάλληλο πεδίο θα πρέπει να γίνει η αναζήτηση λίγο διαφορετικά.

```
elif value_input and tag_input is None:
    res = es.search(index="lib-index", from_= OFFSET,
size= PER_PAGE, body={"query": {"nested":{"path":"datagroup",
"query":{"match": {"datagroup.value": value_input}}}}})
```

Αρχικά θα πρέπει να κάνουμε ένα διαφορετικό έλεγχο. Σε αυτή τη περίπτωση θα πρέπει να ελέγχουμε εάν έχει εκχωρηθεί μια τιμή στο value_input, την οποία την έχει εξάγει από το request ο κώδικας του back-end και την έχει εκχωρήσει στη συγκεκριμένη μεταβλητή, όπως επίσης θα πρέπει να ελέγχουμε ότι το tag_input έχει την τιμή **None**, το οποίο μας βεβαιώνει ότι ο χρήστης δεν έχει επιλέξει καμία τιμή από το dropdown μενού των tags. Στο query το οποίο γράφτηκε, το μόνο στοιχείο που αλλάζει είναι η τιμή που πρέπει να ταιριάζει. Σε αυτή τη περίπτωση, το query πρέπει να ταιριάζει το value_input με το value που υπάρχει στη λίστα datagroup στα δεδομένα που έχουν γίνει index στο elasticsearch.

Για παράδειγμα, αν κάνουμε μια αναζήτηση για το στοιχείο «Platon», το αποτέλεσμα του ερωτήματος θα είναι το παρακάτω:

```
{'_shards': {'failed': 0, 'successful': 5, 'total': 5, 'skipped': 0}, 'timed_out': False, 'took': 6, 'hits':
{'max_score': 3.7786279, 'total': 1, 'hits': [{'_index': 'lib-index', '_source': {'record': [{'data':
'ionioa10001644', 'tag': 1}, {'data': 'ionio', 'tag': 3}, {'data': '20050519203344.0', 'tag': 5},
{'ind2': ' ', 'subfields': [{'code': 'z', 'data': '(ionio)ionioa10001644'}], 'tag': 35, 'ind1': ' '}, {'ind2':
' ', 'subfields': [{'code': 'a', 'data': '20050519agrey010305 ga'}], 'tag': 100, 'ind1': ' '}, {'ind2': '
', 'subfields': [{'code': 'a', 'data': 'AACR2 2nd rev. ed.}], 'tag': 152, 'ind1': ' '}, {'ind2': '0',
'subfields': [{'code': 'a', 'data': 'Πλάτων'}, {'code': 'f', 'data': '427-347 π.Χ.}], 'tag': 200, 'ind1': '
'}, {'ind2': '1', 'subfields': [{'code': 'a', 'data': 'Πλατωνική φιλοσοφία'}], 'tag': 400, 'ind1': ' '},
{'ind2': '1', 'subfields': [{'code': 'a', 'data': 'Πλατωνισμός , Αρχαίος'}], 'tag': 400, 'ind1': ' '},
{'ind2': '1', 'subfields': [{'code': 'a', 'data': 'Plato'}], 'tag': 400, 'ind1': ' '}, {'ind2': '1', 'subfields':
[{'code': 'a', 'data': 'Platon'}], 'tag': 400, 'ind1': ' '}, {'ind2': ' ', 'subfields': [{'code': 'a', 'data':
'184'}, {'code': 'c', 'data': 'Platonic philosophy'}, {'code': 'v', 'data': '23η έκδ.}], 'tag': 676, 'ind1':
' '}, {'ind2': '1', 'subfields': [{'code': 'a', 'data': 'Plato'}, {'code': '2', 'data': 'lcna'}, {'code': '3',
'data': 'http://id.loc.gov/authorities/names/n79139459'}], 'tag': 700, 'ind1': ' '}, {'ind2': '0',
'subfields': [{'code': 'a', 'data': 'GR'}, {'code': 'b', 'data': 'Ionian University'}, {'code': 'c', 'data':
'20100614'}], 'tag': 801, 'ind1': ' '}, {'ind2': ' ', 'subfields': [{'code': 'a', 'data': 'Republic, 1993'},
{'code': 'b', 'data': 't.p. (Plato)'}], 'tag': 810, 'ind1': ' '}, {'ind2': ' ', 'subfields': [{'code': 'a', 'data':
'Oeuvres completes, 1997-'}, {'code': 'b', 'data': 't.p. (Platon)'}], 'tag': 810, 'ind1': ' '}, {'ind2': ' ',
'subfields': [{'code': 'a', 'data': 'E.B.E.}], 'tag': 810, 'ind1': ' '}, {'ind2': ' ', 'subfields': [{'code': 'a',
'data': 'L.C.}], 'tag': 810, 'ind1': ' '}, {'ind2': ' ', 'subfields': [{'code': 'a', 'data': 'Παγκόσμιο
βιογραφικό λεξικό, 1990-1991'}, {'code': 'b', 'data': 'τ.8 (Πλάτων· 427 - 347 π.Χ., Αθήνα·
φιλόσοφος, ο κορυφαίος της ανθρωπότητας)}], 'tag': 810, 'ind1': ' '}, {'ind2': ' ', 'subfields':
[{'code': 'u', 'data': 'http://viaf.org/viaf/108159964'}, {'code': 'x', 'data': 'viaf'}], 'tag': 856,
'ind1': '4'}, {'ind2': ' ', 'subfields': [{'code': 'a', 'data': 'Aslanidi'}, {'code': 'z', 'data': '2010-06'}],
'tag': 970, 'ind1': ' '}], 'datagroup': [{'taggroup': 'index100', 'value': '___ $a
20050519agrey010305 ga '}, {'taggroup': 'index100', 'value': '___ $a AACR2 2nd rev. ed. '},
{'taggroup': 'index200', 'value': '___0 $a Πλάτων $f 427-347 π.Χ. '}, {'taggroup': 'index400',
'value': '___1 $a Πλατωνική φιλοσοφία '}, {'taggroup': 'index400', 'value': '___1 $a Πλατωνισμός ,
Αρχαίος '}, {'taggroup': 'index400', 'value': '___1 $a Plato '}, {'taggroup': 'index400', 'value': '___1
$a Platon '}, {'taggroup': 'index600', 'value': '___ $a 184 $c Platonic philosophy $v 23η έκδ. '},
{'taggroup': 'index700', 'value': '___1 $a Plato $2 lcna $3
http://id.loc.gov/authorities/names/n79139459 '}, {'taggroup': 'index800', 'value': '___0 $a GR
$b Ionian University $c 20100614 '}, {'taggroup': 'index800', 'value': '___ $a Republic, 1993 $b
t.p. (Plato) '}, {'taggroup': 'index800', 'value': '___ $a Oeuvres completes, 1997- $b t.p. (Platon)
'}, {'taggroup': 'index800', 'value': '___ $a E.B.E. '}, {'taggroup': 'index800', 'value': '___ $a L.C. '},
{'taggroup': 'index800', 'value': '___ $a Παγκόσμιο βιογραφικό λεξικό, 1990-1991 $b τ.8
(Πλάτων· 427 - 347 π.Χ., Αθήνα· φιλόσοφος, ο κορυφαίος της ανθρωπότητας) '}, {'taggroup':
'index800', 'value': '4_ $u http://viaf.org/viaf/108159964 $x viaf '}, {'taggroup': 'index900',
'value': '___ $a Aslanidi $z 2010-06 '}}], '_id': 'AXmqW0E66ukx1HPjvl9Y', '_score': 3.7786279,
'_type': 'lib-data']}]}
```

Ουσιαστικά βρίσκει κάθε έγγραφο, στη συγκεκριμένη περίπτωση είναι ένα, στο οποίο υπάρχει το στοιχείο που αναζητούμε, σε όποιο taggroup και αν βρίσκεται.

Η τελευταία περίπτωση αναζήτησης που μπορεί να πραγματοποιηθεί είναι ο χρήστης να ψάξει για μια τιμή σε ένα συγκεκριμένο group από tag. Σε αυτό μας βοηθάει το παρακάτω κομμάτι κώδικα.

```
elif value_input and tag_input:
    res = es.search(index="lib-index", from_= OFFSET,
size= PER_PAGE, body={"query":{"nested":{"path":"datagroup",
"query":{"bool":{"must":[{"match": {"datagroup.value":
value_input}},"match": {"datagroup.taggroup": tag_input}}]}}}}})
```

Όπως και πριν, έτσι και τώρα θα πρέπει να γίνει ένας έλεγχος με μια if για να βεβαιωθούμε ότι θα πληρούνται όλες οι προϋποθέσεις για να εκτελεστεί το ερώτημα που θα γράψαμε. Ο έλεγχος θα πρέπει να γίνει, ώστε να επιβεβαιωθεί ότι τόσο το value_input, όσο και το tag_input έχουν μια τιμή, ο χρήστης δηλαδή έχει επιλέξει ένα group από tag μέσω του dropdown μενού αλλά έχει πληκτρολογήσει και μια τιμή στο κατάλληλο πεδίο.

Σε αυτή την περίπτωση το query θα είναι διαφορετικό και ελαφρώς πιο περίπλοκο. Αρχικά πάλι πρέπει να ορίσουμε το όνομα του index στο οποίο θα ψάξει ο κώδικας, να οριστεί το from_ και το size ώστε να βεβαιωθούμε ότι θα γυρίσει τον επιθυμητό αριθμό από έγγραφα για να μπορέσουμε να τα χωρίσουμε σε σελίδες ώστε να είναι όμορφο το αποτέλεσμα στον χρήστη και φιλικό σε αυτόν και τέλος να ορίσουμε το κύριο σώμα του ερωτήματος μας. Σε αυτό θα πρέπει να οριστεί το path στο οποίο θα πούμε στο ερώτημα, το σημείο στο οποίο θα ψάξει για τα δεδομένα που ζητάμε και στη συνέχεια θα πρέπει να οριστεί το ερώτημα ως bool ώστε να βάλουμε την προϋπόθεση «must», για να βεβαιωθούμε ότι το ερώτημα είναι υποχρεωτικό να ταιριάζει και τα δυο δεδομένα ταυτόχρονα, ώστε να τα επιστρέψει πίσω στην εφαρμογή. Στο τελευταίο κομμάτι του ερωτήματος αναφέρουμε τα δεδομένα που θέλουμε να ταιριάζουμε (match) με τα αντίστοιχα πεδία των δεδομένων που έχουν γίνει index στο elasticsearch.

Για παράδειγμα αν κάνουμε μια αναζήτηση για το στοιχείο «Composer» στο taggroup 300 το αποτέλεσμα του παραπάνω ερωτήματος θα είναι το παρακάτω:

```
{'timed_out': False, 'took': 3, 'hits': {'max_score': 9.507305, 'hits': [{'_index': 'lib-index', '_score': 9.507305, '_type': 'lib-data', '_source': {'record': [{'tag': 1, 'data': 'ionioa10001987'}, {'tag': 3, 'data': 'ionio'}, {'tag': 5, 'data': '20061218103955.0'}, {'subfields': [{'data': '(ionio)ionioa10001987', 'code': 'z'}]}, {'tag': 35, 'ind1': '', 'ind2': ''}, {'subfields': [{'data': '20050519agrey010303 ba', 'code': 'a'}]}, {'tag': 100, 'ind1': '', 'ind2': ''}, {'subfields': [{'data': '0', 'code': 'a'}]}, {'tag': 106, 'ind1': '', 'ind2': ''}, {'subfields': [{'data': 'ba', 'code': 'a'}]}, {'tag': 120, 'ind1': '', 'ind2': ''}, {'subfields': [{'data': 'AACR2 2nd rev. ed.', 'code': 'a'}, {'data': 'lcna', 'code': 'b'}]}, {'tag': 152, 'ind1': '', 'ind2': ''}, {'subfields': [{'data': 'Stravinsky', 'code': 'a'}, {'data': 'Igor', 'code': 'b'}, {'data': '1882-1971', 'code': 'f'}]}, {'tag': 200, 'ind1': '', 'ind2': '1'}, {'subfields': [{'data': 'Composer', 'code': 'a'}]}, {'tag': 340, 'ind1': '', 'ind2': ''}, {'subfields': [{'data': 'GR', 'code': 'a'}, {'data': 'Ionian University', 'code': 'b'}, {'data': '20090220', 'code': 'c'}]}, {'tag': 801, 'ind1': '', 'ind2': '0'}, {'subfields': [{'data': 'http://viaf.org/viaf/17309155', 'code': 'u'}, {'data': 'viaf', 'code': 'x'}]}, {'tag': 856, 'ind1': '4', 'ind2': ''}, {'subfields': [{'data': 'http://id.loc.gov/authorities/names/n79070061', 'code': 'u'}, {'data': 'lcna', 'code': 'x'}]}, {'tag': 856, 'ind1': '4', 'ind2': ''}, {'subfields': [{'data': 'Aslanidi', 'code': 'a'}, {'data': '2014-01', 'code': 'z'}]}, {'tag': 970, 'ind1': '', 'ind2': ''}], 'datagroup': [{'value': '__ $a 20050519agrey010303 ba ', 'taggroup': 'index100'}, {'value': '__ $a 0 ', 'taggroup': 'index100'}, {'value': '__ $a ba ', 'taggroup': 'index100'}, {'value': '__ $a AACR2 2nd rev. ed. $b lcna ', 'taggroup': 'index100'}, {'value': '_1 $a Stravinsky $b Igor $f 1882-1971 ', 'taggroup': 'index200'}, {'value': '__ $a Composer ', 'taggroup': 'index300'}, {'value': '_0 $a GR $b Ionian University $c 20090220 ', 'taggroup': 'index800'}, {'value': '4_ $u http://viaf.org/viaf/17309155 $x viaf ', 'taggroup': 'index800'}, {'value': '4_ $u http://id.loc.gov/authorities/names/n79070061 $x lcna ', 'taggroup': 'index800'}, {'value': '__ $a Aslanidi $z 2014-01 ', 'taggroup': 'index900'}]}, '_id': 'AXmqWOE56ukx1HPjvI9G', '_index': 'lib-index', '_score': 8.093746, '_type': 'lib-data', '_source': {'record': [{'tag': 1, 'data': 'ionioa10002345'}, {'tag': 3, 'data': 'ionio'}, {'tag': 5, 'data': '20050519203356.0'}, {'subfields': [{'data': '(ionio)ionioa10002345', 'code': 'z'}]}, {'tag': 35, 'ind1': '', 'ind2': ''}, {'subfields': [{'data': '20050519agrey010303 ba', 'code': 'a'}]}, {'tag': 100, 'ind1': '', 'ind2': ''}, {'subfields': [{'data': 'ita', 'code': 'a'}]}, {'tag': 101, 'ind1': '', 'ind2': ''}, {'subfields': [{'data': 'IT', 'code': 'a'}]}, {'tag': 102, 'ind1': '', 'ind2': ''}, {'subfields': [{'data': '0', 'code': 'a'}]}, {'tag': 106, 'ind1': '', 'ind2': ''}, {'subfields': [{'data': 'ba', 'code': 'a'}]}, {'tag': 120, 'ind1': '', 'ind2': ''}, {'subfields': [{'data': 'RDA', 'code': 'a'}, {'data': 'lcna', 'code': 'b'}]}, {'tag': 152, 'ind1': '', 'ind2': ''}, {'subfields': [{'data': 'Donizetti', 'code': 'a'}, {'data': 'Gaetano', 'code': 'b'}, {'data': '1797-1848', 'code': 'f'}]}, {'tag': 200, 'ind1': '', 'ind2': '1'}, {'subfields': [{'data': 'Composer', 'code': 'a'}]}, {'tag': 340, 'ind1': '', 'ind2': ''}, {'subfields': [{'data': 'Donizetti', 'code': 'a'}, {'data': 'G.', 'code': 'b'}, {'data': '1797-1848', 'code': 'f'}, {'data': '(Gaetano)', 'code': 'g'}]}, {'tag': 400, 'ind1': '', 'ind2': '1'}, {'subfields': [{'data': 'GR', 'code': 'a'}, {'data': 'Ionian University', 'code': 'b'}, {'data': '20090604', 'code': 'c'}]}, {'tag': 801, 'ind1': '', 'ind2': '0'}, {'subfields': [{'data': 'La favorita, 19--', 'code': 'a'}, {'data': 't.p. (G. Donizetti)', 'code': 'b'}]}, {'tag': 810, 'ind1': '', 'ind2': ''}, {'subfields': [{'data': 'L.C.', 'code': 'a'}]}, {'tag': 810, 'ind1': '', 'ind2': ''}, {'subfields': [{'data': 'http://viaf.org/viaf/74037336', 'code': 'u'}, {'data': 'viaf', 'code': 'x'}]}, {'tag': 856, 'ind1': '4', 'ind2': ''}, {'subfields': [{'data': 'http://id.loc.gov/authorities/names/n79132358', 'code': 'u'}, {'data': 'lcna', 'code': 'x'}]}, {'tag': 856, 'ind1': '4', 'ind2': ''}, {'subfields': [{'data': 'Aslanidi', 'code': 'a'}, {'data': '2013-07', 'code': 'z'}]}, {'tag': 970, 'ind1': '', 'ind2': ''}], 'datagroup': [{'value': '__ $a 20050519agrey010303 ba ', 'taggroup': 'index100'}, {'value': '__ $a ita ', 'taggroup': 'index100'}, {'value': '__ $a IT ', 'taggroup': 'index100'}, {'value': '__ $a 0 ', 'taggroup': 'index100'}, {'value': '__ $a ba ', 'taggroup': 'index100'}, {'value': '__ $a RDA $b lcna ', 'taggroup': 'index100'}, {'value': '_1 $a Donizetti $b Gaetano $f 1797-1848 ', 'taggroup': 'index200'}, {'value': '__ $a Composer ', 'taggroup': 'index300'}, {'value': '_1 $a Donizetti $b G. $f 1797-1848 $g (Gaetano) ', 'taggroup': 'index400'}, {'value': '_0 $a GR $b Ionian University $c 20090604 ', 'taggroup': 'index800'}, {'value': '__ $a La favorita, 19-- $b t.p. (G. Donizetti) ', 'taggroup': 'index800'}, {'value': '__ $a L.C. ', 'taggroup': 'index800'}, {'value': '4_ $u http://viaf.org/viaf/74037336 $x viaf ', 'taggroup': 'index800'}, {'value': '4_ $u http://id.loc.gov/authorities/names/n79132358 $x lcna ', 'taggroup': 'index800'}, {'value': '__ $a Aslanidi $z 2013-07 ', 'taggroup': 'index900'}]}, '_id': 'AXmqWOE66ukx1HPjvI9Z', 'total': 2, '_shards': {'successful': 5, 'failed': 0, 'skipped': 0, 'total': 5}}
```

Το ερώτημα έχει βρει όλα τα έγγραφα με το συγκεκριμένο στοιχείο και το αποτέλεσμα το εκχώρησε στην μεταβλητή `res`.

Υπάρχει όμως και η περίπτωση ο χρήστης να πατήσει το κουμπί `Submit` χωρίς να έχει επιλέξει στοιχεία. Θα πρέπει έχουμε λάβει μέτρα στον κώδικα ώστε να αποφευχθεί αυτή η περίπτωση αλλά να ενημερώσουμε τον χρήστη ότι υπήρξε λάθος με την ενέργεια που πραγματοποιήθηκε.

```
elif not value_input and tag_input is None:
    warning = True

return redirect(url_for('form_bp.index_page',
warning=warning))
```

- **warning:** Λογική μεταβλητή η τιμή της οποίας θα μεταφερθεί όπως είδαμε πιο πάνω μέσα από το URL με την μέθοδο GET.

Το παραπάνω κομμάτι κώδικα είναι ο τελευταίος έλεγχος που πρέπει να γίνει μετά την σειρά από `if` που υπήρξαν για να γίνουν τα ερωτήματα στο `elasticsearch`. Συγκεκριμένα σε αυτό το κομμάτι θα πρέπει να ελεγχθεί ότι το `value_input` δεν έχει κάποια τιμή, δεν έχει πληκτρολογηθεί από τον χρήστη δηλαδή, αλλά και το `tag_input` ότι είναι **None** η οποία είναι η τιμή της μεταβλητής όταν δεν έχει επιλεγθεί `group` από `tag` από το `dropdown` μενού. Εάν υπάρχουν οι παραπάνω προϋποθέσεις, γίνεται η ανακατεύθυνση στην αρχική σελίδα της εφαρμογής στην οποία περνάμε μέσω του URL και μια Boolean μεταβλητή με την τιμή `True` η οποία στην συνέχεια θα μας βοηθήσει στο frontend να εμφανίσουμε το κατάλληλο μήνυμα λάθους.

Στον κώδικα του back-end θα πρέπει να προληφθεί η περίπτωση η αναζήτηση που έχει γίνει από τον χρήστη να μην έχει κανένα αποτέλεσμα. Αν για παράδειγμα αντί να ψάξουμε το στοιχείο «Composer» στο `taggroup 300`, το αναζητήσουμε στο `taggroup 100`, το αποτέλεσμα και στη συνέχεια η τιμή που θα έπαιρνε η μεταβλητή `res` θα ήταν:

```
{'_shards': {'skipped': 0, 'total': 5, 'failed': 0, 'successful': 5}, 'took': 5, 'hits': {'total': 0, 'max_score': None, 'hits': []}, 'timed_out': False}
```

Παρατηρούμε ότι δεν έχει επιστραφεί κανένα έγγραφο και η τιμή `total` στα `hits` είναι μηδενική. Με βάση αυτό το αποτέλεσμα γράφτηκε ο παρακάτω κώδικας για να αποφευχθεί ένα τέτοιο συμβάν.

```

        if res['hits']['total'] == 0:

            no_result = True

            return redirect(url_for('form_bp.index_page',
no_result=no_result))
        else:
            no_result = False

```

Πρώτα πρέπει να γίνει ένας έλεγχος με μια if εάν το στοιχείο total, ο συνολικός αριθμός των εγγράφων που έχει βρει το elasticsearch δηλαδή, το οποίο βρίσκεται μέσα στα hits είναι μηδενικό. Στην περίπτωση που επιβεβαιώνεται η συνθήκη πάει να πει ότι δεν έχει επιστραφεί κανένα έγγραφο. Αρχικά δημιουργούμε μια Boolean μεταβλητή, η μεταβλητή no_result με τιμή True την οποία θα επιστρέψουμε με την ανακατεύθυνση στην αρχική σελίδα μέσω του URL ώστε να εμφανίσουμε στο front-end το κατάλληλο μήνυμα λάθους. Στην περίπτωση που δεν ισχύει η συνθήκη που έχουμε γράψει, δηλαδή υπάρχουν έγγραφα στο αποτέλεσμα της αναζήτησης, η μεταβλητή no_result παίρνει την τιμή False ώστε να συνεχίσουμε στην επεξεργασία των δεδομένων που έχουν επιστραφεί.

Αφού ολοκληρωθεί η διαδικασία των ερωτημάτων στο elasticsearch ώστε να βρούμε τα έγγραφα που ψάχνει ο χρήστης θα πρέπει να επεξεργαστούμε το αποτέλεσμα που γυρνάει ώστε να βρούμε τις απαραίτητες πληροφορίες που θα εμφανίσουμε στον χρήστη.

```

if not no_result:
    res2 = []
    tag_list_1 = []
    tag_list_2 = []
    data_list_1 = []
    data_list_2 = []

    for hit in res['hits']['hits']:
        temp = {}
        for doc_record in hit['_source']:
            temp.update(hit['_source'])

    res2.append(temp)

```

Αρχικά πριν ξεκινήσει η διαδικασία επεξεργασίας των δεδομένων θα πρέπει να βεβαιωθούμε ότι υπάρχει αποτέλεσμα. Σε αυτό θα μας βοηθήσει η μεταβλητή no_result η οποία θα πάρει την τιμή False μόνο αν υπάρχει αποτέλεσμα από κάποιο από τα query που έχουν γραφτεί. Σε περίπτωση που επιβεβαιωθεί η συνθήκη not no_result, δηλαδή εάν είναι True, αρχικοποιούμε μερικές κενές λίστες οι οποίες θα μας βοηθήσουν στην συνέχεια για να γίνει

η διαλογή των απαραίτητων δεδομένων για προβολή στον χρήστη. Οι πληροφορίες για τα έγγραφα τα οποία ψάχνει ο χρήστης βρίσκονται μέσα σε εμφωλευμένα dictionaries τα οποία για να τα προσπελάσουμε θα πρέπει να υπάρχει μια επανάληψη for. Τα έγγραφα τα οποία πρέπει να εμφανιστούν βρίσκονται μέσα σε dictionary με key το `_source` όπου πάλι πρέπει να υπάρχει μια επανάληψη ώστε να φτάσουμε στα ίδια τα έγγραφα. Για ευκολία στην συνέχεια της διαδικασίας, κάθε στοιχείο που περιέχουν τα `_source` dictionaries τα εκχωρούμε σε ένα dictionary με όνομα `temp` το οποίο και στο τέλος το εισάγουμε σε μια από τις λίστες που έχουμε αρχικοποιήσει πιο πάνω.

Για παράδειγμα αν κάνουμε μια αναζήτηση για το στοιχείο «Bach» το αποτέλεσμα της μεταβλητής `res2`, η οποία περιέχει ολόκληρο το έγγραφο που περιλαμβάνει το στοιχείο που ψάχνουμε θα είναι το παρακάτω:

```
[{'record': [{'data': 'ionioa10002993', 'tag': 1}, {'data': 'ionio', 'tag': 3}, {'data': '20050519203407.0', 'tag': 5}, {'ind2': ' ', 'ind1': ' ', 'subfields': [{'code': 'z', 'data': '(ionio)ionioa10002993', 'tag': 35}, {'ind2': ' ', 'ind1': ' ', 'subfields': [{'code': 'a', 'data': '20050519agrey010301 ba', 'tag': 100}, {'ind2': ' ', 'ind1': ' ', 'subfields': [{'code': 'a', 'data': 'ger', 'tag': 101}, {'ind2': ' ', 'ind1': ' ', 'subfields': [{'code': 'a', 'data': 'DE', 'tag': 102}, {'ind2': ' ', 'ind1': ' ', 'subfields': [{'code': 'a', 'data': '0', 'tag': 106}, {'ind2': ' ', 'ind1': ' ', 'subfields': [{'code': 'a', 'data': 'ba', 'tag': 120}, {'ind2': ' ', 'ind1': ' ', 'subfields': [{'code': 'a', 'data': 'AACR2 2nd rev. ed.', 'code': 'b', 'data': 'lcna', 'tag': 152}, {'ind2': '1', 'ind1': ' ', 'subfields': [{'code': 'a', 'data': 'Bach', 'code': 'b', 'data': 'Johann Sebastian', 'code': 'f', 'data': '1685-1750', 'tag': 200}, {'ind2': '1', 'ind1': ' ', 'subfields': [{'code': 'a', 'data': 'Bach', 'code': 'b', 'data': 'J. S.', 'code': 'f', 'data': '1685-1750', 'tag': 400}, {'ind2': '1', 'ind1': ' ', 'subfields': [{'code': 'a', 'data': 'Bach', 'code': 'b', 'data': 'Gio. Seb.', 'code': 'f', 'data': '1685-1750', 'tag': 400}, {'ind2': '1', 'ind1': ' ', 'subfields': [{'code': 'a', 'data': 'Bach', 'code': 'b', 'data': 'G. S.', 'code': 'f', 'data': '1685-1750', 'tag': 400}, {'ind2': '1', 'ind1': ' ', 'subfields': [{'code': 'a', 'data': 'Μπαχ', 'code': 'f', 'data': '1685-1750', 'tag': 400}, {'ind2': '0', 'ind1': ' ', 'subfields': [{'code': 'a', 'data': 'GR', 'code': 'b', 'data': 'Ionian University', 'code': 'c', 'data': '20090514', 'tag': 801}, {'ind2': ' ', 'ind1': ' ', 'subfields': [{'code': 'a', 'data': 'Air on a G string, c1994', 'code': 'b', 'data': 't.p. (J. S. Bach)', 'tag': 810}, {'ind2': ' ', 'ind1': ' ', 'subfields': [{'code': 'a', 'data': 'Composizioni, [18--]', 'code': 'b', 'data': 't.p. (Gio. Seb. Bach)', 'tag': 810}, {'ind2': ' ', 'ind1': ' ', 'subfields': [{'code': 'a', 'data': 'Il clavicembalo bem temperato, 1946', 'code': 'b', 'data': 't.p. (G. S. Bach)', 'tag': 810}, {'ind2': ' ', 'ind1': ' ', 'subfields': [{'code': 'a', 'data': 'Μπαχ [SR] 2001', 'tag': 810}, {'ind2': ' ', 'ind1': ' ', 'subfields': [{'code': 'a', 'data': 'L.C.', 'tag': 810}, {'ind2': ' ', 'ind1': '4', 'subfields': [{'code': 'u', 'data': 'http://viaf.org/viaf/12304462', 'code': 'x', 'data': 'viaf', 'tag': 856}, {'ind2': ' ', 'ind1': '4', 'subfields': [{'code': 'u', 'data': 'http://id.loc.gov/authorities/names/n79021425', 'code': 'x', 'data': 'lcna', 'tag': 856}, {'ind2': ' ', 'ind1': ' ', 'subfields': [{'code': 'a', 'data': 'Aslanidi', 'code': 'z', 'data': '2009-09', 'tag': 970}], 'datagroup': {'value': '__ $a 20050519agrey010301 ba ', 'taggroup': 'index100'}, {'value': '__ $a ger ', 'taggroup': 'index100'}, {'value': '__ $a DE ', 'taggroup': 'index100'}, {'value': '__ $a 0 ', 'taggroup': 'index100'}, {'value': '__ $a ba ', 'taggroup': 'index100'}, {'value': '__ $a AACR2 2nd rev. ed. $b lcna ', 'taggroup': 'index100'}, {'value': '_1 $a Bach $b Johann Sebastian $f 1685-1750 ', 'taggroup': 'index200'}, {'value': '_1 $a Bach $b J. S. $f 1685-1750 ', 'taggroup': 'index400'}, {'value': '_1 $a Bach $b Gio. Seb. $f 1685-1750 ', 'taggroup': 'index400'}, {'value': '_1 $a Bach $b G. S. $f 1685-1750 ', 'taggroup': 'index400'}, {'value': '_1 $a Μπαχ $f 1685-1750 ', 'taggroup': 'index400'}, {'value': '_0 $a GR $b Ionian University $c 20090514 ', 'taggroup': 'index800'}, {'value': '__ $a Air on a G string, c1994 $b t.p. (J. S. Bach) ', 'taggroup': 'index800'}, {'value': '__ $a Composizioni, [18--] $b t.p. (Gio. Seb. Bach) ', 'taggroup': 'index800'}, {'value': '__ $a Il clavicembalo bem temperato, 1946 $b t.p. (G. S. Bach) ', 'taggroup': 'index800'}, {'value': '__ $a Μπαχ [SR] 2001 ', 'taggroup': 'index800'}, {'value': '__ $a L.C. ', 'taggroup': 'index800'}, {'value': '4_ $u http://viaf.org/viaf/12304462 $x viaf ', 'taggroup': 'index800'}, {'value': '4_ $u http://id.loc.gov/authorities/names/n79021425 $x lcna ', 'taggroup': 'index800'}, {'value': '__ $a Aslanidi $z 2009-09 ', 'taggroup': 'index900'}]]}]
```

Παρατηρούμε λοιπόν, ότι μέσα στη λίστα res2 είναι ολόκληρο το έγγραφο με το αναζητούμενο στοιχείο, όπως ακριβώς έχει γίνει index στο elasticsearch. Για τη διαδικασία επεξεργασίας των εγγράφων που έχουν βρεθεί κατά την αναζήτηση θα πρέπει να κρατήσουμε τα δεδομένα που είναι απαραίτητα να προβληθούν στον χρήστη. Αυτά τα

δεδομένα είναι το tag που υπάρχει στη λίστα record και το value που υπάρχει στη λίστα datagroup.

```
for res_record in res2:
    for sub_keys, sub_values in res_record.items():
        if sub_keys == 'record':
            for subfields in sub_values:
                for tag_keys, tag_values in subfields.items():
                    if tag_keys == 'tag' and tag_values >= 100:

                        tag_list_1.append(tag_values)

tag_list_2.append(tag_list_1)
tag_list_1=[]
```

Το παραπάνω κομμάτι κώδικα θα εκτελεστεί και αυτό σε περίπτωση που υπάρχει αποτέλεσμα κατά την αναζήτηση για έγγραφα από τον χρήστη. Ο σκοπός του είναι να δημιουργήσει αρχικά μια λίστα με όλα τα tags των δεδομένων που υπάρχουν μέσα σε κάθε έγγραφο τα οποία θα είναι σε σωστή σειρά μέσα στην λίστα επειδή η ίδια ως δομή περιέχει ordered δεδομένα. Για να γίνει αυτό θα πρέπει με μια επανάληψη να ψάξουμε κάθε έγγραφο που υπάρχει στο res2. Στη συνέχεια η δεύτερη επανάληψη έχει σκοπό να αποκτήσει πρόσβαση τόσο στο key όσο και στο value των δεδομένων που υπάρχουν μέσα στο dictionary που έχει ως key το «record» και ως value μια λίστα με όλα τα έγγραφα που βρέθηκαν. Αφού επιβεβαιώσουμε με μια if ότι το key είναι το record και όχι το datagroup, επαναληπτικά ψάχνουμε σε κάθε record κάθε εγγράφου, τα αρχικά μας δεδομένα δηλαδή χωρίς να έχουν υποστεί επεξεργασία. Η τελευταία επανάληψη for,

```
for tag_keys, tag_values in subfields.items():
```

έχει σκοπό να μας δώσει τη δυνατότητα να ελέγξουμε για ακόμα μια φορά το key και το value των στοιχείων της λίστας fields, η οποία περιέχει τα tag των στοιχείων του εγγράφου, σε κάθε record των δεδομένων. Με τη βοήθεια του tag_keys θα μπορέσουμε να ελέγξουμε με μια if, ότι το key του ζεύγους είναι το tag που ψάχνουμε αλλά και το γεγονός ότι η τιμή του είναι πάνω από 100 μιας και τα tag κάτω από το 100 δεν μας χρειάζονται κατά την προβολή των δεδομένων. Σε περίπτωση που επιβεβαιώνεται η συνθήκη τότε η τιμή κάθε tag προσαρτάται σε μια λίστα. Στο τέλος της επανάληψης, η λίστα με τις τιμές των tag για κάθε έγγραφο προσαρτάται σε μια άλλη λίστα και αρχικοποιείται ξανά κενή η πρώτη. Ο στόχος είναι να δημιουργηθεί μια λίστα από λίστες με τα δεδομένα των tag των εγγράφων.

Για παράδειγμα αν ο χρήστης ψάξει για κάθε έγγραφο που περιέχει ένα στοιχείο με tag 300 τότε το αποτέλεσμα της λίστας tag_list_2 θα είναι το παρακάτω:

```
[[100, 106, 120, 152, 200, 340, 801, 856, 856, 970], [100, 101, 102, 106, 120, 152, 200, 340, 400, 801, 810, 810, 856, 856, 970]]
```

Στη συνέχεια θα πρέπει να γίνει μια ανάλογη επεξεργασία και για τις τιμές των εγγράφων.

```
for res_datagroup in res2:
    for data_keys, data_values in res_datagroup.items():
        if data_keys == 'datagroup':
            for values in data_values:
                for key, value in values.items():
                    if key == 'value':

                        data_list_1.append(value)

                data_list_2.append(data_list_1)
            data_list_1=[]
```

Πιο αναλυτικά, θα πρέπει πάλι με μια επανάληψη να μπορούμε στα δεδομένα που περιέχει το res2, με τη μόνη διαφορά ότι αυτή τη φορά πρέπει να βεβαιωθούμε ότι το data_keys είναι το datagroup, ώστε να κάνουμε αναζήτηση για τις τιμές των εγγράφων όπως έγιναν index στο elasticsearch. Στη συνέχεια θα πρέπει για κάθε dictionary που περιέχει η λίστα datagroup να τρέξουμε μια επανάληψη for για κάθε στοιχείο που έχει το κάθε dictionary. Στη συνέχεια θα πρέπει με την βοήθεια μιας if να βεβαιωθούμε ότι το key αυτού του ζεύγους δεδομένων θα είναι το value, ώστε έπειτα να κάνουμε append σε μια λίστα και με τον ίδιο τρόπο με πριν να δημιουργήσουμε την τελική λίστα από λίστες με values τα οποία θέλουμε να τα προβάλλουμε στον χρήστη. Για το ίδιο παράδειγμα με πριν, η λίστα data_list_2 θα περιέχει τα παρακάτω δεδομένα:

```
[['_ $a 20050519agrey010303 $b $a 0', '___ $a ba', '___ $a AACR2 2nd rev. ed. $b lcna', '___1 $a Stravinsky $b Igor $f 1882-1971', '___ $a Composer', '___0 $a GR $b Ionian University $c 20090220', '___4_ $u http://viaf.org/viaf/17309155 $x viaf', '___4_ $u http://id.loc.gov/authorities/names/n79070061 $x lcna', '___ $a Aslanidi $z 2014-01'], ['___ $a 20050519agrey010303 $b $a ita', '___ $a IT', '___ $a 0', '___ $a ba', '___ $a RDA $b lcna', '___1 $a Donizetti $b Gaetano $f 1797-1848', '___ $a Composer', '___1 $a Donizetti $b G. $f 1797-1848 $g (Gaetano)', '___0 $a GR $b Ionian University $c 20090604', '___ $a La favorita, 19-- $b t.p. (G. Donizetti)', '___ $a L.C.', '___4_ $u http://viaf.org/viaf/74037336 $x viaf', '___4_ $u http://id.loc.gov/authorities/names/n79132358 $x lcna', '___ $a Aslanidi $z 2013-07']]
```

Από την στιγμή που όλες οι λίστες είναι ordered, το μόνο που μένει είναι να «ενώσουμε» με έναν τρόπο τις λίστες data_list_2 και tag_list_2 ώστε να δημιουργήσουμε το τελικό

αποτέλεσμα με τα έγγραφα που θέλουμε να προβάλουμε στον χρήστη. Αυτό θα γίνει απλά με το παρακάτω κομμάτι κώδικα:

```
last_result = [list(zip(*z)) for z in zip(data_list_2, tag_list_2)]
```

Με τη βοήθεια της συνάρτησης zip θα φτιάξουμε μια λίστα με εμφωλευμένες λίστες από tuples τα οποία περιέχουν τα ζευγάρια δεδομένων των εγγράφων τα οποία πρέπει να τα παρουσιάσουμε στον χρήστη. Το αποτέλεσμα της last_result μεταβλητής θα είναι το παρακάτω:

```
[(['__ $a 20050519agrey010303 $b $c $d $e $f $g $h $i $j $k $l $m $n $o $p $q $r $s $t $u $v $w $x $y $z $aa $ab $ac $ad $ae $af $ag $ah $ai $aj $ak $al $am $an $ao $ap $aq $ar $as $at $au $av $aw $ax $ay $az $ba $bb $bc $bd $be $bf $bg $bh $bi $bj $bk $bl $bm $bn $bo $bp $bq $br $bs $bt $bu $bv $bw $bx $by $bz $ca $cb $cc $cd $ce $cf $cg $ch $ci $cj $ck $cl $cm $cn $co $cp $cq $cr $cs $ct $cu $cv $cw $cx $cy $cz $da $db $dc $dd $de $df $dg $dh $di $dj $dk $dl $dm $dn $do $dp $dq $dr $ds $dt $du $dv $dw $dx $dy $dz $ea $eb $ec $ed $ee $ef $eg $eh $ei $ej $ek $el $em $en $eo $ep $eq $er $es $et $eu $ev $ew $ex $ey $ez $fa $fb $fc $fd $fe $ff $fg $fh $fi $fj $fk $fl $fm $fn $fo $fp $fq $fr $fs $ft $fu $fv $fw $fx $fy $fz $ga $gb $gc $gd $ge $gf $gg $gh $gi $gj $gk $gl $gm $gn $go $gp $gq $gr $gs $gt $gu $gv $gw $gx $gy $gz $ha $hb $hc $hd $he $hf $hg $hh $hi $hj $hk $hl $hm $hn $ho $hp $hq $hr $hs $ht $hu $hv $hw $hx $hy $hz $ia $ib $ic $id $ie $if $ig $ih $ii $ij $ik $il $im $in $io $ip $iq $ir $is $it $iu $iv $iw $ix $iy $iz $ja $jb $jc $jd $je $jf $jg $jh $ji $jj $jk $jl $jm $jn $jo $jp $jq $jr $js $jt $ju $jv $jw $jx $jy $jz $ka $kb $kc $kd $ke $kf $kg $kh $ki $kj $kk $kl $km $kn $ko $kp $kq $kr $ks $kt $ku $kv $kw $kx $ky $kz $la $lb $lc $ld $le $lf $lg $lh $li $lj $lk $ll $lm $ln $lo $lp $lq $lr $ls $lt $lu $lv $lw $lx $ly $lz $ma $mb $mc $md $me $mf $mg $mh $mi $mj $mk $ml $mm $mn $mo $mp $mq $mr $ms $mt $mu $mv $mw $mx $my $mz $na $nb $nc $nd $ne $nf $ng $nh $ni $nj $nk $nl $nm $nn $no $np $nq $nr $ns $nt $nu $nv $nw $nx $ny $nz $oa $ob $oc $od $oe $of $og $oh $oi $oj $ok $ol $om $on $oo $op $oq $or $os $ot $ou $ov $ow $ox $oy $oz $pa $pb $pc $pd $pe $pf $pg $ph $pi $pj $pk $pl $pm $pn $po $pp $pq $pr $ps $pt $pu $pv $pw $px $py $pz $qa $qb $qc $qd $qe $qf $qg $qh $qi $qj $qk $ql $qm $qn $qo $qp $qq $qr $qs $qt $qu $qv $qw $qx $qy $qz $ra $rb $rc $rd $re $rf $rg $rh $ri $rj $rk $rl $rm $rn $ro $rp $rq $rr $rs $rt $ru $rv $rw $rx $ry $rz $sa $sb $sc $sd $se $sf $sg $sh $si $sj $sk $sl $sm $sn $so $sp $sq $sr $ss $st $su $sv $sw $sx $sy $sz $ta $tb $tc $td $te $tf $tg $th $ti $tj $tk $tl $tm $tn $to $tp $tq $tr $ts $tt $tu $tv $tw $tx $ty $tz $ua $ub $uc $ud $ue $uf $ug $uh $ui $uj $uk $ul $um $un $uo $up $uq $ur $us $ut $uu $uv $uw $ux $uy $uz $va $vb $vc $vd $ve $vf $vg $vh $vi $vj $vk $vl $vm $vn $vo $vp $vq $vr $vs $vt $vu $vv $vw $vx $vy $vz $wa $wb $wc $wd $we $wf $wg $wh $wi $wj $wk $wl $wm $wn $wo $wp $wq $wr $ws $wt $wu $wv $ww $wx $wy $wz $xa $xb $xc $xd $xe $xf $xg $xh $xi $xj $xk $xl $xm $xn $xo $xp $xq $xr $xs $xt $xu $xv $xw $xx $xy $xz $ya $yb $yc $yd $ye $yf $yg $yh $yi $yj $yk $yl $ym $yn $yo $yp $yq $yr $ys $yt $yu $yv $yw $yx $yy $yz $za $zb $zc $zd $ze $zf $zg $zh $zi $zj $zk $zl $zm $zn $zo $zp $zq $zr $zs $zt $zu $zv $zw $zx $zy $zz]]
```

Το μόνο που μένει με την παραπάνω λίστα είναι η σωστή επεξεργασία των τελικών δεδομένων στο front-end της εφαρμογής ώστε να παρουσιαστούν σωστά και όμορφα. Επιλέξαμε να χτίσουμε τα tables στο back-end, σε μια μεταβλητή την οποία θα την περνάμε στο HTML αρχείο, ώστε να μην γίνει η διαδικασία αυτή με τη χρήση Javascript στο front-end για να μην υπάρχει καθυστέρηση στην εμφάνιση των αποτελεσμάτων.

```

regexCode = "\B(\$[a-z0-9](?!a-z0-9))"
regexUrl = "(https?:\\/(?:www\\.|(?:!www)))[a-zA-Z0-9][a-zA-Z0-9-]+[a-
zA-Z0-9]\\.[^\\s]{2,}|www\\.[a-zA-Z0-9][a-zA-Z0-9-]+[a-zA-Z0-
9]\\.[^\\s]{2,}|https?:\\/(?:www\\.|(?:!www)))[a-zA-Z0-
9]+\\.[^\\s]{2,}|www\\.[a-zA-Z0-9]+\\.[^\\s]{2,}"

to_return = ""

for k_tables in last_result:
    table = "<table class='lib-data'><thead
class='libDataHead'><tr><th>ID</th><th>Data</th></tr></thead>"
    table_body = "<tbody class='libDataBody'>"
    tr = "<tr>"
    for k_rows in k_tables:
        splitedRow = k_rows[0].split()
        temp = ""
        temp_tag=""
        tooltip_tag=""
        marc = get_marc(k_rows[1])
        marc_codes = ""

```

Αρχικά δημιουργούμε δυο regular expressions, το regexCode και το regexUrl, τα οποία θα τα χρησιμοποιήσουμε στη συνέχεια για το tooltip και για να κάνουμε τα URL των δεδομένων clickable. Αρχικοποιούμε μια μεταβλητή to_return στην αρχή, η οποία θα είναι και η μεταβλητή που θα επιστρέφουμε στην HTML στο τέλος. Με τη χρήση μιας for θα προσπελάσουμε τα δεδομένα της λίστας last_result και θα ξεκινήσουμε να χτίζουμε τον πίνακα στη μεταβλητή table. Δημιουργούμε την αρχή του html tag table και δημιουργούμε τα thead αυτού του πίνακα, στη μεταβλητή table_body ανοίγουμε το tag του tbody μαζί με το κατάλληλο class ώστε να μπορούμε να διαμορφώσουμε το στυλ με τη βοήθεια της CSS. Στη συνέχεια με μια for θα καταφέρουμε να προσπελάσουμε όλα τα στοιχεία για το κάθε έγγραφο που έχουμε βρει. Επίσης αρχικοποιούμε 3 κενές μεταβλητές, την temp, την temp_tag και την marc_codes οι οποίες θα μας βοηθήσουν στη συνέχεια για να δημιουργήσουμε τα υπόλοιπα στοιχεία των πινάκων με τα αποτελέσματα της αναζήτησης.

Για παράδειγμα τα k_rows θα έχουν τα παρακάτω δεδομένα.

```

('__ $a 20050519agrey010303 $b ', 100)
('__ $a ita ', 101)
('__ $a IT ', 102)
('__ $a O ', 106)
('__ $a ba ', 120)
('__ $a RDA $b lcna ', 152)
('_1 $a Donizetti $b Gaetano $f 1797-1848 ', 200)
('__ $a Composer ', 340)
('_1 $a Donizetti $b G. $f 1797-1848 $g (Gaetano) ', 400)
('_0 $a GR $b Ionian University $c 20090604 ', 801)
('__ $a La favorita, 19-- $b t.p. (G. Donizetti) ', 810)
('__ $a L.C. ', 810)
('4_ $u http://viaf.org/viaf/74037336 $x viaf ', 856)
('4_ $u http://id.loc.gov/authorities/names/n79132358 $x lcna ', 856)
('__ $a Aslanidi $z 2013-07 ', 970)
('__ $a 20050519agrey010303 $b ', 100)
('__ $a O ', 106)
('__ $a ba ', 120)
('__ $a AACR2 2nd rev. ed. $b lcna ', 152)
('_1 $a Stravinsky $b Igor $f 1882-1971 ', 200)
('__ $a Composer ', 340)
('_0 $a GR $b Ionian University $c 20090220 ', 801)
('4_ $u http://viaf.org/viaf/17309155 $x viaf ', 856)
('4_ $u http://id.loc.gov/authorities/names/n79070061 $x lcna ', 856)
('__ $a Aslanidi $z 2014-01 ', 970)

```

Κάνουμε τα δεδομένα που έχουν βρεθεί από την αναζήτηση split, ώστε να μπορέσουμε να δημιουργήσουμε τα tooltip για τα δεδομένα αυτά στη συνέχεια αλλά και να επεξεργαστούμε τα URL ώστε να δώσουμε τη δυνατότητα στο χρήστη να μπορούν να τα πατάνε. Αρχικά θα δημιουργήσουμε όμως το tooltip για τα tags με τη βοήθεια της συνάρτησης `get_marc`.

```

def get_marc(tag):
    save_path = 'myapp/static/json'
    file_name = 'unimarc.json'
    temp = {}

    completeName = os.path.join(save_path, file_name)
    with open(completeName, 'r') as f:
        json_data = json.load(f)

        for key,value in json_data.items():
            for k in value:
                if k['tag'] == tag:
                    temp.update(k)

    f.close()
    if temp:
        return temp
    else:
        return

```

Στη συνάρτηση get_marc περνάμε το tag του στοιχείου που επεξεργαζόμαστε κάθε φορά. Μέσα σε αυτή τη συνάρτηση, ορίζοντας το path θα ανοίξουμε το json αρχείο το οποίο περιέχει τα απαραίτητα στοιχεία για τη σημασία των δεδομένων στο πρότυπο που χρησιμοποιήθηκε και εκχωρούμε τα στοιχεία στη μεταβλητή json_data. Με τη βοήθεια μιας for θα βρούμε το dictionary στο οποίο υπάρχουν τα δεδομένα για το tag που επεξεργαζόμαστε εκείνη τη στιγμή, τα εκχωρούμε στο dictionary temp, κλείνουμε το αρχείο που έχουμε κάνει open και επιστρέφουμε πίσω το temp εάν βρέθηκαν δεδομένα αλλιώς κάνουμε ένα απλό return εάν δεν βρέθηκαν.

Το αρχείο unimarc το οποίο δημιουργήσαμε βάση των δεδομένων του συγκεκριμένου προτύπου έχει την παρακάτω μορφή.

```
{ "unimarc": [  
  {  
    "tag": 100,  
    "info": "General Processing Data",  
    "codes": [  
      {  
        "$a": "General processing data",  
        "$0": "Instruction Phrase",  
        "$1": "Linking Data",  
        "$2": "System Code",  
        "$3": "Authority Record Identifier",  
        "$5": "Relationship Control",  
        "$6": "Interfield Linking Data",  
        "$7": "Script of Cataloguing and  
Script of the Base Access Point",  
        "$8": "Language of Cataloguing and  
Language of the Base Access Point"  
      }  
    ]  
  }  
]
```

Το παραπάνω κομμάτι είναι ένα παράδειγμα για μια εγγραφή, συγκεκριμένα γι' αυτή με το tag 100. Ουσιαστικά είναι μια λίστα με πολλά dictionaries για κάθε μια εγγραφή, μέσα στα οποία υπάρχουν τα στοιχεία για τα tag και τη σημασία τους μέσα στο info, καθώς επίσης και μια λίστα με όλα τα codes και τη σημασία τους.

```

if marc:
    marc_codes = marc['codes']
    if k_rows[1] == marc['tag']:
        temp_tag = "<div class = 'tooltip'>{}<div
class='tooltiptext'>{}</div></div>"
        tooltip_tag = temp_tag.format(k_rows[1],marc['info'])

    if tooltip_tag:
        for splited in splitedRow:
            foundURL = re.search(regexUrl, splited)
            foundCode = re.search(regexCode,splited)

            if foundURL:

                temp_splited = "<a href='"+splited+"'"
target='_blank'>" + splited + "</a>"
                temp += temp_splited + " "
            elif foundCode:

                if marc_codes:
                    for code in marc_codes:
                        temp += "<div class =
'tooltip'>" + splited + "<div
class='tooltiptext'>" + code[splited] + "</div></div>" + " "
                else:
                    temp += splited + " "

            else:

                temp += splited + " "

        td_pre = "<td>{}</td><td>{}</td>"
        td = td_pre.format(tooltip_tag,temp)

```

Εάν η μεταβλητή marc περιέχει δεδομένα, ουσιαστικά εάν η συνάρτηση get_marc έχει βρει δεδομένα, εκχωρούμε στη συνάρτηση marc_codes τη λίστα με τα codes που υπάρχουν για το συγκεκριμένο tag βάσει του προτύπου που χρησιμοποιείται. Ελέγχουμε εάν το στοιχείο στη δεύτερη στήλη του k_rows είναι ίσο με το tag που επέστρεψε η συνάρτηση και χτίζουμε το tooltip για τα tag, με τη βοήθεια δυο div και των κατάλληλων κλάσεων ώστε να τα διαμορφώσουμε με τη χρήση CSS στο frontend. Στη συνέχεια ελέγχουμε εάν έχει δημιουργηθεί το tooltip για τα tag, ώστε να δημιουργήσουμε το tooltip για τα codes που περιέχουν τα δεδομένα. Εάν επιβεβαιώνεται η συνθήκη του if τότε με τη βοήθεια μια for ελέγχουμε κάθε στοιχείο από τα στοιχεία που κάναμε split στην αρχή. Ύστερα δημιουργούμε 2 μεταβλητές την foundURL και την foundCode με τις οποίες συγκρίνουμε τα δεδομένα με το regexUrl και το regexCode αντίστοιχα. Στη συνέχεια εάν κάποιο στοιχείο είναι URL, επιβεβαιώνεται η πρώτη συνθήκη της if και εκχωρούμε το στοιχείο στην temp με το anchor tag ώστε να γίνει clickable. Εάν το στοιχείο είναι marc code τότε επιβεβαιώνεται η δεύτερη συνθήκη, στην οποία ελέγχουμε αρχικά εάν υπάρχουν στοιχεία στα marc_codes βάσει του προτύπου. Εάν υπάρχουν τότε δημιουργούμε τα div για την δημιουργία του tooltip για τα

codes και τα εκχωρούμε στην temp. Στην περίπτωση που δεν υπάρχουν στοιχεία marc_codes τότε απλά εκχωρούμε το στοιχείο στην temp. Τέλος για την περίπτωση εάν υπάρχουν δεδομένα marc γι' αυτό το στοιχείο, απλά διαμορφώνουμε τα HTML tags «td» για τα κελιά του πίνακα που δημιουργήσαμε στην αρχή και εκχωρούμε τα δεδομένα των tag και την temp στην μεταβλητή td.

```
        else:
            for splited in splitedRow:
                foundURL = re.search(regexUrl, splited)
                foundCode = re.search(regexCode, splited)
                if foundURL:
                    temp_splited = "<a href='"+splited+" '
target='_blank'>"+splited+"</a>"
                    temp += temp_splited+" "
                else:
                    temp += splited+" "
            td_pre = "<td>{}</td><td>{}</td>"
            td = td_pre.format(k_rows[1],temp)

        tr += td + "</tr>"

table_body += tr
table += table_body
table += "</table>"
to_return += table
```

Εάν δεν υπάρχουν δεδομένα marc γι' αυτό το στοιχείο του εγγράφου τότε, δημιουργούμε με τον ίδιο τρόπο τις επαναλήψεις και τις συγκρίσεις των δεδομένων με τα regular expressions, μόνο που αυτή τη φορά δεν δημιουργούμε τα div για τη δημιουργία του tooltip. Στη συνέχεια αφού έχουμε δημιουργήσει τα κελιά των γραμμών του πίνακα εκχωρούμε τα δεδομένα τα οποία βρίσκονται στην μεταβλητή td, στη μεταβλητή tr και κλείνουμε το HTML tag της γραμμής του πίνακα. Εκχωρούμε τη γραμμή στο table_body, το οποίο το εκχωρούμε στον πίνακα κλείνοντας το HTML tag του table και τέλος ολόκληρο το table σε μορφή string το εκχωρούμε στη μεταβλητή to_return το οποίο θα επιστρέψουμε στο τέλος στο HTML αρχείο.

Το τελευταίο πράγμα που θα πρέπει να ληφθεί υπ' όψιν είναι το πως θα γίνει το pagination στο front-end. Θα πρέπει να υπάρχουν από το back-end κάποιες μεταβλητές και μια διαδικασία ώστε ο χρήστης να βλέπει τα επιθυμητά έγγραφα σωστά, χωρισμένα σε σελίδες χωρίς να υπάρχουν λάθη. Το παρακάτω κομμάτι κώδικα είναι υπεύθυνο για όλα τα παραπάνω:

```

page_num1 = 1
if page == 1:
    first_page = True
else:
    first_page = False

total_tables = res['hits']['total']
check_total_pages = total_tables/PER_PAGE

if isinstance(check_total_pages, float):
    total_pages = int(math.ceil(total_tables/PER_PAGE))
elif isinstance(check_total_pages, int):
    total_pages = total_tables/PER_PAGE

if page == total_pages:
    last_page = True
else:
    last_page = False

```

- **page_num1:** Μια σταθερά η οποία είναι δείκτης για την πρώτη σελίδα των αποτελεσμάτων.
- **total_tables:** Τα συνολικά έγγραφα που έχουν βρεθεί κατά την αναζήτηση.
- **check_total_pages:** Μεταβλητή η οποία θα χρησιμοποιηθεί ώστε να υπολογίσουμε πόσες σελίδες θα δημιουργηθούν βάση των εγγράφων που έχουν βρεθεί κατά την αναζήτηση.
- **total_pages:** Οι συνολικές σελίδες που θα δημιουργήσει η εφαρμογή.
- **first_page:** Λογική μεταβλητή ώστε να ελέγχουμε στο front-end εάν βρισκόμαστε στην πρώτη σελίδα ή όχι. Θα χρησιμοποιηθεί για τα navigation buttons.
- **last_page:** Λογική μεταβλητή ώστε να ελέγχουμε στο front-end εάν βρισκόμαστε στην τελευταία σελίδα. Όπως και η first_page έτσι και η last_page θα χρησιμοποιηθεί για τα navigation buttons.

Πιο αναλυτικά, αρχικά γίνεται ένας έλεγχος στη μεταβλητή page (της οποίας η τιμή μεταφέρεται με GET μέσω URL) αν έχει την τιμή 1. Αν η τιμή της είναι το 1, τότε η μεταβλητή first_page γίνεται TRUE, σε οποιαδήποτε άλλη περίπτωση η τιμή της first_page θα είναι FALSE. Στη συνέχεια αφού οριστούν οι μεταβλητές total_tables και check_total_pages, δημιουργούμε μια if με μια συνθήκη η οποία είναι υπεύθυνη με την βοήθεια του εργαλείου isinstance να ελέγχει εάν η τιμή της μεταβλητής check_total_pages είναι δεκαδικός αριθμός ή είναι ακέραιος. Στην περίπτωση που είναι δεκαδικός αριθμός (float) τότε εκτελείται η παρακάτω γραμμή κώδικα:

```
total_pages = int(math.ceil(total_tables/PER_PAGE))
```

Ουσιαστικά υπολογίζει τον συνολικό αριθμό των σελίδων που θα δημιουργήσει η εφαρμογή διαιρώντας τον συνολικό αριθμό των εγγράφων που έχουν βρεθεί με την σταθερά PER_PAGE η οποία είναι ο αριθμός των εγγράφων που θέλουμε να υπάρχει σε κάθε σελίδα. Το αποτέλεσμα της διαίρεσης στρογγυλοποιείται πάντα προς τα πάνω και μετατρέπεται σε ακέραιο αριθμό ώστε να χρησιμοποιηθεί στο frontend για το navigation. Ο λόγος που στρογγυλοποιείται πάντα προς τα πάνω είναι ότι σε περίπτωση που είναι δεκαδικός, πάει να πει ότι στην τελευταία σελίδα δεν έχει τον μέγιστο αριθμό των εγγράφων που έχουμε ορίσει στην σταθερά PER_PAGE αλλά έχει 1 μέχρι 4. Για παράδειγμα όταν τα συνολικά έγγραφα που βρεθούν είναι 12 και η PER_PAGE είναι 5 τότε το αποτέλεσμα της διαίρεσης είναι 2,4 το οποίο γίνεται 3 ώστε να εμφανιστούν όλα τα έγγραφα που έχουν βρεθεί.

Στην συνέχεια υπάρχει μια elif ώστε να ελέγξουμε αν ο αριθμός είναι ακέραιος ώστε να εκτελεστεί η παρακάτω γραμμή κώδικα:

```
total_pages = total_tables/PER_PAGE
```

Σε αυτή τη περίπτωση η διαίρεση γίνεται κανονικά χωρίς τη χρήση κάποιου εργαλείου όπως πριν. Τέλος υπάρχει μια if ώστε να ελέγχεται αν η τιμή της σελίδας (μεταβλητή page) έχει την ίδια τιμή με την μεταβλητή total_pages, με λίγα λόγια ελέγχεται αν η σελίδα είναι η τελευταία. Στην περίπτωση που ισχύει η συνθήκη τότε η λογική μεταβλητή last_page γίνεται TRUE, σε αντίθετη περίπτωση θα πάρει την τιμή FALSE.

Το τελευταίο πράγμα που πρέπει να γίνει από το back-end, σε περίπτωση που βρεθούν κάποια έγγραφα είναι να κάνει render την HTML σελίδα layout.html και να μεταφέρει τις μεταβλητές που θα χρειαστούμε ώστε να φτιάξουμε στο front-end το περιεχόμενό της. Αυτό θα το καταφέρει με την παρακάτω γραμμή.

```
return render_template("layout.html", tag_input=tag_input,
value_input=value_input, json_data=to_return, page=page, first_page
= first_page, last_page = last_page, total_pages = total_pages,
page_num1=page_num1)
```

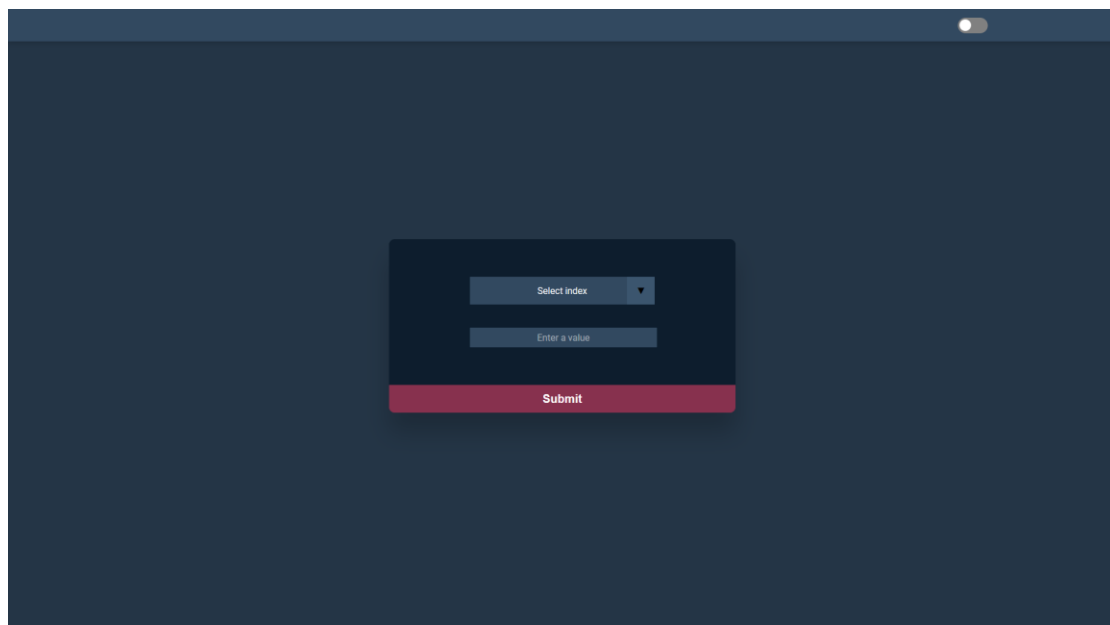
3.2 Αρχιτεκτονική του Front-end

Το τμήμα ενός ιστότοπου με τον οποίο αλληλεπιδρά άμεσα ο χρήστης ονομάζεται front-end. Αναφέρεται επίσης συχνά και ως η «πλευρά του πελάτη» (client-side) σε μια εφαρμογή ή μια ιστοσελίδα. Περιλαμβάνει ό,τι βιώνουν οι χρήστες άμεσα:

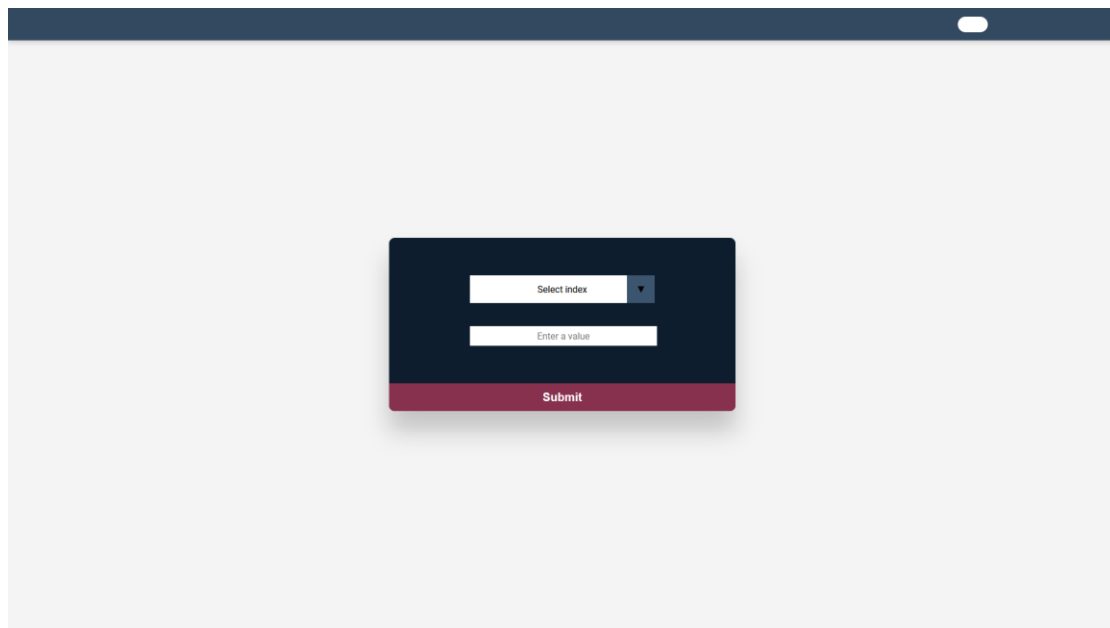
- Χρώμα και στυλ κειμένου
- Εικόνες
- Γραφήματα και πίνακες
- Κουμπιά
- Χρώμα της σελίδας
- Μενού πλοήγησης

Η HTML, η CSS και η JavaScript είναι οι γλώσσες που χρησιμοποιούνται για την ανάπτυξη του front-end. Η δομή, ο σχεδιασμός, η συμπεριφορά και το περιεχόμενο όλων των στοιχείων που εμφανίζονται στις οθόνες του προγράμματος περιήγησης όταν ανοίγει ένας ιστότοπος, μια web ή mobile εφαρμογή εφαρμόζεται από τους προγραμματιστές του front-end.

Η εμφάνιση της αρχικής σελίδας, με το σκουρόχρωμο θέμα, που βλέπει ο χρήστης στην εφαρμογή που αναπτύχθηκε είναι κάπως έτσι:



Ενώ με το ανοιχτόχρωμο θέμα:



The image shows a dark-themed user interface. At the top, there is a dark blue header bar with a small white pill-shaped button on the right. Below the header, the main content area is a light gray rectangle. In the center of this area is a dark blue rounded rectangle representing a form. Inside this form, there are two input fields: the top one is a dropdown menu with the placeholder text 'Select index' and a small blue arrow icon on the right; the bottom one is a text input field with the placeholder text 'Enter a value'. At the bottom of the dark blue form is a red bar with the word 'Submit' in white text.

Πιο αναλυτικά σε επίπεδο κώδικα υπάρχουν κάποια σημεία με τα οποία το back-end επικοινωνεί με το front-end ώστε να παρουσιαστεί σωστά το περιεχόμενο που πρέπει.

Ο κώδικας για την φόρμα όπου ο χρήστης θα επιλέγει τον τρόπο αναζήτησης που επιθυμεί είναι το παρακάτω:

```
<form action = "{{url_for('table_bp.form_input', page=1)}}"
method="POST" class="index_form">

    <div class="form-input-material">
        <div class="select">

            <select name="tag_input" id="tag_input">
                <option disabled selected value>Select index</option>

                {% for taggroup in data_list %}

                    <option value="{{taggroup}}">{{taggroup}}</option>

                {% endfor %}

            </select>
        </div>

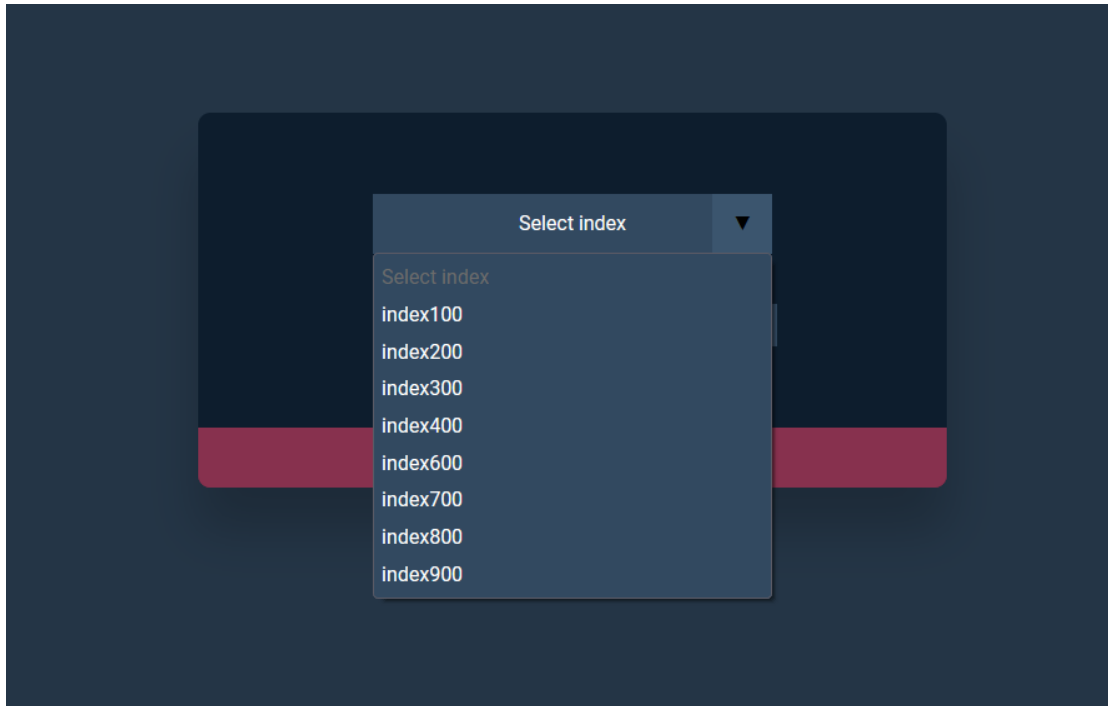
        <input type = "text" name = "value_input" id="value_input"
class="input-text" placeholder="Enter a value"/>
    </div>
    <input type="submit" value = "Submit" class="btn">
</form>
```

Αναλυτικά θα παρατηρήσουμε ότι φτιάχνεται μια HTML φόρμα με action:

```
{{url_for('table_bp.form_input', page=1)}}
```

Θα μεταφέρει τον χρήστη δηλαδή στη HTML σελίδα που κάνει render η συνάρτηση `form_input` που βρίσκεται στο αρχείο `table_bp`. Παράλληλα μεταφέρει στην συνάρτηση μέσω του URL και την μέθοδο GET την μεταβλητή `page`, της οποίας η αρχική τιμή είναι το 1.

Μέσα στην φόρμα υπάρχουν δυο πεδία που μπορεί να χρησιμοποιήσει ο χρήστης ώστε να επιλέξει τις τιμές με τις οποίες επιθυμεί να πραγματοποιήσει μια αναζήτηση. Ο πρώτος τρόπος να επιλέξει μια τιμή είναι μέσω της φόρμας `select`.



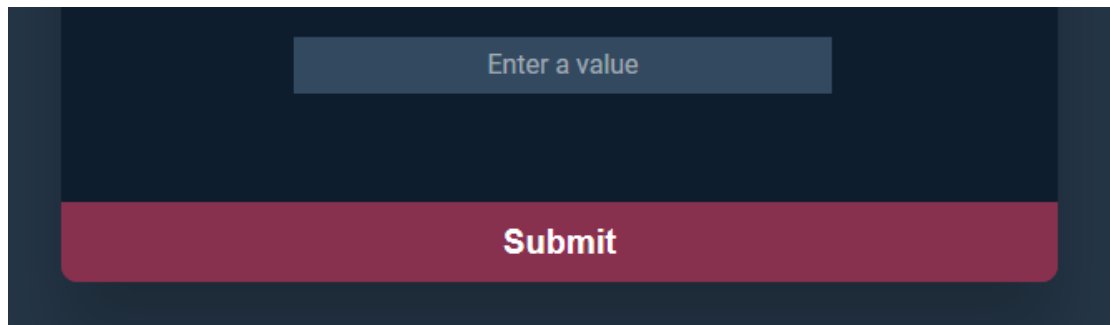
Η δημιουργία του περιεχομένου που περιέχει η φόρμα select είναι δυναμική. Σε αυτό βοηθάει η Jinja2 η οποία χρησιμοποιεί τα στοιχεία που έχουμε μεταφέρει στο front-end από το back-end.

```
{% for taggroup in data_list %}
    <option value="{{taggroup}}">{{taggroup}}</option>
{% endfor %}
```

Συγκεκριμένα το παραπάνω κομμάτι δημιουργεί μια επανάληψη for με την βοήθεια της Jinja2. Τα στοιχεία της λίστας data_list είχαν παρουσιαστεί και στο κομμάτι του back-end και είναι τα παρακάτω:

```
['index100', 'index200', 'index300', 'index400', 'index600', 'index700', 'index800', 'index900']
```

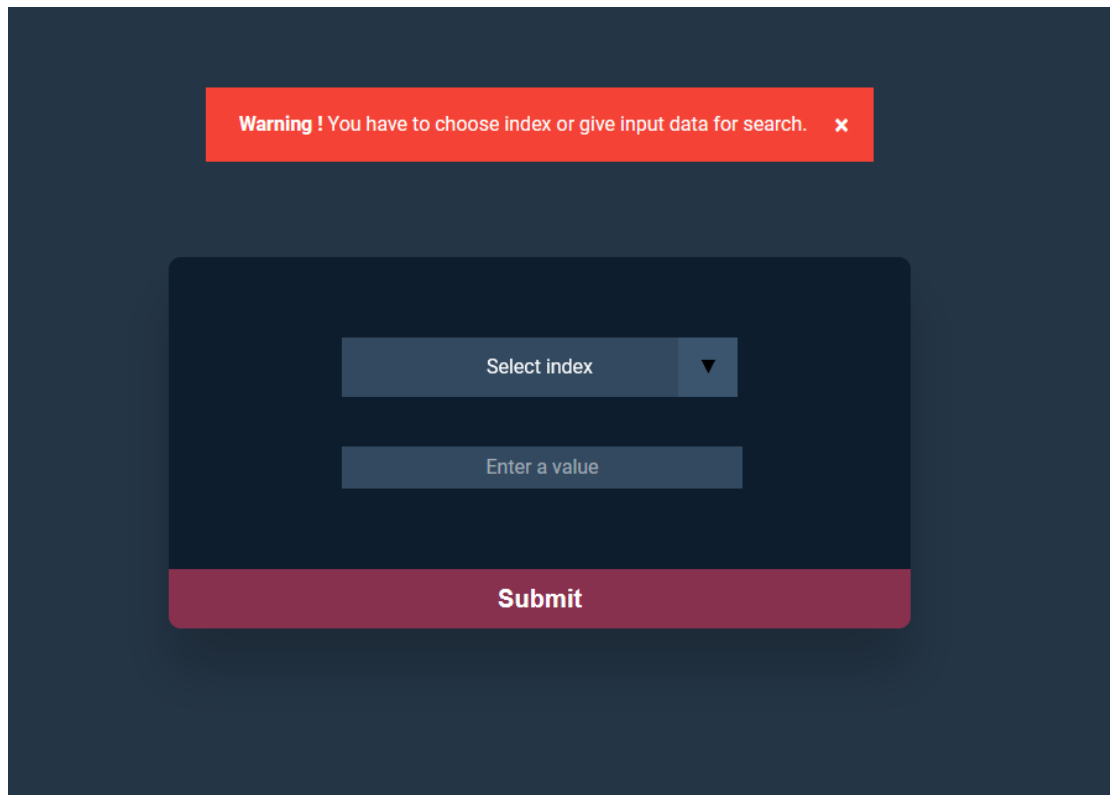
Για κάθε ένα στοιχείο που περιέχει η λίστα data_list δημιουργούμε ένα option στην φόρμα select με τιμή το κάθε ένα στοιχείο της λίστας. Ο δεύτερος τρόπος που μπορεί να αναζητήσει ένα στοιχείο ο χρήστης είναι μέσω του πεδίου που έχει φτιαχτεί στην φόρμα και τέλος υπάρχει και το κουμπί Submit το οποίο θα εκτελέσει και το action της φόρμας όταν το πατήσει ο χρήστης.



Enter a value

Submit

Σε περίπτωση που ο χρήστης πατήσει το κουμπί Submit χωρίς να έχει επιλέξει ένα στοιχείο είτε από τη φόρμα select είτε πληκτρολογώντας το στο ειδικό πεδίο θα εμφανιστεί σε αυτόν η παρακάτω οθόνη



Warning ! You have to choose index or give input data for search. ✕

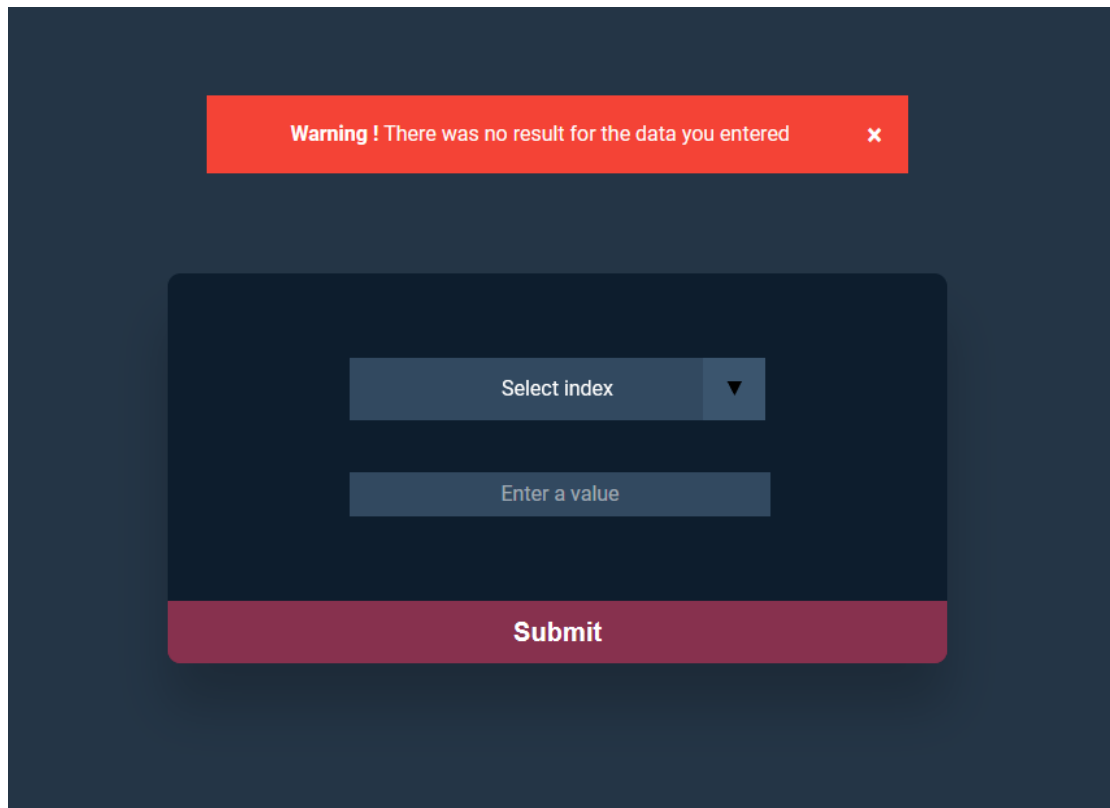
Select index ▼

Enter a value

Submit

Παρατηρούμε ότι έχει εμφανιστεί ένα κουτί το οποίο ενημερώνει τον χρήστη ότι πρέπει να επιλέξει ένα στοιχείο για να πραγματοποιήσει την αναζήτηση σωστά.

Σε περίπτωση που επιλέξει ένα στοιχείο με οποιονδήποτε τρόπο αλλά δεν βρέθηκε κάποιο αποτέλεσμα κατά την αναζήτηση, ο χρήστης θα δει στην οθόνη του το παρακάτω μήνυμα.



Γι' αυτά τα μηνύματα προειδοποίησης του χρήστη υπεύθυνο είναι το παρακάτω κομμάτι κώδικα.

```
{% if warning %}

    <div class="alert">
        <span class="closebtn"
onclick="this.parentElement.style.display='none';">&times;</span>
        <strong>Warning !</strong> You have to choose index
or give input data for search.
    </div>

{% elif no_result %}

    <div class="alert">
        <span class="closebtn"
onclick="this.parentElement.style.display='none';">&times;</span>
        <strong>Warning !</strong> There was no result for
the data you entered
    </div>

{% endif %}
```

Αυτό που γίνεται ουσιαστικά είναι με την βοήθεια της Jinja να υπάρξει ένας έλεγχος εάν η μεταβλητή `warning` έχει την τιμή `TRUE`. Σε περίπτωση που επιβεβαιωθεί η συνθήκη τότε εκτελείται το μήνυμα προειδοποίησης του χρήστη ότι δεν έχει επιλέξει κάποιο στοιχείο ώστε να πραγματοποιηθεί η αναζήτηση. Εάν δεν επιβεβαιωθεί συνθήκη, δηλαδή η λογική μεταβλητή `warning` έχει την τιμή `FALSE` τότε το μήνυμα δεν εμφανίζεται στον χρήστη. Στη δεύτερη περίπτωση ελέγχου, ελέγχεται εάν η τιμή της λογικής μεταβλητής `no_result` είναι `TRUE`, όπου σε αυτή την περίπτωση θα εμφανιστεί το προειδοποιητικό μήνυμα ότι δεν έχει βρεθεί αποτέλεσμα κατά την αναζήτηση των στοιχείων που επιλέχθηκαν από τον χρήστη.

Αφού ο χρήστης επιλέξει τα στοιχεία με τα οποία θέλει να πραγματοποιηθεί μια αναζήτηση από την εφαρμογή, για παράδειγμα το `index200` από τη φόρμα `select` τότε θα δει στην οθόνη του το παρακάτω αποτέλεσμα.

ID	Data
100	... Sa 20050517cagrey01003 bsa
106	... Sa 0
120	... Sa 0a
152	... Sa AACR2 2nd rev. ed. Sa kma
200	... 1 Sa Obituary (2d igor 21 1882-1971
340	... Sa Compressor
801	... 0 Sa 0M Sa kma University Sa 200505220
856	4... Sa http://vial.org/vial/17309185 Sa vial
856	4... Sa http://id.loc.gov/auth/authnames/179170001 Sa kma
970	... Sa Anisette Sa 2014-01

ID	Data
100	... Sa 20050517cagrey01003 bsa
200	... 1 Sa WEISSON Sa Cam
856	4... Sa http://vial.org/vial/1829276 Sa vial
856	4... Sa http://id.loc.gov/auth/authnames/182100601 Sa kma

ID	Data
100	... Sa 20050517cagrey01003 bsa
200	... 1 Sa Boniface Sa Priscilla
856	4... Sa http://vial.org/vial/48131872 Sa vial
856	4... Sa http://id.loc.gov/auth/authnames/183010700 Sa kma

Όπως παρατηρούμε, το back-end έχει στείλει σωστά τα δεδομένα στο front-end ως string και έχει αναγνωρίσει σωστά τα HTML elements.

Για να το καταφέρουμε αυτό, δημιουργούμε στο HTML έγγραφο ένα `div` και με τη βοήθεια της `jinja2` εμφανίζουμε τα αποτελέσματα όπως τα έχουμε διαμορφώσει σε πίνακες στο back-end, χρησιμοποιώντας το φίλτρο `safe` ώστε να μην τα εμφανίσει ως απλό κείμενο, αλλά να καταλάβει η HTML ότι είναι HTML elements.

```
<div id="container" class="table-wrapper">{{ json_data|safe }}</div>
```

Αμα παρατηρήσουμε προσεκτικά θα δούμε ότι κάτω κάθε ID και κάτω από τα codes υπάρχουν διακεκομμένες γραμμές.

ID	Data
100	___ \$a 20050519cgrey0103 ba
200	__1 \$a Braillard \$b Philippe
856	4_ \$u http://viaf.org/viaf/4927298 \$x viaf
856	4_ \$u http://id.loc.gov/authorities/names/n82111885 \$x lcna

Εάν ο χρήστης πάει το ποντίκι του πάνω σε κάποιο από αυτά τα στοιχεία θα εμφανιστεί το tooltip που δημιουργήσαμε με τα κατάλληλα div στο backend και τους δώσαμε στυλ στη συνέχεια με τη χρήση CSS στο frontend.

ID	Data
100	___ \$a 20050519cgrey0103 ba
200	__1 \$a Braillard \$b Philippe
856	4_ \$u http://viaf.org/viaf/4927298 \$x viaf
856	4_ \$u http://id.loc.gov/authorities/names/n82111885 \$x lcna

Επίσης θα παρατηρήσουμε ότι και τα links είναι clickable και μπορεί να τα πατήσει κάποιος και να ανοίξει νέο παράθυρο στον περιηγητή του με τη συγκεκριμένη σελίδα του συνδέσμου.

ID	Data
100	___ \$a 20050519cgrey0103 ba
200	__1 \$a Braillard \$b Philippe
856	4_ \$u http://viaf.org/viaf/4927298 \$x viaf
856	4_ \$u http://id.loc.gov/authorities/names/n82111885 \$x lcna

Όπως είπαμε και πριν όμως σε κάθε σελίδα έχουμε ορίσει να υπάρχουν μέχρι 5 έγγραφα, άρα μέχρι 5 πίνακες. Για να μεταβούμε από την μια σελίδα στην άλλη θα χρειαστούμε ένα μενού πλοήγησης ώστε να μας βοηθάει να εναλλασσόμαστε στις σελίδες.

ID	Data
100	__ \$a 20050519cgrey0103 ba
200	_1 \$a Brailiard \$b Philippe
856	4_ \$u http://viaf.org/viaf/4927298 \$x viaf
856	4_ \$u http://id.loc.gov/authorities/names/n82111885 \$x lcna

ID	Data
100	__ \$a 20050519cgrey0103 ba
200	_1 \$a SENARCLENS \$b Pierre de
856	4_ \$u http://viaf.org/viaf/95159291 \$x viaf
856	4_ \$u http://id.loc.gov/authorities/names/n85804915 \$x lcna

« First

« Previous

Next »

Last »

Για να δημιουργηθεί αυτό το μενού πλοήγησης, ο κώδικας είναι ο ακόλουθος:

```
<div class="page-navigation">

    {% if not first_page %}
        {% if value_input %}
            <a href="{{url_for('table_bp.form_input',
page=page_num1, tag_input=tag_input, value_input=value_input)}}">
class="first-last-button left">&laquo; First</a>
            <a href="{{url_for('table_bp.form_input',
page=page-1, tag_input=tag_input, value_input=value_input)}}">
class="p-n-button left">&#8249; Previous</a>

        {% else %}
            <a href="{{url_for('table_bp.form_input',
page=page_num1, tag_input=tag_input)}}">
class="first-last-button
left">&laquo; First</a>
            <a href="{{url_for('table_bp.form_input',
page=page-1, tag_input=tag_input)}}">
class="p-n-button left">&#8249;
Previous</a>
        {% endif %}
    {% endif %}

    {% if not last_page %}
        {% if value_input %}
            <a href="{{url_for('table_bp.form_input',
page=total_pages, tag_input=tag_input, value_input=value_input)}}">
class="first-last-button right">Last &raquo;</a>
            <a href="{{url_for('table_bp.form_input',
page=page+1, tag_input=tag_input, value_input=value_input)}}">
class="p-n-button right">Next &#8250;</a>
        {% else %}
            <a href="{{url_for('table_bp.form_input',
page=total_pages, tag_input=tag_input)}}">
class="first-last-button
right">Last &raquo;</a>
            <a href="{{url_for('table_bp.form_input',
page=page+1, tag_input=tag_input)}}">
class="p-n-button right">Next
&#8250;</a>
        {% endif %}
    {% endif %}

</div>
```

Με την βοήθεια της Jinja θα πρέπει να κάνουμε κάποιους ελέγχους ώστε να δημιουργήσουμε το HTML έγγραφο σωστά σε κάθε σελίδα που θα βρισκόμαστε. Αρχικά γίνεται έλεγχος αν βρισκόμαστε στην πρώτη σελίδα ώστε στην περίπτωση που δεν βρισκόμαστε σε αυτή, να δημιουργήσουμε το κουμπί first ώστε να μας πηγαίνει κατευθείαν στην πρώτη σελίδα αλλά και το κουμπί previous το οποίο δεν υπάρχει λόγος να υπάρχει εάν βρισκόμαστε ήδη σε αυτή.

Επίσης ο δεύτερος έλεγχος που γίνεται μέσα στον πρώτο είναι εάν υπάρχει στοιχείο `value_input` ώστε να ξέρουμε εάν πρέπει ή όχι να το μεταφέρουμε μέσω του URL στο back-end για να μας επιστρέψει τα στοιχεία της επόμενης σελίδας. Εάν δεν βρισκόμαστε στην πρώτη σελίδα τότε φτιάχνουμε τα κουμπιά `previous` και `first` τα οποία είναι `anchor elements` με `href` το οποίο καλεί τη συνάρτηση `form_input` στο αρχείο `table_br` και μεταφέρει με την μέθοδο GET τα δεδομένα που έχει επιλέξει ήδη ο χρήστης ώστε να μην χαθούν κατά την αλλαγή της σελίδας. Η ίδια διαδικασία γίνεται αντίστοιχα και για την περίπτωση που βρισκόμαστε στην τελευταία σελίδα των αποτελεσμάτων. Ακολουθούν εικόνες με το πως φαίνεται το μενού πλοήγησης ανάλογα με το εάν βρισκόμαστε στην πρώτη ή στην τελευταία σελίδα.

ID	Data
100	__ \$a 20050519agrey010305 ba
101	__ \$a ita
102	__ \$a IT
106	__ \$a 0
120	__ \$a ba
152	__ \$a AACR2 2nd rev. ed. \$b lcna
200	_1 \$a Bellini \$b Vincenzo \$f 1801-1835
801	_0 \$a GR \$b Ionian University \$c 20100223
810	__ \$a L.C.
856	4_ \$u http://viaf.org/viaf/39560983 \$x viaf
856	4_ \$u http://id.loc.gov/authorities/names/n80122768 \$x lcna
970	__ \$a Aslanidi \$z 2015-02

ID	Data
100	__ \$a 20050519cgrey0103 ba
200	_1 \$a SMYTH \$b Sheila A.
856	4_ \$u http://viaf.org/viaf/2617575 \$x viaf
856	4_ \$u http://id.loc.gov/authorities/names/n94038285 \$x lcna

Next ›
Last »

Η παραπάνω οθόνη είναι στιγμιότυπο της πρώτης σελίδας αποτελεσμάτων ενώ η παρακάτω είναι στιγμιότυπο της τελευταίας σελίδας.

ID	Data
100	__ \$a 20050519cgrey0103 ba
200	_1 \$a GOULD \$b Stephen Jay
856	4_ \$u http://viaf.org/viaf/108205470 \$x viaf
856	4_ \$u http://id.loc.gov/authorities/names/n79145199 \$x lca

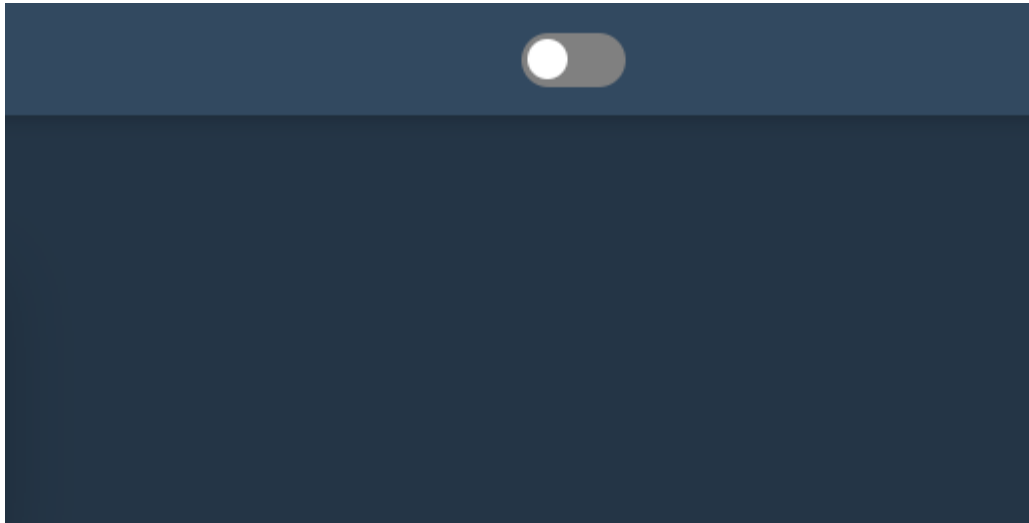
ID	Data
100	__ \$a 20050519cgrey0103 ba
200	_1 \$a Driessen \$b Karen C.
856	4_ \$u http://viaf.org/viaf/12452133 \$x viaf
856	4_ \$u http://id.loc.gov/authorities/names/n94038284 \$x lca

[« First](#) [« Previous](#)

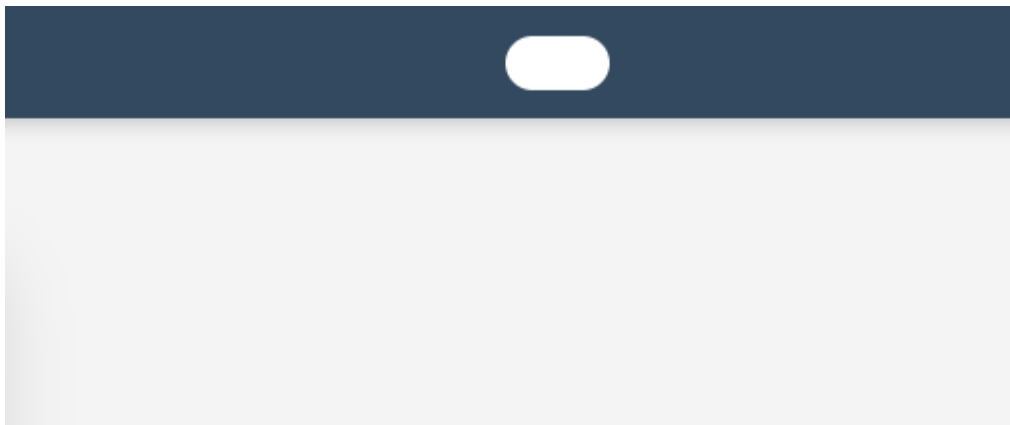
Για την αλλαγή του θέματος, δημιουργήσαμε δυο div με ένα checkbox και ένα label html tag μέσα ώστε με τη βοήθεια της CSS να μπορέσουμε να δώσουμε την εμφάνιση του διακόπτη.

```
<div class="container">
    <div class="toggle-container">
        <input type="checkbox" id="input-color-switch"
name="theme" /><label for="input-color-switch"></label>
    </div>
</div>
```

Το αποτέλεσμα του παραπάνω κώδικα, με ενεργοποιημένο το σκούρο θέμα είναι:



Ενώ με ενεργοποιημένο το ανοιχτόχρωμο θέμα είναι:



Η JavaScript όμως είναι υπεύθυνη για την αλλαγή των χρωμάτων. Ο κώδικας είναι ο ακόλουθος.

```
const colorSwitch = document.getElementById("input-color-switch");

function enableDarkMode(){
    localStorage.setItem('theme', 'theme-dark');
    document.body.classList.remove('theme-light');
    document.body.classList.add('theme-dark');
}

function enableLightMode(){
    localStorage.setItem('theme', 'theme-light');
    document.body.classList.remove('theme-dark');
    document.body.classList.add('theme-light');
}

colorSwitch.addEventListener("click", checkMode);

function checkMode() {
    if (colorSwitch.checked) {
        enableLightMode();
    } else {
        enableDarkMode();
    }
}

(function () {
    if (localStorage.getItem('theme') === 'theme-light') {
        enableLightMode();
        colorSwitch.checked = true;
    } else {
        enableDarkMode();
    }
})();
```

Αρχικά δημιουργούμε μια μεταβλητή colorSwitch, στην οποία καταχωρούμε το label element του διακόπτη για την αλλαγή του θέματος. Στη συνέχεια δημιουργούμε δυο συναρτήσεις, την enableDarkMode και την enableLightMode οι οποίες είναι υπεύθυνες να εισάγουν το όνομα της κλάσης για το ανάλογο mode. Στη σκουρόχρωμη λειτουργία αφαιρούμε από το body το class name theme-light το οποίο έχει τα χρώματα για το ανοιχτόχρωμο θέμα και προσθέτουμε σε αυτό το class name theme-dark το οποίο έχει τα χρώματα για το σκούρο θέμα. Τέλος με τη βοήθεια του local storage του browser αποθηκεύουμε το όνομα ώστε να

το παραμένει ενεργό το θέμα που ενεργοποιήσαμε κατά την αλλαγή σελίδας ή άμα κλείσουμε τον περιηγητή. Στη συνάρτηση `enableLightMode` γίνεται ακριβώς η ίδια διαδικασία για το class name `theme-light` το οποίο περιέχει τα χρώματα για το ανοιχτόχρωμο θέμα. Στη συνέχεια δημιουργούμε ένα event listener για να ελέγχει το click του χρήστη στο label για να αλλάξει το θέμα και καλούμε τη συνάρτηση `checkMode` η οποία θα ελέγχει τη κατάσταση που βρίσκεται ο διακόπτης ώστε να ενεργοποιήσει την αντίστοιχη συνάρτηση. Εάν το switch είναι checked, δηλαδή έχει πατηθεί το checkbox από τον χρήστη τότε θα καλεί την συνάρτηση `enableLightMode` και θα αλλάζουν τα χρώματα της σελίδας στην ανοιχτόχρωμη λειτουργία, ενώ εάν δεν είναι checked τότε θα καλείται η συνάρτηση `enableDarkMode` ώστε να ενεργοποιείται η σκουρόχρωμη. Τέλος χρειαζόμαστε μια συνάρτηση η οποία θα καλείται απευθείας μόλις γίνεται load η σελίδα ώστε να ενεργοποιείται το ανάλογο θέμα. Αυτό γίνεται ελέγχοντας την ονομασία του θέματος στο local storage και ενεργοποιείται το ανάλογο θέμα αναλόγως ποιο είχε ενεργοποιηθεί τη τελευταία φορά, όπως επίσης φροντίζουμε στο ανοιχτόχρωμο θέμα να κρατήσουμε checked το checkbox.

4. Εγκατάσταση εφαρμογής

Πριν την ανάπτυξη της ίδιας της εφαρμογής, η εγκατάσταση και αρχικοποίηση της ήταν απαραίτητη. Αρχικά έπρεπε να δημιουργήσουμε ένα script το οποίο θα επεξεργαστεί το αρχικό json αρχείο με όλα τα δεδομένα εισόδου και θα δημιουργήσει το τελικό JSON αρχείο στην μορφή που απαιτείται ώστε να γίνει το indexing των δεδομένων σωστά στο elasticsearch μέσω της κονσόλας. Στη συνέχεια με τις κατάλληλες εντολές εγκαταστάθηκε και αρχικοποιήθηκε η εφαρμογή, όπου στη συνέχεια αναπτύχθηκε ο κώδικας της.

Αρχικά δημιουργήθηκε ένα python script το οποίο όπως ήδη είπαμε, ήταν υπεύθυνο για τη δημιουργία του τελικού JSON αρχείου το οποίο χρησιμοποιήθηκε για να εισάγουμε τα δεδομένα στην εφαρμογή μας.

```
import json

with open('test.json') as json_file:
    data = json.load(json_file)
    temp_data = []
    temp_dict = {}
    temp_datagroup = {}
    for test in data['test']:
        for fields in test['fields']:
            if fields['tag'] >= 100:
                temp_val = ""
                for subfields in fields['subfields']:
                    temp_val += '$'+subfields['code']+'
'+subfields['data']+' '
                    if fields['ind1'] == ' ' and fields['ind2'] != ' ':
                        ind1 = '_'
                        temp_test = ind1+fields['ind2'] + ' ' + temp_val

                    elif fields['ind2'] == ' ' and fields['ind1'] != ' ':
                        ind2 = '_'
                        temp_test = fields['ind1']+ind2 + ' ' + temp_val

                    elif fields['ind1'] == ' ' and fields['ind2'] == ' ':
                        ind1 = '_'
                        ind2 = '_'
                        temp_test = ind1+ind2 + ' ' + temp_val
```

Το πρώτο πράγμα που κάνουμε στο script είναι να ανοίξουμε το αρχείο με τα δεδομένα τα οποία πρέπει να επεξεργαστούμε και το ονομάζουμε json_file όπου στη συνέχεια το μετατρέπουμε σε python object με όνομα data με την βοήθεια του json.load. Αρχικοποιούμε μια κενή λίστα και δύο κενά dictionaries τα οποία θα μας βοηθήσουν στην επεξεργασία των δεδομένων και στη συνέχεια με την βοήθεια δύο επαναλήψεων ώστε να προσπελάσουμε τα δεδομένα μέσα στα fields του εγγράφου, ελέγχουμε εάν το tag είναι πάνω από 100 μιας και δεν χρειαζόμαστε τα στοιχεία με tag μικρότερο από αυτή την τιμή. Στην περίπτωση που είναι πάνω από 100, επεξεργαζόμαστε τον τρόπο που θα εμφανιστούν τα στοιχεία στο τελικό μας

έγγραφο. Αφού ελέγχουμε την κάθε πιθανή τιμή που μπορούν να έχουν οι δείκτες (indicators) του κάθε στοιχείου, δημιουργούμε τη σωστή μορφή για τους δείκτες αλλά και τα δεδομένα του ίδιου του εγγράφου ώστε να φαίνονται στη σωστή μορφή μαζί με τα code του κάθε στοιχείου.

```
if fields['tag'] >= 100 and fields['tag'] <= 199:
    temp_dict['taggroup'] = 'index100'
elif fields['tag'] >= 200 and fields['tag'] <= 299:
    temp_dict['taggroup'] = 'index200'
elif fields['tag'] >= 300 and fields['tag'] <= 399:
    temp_dict['taggroup'] = 'index300'
elif fields['tag'] >= 400 and fields['tag'] <= 499:
    temp_dict['taggroup'] = 'index400'
elif fields['tag'] >= 500 and fields['tag'] <= 599:
    temp_dict['taggroup'] = 'index500'
elif fields['tag'] >= 600 and fields['tag'] <= 699:
    temp_dict['taggroup'] = 'index600'
elif fields['tag'] >= 700 and fields['tag'] <= 799:
    temp_dict['taggroup'] = 'index700'
elif fields['tag'] >= 800 and fields['tag'] <= 899:
    temp_dict['taggroup'] = 'index800'
elif fields['tag'] >= 900 and fields['tag'] <= 999:
    temp_dict['taggroup'] = 'index900'

temp_dict['value'] = temp_test

if temp_dict:
    temp_data.append(temp_dict)
    temp_dict = {}

temp_datagroup['datagroup'] = temp_data
temp_datagroup['record'] = test['fields']

print(json.dumps({"index":{"_index":"lib-index", "_type":"lib-
data"}},ensure_ascii=False,separators=(',', ':')))

print(json.dumps(temp_datagroup,ensure_ascii=False,separators=(',',
':')))

temp_data = []
temp_datagroup = {}
```

Στη συνέχεια με την βοήθεια πολλαπλών if, ελέγχουμε τις τιμές των tag για κάθε στοιχείο και ορίζουμε στο dictionary temp_dict την ανάλογη τιμή για κάθε tag ως value στο key taggroup και στο τέλος ορίζουμε και την τιμή του δεδομένου που έχουμε επεξεργαστεί πιο πάνω ως value στο key value στο ίδιο dictionary με τα tag και έπειτα τα κάνουμε append στη λίστα temp_data και αδειάζουμε το temp_dict. Στη συνέχεια ορίζουμε στο key datagroup του dictionary temp_datagroup το value temp_data, δηλαδή ολόκληρη τη λίστα που φτιάξαμε

και στο key record ολόκληρο το έγγραφο που περιέχεται μέσα στο fields, το οποίο θα μας χρειαστεί για να βρούμε το tag του κάθε στοιχείου και να το εμφανίσουμε στον χρήστη της εφαρμογής μας. Τέλος με τις παρακάτω γραμμές θα εκτυπώσουμε τα δεδομένα με σωστό τρόπο ώστε να τα εισάγουμε στο elasticsearch.

```
print(json.dumps({"index":{"_index":"lib-index", "_type":"lib-  
data"}},ensure_ascii=False,separators=(',', ':')))  
  
print(json.dumps(temp_datagroup,ensure_ascii=False,separators=(',',  
':')))
```

Η πρώτη γραμμή θα είναι πάνω από κάθε γραμμή στοιχείου που θα εισάγουμε η οποία είναι υπεύθυνη να ορίσει στο elasticsearch το όνομα του index και το type που θα εισάγει τα δεδομένα και στη δεύτερη γραμμή εκτυπώνουμε το temp_datagroup το οποίο είναι όλα τα δεδομένα εμφωλευμένα σε ένα dictionary.

Το script το εκτελέσαμε στη κονσόλα με το παρακάτω τρόπο:

```
index_data.py > index_output.json
```

Ένα παράδειγμα ενός εγγράφου είναι το παρακάτω:

```
{"index":{"_index":"lib-index", "_type":"lib-data"}}  
  
{"datagroup":[{"taggroup":"index100","value":"__ $a 20050519cgrey0103  ba  
"}, {"taggroup":"index200","value":"_1 $a GRANT DUFF $b Shiela $f 1913-  
"}, {"taggroup":"index800","value":"4_ $u http://viaf.org/viaf/115859233 $x viaf  
"}, {"taggroup":"index800","value":"4_ $u http://id.loc.gov/authorities/names/n83064573 $x  
lcna  
"}], "record":[{"tag":1, "data":"ionioa10002161"}, {"tag":3, "data":"ionio"}, {"tag":5, "data":"200  
50519203353.0"}, {"subfields":[{"code":"z", "data":"(ionio)ionioa10002161"}, {"tag":35, "ind2":  
" ", "ind1":" "}, {"subfields":[{"code":"a", "data":"20050519cgrey0103  
ba"}], "tag":100, "ind2":" ", "ind1":" "}, {"subfields":[{"code":"a", "data":"GRANT  
DUFF"}, {"code":"b", "data":"Shiela"}, {"code":"f", "data":"1913-  
"}], "tag":200, "ind2":"1", "ind1":"  
"}, {"subfields":[{"code":"u", "data":"http://viaf.org/viaf/115859233"}, {"code":"x", "data":"viaf  
"}, {"tag":856, "ind2":"  
", "ind1":"4"}, {"subfields":[{"code":"u", "data":"http://id.loc.gov/authorities/names/n830645  
73"}, {"code":"x", "data":"lcna"}], "tag":856, "ind2":" ", "ind1":"4"}]}  
  
{"index":{"_index":"lib-index", "_type":"lib-data"}}
```

Για τη σωστή εισαγωγή των δεδομένων στο elasticsearch θα πρέπει πρώτα να φροντίσουμε να δώσουμε σε αυτό την περιγραφή των δεδομένων που θα εισάγουμε. Αυτό θα γίνει με το ένα JSON αρχείο το οποίο ονομάζεται mapping. Μέσα σε αυτό το JSON αρχείο θα περιγράψουμε τα δεδομένα, τον τύπο τους και εάν είναι εμφωλευμένα ή όχι. Το mapping που χρησιμοποιήθηκε είναι το παρακάτω.

```
{
  "lib-data": {
    "properties": {
      "record": {"enabled": "false"},
      "datagroup": {
        "type": "nested",
        "properties": {
          "taggroup": {"type": "keyword"},
          "value": {"type": "text"}
        }
      }
    }
  }
}
```

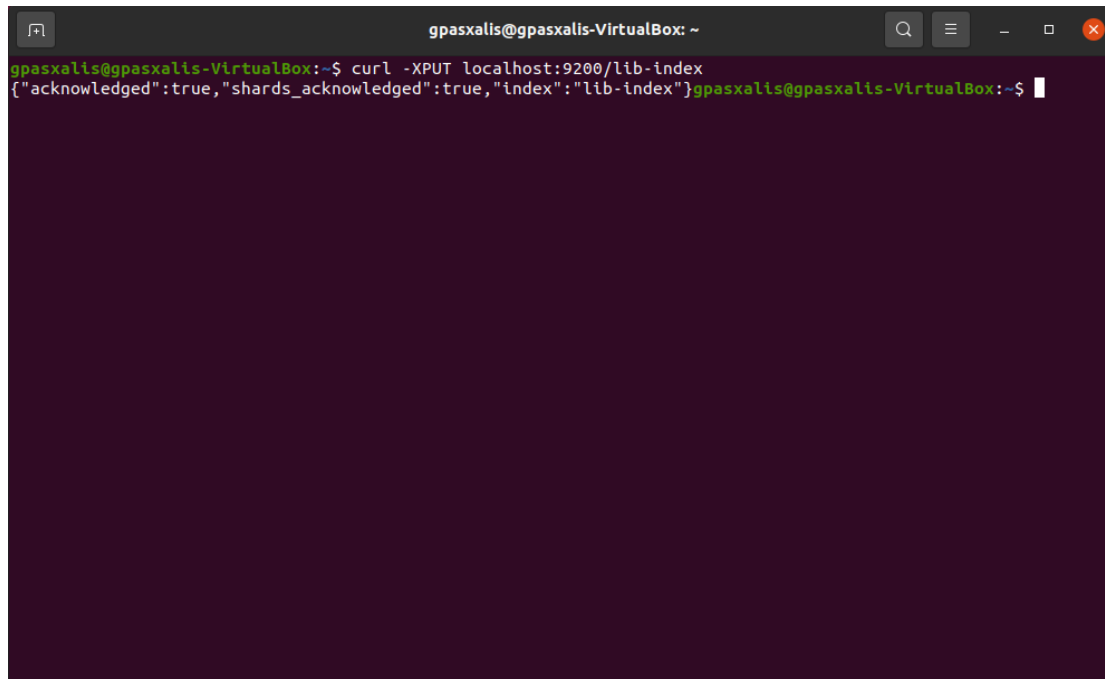
Πρώτα δηλώσαμε τον τύπο των εγγράφων, δηλαδή το lib-data, όπως ακριβώς θα δηλωθούν κατά την δημιουργία του index. Στη συνέχεια δηλώσαμε το record ως enabled false. Αυτό σημαίνει ότι δεν θα γίνει indexing στα δεδομένα που θα έχει μέσα στο record αλλά απλά θα τα αποθηκεύει ώστε να μπορούμε να προσπελάσουμε τα στοιχεία που περιέχει. Έπειτα δηλώσαμε το datagroup το οποίο είναι nested dictionary. Το datagroup περιέχει τα στοιχεία taggroup, δηλωμένο ως keyword το οποίο χρησιμοποιείται για δομημένο περιεχόμενο όπως ID, διευθύνσεις email, tags αλλά και το value το οποίο δηλώθηκε ως text, δηλαδή ως απλό μη δομημένο κείμενο.

Αφού δημιουργήθηκε και το mapping αρχείο το μόνο που μένει είναι να δημιουργήσουμε το index στον server και να εισάγουμε όλα αυτά τα αρχεία που φτιάξαμε πιο πριν.

- 1) Αρχικά δημιουργούμε το index με την παρακάτω εντολή στην οποία θα δώσουμε και το όνομα του index:

```
curl -XPUT localhost:9200/lib-index
```

Μετά την εκτέλεση της εντολής, το elasticsearch θα μας ενημερώσει αν ήταν πετυχημένη ή όχι η εκτέλεσή της.

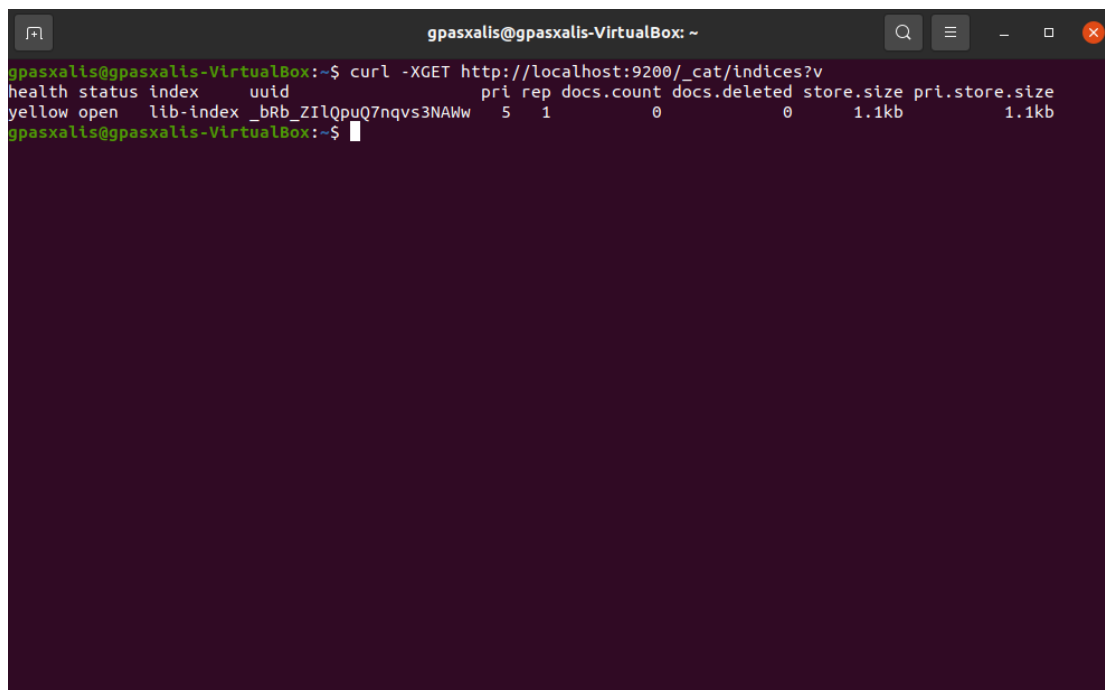


```
gpasxalis@gpasxalis-VirtualBox: ~  
gpasxalis@gpasxalis-VirtualBox:~$ curl -XPUT localhost:9200/lib-index  
{"acknowledged":true,"shards_acknowledged":true,"index":"lib-index"}gpasxalis@gpasxalis-VirtualBox:~$
```

Στην περίπτωση που θέλουμε να επιβεβαιώσουμε ότι έχει δημιουργηθεί όντως το index μπορούμε να το δούμε με την βοήθεια της παρακάτω εντολής.

```
curl -XGET http://localhost:9200/_cat/indices?v
```

Μετά την εκτέλεση της θα δούμε στην οθόνη μας το ένα κείμενο σε μορφή πίνακα με τα απαραίτητα στοιχεία για την κατάσταση του index.



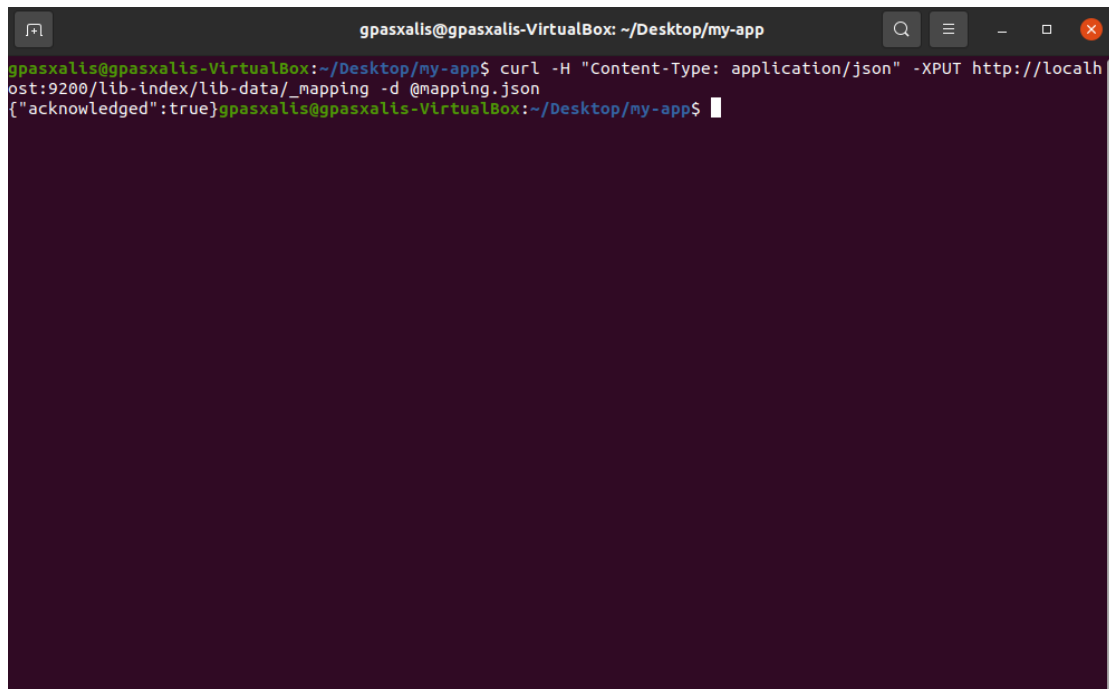
```
gpasxalis@gpasxalis-VirtualBox: ~  
gpasxalis@gpasxalis-VirtualBox:~$ curl -XGET http://localhost:9200/_cat/indices?v  
health status index      uuid                                pri rep docs.count docs.deleted store.size pri.store.size  
yellow open   lib-index _bRb_Zl1QpuQ7nqvs3NAWw      5   1       0           0       1.1kb       1.1kb  
gpasxalis@gpasxalis-VirtualBox:~$
```

- 2) Στη συνέχεια εισάγουμε το mapping στο index που μόλις δημιουργήσαμε. Στην περίπτωση που χρησιμοποιούμε την 5^η έκδοση του elasticsearch θα χρησιμοποιήσουμε την παρακάτω εντολή:

```
curl -XPUT 'http://localhost:9200/lib-index/lib-data/_mapping' -d @mapping.json
```

Ενώ στην περίπτωση που χρησιμοποιούμε την 6^η έκδοση του elasticsearch τότε θα χρησιμοποιήσουμε την ίδια εντολή με την μόνη διαφορά ότι πρέπει να εισάγουμε και το Content-Type.

```
curl -H "Content-Type: application/json" -XPUT http://localhost:9200/lib-index/lib-data/_mapping -d @mapping.json
```



The screenshot shows a terminal window titled 'gpasxalis@gpasxalis-VirtualBox: ~/Desktop/my-app'. The command entered is `curl -H "Content-Type: application/json" -XPUT http://localhost:9200/lib-index/lib-data/_mapping -d @mapping.json`. The output is `{"acknowledged":true}`, indicating the mapping was successfully created.

Με την εκτέλεση της εντολής θα δούμε στην οθόνη μας το απαραίτητο μήνυμα επιβεβαίωσης της δημιουργίας του mapping. Σε διαφορετική περίπτωση θα μας ενημερώσει για τυχόν σφάλμα κατά την εκτέλεσή της.

- 3) Αφού εκτελέσαμε και την εντολή για να εισάγουμε το mapping στο index lib-index, μένει να γίνει το indexing των δεδομένων. Αυτό θα έγινε με την παρακάτω εντολή.

```
curl -s -H "Content-Type: application/x-ndjson" -XPOST localhost:9200/lib-index/_bulk --data-binary "@lib-data.json"; echo
```

Αφού εκτελεστεί επιτυχημένα η παραπάνω εντολή θα δούμε την παρακάτω οθόνη.

```
gpasxalis@gpasxalis-VirtualBox: ~/Desktop/my-app
gpasxalis@gpasxalis-VirtualBox:~/Desktop/my-app$ curl -s -H "Content-Type: application/x-ndjson" -XPOST localhost:9200/lib-index/_bulk --data-binary "@index_output.json"; echo
{"took":257,"errors":false,"items":[{"index":{"_index":"lib-index","_type":"lib-data","_id":"VVGesGnsBFMench_NC1w3","_version":1,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"_seq_no":0,"_primary_term":1,"status":201}},{"index":{"_index":"lib-index","_type":"lib-data","_id":"VoGeGnsBFMench_NC1w3","_version":1,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"_seq_no":0,"_primary_term":1,"status":201}},{"index":{"_index":"lib-index","_type":"lib-data","_id":"V4GeGnsBFMench_NC1w3","_version":1,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"_seq_no":1,"_primary_term":1,"status":201}},{"index":{"_index":"lib-index","_type":"lib-data","_id":"WIGesGnsBFMench_NC1w3","_version":1,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"_seq_no":1,"_primary_term":1,"status":201}},{"index":{"_index":"lib-index","_type":"lib-data","_id":"WYGeGnsBFMench_NC1w3","_version":1,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"_seq_no":0,"_primary_term":1,"status":201}},{"index":{"_index":"lib-index","_type":"lib-data","_id":"WoGeGnsBFMench_NC1w3","_version":1,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"_seq_no":0,"_primary_term":1,"status":201}},{"index":{"_index":"lib-index","_type":"lib-data","_id":"W4GeGnsBFMench_NC1w3","_version":1,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"_seq_no":1,"_primary_term":1,"status":201}},{"index":{"_index":"lib-index","_type":"lib-data","_id":"XIGesGnsBFMench_NC1w3","_version":1,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"_seq_no":0,"_primary_term":1,"status":201}},{"index":{"_index":"lib-index","_type":"lib-data","_id":"XYGeGnsBFMench_NC1w3","_version":1,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"_seq_no":2,"_primary_term":1,"status":201}},{"index":{"_index":"lib-index","_type":"lib-data","_id":"XoGeGnsBFMench_NC1w3","_version":1,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"_seq_no":3,"_primary_term":1,"status":201}},{"index":{"_index":"lib-index","_type":"lib-data","_id":"X4GeGnsBFMench_NC1w3","_version":1,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"_seq_no":4,"_primary_term":1,"status":201}},{"index":{"_index":"lib-index","_type":"lib-data","_id":"YIGesGnsBFMench_NC1w3","_version":1,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"_seq_no":1,"_primary_term":1,"status":201}},{"index":{"_index":"lib-index","_type":"lib-data","_id":"YYGeGnsBFMench_NC1w3","_version":1,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"_seq_no":2,"_primary_term":1,"status":201}},{"index":{"_index":"lib-index","_type":"lib-data","_id":"YoGeGnsBFMench_NC1w3","_version":1,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"_seq_no":2,"_primary_term":1,"status":201}},{"index":{"_index":"lib-index","_type":"lib-data","_id":"Y4GeGnsBFMench_NC1w3","_version":1,"result":"cr
```

Με αυτή την οθόνη, το elasticsearch μας ενημερώνει για την επιτυχημένη εκτέλεση των εντολών, τα στοιχεία που έγιναν indexed καθώς και τυχόν σφάλματα που μπορεί να υπήρξαν κατά την εκτέλεση. Σε περίπτωση που επιθυμούμε να περάσουμε διαφορετικά δεδομένα στο elasticsearch ενώ έχουν γίνει ήδη indexed κάποια, θα πρέπει να διαγράψουμε το υπάρχον index και να κάνουμε την διαδικασία από την αρχή. Η διαγραφή του index θα πραγματοποιηθεί με την παρακάτω εντολή στην οποία πρέπει να εισάγουμε απλά το όνομα που έχουμε δώσει.

```
curl -XDELETE localhost:9200/lib-index
```

Αφού εκτελεστεί, το elasticsearch θα μας ενημερώσει εάν ήταν επιτυχημένη η διαγραφή ή εάν υπήρξε κάποιο σφάλμα.

```
gpasxalis@gpasxalis-VirtualBox: ~/Desktop/my-app
gpasxalis@gpasxalis-VirtualBox:~/Desktop/my-app$ curl -XDELETE localhost:9200/lib-index
{"acknowledged":true}gpasxalis@gpasxalis-VirtualBox:~/Desktop/my-app$
```

5. Σύνοψη – Προτάσεις για το μέλλον

Κλείνοντας, αξίζει να αναφερθεί και πάλι ότι το μεγαλύτερο πρόβλημα των αντίστοιχων προγραμμάτων ILS, είναι ότι πρόκειται για μεγάλα και με πολλά modules προγράμματα που το κάθε ένα από αυτά συμβάλλει στο να είναι πολλές φορές κακή η εμπειρία του χρήστη, μη φιλική προς αυτόν αλλά και προς τους υπολογιστικούς πόρους του συστήματος. Η εφαρμογή που αναπτύχθηκε καταφέρνει να λύσει πολλά από τα προβλήματα του χρήστη αλλά σίγουρα έχει περιθώρια για βελτίωση και προσθήκες ώστε να καταφέρει να αντικαταστήσει ολοκληρωτικά τα προγράμματα που χρησιμοποιούνται μέχρι τώρα.

Με τη χρήση σύγχρονων τεχνολογιών καταφέραμε να κάνουμε την εφαρμογή ελαφριά και γρήγορη αλλά ταυτόχρονα και φιλική προς τον χρήστη μέσω όμορφης διεπαφής (interface). Δίνεται στον χρήστη η δυνατότητα να επιλέξει το θέμα και τον χρωματισμό ανάλογα με την επιθυμία του με επιλογές σκουρόχρωμο ή ανοιχτόχρωμο θέμα ώστε να μην είναι κουραστική η εμπειρία του χρήστη κατά τη διάρκεια της παραμονής του στην εφαρμογή. Επιπλέον του δίνεται η δυνατότητα να κάνει αναζητήσεις με πολλούς και διαφορετικούς τρόπους ώστε να βρει γρήγορα και εύκολα το αποτέλεσμα που επιθυμεί. Συμπληρωματικά, υπάρχει στη διάθεση του χρήστη ο μηχανισμός του tooltip ο οποίος του δίνει τη δυνατότητα να κατανοήσει καλύτερα τα αποτελέσματα που υπάρχουν στους πίνακες, τόσο για τα tags των δεδομένων όσο και για τα codes. Αυτό είναι ιδιαίτερα χρήσιμο για αρχάριους χρήστες μιας και μπορεί να βοηθήσει να αποκτήσουν μεγαλύτερη εμπειρία στην κατανόηση των βιβλιογραφικών δεδομένων.

Παρόλα τα θετικά υπάρχει η δυνατότητα επέκτασης και αναβάθμισης της εφαρμογής με νέες λειτουργίες ώστε να είναι πιο ολοκληρωμένη. Αρχικά η πιο σημαντική αναβάθμιση της εφαρμογής θα ήταν να μπει η δυνατότητα να γίνεται επεξεργασία των εγγράφων που έχουν γίνει indexed στο elasticsearch μέσα από την ίδια την εφαρμογή, χωρίς να υπάρχει η ανάγκη να διαγραφεί το index που υπάρχει και να περαστεί καινούργιο από την αρχή με τα νέα στοιχεία ή την ενημέρωση στοιχείων που υπάρχουν ήδη μέσα στο index. Αυτό θα γλίτωνε αρκετό χρόνο από το indexing των δεδομένων διότι είναι μια διαδικασία που μπορεί να χρειαστεί να γίνει αρκετές φορές από τους διαχειριστές των δεδομένων της βιβλιοθήκης. Επίσης, θα μπορούσε η εφαρμογή να έχει διαφορετική λειτουργία στους απλούς χρήστες μια βιβλιοθήκης και στους διαχειριστές των δεδομένων αυτής. Για παράδειγμα θα μπορούσε να υπάρχει ένα προφίλ για το κάθε έγγραφο, στο οποίο θα υπάρχουν όλα τα δεδομένα του εγγράφου με πιο απλό τρόπο παρουσίασης για τους απλούς χρήστες, όπως για παράδειγμα η χρήση της εικόνας του εξωφύλλου του βιβλίου, ενώ παράλληλα θα δίνει την δυνατότητα στους διαχειριστές των δεδομένων μιας βιβλιοθήκης να δουν το έγγραφο και στην μορφή Marc. Τέλος, μια μεγάλη επέκταση της εφαρμογής στο μέλλον θα ήταν η προσθήκη και άλλων λειτουργιών, όπως για παράδειγμα μια λειτουργία για τους δανεισμούς των βιβλίων στα μέλη της βιβλιοθήκης, επιστροφές καθώς και ολόκληρο το ιστορικό των δανεισμών του κάθε βιβλίου αλλά και το ιστορικό δανεισμών ανά χρήστη.

Βιβλιογραφία-Δικτυογραφία

- [1] What is Flask Python, pythonbasics.org
- [2] Flask Documentation, flask.palletsprojects.com
- [3] What is JavaScript?, developer.mozilla.org
- [4] What is JavaScript? A Definition of the JS Programming Language, freecodecamp.org
- [5] Working with JSON, developer.mozilla.org
- [6] What is JSON? A better format for data exchange, infoworld.com
- [7] What is CSS?, developer.mozilla.org
- [8] Getting started with HTML, developer.mozilla.org
- [9] Introducing JSON, json.org
- [10] Git Handbook, GitHub Guides
- [11] An Introduction to JavaScript, javascript.info
- [12] Understanding MARC, loc.gov
- [13] Universal Bibliographic Control and International MARC Core Programme, ifla.org
- [14] Eric A. Meyer, Estelle Weyl. 2017. CSS: The Definitive Guide, 4th Edition. O'Reilly Media, Inc. ISBN: 9781449393199
- [15] Paul Wilton, Jeremy McPeak. 2009. Beginning JavaScript, 4th Edition. Wrox. ISBN: 9780470525937
- [16] Matt Copperwaite, Charles Leifer. 2015. Learning Flask Framework, 1st Edition. Packt Publishing. ISBN: 9781783983360
- [17] IFLA. 2009, UNIMARC Concise Authorities Format, 3rd Edition
- [18] Michele Seikel, Thomas Steele. 2011. How MARC Has Changed: The History of the Format and its Forthcoming Relationship to RDA. Technical Services Quarterly, 28, no. 3, 322-334, DOI: 10.1080/07317131.2011.574519
- [19] Miguel Grinberg. 2018. Flask Web Development: Developing Web Applications with Python, 2nd Edition. O'Reilly Media, Inc. ISBN: 9781491991732