

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Web εφαρμογή για την συλλογή και την παρουσίαση
δεδομένων επιστημονικών δημοσιεύσεων»

Του φοιτητή
Αθανάσιου Μπογατίνη
Αρ. Μητρώου: 144254

Επιβλέπων
Στέφανος Ουγιάρογλου
Βαθμίδα ΕΔΙΠ

18 Σεπτεμβρίου 2020

Τίτλος Π.Ε: Web εφαρμογή για την συλλογή και την παρουσίαση
δεδομένων επιστημονικών δημοσιεύσεων

Κωδικός Π.Ε.: 20111

Ονοματεπώνυμο: Μπογατίνης Αθανάσιος

Ονοματεπώνυμο εισηγητή: Στέφανος Ουγιάργλου

Ημερομηνία ανάληψης Π.Ε.: 30 Μαρτίου 2020

Ημερομηνία περάτωσης Π.Ε.: 20 Σεπτεμβρίου 2020

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή Αθανάσιου Μπογατίνη που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης, ο συγγραφέας/δημιουργός εκχωρεί στο Διεθνές Πανεπιστήμιο της Ελλάδος άδεια χρήσης του δικαιώματος αναπαραγωγής, δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσης της εργασίας διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος. Η ανοικτή πρόσβαση στο πλήρες κείμενο της εργασίας, δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού, ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, πώληση, εμπορική χρήση, διανομή, έκδοση, μεταφόρτωση (downloading), ανάρτηση (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραίτητα και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

Περίληψη

Ο ανταγωνισμός ήταν από πάντα ένα χαρακτηριστικό της ανθρωπίνης φύσης. Η διαδικασία της σύγκρισης και της αξιολόγησης συνεχώς βελτιώνεται με την επιλογή των παραμέτρων που έχουν μεγαλύτερο “βάρος στη συνάρτηση” και φυσικά με τη χρήση νέων τεχνολογιών. Η βιβλιομετρία και συγκεκριμένα η επιστημονομετρία είναι ένα σύνολο μεθόδων με σκοπό να πραγματοποιηθεί ποιοτική ανάλυση της επιστημονικής επίδοσης και της συμβολής στην επιστημονική κοινότητα.

Οι πλέον διαδεδομένοι και ισχυρότεροι διαδικτυακοί βιβλιομετρικοί ιστότοποι, παρά τον τεράστιο όγκο δεδομένων και την απόλυτη ευκολία πρόσβασης που προσφέρουν, δεν παρέχουν λειτουργίες σύγκρισης μεταξύ των επιστημόνων και των ερευνητών. Η ανάγκη μιας τέτοιας σύγκρισης, εφόσον δεν υπάρχουν άλλα μέσα, εκπληρώνεται με χρονοβόρους και μη αυτόματους τρόπους σε μέσα τα οποία διαθέτουν την πληροφορία αλλά όχι την λειτουργία της σύγκρισής τους. Συνεπώς, δημιουργείται η ανάγκη ύπαρξης ενός βιβλιογραφικού ιστότοπου όπου οι ερευνητές, οι συγγραφείς και οι ερευνητικές ομάδες θα συγκρίνονται μεταξύ τους βάσει του πλήθους και του τύπου των δημοσιεύσεων και των αναφορών τους. Ένας τέτοιος ιστότοπος θα ήταν ιδιαίτερα χρήσιμος τόσο στο προσωπικό που είναι επιφορτισμένο με τη διεξαγωγή των διαδικασιών εσωτερικής και εξωτερικής αξιολόγησης της έρευνας στα πανεπιστήμια και στα τμήματα τους όσο και στις επιτροπές και τα εκλεκτορικά σώματα που είναι τα αρμόδια όργανα για την επιλογή μόνιμου ακαδημαϊκού προσωπικού, ερευνητών και έκτακτου προσωπικού στα πανεπιστήμια και στα ερευνητικά κέντρα καθώς και για την κάλυψη θέσεων ευθύνης σε επιστημονικούς φορείς και οργανισμούς τόσο του δημοσίου όσο και του ιδιωτικού τομέα.

Αυτή ή ανάγκη αποτελεί το κίνητρο για την εκπόνηση της παρούσας εργασίας. Έτσι, υλοποιήθηκε μια εφαρμογή η οποία δίνει τη δυνατότητα αναζήτησης ερευνητών, παρακολούθησης και ανάλυσης των έργων τους και σύγκριση μεταξύ τους. Πλέον υπάρχει η δυνατότητα εμφάνισης των βιβλιομετρικών πληροφοριών συγκεντρωτικά και σε ένα σημείο για πολλούς συγγραφείς με τα αξιόπιστα δεδομένα που προσφέρουν οι πιο γνωστοί βιβλιομετρικοί και επιστημονικοί ιστότοποι. Μέσω της εφαρμογής μπορεί κάποιος να δημιουργήσει ομάδες ερευνητών τους οποίους μπορεί με ευκολία να αναζητήσει. Εν συνεχεία μπορεί να συγκρίνει τις ομάδες είτε συνολικά είτε ατομικά, ακόμη και επιλεκτικά. Τα αποτελέσματα συλλέγονται από αξιόπιστες πηγές και παρουσιάζονται με τη μορφή γραφημάτων για την καλύτερη διαχώριση και κατανόηση από τον χρήστη.

«Web application for collecting and presenting data of scientific publications»

«Athanasios Mpogatinis»

Abstract

Competition has always been a feature of human nature. The process of comparison and evaluation is constantly improving by selecting the parameters that have a greater "weight in the function" and of course, by the use of new technologies. Bibliometry, and specifically scientometrics, is a set of methods in order to carry out a qualitative analysis of scientific performance and contribution to the scientific community.

The most popular and powerful online bibliometric sites, despite the huge amount of data and the absolute ease of access they offer, do not provide comparison functions between scientists and researchers. The need for such a comparison, if there are no other means, is fulfilled in time-consuming and manual ways by means which have the information but not the function of comparison. Therefore, there is a need for a bibliographic website where researchers, authors and research teams will be compared based on the number and type of publications and their citations. Such a site would be particularly useful to staff responsible for conducting either internal or external research evaluation processes at universities and their departments, as well as to the committees and electorate bodies responsible for selecting permanent academic staff, researchers and temporary staff in universities and research centers as well as to fill positions of responsibility in scientific areas and organizations of both the public and the private sector.

This need is the motivation for the elaboration of the present work. Thus, an application was implemented which provides a variety of researchers, monitoring and analyzing their work comparatively. There is now the possibility of displaying bibliometric information collectively with reliable data offered by the most well-known bibliometric and scientific websites. Through the application, someone can create groups of researchers who can easily search. Then, he can compare these groups as a whole, either individually or selectively. The results are collected from reliable sources and presented in the form of graphs for clarification and better understanding by the user.

Περιεχόμενα

Περίληψη.....	3
Abstract.....	4
Κεφάλαιο 1ο:Εισαγωγή.....	11
1.1Γενικά.....	11
1.2Κίνητρο.....	14
1.3Συνεισφορά.....	14
1.4Οργάνωση.....	15
Κεφάλαιο 2ο:Διαδικτυακοί τόποι βιβλιογραφίας.....	16
2.1dblp.....	16
2.1.1Πληροφορίες.....	16
2.1.2Χρήση του API.....	18
2.2Google Scholar.....	19
Κεφάλαιο 3ο:Τεχνολογίες που χρησιμοποιήθηκαν.....	20
3.1Back-end.....	20
3.1.1Java.....	21
3.1.2MVC.....	22
3.1.3Spring – Spring Boot.....	24
3.1.4Rest – Rest API.....	28
3.1.5HtmlUnit.....	30
3.1.6Βάση Δεδομένων.....	31
3.2Front-end.....	32
3.2.1Bootstrap.....	32
3.2.2jQuery.....	33
Κεφάλαιο 4ο:Σχεδίαση και Υλοποίηση της Εφαρμογής.....	35
4.1Σχεδίαση και Υλοποίηση του back-end.....	35
4.1.1Model.....	35
4.1.2DTO.....	38
4.1.3Repository.....	40
4.1.4Service.....	42

4.1.5Controller.....	47
4.1.6Cross-Origin Resource Sharing (CORS).....	49
4.1.7Flyway.....	50
4.1.8Σύνδεση με τη Βάση Δεδομένων.....	56
4.1.9Scraping.....	57
4.1.10Scraping με τη χρήση νημάτων στη Java.....	61
4.1.11Διαθέσιμες μέθοδοι του API.....	65
4.2Υλοποίηση Front – end.....	67
4.3Οδηγός Χρήσης Sciento.....	69
Κεφάλαιο 5ο:Συμπεράσματα και προτάσεις βελτίωσης.....	73
5.1Συμπεράσματα.....	73
5.2Μελλοντικές επεκτάσεις.....	73
5.2.1Δημιουργία προγράμματος για την απαλοιφή των αυτοαναφορών του Google Scholar.....	73
5.2.2Ελαχιστοποίηση των “ανενεργών” χρόνων του χρήστη.....	75
BIBΛΙΟΓΡΑΦΙΑ.....	75

Ευρετήριο εικόνων

Εικόνα 1: Δημοσιεύσεις ανά χρονιά σύμφωνα με τα δεδομένα του dblp. Πηγή: https://dblp.org/statistics/publicationsperyear.html	15
Εικόνα 2: Κατανομή των δημοσιεύσεων σύμφωνα με τον τύπο τους. Πηγή : https://dblp.org/statistics/distributionofpublicationtype.html	16
Εικόνα 3: Αποτέλεσμα κλήσης στο API του dblp για αναζήτηση συγγραφέα.....	18
Εικόνα 4: Μοντέλο MVC. Chaitya Shah. Write your own MVC from scratch in PHP. Available at: https://chaitya62.github.io/2018/04/29/Writing-your-own-MVC-from-Scratch-in-PHP.html	21
Εικόνα 5: Κύκλος ζωής ενός request του χρήστη και η πορεία του εντός του API.....	22
Εικόνα 6: Annotations της Main μεθόδου για την ενεργοποίηση και τη χρήση του Spring Boot Framework	25
Εικόνα 7: Spring Boot dependencies εσωτερικά του αρχείου pom.xml.....	26
Εικόνα 8: Χρήση του @Query σε μέθοδο για τη συγγραφή ερωτήματος προς τη βάση.....	26
Εικόνα 9: Παράδειγμα Automatic Custom Queries του Jpa Repository.....	27
Εικόνα 10: Χρήση του @Column σε πεδίο μιας κλάσης Entity και σύνδεση με το ανρίστοιχο πεδίο του πίνακα στη βάση δεδομένων.....	35
Εικόνα 11: Annotations για την δήλωση μιας κλάσης ως Entity.....	35
Εικόνα 12: Χρήση του @OneToMany για τη δημιουργία σχέσης 1:N με πίνακα της βάσης δεδομένων	36
Εικόνα 13: Χρήση του @ManyToOne για τη δημιουργία σχέσης N:1 με πίνακα της βάσης δεδομένων	36
Εικόνα 14: Δήλωση του primary key στο αντίστοιχο πεδίο του Entity και χρήση του @GeneratedValue για την auto-incremental στρατηγική.....	37
Εικόνα 15: Χρήση του @JsonIgnore για την απαλοιφή της αναδρομικότητας των αποτελεσμάτων....	38
Εικόνα 16: Αναδρομικό αποτέλεσμα χωρίς τη χρήση @JsonIgnore.....	38
Εικόνα 17: Παράδειγμα αντιστοίχισης των Model κλάσεων με τις κλάσεις των DTO.....	39
Εικόνα 18: Υλοποίηση της κλάσης GroupDTO και λειτουργία του Constructor ως μέθοδο μετατροπής ενός Entity σε DTO.....	39
Εικόνα 19: Παράδειγμα αντιστοίχισης των κλάσεων Model με κλάσεις του Repository.....	40

Εικόνα 20: Παράδειγμα υλοποίησης του UserRepository με extension του JpaRepository.....	40
Εικόνα 21: Χρήση του @Service στις Service κλάσεις.....	42
Εικόνα 22: Χρήση του @Autowired για την σύνδεση των Services με τα Repositories.....	42
Εικόνα 23: Η υλοποιημένη μέθοδος για την αναζήτηση όλων των User.....	43
Εικόνα 24: Υλοποιημένη μέθοδος για την εύρεση ενός User με το μοναδικό id του.....	44
Εικόνα 25: Υλοποιημένη μέθοδος για την αποθήκευση ενός User από αντικείμενο τύπου DTO.....	46
Εικόνα 26: Μέθοδος για την επιβεβαίωση του κωδικού ενός χρήστη κατά την είσοδό του στην web εφαρμογή.....	46
Εικόνα 27: Μέθοδος για την διαχείριση της αλλαγής κωδικού ενός χρήστη.....	46
Εικόνα 28: Κλήση της μεθόδου deleteById του JpaRepository για τη διαγραφή ενός User.....	47
Εικόνα 29: Μέθοδος μετρατροπής μιας κλάσης DTO σε Entity.....	47
Εικόνα 30: Annotations ενός Controller.....	48
Εικόνα 31: Παράδειγμα χρήσης του @GetMapping.....	49
Εικόνα 32: Παραδείγματα PUT και POST mappings.....	49
Εικόνα 33: Παράδειγμα με σταθερές που δεχθεί παραμετρικά το CORS policy.....	50
Εικόνα 34: Ενεργοποίηση του CORS policy με προ-δηλωμένες σταθερές.....	51
Εικόνα 35: Δήλωση του flyway στη λίστα των dependencies και ένταξή του στο Project.....	51
Εικόνα 36: Τοποθεσία των database versions εντός του project.....	52
Εικόνα 37: Παραμετροποίηση του Flyway στο .properties του project.....	52
Εικόνα 38: Περιεχόμενα του πίνακα schema_version στη βάση δεδομένων.....	53
Εικόνα 39: Παραμετροποίηση στοιχείων για τη σύνδεση στη βάση δεδομένων.....	54
Εικόνα 40: Schema της MySQL βάσης της εφαρμογής.....	54
Εικόνα 41: Αρχικοποίηση των options του WebClient.....	55
Εικόνα 42: Δημιουργία της HtmlPage σύμφωνα με το profile του συγγραφέα στη σελίδα του Google Scholar.....	56
Εικόνα 43: Παράδειγμα συλλογής πληροφορίας από Div με μέθοδο scraping.....	56
Εικόνα 44: Χρήση του DocumentBuilderFactory για την ανάλυση του rss αρχείου.....	57
Εικόνα 45: Μέθοδος ομαδοποίησης των δεδομένων ανά χρονιά.....	57
Εικόνα 46: Μέθοδος συλλογής των τύπων των δημοσιεύσεων.....	58

Εικόνα 47: Υλοποίηση του νήματος για τις λειτουργίες του Google Scholar.....	59
Εικόνα 48: Υλοποίηση νήματος για τις λειτουργίες του dblp.....	59
Εικόνα 49: Αρχικοποίηση των Thread.....	59
Εικόνα 50: Μέθοδος έναρξης λειτουργίας των Thread.....	60
Εικόνα 51: Μέθοδος τερματισμού των Thread και συλλογή των πληροφοριών τους.....	60
Εικόνα 52: Μέθοδος που δέχεται τα request των χρηστών για τις πληροφορίες του Google Scholar...	62
Εικόνα 53: Παράδειγμα αποτελέσματος μιας κλήσης για συγκεκριμένο συγγραφέα.....	63
Εικόνα 54: Παράδειγμα αποτελέσματος της κλήσης με type = "all"	64
Εικόνα 55: Οδηγός Χρήσης Sciento: Σύνδεση/Δημιουργία λογαριασμού.....	67
Εικόνα 56: Οδηγός Χρήσης Sciento: Σύνδεση σε λογαριασμό.....	67
Εικόνα 57: Οδηγός χρήσης Sciento: Δημιουργία λογαριασμού.....	68
Εικόνα 58: Οδηγός Χρήσης Sciento: Διαθέσιμες λειτουργίες συγγραφέα.....	69
Εικόνα 59: Οδηγός Χρήσης Sciento: παράδειγμα ομάδας.....	70
Εικόνα 60: Οδηγός Χρήσης Sciento: Σύγκριση ομάδων.....	70

Ευρετήριο πινάκων

Πίνακας 1: Χρόνοι εκτέλεσης των μεθόδων scraping και για το Google Scholar και για το dblp χωρίς τη χρήση νημάτων (σε δευτερόλεπτα).....	67
Πίνακας 2: Χρόνοι εκτέλεσης των μεθόδων scraping και για το Google Scholar και για το dblp με τη χρήση νημάτων (σε δευτερόλεπτα).....	67

Κεφάλαιο 1ο: Εισαγωγή

1.1 Γενικά

Η επιστημονομετρία είναι μια μεθοδολογική προσέγγιση η οποία αναλύει την επιστημονική βιβλιογραφία, μπορεί δηλαδή να θεωρηθεί η επιστήμη της επιστήμης [1]. Είναι σημαντική στον κλάδο της επιστήμης διότι με αυτή μπορεί να μετρηθεί η εξέλιξη ενός επιστημονικού τομέα σύμφωνα με το αντίκτυπο που έχουν οι επιστημονικές δημοσιεύσεις στον τομέα αυτόν. Επιπλέον, μέσω της επιστημονομετρίας παρακολουθείται η έρευνα και αξιολογείται η επιστημονική συμβολή των συγγραφέων είτε στον κάθε τομέα τους ή γενικά στην συμβολή τους στην επιστημονική κοινότητα. Αναλύονται έργα συγγραφέων είτε σε περιοδικά είτε σε συγκεκριμένα έργα με σκοπό να αποκτηθούν και να αναλυθούν οι πληροφορίες που έχουν σχέση με τη διάδοση της επιστημονικής γνώσης μέσα από τα έργα αυτά. Για να αναλυθούν οι πληροφορίες αυτές, οι ερευνητές χρησιμοποιούν συγκεκριμένες μεθόδους, όπως είναι η ανάλυση των παραπομπών του έργου, η ανάλυση του κοινωνικού δικτύου, ακόμη και η ανάλυση των λέξεων.

Το Wordstat [2] και το QDA Miner [3] είναι δύο από τα πιο γνωστά εργαλεία εξόρυξης και ανάλυσης αυτών των πληροφοριών. Μια από τις πρώτες μεθόδους που χρησιμοποιήθηκαν ήταν η ανάλυση των λέξεων στους τίτλους των άρθρων, όπου η πρώτη ανάλυση έγινε το 1999 για τους τίτλους 634 άρθρων. Με την ανάλυση κοινών λέξεων και το ιεραρχικό clustering βρέθηκαν σχέσεις μεταξύ των άρθρων αλλά και αλλαγές στη θεματολογία με τον χρόνο [4]. Οι περιλήψεις είναι επίσης καλές ενδείξεις για να ξεχωριστεί το περιεχόμενο της δημοσίευσης καθώς παρέχουν λεπτομερείς περιγραφή του πεδίου του άρθρου, τη μεθοδολογική του προσέγγιση και τα θεωρητικά του θεμέλια [5]. Επίσης οι λέξεις-κλειδιά είναι καλοί δείκτες για το περιεχόμενο ενός άρθρου και έχουν χρησιμοποιηθεί σε πολλές επιστημονομετρικές μελέτες. Το 2008 χρησιμοποιήθηκαν λέξεις κλειδιά και τίτλοι από 175.000 άρθρα από 68 περιοδικά γεωλογίας για να βρεθούν αλλαγές στην επιρροή των κλάδων με το χρόνο [6].

Οι σύγχρονες μέθοδοι επιστημονομετρίας βασίζονται κυρίως στη δουλειά του Derek J. de Solla Price και του Eugene Garfield. Ο Eugene Garfield ίδρυσε το Ινστιτούτο Επιστημονικών Πληροφοριών (Institute for Scientific Information) το οποίο χρησιμοποιείται για ανάλυση των επιστημονομετρικών μεθόδων [7]. Στα τέλη του αιώνα η αξιολόγηση και η κατάταξη των επιστημόνων και των ιδρυμάτων βρέθηκαν στο προσκήνιο. Η επιστημονομετρία έγινε ένα σημαντικό εργαλείο και το Times Higher Education World University Rankings (THE-ranking) έγινε ένας κορυφαίος δείκτης για την κατάσταση και κατάταξη των πανεπιστημίων [8]. Όσον αφορά τους επιστήμονες, ο δείκτης που προσομοιώνει την παραγωγικότητα και το αντίκτυπο της εργασίας του στον επιστημονικό τομέα, είναι ο h (h -index), ενώ πρόσφατα έχουν προταθεί εναλλακτικοί δείκτες για το επίπεδο του συγγραφέα [9].

Οι ερευνητές γρήγορα ανακάλυψαν την σπουδαιότητα του Web ακόμη και για την επιστημονομετρία. Το Web περιέχει ένα τεράστιο πλήθος πληροφοριών και προσφέρει νέες ευκαιρίες

για την εξέταση, τη μελέτη και τη μέτρηση των επιπτώσεων των επιστημονικών άρθρων. Η πρόσβαση σε όλη αυτή τη πληροφορία εφηύρε νέους τρόπους σύγκρισης και ανάλυσης άρθρων και συγγραφέων. Μια σχετική προσέγγιση είναι η ανάλυση δεδομένων λήψης για τα άρθρα η οποία ξεκίνησε με την ανάρτηση της ακαδημαϊκής βιβλιογραφίας στο Web και επιτρέπει την εξέταση των προβολών και των λήψεων για τα άρθρα. Ο Marek και ο Valauskas το 2002 [10] εξέτασαν τα αρχεία καταγραφής του περιοδικού First Monday για να εντοπίσουν ερευνητικά άρθρα τα οποία λήφθηκαν επανειλημμένα μεταξύ του 1999 και του 2001 ως εναλλακτική λύση στην ανάλυση των παραπομπών (citations). Επίσης, το 2009 η αναφορά του έργου MESUR, με την οποία έγινε μια ολοκληρωμένη εξέταση 300 εκατομμυρίων αλληλεπιδράσεων των χρηστών σε ένα ευρύ φάσμα επιστημονικών κλάδων, αφορούσε την ορθότητα των δεδομένων και την ευρεία χρησιμότητα των μετρήσεων με βάση τα δεδομένα στο Web [11].

Παρά αυτά τα ενθαρρυντικά αποτελέσματα αυτή η προσέγγιση περιορίζεται στο γεγονός ότι η ανάλυση βασίζεται εξ ολοκλήρου σε μηχανές αναζήτησης για τα δεδομένα, τα οποία ενδέχεται να μην παρέχουν APIs είτε να παρέχουν αλλά να μην είναι χρήσιμα. Συχνά παρέχονται μη δομημένα HTML και XML αρχεία από τα οποία είναι αρκετά δύσκολο να δομηθεί και να συλλεχθεί η πληροφορία. Ωστόσο, οι αδυναμίες αυτές υποδηλώνουν την ανάγκη για πιο δομημένες, καταναεμμένες και εύκολα προσβάσιμες πηγές δεδομένων.

Με την ανάπτυξη της τεχνολογίας, την ευκολότερη διάδοση της γνώσης και την σημαντική ευκολία πρόσβασης σε δεδομένα τη τελευταία εικοσαετία δημιουργήθηκαν πολλοί διαδικτυακοί τόποι με σκοπό την εύκολη πρόσβαση στη βιβλιογραφία, του συγγραφείς και τις δημοσιεύσεις τους.

Το Google Scholar [12] είναι μια από τις πιο γνωστές βιβλιογραφικές σελίδες και μια από τις πιο εύκολες στη χρήση και για τους απλούς χρήστες αλλά και για τους ίδιους τους ερευνητές. Με μια απλή αναζήτηση μπορεί κάποιος εύκολα να βρει αυτό που αναζητά, είτε είναι κάποια δημοσίευση, κάποιο άρθρο, κάποια έρευνα είτε είναι κάποιος ερευνητής. Εκτός των δημοσιεύσεων, το Google Scholar δίνει μεγάλη έμφαση και στις αναφορές των δημοσιεύσεων, εμφανίζοντας αρκετές πληροφορίες και για το σύνολο των αναφορών του συγγραφέα και για το σύνολο των αναφορών ανά δημοσίευση.

Το dblp [13] είναι και αυτή μια βιβλιογραφική σελίδα με έμφαση στην επιστήμη των υπολογιστών. Οι ερευνητές δεν έχουν πρόσβαση σε προσωπικό τους λογαριασμό σε αντίθεση με το Google Scholar. Ξανά, με μια εύκολη αναζήτηση μπορεί κάποιος να βρεί συγγραφείς, περιοδικά, συνέδρια και βιβλία από τον τομέα της επιστήμης των υπολογιστών. Το dblp στοχεύει στις δημοσιεύσεις ενός συγγραφέα και στον τύπο τους. Προσφέρει αρκετά στατιστικά που αφορούν τον τύπο κάθε δημοσίευσης είτε ανά χρονιά, είτε στο σύνολο όλων των δημοσιεύσεων.

Το Scopus [14] παρέχει ίσως τη μεγαλύτερη βάση δεδομένων όσον αφορά περιλήψεις και αναφορές βιβλιογραφιών για επιστημονικά περιοδικά, βιβλία και συνεδρίες. Παρέχει μια

ολοκληρωμένη επισκόπηση του αντίκτυπου που έχουν στη παγκόσμια έρευνα οι τομείς της επιστήμης, της ιατρικής και των ανθρωπιστικών επιστημών. Διαθέτει έξυπνα εργαλεία για την παρακολούθηση και την ανάλυση τις έρευνας. Οι ερευνητές μπορούν να χρησιμοποιήσουν το Scopus για να βοηθήσουν την έρευνα τους καθώς μπορούν να αναζητήσουν εύκολα συγγραφείς και να μάθουν αρκετά για το θέμα και το περιεχόμενο της έρευνάς τους.

Το ORCID [15] είναι μέρος μιας ευρύτερης ψηφιακής υποδομής η οποία παρέχει την δυνατότητα στους συγγραφείς και στους ερευνητές να μοιράζονται πληροφορίες σε παγκόσμια κλίμακα. Παρέχει ένα μοναδικό αναγνωριστικό για τα άτομα που χρησιμοποιούν το όνομα τους καθώς ασχολούνται με δραστηριότητες έρευνας, προσφέροντας έτσι διαφανείς και αξιόπιστες συνδέσεις μεταξύ των ερευνητών. Σύμφωνα με το ORCID το όραμά τους είναι ένας κόσμος όπου όλοι όσοι συμμετέχουν σε μια έρευνα, μια υποτροφία ακόμη και σε μία καινοτομία αναγνωρίζονται και συνδέονται εύκολα και άμεσα με την συνεισφορά τους.

Όλοι οι βιβλιογραφικοί ιστότοποι στοχεύουν στην προσφορά όσο το δυνατόν μεγαλύτερου όγκου δεδομένων και στην εύκολη πρόσβαση σε αυτή τη πληροφορία. Μια πολύ μεγάλη ανάγκη η οποία δεν παρέχεται από αυτούς τους διαδικτυακούς τόπους είναι η σύγκριση των συγγραφέων μεταξύ τους. Η ανάγκη προέρχεται από την πολύωρη αναζήτηση, πιθανώς σε όλους τους παραπάνω βιβλιογραφικούς ιστότοπους, για την συλλογή όσο το δυνατόν περισσότερων πληροφοριών ενός ερευνητή. Η ανάγκη αυτή αυξάνεται ακόμη περισσότερο όταν θα πρέπει να γίνει σύγκριση μεταξύ δύο και παραπάνω ερευνητών στην περίπτωση όπου θα πρέπει να γίνει επιλογή κάποιων από αυτούς για μια οποιαδήποτε θέση.

1.2 Κίνητρο

Παρά την πολύτιμη χρησιμότητα των διαδικτυακών τόπων βιβλιογραφίας με την εύκολη πρόσβαση και εύρεση οποιασδήποτε πληροφορίας αναζητηθεί σε έναν τεράστιο όγκο δεδομένων, δεν δίνουν λύση στην ανάγκη σύγκρισης ερευνητών και συγγραφέων. Η ανάγκη εύκολης σύγκρισης μεταξύ επιστημόνων υπήρχε πάντα και είναι μια πολύχρονη διαδικασία με την οποία θα πρέπει να συλλεχθούν πληροφορίες που αφορούν τις δημοσιεύσεις, τον τύπο αυτών των δημοσιεύσεων, τις αναφορές και το αντίκτυπο των δημοσιεύσεων και των ερευνών του κάθε συγγραφέα από διάφορους βιβλιογραφικούς ιστότοπους.

Μια σελίδα που δίνει τη δυνατότητα της εύκολης σύγκρισης μεταξύ ερευνητών και επιστημόνων δεν υπάρχει. Μια σελίδα όπου ο χρήστης θα μπορεί και πάλι να αναζητήσει εύκολα συγγραφείς και να δει τις πιο σημαντικές πληροφορίες των έργων του, όπως είναι οι δημοσιεύσεις του συνολικά και ανά χρονιά και ο τύπος τους και έπειτα να τους συλλέξει είτε ατομικά είτε ομαδικά με

σκοπό να τους συγκρίνει. Η σύγκριση αυτή θα πρέπει να γίνεται εύκολη στο χρήστη με αξιόπιστα δεδομένα και κυρίως με ευκολία στη διάκριση των αποτελεσμάτων, όπως η χρήση γραφικών.

Ένας τέτοιος βιβλιογραφικός ιστότοπος θα ήταν επίσης χρήσιμος στο προσωπικό που είναι επιφορτισμένο για τη διεξαγωγή των διαδικασιών εσωτερικής και εξωτερικής αξιολόγησης της έρευνας στα πανεπιστήμια και τα τμήματά τους. Χρήσιμος θα ήταν επίσης και σε επιτροπές και ερευνητικά κέντρα τα οποία έχουν ανάγκη κάλυψης θέσεων ευθύνης. Η συγκριτική ανάλυση απαντά σε ερωτήσεις σχετικά με το πως και αν μπορεί ένας ερευνητής να βοηθήσει τη συγκεκριμένη δομή που αναζητά ένα άτομο ή κάποια ομάδα ατόμων για κάποια θέση.

1.3 Συνεισφορά

Για τους λόγους που αναφέρθηκαν στα κεφάλαια 1.1 και 1.2 δημιουργήθηκε η web εφαρμογή Sciento όπου ο χρήστης μπορεί να αναζητήσει γρήγορα συγγραφείς με το όνομα τους και να δει στατιστικές πληροφορίες των δημοσιεύσεων και των αναφορών τους, οι οποίες συλλέγονται με μεθόδους scraping από γνωστές βιβλιομετρικές σελίδες. Επιπλέον, μπορεί να εντάξει τις αναζητήσεις του σε ομάδες όπου μπορεί να βλέπει αναλυτικά τα στατιστικά κάθε μέλους, αλλά και τα συνολικά στατιστικά της ομάδας.

Το Sciento δίνει την δυνατότητα σύγκρισης μεταξύ των παραπάνω ομάδων. Ο χρήστης μπορεί να επιλέξει δύο από τις ομάδες που έχει δημιουργήσει και να τις συγκρίνει είτε στο σύνολό τους, είτε επιλέγοντας συγκεκριμένα μέλη από κάθε ομάδα. Τα αποτελέσματα των συγκρίσεων εμφανίζονται σε διαφόρων ειδών γραφημάτων και αφορούν το σύνολο των δημοσιεύσεων ανά χρονιά, το σύνολο των τύπων των δημοσιεύσεων και το σύνολο των αναφορών ανά χρονιά. Με αυτόν τον τρόπο μπορεί εύκολα να κάνει τις απαραίτητες συγκρίσεις μεταξύ των δύο ομάδων και να βγάλει τα ανάλογα συμπεράσματα τα οποία στην συνέχεια μπορούν να φανούν χρήσιμα στη λήψη κρίσιμων αποφάσεων όπως για παράδειγμα η επιλογή προσωπικού για την κάλυψη κενών ακαδημαϊκών θέσεων.

Ένας χρήστης μπορεί να δημιουργήσει λογαριασμό στην web εφαρμογή, δίνοντας όνομα, επώνυμο, email, όνομα χρήστη και κωδικό. Οι ομάδες που έχουν δημιουργήσει οι εγγεγραμμένοι χρήστες αποθηκεύονται σε βάση δεδομένων ενώ έχουν πρόσβαση σε ορισμένες λειτουργίες scraping που παρέχει το API, ώστε να έχουν πρόσβαση στις μεθόδους και τις πληροφορίες που συλλέγονται οποτεδήποτε για δική τους χρήση. Οι μη εγγεγραμμένοι χρήστες δεν έχουν την δυνατότητα να αποθηκεύουν τις ομάδες που έχουν δημιουργήσει με αποτέλεσμα να πρέπει να τις ξαναδημιουργούν με κάθε σύνδεση τους στην web σελίδα. Δεν παύουν όμως να μπορούν να αναζητούν συγγραφείς, να δημιουργούν ομάδες με μέλη και να τις συγκρίνουν μεταξύ τους.

Το dblp και το Google Scholar είναι δικτυακοί τόποι βιβλιογραφίας από τους οποίους συλλέγονται οι πληροφορίες που αφορούν τις δημοσιεύσεις και τις αναφορές των ερευνητών χρησιμοποιώντας κατάλληλες μεθόδους scraping. Οι μέθοδοι που υλοποιούν τις λειτουργίες συλλογής αυτών των πληροφοριών είναι διαθέσιμες προς τους προγραμματιστές μέσω του API ώστε να έχουν οι ίδιοι πρόσβαση σε αυτά τα δεδομένα.

1.4 Οργάνωση

Στο Κεφάλαιο 2 αναλύονται οι δύο από τους πιο γνωστούς Διαδικτυακούς τόπους βιβλιογραφίας. Είναι οι τόποι από τους οποίους συλλέγεται η πληροφορία με τις μεθόδους scraping οι οποίοι αναφέρονται στα επόμενα κεφάλαια. Στο πρώτο μέρος αναφέρεται η βιβλιογραφική ιστοσελίδα dblp και τα δεδομένα που προσφέρει και στο δεύτερο παρουσιάζεται το Google Scholar.

Στο Κεφάλαιο 3 παρουσιάζονται οι βασικές τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής. Αναφέρεται η πορεία τους ιστορικά, η χρησιμότητα τους τόσο στην εφαρμογή αλλά και γενικότερα και αναφέρονται οι βασικές αρχές οι οποίες θα πρέπει να ακολουθηθούν ώστε να μπορεί μια εφαρμογή να είναι όσο το δυνατόν καλύτερα δομημένη, εύκολα επεκτάσιμη και σωστά διαχειρίσιμη.

Στο Κεφάλαιο 4 αναλύεται η υλοποίηση της εφαρμογής. Το κεφάλαιο χωρίζεται σε δύο μέρη. Στο πρώτο μέρος αναλύεται η υλοποίηση του back-end με τη χρήση της γλώσσας Java και την ενσωμάτωση του Spring και Spring Boot Framework. Η ροή της υλοποίησης παρουσιάζεται αναλυτικά ώστε να υπάρχει η πλήρης εικόνα από οποιοδήποτε σημείο του κώδικα με σκοπό να μπορεί να γίνει όσο το δυνατόν πιο κατανοητή. Σε αυτό το μέρος αναφέρονται και οι υλοποιήσεις που αφορούν τις μεθόδους scraping και τεχνικές με τις οποίες επιτεύχθηκε η ταχύτερη συλλογή των πληροφοριών. Στο δεύτερο μέρος αναλύονται οι τρόποι με τους οποίους υλοποιήθηκε το front-end καθώς και μέθοδοι με τις οποίες γίνεται ανάλυση των δεδομένων και προετοιμασία τους για εμφάνιση στα γραφήματα.

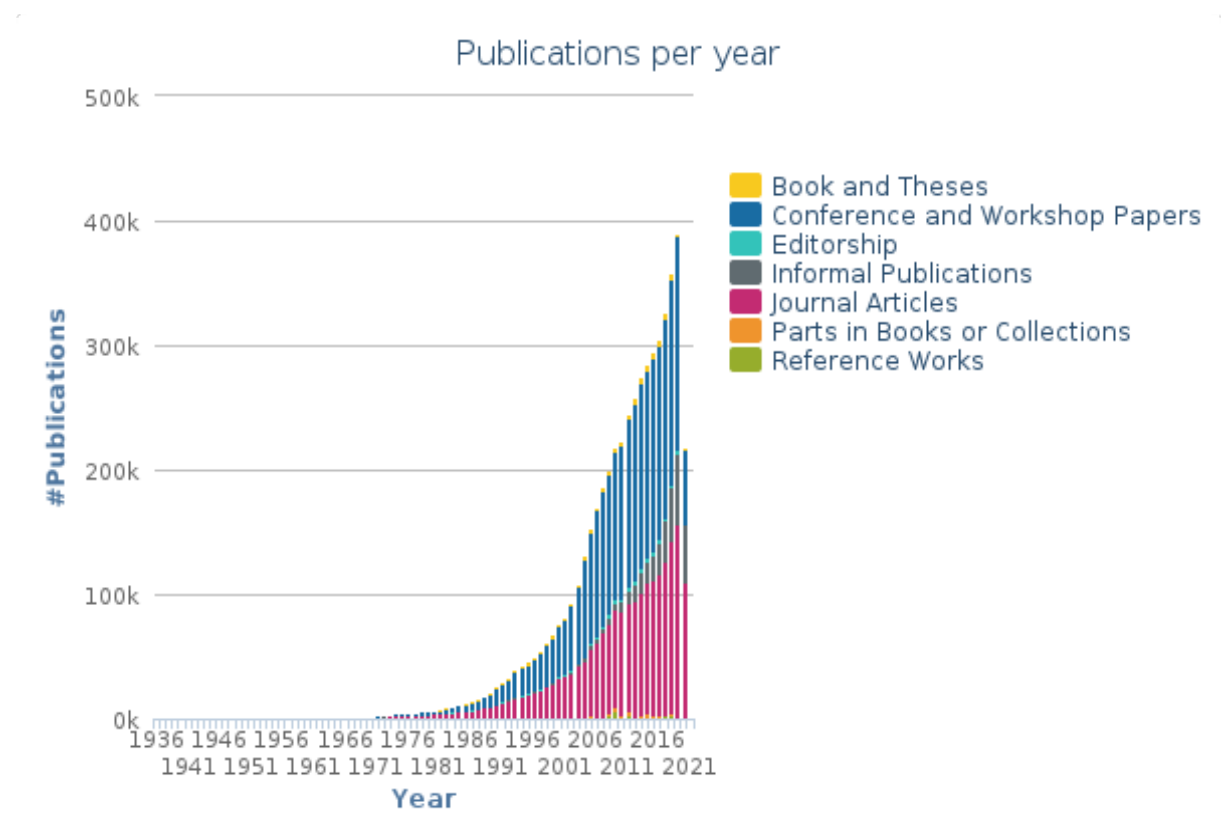
Τέλος, στο Κεφάλαιο 5 δίνεται το τελικό συμπέρασμα που σχηματίστηκε με την ολοκλήρωση της έρευνας και της εφαρμογής και παράλληλα παρουσιάζονται οι μελλοντικές επεκτάσεις που θα υλοποιηθούν για την ακόμη πιο αποδοτική και λειτουργική χρήση της εφαρμογής.

Κεφάλαιο 2ο: Διαδικτυακοί τόποι βιβλιογραφίας

2.1 dblp

2.1.1 Πληροφορίες

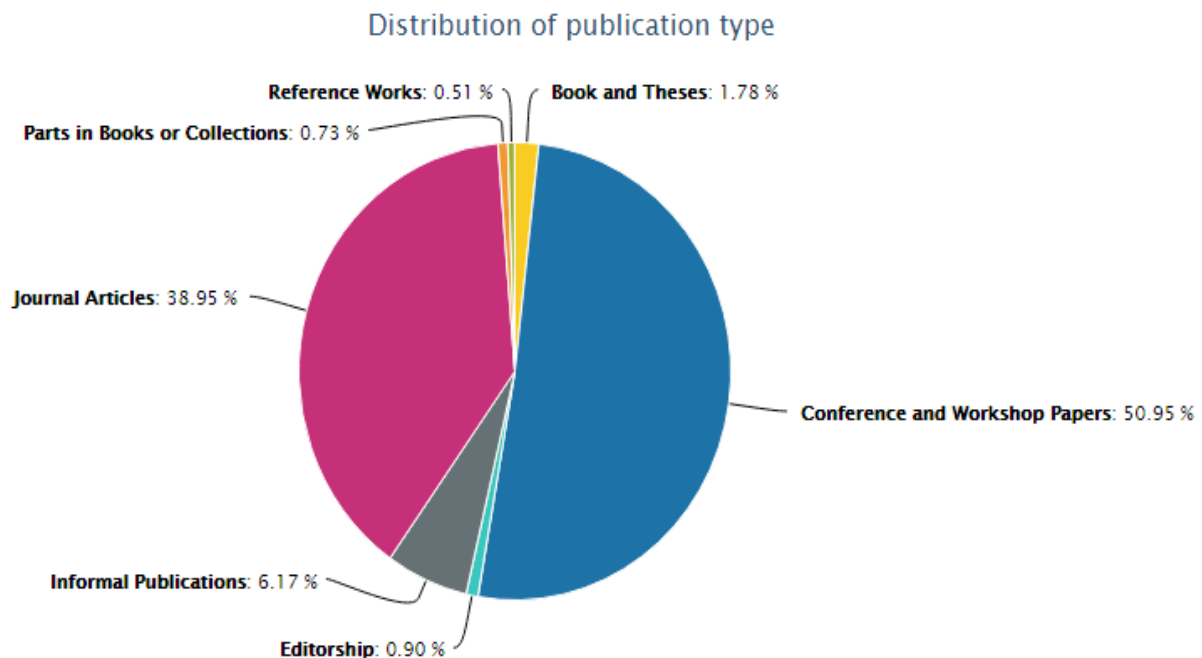
Το dblp computer science bibliography είναι ένας διαδικτυακός ιστότοπος για βιβλιογραφικές πληροφορίες με κυρίως θέμα την επιστήμη των υπολογιστών. Σύμφωνα με τους ανθρώπους του, εξελίχθηκε από ένα μικρό πειραματικό web server σε μια διάσημη υπηρεσία με προσβάσιμα δεδομένα από όλους. Σκοπός του dblp είναι να παρέχει βοήθεια στους επιστημονικούς ερευνητές με τη πρόσβαση σε ένα τεράστιο όγκο βιβλιογραφικών δεδομένων με link για τις ηλεκτρονικές εκδόσεις των δημοσιεύσεων.



Εικόνα 1: Δημοσιεύσεις ανά χρονιά σύμφωνα με τα δεδομένα του dblp. Πηγή: <https://dblp.org/statistics/publicationsperyear.html>

Μέχρι και τον Ιούλιο του 2020, το dblp προβάλλει τα δεδομένα για 5.2 εκατομμύρια δημοσιεύσεις και πληροφορίες από 2.3 εκατομμύρια συγγραφείς. Σύμφωνα με τα στατιστικά του dblp,

φαίνεται η ραγδαία αύξηση των δημοσιεύσεων ανά χρονιά καθώς το 2000 σημειώνονται περίπου 76.500, ενώ το 2019 ξεπερνούν τις 390.500 δημοσιεύσεις (εικόνα 1).



Εικόνα 2: Κατανομή των δημοσιεύσεων σύμφωνα με τον τύπο τους. Πηγή : <https://dblp.org/statistics/distributionofpublicationtype.html>

Μια ακόμη σημαντική πληροφορία που παρέχει το dblp είναι η διανομή όλων αυτών δημοσιεύσεων ανάλογα τον τύπο της δημοσίευσης. Μια δημοσίευση μπορεί να είναι Book and Theses, Conference and Workshop Paper, Editorship, Informal Publication, Journal Article, Parts in Books or Collections ή Reference Work. Το μεγαλύτερο μερίδιο των δημοσιεύσεων αφορούν Conference and Workshop Papers τα οποία αποτελούν περίπου το 51% των συνολικών δημοσιεύσεων ενώ σε επίσης μεγάλο ποσοστό είναι τα Journal Articles με περίπου 39% (εικόνα 2).

Το dblp παρέχει τα δεδομένα του και μέσω API. Συγκεκριμένα, δίνει τη δυνατότητα στον οποιοδήποτε να αναζητήσει και να δημιουργήσει queries στη βάση του που αφορούν :

- δημοσιεύσεις : με url βάση το <http://dblp.org/search/publ/api>
- συγγραφείς : με url βάση το <http://dblp.org/search/author/api>
- venue : με url βάση το <http://dblp.org/search/venue/api>

Τα παραπάνω url δέχονται ως παραμέτρους τα :

- **q** : δηλώνει το query για το οποίο θα γίνει αναζήτηση

- **format** : δηλώνει την μορφοποίηση την οποία θα έχει το αποτέλεσμα της αναζήτησης με διαθέσιμες τιμές “xml”, “json”, “jsonp”
- **h** : ο μέγιστος αριθμός αποτελεσμάτων που θα επιστρέψει με μέγιστο τα 1000 αποτελέσματα
- **f** : παράμετρος η οποία μαζί με την παράμετρο **q** μπορεί να χρησιμοποιηθεί για σελιδοποίηση των αποτελεσμάτων
- **c** : ο μέγιστος αριθμός αυτόματης συμπλήρωσης του query για κοινά στοιχεία

Για παράδειγμα, η κλήση του url: <https://dblp.org/search/author/api?q=newton&format=json> επιστρέφει τους συγγραφείς όπου το ονοματεπώνυμό τους περιέχουν το *newton* (εικόνα 3).

```

"result": {
  "query": "newton*",
  "status": {
    "@code": "200",
    "text": "OK"
  },
  "time": {
    "@unit": "msecs",
    "text": "1.46"
  },
  "completions": {
    "@total": "6",
    "@computed": "6",
    "@sent": "6",
    "c": [
      {
        "@sc": "1",
        "@dc": "219",
        "@oc": "227",
        "@id": "12633487",
        "text": "newton"
      },
      {
        "@sc": "1",
        "@dc": "1",
        "@oc": "1",
        "@id": "12633488",
        "text": "newton"
      }
    ]
  }
},
"hits": {
  "@total": "219",
  "@computed": "100",
  "@sent": "30",
  "@first": "0",
  "hit": [
    {
      "@score": "1",
      "@id": "100298",
      "info": {
        "author": "Newton Armstrong",
        "url": "https://dblp.org/pid/175/0485"
      },
      "url": "URL#100298"
    },
    {
      "@score": "1",
      "@id": "261787",
      "info": {
        "author": "Newton G. Bretas",
        "aliases": {
          "alias": "Newton Geraldo Bretas"
        },
        "notes": {
          "note": {
            "@type": "affiliation",
            "text": "University of Sao Paulo"
          }
        }
      }
    }
  ]
}

```

Εικόνα 3: Αποτέλεσμα κλήσης στο API του dblp για αναζήτηση συγγραφέα

2.2 Google Scholar

Το Google Scholar “παρέχει έναν απλό τρόπο ευρείας αναζήτησης επιστημονικής λογοτεχνίας”. Χρειάζεται η αναζήτηση μόνο από ένα σημείο για να βρεθούν αποτελέσματα για

άρθρα, διατριβές, βιβλία, περιλήψεις από ακαδημαϊκούς συγγραφείς, ερευνητές, εκδότες, επαγγελματικές εταιρείες και πανεπιστήμια. Οι πληροφορίες του αφορούν ένα τεράστιο πλήθος τομέων και βοηθά στην εύρεση εργασιών από όλο τον κόσμο της επιστημονικής έρευνας. Το Google Scholar στοχεύει στην κατάταξη των εγγράφων με τον τρόπο που τα κατατάσουν οι ίδιοι οι ερευνητές, ζυγίζοντας δηλαδή όλη τη πληροφορία του εγγράφου, το που δημοσιεύθηκε, ποιός ήταν ο συγγραφέας ακόμη και το πόσο συχνά έχει αναφερθεί σε άλλη δημοσίευση.

Από τα σημαντικά χαρακτηριστικά του Google Scholar είναι η βολική και εύκολη αναζήτηση από ένα μόνο μέρος καθώς και η εξερεύνηση σχετικών έργων, παραπομπών και συγγραφέων. Μέσω της ιστοσελίδας δίνεται η δυνατότητα να εντοπιστεί το πλήρες έγγραφο μιας δημοσίευσης και οι όλες οι παραπομπές της με αρκετά απλή για τον χρήστη διεπαφή. Τέλος, μπορεί ένας ερευνητής ή συγγραφέας να παρακολουθεί με εύκολο τρόπο τις δημοσιεύσεις του, να ελέγχει το ποιος αναφέρεται σε αυτές και να δημιουργήσει ένα δημόσιο προφίλ.

Το Google Scholar κατά κύριο λόγο μετράει τις αναφορές (citations) των δημοσιεύσεων. Οι αναφορές τεκμηριώνουν τις πηγές πληροφοριών που χρησιμοποιούνται στην ακαδημαϊκή έρευνα και χρησιμεύουν ως “διεύθυνση” αυτών των πληροφοριών. Οι μεταβλητές που χρησιμοποιεί το Scholar για την ανάδειξη ενός συγγραφέα, εκτός των citations, είναι το h-index και το i10-index. Το h-index μετράει και τη παραγωγικότητα και την επίπτωση που έχουν οι αναφορές μιας δημοσίευσης ενός συγγραφέα. Σχετίζεται επίσης με δείκτες επιτυχίας όπως είναι η αποδοχή για υποτροφίες έρευνας και η κατοχή θέσης σε κορυφαία πανεπιστήμια. Ορίζεται ως η μέγιστη τιμή του h έτσι ώστε ο συγκεκριμένος συγγραφέας να έχει δημοσιεύσει h άρθρα που έχουν αναφερθεί σε κάθε τουλάχιστον h ώρες. Το i10-index δημιουργήθηκε από το Google Scholar και αντιπροσωπεύει τον αριθμό των δημοσιεύσεων με τουλάχιστον 10 αναφορές. Αυτή η απλή μέτρηση χρησιμοποιείται μόνο από το Google Scholar και είναι ένας ακόμη τρόπος μέτρησης της παραγωγικότητας ενός έργου. Τα θετικά του i10-index είναι η απλότητα υπολογισμού του και η πολύ σημαντική πληροφορία που δείχνει, αλλά χρησιμοποιείται μόνο από το Google Scholar [16].

Επί του παρόντος, ο αριθμός των δημοσιεύσεων και ο αριθμός των αναφορών είναι ευρέως αποδεκτός ως ένας εύκολος και σωστός τρόπος σύγκρισης μεταξύ των επιστημόνων. Ο υπολογισμός τέτοιων στατιστικών εξαρτάται από τη διαθεσιμότητα μιας βάσης δεδομένων να παρέχει αυτές τις πληροφορίες συγκεντρωμένες και το Google Scholar κάνει αυτό ακριβώς. Στοχεύει στη παροχή αυτών των υπηρεσιών και αυτή τη στιγμή είναι η πιο ευρέως χρησιμοποιούμενη μηχανή αναζήτησης για επιστημονική και ακαδημαϊκή βιβλιογραφία. Ωστόσο, οι αναφορές που χρησιμοποιούνται για τον υπολογισμό των αναφορών καθώς και του h-index συμπεριλαμβάνουν τις αυτοαναφορές (self-citations) οι οποίες αποκλίνουν τους υπολογισμούς από τη πραγματικότητα και εν συνεχεία την αντανάκλαση του ερευνητικού αντίκτυπου. Μια λύση αυτού του προβλήματος δίνεται στο κεφάλαιο 5.2.1 στην οποία γίνεται προσπάθεια απομόνωσης των αυτοαναφορών με μεθόδους scraping απευθείας στο Google Scholar.

Κεφάλαιο 3ο: Τεχνολογίες που χρησιμοποιήθηκαν

3.1 Back-end

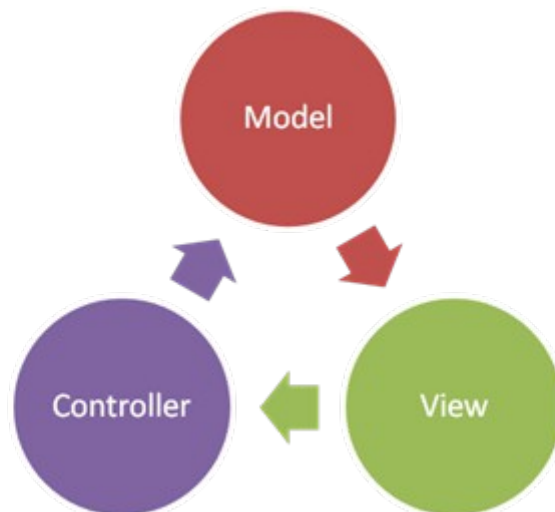
Η δημιουργία του back-end συστήματος υλοποιήθηκε με τη γλώσσα Java ακολουθώντας μία παραλλαγή του MVC μοντέλου και με την τεχνολογία της Spring και κυρίως της Spring Boot για την δημιουργία ενός REST API. Όσον αφορά το scraping χρησιμοποιήθηκε η βιβλιοθήκη HtmlUnit στη Java με την οποία δημιουργήθηκαν οι λειτουργίες συλλογής των πληροφοριών από το Google Scholar και το dblp.

3.1.1 Java

Η Java είναι ίσως η πιο γνωστή γλώσσα προγραμματισμού η οποία δημιουργήθηκε το 1995 από τον James Gosling [17] στην Sun Microsystems και παραμένει από τότε να είναι μία από τις βασικές γλώσσες για την υλοποίηση back-end εφαρμογών. Η Java ανήκει πλέον στην Oracle, η οποία εξαγόρασε την εταιρεία, και χρησιμοποιείται σε περισσότερες από 3 δισεκατομμύρια συσκευές και εφαρμογές έως σήμερα. Χρησιμοποιείται κυρίως σε desktop, mobile και web εφαρμογές, game development και συνδέσεις με βάσεις δεδομένων.

Οι λόγοι που παραμένει η Java μια από τις πιο γνωστές και βασικές γλώσσες προγραμματισμού είναι αρκετοί. Είναι μία αντικειμενοστραφής γλώσσα η οποία δίνει μία καθαρή δομή του προγράμματος και επιτρέπει την επαναχρησιμοποίηση του κώδικα είτε εντός του προγράμματος, είτε εκτός σε διαφορετικά προγράμματα ελαχιστοποιώντας το κόστος υλοποίησης. Ένα επίσης τεράστιο πλεονέκτημα είναι ότι λειτουργεί σε διαφορετικές πλατφόρμες όπως Windows, Linux, Mac και Raspberry ενώ ταυτόχρονα είναι ασφαλής, γρήγορη και με πάρα πολλές δυνατότητες [18]. Η Java είναι μια αρκετά εύκολη στην εκμάθηση γλώσσα προγραμματισμού για αυτό και έχει ένα τεράστιο community support το οποίο αποτελείται από δεκάδες εκατομμύρια προγραμματιστές με αποτέλεσμα να υπάρχει πρόσβαση σε τεράστιο πλήθος πληροφοριών σχεδόν για οποιαδήποτε λειτουργία. Ένας ακόμη λόγος που προτιμάται ακόμη και σήμερα είναι ο μεγάλος και ουσιαστικός όγκος βιβλιοθηκών που έχουν δημιουργηθεί ώστε να κάνουν επιτρέπουν στους προγραμματιστές να δημιουργούν και να υλοποιούν λειτουργίες ακόμη πιο γρήγορα και εύκολα. Βιβλιοθήκες για Logging και Testing βοηθούν τους προγραμματιστές στο debugging των εφαρμογών και βιβλιοθήκες για scraping και JSON, XML, Excel Parsing βοηθούν τους προγραμματιστές στο να δημιουργούν εφαρμογές χωρίς το άγχος της προσβασιμότητας σε δεύτερα συστήματα [19].

3.1.2 MVC



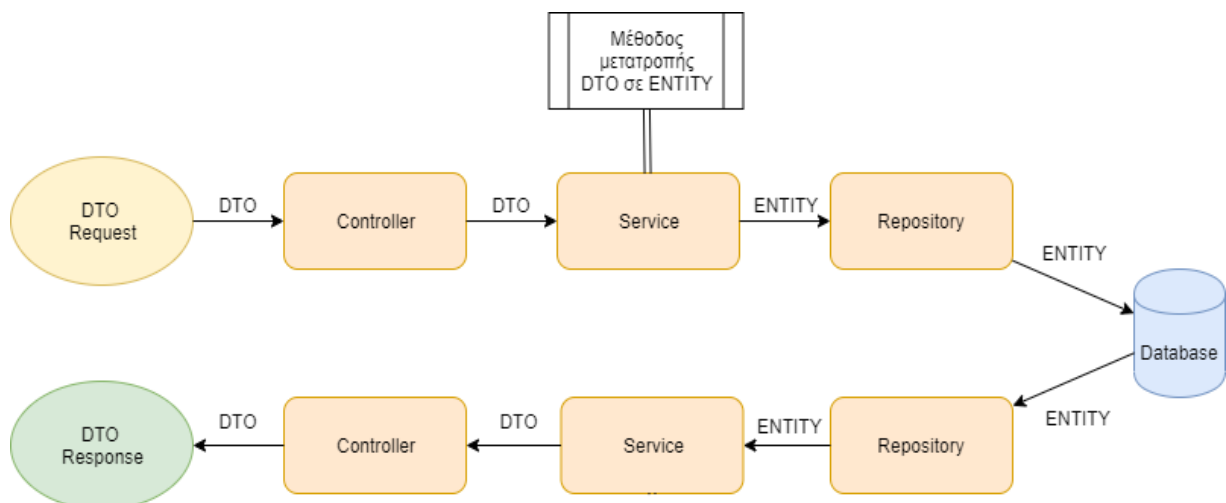
Εικόνα 4: Μοντέλο MVC. Chaitya Shah. Write your own MVC from scratch in PHP. Available at: <https://chaitya62.github.io/2018/04/29/Writing-your-own-MVC-from-Scratch-in-PHP.html>

Το MVC (εικόνα 4) είναι μια αρχιτεκτονική σχεδίασης λογισμικών η οποία χρησιμοποιείται συνήθως για την υλοποίηση διεπαφών χρήστη [20]. Η αρχιτεκτονική αυτή διαχωρίζει την εφαρμογή σε τρία βασικά μέρη το Model, το View και τον Controller. Κάθε ένα από αυτά τα μέρη είναι υπεύθυνο για την υλοποίηση διαφορετικών πτυχών στην εφαρμογή, κάθε μία με τη δική της λογική. Η βασική ιδέα του MVC είναι στο να διαχωριστούν τα κομμάτια του επιχειρησιακή λογική και τα κομμάτια της παρουσίασης ώστε να γίνεται “καθαρή” η υλοποίηση και κυρίως πιο “καθαρός” ο κώδικας της εφαρμογής ώστε να είναι εύκολα επεκτάσιμη. Το Model είναι υπεύθυνο για την επιχειρησιακή λογική της εφαρμογής και περιλαμβάνει όλα τα δεδομένα και τις κλάσεις στα οποία βασίζεται η εφαρμογή. Το View είναι υπεύθυνο για την παρουσίαση των δεδομένων του Model και δέχεται την αλληλεπίδραση του χρήστη με την εφαρμογή. Ο Controller είναι ο συνδεδετικός κρίκος των δύο προηγούμενων. Δέχεται και αναλαμβάνει την αλληλεπίδραση του χρήστη μέσω του View και αλληλεπιδρά με το Model για τα δεδομένα τα οποία σύλλεξε με σκοπό να επικοινωνήσει και πάλι με το View για να εμφανίσει τα καινούργια.

Η αρχιτεκτονική αυτή έχει πλέον καθιερωθεί και ο λόγος είναι ο διαχωρισμός της επιχειρησιακής λογικής με την διεπαφή του χρήστη και την παρουσίαση των δεδομένων. Η χρήση της βοηθάει τους προγραμματιστές στην συντήρηση της εφαρμογής και στην ευκολία της μελλοντικής επεκτασιμότητάς της. Επιπλέον, αυτός ο διαχωρισμός που προσφέρει, βοηθάει στην αποσφαλμάτωση της εφαρμογής, εφόσον οι λειτουργίες επίσης διαχωρίζονται και ο εντοπισμός των λαθών είναι πιο εύκολος. Με τον διαχωρισμό δεν σταματά η παράλληλη υλοποίηση των μερών, αντιθέτως βοηθάει

τον προγραμματιστή να δημιουργεί λειτουργίες παράλληλα και να “βλέπει” αμέσως το αποτέλεσμα εξαλείφοντας έτσι και την πολυπλοκότητα της εφαρμογής. Το MVC συνδυάζεται επιτυχημένα και με τις Web εφαρμογές στις οποίες συνήθως συνεργάζονται πολλοί προγραμματιστές και σχεδιαστές.

Για την καλύτερη οργάνωση του project ακολουθήθηκε μια παραλλαγή της μεθοδολογίας του MVC. Μια εξωτερική πηγή, όπως ένας χρήστης, επικοινωνεί με το πρόγραμμα μέσω του **Controller**. Η επικοινωνία αυτή γίνεται με ένα request προς τον Controller, το οποίο είναι ένα HTTP request το οποίο αναλαμβάνει να διαχειριστεί ο Controller. Από εκεί ο Controller επικοινωνεί με το Model. Το Model έχει χωριστεί σε τρεις ενότητες – πακέτα, το Model, το Repository και το Service. Το **Model** περιέχει τις κλάσεις που είναι υπεύθυνες για την επιχειρησιακή λογική του προγράμματος. Το Model αφορά Entities, για αυτό το λόγο δημιουργούνται **DTO (Data Transferred Objects)** ώστε να είναι αυτά τα αντικείμενα τα οποία επικοινωνούν με τον χρήστη. Η χρήση των Entity και των DTO αναλύεται περισσότερο στο κεφάλαιο 4.2.1. Το **Repository** είναι υπεύθυνο για την απευθείας σύνδεση και επικοινωνία με τη βάση δεδομένων και είναι το σημείο στο οποίο υλοποιούνται όλων των ειδών των query προς τη βάση. Τέλος, το **Service**, είναι υπεύθυνο για την επικοινωνία του Controller με το Repository αλλά και του Repository με τον Controller. Στο Service γίνεται όλη η επεξεργασία των δεδομένων όπου είτε τα λαμβάνει από τον Controller με σκοπό να τα προετοιμάσει για το Repository όπου θα αποθηκευτούν, είτε θα δεχθεί τα δεδομένα της βάσης από το Repository και θα τα προετοιμάσει για να τα προωθήσει στον Controller υπό κάποια μορφή όπως HTML, JSON και XML (εικόνα 5).



Εικόνα 5: Κύκλος ζωής ενός request του χρήστη και η πορεία του εντός του API.

3.1.3 Spring – Spring Boot

Η Spring είναι ένα πολύ ισχυρό και “ελαφρύ” framework η οποία χρησιμοποιεί τη Java ως γλώσσα προγραμματισμού με δυνατότητα αλλαγής σε Kotlin και Groovy. Η πρώτη έκδοσή της υλοποιήθηκε από τον Rod Johnson τον Οκτώβριο του 2002 όπου τον επόμενο χρόνο δημοσιεύθηκε. Μεταγενέστερες εκδόσεις της βραβεύτηκαν με το Jolt productivity award και το JAX Innovation Award to 2006. Σημαντικές αλλαγές της Spring έγιναν το 2013 με την έκδοση 4.0 με την οποία προστέθηκε υποστήριξη για την Java Standard edition 8. Η Spring 4 θα υποστηρίζεται μέχρι το 2020 όπου αργότερα θα ανακοινωθεί η Spring 5.

Ένα από τα βασικά πλεονεκτήματα της Spring είναι ότι βασίζεται στο Aspect Oriented Programming (AOP) [21], το οποίο όπως λέει και η ονομασία του, χρησιμοποιεί aspects (πτυχές). Το AOP μπορεί να οριστεί ως η διάσπαση του κώδικα σε διαφορετικά modules γνωστό και ως modularisation, όπου το aspect είναι η βασική μονάδα του modularity. Τα aspect επιτρέπουν την υλοποίηση μεθόδων και λειτουργιών οι οποίες δεν είναι επικεντρωμένες στο business logic και χωρίς να χρειάζεται να επεξεργάζεται ο κώδικας του πυρήνα. Για παράδειγμα, η ασφάλεια εφαρμόζεται σε πολλές μεθόδους μέσα στο σύστημα, πολλές φορές ακόμη και σε κάθε request που δέχεται ένα API, οπότε αντί να επαναλαμβάνεται ο κώδικας σε κάθε μέθοδο, καθορίζεται η λειτουργικότητα σε μία κοινή κλάση και εφαρμόζεται σε ολόκληρη την εφαρμογή.

Υποστηρίζει το dependency-injection το οποίο είναι ένα πρότυπο για να υλοποιηθούν decoupled συστήματα. Με τον όρο decoupled (αποσυνδεδεμένα) συστήματα περιγράφονται τα συστήματα τα οποία μπορούν να επικοινωνήσουν μεταξύ τους χωρίς να είναι συνδεδεμένα. Επίσης, στα decoupled συστήματα μπορούν εύκολα να γίνουν αλλαγές χωρίς να επηρεάζονται τα υπόλοιπα συστήματα, με αποτέλεσμα το testing αυτών των συστημάτων να γίνεται πιο εύκολο για τον προγραμματιστή.

Αυτό που κάνει η Spring είναι ότι συνδέει τις κλάσεις χρησιμοποιώντας ένα αρχείο XML ή annotations. Με αυτόν τον τρόπο τα αντικείμενα δημιουργούνται και αρχικοποιούνται από την Spring και γίνονται injected στα “σωστά” μέρη, όπως Servlets, Web Frameworks, Business logic Classes, Repositories και Services [22].

Spring Boot

Η Spring Boot παρέχει μια διαφορετική οπτική στο οικοσύστημα της Spring. Δημοσιεύτηκε για πρώτη φορά στα μέσα του 2014 και έχει βελτιωθεί αλλά και συνεχίζει να βελτιώνεται αρκετά από τότε. Έπειτα από 6800 commits από 215 διαφορετικούς προγραμματιστές η έκδοση 2.0 δημοσιεύθηκε το 2018 με την οποία προστέθηκε υποστήριξη για τη Spring 5. Με την νέα έκδοση προστέθηκε και η υποστήριξη για την Java 9 ενώ η ελάχιστη έκδοση της Java είναι πλέον η 8.

Η Spring Boot παρέχει στους προγραμματιστές μια πλατφόρμα στη Java με την οποία μπορούν να υλοποιήσουν stand-alone και παραγωγικές εφαρμογές τις οποίες μπορούν να εκτελέσουν άμεσα με ελάχιστες παραμετροποιήσεις και χωρίς την παραμετροποίηση μιας ολόκληρης Spring εφαρμογής. Είναι ένα ανοιχτού κώδικα framework το οποίο προσφέρει ευκολία στην υλοποίηση Spring εφαρμογών, αυξάνει την παραγωγικότητα και ελαχιστοποιεί των χρόνο υλοποίησης λειτουργιών ενώ ταυτόχρονα είναι εύκολη στην κατανόηση και την εκμάθηση.

Micro Services

Η Spring Boot βοηθά στην δημιουργία micro-services [23] εύκολα και γρήγορα. Τα micro-services είναι μια αρχιτεκτονική που ακολουθείται όλο και περισσότερο σε μεγάλων προδιαγραφών προγράμματα. Με την αρχιτεκτονική αυτή μπορεί ο προγραμματιστής να αναπτύξει services ανεξάρτητα το ένα με το άλλο. Κάθε service έχει τις δικές του λειτουργίες και πετυχαίνει ένα “ελαφρύ” μοντέλο το οποίο μπορεί να υποστηρίξει μεγάλες και πολύπλοκες εφαρμογές. Τα προτερήματα που κάνουν τα micro Services σημαντικά είναι :

- Ευκολία στην ανάπτυξη και τη παραγωγή
- Εύκολη και δομημένη επεκτασιμότητα
- Ελάχιστη παραμετροποίηση
- Λιγότερος χρόνο υλοποίησης
- Επαναχρησιμοποιήσιμα

Η αρχιτεκτονική των micro-services είναι η ακριβώς αντίθετη από αυτή της μονολιθικής αρχιτεκτονικής όπου ο κώδικας είναι συγκεντρωμένος σε λίγα εκτελέσιμα αρχεία. Σε αυτή την αρχιτεκτονική είναι αρκετά δύσκολη η βελτιστοποίηση του κώδικα και ακόμη πιο δύσκολη η επεκτασιμότητα του. Επειδή όμως όλος ο κώδικας βρίσκεται σε ένα σημείο, η διαχείριση του γίνεται πιο εύκολη από αυτή μιας micro Services αρχιτεκτονικής.

Η Spring Boot έχει σχεδιαστεί για τους παρακάτω λόγους:

- Για την αποφυγή της πολύπλοκης παραμετροποίησης της Spring με XML
- Για την υλοποίηση μιας παραγωγικής Spring εφαρμογής ακόμη πιο εύκολα
- Για την ελαχιστοποίηση του χρόνου υλοποίησης της εφαρμογής
- Για την δυνατότητα να ξεκινήσει μία εφαρμογή άμεσα με μικρή παραμετροποίηση

Οι παραπάνω λόγοι έχουν κάνει την Spring Boot ένα από τα πιο γνωστά framework η οποία προσφέρει πολλές έτοιμες λειτουργίες και πλεονεκτήματα:

- Παρέχει εύκολους τρόπους παραμετροποίησης των Java Beans, των πολύπλοκων XML και ευκολία στην επικοινωνία και τις συναλλαγές με τη βάση δεδομένων
- Προσφέρει Spring εφαρμογές με την χρήση annotation
- Διευκολύνει την διαχείριση των dependencies
- Παρέχει πλήρη διαχείριση των REST endpoints

Πως λειτουργεί η Spring Boot

Το entry-point μιας Spring Boot εφαρμογής είναι η κλάση στην οποία έχει ανατεθεί το annotation `@SpringBootApplication` η οποία είναι σχεδόν πάντα η κλάση στην οποία βρίσκεται η main μέθοδος (εικόνα 6).

```
@SpringBootApplication(scanBasePackageClasses = MyapiApplication.class)
@ComponentScan("com.pti.myapi")
@EnableJpaRepositories(basePackageClasses = {MyapiApplication.class})
```

Εικόνα 6: Annotations της Main μεθόδου για την ενεργοποίηση και τη χρήση του Spring Boot Framework

Η Spring Boot παραμετροποιεί αυτόματα την εφαρμογή με βάση τα dependencies, τα οποία έχουν προστεθεί ομαδοποιημένα σε ένα XML αρχείο με όνομα `pom.xml` (εικόνα 6), και χρησιμοποιώντας το annotation `@EnableAutoConfiguration`. Με το annotation `@ComponentScan` η Spring Boot ελέγχει όλα τα Component τα οποία περιέχονται στην εφαρμογή αυτόματα (εικόνα 7).

```
<dependencies>

  <!-- Spring Boot -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-cache</artifactId>
  </dependency>

</dependencies>
```

Εικόνα 7: Spring Boot dependencies εσωτερικά του αρχείου `pom.xml`

Ένα αρκετά χρήσιμο annotation είναι το `@EnableJpaRepositories` με το οποίο, κατά κύριο λόγο, γίνεται πολύ πιο εύκολη η επικοινωνία με την βάση δεδομένων. Με τη προσθήκη του Jpa Repository δημιουργούνται αυτόματα κάποιες βασικές CRUD (Create – Read – Update – Delete) εντολές και ορισμένες ακόμη απλές μέθοδοι για ερωτήματα στη βάση. Στο σημείο που προστέθηκε το Jpa Repository υπάρχει δυνατότητα δημιουργίας ερωτημάτων προς τη βάση με διάφορους τρόπους. Ένας από του πιο σύνθητες είναι με τη χρήση του JPQL query με τη προσθήκη του `@Query` annotation στην υλοποιημένη μέθοδο. Η JPQL (Java Persistence Query Language) είναι μια αντικειμενοστραφής γλώσσα ερωτημάτων και συντάσσεται όπως η SQL με τη διαφορά ότι οι αναφορές στους πίνακες και στα πεδία τους γίνεται με τα αντίστοιχα Entity της εφαρμογής. Για παράδειγμα ένα απλό SELECT

query θα συντασσόταν : “*SELECT u FROM User u WHERE u.username = username*” και η μέθοδος θα δεχόταν σαν παράμετρος μια μεταβλητή username. Ένα ακόμη παράδειγμα με υλοποιημένη μέθοδο της εφαρμογής φαίνεται στην εικόνα 8.

```
@Modifying(clearAutomatically = true)
@Query("UPDATE Group g SET g.title = ?1, g.description = ?2 WHERE g.id = ?3")
void updateGroupProperties(String title, String description, Integer id);
```

Εικόνα 8: Χρήση του @Query σε μέθοδο για τη συγγραφή ερωτήματος προς τη βάση

Ένας ακόμη τρόπος δημιουργίας ερωτημάτων στη βάση είναι με τη χρήση των Automatic Custom Queries. Με τη προσθήκη του Jpa Repository η Spring αναλύει όλες τις μεθόδους που έχουν δημιουργηθεί στη συγκεκριμένη κλάση και προσπαθεί να δημιουργήσει αυτόματα queries από τα ονόματα των μεθόδων. Το προηγούμενο παράδειγμα με την υλοποίηση ενός απλού SELECT μπορεί να υλοποιηθεί εύκολα με τα Automatic Custom Queries. Η μέθοδος *findByUsername(String username)* εκτελεί ακριβώς το ίδιο όνομα χωρίς να χρειαστεί περαιτέρω υλοποίηση (εικόνα 9). Αυτός ο τρόπος δημιουργίας ερωτημάτων είναι αρκετά κομψός και πάρα πολύ απλός, αλλά πάντα με κάποιους περιορισμούς στα ερωτήματα [24, 25].

```
List<User> findByUsername(String username);

List<User> findByFirstNameAndLastNameOrderByFirstName(String firstName, String lastName);
```

Εικόνα 9: Παράδειγμα Automatic Custom Queries του Jpa Repository

3.1.4 Rest – Rest API

Το Representational State Transfer, γνωστό ως REST, είναι μια ακόμη αρχιτεκτονική σχεδίασης λογισμικών και ειδικότερα για κατανεμημένα συστήματα. Η αρχιτεκτονική παρουσιάστηκε για πρώτη φορά από τον Roy Fielding στο διδακτορικό του με τίτλο “Architectural Styles and the Design of Network-based Software Architectures” το 2000. Το REST ορίζει κάποιους περιορισμούς για την δημιουργία Web Services, οι οποίες λέγονται RESTful Web Services με σκοπό να προσφέρουν διαλειτουργικότητα μεταξύ των υπολογιστικών συστημάτων στο διαδίκτυο [26].

Για να αποκαλείται ένα Web Service RESTful πρέπει να ικανοποιεί 6 περιορισμούς.

Client – Server

Ένας από τους πιο βασικούς περιορισμούς της αρχιτεκτονικής REST είναι ο διαχωρισμός του Client με τον Server. Με τον διαχωρισμό των λειτουργιών του χρήστη μέσω της διεπαφής του με τα δεδομένα που επεξεργάζεται και περιέχει ο Server βελτιώνεται η φορητότητα της διεπαφής του χρήστη μεταξύ πολλαπλών πλατφορμών ενώ ταυτόχρονα βελτιώνεται η επεκτασιμότητα του server εφόσον απλοποιούνται οι εξαρτήσεις οι λειτουργίες του.

Stateless

Κάθε request του Client προς τον Server πρέπει να περιέχει όλη την απαραίτητη πληροφορία που θα χρειαστεί ο Server για να κατανοήσει την ανάγκη που προκλήθηκε. Ένας ακόμη βασικός περιορισμός είναι πως δεν πρέπει το request να δέχεται δεδομένα από το session και το local storage του client. Αυτό γίνεται για να παρέχει το REST ασφάλεια της εφαρμογής και των δεδομένων, καθώς τα session και local storages είναι προσβάσιμα από τον χρήστη και έχει τη δυνατότητα να τα τροποποιήσει. Με το REST οι αλλαγές στα δεδομένα αυτά δεν θα επηρεάσουν καμία λειτουργία ή δεδομένο της εφαρμογής.

Cacheable

Με τον όρο αυτό, η REST περιορίζει τα response των request να διευκρινίζουν για το εάν είναι cacheable ή όχι. Cacheable θεωρούνται τα δεδομένα του response τα οποία είναι real-data, δηλαδή αληθή. Αν ένα response είναι cacheable τότε η cache του Client μπορεί να τα ξαναχρησιμοποιήσει για μελλοντικά requests, ενώ αν είναι non-cacheable θα πρέπει να βρεθεί άλλος τρόπος για να δεχθεί τα real-data με πιο πιθανή λύση την δημιουργία ενός νέου request.

Uniform Interface

Ο περιορισμός του Uniform Interface έχει δημιουργηθεί για να απλοποιήσει τις υλοποιήσεις και να βεβαιώσει πως τα services μπορούν να διαχειριστούν ανεξάρτητα το ένα με το άλλο. Ο περιορισμός αυτός χωρίζεται σε τέσσερις υποενότητες.

- **Identification Of Resources** : ορίζει πως τα request θα πρέπει να αναγνωρίζουν τους πόρους τους οποίους θα πρέπει να βρουν μέσω του URL. Αυτό όμως δεν αναγκάζει τα δεδομένα να έχουν κάποια σχέση με τον τρόπο ή τη μορφή που επιστρέφουν και εξαρτώνται από τον τρόπο που έχουν υλοποιηθεί οι μέθοδοι.
- **Resource Manipulation through Representation** : ορίζει πως όταν δίνεται κάποιος πόρος ή κάποια δεδομένα στον χρήστη, θα πρέπει να οι πληροφορίες που επιστρέφονται μαζί με το response να επαρκούν ώστε με αυτά τα δεδομένα να μπορεί να επεξεργαστεί ή να διαγράψει τον πόρο ή τα δεδομένα αυτά. Με τον τρόπο αυτό ελαχιστοποιούνται οι κλήσεις εφόσον δεν χρειάζεται να ξαναγίνεται κάποιο request στο API για να πάρει τις πληροφορίες που χρειάζεται για την επεξεργασία ή τη διαγραφή, οι οποίες από μόνες τους αποτελούν ένα ξεχωριστό request στο API.

- **Self Descriptive Messages** : τα μηνύματα προς τον server θα πρέπει να έχουν αρκετή πληροφορία ώστε να γνωρίζει τον τρόπο που θα τα επεξεργαστεί.
- **Hypermedia as the Engine of Application State (HATEOAS)** : Η πρόσβαση στο API θα πρέπει να είναι παρόμοια με την πρόσβαση σε μια web σελίδα, δηλαδή θα πρέπει να κάποιος να είναι σε θέση να ανακαλύψει περιοχές του API τόσο εύκολα όσο θα ανακάλυπτε περιοχές σε μια web σελίδα. Πάνω σε αυτό, ένα response του API θα μπορούσε να περιέχει links και να δείχνει σε περιοχές του API οι οποίες είναι διαθέσιμες.

Layered System

Αυτή η λογική υλοποίησης επιτρέπει μια αρχιτεκτονική να είναι σε θέση να συνθέσει μια ιεραρχική δομή μεταξύ των επιπέδων που υλοποιούν την εφαρμογή. Με αυτόν τον τρόπο ένα στοιχείο δεν θα μπορεί να έχει πρόσβαση πέρα από το δικό του επίπεδο και το επίπεδο που συνδέεται άμεσα. Η λογική αυτή εκτός από την δομημένη συγγραφή και την ευκολία στον διαχωρισμό των λειτουργιών του κάθε επιπέδου που προσφέρει, βοηθάει αρκετά τους προγραμματιστές στην αποσφαλμάτωση του κώδικα καθώς θα μπορεί να αναγνωρίσει πολύ πιο εύκολα από που προήλθε το σφάλμα και να το διορθώσει χωρίς να έχουν επίπτωση τα υπόλοιπα επίπεδα.

Code on demand

Αυτός ο περιορισμός δεν είναι υποχρεωτικός. Το REST επιτρέπει στον client να έχει τη περισσότερες δυνατότητες με το να μπορεί να “κατεβάσει” και να εκτελέσει κώδικα σε μορφή applet ή script. Αυτό βοηθάει τους clients καθώς ελαχιστοποιεί τον αριθμό των χαρακτηριστικών που θα πρέπει να εφαρμοστεί από πριν. Ο περιορισμός αυτός χρησιμοποιείται αρκετά σπάνια για αυτό και πλέον δεν θεωρείται υποχρεωτικός.

Ο βασικός όρος των πληροφοριών στη REST αρχιτεκτονική είναι ο πόρος (resource). Κάθε πληροφορία η οποία μπορεί να ονομαστεί μπορεί να είναι ένα resource και το REST χρησιμοποιεί ένα αναγνωριστικό πόρου (resource identifier) με σκοπό να αναγνωρίζει τον κάθε πόρο που αλληλεπιδράται μεταξύ των στοιχείων. Η κατάσταση ενός πόρου σε κάποια συγκεκριμένη στιγμή ονομάζεται αναπαράσταση πόρου (resource representation) και αποτελείται από τα δεδομένα, metadata με πληροφορίες για τα δεδομένα και hypermedia link, όπως εξηγήθηκαν στο Uniform Interface και στην ενότητα Hypermedia as the Engine of Application State τα οποία θα βοηθούν τον client να μεταφερθεί στο επόμενο επιθυμητό state, ανάλογα τον αρχικό πόρο.

3.1.5 HtmlUnit

“Το HtmlUnit είναι ένας browser χωρίς διεπαφή για προγράμματα σε Java” [27]. Αποτελεί μία βιβλιοθήκη η οποία λειτουργεί σαν browser με την οποία μπορεί να εκτελεστεί κάθε λειτουργία σε μία σελίδα, όπως θα εκτελούταν σε έναν κανονικό browser. Έχει την δυνατότητα να προσομοιώσει

browsers όπως τον Chrome, τον Firefox και τον Internet Explorer ενώ παρέχει μια πολύ καλή μηχανή μετάφρασης JavaScript και είναι ικανό να λειτουργήσει ακόμη και με πολύπλοκες AJAX βιβλιοθήκες. Αν και η τυπική χρήση της συγκεκριμένης βιβλιοθήκης είναι για testing σε web εφαρμογές και site, η απλότητα, η ταχύτητα και η δύναμη της την έχουν κάνει μία από τις καλύτερες βιβλιοθήκες για συλλογή πληροφοριών από website.

Η HtmlUnit παρέχει ένα μεγάλο πλήθος χαρακτηριστικών τα οποία ολοκληρώνουν τις δυνατότητες ενός browser. Τα σημαντικότερα από αυτά είναι :

- Υποστήριξη για HTTP και HTTPS πρωτόκολλα
- Υποστήριξη των cookies
- Δυνατότητα προσδιορισμού των αποτυχημένων responses του server
- Υποστήριξη των βασικών submit μεθόδων POST, GET αλλά και πολλών ακόμη
- Εύκολη πρόσβαση στις HTML σελίδες και στις πληροφορίες που περιέχουν
- Υποστήριξη των proxy server
- Υποστήριξη για βασική και NTLM αυθεντικοποίηση
- Άριστη υποστήριξη της JavaScript

Η άριστη υποστήριξη της JavaScript πετυχαίνεται με την προσομοίωση των παραμετροποιημένων browser, Chrome, Firefox, Internet Explorer ενώ ταυτόχρονα χρησιμοποιεί την Rhino JavaScript μηχανή ως βασική γλώσσα. Ο κώδικας σε JavaScript εκτελείται όπως σε έναν κανονικό browser όταν φορτώνει η σελίδα ή όταν ενεργοποιείται ένα event. Επιπροσθέτως, η HtmlUnit παρέχει την δυνατότητα εκτέλεσης JavaScript κώδικα σε υπάρχουσα web σελίδα.

Η HtmlUnit είναι ένας headless browser. Είναι δηλαδή ένας web browser ο οποίος βοηθάει στην γρήγορη αυτοματοποίηση διαφόρων ενεργειών μιας ιστοσελίδας και στη γνώση για την επίδοση της σελίδας κάτω από συνθήκες. Οι headless browsers δεν έχουν GUI και λειτουργούν με command-line εντολές. Χρησιμοποιούνται κυρίως για testing σε web σελίδες επειδή “καταλαβαίνουν” τις HTML σελίδες όπως κάθε άλλος browser. Χρησιμοποιούνται επίσης για testing σε βιβλιοθήκες της JavaScript, αλληλεπιδράσεις της JavaScript και για αυτοματοποιημένα UI test. Συγκεκριμένα, ένα headless browser περιβάλλον παρέχει στον προγραμματιστή τη δυνατότητα να υλοποιήσει και να εκτελέσει διάφορα script με σκοπό να:

- δοκιμάσει clicks σε link και buttons
- αυτοματοποιήσει συμπληρώσεις φορμών και να τις υποβάλει
- αποκτήσει αναφορές για τους χρόνους απόκρισης της σελίδας
- δοκιμάσει την επίδοση του SSL
- αποκτήσει κώδικα της σελίδας χωρίς καθόλου κόπο

Οι headless browsers είναι πολύ πιο γρήγοροι από τους κανονικούς browsers και αυτό συμβαίνει διότι οι κανονικοί browsers χρειάζονται χρόνο για να ανοίξουν, να φορτώσουν την HTML,

τη CSS, τη JavaScript και τις εικόνες, ενώ οι headless ξεκινούν κατευθείαν να εκτελούν μεθόδους χωρίς να περιμένουν τη σελίδα να φορτώσει πλήρως. Από την άλλη η ανίχνευση των σφαλμάτων γίνεται πολύ πιο εύκολα στους κανονικούς browsers. Εκεί, καθώς οι μέθοδοι εκτελούνται με τις ενέργειες του χρήστη, μπορεί εύκολα να αλληλεπιδράσει μαζί τους και να κατανοήσει πιο εύκολα και γρήγορα το λάθος.

3.1.6 Βάση Δεδομένων

Για την βάση δεδομένων χρησιμοποιήθηκε η MySQL οι οποία είναι ένα open-source RDBMS (Relational Database Management System) η οποία χρησιμοποιεί SQL για τα ερωτήματα στην βάση. Η MySQL μετρά περισσότερες από 11 εκατομμύρια εγκαταστάσεις κάνοντάς τη το πιο δημοφιλές σύστημα διαχείρισης σχεσιακών βάσεων το οποίο ανήκει στην Oracle και έχει συνταχθεί σε C και C++.

Μια σχεσιακή βάση δεδομένων [28] οργανώνει τα δεδομένα σε πίνακες οι οποίοι μπορούν να συνδεθούν με κάποια κοινά δεδομένα μεταξύ τους. Αυτή η τεχνική δίνει την δυνατότητα να συλλεχθούν δεδομένα από πολλαπλούς πίνακες με μόνο ένα query. Για παράδειγμα, εάν ένας πίνακας *User* τις στήλες *id*, *first_name*, *last_name*, και ένας δεύτερος *Group* τις στήλες *id*, *user_id*, *group_name* τότε οι δύο πίνακες μπορούν να συνδεθούν με το κοινό τους πεδίο *id* για τον πίνακα *User* και *user_id* για τον πίνακα *Group*. Έτσι, με ένα απλό ερώτημα μπορούμε να πάρουμε πληροφορίες για το όνομα του *User* και το όνομα του/των *Group* που έχει. Το βασικό πλεονέκτημα μιας σχεσιακής βάσης είναι η δυνατότητα απόκτησης πληροφορίας με σημασία με μια απλή σύνδεση πινάκων. Η σύνδεση πινάκων βοηθά στην καλύτερη κατανόηση των δεδομένων και στη καλύτερη διαχείρισή τους. Για να μπορέσει κάποιος να επικοινωνήσει με μια σχεσιακή βάση δεδομένων θα χρειαστεί να χρησιμοποιήσει μια Structured Query Language (SQL) η οποία είναι η βασική γλώσσα για αλληλεπίδραση με RDBMS.

Επιπλέον, τα πλεονεκτήματα των σχεσιακών βάσεων δεδομένων σε σχέση με άλλες αρχιτεκτονικές είναι η ευελιξία, οι περιττές εγγραφές και η ευκολία ανάκτησης ενός backup μετά από καταστροφή του συστήματος. Η ευελιξία της προέρχεται από την ενσωματωμένη γλώσσα που παρέχει η SQL, την DDL η οποία επιτρέπει την δημιουργία πινάκων, στηλών και πολλών ακόμα αλλαγών όσο η βάση χρησιμοποιείται. Σε μια σχεσιακή βάση η πληροφορία χρειάζεται να υπάρχει μόνο μία φορά αποθηκευμένη. Αυτό ωθεί τους σχεδιαστές στην κανονικοποίηση των σχημάτων, άρα και στην καλύτερη κατανόηση των δεδομένων αλλά και στην ελαχιστοποίηση των περιττών εγγραφών στη βάση.

3.2 Front-end

Η web σελίδα δημιουργήθηκε με χρήση των τριών βασικών τεχνολογιών HTML, CSS και JavaScript. Για την διευκόλυνση στην υλοποίηση χρησιμοποιήθηκε επίσης η jQuery η οποία βοήθησε στην ταχύτερη ανάπτυξη της σελίδας με τις απλοποιημένες λειτουργίες της JavaScript καθώς και με

περισσότερες λειτουργίες οι οποίες ήταν αρκετά χρήσιμες. Για την εμφάνιση χρησιμοποιήθηκε το template SB Admin 2 το οποίο προσφέρει ένα όμορφο responsive περιβάλλον με αρκετά πλαίσια χρησιμοποιώντας Bootstrap. Για τα γραφήματα χρησιμοποιήθηκε η βιβλιοθήκη chart.js.

3.2.1 Bootstrap

Η Bootstrap [29] είναι ένα open-source framework σχεδιασμένο για την διευκόλυνση στην συγγραφή CSS αρχείων. Το αρχικό όνομα του framework ήταν Twitter Blueprint το οποίο είχε δημιουργηθεί από δύο προγραμματιστές του Twitter, τον Mark Otto και τον Jacob Thornton. Όπως αναφέρουν οι ίδιοι, η αρχική ιδέα ήταν να σχεδιαστεί ένα εσωτερικό εργαλείο για την διευκόλυνση τους και είδαν την ευκαιρία για κάτι μεγαλύτερο. Η μετονομασία σε Bootstrap έγινε τον Αύγουστο του 2011 όπου και εκδόθηκε. Η σημαντικότερη έκδοση ήταν η Bootstrap 4 η οποία δημοσιεύθηκε τον Οκτώβριο του 2014 με την οποία έγινε ριζική αλλαγή του κώδικα καθώς προστέθηκαν και πάρα πολλές επιπλέον λειτουργίες, αλλά και υποστήριξη των τελευταίων εκδόσεων του Google Chrome, Firefox, Opera, Safari και Internet Explorer. Το framework συνεχίζουν να το συντηρούν και να το αναπτύσσουν οι ίδιοι προγραμματιστές αλλά πλέον και ένα τεράστιο πλήθος προγραμματιστών που συνεισφέρουν, εφόσον το framework είναι open-source.

Είναι ένα πάρα πολύ δυνατό εργαλείο το οποίο περιέχει μια συλλογή από HTML , CSS και JavaScript εργαλεία για την υλοποίηση web σελίδων και web εφαρμογών. Έγινε γρήγορα διάσημο και αυτό οφείλεται στους προγραμματιστές και τους σχεδιαστές που το χρησιμοποιούσαν διότι “έλυne” τα χέρια τους όσον αφορά τη σχεδίαση και την εμφάνιση των web εφαρμογών. Είναι ένα πολύ εύκολο εργαλείο το οποίο προσφέρει πάρα πολλές λειτουργίες. Η κυριότερη χρήση του γίνεται διότι προσφέρει ένα responsive design στις εφαρμογές, κάτι που θεωρείται πλέον απαραίτητο λόγω της τεράστιας αύξησης στη χρήση των κινητών τηλεφώνων. Οι σχεδιαστές πλέον δεν θα χρειάζεται να δημιουργούν πολλές διαφορετικές σελίδες ανάλογα την συσκευή που τις χρησιμοποιεί (σταθερός υπολογιστής, tablet, κινητό τηλέφωνο) διότι η Bootstrap αναλαμβάνει να προσφέρει με εργαλεία λύσεις για την καλύτερη εμπειρία του χρήστη. Εκτός από τις cross-platform λειτουργίες, η Bootstrap βοηθάει στην σταθερή και οργανωμένη σχεδίαση με τη χρήση επαναχρησιμοποιήσιμων component. Μια ακόμη πολύ σημαντική λειτουργία που προσφέρει είναι το wide browser compatibility (cross-browser). Μέχρι τότε οι προγραμματιστές και οι σχεδιαστές ήταν αναγκασμένοι να σχεδιάζουν διαφορετικά ορισμένες υλοποιήσεις για κάθε browser, κάτι που ήταν αρκετά χρονοβόρο και στην υλοποίηση αλλά και στην αποσφαλμάτωση του κώδικα. Με τη Bootstrap αυτές οι υλοποιήσεις δεν χρειάζονται, διότι είναι σχεδιασμένη με τρόπο ώστε να μπορεί να λειτουργήσει σε κάθε browser. Με τη χρήση αυτού του εύκολου στη χρήση και στην εκμάθηση εργαλείου, οι web προγραμματιστές μπορούν να συγκεντρωθούν στην υλοποίηση των λειτουργιών της εφαρμογής τους χωρίς να πρέπει να ασχοληθούν με περαιτέρω με τη σχεδίαση και την όμορφη εμφάνιση της.

3.2.2 JQuery

Η jQuery [30] είναι μια μικρή και γρήγορη βιβλιοθήκη της JavaScript με πάρα πολλές λειτουργίες που βοηθούν στην γρήγορη και εύκολη συγγραφή σε JavaScript. Η πρώτη έκδοση δημοσιεύθηκε τον Αύγουστο του 2006 και από τότε αλλάζει έκδοση σχεδόν κάθε χρόνο. Ο αρχικός συγγραφέας της ήταν ο John Resig όπου αργότερα δημιουργήθηκε η The jQuery Team και είναι αυτή που συνεχίζει με τις εκδόσεις και τις βελτιώσεις. Η jQuery είναι μια πολύ μικρή βιβλιοθήκη, ενδεικτικά, η πρώτη έκδοση είχε μέγεθος περίπου 50 KB ενώ η τελευταία (3.5.1, Μάιος 2020) έχει μέγεθος περίπου 87 KB.

Η ευκολία στη συγγραφή και τη γρήγορη υλοποίηση λειτουργιών χωρίς τις αμέτρητες γραμμές κώδικα που θα χρειαζόντουσαν με τις λειτουργίες της JavaScript, έχουν κάνει τη jQuery ίσως το πιο διάσημο JavaScript framework. Οι σημαντικότερες απλοποιημένες λειτουργίες της JavaScript που παρέχει είναι:

- ευκολότερη διάσχιση του DOM
- καλύτερο χειρισμό HTML αρχείων
- καλύτερη και πιο εύκολη διαχείριση των event
- animation
- συμβατός κώδικας για cross-browser
- ευκολότερο και απλοποιημένο Ajax

Η jQuery κατάφερε να ελαχιστοποιήσει την δυσκολία, ειδικότερα των παλαιότερων browsers, να περιηγηθούν στο DOM μέσω της JavaScript [31]. Μέχρι τότε κάτι τέτοιο απαιτούσε πολύπλοκες λειτουργίες καθώς δεν υπήρχε κάποιος καθορισμένος τρόπος υλοποίησης του. Ίσως η αγαπημένη λειτουργία των προγραμματιστών είναι ο τρόπος που χειρίζεται η jQuery τα DOM στοιχεία με τους selectors. Με τους selectors γίνεται η αναφορά σε ένα στοιχείο του DOM και συμπεριλαμβάνοντας μια μέθοδο δημιουργεί τις επιθυμητές αλλαγές. Μία ακόμη αγαπημένη λειτουργία είναι η Ajax μέθοδος της jQuery για την εύκολη διαχείριση των HTTP requests σε σχέση με την κουραστική XMLHttpRequest της JavaScript. Η jQuery καταφέρνει να δώσει λύση στο επίπεδο της συμβατότητας μεταξύ browser. Οι παλαιότεροι browser διαχειρίζονται τον κώδικα με διαφορετικές μεθόδους καθώς δεν υπήρχε κάποιος καθορισμένος τρόπος με αποτέλεσμα οι προγραμματιστές να πρέπει να υλοποιούν διαφορετικό κώδικα για τον κάθε browser ξεχωριστά, ακόμη και για κάθε version τους. Η jQuery assφαλίζει πως οι σελίδες μεταφράζονται σωστά ανεξάρτητα τον browser του κάθε χρήστη.

Η jQuery χρησιμοποιείται σε ποσοστό μεγαλύτερο του 80% στις 1 εκατομμύριο κορυφαίες σελίδες στο διαδίκτυο. Η διασημότητα της οφείλεται, εκτός των λειτουργιών που προσφέρει, στη χρήση της στα πιο δημοφιλή CMS (Content Management Systems) όπως είναι το WordPress, το Joomla και το Drupal. Αυτή η διασημότητα έχει οδηγήσει σε ένα τεράστιο documentation για την συγκεκριμένη βιβλιοθήκη και σε ένα ακόμα πιο μεγάλο οικοσύστημα της. Για αυτό τον λόγο είναι

σύνηθες να μαθαίνεται αρκετά γρήγορα από τους προγραμματιστές, χωρίς δυσκολία και με γρήγορα αποτελέσματα στις υλοποιήσεις τους.

Την τελευταία δεκαετία οι browsers έχουν ξεκινήσει και ακολουθούν όλο και πιο συγκεκριμένα πρότυπα και συγκεκριμένα για την χρήση του browser στα κινητά τηλέφωνα. Αυτό πηγαίνει κόντρα στο πρότυπο της jQuery της οποίας ο σκοπός είναι η εξάλειψη των διαφορών μεταξύ των browsers και η κανονικοποίηση των λειτουργιών τους. Ένα ακόμη μειονέκτημα της είναι πως δεν είναι το καταλληλότερο framework για κινητά, καθώς χρησιμοποιεί αρκετούς πόρους για να “περιηγηθεί” στο DOM όπως και όταν τοποθετεί events σε στατικές σελίδες. Προσθέτοντας στα παραπάνω και τα νέα framework που είναι εστιασμένα στην γρήγορη, εύκολη και βασισμένη σε πρότυπα σχεδίαση front-end εφαρμογών, η jQuery αρχίζει την πτώση της. Παραμένει και θα παραμένει όμως για πολλά χρόνια ακόμη ένα από τα πιο διάσημα και framework της JavaScript καθώς τα περισσότερα συστήματα έχουν υλοποιηθεί σύμφωνα με αυτή και χρειάζεται περισσότερος κόπος στο να γίνει αλλαγή σε ένα νέο, μοντέρνο και κατά πάσα πιθανότητα καλύτερο framework εφόσον η jQuery είναι αρκετά απλή και εύκολη στη χρήση αλλά και με εύκολη δυνατότητα κλιμάκωσης.

Κεφάλαιο 4ο: Σχεδίαση και Υλοποίηση της Εφαρμογής

4.1 Σχεδίαση και Υλοποίηση του back-end

Όπως αναφέρθηκε στο κεφάλαιο 3.1.2, στο project χρησιμοποιήθηκε μια παραλλαγή της MVC μεθοδολογίας. Πιο αναλυτικά, το project αποτελείται από 5 βασικά πακέτα (packages). Αυτά είναι τα : model, dto, repository, service και controller.

4.1.1 Model

Στο model εμπεριέχονται οι βασικές κλάσεις που δηλώνουν τα πεδία της βάσης. Αυτές είναι :

- **User:** id, username, password, email, groups
- **Group:** id, user, title, description, members
- **Member:** id, group, authorCode

Αυτές οι κλάσεις λειτουργούν σαν οντότητες. Είναι αυτές που αντιπροσωπεύουν τους πίνακες και τις στήλες στην βάση δεδομένων και για αυτό έχουν τα κατάλληλα annotations που τα προσδιορίζουν. Στην δήλωση των κλάσεων, δηλαδή στο σημείο *public class ClassName { ...*, όπου *ClassName* το όνομα ενός των τριών(3) κλάσεων παραπάνω, χρειάζεται το annotation *@Entity* το οποίο καθορίζει ότι η κλάση είναι ένα entity (οντότητα). Επιπλέον, χρειάζεται και το annotation *@Table(name="table_name")* , όπου *table_name* το όνομα του πίνακα στην βάση δεδομένων, για να δηλωθεί το όνομα του πίνακα και να γίνει σύνδεση (map) με τον πίνακα στην βάση (εικόνα 10).

```
@Column(name="username")
private String username;

@Column(name="password")
private String password;
```

Εικόνα 10: Χρήση του *@Column* σε πεδίο μιας κλάσης *Entity* και σύνδεση με το ανρίστοιχο πεδίο του πίνακα στη βάση δεδομένων

```
@Entity
@Table(name="users")
public class User {
```

Εικόνα 11: Annotations για την δήλωση μιας κλάσης ως *Entity*

Μια απλή αναφορά σε ένα πεδίο στη βάση, όπως είναι τα πεδία *username*, *password*, *email* για την κλάση *User*, αρκεί το annotation *@Column(name="column_name")*, όπου *column_name* το όνομα της στήλης που προσδιορίζει στην βάση, για να γίνει η σύνδεση των πεδίων του *Entity* με τα πεδία του πίνακα (εικόνα 11).

Στα πιο σύνθετα πεδία, δηλαδή στα πεδία που δηλώνουν την σύνδεση με πεδία άλλης κλάσης, πρέπει να δηλωθεί η σχέση μεταξύ τους. Αυτές οι σχέσεις είναι:

- **OneToMany**: για σχέσεις ένα προς πολλά [1:N]
- **ManyToOne**: για σχέσεις πολλά προς ένα [N:1]
- **OneToOne**: για σχέσεις ένα προς ένα [1:1]
- **ManyToMany**: για σχέσεις πολλά προς πολλά [N:N]

Μία ManyToMany σχέση μεταξύ δύο(2) πινάκων, μπορεί να υλοποιηθεί με έναν ενδιάμεσο πίνακα. Οι δύο αυτοί πίνακες θα πρέπει, και οι δύο, να έχουν OneToMany σχέση με τον ενδιάμεσο πίνακα. Αυτές οι σχέσεις βασίζονται στις σχέσεις που έχουν δηλωθεί στη βάση. Για παράδειγμα εάν η σχέση είναι πολλά προς ένα [N:1] (ManyToOne), θα πρέπει στην βάση να έχει δηλωθεί σωστά το ξένο κλειδί (foreign key) του σχετικού πεδίου του πρώτου πίνακα, στο σχετικό πεδίο του δεύτερου.

Στο συγκεκριμένο project δεν υπάρχουν ManyToMany σχέσεις. Οι σχέσεις μεταξύ των κλάσεων είναι όλες πολλά προς ένα (ManyToOne) και ένα προς πολλά (OneToMany). Η πρώτη σχέση είναι η σχέση της κλάσης User με την κλάση Group, όπου ένας user μπορεί να έχει πολλά group. Έτσι, εκτός από την δήλωση του id του πίνακα users ως foreign key για το πεδίο user_id του πίνακα group στη βάση, πρέπει να δηλωθεί και στις κλάσεις model το κατάλληλο annotation. Εφόσον η σχέση User-Group είναι ένα προς πολλά [1:N], χρησιμοποιείται το `@OneToMany(mappedBy="user")` στο πεδίο groups της κλάσης User και το `@ManyToOne` στο πεδίο user της κλάσης Group. Η παράμετρος mappedBy που δέχεται το `@OneToMany` δηλώνει το πεδίο του Entity της δεύτερης κλάσης στο οποίο πρέπει να γίνει η σύνδεση (map). Επίσης, εφόσον η σχέση είναι ένα προς πολλά, το πεδίο που χρησιμοποιεί το `@OneToMany` πρέπει να είναι μια λίστα τύπου Group, διότι πλέον μια κλάση τύπου User έχει πολλές κλάσεις τύπου Group (εικόνα 12). Τα πεδία που χρησιμοποιούν το annotation `@ManyToOne` είναι πεδία κλάσεων. Για παράδειγμα, στην κλάση Group, `@ManyToOne` δέχεται το πεδίο user το οποίο είναι τύπου User και με το οποίο, όταν συνδεθεί κατάλληλα, μπορούμε να έχουμε γνώση των πεδίων του user. Τέτοιου είδους πεδία δέχονται σαν annotation το `@JoinColumn(name="table_row_id")`, όπου table_row_id το id της γραμμής του πίνακα στον οποίο αναφέρεται. Αυτό γίνεται διότι το πεδίο της κλάσης είναι τύπου Object και δεν είναι δυνατή η απευθείας αντιστοίχιση με πεδίο της βάσης όπως γίνεται με το `@Column(name="column_name")` (εικόνα 13).

```

@OneToMany(
    mappedBy="user",
    cascade={CascadeType.MERGE,
             CascadeType.REMOVE},
    fetch = FetchType.LAZY
)
private List<Group> groups;

```

Εικόνα 12: Χρήση του @OneToMany για τη δημιουργία σχέσης 1:N με πίνακα της βάσης δεδομένων

```

@ManyToOne(cascade= {
    CascadeType.DETACH,
    CascadeType.MERGE,
    CascadeType.REFRESH})
@JoinColumn(name="user_id")
@JsonIgnore
private User user;

```

Εικόνα 13: Χρήση του @ManyToOne για τη δημιουργία σχέσης N:1 με πίνακα της βάσης δεδομένων

Και στα δύο annotation (@ManyToOne και @OneToMany) υπάρχει η παράμετρος cascade [32]. Αυτή μπορεί να πάρει τις τιμές:

- CascadeType.MERGE
- CascadeType.PERSIST
- CascadeType.REFRESH
- CascadeType.REMOVE
- CascadeType.DETACH
- CascadeType.ALL

Από τις παραπάνω τιμές, στο πρόγραμμα εφαρμόζονται οι CascadeType.MERGE, ώστε όταν συγχωνευθεί ένα κύριο Entity να συγχωνευθούν και τα σχετικά Entities, και CascadeType.REMOVE, ώστε όταν διαγράφεται ένα κύριο Entity, να διαγράφονται και οι εγγραφές των πινάκων που το αφορούν. Για παράδειγμα, όταν διαγράφεται μια εγγραφή στο User, να διαγράφονται από την βάση και οι εγγραφές του πίνακα groups που αφορούν τον συγκεκριμένο User και με την ίδια ροή, να διαγράφονται από την βάση και οι εγγραφές του πίνακα members που αφορούν το συγκεκριμένο Group.

Στα πεδία που δηλώνουν το id της εγγραφής εφαρμόζονται τα annotation @Id και @GeneratedValue(). Το @Id δηλώνει πως το συγκεκριμένο πεδίο αφορά id, ενώ το @GeneratedValue πως η τιμή του πεδίου αλλάζει αυτόματα, χωρίς να χρειαστεί παρέμβαση. Το @GeneratedValue δέχεται ως παράμετρο το strategy= GenerationType.IDENTITY, ώστε να γνωρίζει πως η τιμή αλλάζει με βάση το id (εικόνα 14).

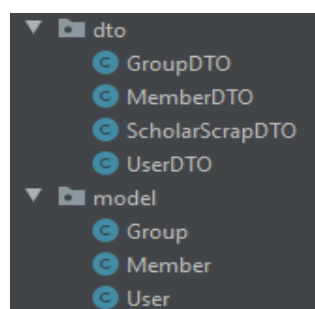
```
@Id
@GeneratedValue(strategy= GenerationType.IDENTITY)
@Column(name="id")
private Integer id;
```

Εικόνα 14: Δήλωση του primary key στο αντίστοιχο πεδίο του Entity και χρήση του @GeneratedValue για την auto-incremental στρατηγική

Ένα ακόμη annotation που έχει χρησιμοποιηθεί, όσον αφορά το package του Model, είναι το @JsonIgnore. Χωρίς το συγκεκριμένο annotation, κατά την κλήση ενός αντικειμένου στο api, επιστρέφονται αναδρομικά τα πεδία που περιέχουν αντικείμενα. Για παράδειγμα, τα Group έχουν Members και το κάθε Member έχει Group, το οποίο Group έχει Members και ούτω καθεξής, με αποτέλεσμα να επιστρέφεται ένα τεράστιο Json αρχείο, το οποίο δεν είναι χρήσιμο. Με το @JsonIgnore σταματάμε αυτή την επανάληψη αγνοώντας τη λίστα Group των Members (εικόνες 15 και 16).

4.1.2 DTO

Οι κλάσεις που ανήκουν στο Model είναι τύπου Entity, είναι δηλαδή οντότητες τις οποίες χρησιμοποιούμε για να έχουμε προβολή στα δεδομένα της βάσης. Αυτές οι κλάσεις δεν μπορούν να προωθηθούν για εμφάνιση στον χρήστη (response), για αυτό τον λόγο υλοποιούνται τα DTO. Τα Data Transferred Objects είναι αυτό που λέει και η ονομασία τους, αντικείμενα για μεταφορά (εικόνα 17). Το πρόγραμμα δέχεται (request) και επιστρέφει (response) δεδομένα τύπου DTO. Οι DTO κλάσεις έχουν επικοινωνία μόνο μεταξύ τους και καμία επικοινωνία με τη βάση.



Εικόνα 15: Παράδειγμα αντιστοίχισης των Model κλάσεων με τις κλάσεις των DTO

```

"id": 8,
"title": "title5",
"description": "description5",
"members": [
  {
    "id": 4,
    "group": {
      "id": 8,
      "title": "title5",
      "description": "description5",
      "members": [
        {
          "id": 4,
          "group": {
            "id": 8,
            "title": "title5",
            "description": "description5",
            "members": [
              {
                "id": 4,
                "group": {
                  "id": 8,
                  "title": "title5",
                  "description": "description5",
                  "members": [
                    {
                      "id": 4,

```

Εικόνα 16: Αναδρομικό αποτέλεσμα χωρίς τη χρήση @JsonIgnore

```

"id": 8,
"title": "title5",
"description": "description5",
"members": [
  {
    "id": 4,
    "authorCode": "authorCode1"
  },
  {
    "id": 5,
    "authorCode": "authorCode2"
  },
  {

```

Εικόνα 17: Χρήση του @JsonIgnore για την απαλοιφή της αναδρομικότητας των αποτελεσμάτων

Σε κάθε DTO κλάση πρέπει να υπάρχει και μια μέθοδος μετατροπής Entity αντικειμένου, σε αντικείμενο DTO. Αυτό, στο παραπάνω παράδειγμα, γίνεται με την μέθοδο `public GroupDTO(Group group)` η οποία δέχεται ως παράμετρο ένα αντικείμενο τύπου Group. Έχοντας το Entity αντικείμενο, και με την μέθοδο `BeanUtils.copyProperties(group, this)` αντιστοιχίζουμε όλα τα πεδία του group στα πεδία της κλάσης this, δηλαδή της GroupDTO. Η παραπάνω μέθοδος αντικαθιστά την μία-μία μετατροπή των πεδίων και την κάνει αυτόματα, αρκεί τα πεδία των δύο κλάσεων να έχουν το ίδιο όνομα. Η μέθοδος αυτή είναι ιδιαίτερα χρήσιμη σε κλάσεις με πολλά πεδία. Η `BeanUtils.copyProperties()` όμως δεν μπορεί να αναγνωρίσει το id του user του group, οπότε θα πρέπει να μετατραπεί ξεχωριστά με την εντολή `this.setUserId(group.getUser().getId());` (εικόνα 18).

```

public class GroupDTO {

    private Integer id;
    private Integer userId;
    private String title;
    private String description;
    private List<MemberDTO> members;

    public GroupDTO(Group group) {

        BeanUtils.copyProperties(group, target: this);
        this.setUserId(group.getUser().getId());
    }

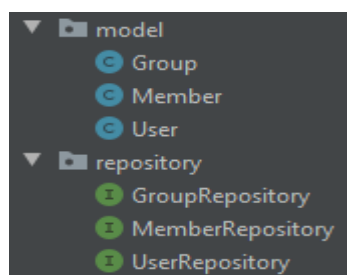
}

```

Εικόνα 18: Υλοποίηση της κλάσης *GroupDTO* και λειτουργία του *Constructor* ως μέθοδο μετατροπής ενός *Entity* σε *DTO*

4.1.3 Repository

Στο package repository πρέπει να υλοποιούνται ξεχωριστά για κάθε Entity τα αντίστοιχα Repository interfaces (εικόνα 19). Στην δήλωση του interface χρειάζεται το annotation *@Repository* για να αναγνωρίζεται το interface σαν repository. Εδώ γίνεται η σύνδεση (extend) με το *JpaRepository*.



Εικόνα 19: Παράδειγμα αντιστοίχισης των κλάσεων *Model* με κλάσεις του *Repository*

Το *JpaRepository* περιλαμβάνει το *CrudRepository* και το *PagingAndSortingRepository*. Το *CrudRepository* περιλαμβάνει όλες τις CRUD (Create, Read, Update, Delete) εντολές που είναι απαραίτητες για ένα API, ενώ το *PagingAndSortingRepository* μεθόδους για σελιδοποίηση και ταξινόμηση των δεδομένων της βάσης. Το *JpaRepository* αναλαμβάνει την απευθείας επικοινωνία με τη βάση και με τις μεθόδους που προσφέρει διευκολύνει την υλοποίηση βασικών εντολών για την

ανάκτηση πληροφοριών από τους πίνακες της βάσης. Δέχεται σαν παραμέτρους την κλάση για την οποία έχει δημιουργηθεί, έτσι ώστε να γίνει η σύνδεση με το κατάλληλο Entity, άρα και η σύνδεση με τους κατάλληλους πίνακες της βάσης. Η δεύτερη παράμετρος αφορά τον τύπο που έχει δηλωθεί το id. Εάν οι βασικές CRUD εντολές αρκούν για τη συγκεκριμένη κλάση-Entity τότε αρκεί και η αναφορά, μέσω του extends, στο JpaRepository (εικόνα 20).

```
@Repository
public interface UserRepository extends JpaRepository<User, Integer> {

    @Modifying(clearAutomatically = true)
    @Query("UPDATE User u SET u.username = ?1, u.password = ?2, u.email = ?3 WHERE u.id = ?4")
    void updateUserProperties(String username, String password, String email, Integer id);
}
```

Εικόνα 20: Παράδειγμα υλοποίησης του UserRepository με extension του JpaRepository

Τα Repositories, είναι καλό σημείο για να υλοποιηθούν custom ερωτήματα (queries) στη βάση με τη λογική ότι για να επικοινωνήσει αργότερα ένα Service με τη βάση, θα χρειαστεί να “περάσει” από το αντίστοιχο Repository. Για την δημιουργία ερωτημάτων χρειάζεται μια μέθοδος η οποία θα δέχεται τις κατάλληλες παραμέτρους και θα γυρνάει ή εκτελεί τα queries. Η σύνθεση των ερωτημάτων γίνεται με το annotation `@Query`. Σαν παράμετρο δέχεται ένα String το οποίο είναι ένα query προς τη βάση και συντάσσεται σε JPQL.

Στην προκειμένη περίπτωση τα Query που έχουν δημιουργηθεί είναι για την περίπτωση που ένας χρήστης (User) θέλει να αλλάξει (UPDATE) κάποια από τα στοιχεία του (username, password, email) και για την περίπτωση που ο ένας χρήστης θέλει να επεξεργαστεί (UPDATE) τα στοιχεία ενός group του (title, description). Όσον αφορά το Query για τα στοιχεία του χρήστη, το JPQL ερώτημα είναι `UPDATE User u SET u.username = ?1, u.password = ?2, u.email = ?3 WHERE u.id = ?4`, όπου τα ερωτηματικά (?) και δίπλα η αρίθμηση (1, 2, 3, 4) δηλώνουν την παράμετρο της μεθόδου και τον αριθμό της στη σειρά. Στη μέθοδο χρησιμοποιείται επίσης το annotation `@Modifying(clearAutomatically = true)` το οποίο είναι ένας ασφαλής τρόπος να “καθαριστούν” τα μη committed δεδομένα στη βάση, όταν το transaction δεν έχει ολοκληρωθεί. Δύο ακόμη queries που έχουν δημιουργηθεί είναι αυτά για την αναζήτηση βάση username και βάση email, ώστε κατά την δημιουργία νέου λογαριασμού (register) να μην υπάρχουν ίδια username και email.

4.1.4 Service

Στο package του service υλοποιούνται τα Services του JPA. Για κάθε Entity υλοποιείται το αντίστοιχο Service του το οποίο είναι interface και για κάθε Service το αντίστοιχο Service Implementation, τα οποία έχουν ομαδοποιηθεί ξεχωριστά σε ένα package “impl” εντός του service.

Στα interfaces αναφέρονται οι μέθοδοι που θα υλοποιηθούν στα Service Implementations και θα καλούνται αργότερα από τους αντίστοιχους Controllers. Υπάρχουν τέσσερις (4) βασικές μέθοδοι που χρησιμοποιούνται οι οποίες αναφέρουν τις βασικές CRUD εντολές που προσφέρει το JpaRepository μέσω των αντίστοιχων υλοποιημένων Repositories που έχουν δημιουργηθεί προηγουμένως. Αυτές είναι :

- *List<DTO> findAll();*
- *DTO findById(Integer id);*
- *DTO save(DTO dto);*
- *void delete(Integer id);*

όπου DTO η αντίστοιχη DTO κλάση για την οποία αναφέρεται.

Τα ServiceImpl (Service Implementation) του κάθε Entity αναλαμβάνουν να υλοποιήσουν τις παραπάνω μεθόδους επικοινωνώντας με τα αντίστοιχα Repositories που δημιουργήθηκαν νωρίτερα. Όσον αφορά τον User και το UserServiceImpl, η πρώτη “κίνηση” είναι να δηλωθεί το implementation του στο UserService και να αποκτήσει η κλάση το annotation *@Service*, ώστε να γίνει γνωστό από το JPA ότι η συγκεκριμένη κλάση αφορά ένα Service (εικόνα 21).

```
@Service
public class UserServiceImpl implements UserService {
```

Εικόνα 21: Χρήση του *@Service* στις Service κλάσεις

Μέσα στην κλάση, αρχικά πρέπει να γίνει κλήση του αντίστοιχου Repository, δηλαδή του UserRepository, ώστε να μπορούν οι μέθοδοι του UserService να έχουν πρόσβαση στις μεθόδους του JpaRepository. Αυτό γίνεται με την χρήση του annotation *@Autowired* της Spring, με το οποίο καταλαβαίνει πως πρέπει να ανατεθεί ένα Bean, δηλαδή να γίνει γνωστό από την εφαρμογή, για το συγκεκριμένο πεδίο (εικόνα 22).

```
@Autowired
private UserRepository userRepository;
```

Εικόνα 22: Χρήση του *@Autowired* για την σύνδεση των Services με τα Repositories

Στη συνέχεια, πρέπει να υλοποιηθούν οι μέθοδοι της UserService στην οποία κάνει implement.

List<UserDTO> findAll()

Σκοπός της μεθόδου είναι να ανακτήσει όλο τον πίνακα με όλα τα δεδομένα του από τη βάση και να τα επιστρέψει σε μια λίστα τύπου UserDTO. Εφόσον θέλουμε όλα τα δεδομένα της βάσης δεν χρειάζεται κάποια παράμετρος. Όπως προαναφέρθηκε, απευθείας σύνδεση με δεδομένα της βάσης έχουν οι Entity-κλάσεις, για αυτό και αρχικά δημιουργείται μια λίστα με όνομα result τύπου User η οποία παίρνει τις τιμές που επιστρέφει η μέθοδος findAll() του userRepository(). Πλέον, υπάρχουν τα δεδομένα της βάσης στο πρόγραμμα αλλά σε κλάση τύπου Entity η οποία δεν γίνεται να προωθηθεί για εμφάνιση. Επόμενο βήμα, είναι η μετατροπή του Entity σε DTO το οποίο έχει υλοποιηθεί μέσα στις κλάσεις των DTO, όπως έχει προαναφερθεί στο κεφάλαιο του DTO, για αυτό και δημιουργείται μια λίστα τύπου UserDTO. Η μετατροπή πρέπει γίνει με μια επανάληψη για το κάθε User τύπου στοιχείο της λίστας result. Για κάθε στοιχείο της λίστας result δημιουργείται ένα στοιχείο τύπου UserDTO το οποίο δημιουργείται με την εντολή `UserDTO userDTO = new UserDTO(user);` όπου user το κάθε στοιχείο της λίστας result. Με αυτόν τον τρόπο μετατρέπεται ένα Entity σε DTO, εφόσον φυσικά υλοποιηθεί κατάλληλα στις κλάσεις του DTO. Τέλος, αρκεί σε κάθε επανάληψη να προστίθεται στην λίστα DTO, που δημιουργήθηκε προηγουμένως, το UserDTO αντικείμενο που δημιουργείται. Αυτή η λίστα είναι αυτή που επιστρέφει η μέθοδος και αυτή που θα καταλήξει στον χρήστη (response) (εικόνα 23).

```
@Override
public List<UserDTO> findAll() {

    List<User> result = userRepository.findAll();

    List<UserDTO> userDTOS = new ArrayList<>();
    for(User user : result){
        UserDTO userDTO = new UserDTO(user);
        userDTOS.add(userDTO);
    }

    return userDTOS;
}
```

Εικόνα 23: Η υλοποιημένη μέθοδος για την αναζήτηση όλων των User

UserDTO findById(Integer id)

Η συγκεκριμένη μέθοδος είναι αρκετά σημαντική, εφόσον επιστρέφει συγκεκριμένη εγγραφή ενός χρήστη σύμφωνα με το μοναδικό id του. Η μέθοδος χρειάζεται σαν παράμετρο το id ώστε να το χρησιμοποιήσει στην μέθοδο της UserRepository `findById(id)`, η οποία επιστρέφει ένα αντικείμενο τύπου User με την εντολή `Optional<User> result = userRepository.findById(id);`. Είναι αναγκαία η χρήση του Optional σε περίπτωση που δεν βρεθεί το συγκεκριμένο id και για την αποφυγή Exception. Στην συνέχεια, ελέγχοντας αρχικά εάν έχει βρεθεί τιμή στο result, γίνεται η μετατροπή του Entity σε ένα νέο αντικείμενο DTO το οποίο και επιστρέφεται (εικόνα 24).

```
@Override
public UserDTO findById(Integer id) {

    Optional<User> result = userRepository.findById(id);

    if(result.isPresent()) {
        UserDTO userDTO = new UserDTO(result.get());
        return userDTO;
    }

    return null;
}
```

Εικόνα 24: Υλοποιημένη μέθοδος για την εύρεση ενός User με το μοναδικό id του

UserDTO save(UserDTO userDto)

Η save() είναι μια μέθοδος που χρησιμοποιεί το JpaRepository και για την δημιουργία μιας νέας εγγραφής (POST – CREATE) αλλά και για την επεξεργασία μιας υπάρχουσας (PUT – UPDATE). Η μέθοδος δέχεται ένα DTO αντικείμενο με συμπληρωμένα πεδία. Εάν δίνεται συμπληρωμένο το id του αντικειμένου, τότε το save() του JpaRepository αναγνωρίζει πως πρόκειται για επεξεργασία υπάρχουσας εγγραφής και ψάχνει την εγγραφή με το δοθέν id. Εάν δεν βρεί εγγραφή με αυτό το id, τότε δεν αποθηκεύει τίποτα στη βάση. Εάν βρεθεί το id, τότε επεξεργάζεται την εγγραφή με τις νέες τιμές. Στην περίπτωση που δεν δίνεται id στο αντικείμενο, τότε αναγνωρίζει πως πρέπει να γίνει μια νέα εγγραφή με id ίσο με το τελευταίο auto-incremented id + 1 και συμπληρώνει τα υπόλοιπα πεδία σύμφωνα με αυτά του αντικειμένου.

Στο συγκεκριμένο πρόγραμμα έχει διαχωριστεί η έννοια της εγγραφής ανάλογα το request. Εάν πρόκειται για νέα εγγραφή, δηλαδή request τύπου POST, τότε θα κληθεί η παραπάνω μέθοδος save(UserDTO userDTO), όπου σε αυτό το σημείο θα κρυπτογραφηθεί ο κωδικός του χρήστη πριν την αποθήκευση στη βάση και θα γίνει αυτή τη φορά μετατροπή DTO αντικειμένου σε Entity. (εικόνα

25). Εάν πρόκειται για επεξεργασία εγγραφής, καλείται η μέθοδος `updatePropetries(UserDTO userDTO)` η οποία επικοινωνεί με το query `updateUserProperties` που δημιουργήθηκε και αναλύθηκε στην ενότητα του Repository. Τέλος, εάν αφορά επεξεργασία/αλλαγή του κωδικού ενός χρήστη, τότε αρχικά καλείται η μέθοδος `confirmPassword()`, για επιβεβαίωση του προηγούμενου κωδικού και έπειτα η `updatePasssword()`, η οποία εκτελεί ένα παρόμοιο query με αυτό του `updateUserProperties`, αλλά για το πεδίο του κωδικού (εικόνες 26, 27).

`void delete(Integer id)`

Αυτή η μέθοδος είναι υπεύθυνη για τη διαγραφή μιας εγγραφής από τη βάση σύμφωνα με το μοναδικό `id` της γραμμής. Αρκεί μόνο το `id` και η κλήση στο κατάλληλο repository ώστε να πραγματοποιηθεί η διαγραφή (εικόνα 28).

```
@Override
public UserDTO save(UserDTO userDTO) throws Exception {

    userDTO.setPassword(HashPassword.getSaltedHash(userDTO.getPassword()));

    User user = this.dtoToEntity(userDTO);

    user = userRepository.save(user);

    return new UserDTO(user);
}
```

Εικόνα 25: Υλοποιημένη μέθοδος για την αποθήκευση ενός User από αντικείμενο τύπου DTO

```
@Override
public Boolean confirmPassword(UserDTO userDTO) throws Exception {

    UserDTO result = this.findById(userDTO.getId());

    return HashPassword.check(userDTO.getPassword(), result.getPassword());
}
```

Εικόνα 26: Μέθοδος για την επιβεβαίωση του κωδικού ενός χρήστη κατά την είσοδό του στην web εφαρμογή

```

@Override
public void updatePassword(UserDTO userDTO) throws Exception {

    String password = HashPassword.getSaltedHash(userDTO.getPassword());

    userRepository.updateUserPassword(
        password,
        userDTO.getId()
    );
}

```

Εικόνα 27: Μέθοδος για την διαχείριση της αλλαγής κωδικού ενός χρήστη

Μια ακόμη σημαντική μέθοδος η οποία βρίσκεται εντός των Service Implementation κλάσεων, είναι αυτή της μετατροπής ενός αντικειμένου από DTO σε Entity, η οποία είναι αναγκαία για την περίπτωση αποθήκευσης νέας εγγραφής ή την επεξεργασία υπάρχουσας. Πρόκειται για μια private μέθοδο η οποία είναι τύπου ενός Entity και δέχεται σαν παραμέτρους ένα αντικείμενο του αντίστοιχου DTO. Συνεχίζοντας νέα εγγραφή, άρα δημιουργία νέου αντικειμένου, ή επεξεργασία υπάρχουσας, άρα αναζήτησή της και αλλαγή των πεδίων σύμφωνα με το αντικείμενο DTO. Αυτό γίνεται με την μέθοδο getId() του παραμετρικού DTO αντικειμένου, όπου εάν υπάρχει id, αφορά επεξεργασία εγγραφής, ενώ εάν δεν υπάρχει δημιουργία νέας. Στη μέθοδο χρησιμοποιείται για ακόμη μια φορά η μέθοδος copyProperties για την γρήγορη αντιγραφή των πεδίων στο νέο αντικείμενο (εικόνα 29).

```

@Override
public void delete(Integer id) {

    userRepository.deleteById(id);
}

```

Εικόνα 28: Κλήση της μεθόδου deleteById του JpaRepository για τη διαγραφή ενός User

```

private User dtoToEntity(UserDTO userDTO) {

    Integer id = userDTO.getId();

    User user = null;

    // ALREADY DEFINED USER, ACCESSED FOR UPDATE
    if (id != null) {
        user = this.userRepository.findById(id).orElse( other: null);
    }

    // NEW USER
    if (user == null) {
        user = new User();
    }

    BeanUtils.copyProperties(userDTO, user);

    return user;
}

```

Εικόνα 29: Μέθοδος μετατροπής μιας κλάσης DTO σε Entity

4.1.5 Controller

Η σύνδεση όλων των παραπάνω με τον χρήστη γίνεται με τους Controllers του κάθε Entity. Οι Controllers είναι υπεύθυνοι για την κλήση των κατάλληλων μεθόδων, των κατάλληλων Services ανάλογα το mapping και τη μέθοδο (GET, PUT, POST, DELETE) που έχει επιλέξει να εκτελέσει ο χρήστης. Οι Controllers είναι επίσης υπεύθυνοι για την επιστροφή/εμφάνιση των αποτελεσμάτων στον χρήστη.

Αρχικά, η κλάση δέχεται δύο annotations. Το *@RestController* δηλώνει πως η κλάση αφορά έναν Controller ώστε να είναι αναγνωρίσιμη από την Spring. Το *@RequestMapping* αφορά το mapping που πρέπει να γίνει, ώστε να είναι ανεξάρτητοι οι Controllers και να υπάρχει ευκολία από τους χρήστες να κάνουν κλήσεις στους κατάλληλους Controllers. Το *@RequestMapping* μπορεί να δεχθεί σαν παράμετρο το path στο οποίο θα πρέπει να κάνει κλήση ο χρήστης ώστε να εκτελεστεί η κατάλληλη μέθοδος (εικόνα 21). Για παράδειγμα, για να μπορέσει να έχει πρόσβαση ένας χρήστης στις μεθόδους της κλάσης UserController, θα πρέπει να στείλει, εάν το πρόγραμμα εκτελείται τοπικά και στην πόρτα 8080, το κατάλληλο request στη διεύθυνση <http://localhost:8080/user> (εικόνα 30).

```
@RestController
@RequestMapping("/user")
public class UserController {
```

Εικόνα 30: Annotations ενός Controller

Εντός της κλάσης, υπάρχουν όλες οι μέθοδοι στις οποίες μπορεί να έχει πρόσβαση ο χρήστης το οποίο γίνεται με το κατάλληλο annotation. Τα τέσσερα βασικά annotation για κάθε μέθοδο προσδιορίζουν και τον τύπο request που μπορούν να δεχθούν. Αυτά είναι :

- @GetMapping()
- @PostMapping()
- @PutMapping()
- @DeleteMapping()

Οι παραπάνω μέθοδοι δέχονται ως παραμέτρους το path που θέλουμε να δημιουργήσουμε ώστε να υπάρχει απευθείας πρόσβαση σε αυτές. Μέσω αυτών, γίνεται η σύνδεση των χρηστών με τις CRUD δυνατότητες ενός Rest API υλοποιημένο με Spring Boot σε Java. Για να έχει πρόσβαση ο χρήστης στη κατάλληλη μέθοδο πρέπει, εκτός της σωστής διεύθυνσης, να στείλει και τον κατάλληλο τύπο request (εικόνα 31)

```
@GetMapping("/all")
public List<UserDTO> findAll() { return userService.findAll(); }

@GetMapping("/{id}")
public UserDTO getUser(@PathVariable Integer id) {
```

Εικόνα 31: Παράδειγμα χρήσης του @GetMapping

Μία μέθοδος μπορεί να έχει το ίδιο path με μία άλλη, αλλά θα πρέπει οπωσδήποτε να έχει διαφορετικό annotation. Οι δύο αυτές μέθοδοι βρίσκονται στο ίδιο path, οπότε στο request του χρήστη και στις δύο περιπτώσεις θα υπάρχει το ίδιο url, αλλά στην πρώτη περίπτωση θα πρέπει το request να είναι τύπου POST, ενώ στην δεύτερη τύπου PUT.

```
@PostMapping("/create")
public UserDTO addUser(@RequestBody UserDTO userDTO) throws Exception {
```

```
@Transactional
@PutMapping("/update")
public UserDTO updateUser(@RequestBody UserDTO userDTO){
```

Εικόνα 32: Παραδείγματα PUT και POST mappings

@RequestBody - @PathVariable

Οι μέθοδοι των Controller δέχονται ως παραμέτρους τα annotation *@RequestBody* και *@PathVariable*. Το *@RequestBody* χρησιμοποιείται όταν τα δεδομένα βρίσκονται στο σώμα (body) του request, ενώ το *@PathVariable* όταν τα δεδομένα βρίσκονται στο url που στέλνει ο χρήστης. Στην πρώτη περίπτωση, δίνεται η δυνατότητα να σταλούν ολόκληρα αντικείμενα, λίστες και κυρίως μεγάλος όγκος δεδομένων, σε αντίθεση με την δεύτερη περίπτωση η οποία χρησιμοποιείται κυρίως για μικρό όγκο δεδομένων, ενώ δεν έχει τη δυνατότητα να στείλει αντικείμενα. Στις εικόνες 31 και 32 φαίνονται οι δύο περιπτώσεις όπου στην πρώτη, ο χρήστης πρέπει να στείλει μια απλή μεταβλητή id ώστε να δεχθεί την εγγραφή με το συγκεκριμένο id από τη βάση και στη δεύτερη πρέπει να στείλει ένα αντικείμενο τύπου UserDTO με πολλαπλά πεδία για επεξεργασία μιας υπάρχουσας εγγραφής στη βάση.

4.1.6 Cross-Origin Resource Sharing (CORS)

Το CORS είναι ένας μηχανισμός που χρησιμοποιεί επιπλέον HTTP κεφαλίδες (headers) οι οποίοι δείχνουν στους browsers πως μια web εφαρμογή η οποία βρίσκεται σε ένα σημείο χρειάζεται πρόσβαση σε ένα άλλο σημείο σε μια άλλη εφαρμογή (API) [33]. Μια web εφαρμογή χρησιμοποιεί τέτοιες κεφαλίδες στα HTTP request όταν το request βρίσκεται είτε σε διαφορετικό domain, είτε σε διαφορετικό πρωτόκολλο, είτε σε διαφορετική πόρτα από 'τι η web εφαρμογή. Για λόγους ασφάλειας τα APIs έχουν τη δυνατότητα να ελέγχουν τις κεφαλίδες αυτές και να επιτρέπουν ή να αποτρέπουν την πρόσβαση στις λειτουργίες τους, ανάλογα το CORS του κάθε request, δηλαδή ανάλογα από που προέρχεται.

Στη Spring αυτή η υλοποίηση για τον έλεγχο του CORS γίνεται με την δημιουργία ενός WebMvcConfigurer. Αρχικά, δηλώνουμε τέσσερις σταθερές που αφορούν τον έλεγχο του CORS (εικόνα 39).

- Max Age : αφορά τον χρόνο που θα μπορούν τα αποτελέσματα ενός request να γίνουν cached

- Allowed Origins : αφορά την προέλευση των request. Είναι ένας πίνακας με τις προελεύσεις των request στις οποίες επιτρέπεται η πρόσβαση στο API
- Allowed Methods : αφορά τις μεθόδους που επιτρέπονται να κάνουν τα HTTP requests (GET, POST, PUT, DELETE)
- Allowed Headers : αφορά τις κεφαλίδες που επιτρέπεται να έχει ένα HTTP request προς το API

Στην εικόνα 33 δηλώνεται πως το Max Age θα είναι 3600 δευτερόλεπτα, δηλαδή για 1 ώρα. Τα Allowed Origins είναι "*", δηλαδή δεν υπάρχει κάποιος περιορισμός και πρόσβαση έχουν όλοι. Οι διαθέσιμες μέθοδοι είναι οι POST, GET, OPTIONS, PUT και DELETE και οι επιτρεπόμενες κεφαλίδες θα πρέπει να περιέχουν τα "X-Requested-With", "content-type" και "Authorization". Εφόσον έχουν δημιουργηθεί οι σταθερές, θα πρέπει να "περάσουν" στον WebMvcConfigurer ως ένα νέο mapping (εικόνα 34).

```
private static final int CORS_MAX_AGE = 3600;
private static final String[] CORS_ALLOWED_ORIGINS = {"*"};
private static final String[] CORS_ALLOWED_METHODS = {
    "POST", "GET", "OPTIONS", "PUT", "DELETE"
};
private static final String[] CORS_ALLOWED_HEADERS = {
    "X-Requested-With", "content-type", "Authorization"
};
```

Εικόνα 33: Παράδειγμα με σταθερές που δεχθεί παραμετρικά το CORS policy

```
public WebMvcConfigurer corsConfigurer() {
    return addCorsMappings(registry) -> {
        registry.addMapping( pathPattern: "/"**)
            .allowedOrigins(CORS_ALLOWED_ORIGINS)
            .allowedMethods(CORS_ALLOWED_METHODS)
            .maxAge(CORS_MAX_AGE)
            .allowedHeaders(CORS_ALLOWED_HEADERS)
    };
}
```

Εικόνα 34: Ενεργοποίηση του CORS policy με προ-δηλωμένες σταθερές

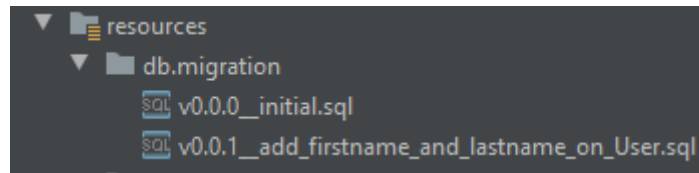
4.1.7 Flyway

Το Flyway [34] είναι ένα εργαλείο για την διευκόλυνση στη συντήρηση των αλλαγών στη βάση δεδομένων (database migrations). Οι αλλαγές σε μία βάση δεδομένων μπορεί να γίνει ένα αρκετά πολύπλοκο πρόβλημα καθώς οι προγραμματιστές θα πρέπει να έχουν υπόψιν όλες τα προβλήματα που μπορεί να προκύψουν σε μια τέτοια αλλαγή. Το Flyway λειτουργεί σαν ένα version control της βάσης δεδομένων το οποίο βοηθάει σε αυτά τα προβλήματα καθώς βοηθάει στο να εξελιχθεί εύκολα η βάση δεδομένων με ακόμη πιο εύκολο τρόπο. Υποστηρίζει ένα τεράστιο πλήθος των πιο γνωστών βάσεων δεδομένων όπως είναι η MySQL, η Oracle, η DB2 και η MariaDB.

```
<!-- FlyWay -->
<dependency>
  <groupId>org.flywaydb</groupId>
  <artifactId>flyway-core</artifactId>
  <version>6.4.4</version>
</dependency>
```

Εικόνα 35: Δήλωση του flyway στη λίστα των dependencies και ένταξή του στο Project

Για να συνδεθεί το Flyway με τη Java θα χρειαστεί να δηλωθεί στο αρχείο pom.xml, το οποίο αναφέρεται στο κεφάλαιο 3.1.3, ως ένα dependency στην λίστα των dependencies (εικόνα 35). Στη συνέχεια θα πρέπει να δηλωθούν στα properties του project τέσσερα στοιχεία για να γνωρίζει το Flyway πως θα διαχειριστεί τα migrations. Αυτά είναι το *baseline-on-migration*, το οποίο παίρνει τιμές true ή false και με τον τρόπο αυτό αναγνωρίζει το Flyway πως υπάρχει μια βασική αρχή της βάσης και να αγνοεί τα migrations από αυτό το σημείο και μετά. Έτσι, τα νέα migrations θα εφαρμοστούν κανονικά. Με το *locations* δηλώνεται το path στο οποίο βρίσκονται τα SQL αρχεία ώστε να γνωρίζει το Flyway το που βρίσκονται. Τα αρχεία αυτά πρέπει να βρίσκονται στα resources του project (εικόνα 36). Με το *sql-migration-prefix* μπορεί το Flyway να ξεχωρίσει τα version κάθε migration. Στο συγκεκριμένο project το prefix είναι το “v” άρα αυτή θα είναι και η τιμή που θα έχει το sql-migration-prefix στα properties του project. Η ονομασία των αρχείων είναι καθολική και χωρίζεται σε τέσσερα μέρη. Το πρώτο μέρος είναι το prefix, έπειτα ακολουθεί ο αριθμός της έκδοσης (version) ο οποίος πρέπει να είναι της μορφής x.y.z. Το Flyway, όταν έρθει η στιγμή να εκτελέσει τα migrations, θα τα εκτελέσει σειριακά σύμφωνα από το μικρότερο νούμερο που έχει δηλωθεί έως το μεγαλύτερο. Το τρίτο μέρος είναι οι δύο κάτω παύλες “__” με τις οποίες χωρίζονται τα version από το όνομα του version το οποίο βρίσκεται στο τέταρτο μέρος και θα πρέπει να είναι όσο πιο περιγραφικό γίνεται. Τέλος, με το *table* δηλώνεται το όνομα του πίνακα που θα υπάρχει στη βάση και σύμφωνα με αυτό, θα γνωρίζει το Flyway για το ποιές εκδόσεις λειτουργούν ή έχουν αλλάξει στη βάση (εικόνα 37).



Εικόνα 36: Τοποθεσία των database versions εντός του project

```
# Flyway
spring.flyway.baseline-on-migrate=true
spring.flyway.locations=classpath:db/migration
spring.flyway.sql-migration-prefix=v
spring.flyway.table=schema_version
```

Εικόνα 37: Παραμετροποίηση του Flyway στο .properties του project

Στο project ο πίνακας που δημιουργεί το Flyway ονομάζεται schema_version και περιέχει 10 στήλες (εικόνα 38).

- *installed_rank* : με αυτή την εγγραφή δείχνει το επίπεδο του version το οποίο το αναγνωρίζει από την αρίθμηση του κάθε version
- *version* : δηλώνεται η αρίθμηση του κάθε version σύμφωνα με το δεύτερο μέρος κατά την ονομασία του SQL αρχείου στον φάκελο των migrations
- *description* : εγγράφεται το περιγραφικό όνομα το οποίο έχει δηλωθεί στο τελευταίο μέρος της ονομασίας του SQL αρχείου
- *script* : στη θέση αυτή εγγράφεται η ονομασία όλου του αρχείου ώστε να το αναγνωρίζει στον φάκελο των migrations
- *checksum* : ένας μοναδικός κωδικός που παράγεται σύμφωνα με αλγόριθμο του Flyway σύμφωνα με τα περιεχόμενα του αρχείου SQL. Εάν γίνει κάποια αλλαγή σε ένα version, κατά την εκτέλεση του Flyway θα εμφανιστεί πρόβλημα καθώς θα έχει παραχθεί ένα καινούριο checksum, ενώ η βάση δεδομένων περιέχει το checksum του version πριν τις αλλαγές.
- *installed_by* : εμφανίζεται ο χρήστης που δημιούργησε το version
- *installed_on* : εμφανίζεται η ημερομηνία που δημιουργήθηκε το version
- *execution_time* : ο χρόνος εκτέλεσης του SQL αρχείου
- *success* : μια boolean μεταβλητή με τιμές 0 ή 1 με 1 εάν το migration του version έγινε με επιτυχία

	installed_rank	version	description	type	script
	1	0.0.0	initial	SQL	v0.0.0__initial.sql
	2	0.0.1	add firstname and lastname on User	SQL	v0.0.1__add_firstname_and_lastname_on_User.sql
▶▶	NULL	NULL	NULL	NULL	NULL

checksum	installed_by	installed_on	execution_time	success
1338033273	root	2020-06-22 21:21:19	126	1
2096452702	root	2020-07-18 20:04:26	205	1
NULL	NULL	NULL	NULL	NULL

Εικόνα 38: Περιεχόμενα του πίνακα *schema_version* στη βάση δεδομένων

Τα SQL script τα οποία υπάρχουν στο φάκελο *resources* και τα αναγνωρίζει το Flyway εκτελούνται κατά την εκτέλεση του project. Αυτό σημαίνει πως ακόμη και αν δεν υπάρχουν στη βάση δεδομένων οι πίνακες του project, θα δημιουργηθούν αυτόματα από το Flyway ακριβώς όπως ορίζουν τα αρχεία στο *resources* και με τη σειρά που έχουν δημιουργηθεί και δηλωθεί τα *versions*. Σε κάθε εκτέλεση το Flyway ελέγχει εάν τα *versions* έχουν εκτελεστεί ελέγχοντας τον πίνακα *schema-version* (εικόνα 34) και αν δεν υπάρχουν όλα ή κάποια από τα *versions* τα δημιουργεί, ενώ αν υπάρχουν τα προσπερνά.

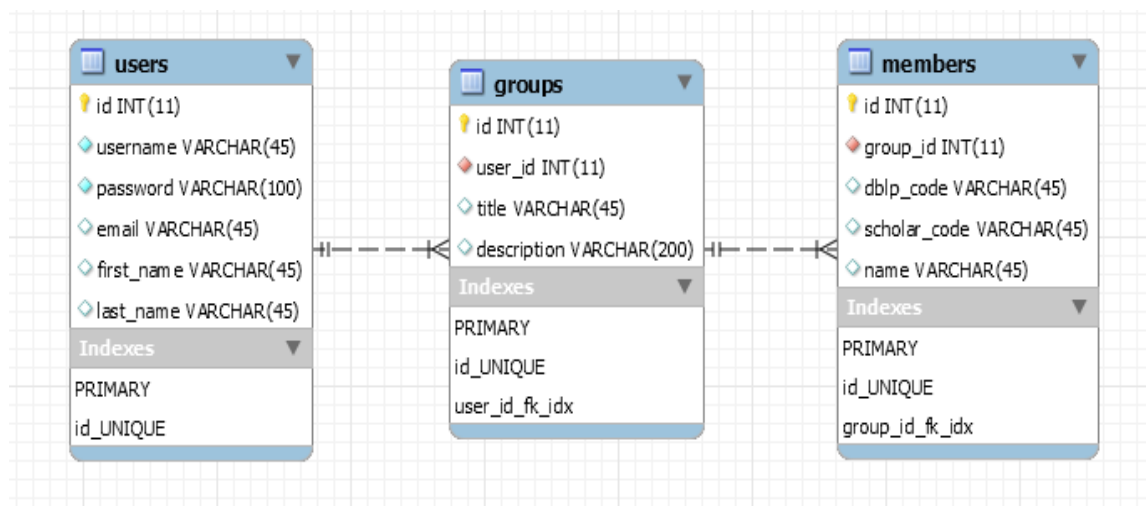
4.1.8 Σύνδεση με τη Βάση Δεδομένων

Τη σύνδεση με τη βάση δεδομένων αναλαμβάνει και πάλι η Spring. Ο προγραμματιστής αρκεί να προσθέσει τέσσερις εγγραφές στο αρχείο *properties* του project. Αυτές αφορούν τον Driver ο οποίος θα πρέπει να χρησιμοποιηθεί, το url στο οποίο “τρέχει” η βάση δεδομένων, το *username* και το *password* για τη σύνδεση με τη βάση (εικόνα 39). Αφού γίνει η σύνδεση, την επικοινωνία αναλαμβάνει το *JpaRepository* της Spring Boot το οποίο έχει αναλυθεί στο κεφάλαιο 4.1.3.

```
# Database
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/sciento?rewriteBatchStatements=true
spring.datasource.username=root
spring.datasource.password=root
```

Εικόνα 39: Παραμετροποίηση στοιχείων για τη σύνδεση στη βάση δεδομένων

Εφόσον στο project έχει προστεθεί η τεχνολογία του Flyway κατά την εκτέλεση της εφαρμογής θα δημιουργηθούν αυτόματα οι πίνακες και τα πεδία σύμφωνα με τα database versions. Στη εικόνα 40 φαίνεται το schema της βάσης.



Εικόνα 40: Schema της MySQL βάσης της εφαρμογής

4.1.9 Scrapping

Google Scholar

Όπως αναφέρθηκε σε προηγούμενα κεφάλαια, το Google Scholar δεν παρέχει API για την αναζήτηση συγγραφέων και των αναφορών τους στη βάση δεδομένων του. Η λύση σε ένα τέτοιο πρόβλημα είναι να γίνει scrap των html σελίδων του κάθε συγγραφέα στο Google Scholar.

Για το scrapping των σελίδων χρησιμοποιήθηκε η βιβλιοθήκη HtmlUnit στη Java με την οποία δίνεται η δυνατότητα αναζήτησης συγκεκριμένων tag μέσα σε μια html σελίδα, δυνατότητα redirect σε μία άλλη και η δυνατότητα χρήσης των xPath. Το xPath είναι μια τεχνολογία με την οποία δίνεται η δυνατότητα για απευθείας πρόσβαση σε ένα tag μέσα σε μια html σελίδα, ακολουθώντας το “μονοπάτι” (path) από ένα αρχικό tag, το οποίο έχει id. Με το xPath λοιπόν, γίνεται αρκετά εύκολη η αναζήτηση μέσα σε μια html σελίδα.

Χρησιμοποιώντας το HtmlUnit, αρχικά, πρέπει να δημιουργηθεί ένας Web Client ο οποίος θα λειτουργεί με παρόμοιο τρόπο, όπως ένας Web Browser σε έναν υπολογιστή ή κινητό. Η δημιουργία του γίνεται σε ξεχωριστή μέθοδο, διότι πρέπει για κάθε Client να -setarisτούν- κάποιες επιλογές ώστε να είναι σε θέση να μπορεί να ανταπεξέλθει στις ανάγκες μας και της σελίδας (εικόνα 41). Η παραπάνω μέθοδος επιστρέφει έναν WebClient ο οποίος έχει τις βασικές επιλογές ενός Chrome, οποίος είναι μέχρι στιγμής και ο πιο γρήγορος Browser από τις διαθέσιμες επιλογές. Επίσης, έχει απενεργοποιημένο τα CSS και τα Javascript αρχεία για λιγότερο χρόνο απόκρισης. Αυτό γίνεται διότι

```
private static WebClient initClient() {

    WebClient client = new WebClient(BrowserVersion.CHROME);
    client.getOptions().setCssEnabled(false);
    client.getOptions().setJavaScriptEnabled(false);
    client.getOptions().setRedirectEnabled(true);
    client.getOptions().setThrowExceptionOnFailingStatusCode(false);

    return client;
}
```

Εικόνα 41: Αρχικοποίηση των options του WebClient

τα δεδομένα που θέλουμε να συλλέξουμε από το Google Scholar βρίσκονται σε μία σελίδα της οποίας γνωρίζουμε το link και δεν χρειάζεται να “πατηθεί” κάποιο κουμπί ώστε να ενεργοποιηθεί ένα Javascript αρχείο. Τα CSS αρχεία είναι καθαρά για εμφάνιση, η οποία αγνοείται. Τέλος, έχει ενεργοποιηθεί η δυνατότητα για redirect δηλαδή, μία σελίδα να μπορεί να αλλάξει σε μία άλλη οποιαδήποτε στιγμή. Πριν επιστραφούν στον χρήστη τα δεδομένα που συλλέχθηκαν πρέπει να κλείνει ο κάθε WebClient με την μέθοδο του *close()*, διότι παραμένει ενεργός και χρησιμοποιεί πόρους.

Εφόσον υπάρχει WebClient, μπορούμε πλέον να καλέσουμε τις σελίδες που θέλουμε για να συλλέξουμε τα δεδομένα. Η κλήση μιας σελίδας γίνεται με την μέθοδο *getPage()* του WebClient η οποία δέχεται σαν παράμετρο το link της σελίδας που θέλουμε (εικόνα 42). Στην δική μας περίπτωση το link είναι αυτό του Google Scholar το οποίο δέχεται τον μοναδικό κωδικό του κάθε συγγραφέα στο σημείο “user=”. Τον μοναδικό αυτόν κωδικό ή μέθοδος τον δέχεται παραμετρικά στην κλήση της. Πλέον, υπάρχει και η html σελίδα σε μορφή ενός αντικειμένου τύπου HtmlPage. Υπάρχουν πάρα πολλές δυνατότητες όσον αφορά το αντικείμενο HtmlPage και μία από αυτές είναι η μέθοδος *getFirstByXPath()* η οποία δέχεται σαν παράμετρο το xPath του αντικειμένου που θέλουμε από την html σελίδα. Το αντικείμενο αυτό αποθηκεύεται σε μια μεταβλητή τύπου DomElement από το οποίο μπορούμε να αποκτήσουμε το text με την μέθοδο *getTextContent()*. Η μέθοδος αυτή επιστρέφει το text σε μία String μεταβλητή με αποτέλεσμα να έχουμε τα δεδομένα που θέλουμε από μια σελίδα σε ένα αντικείμενο εντός του προγράμματος (εικόνα 43).

```
try {
    //initialize WebClient
    WebClient profileClient = initClient();

    //google scholar link from given author code
    HtmlPage profilePage = profileClient.getPage( url: "https://scholar.google.com/citations?hl=en&user=" + code);
```

Εικόνα 42: Δημιουργία της HtmlPage σύμφωνα με το profile του συγγραφέα στη σελίδα του Google Scholar

Όταν πλέον υπάρχουν τα δεδομένα που θέλαμε σε μια μεταβλητή, αποθηκεύονται στο αντικείμενο το οποίο θα επιστρέψει η αρχική μέθοδος που αφορά το scrap στο Google Scholar. Με παρόμοιο τρόπο συλλέγονται και τα υπόλοιπα δεδομένα που χρειαζόμαστε δηλαδή τα: h-index, i10-index και citations per year.

```
// ----- CITATIONS SUM -----  
//element with sum of citations  
DomElement citationsTR = profilePage.getFirstByXPath( xpathExpr: "//*[ @id=\"gsc_rsb_st\"]/tbody/tr[1]/td[2]");  
String citationsSumTXT = citationsTR.getTextContent();  
//set citationsSum  
scholarScrap.setCitationsSum(Integer.parseInt(citationsSumTXT));
```

Εικόνα 43: Παράδειγμα συλλογής πληροφορίας από Div με μέθοδο scraping

dblp

Το dblp παρέχει API για την αναζήτηση περιεχομένων στη βάση δεδομένων τους. Δυστυχώς όμως δεν αρκεί για τις λειτουργίες και τον σκοπό του project. Ο σκοπός του project και οι πληροφορίες που χρειάζεται από τη βάση δεδομένων του dblp είναι η λίστα των δημοσιεύσεων ενός συγγραφέα με επιπλέον πληροφορίες για το έτος δημοσίευσης και το link της δημοσίευσης.

Τα δεδομένα αυτά εμπεριέχονται σε ένα RSS αρχείο το οποίο εμφανίζεται στην αρχική σελίδα κάθε συγγραφέα, τον οποίο επιλέγει ο χρήστης μέσω της διεπαφής του. Ο χρήστης δηλαδή, επιλέγοντας τον συγγραφέα που τον ενδιαφέρει, δημιουργεί ένα GET request στο API με παράμετρο το μοναδικό id του συγγραφέα στο dblp και ένα πεδίο type το οποίο μπορεί να δεχθεί τρεις τιμές, types, pubInf και all. Ο ScrapController είναι υπεύθυνος για τη διαχείριση αυτού του request ο οποίος με τη σειρά του καλεί το ScrapService και τη μέθοδο *getInformation*. Η μέθοδος δέχεται δύο String παραμέτρους, το μοναδικό id που αναφέρθηκε και το type. Το type υπάρχει για να διαχωρίσει τις λειτουργίες που θα εκτελέσει η μέθοδος *getInformation* διότι κάποιος μπορεί να μην έχει την ανάγκη όλων των πληροφοριών που συλλέγονται.

Έχοντας το id του συγγραφέα, μπορεί να βρεθεί και η αρχική του σελίδα, η οποία περιέχει το RSS αρχείο. Με μια απλή διαδικασία scraping όπως υποδείχθηκε στην προηγούμενη ενότητα υπάρχει πλέον και το link του αρχείου το οποίο αρκεί να το διαβαστεί μέσω του προγράμματος (εικόνα 44). Για να γίνει προσπέλαση του αρχείου αρκεί η μέθοδος *getElementsByTagName* που παρέχει το αντικείμενο Document. Η πληροφορία που χρειάζεται βρίσκεται στα tag με όνομα item τα οποία προσδιορίζουν τη κάθε δημοσίευση του συγγραφέα, οπότε με τη *getElementsByTagName("item")* δημιουργείται μια λίστα τύπου NodeList η οποία περιέχει όλες τις δημοσιεύσεις του συγγραφέα και τις πληροφορίες της κάθε μιας. Με μια απλή μέθοδο επανάληψης για όλα τα στοιχεία της λίστας

συλλέγονται και αποθηκεύονται ο τίτλος της δημοσίευσης, το έτος και το link της και αποθηκεύονται σε μια λίστα.

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse(new URL(rss).openStream());
```

Εικόνα 44: Χρήση του *DocumentBuilderFactory* για την ανάλυση του rss αρχείου

Η λειτουργία συλλογής των δεδομένων που αφορούν το αρχείο RSS ολοκληρώνεται με μια μέθοδο μέτρησης των δημοσιεύσεων ανά χρονιά. Αυτή η πληροφορία δεν υπάρχει στο RSS αρχείο με αποτέλεσμα να πρέπει να δημιουργηθεί η μέθοδος *countPublicationsByYear* η οποία δέχεται σαν παράμετρο την παραπάνω λίστα με τις πληροφορίες του RSS που συλλέχθηκαν και επιστρέφει μια νέα λίστα τύπου *PublicationsByYear* η οποία έχει ένα πεδίο *year* για την χρονολογία και ένα πεδίο *publications* για τον αριθμό των δημοσιεύσεων που υπολογίστηκαν. Για να υπολογιστούν αυτοί οι αριθμοί χρησιμοποιείται η μέθοδος *groupingBy* της βιβλιοθήκης των *Collectors* με την οποία ομαδοποιείται η λίστα της παραμέτρου σύμφωνα με το πεδίο *year* που περιέχει κάθε αντικείμενο της (εικόνα 45). Τα ομαδοποιημένα ανά χρονιά πλέον στοιχεία αποθηκεύονται σε μια *Map* λίστα η οποία θα πρέπει να προσπελαθεί για να μετατραπεί σε μια λίστα τύπου *PublicationByYear* η οποία τελικά, θα ταξινομηθεί με την εσωτερική μέθοδο των λιστών *sort*.

```
Map<Integer, List<PublicationInf>> groupedList =
    pi.stream().collect(Collectors.groupingBy(PublicationInf::getYear));
```

Εικόνα 45: Μέθοδος ομαδοποίησης των δεδομένων ανά χρονιά

Η δεύτερη λειτουργία της μεθόδου είναι η απόκτηση της πληροφορίας που αφορά τον τύπο κάθε δημοσίευσης του κάθε συγγραφέα. Η πληροφορία αυτή βρίσκεται στην αρχική σελίδα του συγγραφέα και μπορεί εύκολα να αποκτηθεί με μεθόδους *scrapping*. Κάθε τύπος εμφανίζεται σε ένα *div* με κλάση 'nr' και είναι ο τρόπος με τον οποίο θα συλλεχθούν όλοι οι τύποι. Δίνοντας σαν παράμετρο το *//div[contains(@class, 'nr')]* στην μέθοδο *getByXPath*, αποκτάται η λίστα με όλα τα *div* τα οποία περιέχουν την κλάση *nr*, άρα τα *div* που περιέχουν και την πληροφορία για τον τύπο της κάθε δημοσίευσης. Το *dblp* δηλώνει τον τύπο κάθε δημοσίευσης με έναν χαρακτήρα όπου για παράδειγμα ο χαρακτήρας 'c' δηλώνει τις δημοσιεύσεις τύπου *Conference and Workshop Papers*. Έτσι, από τη λίστα με τα *div* που έχει δημιουργηθεί, αρκεί να συλλεχθεί ο πρώτος χαρακτήρας από το περιεχόμενο τους και να προστεθεί στην αντίστοιχη μεταβλητή-μετρητή (εικόνα 46).


```

//all divs containing types
List<HtmlDivision> typeListDivs = profilePage.getByXPath( xpathExpr: "//div[contains(@class, 'nr')]");
for (DomElement typeListDiv: typeListDivs ) {
    char firstChar = typeListDiv.getTextContent().charAt(1);
    switch (firstChar) {
        case 'c': conferenceAndWorkshopPapers++; break;
        case 'j': journalArticles++; break;
        case 'i': informalAndOtherPublications++; break;
        case 'b': bookAndTheses++; break;
        case 'p': partsInBooksOrCollections++; break;
        case 'e': editorship++; break;
        case 'r': referenceWorks++; break;
        case 'd': dataAndArtifacts++; break;
    }
}
}

```

Εικόνα 46: Μέθοδος συλλογής των τύπων των δημοσιεύσεων

Εάν το type που επέλεξε ο χρήστης είναι “pubInf”, τότε η μέθοδος θα επιστρέψει μόνο τις πληροφορίες που συλλέχθηκαν από το RSS αρχείο του κάθε συγγραφέα καθώς και η λίστα των δημοσιεύσεων ανά χρονιά. Εάν το type είναι “types” τότε η μέθοδος θα επιστρέψει τις πληροφορίες που αφορούν τον τύπο των δημοσιεύσεων μόνο, ενώ αν έχει επιλέξει “all”, θα επιστραφούν και οι δύο παραπάνω πληροφορίες στον χρήστη.

4.1.10 Scraping με τη χρήση νημάτων στη Java

Τι είναι τα νήματα

Τα νήματα είναι μια ανεξάρτητη διαδρομή εκτέλεσης μέσω του κώδικα προγράμματος. Όταν εκτελούνται πολλά νήματα, η διαδρομή του ενός διαφέρει από τη διαδρομή του άλλου. Το multithreading επιτρέπει την ταυτόχρονη εκτέλεση δύο ή περισσότερων τμημάτων ενός προγράμματος για να επιτευχθεί η μέγιστη χρήση της CPU και να ελαχιστοποιηθούν οι χρόνοι εκτέλεσης. Τα νήματα είναι δηλαδή, ελαφριές διαδικασίες μέσα σε μια διαδικασία. Υπάρχουν δύο μηχανισμοί για να δημιουργηθούν νήματα στη Java

- με το extend της κλάσης Thread
- με το implement του Runnable Interface

Αυτή η τεχνική είναι αρκετά πολύτιμη για το project. Κατά μέσο όρο, για να συλλεχθούν οι πληροφορίες ενός συγγραφέα από το Google Scholar και το dblp, χρειάζονται περίπου τέσσερα (4) δευτερόλεπτα. Όταν έρθει το αίτημα στο API για τη σύγκριση των συγγραφέων, αυτό θα πρέπει να συλλέξει πληροφορίες ακόμη και για 30 διαφορετικούς συγγραφείς. Κάτι τέτοιο θα ήταν αρκετά κουραστικό για τον χρήστη καθώς θα έπρεπε να περιμένει για τουλάχιστον δύο (2) λεπτά χωρίς να υπάρχει κάποια ανταπόκριση από το σύστημα. Την λύση σε αυτό το πρόβλημα δίνουν τα νήματα (Threads) και η παράλληλη εκτέλεση που προσφέρουν [35].

Για τη χρήση των νημάτων θα χρειαστεί αρχικά να δημιουργηθούν δύο κλάσεις, μια για τις μεθόδους του Google Scholar και μια για τις μεθόδους του dblp, οι οποίες κάνουν extend το αντικείμενο των νημάτων. Η υλοποίηση και των δύο κλάσεων είναι παρόμοια. Και οι δύο κλάσεις έχουν ένα πεδίο που αφορά είτε το μοναδικό κωδικό του Google Scholar είτε το μοναδικό κωδικό του dblp του κάθε συγγραφέα και ένα πεδίο ανάλογο των πληροφοριών που συλλέγει. Ένα Thread περιέχει την μέθοδο *run()*, η οποία εκτελείται κατά την εκκίνηση του Thread με την εντολή *start()*. Για την κλάση που αφορά τα δεδομένα του Google Scholar καλείται η μέθοδος *getScholarStatsByAuthorCode*, ενώ για τη κλάση που αφορά τη συλλογή δεδομένων από το dblp καλείται η μέθοδος *getInformation* (εικόνες 47, 48).

```
@Getter @Setter
class ScholarThread extends Thread {

    private String code; //Scholar code
    private ScholarScrap scholarScrap;

    public void run(){
        scholarScrap = getScholarStatsByAuthorCode(code);
    }
}
```

Εικόνα 47: Υλοποίηση του νήματος για τις λειτουργίες του Google Scholar

```
@Getter @Setter
class DbpThread extends Thread {

    private String dblpCode; //DBLP id
    private DbpScrap dblpScrap;

    public void run(){
        dblpScrap = getInformation(dbpCode, type: 0);
    }
}
```

Εικόνα 48: Υλοποίηση νήματος για τις λειτουργίες του dblp

Εφόσον έχουν δημιουργηθεί οι κλάσεις, σειρά έχει η δημιουργία της μεθόδου *getMemberTotalInformation()* η οποία ανήκει στη κλάση *ScrapService* με την οποία θα επικοινωνήσει ο Controller όταν το request έχει σχέση με τη σύγκριση των συγγραφέων. Η μέθοδος δέχεται ως παράμετρο μία λίστα από μέλη-συγγραφείς τους οποίους θέλει να συγκρίνει ο χρήστης. Το πρώτο βήμα είναι να αρχικοποιηθούν οι κλάσεις που δημιουργήθηκαν προηγουμένως σε μορφή πίνακα με μέγεθος όσο το μέγεθος των μελών του request (εικόνα 51). Στη συνέχεια, σε μια δομή

```
int membersSize = members.size();
ScholarThread [] scholarThreads = new ScholarThread[membersSize];
DbpThread [] dbpThreads = new DbpThread[membersSize];
```

Εικόνα 49: Αρχικοποίηση των Thread

επανάληψης, για κάθε συγγραφέα δημιουργείται μια θέση στον πίνακα των Thread και για κάθε Thread “περνάνε” οι κωδικοί του Scholar και του dblp στην αντίστοιχη κλάση. Η εκκίνηση των Thread γίνεται με την εντολή *start()*. Εφόσον τα Thread εκτελούνται παράλληλα, η εντολή δεν περιμένει κάποιο αποτέλεσμα και συνεχίζει στην επόμενη (εικόνα 49). Όταν ολοκληρωθεί η εκκίνηση και των δύο Thread για κάθε μέλος, πρέπει να προσπελαστούν όλα τα Thread τα οποία έχουν δημιουργηθεί και να αποθηκευτεί η πληροφορία που συνέλεξαν. Η εντολή *join()* περιμένει να ολοκληρωθεί η λειτουργία του Thread και εφόσον έχει ολοκληρωθεί, μπορεί πλέον να αποθηκευτεί

```

for (int j=0; j<membersSize; j++) {
    scholarThreads[j] = new ScholarThread();
    scholarThreads[j].setCode(members.get(j).getScholarCode());
    scholarThreads[j].start();

    dblpThreads[j] = new DblpThread();
    dblpThreads[j].setDblpCode(members.get(j).getDblpCode());
    dblpThreads[j].start();
}

```

Εικόνα 50: Μέθοδος έναρξης λειτουργίας των Thread

```

for (int k=0; k<membersSize; k++) {
    scholarThreads[k].join();
    dblpThreads[k].join();

    MemberTotalScrap memberTotalScrap = new MemberTotalScrap();
    memberTotalScrap.setMember(members.get(k));

    ScholarScrap scholarScrap = scholarThreads[k].getScholarScrap();
    memberTotalScrap.setScholarInfo(scholarScrap);

    DblpScrap dblpScrap = dblpThreads[k].getDblpScrap();
    memberTotalScrap.setDblpInfo(dblpScrap);

    memberTotalScraps.add(memberTotalScrap);
}

```

Εικόνα 51: Μέθοδος τερματισμού των Thread και συλλογή των πληροφοριών τους

και να συνεχιστεί η προσπέλαση μέχρις ότου δεν υπάρχει κάποιο Thread που εκτελείται (εικόνα 50). Πλέον όλα τα δεδομένα που έχουν συλλεχθεί είναι αποθηκευμένα σε μία λίστα η οποία περιέχει τρία αντικείμενα, ένα αντικείμενο Member που αφορά τον συγγραφέα και τις πληροφορίες του, ένα αντικείμενο ScholarScrap με τις πληροφορίες που σύλλεξε το αντίστοιχο Thread και το DblpScrap με τις πληροφορίες που σύλλεξε το DblpThread. Τα δεδομένα πλέον επιστρέφουν στον χρήστη.

Σύγκριση χρόνων χωρίς τη χρήση νημάτων και με χρήση νημάτων

Η χρήση των νημάτων ήταν καθοριστική για την καλύτερη εμπειρία του χρήστη και πιθανώς για το μέλλον της εφαρμογής. Καταμετρήθηκαν οι χρόνοι εκτέλεσης της μεθόδου για την συλλογή των πληροφοριών και για το Google Scholar και για το Dblp για τυχαίους συγγραφείς. Η τυχαία επιλογή των συγγραφέων δεν επηρεάζει αρκετά τους χρόνους, ακόμη και αν ένας συγγραφέας έχει εκατοντάδες δημοσιεύσεις και αναφορές και ένας άλλος μηδενικές. Οι χρόνοι που επηρεάζουν μια μέθοδο scraping είναι κυρίως η προσπάθεια σύνδεσης στην αντίστοιχη σελίδα και η περιήγηση μέσα σε αυτή, καθώς κάθε λειτουργία περιμένει ανταπόκριση από τη πλευρά του δικού της server. Οι

χρόνοι, λοιπόν, για εκτελέσεις της μεθόδου για συγγραφείς χωρίς τη χρήση νημάτων και οι χρόνοι για εκτελέσεις για συγγραφείς με την χρήση νημάτων φαίνονται στους πίνακες 1 και 2.

Αριθμός συγγραφέων / αριθμός εκτέλεσης	1	2	3	4	5
2	8.17	8.32	8.13	7.51	8.21
5	21.56	20.17	20.48	20.49	20.71
10	43.02	42.08	43.16	45.96	43.21
20	89.08	84.92	90.78	91.42	96.51

Πίνακας 1: Χρόνοι εκτέλεσης των μεθόδων scraping και για το Google Scholar και για το dblp χωρίς τη χρήση νημάτων (σε δευτερόλεπτα)

Αριθμός συγγραφέων / αριθμός εκτέλεσης	1	2	3	4	5
2	4.75	3.81	4.02	3.91	4.21
5	9.18	9.61	9.52	9.89	9.16
10	14.78	15.61	14.24	14.79	15.35
20	30.96	31.77	32.56	33.48	27.68

Πίνακας 2: Χρόνοι εκτέλεσης των μεθόδων scraping και για το Google Scholar και για το dblp με τη χρήση νημάτων (σε δευτερόλεπτα)

Κατά μέσο όρο η χρήση των νημάτων ελαχιστοποιούν τους χρόνους σημαντικά, ειδικά στους υπολογισμούς πολλών συγγραφέων ταυτόχρονα, τους οποίους δεν θα χρειαστεί να “χάσει” ο χρήστης. Παρατηρούμε επίσης πως οι χρόνοι χωρίς τη χρήση νημάτων αυξάνονται σταθερά και αυτό οφείλεται στο ότι οι εκτελέσεις γίνονται σειριακά και εφόσον η συλλογή των πληροφοριών ενός συγγραφέα και για το Google Scholar και για το dblp διαρκεί περίπου 4 δευτερόλεπτα, τα δεδομένα μετά από τις επαναλήψεις είναι αυτά που περιμέναμε. Από την άλλη, με τη χρήση νημάτων, παρατηρούμε πως οι χρόνοι δεν αυξάνονται σταθερά, δηλαδή δεν μπορούμε να συμπεράνουμε, πως εφόσον στους δύο συγγραφείς οι χρόνοι εκτέλεσης είναι περίπου 4 δευτερόλεπτα, άρα ο χρόνος εκτέλεσης για έναν είναι 2 δευτερόλεπτα. Ενώ για 2 συγγραφείς η χρήση των νημάτων μειώνει τον χρόνο κατά 50%, για 20 συγγραφείς ο χρόνος μειώνεται κατά 60%-70%.

Θα πρέπει να συμπεριληφθεί πως οι δοκιμές για τις μετρήσεις των χρόνων έγιναν με μέτρια έως κακή ταχύτητα σύνδεσης στο internet και όπως έχει ήδη αναφερθεί, ο κυριότερος χρόνος στο scraping μιας web σελίδας είναι η ταχύτητα σύνδεσης σε αυτή τη σελίδα και όχι τόσο η συλλογή των πληροφοριών από αυτή. Ενημερωτικά, δεδομένα τα οποία επιστρέφει το API για 20 συγγραφείς ξεπερνούν τις 45.000 γραμμές και το 1.5 MB, ενώ σε κάποιες δοκιμές, λόγω της τυχαιότητας της επιλογής συγγραφέων, ξεπέρασαν και τις 100.000 γραμμές (4 MB).

4.1.11 Διαθέσιμες μέθοδοι του API

Οι μέθοδοι που αναλύθηκαν στο κεφάλαιο 4.2 και αφορούν τις λειτουργίες scraping στο Google Scholar και στο dblp που προσφέρει η υλοποιημένη εφαρμογή, είναι διαθέσιμες προς όλους μέσω των αντίστοιχων mapping των μεθόδων.

Πρόσβαση στις πληροφορίες του Google Scholar

Για να έχει κάποιος πρόσβαση στις πληροφορίες που συλλέγονται όσον αφορά το Google Scholar θα χρειαστεί ένα GET request στο `/api/scholar/{code}`, όπου `code` ο μοναδικός κωδικός κάθε συγγραφέα στο Scholar σε μορφή *String* (εικόνα 52). Ο μοναδικός αυτός κωδικός μπορεί να βρεθεί από το link του profile κάθε συγγραφέα στο Google Scholar στο σημείο “user=xxxxxxxxxx”.

```
@GetMapping("/scholar/{code}")
public ScholarScrap getScholar(@PathVariable String code) throws IOException {
```

Εικόνα 52: Μέθοδος που δέχεται τα request των χρηστών για τις πληροφορίες του Google Scholar

Οι πληροφορίες που επιστρέφει η συγκεκριμένη μέθοδος αφορούν το σύνολο των αναφορών (citations) του συγγραφέα, τις αναφορές ανά χρονιά, το h-index και το i10-index τα οποία αναλύονται στο κεφάλαιο 2.2. Οι πληροφορίες είναι σε μορφή json και δομημένες με τρόπο ώστε να είναι εύκολη η πρόσβαση σε κάθε πληροφορία ξεχωριστά (εικόνα 53).

```

1  {
2    "authorScholarCode": "
3    "citationsSum": 40556,
4    "i10Index": 291,
5    "citationsByYearList": [
6      {
7        "year": 2000,
8        "citations": 111
9      },
10     {
11       "year": 2001,
12       "citations": 157
13     },
14     {
15       "year": 2002,
16       "citations": 117
17     },
18     {
19       "year": 2003,
20       "citations": 100
21     },
22     {
23       "year": 2004,
24       "citations": 100
25     },
26     {
27       "year": 2005,
28       "citations": 100
29     },
30     {
31       "year": 2006,
32       "citations": 100
33     },
34     {
35       "year": 2007,
36       "citations": 100
37     },
38     {
39       "year": 2008,
40       "citations": 100
41     },
42     {
43       "year": 2009,
44       "citations": 100
45     },
46     {
47       "year": 2010,
48       "citations": 100
49     },
50     {
51       "year": 2011,
52       "citations": 100
53     },
54     {
55       "year": 2012,
56       "citations": 100
57     },
58     {
59       "year": 2013,
60       "citations": 100
61     },
62     {
63       "year": 2014,
64       "citations": 100
65     },
66     {
67       "year": 2015,
68       "citations": 100
69     },
70     {
71       "year": 2016,
72       "citations": 100
73     },
74     {
75       "year": 2017,
76       "citations": 4161
77     },
78     {
79       "year": 2018,
80       "citations": 3930
81     },
82     {
83       "year": 2019,
84       "citations": 4041
85     },
86     {
87       "year": 2020,
88       "citations": 2732
89     }
90   ],
91   "hindex": 94
92 }

```

Εικόνα 53: Παράδειγμα αποτελέσματος μιας κλήσης για συγκεκριμένο συγγραφέα

Πρόσβαση στις πληροφορίες του dblp

Για τη πρόσβαση στις μεθόδους scraping του dblp χρειάζεται ένα GET request στο `api/dblp/{type}`, όπου `type` οι πληροφορίες που θέλουμε να επιστρέψει. Στο συγκεκριμένο GET request θα πρέπει να συμπεριληφθεί στις παραμέτρους (params) και το πεδίο `pid` το οποίο αφορά το link του συγγραφέα στη σελίδα του dblp. Το πεδίο `type` δέχεται τρεις διαφορετικές παραμέτρους τύπου *String* ανάλογα την πληροφορία που θέλουμε να επιστρέψει.

- **pubInf:** επιστρέφει πληροφορίες για τις δημοσιεύσεις του συγγραφέα, δηλαδή τον τίτλο της δημοσίευσης, τη χρονιά έκδοσης και το link στο οποίο είναι αναρτημένη η επίσημη δημοσίευση και το σύνολο των δημοσιεύσεων ανά χρονιά.
- **types:** επιστρέφει πληροφορίες που αφορούν τον τύπο των δημοσιεύσεων σύμφωνα με το dblp. Επιστρέφονται 8 μεταβλητές όπου τα ονόματα της κάθε μιας ορίζουν τον τύπο της δημοσίευσης και περιλαμβάνουν το σύνολο των δημοσιεύσεων αυτού του τύπου.
- **all:** επιστρέφει τις προηγούμενες δύο μεθόδους ταυτόχρονα.

Οι πληροφορίες επιστρέφονται σε μορφή json. Παράδειγμα ενός response φαίνεται στην εικόνα 54.

```

1  {
2    "pid": "https://dblp.org/pid,
3    "rss": "https://dblp.org/pid,
4    "conferenceAndWorkshopPapers": 23,
5    "journalArticles": 8,
6    "informalAndOtherPublications": 1,
7    "bookAndTheses": 0,
8    "partsInBooksOrCollections": 0,
9    "editorship": 0,
10   "referenceWorks": 0,
11   "dataAndArtifacts": 0,
12   "publicationsByYear": [
13     {
14       "year": 2007,
15       "publications": 2
16     },
17     {
18       "year": 2009,
19       "publications": 1
20     },
21     {
22       "year": 2011,
23       "publications": 2
24   ]
25 }

```

```

68 {
69   "year": 2020,
70   "title": "[...]",
71   "link": "https://doi.org/10.1007/978-3-319-91234-1_1",
72 },
73 {
74   "year": 2020,
75   "title": "[...]",
76   "link": "https://doi.org/10.1007/978-3-319-91234-1_2",
77 },
78 {
79   "year": 2019,
80   "title": "[...]",
81   "link": "https://doi.org/10.1007/978-3-319-91234-1_3",
82 },
83 {
84   "year": 2019,
85   "title": "[...]",
86   "link": "https://doi.org/10.1007/978-3-319-91234-1_4",
87 },
88 {
89   "year": 2019,
90   "title": "[...]"

```

Εικόνα 54: Παράδειγμα αποτελέσματος της κλήσης με *type = "all"*

4.2 Υλοποίηση Front – end

Η υλοποίηση του front-end, δηλαδή η υλοποίηση του συστήματος με την οποία θα αλληλεπιδρά ο χρήστης πραγματοποιήθηκε με τη χρήση HTML, CSS και JavaScript. Για τη διευκόλυνση των λειτουργιών χρησιμοποιήθηκε η βιβλιοθήκη jQuery η οποία αναλύεται στο κεφάλαιο 3.2.2. Για την εμφάνιση και τη δομή της εφαρμογής έχουν χρησιμοποιηθεί κυρίως μορφοποιήσεις και δομημένες σελίδες του template SB Admin 2 το οποίο παρέχει ένα όμορφο και φιλικό προς το χρήστη περιβάλλον για περιήγηση. Για την προσθήκη επιπλέον κομματιών στη σελίδα χρησιμοποιήθηκαν έτοιμα components του template ώστε η συνολική εικόνα να μην αποκλείνει σε σημεία και να παραμένει ομοιόμορφη.

Προετοιμασία των δεδομένων για την ένταξη τους στα γραφήματα

Η κυρίως προετοιμασία των δεδομένων, όπως είναι η μέθοδος ομαδοποίησης των δημοσιεύσεων ανά χρονιά και η τοποθέτηση τους σε σειρά από την μικρότερη χρονιά προς τη μεγαλύτερη γίνεται στο API. Η εμφάνιση των γραφημάτων των δημοσιεύσεων ανά χρονιά και των αναφορών ανά χρονιά, απαιτεί τον αριθμό των δημοσιεύσεων ή αναφορών για κάθε χρονιά, κάτι το οποίο μπορεί να μην υπάρχει στην πληροφορία που έχει συλλεχθεί από τους βιβλιομετρικούς

ιστότοπους. Για αυτό τον λόγο, η υλοποίηση θα πρέπει να γίνει στο front-end και με κώδικα JavaScript. Η περίπτωση όπου η σύγκριση γίνεται για μια ολόκληρη ομάδα δυσκολεύει ακόμη περισσότερο τον αλγόριθμο, διότι πλέον δεν αναζητούμε τις χρονιές οι οποίες λείπουν από ένα γνωστό διάστημα, αλλά από ένα διάστημα το οποίο βασίζεται στην χαμηλότερη από τις χρονιές των μελών και των δύο ομάδων και από τη μεγαλύτερή τους.

Εφόσον γίνονται δύο ξεχωριστές κλήσεις στο API, μία για κάθε ομάδα που θα συγκριθεί, θα δημιουργηθούν δύο λίστες με τα αποτελέσματα και τι πληροφορίες της κάθε μιας οι οποίες χρησιμοποιούνται ως παράμετροι της μεθόδου *getYearRange(List_1, List_2)*. Σκοπός της μεθόδου είναι να υπολογίσει και να επιστρέψει την ελάχιστη και τη μέγιστη χρονιά των δύο λιστών ώστε να γίνουν τα όρια του γραφήματος. Η μέθοδος *List.map(x => x.field)* δημιουργεί μια νέα λίστα με απομονωμένα τα πεδία “field” της αρχικής λίστας. Δίνοντας σαν “field” το πεδίο *year* και καλώντας την μέθοδο και για τις δύο λίστες, έχουν πλέον δημιουργηθεί δύο νέες λίστες με στοιχεία που αφορούν μόνο τις χρονιές. Επόμενο βήμα είναι η εύρεση του μικρότερου και του μεγαλύτερου στοιχείου αυτών των νέων λιστών. Με τις μεθόδους *Math.min(...List)* και *Math.max(...List)* δημιουργούνται δύο μεταβλητές με την μικρότερη και τη μεγαλύτερη τιμή αντίστοιχα από τη λίστα της παραμέτρου. Η *Math.min* και η *Math.max* πρέπει να υπολογιστεί και για τις δύο λίστες με αποτέλεσμα να υπάρχουν τέσσερις μεταβλητές, δύο που αφορούν την ελάχιστη τιμή των δύο λιστών και δύο που αφορούν τη μέγιστη. Χρησιμοποιώντας ακόμη μια φορά τις *Math.min* και *Math.max* με παραμέτρους τις παραπάνω μεταβλητές αντίστοιχα, *Math.min([List1_min, List2_min])* και *Math.max([List1_max, List2_max])*, υπολογίζονται η ελάχιστη και η μέγιστη χρονιά των δύο αυτών λιστών, οι οποίες επιστρέφονται.

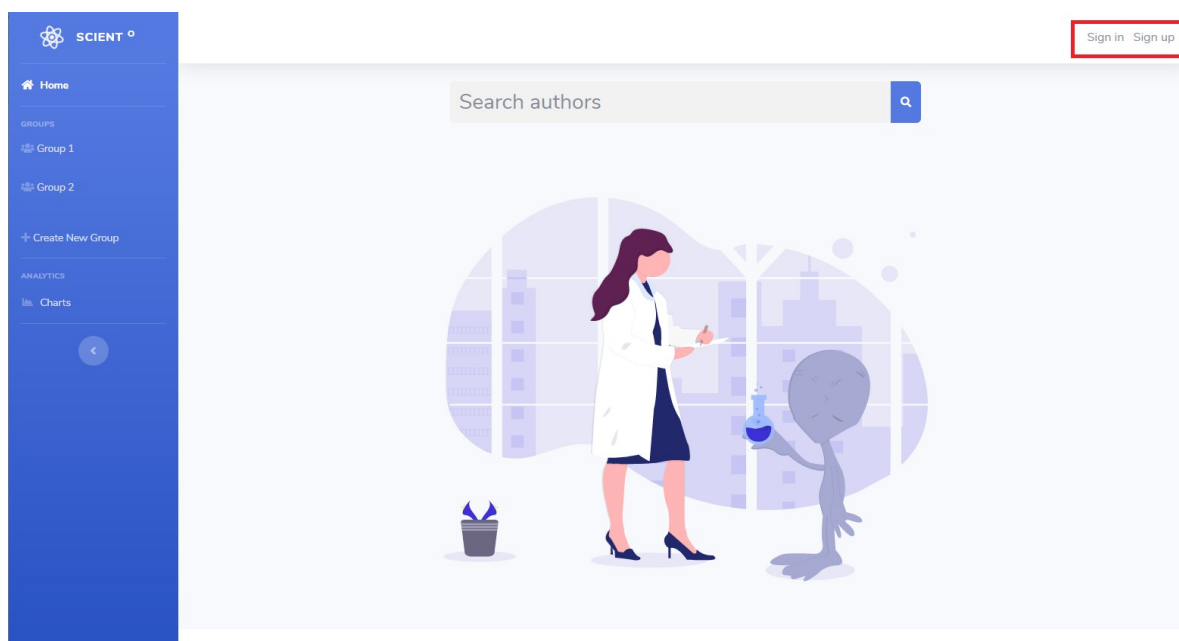
Για να ολοκληρωθεί η πρώτη διάσταση του γραφήματος θα πρέπει να συμπληρωθούν με μηδενικές τιμές οι “κενές” χρονιές όλων των μελών στις οποίες δεν υπάρχει δημοσίευση/αναφορά. Για τη λύση του προβλήματος δημιουργήθηκε η μέθοδος *FillEmptyYears()* η οποία δέχεται τρεις παραμέτρους. Η πρώτες δύο αφορούν τη μέγιστη και ελάχιστη χρονιά που υπολογίστηκε προηγουμένως και η τρίτη τη λίστα η οποία θα πρέπει να συμπληρωθεί. Εντός της μεθόδου αρχικοποιείται μια κενή λίστα η οποία θα είναι αυτή που θα περιέχει το σύνολο των δημοσιεύσεων/αναφορών για κάθε χρονιά, από την ελάχιστη έως τη μέγιστη. Η λογική είναι πως θα γίνει αναζήτηση για κάθε χρονιά από την ελάχιστη έως τη μέγιστη και για κάθε χρονιά θα ελέγχεται εάν η συγκεκριμένη υπάρχει στη λίστα. Με τη μέθοδο *List.find(element => element.year == y)* γίνεται αναζήτηση στη λίστα *List*, όπου στην συγκεκριμένη περίπτωση θα είναι η αρχική, μη συμπληρωμένη λίστα, και συγκεκριμένα στο πεδίο της *year* το οποίο θα ελέγχεται για το εάν είναι ίσο με την μεταβλητή *y*, η οποία προσδιορίζει κάθε φορά την επόμενη χρονιά. Εάν βρεθεί αυτή η χρονιά στη λίστα τότε στον αρχικό κενό πίνακα θα προστεθεί ένα αντικείμενο που περιέχει την χρονιά, άρα το *y*, και τον αριθμό των δημοσιεύσεων/αναφορών από το αντικείμενο που βρέθηκε η χρονιά. Εάν δεν βρεθεί η χρονιά, στη λίστα προστίθεται ξανά η χρονιά (*y*) και για τον αριθμό των δημοσιεύσεων/αναφορών το μηδέν (0). Η μέθοδος επιστρέφει μια ολοκληρωμένη λίστα με

συμπληρωμένες όλες τις χρονιές στα όρια που έχουν υπολογιστεί. Η μέθοδος χρειάζεται να εκτελεστεί και για τις δύο αρχικές λίστες.

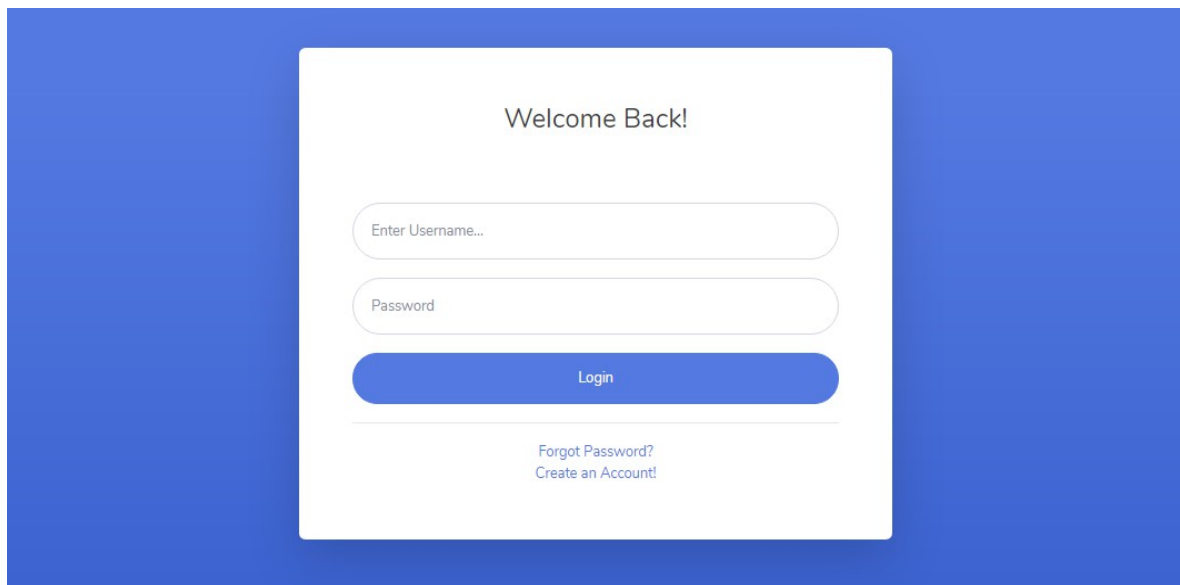
Εφόσον υπάρχουν τα όρια της πρώτης διάστασης του γραφήματος, πρέπει να βρεθούν και όρια της δεύτερης η οποία αφορά το σύνολο των δημοσιεύσεων ή των αναφορών. Αυτή τη φορά είναι γνωστό το ελάχιστο όριο, το οποίο θα είναι το 0, και μένει να βρεθεί το μέγιστο. Με μια παραλλαγή της μεθόδου `getYearRange()` και με τη χρήση των `List.map(x => field)` και `Math.max` γίνεται γνωστό και το μέγιστο όριο της δεύτερης διάστασης του γραφήματος. Η προετοιμασία των δεδομένων έχει ολοκληρωθεί και μπορούν πλέον να συμπληρωθούν στα γραφήματα.

4.3 Οδηγός Χρήσης Sciento

Κατά την είσοδο στην εφαρμογή, ένας χρήστης έχει δύο επιλογές, είτε να παραμείνει ως “ανώνυμος” χρήστης είτε να συνδεθεί στην εφαρμογή. Εάν επιλέξει να συνδεθεί (εικόνα 55) έχει την επιλογή είτε να συνδεθεί με υπάρχων λογαριασμό (εικόνα 56), είτε να δημιουργήσει έναν καινούριο, συμπληρώνοντας κάποια βασικά στοιχεία (εικόνα 57).



Εικόνα 55: Οδηγός Χρήσης Sciento: Σύνδεση/Δημιουργία λογαριασμού



>Welcome Back!

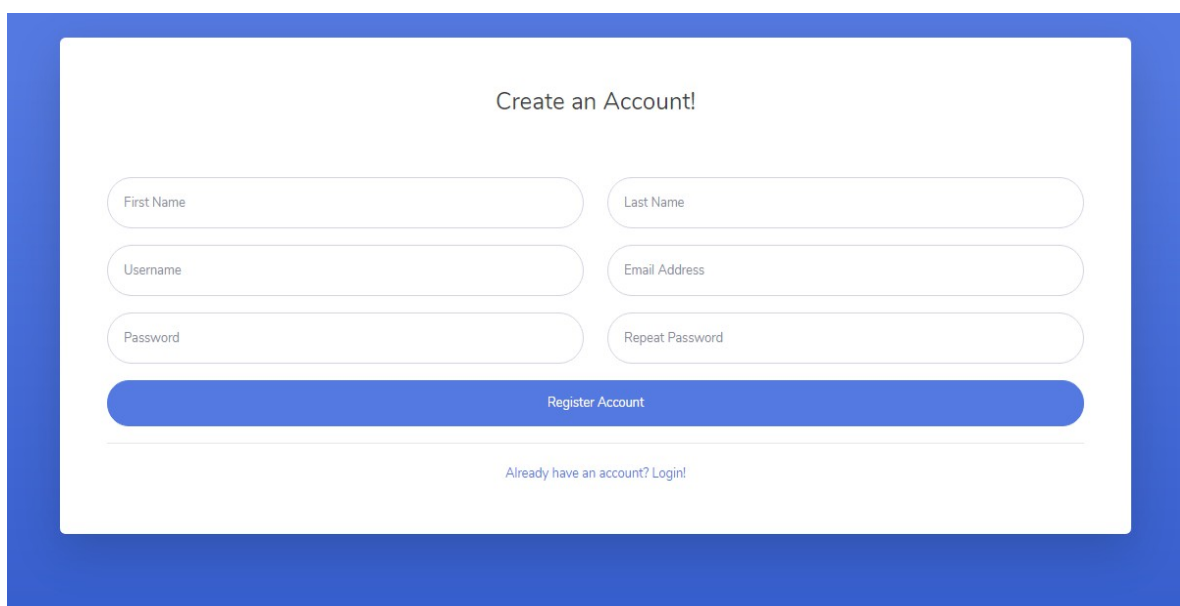
Enter Username...

Password

Login

[Forgot Password?](#)
[Create an Account!](#)

Εικόνα 56: Οδηγός Χρήσης Sciento: Σύνδεση σε λογαριασμό



Create an Account!

First Name

Last Name

Username

Email Address

Password

Repeat Password

Register Account

[Already have an account? Login!](#)

Εικόνα 57: Οδηγός χρήσης Sciento: Δημιουργία λογαριασμού

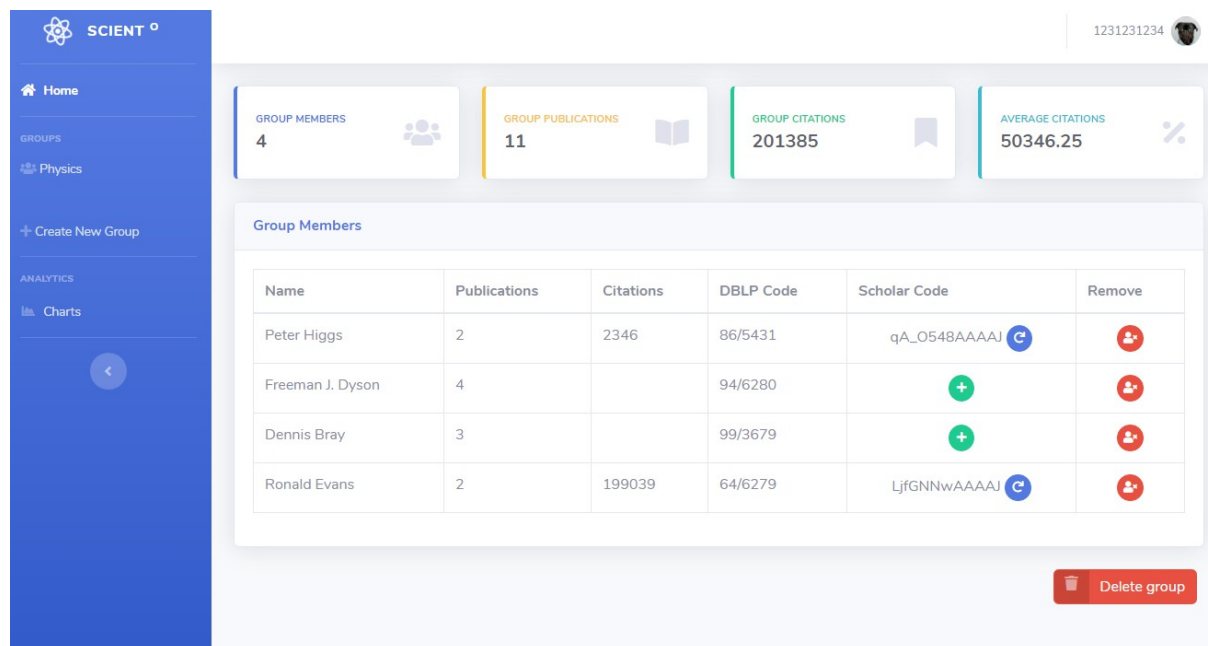
Εφόσον συνδεθεί, θα βρεθεί και πάλι στην αρχική σελίδα της εφαρμογής, στην οποία μπορεί να κάνει αναζητήσεις συγγραφέων με το όνομα τους. Στα αποτελέσματα της αναζήτησης του έχει την δυνατότητα να δει τις δημοσιεύσεις του συγγραφέα (εικόνα 58, επιλογή 1), να δει τα στατιστικά των δημοσιεύσεων του (εικόνα 58, επιλογή 2) ή να προσθέσει τον συγγραφέα σε μία ομάδα (εικόνα 4, επιλογή 58) είτε αυτή η ομάδα υπάρχει είτε δημιουργηθεί εκείνη τη στιγμή.



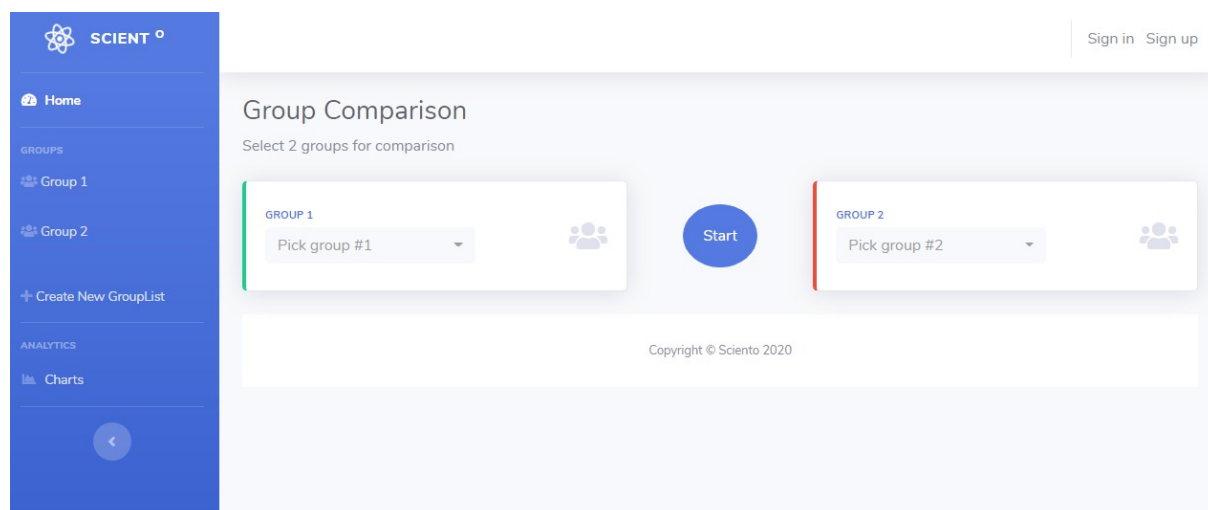
Εικόνα 58: Οδηγός Χρήσης Sciento: Διαθέσιμες λειτουργίες συγγραφέα

Αφού υπάρχουν μέλη σε μια ομάδα, ο χρήστης μπορεί να την επιλέξει από την αριστερή πλευρά της οθόνης, στο σημείο Groups. Κατά την είσοδο στην καρτέλα του group που επιλέχθηκε, γίνεται αυτόματα συλλογή πληροφοριών για όλο το group και συμπληρώνεται η καρτέλα με τις πληροφορίες και τα στατιστικά του group και ατομικά αλλά και ομαδικά (παράδειγμα εικόνα 59).

Εν συνεχεία, και εφόσον ο χρήστης έχει δημιουργήσει ομάδες, μπορεί να μεταβεί στην καρτέλα Charts, στην οποία μπορεί να επιλέξει μέλη από κάθε ομάδα και να τα συγκρίνει μεταξύ τους (εικόνα 60). Τέλος, στην καρτέλα API, μπορεί ο χρήστης να βρει πληροφορίες για τις διαθέσιμες κλήσεις των μεθόδων, οι οποίες συλλέγουν με μεθόδους scraping πληροφορίες από το dblp και το Google Scholar.



Εικόνα 59: Οδηγός Χρήσης Sciento: παράδειγμα ομάδας



Εικόνα 60: Οδηγός Χρήσης Sciento: Σύγκριση ομάδων

Κεφάλαιο 5ο: Συμπεράσματα και προτάσεις βελτίωσης

5.1 Συμπεράσματα

Η εργασία εστίασε στο πρόβλημα της ανάγκης ενός διαδικτυακού τόπου για εύκολη σύγκριση του έργου των ερευνητών και ερευνητικών ομάδων. Οι χρονοβόρες διαδικασίες συλλογής πληροφοριών για την τελική σύγκριση των ερευνητών πλέον ελαχιστοποιούνται και οι αναζητήσεις σε διαφορετικούς τόπους καταργούνται καθώς τα δεδομένα συλλέγονται αυτόματα. Η οπτικοποίηση των αποτελεσμάτων προσφέρει ένα ακόμη βοήθημα στον χρήστη καθώς η σύγκριση των δημοσιεύσεων και στο σύνολο τους και ανά χρονιά αλλά και η σύγκριση των αναφορών και στο σύνολο τους και ανά χρονιά, δεν παραμένει στον έλεγχο απλών αριθμών, αλλά η διαφορές πλέον είναι εμφανής και εύκολες στη διάκριση. Εφαρμογές σαν και αυτή, βοηθούν στην κατάταξη και στην εκτίμηση της επίδοσης ενός ερευνητικού ιδρύματος ή μιας ερευνητικής ομάδας ώστε να σχεδιαστεί η μελλοντική στρατηγική που θα επιφέρει χρηματοδοτικά κεφάλαια και θα προάγει την ακαδημαϊκή και την ερευνητική δραστηριότητα.

Η εφαρμογή δεν αντικαθιστά τη χρήση των διαδικτυακών τόπων βιβλιογραφίας καθώς σε αυτές τα δεδομένα που παρουσιάζονται είναι σαφώς περισσότερα και η πρόσβαση σε αυτά πάρα πολύ εύκολη. Η εφαρμογή συλλέγει και παρουσιάζει τα πιο σημαντικά από αυτά τα δεδομένα, αυτά δηλαδή που θα χρειαστούν για τη σύγκριση δύο ή περισσότερων ερευνητών. Η εφαρμογή φιλοδοξεί να αποτελέσει ένα χρήσιμο εργαλείο για τις διαδικασίες αξιολόγησης της έρευνας στα πανεπιστήμια και στα τμήματά τους για την επιλογή μόνιμου ακαδημαϊκού προσωπικού, ερευνητών και έκτακτου προσωπικού για την κάλυψη θέσεων ευθύνης σε επιστημονικούς φορείς και οργανισμούς.

5.2 Μελλοντικές επεκτάσεις

5.2.1 Δημιουργία προγράμματος για την απαλοιφή των αυτοαναφορών του Google Scholar

Όπως αναφέρθηκε στο κεφάλαιο 2.2 το Google Scholar υπολογίζει στο σύνολο των αναφορών ενός συγγραφέα και τις αυτοαναφορές του. Συμπεριλαμβάνοντας τις αυτοαναφορές στο σύνολο των αναφορών τα πραγματικά νούμερα αποκλίνουν από αυτά που υπολογίζει το Google Scholar με αποτέλεσμα να μην υπάρχει η σωστή αντανάκλαση του ερευνητικού αντίκτυπου του ερευνητή στην κοινότητα. Για αυτό τον λόγο έγινε προσπάθεια απαλοιφής των αυτοαναφορών με μεθόδους web-scraping στην σελίδα του Google Scholar.

Το πρόγραμμα που υλοποιήθηκε διαβάζει από ένα στατικό αρχείο τους κωδικούς των ερευνητών για τους οποίους θα γίνει απαλοιφή των αυτοαναφορών. Για να έχει το πρόγραμμα πρόσβαση σε ορισμένες λειτουργίες πρέπει να δημιουργηθεί αρχικά ένας λογαριασμός στο Google

Scholar. Κατα την εκκίνηση του προγράμματος γίνεται το *Bot* συνδέεται αυτόματα σε αυτό τον λογαριασμό και στη συνέχεια διαβάζει τον αρχείο με τους κωδικούς. Για κάθε κωδικό ερευνητή, ανακατευθύνεται στο profile του όπου θα πρέπει να “φορτωθούν” όλες του οι δημοσιεύσεις. Για την εμφάνιση όλων των δημοσιεύσεων θα πρέπει να πατηθεί επανειλημμένα το κουμπί “Show More” έως ότου δεν εμφανίζεται πλέον, άρα στην οθόνη και σημαντικότερα στην HTML εμφανίζονται όλες οι δημοσιεύσεις οι οποίες συλλέγονται και αποθηκεύονται οι τίτλοι τους σε μια λίστα. Για κάθε μια από αυτές τις δημοσιεύσεις αποθηκεύεται ο εμφανής αριθμός των αναφορών (cited by) και ακολουθείται το link του το οποίο παραπέμπει σε μια νέα σελίδα στην οποία εμφανίζονται όλες οι αναφορές της συγκεκριμένης δημοσίευσης σελιδοποιημένες. Για κάθε μια από αυτές τις σελίδες και για κάθε μια δημοσίευση αυτών των σελίδων συλλέγεται ο τίτλος της και αποθηκεύεται σε μια δεύτερη λίστα. Η επανάληψη συνεχίζεται έως ότου δεν υπάρχει άλλη σελίδα με αναφορές δημοσιεύσεων. Με το πέρας όλων αυτών των επαναλήψεων έχουν δημιουργηθεί δύο λίστες από τις οποίες η πρώτη περιέχει τους τίτλους των δημοσιεύσεων του συγγραφέα και η δεύτερη όλους τους τίτλους των δημοσιεύσεων οι οποίες έχουν κάνει αναφορά σε κάποια δημοσίευση της πρώτης λίστας. Με μία απλή μία προς μία σύγκριση των κοινών περιεχομένων αυτών των λιστών, επιστρέφεται ένας αριθμός που αφορά τις αυτο-αναφορές του συγγραφέα. Η εφαρμογή έχει τη δυνατότητα να εκτελείται ανά συγκεκριμένη ώρα ώστε να ανανεώνονται τα δεδομένα.

Το Google Scholar ασφαλώς χρησιμοποιεί τεχνικές ασφαλείας από τέτοιου είδους προγράμματα καθώς μια συνεχής χρήση με απεριόριστα request στον server του μπορεί να έχει επιπτώσεις στην ταχύτητα και να επηρεάσει την αποδοτικότητα της σελίδας. Μια από τις άμυνες του Google Scholar είναι ο αλγόριθμος για τη χρήση CAPTCHA μετά από επανειλημμένα request στον server ή “ύποπτη” κίνηση και περιήγηση στη σελίδα. Ακόμη και μετά τη χρήση τυχαίων και παραπλανητικών “κλικ” στη σελίδα, τυχαίων χρονικών διαστημάτων μεταξύ των “κλικ” ακόμη και χρήση διαφορετικών εξυπηρετητών (browsers) για τις περιηγήσεις σε κάθε πιθανή διαφορετική σελίδα, ο αλγόριθμος αντιλαμβάνεται την μη ανθρώπινη χρήση της σελίδας και χρησιμοποιεί το CAPTCHA ως άμυνα.

Έπειτα και από την προσπέλαση του CAPTCHA που δημιουργείται, το πρόγραμμα συνεχίζει την “περιήγησή” του για τη συλλογή των πληροφοριών αλλά το Google Scholar χρησιμοποιεί άμυνες και για αυτό. Έπειτα από κάποια request, η IP από την οποία δημιουργούνται “μπλοκάρεται” από τον server για περίπου μία ημέρα και δεν υπάρχει καθόλου η δυνατότητα πρόσβασης. Η θεωρητική λύση αυτού του προβλήματος είναι η χρήση proxy server οι οποίοι θα είναι αυτοί που δημιουργούν τα request με κάποιο όριο. Φυσικά, ανάλογα τον όγκο των συγγραφέων, των δημοσιεύσεων τους και των αναφορών τους θα πρέπει να υπάρχει και ο ανάλογος αριθμός από διαθέσιμα proxy server τα οποία θα χρησιμοποιούνται εναλλάξ και με ρυθμό με τον οποίο δεν θα γίνουν αντιληπτοί από το Google Scholar.

5.2.2 Ελαχιστοποίηση των “ανενεργών” χρόνων του χρήστη

Ίσως από τους πιο σημαντικούς λόγους δυσαρέσκειας ενός χρήστη σε μία οποιαδήποτε σελίδα, είναι ο χρόνος ανταπόκρισης του συστήματος κατά την περιήγηση του χρήστη. Με την έως τώρα υλοποίηση του χρήστη όταν χρησιμοποιήσει την λειτουργία της συλλογής πληροφοριών για τη σύγκριση των συγγραφέων θα πρέπει να παραμείνει “ανενεργός” έως ότου συλλεχθούν και επιστραφούν τα δεδομένα από το API. Αυτός ο χρόνος δεν μπορεί να παραληφθεί για έναν χρήστη ο οποίος δεν είναι συνδεδεμένος στην εφαρμογή. Για έναν συνδεδεμένο όμως χρήστη, ο οποίος έχει αποθηκευμένες ομάδες με μέλη, θα μπορούσε το API να υπολογίζει σε ορισμένες χρονικές περιόδους, είτε ανά μέρα είτε ανά βδομάδα, να εκτελεί αυτόματα τις μεθόδους συλλογής των πληροφοριών για τα μέλη αυτά και να αποθηκεύει σε νέους πίνακες τις πληροφορίες που συλλέχθηκαν έτσι ώστε όταν χρησιμοποιήσει την σύγκριση, τα δεδομένα να επιστραφούν απευθείας από τη βάση, με ενημερωτικό μήνυμα πως τα δεδομένα μπορεί να μην είναι απολύτως έγκυρα, άρα και αρκετά πιο γρήγορα. Φυσικά, θα μπορεί να συνεχίσει να έχει την επιλογή της συλλογής των δεδομένων αυτής της στιγμής.

Με τον ίδιο τρόπο μπορούν να ανανεώνονται τα δεδομένα σε όλες τις περιοχές της σελίδας με αποτέλεσμα οι χρόνοι απόκρισης των λειτουργιών της να μειώνονται όχι απλά κατά 50%, αλλά έως και 99% στην περίπτωση που μια συλλογή πληροφοριών για συγγραφείς ξεπεράσει τα 100 δευτερόλεπτα και η συλλογή των δεδομένων της βάσης διαρκεί 1 δευτερόλεπτο.

ΒΙΒΛΙΟΓΡΑΦΙΑ

- [1] Jogn Mingers and Loet Leydesdorff, *A review of theory and practice in scientometrics*. October 1, 2015
- [2] Irene Pollach, *Software Review: WordStat 5.0*. Aarhus, Denmark. March 26, 2010
- [3] R. Barry Lewis, Steven M. Maas, *QDA Miner 2.0: Mixed-Model Qualitive Data Analysis Software*. University of Illinois at Urbana-Champaign. February 1, 2007
- [4] Feng Zhou, Huai-Cheng Guo, Yuh-Shan Ho and Chao-Zhong Wu, *Scientometric analysis of geostatistics using multivariate methods*. December 1, 2007
- [5] Haiyan Hou, Hildrun Kretschmer and Zeyuan Liu, *The structure of scientific collaboration networks in Scienrometrics*. 2008
- [6] S. Ravikumar, Ashutosh Agrahari and S. N. Singh, *Mapping the intellectual structure of scientrometrics: a co-word analysis of the journal Scientrometrics (2005-2010)*. 2015
- [7] E. Garfield, RK Merton, *Citation indexing: Its theory and appication in science, technology and humanities*. 1979
- [8] Simon Marginson, Marijk van der Wende, *To Rank or To Be Ranked: The Impact of Global Rankings in Higher Education*. September 1, 2007
- [9] Ludo Waltman, Nees Jav van Eck, *The inconsistency of the h-index*. October 31, 2011
- [10] Kate Marek and Edward J. Valauskas, *Web Logs as Indices of Electronic Journal User: Tools for Identifying a "Classic" Article*. 2002
- [11] Johan Bollen, Marko A. Rodriguez, Herbert Van de Sompel, *MESUR: usage-based metrics of scholarly impact*. 2007
- [12] Created By: Anurag Acharya and Alex Verstak, Owned By: Google, Available at: scholar.google.com
- [13] dblp, *computer science bibliography*. Available at: dblp.org
- [14] Scopus, Available at: scopus.com
- [15] ORCID, *Connecting research and researchers*. Available at: orcid.org
- [16] Alireza Noruzi, *Impact Factor, h-index, i10-index and i20-index of Webology* . 2016

- [17] K Arnold, J Gosling, D Holmes, *The Java programming language*. 2000
- [18] Doug Lea, *Concurrent programming in Java: design principles and patterns*. December 2003
- [19] Bruce Eckel, *On Java 8*. MindView LLC., 2017.
- [20] Dragos-Paul Pop and Adam Altar, *Designing an MVC Model for Rapid Web Application Development*. Bucharest, Romania, 2014.
- [21] G. Kiczales, *Aspect-oriented programming*. December, 1996
- [22] R Johnson, *J2EE development frameworks*. January, 2005
- [23] Hatma Suryotrisongko, Dedy Puji Jauanto, Aris Tjahyanto, *Design and Development of Backend Application for Public Complaint Systems Using Microservice Spring Boot*, Indonesia, 2017
- [24] Rango Rao Karanam, *Mastering Spring 5 : an effective guide to build enterprise application using Java Spring and Spring Boot framework*, Birmingham, UK, 2019
- [25] Craig Walls foreword by Anddrew Glover, *Spring Boot in Action*, 2016
- [26] M Masse, *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. 2011
- [27] Gargoyl Software Inc., *HtmlUnit*. Available at: htmlunit.sourceforge.io/
- [28] IBM Cloud Education, *Relational Databases*, Available at: ibm.com/cloud/learn/relational-databases
- [29] Bootstrap. Available at: getbootstrap.com
- [30] jQuery. Available at: jquery.com
- [31] Jonathan Chaffer, *Learning jQuery – Fourth Edition*, 2013
- [32] Lokesh Gupta, *Hibernate JPA Cascade Types*, [Online] Available: <https://howtodoinjava.com/hibernate/hibernate-jpa-cascade-types>
- [33] MDN contributors, *Cross-Origin Resource Sharing (CORS)*. Available at: developer.mozilla.org/en-US/docs/Web/HTTP/CORS
- [34] Redgate, *Flyway*. Available at: flywaydb.org

[35] S. Oaks, H. Wong, *Java Threads: Understanding and Mastering Concurrent Programming*.
2004