ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
Σχολή Εφαρμοσμένων Μαθηματικών και
Φυσικών Επιστημών

ΑΝΑΠΤΥΞΗ ΠΛΑΤΦΟΡΜΑΣ ΥΛΟΠΟΙΗΣΗΣ ΠΡΟΗΓΜΕΝΩΝ ΛΕΙΤΟΥΡΓΙΩΝ ΛΕΞΙΚΟΥ

Διπλωματική Εργασία: Μπότσης Ιάσων

Επιβλέπων: Συμβώνης Αντώνιος, Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2022

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
Σχολή Εφαρμοσμένων Μαθηματικών και
Φυσικών Επιστημών

ΑΝΑΠΤΥΞΗ ΠΛΑΤΦΟΡΜΑΣ ΥΛΟΠΟΙΗΣΗΣ ΠΡΟΗΓΜΕΝΩΝ ΛΕΙΤΟΥΡΓΙΩΝ ΛΕΞΙΚΟΥ

Διπλωματική Εργασία: Μπότσης Ιάσων

Επιβλέπων: Συμβώνης Αντώνιος, Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή:

…….	…….	…….
Συμβώνης Αντώνιος	Κολέτσος Ιωάννης	Στεφανέας Πέτρος
Καθηγητής Ε.Μ.Π.	Αναπληρωτής Καθηγητής Ε.Μ.Π.	Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2022

. . . . . .

Μπότσης Ιάσων

Διπλωματούχος Εφαρμοσμένων Μαθηματικών και Φυσικών Επιστημών Ε.Μ.Π.

Ευχαριστίες

Ευχαριστώ θερμά τον επιβλέποντα της διπλωματικής μου εργασίας κ. Συμβώνη Αντώνιο, Καθηγητή Ε.Μ.Π., για την ανάθεση της συγκεκριμένης εργασίας η οποία μου έδωσε την εμπειρία και τις γνώσεις που αναζητούσα κατά την περάτωση των σπουδών μου.

Εν συνεχεία, θα ήθελα να ευχαριστήσω ξεχωριστά τα μέλη της επιτροπής, τον κ. Κολέτσο Ιωάννη, Αναπληρωτή Καθηγητή Ε.Μ.Π., καθώς και τον κ. Στεφανέα Πέτρο, Αναπληρωτή καθηγητή Ε.Μ.Π., για τη σημαντική συμβολή τους στην περαίωση της.

Ευχαριστίες θα ήθελα, επίσης να απευθύνω στον Παναγιωτόπουλο Διονύση, για την πολύτιμη καθοδήγηση του κατά την έναρξη της εκπόνησης της εργασίας.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένεια μου, τους φίλους μου, και τους συνεργάτες μου οι οποίοι με στήριξαν και με βοήθησαν ο καθένας με τον τρόπο του χωριστά.

Περίληψη

Αν θέλει κανείς να ψάξει κάποια ελληνική λέξη στο διαδίκτυο δεν έχει πολλές επιλογές. Υπάρχει ανάγκη λοιπόν, από ένα λεξικό το οποίο εκτός από τις βασικές πληροφορίες, τις οποίες μπορεί κανείς να βρει σχετικά εύκολα, θα παρέχει και επιπλέον πληροφορίες για ένα λήμμα όπως φωνητική διατύπωση, εικόνες, κατηγοριοποίηση λέξεων κ.α. Ένα τέτοιο λεξικό απαιτεί μια περίπλοκη πλατφόρμα που να το υποστηρίζει. Η διπλωματική αυτή έχει ως στόχο σε πρώτη φάση τον σχεδιασμό ενός τέτοιου συστήματος και σε δεύτερη φάση την υλοποίηση βάσης δεδομένων, υλοποίηση διασύνδεση προγραμματισμού εφαρμογών (API) και τέλος υλοποίηση γραφικού περιβάλλοντος μέσω του οποίου ο χρήστης θα έχει πρόσβαση στην εφαρμογή. Η εφαρμογή παρέχει έλεγχο χρηστών και την δυνατότητα εισαγωγής, επεξεργασίας και διαχείρισης των λημμάτων. Βαθύτερος στόχος της διπλωματικής είναι η εξοικείωση μου με τον σχεδιασμό και την υλοποίηση ενός ολοκληρωμένου συστήματος.

Abstract

If you want to search for a Greek word over the internet, you do not have many options. Therefore, there is a need for dictionary which, in addition to the basic information which can be found relatively easily, will also provide additional information for an entry, such as acoustic spelling of the word, images, word categorization etc. Such a dictionary requires the support of a sophisticated platform. Firstly, this dissertation aims on designing such a system. Secondly, aims on the implementation of the database, the application programming interface (API), and finally the implementation of the graphical environment providing the user access to the application. The application provides user control functionalities and the ability to insert and manage lemmas. The deeper goal of the dissertation is to familiarize myself designing and implementing an integrated system.

Πίνακας περιεχομένων

1 Θεωρητικό μέρος

1.1 Λεξικό

1.1.1 Λήμμα

Ένα λεξικό αποτελείται από διαφορετικά λήμματα λέξεων. Τα λήμματα εκτός από την ίδια τη λέξη παρέχουν και άλλες πληροφορίες σχετικές με την λέξη. Οι πληροφορίες που μπορεί να βρει κανείς σε ένα λήμμα είναι η ίδια η λέξη, ορισμούς και έννοιες αυτής, την ετοιμολογία της, την φωνητική της μορφή, τα μορφολογικά της χαρακτηριστικά, παραδείγματα χρήσης της, κάποια αντώνυμα ή συνώνυμα, την κλιτική της πληροφορία κ.α. Στα ηλεκτρονικά λεξικά τα λήμματα εμπλουτίζονται αρκετές φορές με φωνητικό αρχείο όπου ο χρήστης μπορεί να ακούσει την λέξη και το πως προφέρεται καθώς, εικόνα της λέξης καθώς και κατηγοριοποιήσεις (ξενόγλωσσα ηλεκτρονικά λεξικά).

1.1.2 ELG λεξικό, δομή ενός λήμματος και αναφορά δικαιωμάτων.

Το ELG (European Language Grid) είναι μια πλατφόρμα η οποία έχει δημιουργηθεί με την χρηματοδότηση της Ευρωπαϊκής ένωσης κατά το πρόγραμμα HORIZON 2020. Έχει ως στόχο να στηρίξει τις Ευρωπαϊκές Γλώσσες να εδραιώσουν την δική τους θέση σε μια πανευρωπαϊκή αγορά. Στο ELG έχει δημιουργηθεί μια συνεχώς αναπτυσσόμενη κοινότητα που βοηθά τον συντονισμό και την υποστήριξη των Γλωσσικών τεχνολογιών του ELG. Έχει δημιουργήσει μια ισχυρή και επεκτάσιμη πλατφόρμα η οποία φιλοξενεί τα δεδομένα που έχουν παραχθεί από έρευνες σε πολλά πανεπιστήμια της Ευρώπης. Εμπεριέχει χιλιάδες σύνολα δεδομένων τα οποία υποστηρίζονται από εκατοντάδες Γλωσσικές τεχνολογίες. Κάποια παραδείγματα που θα βρει κανείς στην βάση του είναι δεδομένα για μεταφραστικά λεξικά, π.χ. να μεταφράζουν λέξεις της Ελληνικής σε Αγγλική γλώσσα και αντίστροφα, δεδομένα συνωνύμων και αντωνύμων, δεδομένα λημμάτων, που παρέχουν δηλαδή επιπλέον πληροφορίες για την κάθε λέξη, κ.α. Όλες οι παραπάνω πληροφορίες παρέχονται για διαφορετικές Γλώσσες, ανάλογα με το τι έρευνες έχουν γίνει πάνω στην εκάστοτε

περίπτωση. Σε κάθε έρευνα μπορεί να έρχονται σε συνεργασία ομάδες από διαφορετικά μέρη της Ευρώπης.

Όλα τα δεδομένα που αναρτώνται στο ELG όπως και το λογισμικό, τα μοντέλα κ.α. παρέχονται κάτω από συγκεκριμένες άδειες χρήσης. (“ELG - Greek Annotated Dictionary with Morphological and Linguistic Features,” n.d.)

1.2 Database

Οι βάσεις δεδομένων (Databases), αποτελούν ένα σύνολο πληροφοριών με συγκεκριμένη δομή και οργάνωση, τέτοια ώστε ο υπολογιστής να μπορεί να αναζητήσει και να βρει την επιθυμητή πληροφορία. Συγκεκριμένα, αποτελούν μια κοινόχρηστη, ενσωματωμένη υπολογιστική δομή, η οποία εμπεριέχει τα σχετιζόμενα δεδομένα (data). Μια βάση δεδομένων, περιέχει δύο ειδών δεδομένα. Το πρώτο είδος, end-user data, είναι τα δεδομένα που αποθηκεύονται στην βάση δεδομένων και το δεύτερο είδος είναι το metadata, τα οποία είναι δεδομένα που περιγράφουν τα χαρακτηριστικά των δεδομένων και τις σχέσεις μεταξύ τους. (Coronel and Morris, 2019).

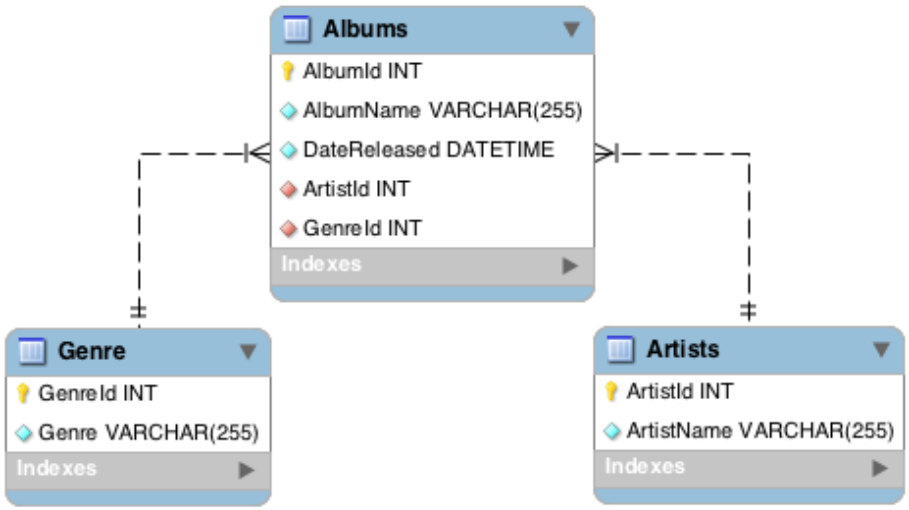
Για τη λειτουργία μιας βάσης δεδομένων είναι απαραίτητη και η ύπαρξη ενός συστήματος διαχείρισης βάσης δεδομένων “Database management system (DBMS)”. Αποτελείται από ένα σύνολο δεδομένων και προγράμματα. Η λειτουργία του συστήματος είναι να διαχειρίζεται τη δομή της βάσης δεδομένων, δηλαδή προσθαφαίρεση δεδομένων και ο έλεγχος στην πρόσβαση των δεδομένων αυτών. (Coronel and Morris, 2019)

Υπάρχουν πολλά είδη μοντέλων που περιγράφουν τις βάσεις δεδομένων. Το ιεραρχικό μοντέλο, το σχεσιακό μοντέλο, το αντικειμενοστραφές μοντέλο, το μοντέλο-οντοτήτων, το μοντέλο μεγάλων δεδομένων (big data) και το μοντέλο NoSQL.

Το σχεσιακό μοντέλο βάσης δεδομένων (relational databases) παριστάνει δεδομένα και τις σχέσεις τους ως ένα σύνολο πινάκων η πιο διαδεδομένη γλώσσα

διαχείρισης του μοντέλου αυτού είναι η γλώσσα αιτημάτων "Structured query language" (SQL).

. Κάθε πίνακας (οντότητα/entity) αποτελείται από στήλες (γνωρίσματα/columns) με μοναδικά ονόματα και κάθε γραμμή του πίνακα (πλειάδα, tuple) παριστάνει μια σχέση ανάμεσα σε ένα σύνολο από τιμές. ("What is a Relational Database?," n.d.). Η σχέση ανάμεσα σε δύο γνωρίσματα ονομάζεται συναρτησιακή εξάρτηση, και ουσιαστικά είναι ο τρόπος που η τιμή ενός γνωρίσματος καθορίζει την τιμή ενός άλλου γνωρίσματος. Το γνώρισμα το οποίο καθορίζει την τιμή του άλλου, ονομάζεται determinant ή key, ενώ το γνώρισμα του οποίου καθορίζεται ονομάζεται, εξαρτώμενο (dependent). (Coronel and Morris, 2019). Γενικότερα τα κλειδιά αποτελούνται από ένα ή παραπάνω γνωρίσματα και καθορίζουν άλλα γνωρίσματα.



Εικόνα 1: Μια μικρή βάση δεδομένων, αποτελούμενη από τρεις οντότητες,

Τα είδη των κλειδιών που υπάρχουν σε μια σχεσιακή βάση δεδομένων είναι το πρωτεύων κλειδί (Primary key), που αποτελείται από ένα ή περισσότερα γνωρίσματα τα οποία προσδιορίζουν με μοναδικό τρόπο κάθε σειρά σε έναν πίνακα βάσης δεδομένων. Το σύνθετο κλειδί, που είναι ένα σύνολο από περισσότερα από ένα κλειδιά, τα οποία από κοινού, προσδιορίζουν μια εγγραφή. Το υπερκλειδί, είναι ένα κλειδί το οποίο μπορεί μοναδικά να προσδιορίσει οποιαδήποτε σειρά σε έναν πίνακα. Το υποψήφιο κλειδί αποτελεί ένα υπερκλειδί με τα λιγότερα γνωρίσματα. Τέλος το ξένο κλειδί (foreign key) αποτελείται από ένα γνώρισμα ή ένα σύνολο γνωρισμάτων ενός πίνακα, του οποίου οι τιμές είναι ίδιες με τις τιμές του πρωτεύοντος κλειδιού σε έναν άλλον πίνακα ή τιμές του σε αυτόν τον πίνακα είναι κενές (null). Με τη χρήση του

ξένου κλειδιού επιτυγχάνεται αναφορική ακεραιότητα, δηλαδή αποφεύγονται αναφορές σε ανύπαρκτες γραμμές (πλειάδες) ενός πίνακα.

Πολύ σημαντικό για την σχεδίαση ενός πίνακα είναι και η ακεραιότητα οντότητας (entity integrity). Η ακεραιότητα οντότητας, είναι η συνθήκη στην οποία κάθε σειρά του πίνακα (entity instance) έχει τη δικιά της μοναδική ταυτότητα. Για να εξασφαλιστεί αυτή η συνθήκη, το πρωτεύον κλειδί θα πρέπει να έχει τιμές μοναδικές και κανένα κενό (null) γνώρισμα.(Coronel and Morris, 2019)

Στις βάσεις δεδομένων, χρησιμοποιείται και μια βοηθητική δομή αρχείου, τα ευρετήρια (indexes). Σκοπός των ευρετηρίων είναι να κάνει πιο γρήγορη και εύκολη την αναζήτηση συγκεκριμένων γνωρισμάτων. Συνήθως καθορίζεται σε ένα γνώρισμα του αρχείου που ονομάζεται πεδίο ευρετηρίασης (indexing field). Το κέρδος από τη χρήση ενός ευρετηρίου είναι ότι δεν χρειάζεται να ανακληθεί όλο το σύνολο των εγγραφών στο αρχείο, παρά μόνο την τιμή στο πεδίο ευρετηρίασης και το δείκτη του μπλοκ στον οποίο είναι η εγγραφή. Επίσης τα ευρετήρια δημιουργούν μεγάλο φόρτο στην βάση δεδομένων, καθώς ανανεώνονται με κάθε εισαγωγή, διαγραφή ή ακόμα και αλλαγή της γραμμής του πίνακα, αυτό απαιτεί την χρήση τους με σύνεση. (“Indexes,” 2016)

1.2.1 Entity Relationship Diagram

Το διάγραμμα οντοτήτων συσχετίσεων παριστάνει το εννοιολογικό σχήμα (conceptual schema) της βάσης δεδομένων όπως φαίνεται στον τελικό χρήστη. Το διάγραμμα αυτό αναπαριστά τα βασικά στοιχεία της βάσης δεδομένων, τα οποία είναι: οντότητες, γνωρίσματα και οι σχέσεις μεταξύ τους. (Silva, n.d.) Το διάγραμμα αυτό είναι σχετικά απλό, αλλά περιέχει επαρκή πληροφορία, τέτοια ώστε να είναι δυνατή η άμεση μετατροπή του σε ένα μοντέλο υλοποίησης.

Οι οντότητες (entities) είναι αντικείμενα, πρόσωπα ή έννοιες του φυσικού κόσμου, ενώ οι συσχετίσεις (relationships) είναι οι σχέσεις που συνδέουν τις οντότητες μεταξύ του. Τόσο οι οντότητες, όσο και οι συσχετίσεις έχουν γνωρίσματα (attributes), τα οποία αφορούν την πληροφορία που θέλουμε να εξασφαλίσουμε στις οντότητες ή τα γνωρίσματα. Για το σχεδιασμό μιας βάσης δεδομένων, χρησιμοποιούνται οι τύποι

οντοτήτων και οι τύποι συσχετίσεων. Ο τύπος οντοτήτων, ορίζει ένα σύνολο από οντότητες που έχουν τα ίδια γνωρίσματα και περιγράφεται από ένα όνομα και ένα σύνολο γνωρισμάτων. Αντίστοιχα, ο τύπος συσχέτισης, περιγράφει ένα σύνολο συσχετίσεων ανάμεσα σε τύπους οντοτήτων και περιγράφεται επίσης από ένα όνομα, από τους σχετιζόμενους τύπους οντοτήτων και τα πιθανά γνωρίσματά τους.

Οι οντότητες διακρίνονται σε δύο τύπους, τους ισχυρούς και μη ισχυρούς. Μη ισχυροί τύποι οντοτήτων, είναι αυτοί που δεν έχουν κλειδιά με δικά τους μόνο γνωρίσματα, ενώ ισχυροί τύποι οντοτήτων είναι οντότητες με κλειδί που περιλαμβάνει γνωρίσματα δικά του γνωρίσματα. (Elmasri and Navathe, 2007)

Αντίστοιχα με τις οντότητες και τα γνωρίσματα έχουν και αυτά διαφορετικά είδη, τα οποία είναι τα εξής:

- **Απλά:** Έχουν μια ατομική τιμή
- **Σύνθετα:** Αποτελούνται από περισσότερα τμήματα στο πεδίο τιμών. Ένα τυπικό τέτοιο παράδειγμα είναι το γνώρισμα της διεύθυνσης, όπου στις τιμές μπορεί να είναι η οδός, ο αριθμός και ο ταχυδρομικός κώδικας.
- **Πλειότιμα:** Σε αυτά τα γνωρίσματα είναι δυνατόν να υπάρχουν περισσότερες από μία τιμές για μια συγκεκριμένη οντότητα.

Σε ένα διάγραμμα του μοντέλου Οντοτήτων-Συσχετίσεων, οι οντότητες αναπαρίστανται ως ορθογώνια παραλληλόγραμμα, οι τύποι συσχετίσεων ως ρόμβοι και τα γνωρίσματα ως ελλείψεις.

Με βάση τις συσχετίσεις ανάμεσα στις οντότητες, προκύπτει ο βαθμός συσχέτισης. Ειδικότερα, Ο βαθμός συσχέτισης αναφέρεται στον αριθμό των οντοτήτων που συμμετέχουν σε αυτή τη σχέση. Οι πιο συνηθισμένες συσχετίσεις μπορεί να είναι οι μοναδικές, στις οποίες ο βαθμός είναι 1, δυαδικές στις οποίες ο βαθμός είναι 2 και τριαδικές που ο βαθμός τους είναι 3.

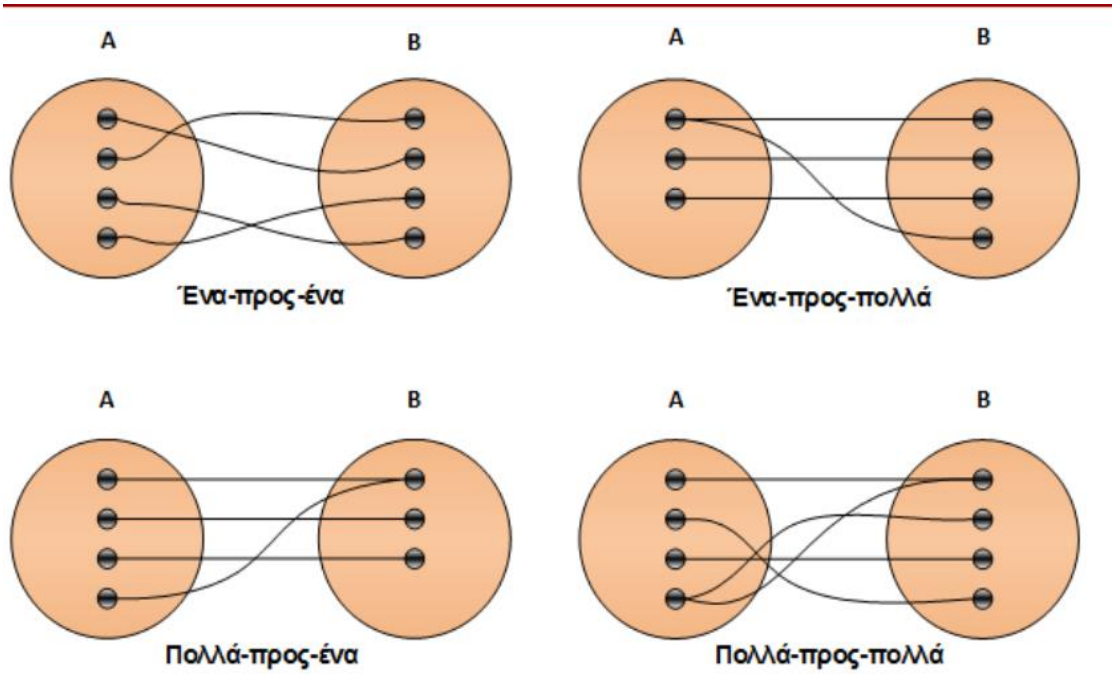
Όπως αναφέρθηκε και προηγουμένως το σύνολο των συσχετίσεων είναι η συλλογή από συσχετίσεις ίδιου τύπου, ενώ ο βαθμός συσχέτισης είναι το πλήθος των οντοτήτων που συμμετέχουν σε μια συσχέτιση. Σημαντικό για το σχεδιασμό μιας βάσης δεδομένων είναι και ο λόγος πληθικότητας. Ως πληθικότητα (cardinality ratio)

ορίζεται ως ο αριθμός των οντοτήτων από ένα σύνολο οντοτήτων που μπορούν να μετέχουν σε μια συσχέτιση.

Οι συνηθέστεροι λόγοι πληθικότητας είναι:

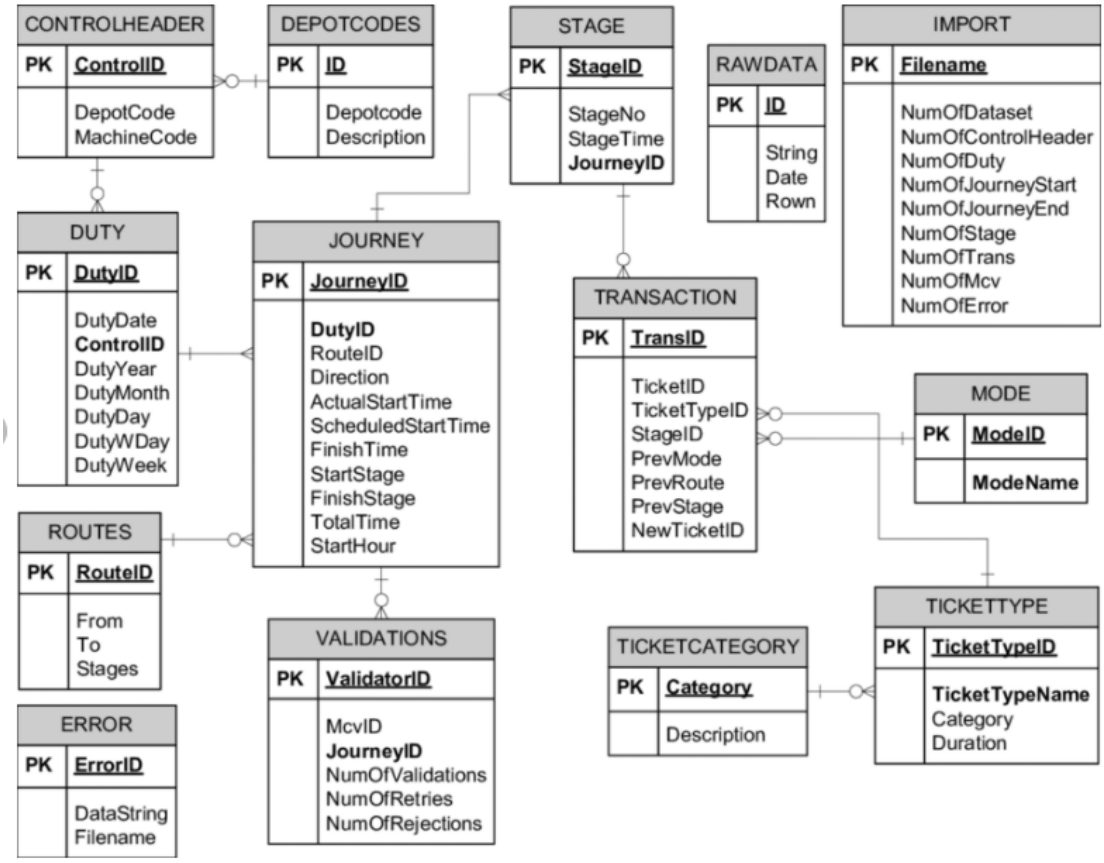
- 1:1 (ένα προς ένα)
- 1:M (ένα προς πολλά)
- M:1 (πολλά προς ένα)
- M:N (πολλά προς πολλά)

Στην ακόλουθη εικόνα απεικονίζονται όλοι οι λόγοι πληθικότητας που αναφέρθηκαν παραπάνω.



Εικόνα 2: Οι λόγοι πληθικότητας σε μια βάση δεδομένων και η φύση τους με την οποία συνδέονται.

Έχοντας όλες αυτές τις πληροφορίες, το διάγραμμα που προκύπτει για μια σχεσιακή βάση δεδομένων φαίνεται ακόλουθα (Hofmann and O'Mahony, 2003)



Εικόνα 3: Διάγραμμα οντοτήτων – συσχετίσεων μιας βάσης δεδομένων για την δομή αστικής συγκοινωνίας λεωφορείων.

1.2.2 SQL

Όπως αναφέρθηκε και παραπάνω για τη διαχείριση μιας σχεσιακής βάσης δεδομένων χρησιμοποίεται η γλώσσα προγραμματισμού Structured Querry Language (SQL). Αναπτύχθηκε τη δεκαετία του 1970 από την IBM, μαζί με την ανάπτυξη του σχεσιακού μοντέλου από τον F. Codd.

Η SQL είναι μια πλήρης γλώσσα βάσεων δεδομένων που βασίζεται στο σχεσιακό λογισμό πλειάδων και τη σχεσιακή άλγεβρα και περιλαμβάνει εντολές για τον ορισμό των δεδομένων στη βάση δεδομένων (DDL), προσφέρει ενημέρωση της βάσης δεδομένων (DML) και τη δυνατότητα αιτημάτων (queries) προς τη βάση δεδομένων. Επίσης είναι μια γλώσσα της οποίας οι εντολές μπορούν να εμφυτευτούν σε πολλές γλώσσες προγραμματισμού όπως η Java, C/C++ και η σύνταξή της είναι

σχετικά απλή. Δεν ορίζεται ο τρόπος με τον οποίο ζητείται κάτι να γίνει, αλλά το τι είναι να γίνει, δηλαδή είναι μια δηλωτική γλώσσα προγραμματισμού.

Η SQL υποστηρίζει επίσης τον ορισμό όψεων (view definition), την εξουσιοδότηση (authentication), την ακεραιότητα (integrity) και τον έλεγχο των συναλλαγών (concurrency control).

Υποστηρίζεται από κάθε σχεσιακό σύστημα, επομένως η διατύπωση των ερωτήσεων δεν εξαρτάται από τη σχεσιακή βάση δεδομένων.

Η χαρακτηριστική δομή μιας ερώτησης σε SQL έχει την εξής μορφή:

Select $A_1, A_2, \ldots, A_n$

From $R_1, R_2, \ldots, R_n$

Where P

Όπου:

Select: τα ονόματα των γνωρισμάτων

From: Τα ονόματα των σχέσεων

Where: Η συνθήκη.

Το πιο συνηθισμένο αίτημα (query) που μπορεί να αιτηθεί ο χρήσης έχει τη μορφή

Select γνώρισμα, From το όνομα του πίνακα, Where η συνθήκη.

Το αποτέλεσμα από το παραπάνω αίτημα είναι η εμφάνιση ενός πίνακα με τα παραπάνω στοιχεία. Στην περίπτωση όπου απουσιάζουν οι γραμμές από τον πίνακα, τότε το αποτέλεσμα είναι ένας κενός πίνακας.

Με αντίστοιχο τρόπο λειτουργεί και το κομμάτι της διαχείρισης (manipulation) της βάσης δεδομένων. (Meier and Kaufmann, 2019, p. 6)

1.2.3 PostgreSQL

Η PostgreSQL είναι ένα σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων, το οποίο χρησιμοποιεί και επεκτείνει τη γλώσσα SQL, είναι ελεύθερο και ανοιχτού κώδικα. Αναπτύχθηκε το 1986, ως μέρος του έργου POSTGRES που αναπτυσσόταν

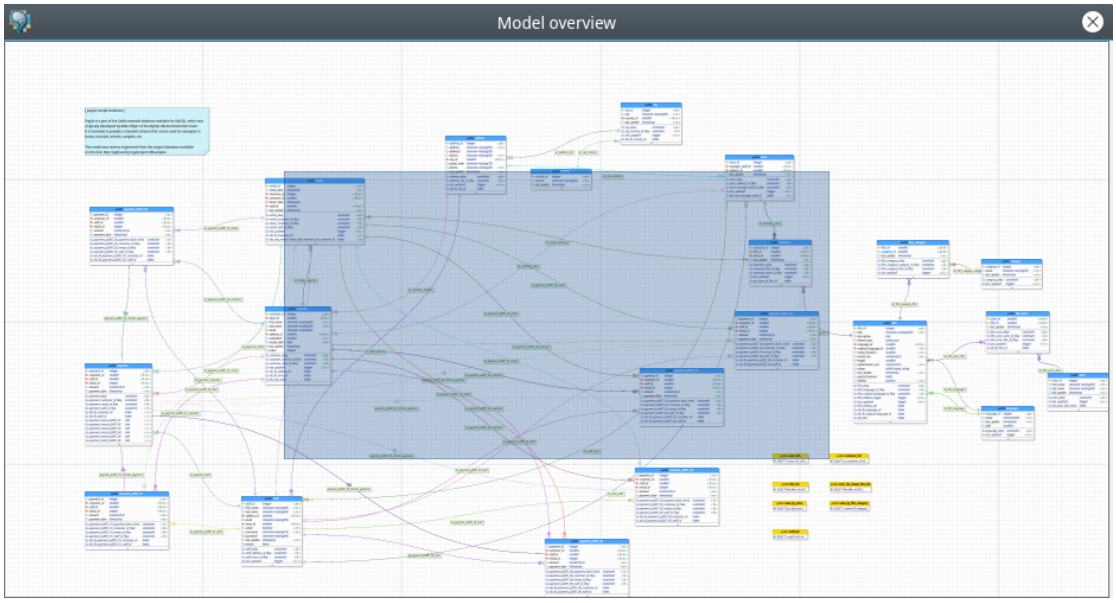
από το Πανεπιστήμιο Καλιφόρνιας. (“PostgreSQL: About,” n.d.) Υποστηρίζει διάφορες λειτουργίες, όπως συναρτήσεις, δείκτες, κανόνες και έχει μεγάλο εύρος σε προκαθορισμένους τύπους δεδομένων και αντικειμένων και υποστηρίζεται από όλες τις κορυφαίες γλώσσες προγραμματισμού.

Η PostgreSQL εκτός από τα SQL (σχεσιακά) queries υποστηρίζει και τα JSON (μη σχεσιακά) και είναι μια από τις κορυφαίες επιλογές για την αποθήκευση μεγάλου όγκους δεδομένων για πολλές διαδικτυακές (web), κινητές (mobile), γεωγραφικές (geospatial) και ανάλυσης (analytics) εφαρμογές. (“What is PostgreSQL?,” n.d.)

1.2.4 Pgmodeler

Για το σχεδιασμό της βάσης δεδομένων βοηθάει ιδιαίτερα η χρήση ενός σχεδιαστικού εργαλείου. Ειδικότερα για το σχεδιασμό μιας σχεσιακής βάσης δεδομένων PostgreSQL, το Pgmodeler αποτελεί ένα εργαλείο για το σχεδιασμό βάσης δεδομένων, το οποίο είναι ανοιχτού κώδικα και είναι ειδικά σχεδιασμένο για αυτήν την χρήση.

Η δυνατότητα του εργαλείου είναι η μοντελοποίηση της βάσης δεδομένων και η εξαγωγή του σε διαφόρων τύπων αρχείων, από εικόνα έως και σε κώδικα SQL για την εισαγωγή του κατευθείαν σε κάποιον server. Εκτός από αυτό, προσφέρει τη δυνατότητα από μια υπάρχουσα βάση δεδομένων να παράξει το εννοιολογικό σχήμα της βάσης (conceptual schema). Τέλος, ένα από τα σημαντικά χαρακτηριστικά του λογισμικού αυτού είναι η δυνατότητα επαλήθευσης των αντικειμένων της βάσης δεδομένων, που προσφέρει στο σχεδιαστή τον περιορισμό λαθών κατά την δημιουργία του κώδικα. (Silva, n.d.)



Εικόνα 4: Αναπαράσταση μιας βάσης δεδομένων, με όλους τους κανόνες και τις σχέσεις μεταξύ των αντικειμένων. Πηγή: (Silva, n.d., p. 1)

1.3 Java

Η Java αποτελεί μια από τις κυρίαρχες γλώσσες προγραμματισμού, αν όχι η κυρίαρχη. Είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού με πολύ μεγάλη συγγένεια με την C++. Ο λόγος για τη δημιουργία αυτής της γλώσσας προγραμματισμού, ήταν η ανάγκη για μια γλώσσα προγραμματισμού η οποία θα μπορούσε να εφαρμοστεί σε οποιαδήποτε πλατφόρμα, όπως οικιακές συσκευές, ανεξαρτήτου αρχιτεκτονικής επεξεργαστή. Ένας από τους πιο σημαντικούς δημιουργούς της ήταν ο James Gosling, όπου μαζί με τους Patrick Naughton, Cris Warth, Ed. Frank, Mike Sheridan στην εταιρία "Sun Microsystems" το 1991, ξεκίνησαν την ανάπτυξη της γλώσσας και ύστερα από 18 μήνες έγινε διαθέσιμη η πρώτη λειτουργική έκδοση της γλώσσας. Το αρχικό όνομα της ήταν "Oak", αλλά μετονομάστηκε σε "Java" το 1995. (Schildt, 2019, p. 54)

Καθοριστικό ρόλο στη διάδοση της γλώσσας ήταν και η ταυτόχρονη ανάπτυξη του παγκόσμιου ιστού (word wide web), καθώς αυξήθηκε η ζήτηση σε προγράμματα τα οποία θα μπορούσαν να τρέξουν ανεξάρτητα σε κάθε υπολογιστή χωρίς να χρειάζεται μεταγλώττιση ή αλλαγή του πηγαίου κώδικα και ανεξάρτητα από το είδος του επεξεργαστή ή του λειτουργικού συστήματος. ("Java | Definition & Facts |

Britannica," n.d.) Ο τρόπος με τον οποίο επιτεύχθηκε αυτό, είναι από το περιβάλλον χρόνου εκτέλεσης (Java Runtime Environment) και συγκεκριμένα από την εικονική μηχανή της Java, Java Virtual Machine (JVM), το οποίο υπάρχει σε διαφορετικές εκδόσεις, για κάθε επεξεργαστή και κάθε λειτουργικό σύστημα και μέσω αυτού μπορούν να τρέξουν όλα τα προγράμματα που είναι γραμμένα σε Java. Η εκτέλεση των προγραμμάτων γίνεται μέσω του bytecode, που είναι το αποτέλεσμα που προκύπτει από το μεταγλωττιστή (compiler) της Java. Έτσι, ένα πρόγραμμα γραμμένο σε Java, εισάγεται στο JVM και μεταγλωττίζεται σε γλώσσα μηχανής (assembly), την οποία καταλαβαίνει ο επεξεργαστής. Ένα άλλο πλεονέκτημα, από τη χρήση του JVM είναι το θέμα της ασφάλειας, καθώς ένα κακόβουλο λογισμικό εκτελείται σε εικονική μηχανή, προστατεύοντας έτσι τη συσκευή. (Schildt, 2019, p. 59)

Πλέον είναι διαθέσιμη η έκδοση JDK 17, με ημερομηνία έκδοσης στις 14/9/2021 και υποστηρίζονται πλέον οι εκδόσεις 8, 11 και 17. ("Java Release History - Dev.java," n.d.)

1.3.1 Java 8

Η 8η έκδοση της Java, Java SE 8, ήταν μια μεγάλη αναβάθμιση της γλώσσας καθώς συμπεριέλαβε τις εκφράσεις λάμδα (lambda expressions) οι οποίες εκφράσεις απλοποιούν και μειώνουν τον πηγαίο κώδικα για την κατασκευή συγκεκριμένων δομών, όπως κάποιες ανώνυμες κλάσεις. Οι εκφράσεις αυτές επηρέασαν και σε μεγάλο βαθμό και τις βιβλιοθήκες της Java, (Java libraries), όπου προστέθηκαν νέες δυνατότητες για την καλύτερη εκμετάλλευσή τους, όπως είναι η νέα διεπαφή (API) για την ώρα και την ημερομηνία, που δεν είχε ανανεωθεί και ήταν αναγκαίο η χρήση βιβλιοθηκών από τρίτους, όπως η Joda-Time (Στάικου and Παλλαδινού, 2015) και τη μέθοδο της παράλληλης ταξινόμησης, parallelSort(), η οποία βελτιώνει τη διαχείριση των στοιχείων των πινάκων, καθώς διαιρεί τον πίνακα σε υποπίνακες ως το ελάχιστο σημείο στο οποίο μπορούν να διαιρεθούν.

1.3.2 Java Beans

Τα JavaBeans έχουν εισαχθεί από την Java 1.0 (1997) όταν η Java ανήκε ακόμα στην Sun microsystems. Το JavaBean είναι μια πολλαπλών χρήσεων οντότητα

λογισμικού που μπορεί να χρησιμοποιηθεί από τον κατασκευαστή. Αυτό μας βοηθάει εξοικονόμηση πόρων καθώς αντί στον κώδικα να δημιουργούμε την ίδια οντότητα πολλές φορές και να κάνουμε μοναδική χρήση ανά κλάση, χρησιμοποιούμε την ίδια οντότητα πολλές φορές. Αυτό μας δίνει επίσης και το πλεονέκτημα να μπορούμε με να μεταφέρουμε να μεταφέρουμε οντότητες μεταξύ διασυνδεμένων έργων των έργων, π.χ. ένα έργο (A) το οποίο μιλάει με την βάση δεδομένων χρειάζεται έναν διαχειριστή συνεδρίας τον οποίο και ορίζουμε ως JavaBean, ένα γονεϊκό του Α έργο (B) χρειάζεται και αυτό να έναν διαχειριστή συνεδρίας, έτσι αντί να δημιουργήσουμε μια καινούρια οντότητα χρησιμοποιούμε τον διαχειριστή που έχουμε ορίσει στο έργο Α φορτώνοντας το αντίστοιχο JavaBean. Τα JavaBeans είναι οντότητες οι οποίες φορτώνονται κατά την έναρξη/χτίσιμο μιας εφαρμογής. (“JavaBeans(TM) Specification 1.01 Final Release,” n.d.)

1.3.3 Spring Framework

Η δομή Spring είναι μια ανοικτού κώδικα δομή εφαρμογών η οποία παρέχει υλοποιήσεις της πλατφόρμας Java. Τα στοιχεία της δομής μπορούν να χρησιμοποιηθούν από εφαρμογές οι οποίες είναι υλοποιημένες σε Java, δεν απαιτεί κάποιο συγκεκριμένο προγραμματιστικό μοντέλο και χρησιμοποιείται ευρέως από την κοινότητα της Java σε συνδυασμό με τα JavaBeans. Η πρώτη παραγωγική έκδοση της δομής Spring κυκλοφόρησε το 2004 και ήταν γραμμένη από τον Rod Johnson και δημοσιεύτηκε υπό την άδεια Apache 2.0. (“Spring Framework 1.0 Final Released,” 15:00:00.0) Η τελευταία έκδοση της είναι η Spring 5 η οποία δημοσιεύτηκε το 2017. Η δομή αυτή παρέχει υλοποιημένα πακέτα, κάποια από αυτά είναι:

- **MVC:** Πακέτο το οποίο βασίζεται σε 3 στοιχεία, το μοντέλο που καθορίζει την δομή των δεδομένων, τον διαχειριστή ο οποίος παρέχει την λογική και διαχειρίζεται τα αιτήματα που λαμβάνει και την όψη η οποία καθορίζει τα στοιχεία του περιβάλλοντος του χρήστη. Είναι βασικό πακέτο για ανάπτυξη εφαρμογών που χρησιμοποιούν το πρωτόκολλο REST.

- **Authentication and Authorization:** Είναι πακέτο το οποίο βοηθάει στην εδραίωση ασφάλειας μιας εφαρμογής. Παρέχει υποστήριξη για μεγάλη γκάμα

από πρωτόκολλα, κανόνες και εργαλεία τα οποία βοηθούν στην ταυτοποίηση αλλά και την εξουσιοδότηση μια διεργασίας, είτε αυτή είναι κάποιο HTTP αίτημα είτε είναι η είσοδος ενός χρήστη στην εφαρμογή με χρήση της κονσόλας.

- **Dependency Injection:** Η λειτουργία (πακέτο) αυτή βρίσκεται στον πυρήνα της δομής Spring. Με την χρήση των JavaBeans ο container της εφαρμογής είναι υπεύθυνος για την διαχείριση του κύκλου ζωής των αντικειμένων, δηλαδή της δημιουργίας τους, της αρχικοποίησής τους, της παραμετροποίησης τους και της διασύνδεσης τους μεταξύ τους. Ο container διαμορφώνεται/παραμετροποιείται από ειδικά αρχεία γνωστά και ως αρχείο ιδιοτήτων (property files). Τα αρχεία αυτά είναι γραμμένα σε μορφή XML (Extensive Markup Language) και περιέχουν τον ορισμό και την παραμετροποίηση των beans καθώς και με ποια κλάση συνδέεται το κάθε bean. Ένα αρχείο παραμετροποίησης μπορεί να φορτώσει και διαδοχικά ένα άλλο αρχείο παραμετροποίησης και να χρησιμοποιήσει τα beans που έχουν οριστεί εκεί, με αυτόν τον τρόπο αποφεύγουμε το να ορίσουμε παραπάνω από μία φορά το ίδιο bean. Κατά την εκκίνηση μίας εφαρμογής φορτώνονται τα αρχεία αυτά και αρχικοποιούνται και τα απαραίτητα beans. Τα beans αυτά στον κώδικα καλούνται με την χρήση του annotation '@Autowired'. Έτσι σε όλα τα σημεία που χρησιμοποιείται αυτό το χαρακτηριστικό έχουμε το dependency injection, γιατί εισάγεται σε όλα αυτά τα σημεία η ίδια υπόσταση ενός αντικειμένου.
  ("Spring Framework Overview," n.d.)

- **Δομή πρόσβασης σε δεδομένα (Data access framework):** Στο οικοσύστημα του Spring υπάρχουν πολλά πακέτα που δίνουν πρόσβαση στον χρήστη στα δεδομένα, το πακέτο καθορίζεται από την βάση δεδομένων (postgresql, mySQL, OracleSQL κ.α.) με την οποία έχει επικοινωνία η εφαρμογή. Στην Java χρησιμοποιείται ευρέως η δομή Hibernate η οποία υλοποιεί την αντιστοίχιση του αντικειμενοστραφούς μοντέλου σε μοντέλο σχεσιακής βάσης δεδομένων. Έχει δικό της συντακτικό παρόμοιο με αυτό της της SQL αλλά είναι υψηλότερου επιπέδου (επίπεδα εφαρμογής). Ανάλογα με την βάση

με την οποία επικοινωνεί η εφαρμογή, πρέπει να οριστεί και αντίστοιχη διάλεκτος στην παραμετροποίηση της hibernate. (“Your relational data. Objectively. - Hibernate ORM,” n.d.)

```
18      @Autowired
19      private SessionController sessionHandler;
20
21      @Autowired
22      private LemmaServiceProvider lemmaServiceProvider;
23
```

Εικόνα 5: Παράδειγμα με χρήση του annotation 'Autowired'

Insert_caption

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
        http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

    <import resource="classpath*:linguistics_controllers_spring.xml"/>

    <!-- APIs -->
    <bean id="userApi" class="dictionary.services.apis.UserManagement"/>

    <bean id="lemmaService" class="dictionary.services.apis.LemmaService"/>   Botsis, 19/12/2021 22:01 • Initial Lemma API added

    </beans>
```

Εικόνα 6: XML αρχείο παραμετροποίησης της Spring

Στην τελευταία εικόνα φαίνεται πως είναι ένα αρχείο παραμετροποίησης της Spring. Διακρίνεται και η καταχώρηση ‘import’ με την οποία φορτώνεται σε αυτό το project το αρχείο παραμετροποίησης της Spring από ένα άλλο project (για να επιτευχθεί αυτό τα projects πρέπει να συσχετίζονται μέσω maven).

Εικόνα 7: Παραμετροποίηση δομής Hibernate για διάλεκτο συμβατή με την PostgreSQL

1.3.4 Java Swing (GUI Development)

Το Java Swing είναι ένα εργαλείο για την ανάπτυξη γραφικού περιβάλλοντος χρήστη (Graphical User Interface, GUI) το οποίο περιλαμβάνει τα components του GUI. Το Swing είναι μέρος του Java Foundation Classes (JFC), το οποίο είναι ένα γραφικό πλαίσιο για την ανάπτυξη φορητών GUI, βασισμένα σε Java.

1.3.5 Apache Maven

Το Apache Maven είναι ένα εργαλείο διαχείρισης και κατανόησης ενός έργου. Έχει ως στόχο να κάνει την διαδικασία χτισίματος μιας εφαρμογής ευκολότερη, παρέχει ένα ενοποιημένο περιβάλλον χτισίματος και παρέχει και οδηγίες για άρτια λειτουργία (best practices guidelines). Τα έργα βασίζονται πάνω στα αρχεία POM (project object model) μέσω τον οποίων καθορίζονται τα χαρακτηριστικά του έργου (“Maven – Introduction,” n.d.; “Maven – Welcome to Apache Maven,” n.d.) όπως, μοναδικό όνομα (artifact id), γκρουπ στο οποίο ανήκει, «εξαρτήσεις» dependencies, ιδιότητες (properties), παραμετροποίηση χτισίματος εφαρμογής (build), επεκτάσεις (plug-ins) κ.α. Οι εξαρτήσεις είναι ένα σημαντικό κομμάτι στα έργα maven, καθώς αφήνουν τον χρήστη να υλοποιήσει βιβλιοθήκες υλοποιημένες από την Apache, από τρίτους ή ακόμα και υλοποιήσεις του ίδιου του χρήστη. Έτσι, επιτρέπει στον χρήστη να έχει μια πολυεπίπεδη εφαρμογή, βοηθάει στην αποφυγή δημιουργίας κώδικα που επαναλαμβάνεται. Μέσα από την διασύνδεση των βιβλιοθηκών γίνεται και η διασύνδεση ενός έργου με δομές και εργαλεία τρίτων όπως είναι η δομή Spring, η βιβλιοθήκη GSON κ.α. Όταν εισάγεται ένα έργο σαν διασύνδεση, εισάγονται και τα οι βιβλιοθήκες με τις οποίες είναι διασυνδεδεμένο το εισαγόμενο έργο.

1.3.6 BitBucket

Για τον έλεγχο και την ανάπτυξη ενός προγραμματιστικού έργου, στο οποίο εργάζονται πολλά άτομα και σε διαφορετικά κομμάτια, είναι απαραίτητο να υπάρχει ένα λογισμικό το οποίο θα προσφέρει τη δυνατότητα να παρακολουθεί (track) τις αλλαγές που γίνονται από το κάθε άτομο και να μπορεί να ενσωματώνονται στο έργο, χωρίς να δημιουργούνται προβλήματα στο κομμάτι εργασίας ενός άλλου ατόμου. Αυτά τα προγράμματα ονομάζονται συστήματα ελέγχου και αναθεώρησης. Τα πιο γνωστά και ευρέως χρησιμοποιούμενα είναι το git και το Bitbucket.

Το git είναι ένα δωρεάν και ανοιχτού κώδικα σύστημα ελέγχου, που εμφανίστηκε πρώτη φορά το 2005 όταν πολλοί προγραμματιστές του Linux kernel δεν είχανε πρόσβαση στο BitKeeper, ένα σύστημα ελέγχου το οποίο χρησιμοποιούσαν για την ανάπτυξη του έργου. Το git είναι φτιαγμένο για τη διαχείριση μικρών έως πολύ μεγάλων έργων με ταχύτητα και μεγάλη απόδοση. (“Git,” n.d.)

Το Bitbucket είναι ένα σύστημα ελέγχου και αναθεώρησης για την ανάπτυξη λογισμικού. Είναι γραμμένο σε γλώσσα προγραμματισμού Python.

Η χρήση του αποσκοπεί στον έλεγχο και στην επισήμανση των αλλαγών που γίνονται στον πηγαίο κώδικα που αναπτύσσεται, συνήθως σε μεγάλα έργα, προκειμένου να διασφαλίζεται η συνέχεια και η παρακολούθηση των αλλαγών.

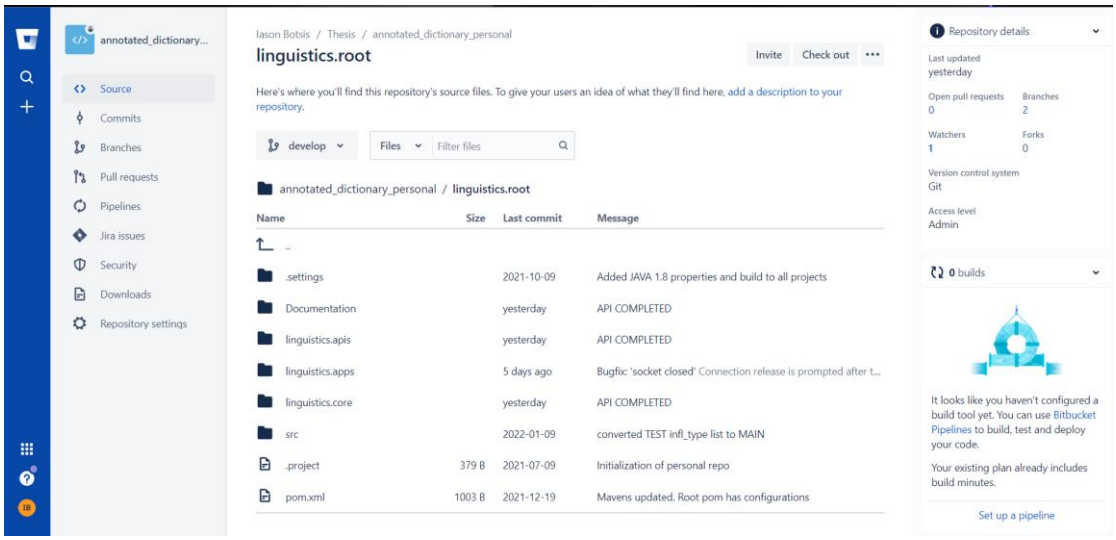
Η αρχή λειτουργίας του βασίζεται σε απομακρυσμένους αποθηκευτικούς χώρους (repository), οι οποίοι χώροι βρίσκονται στο cloud. Στους χώρους αυτούς, γίνεται η εισαγωγή όλων των αρχείων του κώδικα πάνω στον οποίο μια ομάδα εργάζεται. Από εκεί κλωνοποιούνται σε ένα κοινόχρηστο σύστημα αρχείων, όπου έχουν πρόσβαση όλοι οι εμπλεκόμενοι.

Μπορούν να δημιουργηθούν κλαδιά (branches), όπου κάθε εμπλεκόμενος μπορεί να εργάζεται χωρίς να επηρεάζει την εργασία των άλλων χρηστών. Τέλος με την εντολή «commit», εισάγεται η αλλαγή στον πηγαίο κώδικα και καταγράφεται η αλλαγή και το ιστορικό. Η αλλαγή αυτή εισάγεται στο τοπικό αντίγραφο του κλαδιού που δουλεύει ο χρήστης. Με την εντολή «push», οι αλλαγές οι οποίες έχουν γίνει «commit» αποστέλλονται στο cloud όπου βρίσκεται η remote έκδοση του κλαδιού. Κατόπιν, ενημερώνονται οι άλλοι χρήστες για τις αλλαγές στον κώδικα, ωστόσο για να

τις δουν και να τις έχουν διαθέσιμες πρέπει σε τακτά χρονικά διαστήματα να χρησιμοποιούν την εντολή «pull».

Όταν ο χρήστης ενημερώσει το αρχείο του, οι αλλαγές και οι διαγραφές φαίνονται με κόκκινο χρώμα, ενώ οι προσθήκες φαίνονται με πράσινο χρώμα. και πέρα, δίνεται πρόσβαση στους χρήστες στον κώδικα. Η πρόσβαση μπορεί να ελεγχθεί, έτσι ώστε ο κάθε χρήστης να έχει πρόσβαση σε ένα συγκεκριμένο σημείο του κώδικα. (Atlassian, n.d.)

Οι χρήστες από εκεί, κατεβάζουν το αρχείο στον υπολογιστή τους (clone) και αρχίζουν την επεξεργασία του.



Εικόνα 8: Εικόνα από τον ιστότοπο του Bitbucket, όπου βλέπουμε τον πηγαίο κώδικα ενός project και συγκεκριμένα το κλαδί develop.

1.4 Εφαρμογές REST

REST καλείται ένα σύνολο από αρχιτεκτονικούς κανόνες/αρχές μέσω των οποίων είναι δυνατή η σχεδίαση υπηρεσιών ιστού (Web Services). Το αρχιτεκτονικό αυτό μοντέλο σχεδίασης υπηρεσιών, παρόλο που εμφανίστηκε αρχικά το 2000 τα τελευταία χρόνεια έχει γίνει το κυρίαρχο πρότυπο σχεδίασης, ξεπερνώντας τα πρωτόκολλα διεπαφών SOAP και WSDL λόγω της απλούστερης εφαρμογής τους.

Οι υπηρεσίες ιστού (web services) εν γένει αποτελούν αυτόνομες εφαρμογές, που μπορούν να δημοσιευθούν, εντοπισθούν και να κληθούν μέσω του ιστού με τη χρήση

αποδεκτών προτύπων, όπως είναι τα πρότυπα XML, HTTP και SOAP. Με βάση τον ορισμό της Microsoft, οι υπηρεσίες ιστού αποτελούνται από μια εφαρμογή η οποία παρέχει δεδομένα ή προσφέρει μια υπηρεσία για τη λειτουργία κάποιας άλλης εφαρμογής, μέσω των διαδικτυακών πρωτοκόλλων επικοινωνίας που αναφέρθηκαν παραπάνω. (Graham, n.d.)

Οι βασικές αρχές σχεδιασμού μιας εφαρμογής που βασίζεται στο αρχιτεκτονικό μοντέλο REST είναι η αποκλειστική χρήση HTTP μεθόδων, η υλοποίηση χωρίς αποθήκευση της κατάστασης, η διευθυνσιοδότητη η οποία είναι ίδια με τη δομή των φακέλων και τέλος η μεταφορά XML αρχείων ή JavaScript Object Notation (JSON) αρχείων ή και των δύο

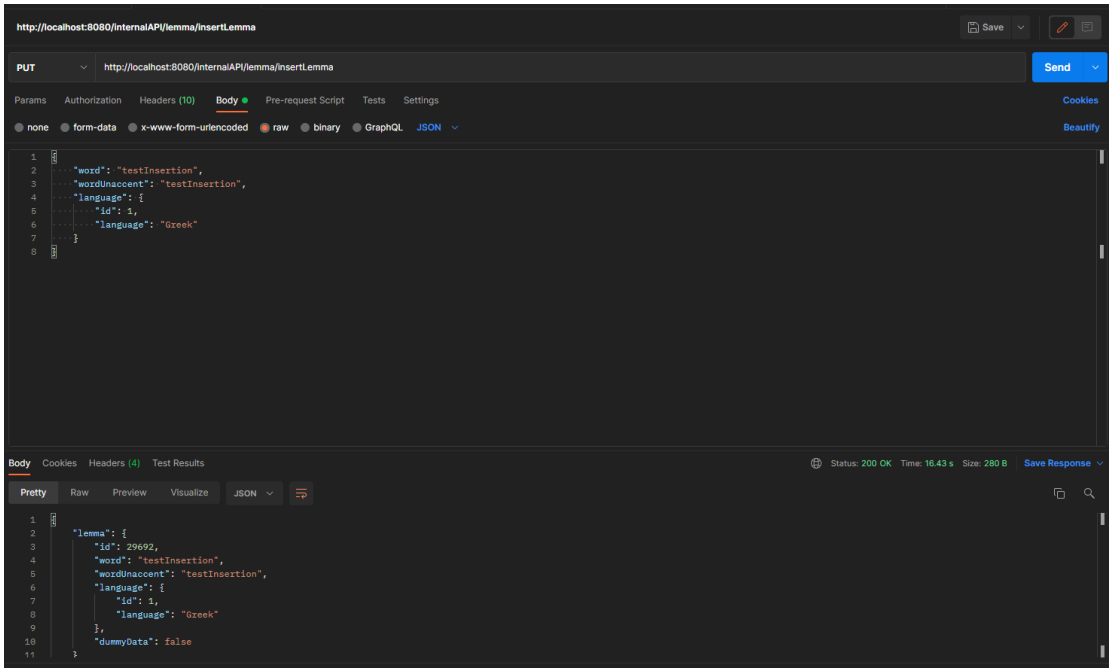
1.4.1 Τι είναι το API

Η διεπαφή προγραμματισμού εφαρμογών, Application programming interface (API) είναι ένα σύνολο από ορισμούς και πρωτόκολλα μέσω των οποίων είναι δυνατή η διασύνδεση των διαφόρων συστημάτων με άλλα συστήματα. Ουσιαστικά είναι η διεπαφή που χρησιμοποιείται από μια εφαρμογή για να επικοινωνήσει με μια άλλη εφαρμογή και να ανταλλάξει πληροφορίες, με ασφάλεια και έλεγχο στο ποιος έχει πρόσβαση στη πληροφορία. (Κατσαρός, 2012)

1.4.2 Postman

Για την ανάπτυξη και έλεγχο μιας διεπαφής είναι δυνατόν να χρησιμοποιηθεί το εργαλείο “Postman”, με σκοπό την βελτίωση της διεπαφής στο να δίνονται γρηγορότερες απαντήσεις στις αιτήσεις HTTP, να αποφεύγονται λάθη και να διασφαλίζεται ο κύκλος ζωής της διεπαφής. Το εργαλείο αυτό, προσφέρεται είτε σαν λογισμικό που μπορεί να εγκατασταθεί στον υπολογιστή είτε σαν εφαρμογή που μπορεί να συνδεθεί κάποιος μέσω του λογαριασμού του.

Το πλεονέκτημα του εργαλείου αυτού είναι πως δύναται η δυνατότητα να ελεγχθεί το API μιας αναπτυσσόμενης εφαρμογής και να ελεγχθεί η λειτουργία της όσον αφορά το κομμάτι της επικοινωνίας της με τη βάση δεδομένων και με τον τελικό χρήστη.



Εικόνα 9: Παράδειγμα χρήσης postman

1.4.3 HTTP REQUESTS

Τα βασικά στοιχεία ενός αιτήματος HTTP αιτήματος είναι τα εξής:

- **Μέθοδος** (π.χ. GET, PATCH, DELETE, HEAD κ.α.)
- **URL link:** το οποίο είναι ένα μοναδικό λινκ/διεύθυνση στον παγκόσμιο ιστό το οποίο καθορίζει την διεύθυνση στην οποία αποστέλλεται το αίτημα.
- **Παράμετροι του URL:** είναι παράμετροι οι οποίες προστίθενται στο τέλος του λινκ με το αναγνωριστικό '?'. Μπορούν να προστεθούν περισσότερες από μία παράμετροι με την χρήση του αναγνωριστικού '&'. (π.χ. http://www.domain.com/adomain?parameter1=value1&parameter2=value)
- **Επικεφαλίδες (headers):** παρόμοια με τις παραμέτρους αποτελούνται από ένα ζευγάρι (μεταβλητή, τιμή). Η διαφορά είναι ότι οι επικεφαλίδες δεν μπαίνουν στο τέλος του λινκ και αποστέλλονται μαζί με το σώμα. Μάλιστα στις περιπτώσεις που έχουμε HTTP αίτημα με κρυπτογράφηση, δηλαδή HTTPS, οι επικεφαλίδες μαζί με το σώμα κρυπτογραφούνται.

- **Σώμα (body):** στο σώμα παρέχεται η κύρια πληροφορία που μας ενδιαφέρει να επικοινωνήσουμε με ένα HTTP αίτημα. Μπορεί να περιέχει πολλά ήδη δεδομένων όπως json, text, html, xml κ.α.

Αναλυτικότερα οι μέθοδοι που θα συναντήσουμε συνήθως είναι οι εξής:

- **'GET':** χρησιμοποιείται για αιτήματα τα οποία θα μας φέρουν δεδομένα.
- **'POST':** χρησιμοποιείται για αιτήματα στα οποία αποστέλλουμε δεδομένα για επεξεργασία, ενδεχομένως και αποθήκευση και περιμένουμε την επιστροφή αυτών ή κάποιου σχετικού με τα δεδομένα μηνύματος.
- **'PUT':** χρησιμοποιείται για αιτήματα στα οποία θέλουμε να αποστείλουμε δεδομένα προς αποθήκευση.
- '**DELETE':** χρησιμοποιούνται για αιτήματα τα οποία έχουν σκοπό την διαγραφή δεδομένων.

(Κατσαρός, 2012)

Για κάθε HTTP αίτημα, ο διακομιστής ο οποίος λαμβάνει το αίτημα απαντάει με μία HTTP απάντηση. Η απάντηση αυτή, εκτός από την πληροφορία που ενδεχομένως να περιέχει, περιέχει επίσης και ένα χαρακτηριστικό κωδικό της κατάστασης του αιτήματος. Αυτοί οι κωδικοί χρησιμοποιούνται ευρέως στον παγκόσμιο ιστό του διαδικτύου και έχουν συγκεκριμένη περιγραφή ο καθένας. Παραθέτουμε κάποιους ενδεικτικά:

- **200 – OK:** Χρησιμοποιείται όταν το η HTTP επικοινωνία έχει γίνει με επικοινωνία και δεν συνάντησε κάποιο πρόβλημα.
- **400 – Bad Request:** Χρησιμοποιείται όταν ο διακομιστής δεν μπορεί να διαχειριστεί το αίτημα λόγω προβλήματος με το συντακτικό.
- **401 – Unauthorized:** Χρησιμοποιείται συνήθως όταν το HTTP αίτημα φέρει αναγνωριστικά ταυτότητας, το αίτημα ταυτοποιείται αλλά δεν του δίνεται πρόσβαση στην υπηρεσία στην οποία στοχεύει.
- **403 – Forbidden:** Χρησιμοποιείται όταν το HTTP δεν ταυτοποιείται από τον διακομιστή και "απαγορεύεται η πρόσβαση".
- **404 – Not Found:** Χρησιμοποιείται όταν ο διακομιστής δεν αναγνωρίζει το URL που φέρει το αίτημα και δεν μπορεί να το εξυπηρετήσει.

- **500 – Internal Server Error:** Χρησιμοποιείται όταν ο διακομιστής έρχεται «αντιμέτωπο» με μια κατάσταση που δεν μπορεί να διαχειριστεί.

(“HTTP response status codes - HTTP | MDN,” n.d.)

1.5 JSON files και χρήση της βιβλιοθήκης GSON

Τα αρχεία τύπου JSON (JavaScript Object Notation) είναι αρχεία στα οποία αποθηκεύονται απλά δεδομένα και αντικείμενα γραμμένα σε JSON, το οποίο είναι ένα ελαφρύ πρότυπο ανταλλαγής δεδομένων, το οποίο είναι πολύ εύχρηστο για τους χρήστες να διαβάσουν και να γράψουν αυτά τα αρχεία.

Το JSON είναι ένα πρότυπο κειμένου βασισμένο στη γλώσσα προγραμματισμού JavaScript και βρίσκει εφαρμογές σχεδόν σε όλες τις γλώσσες προγραμματισμού για ανταλλαγή δεδομένων. https://www.json.org/json-el.html

Η δομή του βασίζεται σε δύο πυλώνες:

- Μια συλλογή από ζευγάρια – τιμές.
- Μια ταξινομημένη λίστα τιμών.

Τα πλεονεκτήματα αυτού του τύπου αρχείου, είναι πως πρόκειται για μια πιο απλή και εύχρηστη αντικατάσταση των αρχείων τύπου XML, είναι γρήγορο και ελαφρύ και υποστηρίζεται από όλες τις γλώσσες προγραμματισμού.

Η χρήση του προορίζεται κυρίως για ανταλλαγή δεδομένων μεταξύ ενός εξυπηρετητή (server) και κάποιας διαδικτυακής εφαρμογής (web application).


```
{"cvForm":"-cvv-cv-cv-","graphemePhoneme":["κ-k","ου-u","ρ-r","ά-ˈa","ζ-z","ω-
o"],"name":"κουράζω","numberOfCharacters":7,"numberOfPhonemes":6,
"numberOfSyllables":3,"partOfSpeech":"verb","phonetic":"/kurˈazo/","syllables":
["κου","ρά","ζω"],"additionalInfo":{"wordFrequency":246},
"morphologicalInfo":{"suffix":"ω","suffixType":"visual","derivationalInfo":[{"suffix":"άζω"},
{"suffix":"άζω"}]}}
```


Εικόνα 10: Παράδειγμα από ένα JSON αρχείο

Για τη κωδικοποίηση (serialization) και αποκωδικοποίηση (deserialization) χρησιμοποιείται η βιβλιοθήκη Gson της Google. Βασίζεται στην Java και σκοπός είναι η κωδικοποίηση Java αντικειμένων (Java Objects) σε JSON αρχεία και η

αποδικωποίηση αυτών των αρχείων σε Java αντικείμενα. Τα πλεονεκτήματα αυτής της βιβλιοθήκης είναι ότι εύκολη στη χρήση, είναι ανοιχτού κώδικα, επομένως όλοι έχουν πρόσβαση σε αυτή. Διαχειρίζεται από την Google, είναι γρήγορη, δεν έχει μεγάλες απαιτήσεις σε μνήμη και για τη χρήση της δεν χρειάζεται κάποια άλλη βιβλιοθήκη πέρα από την JDK. Επίσης μπορεί να κωδικοποιήσει και να αποκωδικοποιήσει ένα πολύ μεγάλο όγκο αντικειμένων (“Gson User Guide - gson,” n.d.)

Η βιβλιοθήκη αυτή διαχειρίζεται τα JSON αρχεία ως εξής:

- **Streaming API:** Διαβάζει και γράφει περιεχόμενο JSON, σαν ξεχωριστά γεγονότα με τις εντολές JsonReader και JsonWriter
- **Tree model:** Δημιουργεί μια απεικόνιση του JSON αρχείου υπό τη μορφή δένδρου.
- **Data Binding:** Μετατρέπει τα αρχεία JSON σε Plain Old Java Object (POJO) κάνοντας χρήση του property accessor.

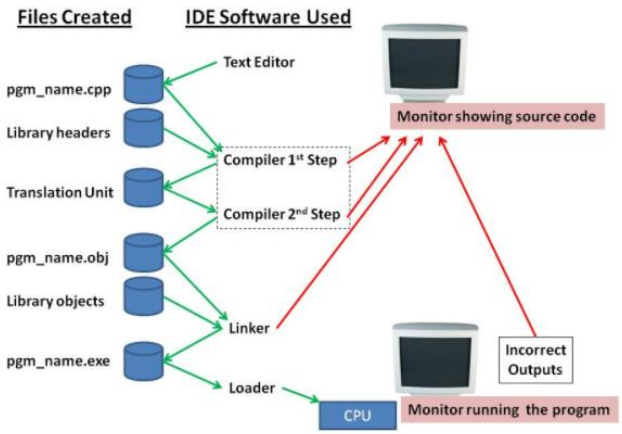
1.6 IDEs

Για την ανάπτυξη ενός κώδικα είναι απαραίτητη η ύπαρξη ενός λογισμικού στο οποίο ο χρήστης θα γράψει τον κώδικα στη γλώσσα προγραμματισμού και αυτό το κείμενο θα μετατραπεί στη γλώσσα που καταλαβαίνει ο υπολογιστής. Συγκεκριμένα, αυτό επιτυγχάνεται από τα λογισμικά που ονομάζονται «Ολοκληρωμένα Περιβάλλοντα ανάπτυξης», «Integrated Development Environment”, (IDE).

Τα λογισμικά αυτά αποτελούνται από τα εξής:

- Επεξεργαστής κειμένου για τον κώδικα, στον οποίο γίνεται η γραφή και η μορφοποίηση του κώδικα (editor)
- Η αποσφαλμάτωση του κώδικα (debugger)
- Μεταγλωττιστής (compiler)
- Διερμηνέας (interpreter)

Τα προγράμματα αυτά χαρακτηρίζονται ως ολοκληρωμένα, καθώς στο παρελθόν για την εγγραφή ενός κώδικα, θα έπρεπε να χρησιμοποιηθούν πολλά διαφορετικά λογισμικά για κάθε χρήση που αναφέρθηκε παραπάνω. (Busbee, 2018)



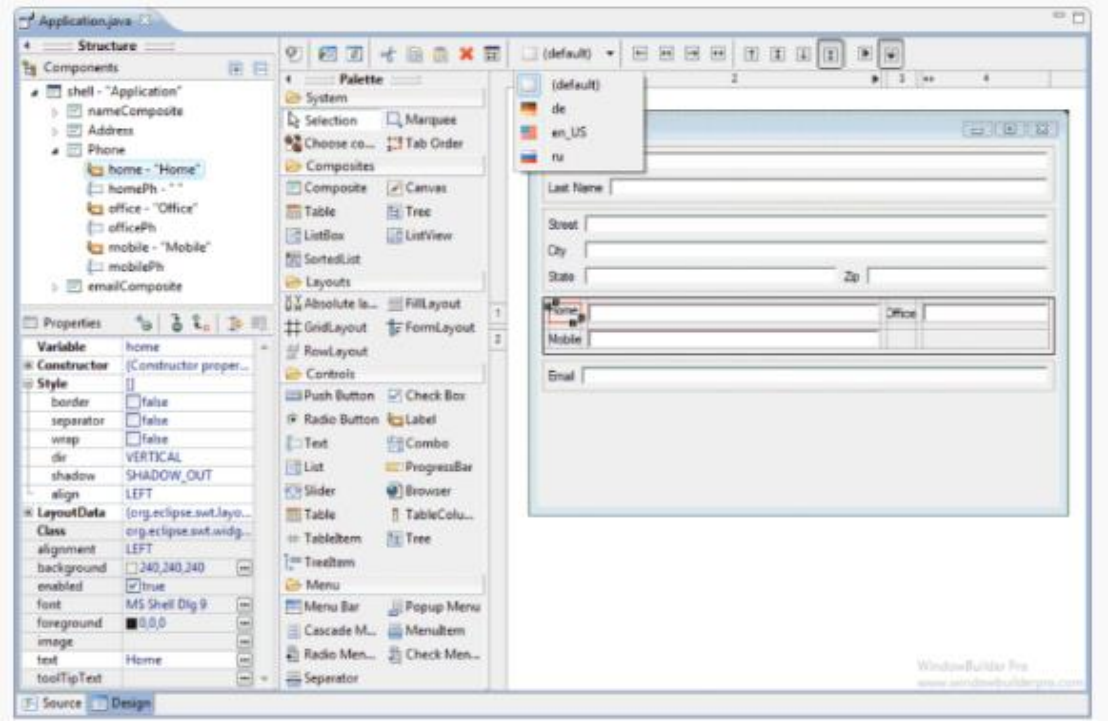
Εικόνα 11: Η λειτουργία ενός ολοκληρωμένου περιβάλλον ανάπτυξης. Πηγή: (Busbee, 2018)

Κάποια από τα πιο γνωστά λογισμικά που χρησιμοποιούνται για την δημιουργία κώδικα είναι το Visual Studio, το Eclipse, το PyCharm και το IntelliJ. (“TOP IDE Top Integrated Development Environment index,” n.d.)

1.6.1 Eclipse

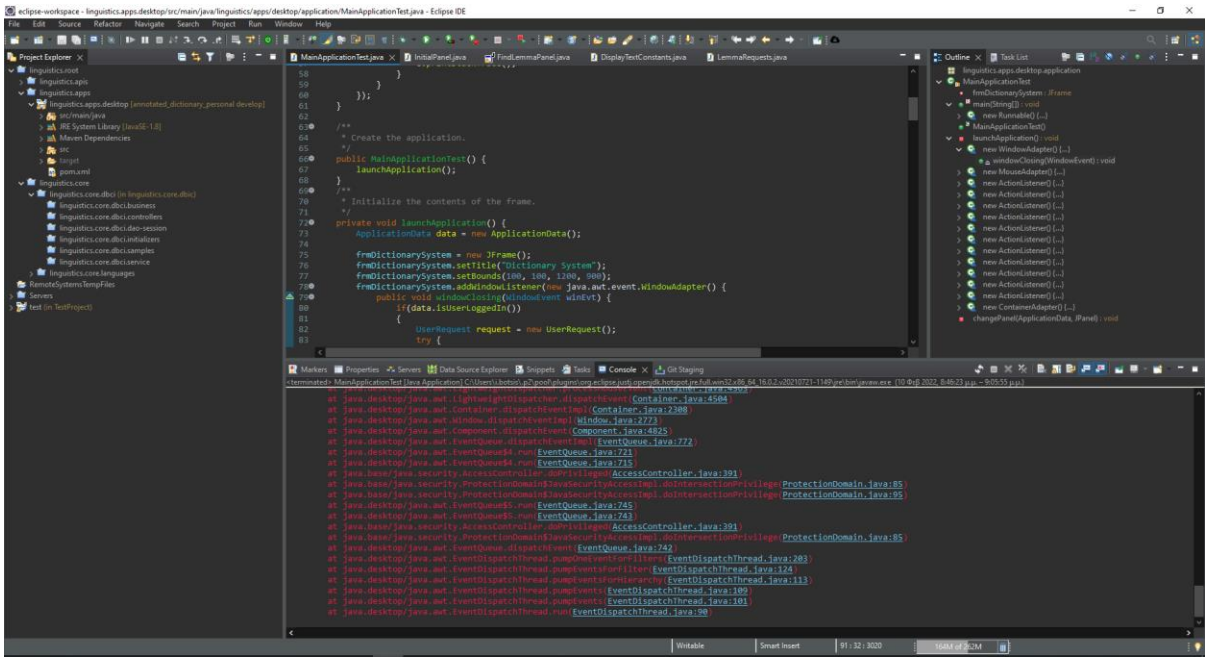
Το λογισμικό Eclipse, γραμμένο σε γλώσσα Java, έχει μεγάλη απήχηση κυρίως στους χρήστες που θέλουν να γράψουν σε Java τον κώδικά τους. (Guindon, n.d.)

Προσφέρει ένα εύχρηστο γραφικό περιβάλλον για την ανάπτυξη κώδικα, καθώς μπορεί να δεχτεί επιπλέον λειτουργίες (plug-ins). Παρόλο που είναι γραμμένο σε Java και χρησιμοποιείται κυρίως για την ανάπτυξη κώδικα σε Java, πολλοί προγραμματιστές το χρησιμοποιούν για την ανάπτυξη κώδικα και σε άλλες γλώσσες προγραμματισμού. Εκτός από την προσθήκη νέων λειτουργιών, δίνεται και η δυνατότητα για εύκολη ανάπτυξη εφαρμογών γραφικού περιβάλλοντος χρήστη (Graphical User Interface, GUI), όπως το WindowBuilder, που είναι πρόσθετη λειτουργία που μπορεί να εγκατασταθεί στο λογισμικό (plug-in). (Wren, n.d.)



Εικόνα 12: Περιβάλλον του WindowBuilder, της πρόσθετης λειτουργίας του ολοκληρωμένου περιβάλλοντος ανάπτυξης κώδικα Eclipse.

Τέλος, το λογισμικό είναι συνδεδεμένο αυτόματα με την πλατφόρμα Maven με αποτέλεσμα να είναι πολύ εύκολη η διαχείριση οποιασδήποτε αλλαγής στον πηγαίο κώδικα και τις όποιες εξαρτήσεις υπάρχουν με άλλα πακέτα.

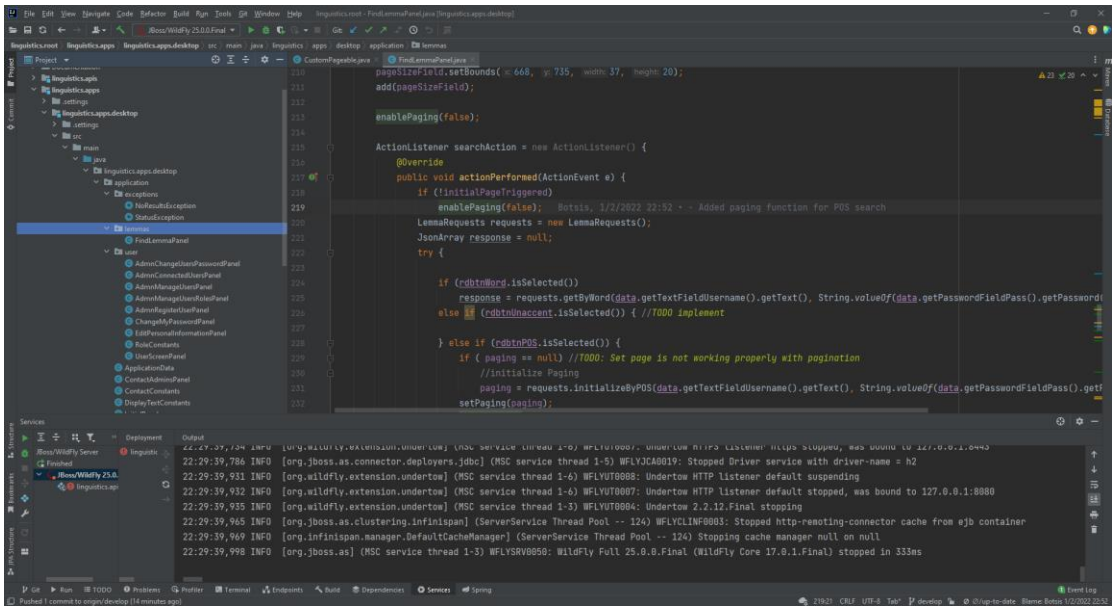


Εικόνα 13: Εικόνα από το περιβάλλον του Eclipse. Αριστερά: Βρίσκουμε την εξερεύνηση έργου η οποία είναι σε δενδρική δομή. Πάνω: είναι το μενού με τις διαφορετικές επιλογές του εργαλείου. Κέντρο πάνω: εκεί διαμορφώνεται ο κώδικας (editor). Δεξιά: σημείο στο οποίο έχουμε μία εποπτεία για τις μεθόδους και τις μεταβλητές της κλάσης που είναι ανοικτή στον editor. Κάτω: είναι το παράθυρο της κονσόλας στο οποίο εκτυπώνονται σχετικά μηνύματα αλλά και από εκεί χρησιμοποιούμε τον debugger.

1.6.2 IntelliJ

Αποτελεί ένα ολοκληρωμένο περιβάλλον ανάπτυξης γραμμένο σε Java, με εφαρμογή σε κώδικες που γράφονται σε Java. Έχει αναπτυχθεί από την JetBrains και διατίθεται ως λογισμικό ανοιχτού κώδικα, αλλά υπάρχει και επιλογή για επί πληρωμή, στο οποίο η κύρια διαφορά του με το δωρεάν είναι ότι υποστηρίζει και τις άλλες γλώσσες προγραμματισμού, χωρίς κάποια άλλη αλλαγή στο περιβάλλον του.

Τα βασικά πλεονεκτήματα του IntelliJ, είναι το Smart Code Completion, δηλαδή η εμφάνιση προτεινόμενων προτάσεων για την εγγραφή του κώδικα καθώς επίσης μπορεί και να αναγνωρίσει τυχόν λάθη που μπορεί να προκύπτουν κατά την εγγραφή, το Framework – specific assistance, το οποίο αναγνωρίζει και προσφέρει βοήθεια στην αναγνώριση πολλών άλλων γλωσσών προγραμματισμού, όπως η SQL. Επίσης, όπως και το Eclipse, και αυτό προσφέρει την προσθήκη επιπλέον δυνατοτήτων, υπό τη μορφή των plug-ins. (“Features - IntelliJ IDEA,” n.d.)

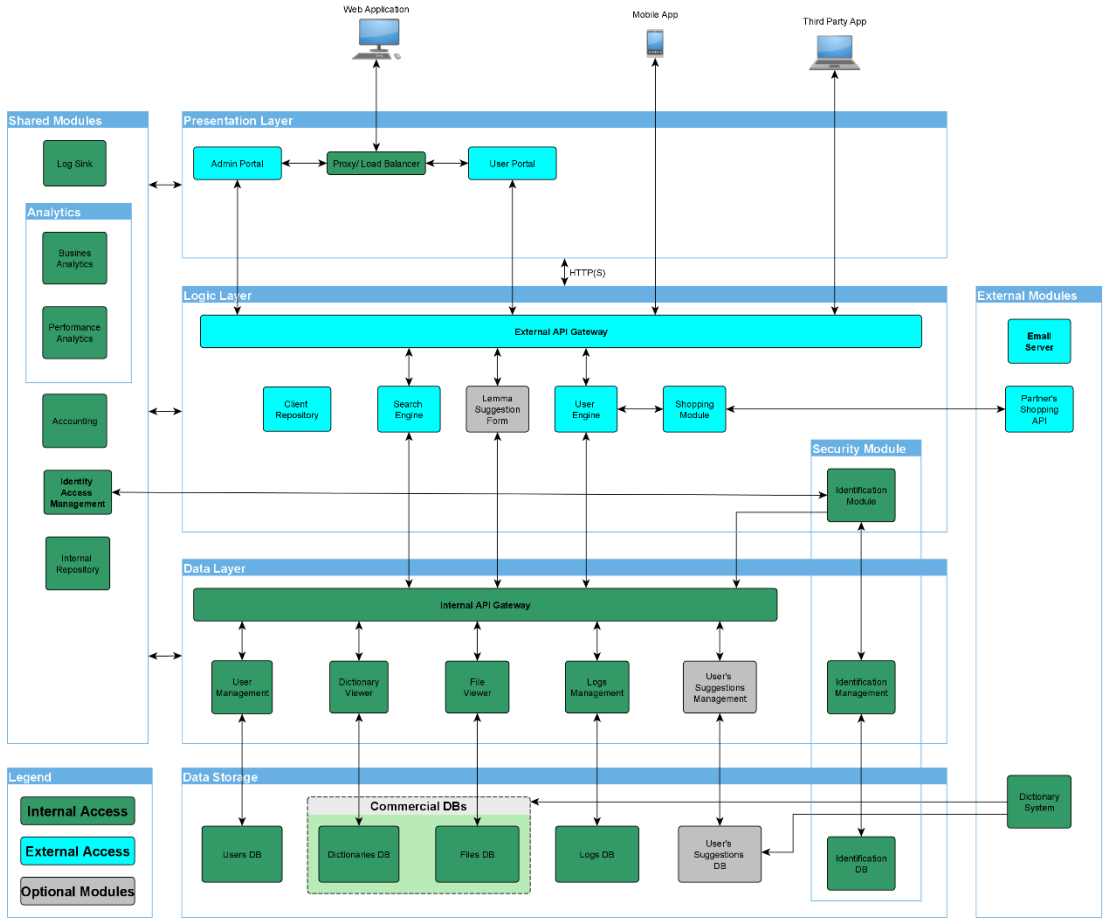


Εικόνα 14: Εικόνα από το περιβάλλον ανάπτυξης κώδικα του IntelliJ. Αριστερά: περιηγητής έργου, Κέντρο: editor, Πάνω: μενού επιλογών και κουμπιά άμεσης πρόσβασης, Κάτω: Κονσόλα μηνυμάτων

2 Η Πλατφόρμα

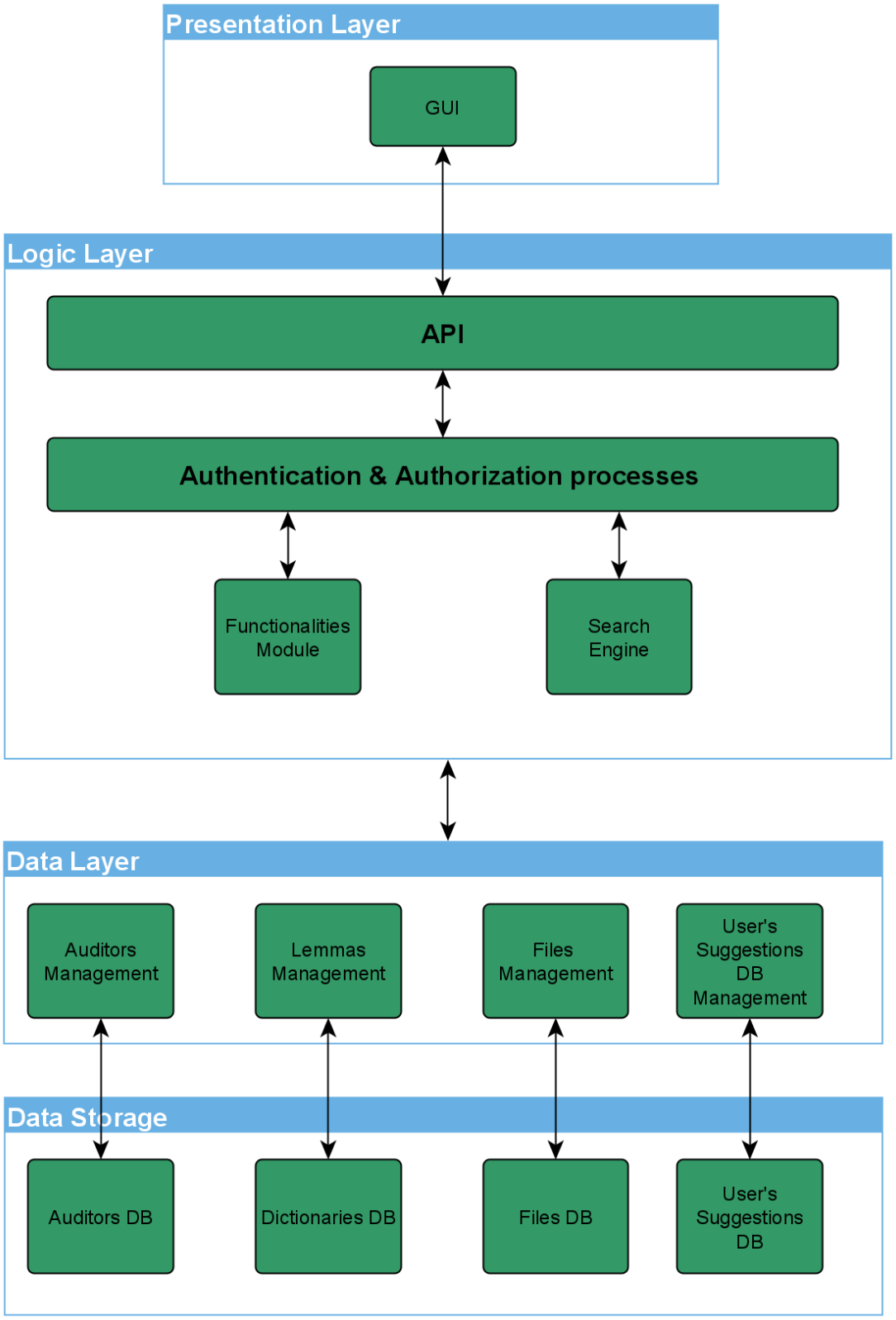
2.1 Γενική Ιδέα

Η γενική ιδέα είναι ότι υπάρχει ανάγκη στην αγορά για ένα Ελληνικό λεξικό το οποίο δεν θα παρέχει μόνο τις βασικές πληροφορίες (ορισμούς, φωνητική πληροφορία, συνώνυμα, αντώνυμα κ.α. Υπάρχει ανάγκη για ένα σύστημα το οποίο θα προσφέρει στον χρήστη, προηγμένες λειτουργίες αναζήτησης, εικόνες, ηχητικό με το οποίο ακούγεται το πως προφέρεται η εικόνα, κατηγοριοποίηση η οποία θα βοηθάει τον χρήστη να βλέπει σύνολα λέξεων κ.α. Ένα τέτοιο οικοσύστημα για να είναι πλήρως λειτουργικό έχει απαιτητικό σχεδιασμό, η υλοποίηση του οποίου ξεφεύγει από τα πλαίσια και τους σκοπούς της διπλωματικής αυτής.



Εικόνα 15: Εικόνα του αρχικού σχεδιασμού

Όπως βλέπουμε και στην παραπάνω εικόνα (11) το σύστημα αποτελείται από πολλές οντότητες που συνδέονται μεταξύ τους. Η Συγκεκριμένη διπλωματική αφορά την υλοποίηση της οντότητας ‘Dictionary System’ η οποία είναι υπεύθυνη με το να τροφοδοτεί τη βάση του κύριου συστήματος με επικαιροποιημένα από γλωσσολόγους λήμματα και να διαχειρίζεται τα λεξικά που θα παρέχει το κύριο σύστημα στον τελικό χρήστη.



Εικόνα 16: Σχεδιασμός Εφαρμογής Λεξικού

Αναλυτικότερα στην παραπάνω εικόνα έχουμε ξεχωρίσεις τις εξής οντότητες:

Data Storage

Πρόκειται για το στρώμα που περιέχει τις βάσεις δεδομένων του συστήματος.

- **Auditors DB:** Είναι η βάση των χρηστών που έχουν πρόσβαση σε αυτή την εφαρμογή καθώς και οι ρόλοι που έχει ο κάθε χρήστης. Αυτοί οι χρήστες θα είναι υπεύθυνοι ώστε να ελέγξουν τα λήμματα που υπάρχουν στην βάση και να πιστοποιήσουν την πληροφορία που είναι καταχωρημένη για αυτά.
- **Dictionaries DB:** συμβολίζει την βάση δεδομένων που περιλαμβάνει τα λήμματα και τις σχετικές γύρω από αυτά πληροφορίες.
- **Files DB:** Συμβολίζει την βάση στην οποία θα αποθηκεύονται αρχεία. Η PostgreSQL που χρησιμοποιούμε μπορεί να αποθηκεύσει αρχείο δεδομένων που φτάνει μέχρι και τα 2GB.
- **User's Suggestions DB:** Έχουμε προβλέψει στο κύριο σύστημα μέσω φόρμας να μπορούν οι χρήστες να καταχωρούν προτάσεις σχετικά με τα λήμματα, κατηγοριοποιήσεις τους (Tags) αλλά και προτάσεις σχετικά με το σύστημα. Η βάση αυτή είναι η οντότητα που θα αποθηκεύει αυτές τις προτάσεις, ώστε σε δεύτερο χρόνο να τις αναθεωρήσει κάποιος διαχειριστής του ή κάποιος γλωσσολόγος.

Data Layer

Πρόκειται για το στρώμα που περιέχει τα DAOs (Data Access Objects) είναι τα αντικείμενα που αντιστοιχούν σε κάθε πίνακα της βάσης δεδομένων και μέσω αυτών μπορεί η εφαρμογή να έχει επικοινωνία με την βάση.

- **Auditors Management:** Είναι η οντότητα που μας παρέχει την επικοινωνία με την βάση των χρηστών και μας επιτρέπει να κάνουμε εισαγωγές, διαγραφές και τροποποιήσεις.
- **Lemmas Management:** Είναι η οντότητα που μας παρέχει την επικοινωνία με την βάση των λεξικών και μας επιτρέπει να κάνουμε εισαγωγές, διαγραφές και τροποποιήσεις.
- **Files Management:** Είναι η οντότητα που μας παρέχει την επικοινωνία με την βάση των αρχείων ήχου και εικόνας και μας επιτρέπει να κάνουμε εισαγωγές, διαγραφές και τροποποιήσεις.

- **User's Suggestions DB Management:** Είναι η οντότητα που μας παρέχει την επικοινωνία με την βάση των προτάσεων των χρηστών και μας επιτρέπει να κάνουμε εισαγωγές, διαγραφές και τροποποιήσεις.

Logic Layer

Σε αυτό το στρώμα εντάσσεται η λογική που υπάρχει στην εφαρμογή μας.

- **Search Engine:** Είναι η οντότητα μέσα στην οποία υλοποιούνται οι λειτουργίες αναζήτησης, όπως αναζήτησης λέξης, αναζήτηση με 'μέρος του λόγους κ.α.'
- **Functionalities Module:** Είναι η οντότητα που δίνει κάποιες σύνθετες λειτουργίες χρησιμοποιώντας μεθόδους από πιο κάτω στρώματα, π.χ. για να μην εκτελούμε queries στην βάση τα οποία θα φορτίσουν πολύ την βάση αλλά και την εφαρμογή, έχουμε υλοποιήσει μια σελιδοποίηση η οποία επάγεται στην οντότητα 'functionalities'.
- **Authentication and Authorization:** Για την ασφάλεια της εφαρμογής χρειάζεται έλεγχος πρόσβασης χρηστών. Για τον σκοπό αυτό δεν χρησιμοποιήθηκε έτοιμη δομή της Spring αλλά έχει γίνει χωριστή υλοποίηση και παριστάνεται σε αυτήν την οντότητα. Είναι υπεύθυνη να κάνει ταυτοποίηση τα στοιχεία που έχει αποστείλει ο χρήστης με το HTTP αίτημα του και δίνει άδεια (ή όχι) να συνεχίσει το αίτημα
- **API:** Όπως περιγράψαμε καις το θεωρητικό μέρος, αυτή είναι η οντότητα η οποία διαχειρίζεται την επικοινωνία. Λαμβάνει HTTP αιτήματα, τα δρομολογεί κατάλληλα και είναι αρμόδια να απαντήσει σχετικό μήνυμα στον αποστολέα.

Presentation Layer

Σε αυτό το στρώμα είναι οι τρόποι με τους οποίους η εφαρμογή αποκτά υπόσταση στον τελικό χρήστη. Αυτό μπορεί να γίνει είτε με την ιστοσελίδας, εφαρμογής στο κινητό ή τάμπλετ αλλά και εφαρμογής στον υπολογιστή. Στην διπλωματική αυτή έχει υλοποιηθεί εφαρμογή για τον υπολογιστή. Η υλοποίηση έχει γίνει στο πακέτο Swing της Java και παράγεται μια jar εφαρμογή την οποία μπορούν να τρέξουν όλα τα συστήματα που έχουν εγκατεστημένη την Java.

Στην βάση δεδομένων χρησιμοποιούμε την PostgreSQL και συγκεκριμένα postgres12. Έχει γίνει σχετική παραμετροποίηση της δομής Hibernate στην Spring ώστε να μπορεί να επικοινωνήσει με την βάση με επιτυχία. Στο περιβάλλον IntelliJ σηκώνουμε έναν JBoss διακομιστή στον οποίο εκτελείται και 'τρέχει' το API και η υλοποίηση της εφαρμογής. Για το οπτικό τμήμα της εφαρμογής παράγεται ένα εκτελέσιμο jar αρχείο της java το οποίο επικοινωνεί με τον τοπικό μας διακομιστή μέσω HTTP αιτημάτων για να έχουμε ανταλλαγή δεδομένων.

2.2 Σχεδιασμός – Δεδομένα

Ο σχεδιασμός των δεδομένων υποστηρίζει και λήμματα από την Ελληνική γλώσσα αλλά και από την Αγγλική. Συγκεκριμένα για κάθε λήμμα υποστηρίζονται από την βάση οι παρακάτω πληροφορίες:

1. **Name:** Είναι η ίδια η λέξη ορθογραφημένη
2. **Name Unaccent:** Πρόκειται για την λέξη χωρίς τονισμό
3. **Grapheme Phoneme:** Είναι η αντιστοιχία κάθε γράμματος της λέξης με φωνητικό χαρακτήρα
4. **Phonetic:** Είναι ο φωνητικός προσδιορισμός της λέξης για το πως προφέρεται
5. **Etymology:** Είναι η ετοιμολογία της λέξης
6. **Definitions:** Ορισμοί της λέξης
7. **Examples:** Παραδείγματα διευκρίνησης της λέξης, π.χ. ένα ποδήλατο μπορεί να είναι αγωνιστικό ή ακόμα και ποδήλατο θαλάσσης
8. **Phrases:** Παραδείγματα με χρήση της λέξης σε πρόταση
9. **Synonyms/Antonyms:** Συνώνυμα και αντώνυμα
10. **Syllabification:** Συλλαβισμός
11. **Prefix:** Πρόθεμα
12. **Suffix:** Κατάληξη
13. **CV-Form:** Είναι η ακολουθία γραμμάτων (φωνήεν ή σύμφωνο) χωρισμένη σε συλλαβές
14. **Thema:** Το θέμα της λέξης
15. **Number of Characters:** Αριθμός χαρακτήρων
16. **Number of Phonemes:** Αριθμός φωνήεντων
17. **Number of Syllables**: Αριθμός συλλαβών

18. **Part of Speech:** Μέρος του λόγου
19. **Inflectional Info:** Κλητική πληροφορία
20. **Conjugational Information:** Πληροφορίες για την κατάληξη της λέξης
21. **Stem:** Είναι μια άλλη λέξη/λήμμα την οποία έχει ως αναφορά
22. **Main Lemma:** Πληροφορία αν το λήμμα είναι κύριο
23. **Audio file:** Αρχείο ήχου
24. **Audio File Rights:** Άδεια χρήσης του ηχητικού αρχείου
25. **Image File:** Αρχείο εικόνας
26. **Image File Rights:** Άδεια χρήσης του αρχείου εικόνας
27. **Language:** Η γλώσσα στην οποία ανήκει η λέξη
28. **Tags:** Κατηγορίες στις οποίες ανήκει η λέξη
29. **Age Availability:** Ηλικία για την οποία προορίζεται η λέξη, π.χ. οι δύσκολες ή οι χαμηλής συχνότητας λέξεις δεν προτείνονται να τις μαθαίνουν τα παιδιά πριν από μια συγκεκριμένη ηλικία
30. **Frequency:** Πόσο συχνά εμφανίζεται μια λέξη
31. **Difficulty:** Επίπεδο δυσκολίας μίας λέξης
32. **Derivative Words:** Παράγωγες λέξεις
33. **Relative Words:** Λέξεις που προέρχονται από το ίδιο θέμα, συγγενικές λέξεις
34. **Diminutive:** Υποκοριστικό της λέξης
35. **Incremental:** Υπερθετικός βαθμός της λέξης
36. **Homograph:** Ομόγραφες λέξεις (found: βρήκα – found: ιδρύθηκε)
37. **Homophones:** Ομόφωνες (όμως-ώμος, run – ran)
38. **Source:** Η πηγή από την οποία αποκτήθηκε η λέξη
39. **Auditor Id:** Το αναγνωριστικό του χρήστη που είδε την λέξη και την επικύρωσε
40. **QA Status:** Είναι η κατάσταση στην οποία βρίσκεται το λήμμα σχετικά με την επικύρωση του.
41. **Last Audit:** Είναι η τελευταία ημερομηνία που έγινε επικύρωση κάποιου πεδίου του λήμματος.

Τα δεδομένα για την τροφοδότηση της βάσης προήλθαν από την πλατφόρμα ELG. Ήταν σε αρχείο JSON και είχαν την μορφή { “lemmas” : [{Lemma}, {Lemma}, …. {Lemma}] } και περιείχε περίπου 8400 λήμματα. (“ELG - Greek Annotated Dictionary with Morphological and Linguistic Features,” n.d.) Το λεξικό αυτό παρέχεται κάτω από την άδεια Creative Commons Attribution Share Alike 4.0 International (“Creative Commons Αναφορά-Παρόμοια Διανομή 4.0 — CC BY-SA 4.0,” n.d.)


```
{"cvForm":"-vv-cv-cv-cv-",
"featureInfo":{"30":[[0,2]],"74":[[2,3],[6,7]],"79":[[4,5]],"190":[[5,8]],"405":[[0,1],[5,6],[7,8]],"419":[[3,4]],"420":[[4,5]],"422":[[2,3],[6,7]]},
"graphemePhoneme":["αι-e","τ-t","ή-'i","μ-m","α-a","τ-t","α-a"],
"name":"αιτήματα",
"numberOfCharacters":8,
"numberOfPhonemes":7,
"numberOfSyllables":4,
"partOfSpeech":"noun",
"phonetic":"/et'imata/",
"syllables":["αι","τή","μα","τα"],
"additionalInfo":{"wordFrequency":218},
"morphologicalInfo":{"suffix":"ατα","suffixType":"visual"},
"inflectionalInfo":[
{"gender":"neutral","inflectionSuffix":"ατα","isMainLemma":false,"number":"plural","stem":"αίτημα","case":"accusative"},
{"gender":"neutral","inflectionSuffix":"α","isMainLemma":false,"number":"plural","stem":"αίτημα","case":"clitic"},
{"gender":"neutral","inflectionSuffix":"ατα","isMainLemma":false,"number":"plural","stem":"αίτημα","case":"nominative"}]},
```


Εικόνα 17: Λήμμα από το ELG

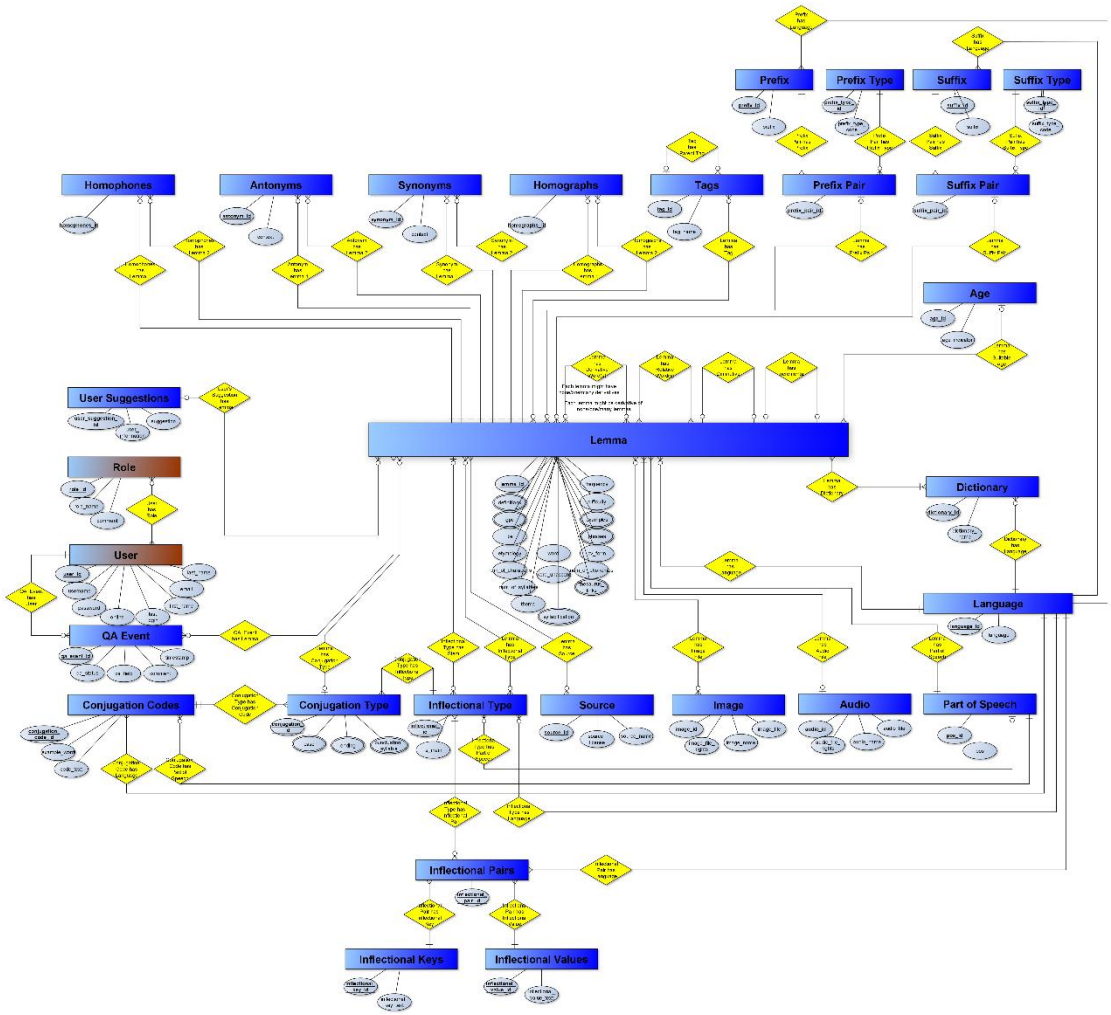
2.3 Βάση Δεδομένων

Η υλοποίηση αυτού του συστήματος έχει γίνει σε τοπικό περιβάλλον. Συγκεκριμένα έχει εγκατασταθεί η PostgreSQL και συγκεκριμένα χρήση της postgres12, στην οποία φορτώνεται ο σχεδιασμός μέσω sql αρχείων που σχεδιάστηκαν στο εργαλείο pg modeler.

2.3.1 Σχεδιασμός

Πριν τον σχεδιασμό και την υλοποίηση μιας βάσης σε χαμηλό επίπεδο πρέπει πρώτα να σχεδιαστεί το ERD μοντέλο της βάσης. Σε αυτό το μοντέλο οι οντότητες παρουσιάζονται ως ορθογώνια παραλληλόγραμμα, τα χαρακτηριστικά της κάθε οντότητας συμβολίζονται με ελλειπτικό σχήμα και οι συσχετίσεις μεταξύ των οντοτήτων συμβολίζονται με ρόμβους. Η βάση είναι κανονικοποιημένη ώστε να αποφεύγεται η επανάληψη δεδομένων. Μέσω των εξωτερικών αναφορών οποιαδήποτε αλλαγή γίνει σε μια οντότητα ενημερώνει αυτόματα τις οντότητες που έχουν εξωτερική αναφορά σε αυτή. Με αυτόν τον τρόπο τα δεδομένα μας μένουν ενημερωμένα. Μέσα από το διάγραμμα ERD φαίνονται οι λόγους πληθικότητας που έχουν οι συσχετίσεις μεταξύ των οντοτήτων.

- Όταν υπάρχει μια σχέση ‘πολλά προς ένα’ σημαίνει ότι πολλές καταχωρήσεις από αυτόν τον πίνακα αναφέρονται στην ίδια τιμή. Αυτή η τιμή καταχωρείται σε άλλον πίνακα και οι καταχωρήσεις αναφέρονται στην καταχώρηση αυτού του πίνακα. Για παράδειγμα, έστω ο πίνακας που είναι καταχωρημένα τα λήμματα, πολλά λήμματα είναι ουσιαστικά, επομένως δημιουργείται ένας πίνακας με τα μέρη του λόγου και καταχωρείται η τιμή ‘ουσιαστικό’. Όλα τα λήμματα πλέον που είναι ουσιαστικά καταχωρούνται και στο πεδίο ‘μέρος του λόγου’ υπάρχει εξωτερικό κλειδί (foreign key) που οδηγεί στο ουσιαστικό.
- Όταν υπάρχει σχέση ‘πολλά προς πολλά’ τότε πάλι το χαρακτηριστικό αποθηκεύεται σε έναν ξεχωριστό πίνακα, αλλά υπάρχει ένας ενδιάμεσος πίνακας (map) ο οποίος αντιστοιχεί μια καταγραφή από τον πρώτο πίνακα (π.χ. πίνακας λημμάτων) με μια γραμμή από τον πίνακα του χαρακτηριστικού (π.χ. πίνακας με κλητική πληροφορία) χρησιμοποιώντας τα κλειδιά τους.

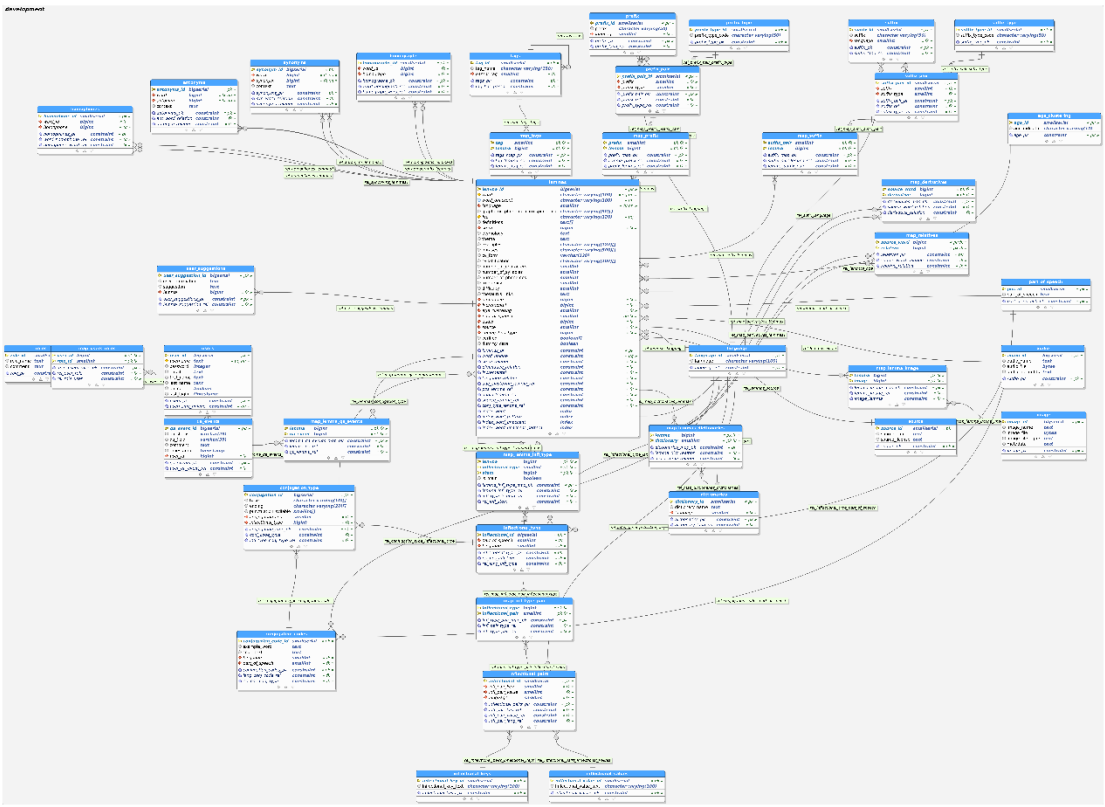


Αυτά τα δύο γνωρίσματα του ενδιάμεσου πίνακα είναι και το πρωτεύον κλειδί του, έτσι εξασφαλίζουμε και ότι κάθε συνδυασμός είναι μοναδικός.

Εικόνα 18: Το διάγραμμα ERD της βάσης δεομένων του συστήματος.

2.3.2 Υλοποίηση

Μετά τον ERD σχεδιασμό έχει σειρά ο σχεδιασμός υλοποίησης του. Για τον σχεδιασμό της υλοποίησης έχει χρησιμοποιηθεί το λογισμικό pgmodeler.

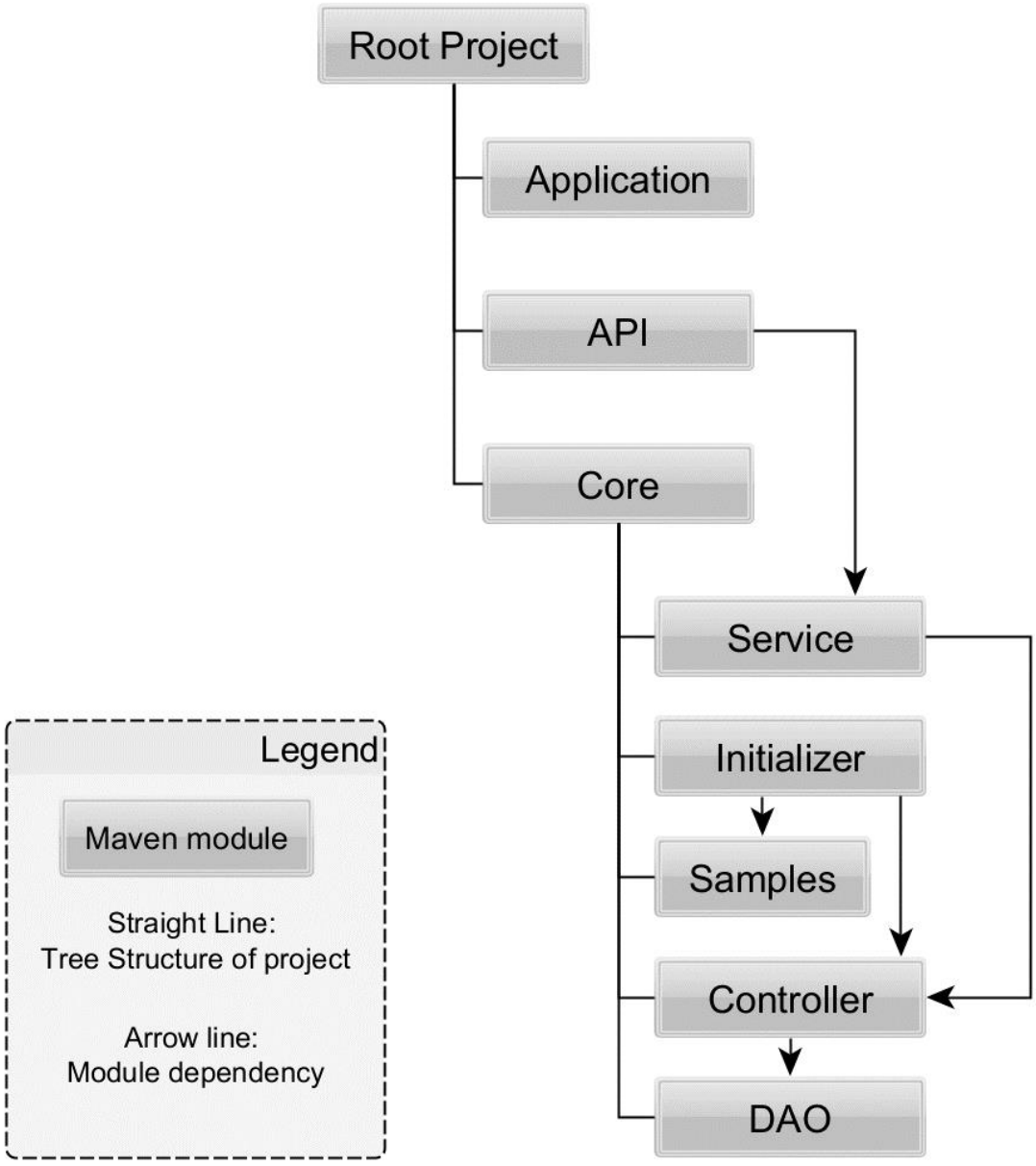


Εικόνα 19: Τελικός σχεδιασμός βάσης δεδομένων

Από τις εικόνες του ERD και του τελικού σχεδιασμού της βάσης δεδομένων φαίνεται η πολυπλοκότητα του συστήματος και ξεχωρίζει η κεντρική οντότητα του λήμματος. Η κυριότερη χρήση του συστήματος είναι η εύρεση λέξεων επομένως ήταν απαραίτητος ο ορισμός ευρετηρίων (indexes) στα πεδία word και wordUnaccent. Αξίζει να σημειωθεί εδώ ότι τα πεδία αυτά έχουν υλοποιηθεί με τύπο 'varchar' μεγέθους 100 τιμών. Τα ευρετήρια υλοποιούνται με B-δέντρα (B-trees) όπου και καλύπτουν τις βασικές πράξεις των < (μικρότερο), <= (μικρότερο ή ίσο), = (ίσο), >= (μεγαλύτερο ή ίσο) και > (μεγαλύτερο). Στην PostgreSQL η υλοποίηση αυτή γίνεται με την χρήση της ενσωματωμένης μεθόδου varchar_ops. Για την κάλυψη αναζήτησης με χρήση των τελεστών κανονικών εκφράσεων LIKE και POSIX έχει χρησιμοποιηθεί και η μέθοδος ευρετηρίου varchar_pattern_ops που παρέχει η PostgreSQL. Πρόκειται για σημαντική προσθήκη γιατί δίνει στην βάση την δυνατότητα να ανταπεξέλθει γρήγορα και αποδοτικά σε αναζητήσεις όπως, «λέξεις που περιέχουν -δηλα-», «λέξεις που ξεκινούν από κιτ-», «λέξεις που καταλήγουν σε -φορα» κ.α. Τα παραπάνω παραδείγματα αποτελούν τα πιο απλά των τελεστών LIKE και POSIX.

2.4 Backend

Το Backend κομμάτι μιας εφαρμογής περιλαμβάνει τις οντότητες που παρέχουν πρόσβαση στην βάση δεδομένων (DAOs) και τις οντότητες που παρέχουν την λογική στην εφαρμογή.



Εικόνα 20: Δενδρική δομή του έργου και οι συσχετίσεις μεταξύ των έργων

Στην παραπάνω εικόνα φαίνεται η δομή της εφαρμογής.
Κάτω από το Root Project διακρίνονται τρία έργα παιδιά (child projects).

- **Application:** Περιλαμβάνει την υλοποίηση του γραφικού περιβάλλοντος της εφαρμογής (frontend). Παρατηρείται στο διάγραμμα ότι η εφαρμογή δεν έχει κάποιο άλλο έργο σαν εξάρτηση. Αυτό συμβαίνει γιατί η εφαρμογή επικοινωνεί με το API με χρήση HTTP αιτημάτων, για τα οποία λαμβάνει απαντήσεις τις οποίες μετά από επεξεργασία προβάλει στον χρήστη.
- **CORE:** Αποτελεί τον κύριο πυρήνα υλοποίησης του συστήματος περιέχει διαφορετικά υπο-έργα τα οποία παρέχουν λειτουργικότητα στο σύστημα.
    - **DAO:** Είναι η οντότητα η οποία παρέχει την πρόσβαση στα δεδομένα της βάσης. Εκεί χρησιμοποιούνται ειδικά queries της hibernate για την υλοποίηση των αναγκών της εφαρμογής μας. Συγκεκριμένα αυτή η οντότητα χωρίζεται σε τρεις υποκατηγορίες,
        - **Οντότητες:** Πρόκειται για την αναπαράσταση των πινάκων της βάσης δεδομένων σε αντικείμενα τις java. Η Hibernate αναλαμβάνει την αντιστοίχιση αυτή. Η μη σωστή παραμετροποίηση των οντοτήτων θα έχει ως αποτέλεσμα σε σφάλματα. Αξίζει να σημειωθεί ότι για τους πίνακες που χρησιμοποιούνται για την υλοποίηση της σχέσης ‘πολλά προς πολλά’ χρειάζεται να οριστεί με κατάλληλο τρόπο οντότητα της Java που υλοποιεί το κλειδί αυτού του πίνακα.
        - **Διαπροσωπίες:** Πρόκειται για τις κλάσεις μέσω των οποίων παρέχεται η πρόσβαση στα δεδομένα της βάσης.
        - **Υλοποιήσεις διαπροσωπιών:** Πρόκειται για την υλοποίηση των διαπροσωπιών που αναφέρθηκαν νωρίτερα.
    - **Controller:** Πρόκειται για την οντότητα που περιέχει λογική. Παρέχει λειτουργικότητες οι οποίες βοηθούν την εφαρμογή να επεξεργαστεί τα δεδομένα και να τα φέρει στην επιθυμητή μορφή, δηλαδή την μορφή που έχει οριστεί από τα DAO. Για παράδειγμα, σε προηγούμενη εικόνα είδαμε παράδειγμα από ένα λήμμα που περιέχεται στο ELG, σε αυτή την οντότητα γίνεται η αποδόμηση του, η αντιστοίχιση του στις κατάλληλες οντότητες και η διαχείριση του. Έχει ως εξάρτηση την οντότητα DAO και

χρησιμοποιεί τις Διαπροσωπίες για να αποκτήσει επικοινωνία με την βάση δεδομένων. Σε αυτή την οντότητα γίνεται εισαγωγή του αρχείου του λεξικού ELG και μετά από κατάλληλες μετατροπές φορτώνεται το λεξικό στην βάση δεδομένων.

  - **Samples:** Σε αυτή την οντότητα έχει φορτωθεί το λεξικό του ELG σε bean ώστε να μπορεί ο Initializer να το διαβάσει.
  - **Initializer:** Αρμόδιος για την αρχικοποίηση του συστήματος. Έχει ως εξαρτήσεις samples και controller. Έχει κλάση με main μέθοδο η οποία καλείται και με χρήση των συναρτήσεων που παρέχει ο Controller κάνει την αρχικοποίηση του συστήματος. Το σύστημα σε τοπικό περιβάλλον κάνει περίπου 3,5 ώρες να αρχικοποιηθεί.
  - **Service:** Πρόκειται για την οντότητα που παρέχει στο API συναρτήσεις οι οποίες λαμβάνουν την είσοδο που παρέχει ένα HTTP αίτημα, χρησιμοποιούν συναρτήσεις από τον Controller για να κάνουν επεξεργασία και επικοινωνία με την βάση δεδομένων, μετατρέπουν το αποτέλεσμα (όπου χρειάζεται) σε JSON μορφή και το επιστρέφουν σαν έξοδο. Έχει ως εξάρτηση τον Controller.

- **API:** Είναι η οντότητα που παρέχει τα endpoints του συστήματος. Παρέχει πάνω από 66 endpoints (δες API documentation) τα οποία προσφέρουν και διαφορετική λειτουργικότητα το καθένα. Επικοινωνεί υποστηρίζει επικοινωνία μόνο μέσω JSON μορφής δεδομένων. Η ταυτοποίηση και εξουσιοδότηση των χρηστών να αποκτήσουν πρόσβαση στο σύστημα γίνεται σε αυτή την οντότητα. Έχει ως εξάρτηση το Service. Το API τρέχει σε έναν JBoss διακομιστή ο οποίος δημιουργείται με την βοήθεια του IntelliJ.

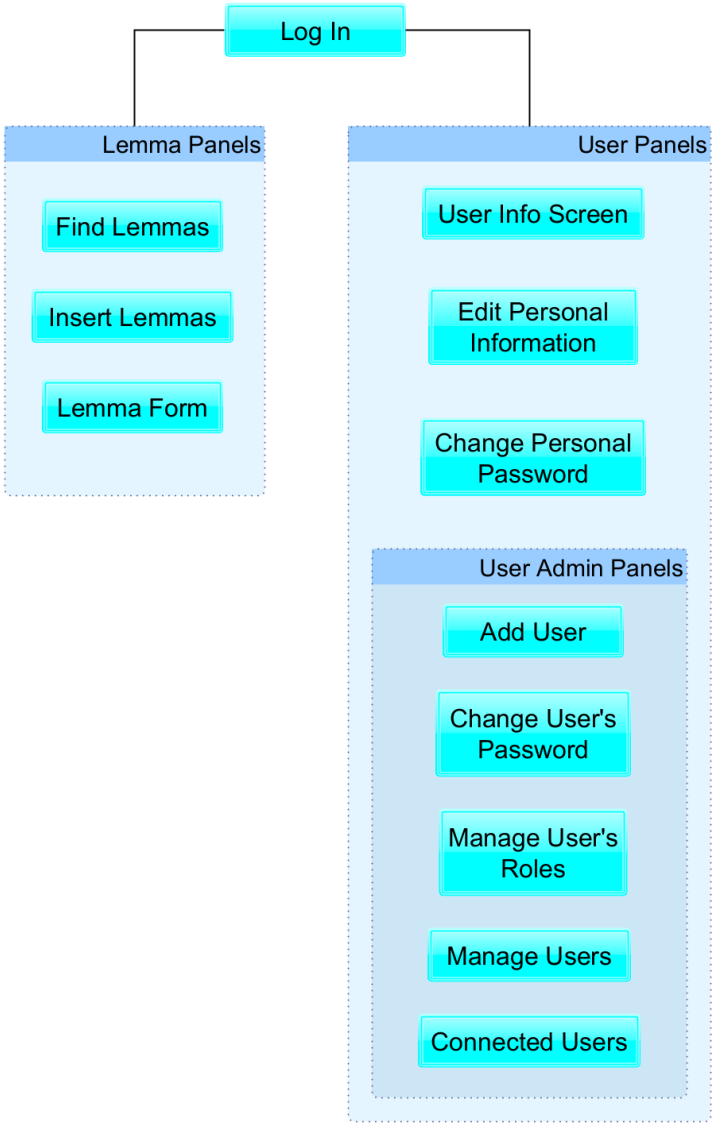
Η εξουσιοδότηση προκύπτει με βάσει τους ρόλους οι οποίοι έχουν ανατεθεί στο χρήστη που κάνει το αίτημα και την λειτουργία στην οποία θέλει να αποκτήσει πρόσβαση. Οι διαθέσιμοι ρόλοι είναι οι ακόλουθοι:

- **User Manager:** Είναι κατηγορίας διαχειριστή σε ότι αφορά την διαχείριση χρηστών. Κάποιες από τις δυνατότητες που έχει είναι η δημιουργία χρήστη, αλλαγή κωδικού του, αλλαγή κατάστασης online του χρήστη κ.α.

- **Lemma Editor:** Δίνει την δυνατότητα στον χρήστη να τροποποιήσει τα ήδη υπάρχοντα λήμματα.
- **Lemma Validator:** Δίνει την δυνατότητα στο χρήστη να αλλάξει τις τιμές του πεδίου verified. Το πεδίο έχει 10 θέσεις true/false και η κάθε θέση αντιστοιχεί σε συγκεκριμένο γνώρισμα του λήμματος. Όταν είναι true σημαίνει ότι το αντίστοιχο γνώρισμα έχει ελεγχθεί και είναι σωστό. Για παράδειγμα η 5η θέση αντιστοιχεί στον συλλαβισμό της λέξης, αν η τιμή στην θέση είναι true σημαίνει ότι ο συλλαβισμός έχει ελεγχθεί και είναι σωστός, false διαφορετικά.
- **Dictionary Manager:** Το σύστημα υποστηρίζει την δημιουργία διαφορετικών λεξικών. Για παράδειγμα ένα λεξικό με τις 100 πιο συχνές λέξεις της ελληνικής γλώσσας. Ο χρήστης με αυτόν τον ρόλο έχει το δικαίωμα να δημιουργήσει τέτοια λεξικά και να τους προσθέσει λήμματα.
- **Linguistics Content Manager:** Πρόκειται για τον ρόλο που δίνει την δυνατότητα στον χρήστη να διαχειριστεί κάποιες από τις διαθέσιμες «περιφερειακές» πληροφορίες που υπάρχουν για ένα λήμμα. Για παράδειγμα τις διαθέσιμες γλώσσες, καταλήξεις και προθέματα, κλιτικές πληροφορίες κ.α.
- **Audio Files Manager:** Ο ρόλος αυτός δίνει την δυνατότητα στο χρήστη να διαχειριστεί ό,τι αφορά τα αρχεία ήχου καθώς και τα ίδια τα αρχεία.
- **Image Files Manager:** Ο ρόλος αυτός δίνει την δυνατότητα στο χρήστη να διαχειριστεί ό,τι αφορά τα αρχεία εικόνας καθώς και τα ίδια τα αρχεία.
- **Lemma Creator:** Ο ρόλος αυτός δίνει την δυνατότητα στο χρήστη να προσθέσει κάποιο λήμμα.

2.5 Frontend (GUI)

Στο Frontend επάγονται τα γραφικά παράθυρα με τα οποία αλληλοεπιδρά ο χρήστης καθώς και η λογική που μπορεί να «κρύβουν» από πίσω προκειμένου να παρέχουν προηγμένες λειτουργίες. Σε περίπτωση σφάλματος είτε της εφαρμογής είτε της επικοινωνίας με το API παρουσιάζεται σχετικό μήνυμα στον χρήστη με πληροφορίες σχετικά με το σφάλμα.



Εικόνα 21: Δομή εφαρμογής

Στην παραπάνω εικόνα φαίνεται η δομή της εφαρμογής. Αναλυτικότερα έχουμε:

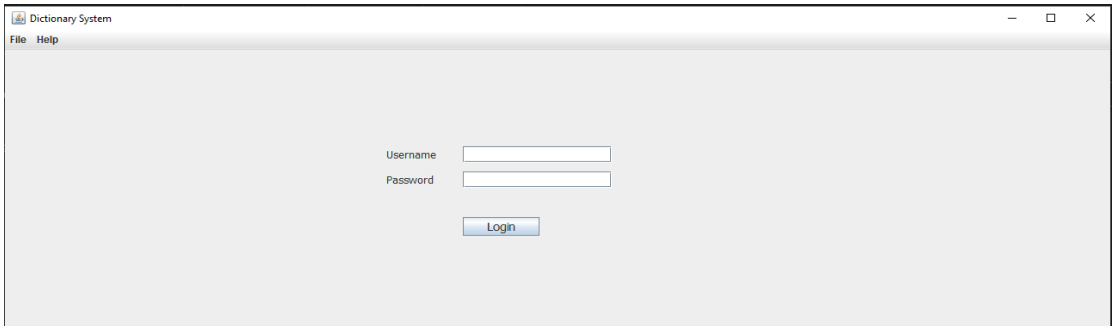
- **Log In:** Είναι το παράθυρο που φαίνεται κατά την εκτέλεση της εφαρμογής. Εκεί ο χρήστης τοποθετεί τα στοιχεία του (Username/ Password) και γίνεται η ταυτοποίηση.

- **Lemma Panels:** Πρόκειται για όλα τα παράθυρα που αφορούν το λεξικό αυτό καθ' αυτό.
    - **Find Lemmas:** Είναι το παράθυρο που προσφέρει τις περισσότερες λειτουργίες και πληροφορίες στην εφαρμογή. Διαθέτει αναζήτηση με βάση επιλογή του χρήστη, κάποιες από τις διαθέσιμες επιλογές είναι αναζήτηση βάσει λέξης, Γλώσσας, μέρος του λόγου, αριθμό γραμμάτων και συλλαβών, συχνότητας της λέξης κ.α. Κατόπιν αναζήτησης εμφανίζεται λίστα με τα διαθέσιμα αποτελέσματα. Στη λίστα με τα αποτελέσματα υπάρχει λειτουργία σελιδοποίησης για τις αναζητήσεις που φέρνουν πολλά αποτελέσματα. Ο χρήστης διαλέγει την λέξη που θέλει και κατόπιν εμφανίζονται στο παράθυρο οι σχετικές πληροφορίες που είναι καταχωρημένες στη βάση δεδομένων. Δίνεται επίσης η επιλογή στον χρήστη να κάνει αλλαγές στην επιλεγμένη λέξη με την δημιουργία νέου παραθύρου το οποίο έχει τη μορφή Lemma Form. Επιπλέον δίνεται η δυνατότητα διαγραφής της λέξης με αντίστοιχο κουμπί, επειδή η λειτουργία αυτή είναι μη αναστρέψιμη υπάρχει σχετικό παράθυρο επιβεβαίωσης.
    - **Insert Lemma:** Μοιάζει με την δομή που έχει το Lemma Form. Ο χρήστης συμπληρώνει τα απαραίτητα πεδία. Βασική προϋπόθεση είναι να είναι μοναδικός ο συνδυασμός της λέξης (word), μέρος του λόγου (part of speech) και του φωνητικού συλλαβισμού (ipa). Διαθέτει κουμπί το οποίο αποθηκεύει το λήμμα και κουμπί που επαναφέρει τα παιδία στην αρχική τους κατάσταση (κενά).
    - **Lemma Form:** πρόκειται για το παράθυρο που παρουσιάζεται στον χρήστη όταν επιλέξει να τροποποιήσει κάποιο λήμμα Υπάρχει κουμπί για να προχωρήσει σε αποθήκευση των αλλαγών του και αντίστοιχο κουμπί για να ακυρώσει τις αλλαγές του.
- **User Panels:** Σύνολο το οποίο περιέχει παράθυρα που έχουν να κάνουν με λειτουργίες γύρω από την οντότητα του χρήστη.
    - **User Information Screen:** Στο παράθυρο αυτό ο χρήστης μπορεί να δει τις πληροφορίες του λογαριασμού του όπως και τους ρόλους που του έχουν ανατεθεί.

- **Edit Personal Information:** Στο παράθυρο αυτό ο χρήστης μπορεί να αλλάξει τις πληροφορίες του λογαριασμού του πλην του username του και του κωδικού του.
- **Change Personal Password:** Εκεί ο χρήστης μπορεί να αλλάξει τον προσωπικό του κωδικό πρόσβασης αφού πρώτα πληκτρολογήσει τον τρέχοντα κωδικό πρόσβασης.
- **Admin:**
  - **Add user:** Σε αυτό το παράθυρο οι διαχειριστές μπορούν να προσθέσουν κάποιον νέο χρήστη. Αυτός είναι ο μόνος τρόπος με τον οποίο μπορεί να προστεθεί κάποιος νέος χρήστης στο σύστημα.
  - **Change Users Password:** Μέσω αυτού του παραθύρου οι διαχειριστές μπορούν να αλλάξουν τον κωδικό κάποιου χρήστη.
  - **Manage User's Roles:** Σε αυτό το παράθυρο οι διαχειριστές βλέπουν για κάποιον επιλεγμένο χρήστη μία λίστα με τους ρόλους που του έχουν ανατεθεί και μία λίστα με τους ρόλους που δεν του έχουν ανατεθεί. Έχουν τη δυνατότητα να μεταφέρουν έναν οι παραπάνω ρόλους από τη μία λίστα στην άλλη και αντίστροφα. Με αυτόν τον τρόπο αποδίδονται οι ρόλοι σε κάθε χρήστη.
  - **Connected Users:** Σε αυτό το παράθυρο φαίνονται ποιοι χρήστες είναι συνδεδεμένοι. Οι διαχειριστές έχουν την δυνατότητα από αυτό το παράθυρο να αλλάξουν την κατάσταση σύνδεσης ενός χρήστη.

2.6 Παραδείγματα

Κατά την έναρξη της εφαρμογής βλέπουμε την οθόνη που κάνει login ο χρήστης.



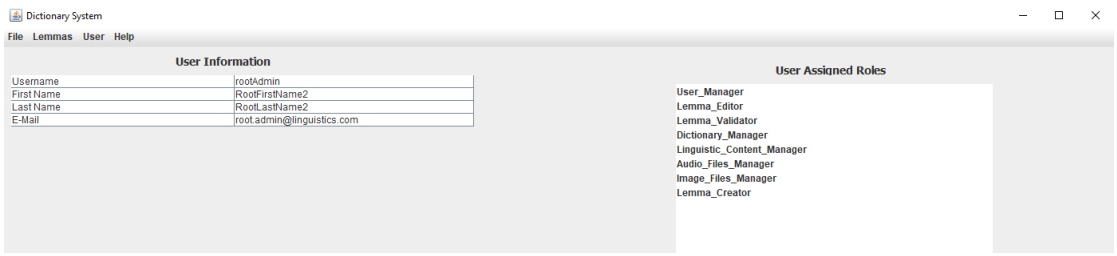
Εικόνα 22: Login window

2.6.1 Τμήμα διαχείρισης χρηστών

2.6.1.1 Λειτουργίες Απλού χρήστη

Ανεξάρτητα με τους ρόλους που έχουν αναχθεί στον χρήστη έχει πρόσβαση σε κάποιες βασικές λειτουργίες.

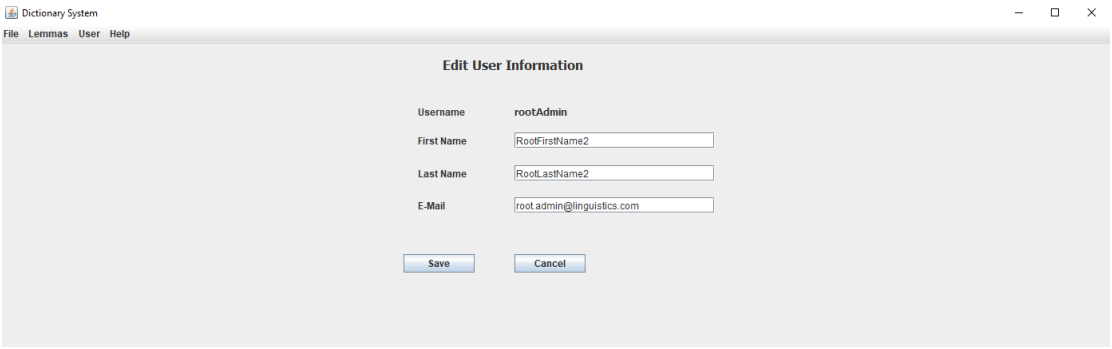
Πληροφορίες λογαριασμού χρήστη



Εικόνα 23: User information

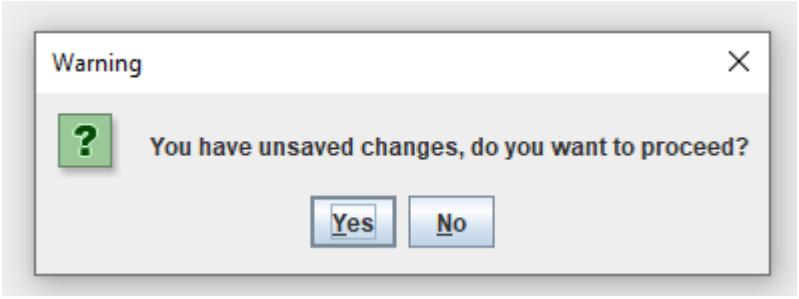
Όπως φαίνεται και στην παραπάνω εικόνα σε αυτή τη σελίδα ο χρήστης μπορεί να δει το username, το όνομα, το επώνυμο και το mail που είναι περασμένα στο σύστημα. Μπορεί επίσης να δει και ποιοι ρόλοι του έχουν ανατεθεί. Στο παράδειγμα τις εικόνας βλέπουμε να έχουν ανατεθεί όλοι οι διαθέσιμοι ρόλοι και πρόκειται για έναν super user.

Αλλαγή πληροφοριών χρήστη



Εικόνα 24: Edit User Information

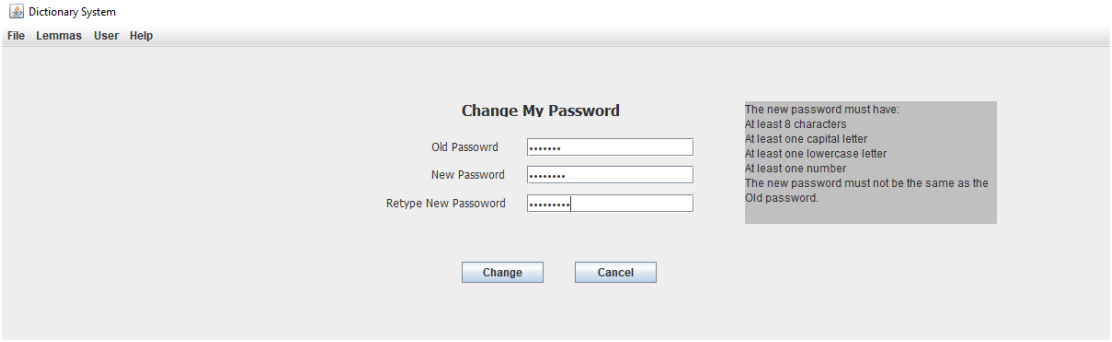
Στην παραπάνω εικόνα φαίνεται πως μπορεί κάποιος χρήστης να αλλάξει τις πληροφορίες του λογαριασμού του. Σημειώνεται πως κάποιος χρήστης δεν μπορεί να αλλάξει το username του. Σε περίπτωση που ο χρήστης κάνει αλλαγές και πάει να αλλάξει παράθυρο πριν αποθηκεύσει τις αλλαγές του, βλέπει σχετικό μήνυμα.



Εικόνα 25: Unsaved changes popup message

Αυτό το μήνυμα μπορεί να παραχθεί και σε άλλα σημεία της εφαρμογής όπου υπάρχουν αντίστοιχα αλλαγές οι οποίες δεν έχουν αποθηκευτεί.

Αλλαγή προσωπικού κωδικού πρόσβασης

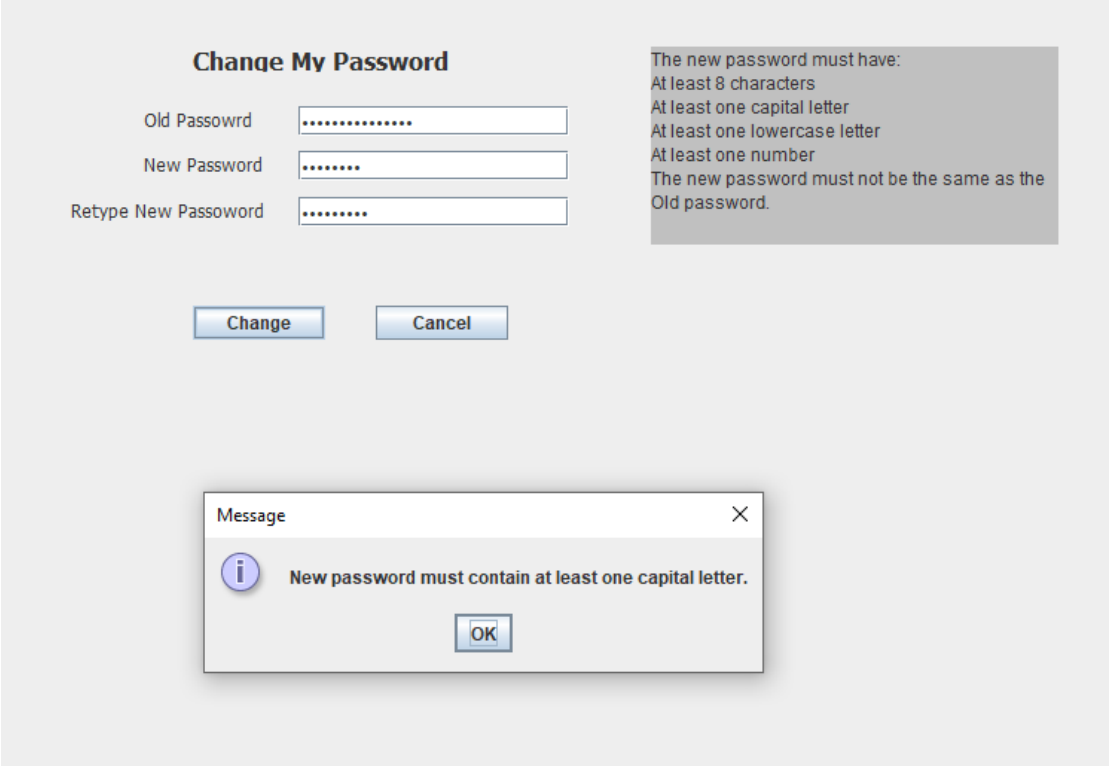


Εικόνα 26: Change personal password window

Μέσα από το παράθυρο αυτό ο χρήστης μπορεί να αλλάξει τον προσωπικό του κωδικό πρόσβασης. Για να ολοκληρωθεί η αλλαγή αυτή με επιτυχία πρέπει πρώτα να πληκτρολογήσει σωστά τον παλαιό του κωδικό. Για τον καινούριο κωδικό πρόσβασης πρέπει πρώτα να πληρούνται κάποιες προδιαγραφές όπως φαίνεται και στην εικόνα.

- Ο κωδικός πρέπει να αποτελείται από τουλάχιστον 8 χαρακτήρες.
- Να έχει τουλάχιστον ένα κεφαλαίο γράμμα
- Να έχει τουλάχιστον ένα πεζό γράμμα
- Να έχει τουλάχιστον έναν αριθμό
- Να μην είναι ίδιος με τον παλαιό κωδικό

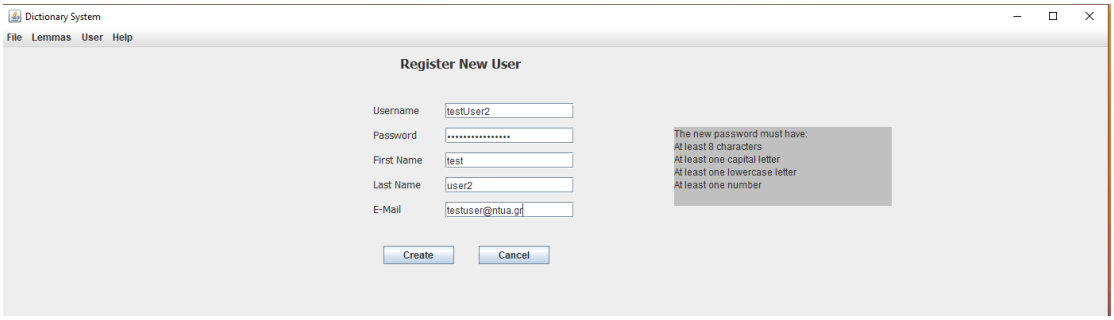
Σε περίπτωση που κάποια από τις παραπάνω απαιτήσεις δεν εκπληρώνεται, ο χρήστης λαμβάνει αντίστοιχο μήνυμα.



Εικόνα 27:Μήνυμα μη εκπλήρωσης απαιτήσεων νέου κωδικού

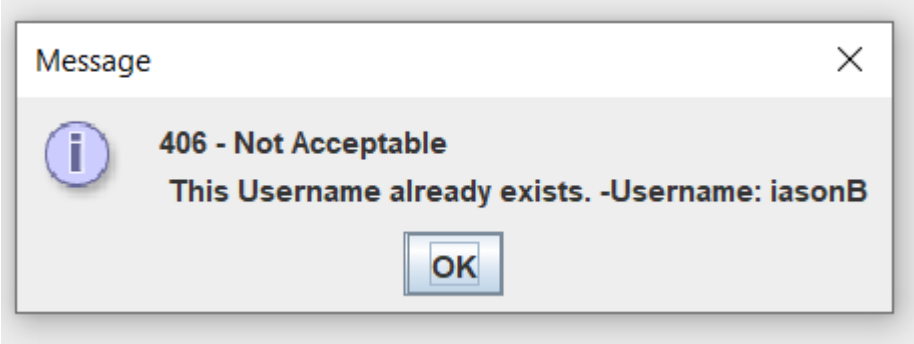
2.6.1.2 Λειτουργίες διαχειριστή χρηστών

Καταχώρηση νέου χρήστη

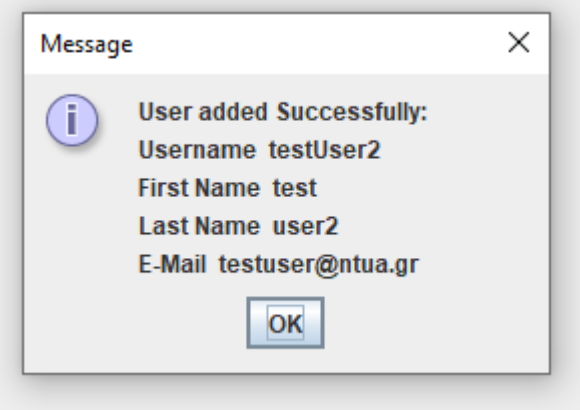


Εικόνα 28: Καταχώρηση νέου χρήστη

Μέσα από αυτό το παράθυρο οι διαχειριστές χρηστών μπορούν να καταχωρήσουν έναν νέο χρήστη. Ο κωδικός πρόσβασης πρέπει να τηρεί τα κριτήρια που αναφέρθηκαν και νωρίτερα. Στο σύστημα μας έχουμε εξασφαλίσει το username να είναι μοναδικό ανά χρήστη. Στην περίπτωση που εισαχθεί ήδη υπάρχον username, εμφανίζεται σχετικό μήνυμα στον χρήστη.

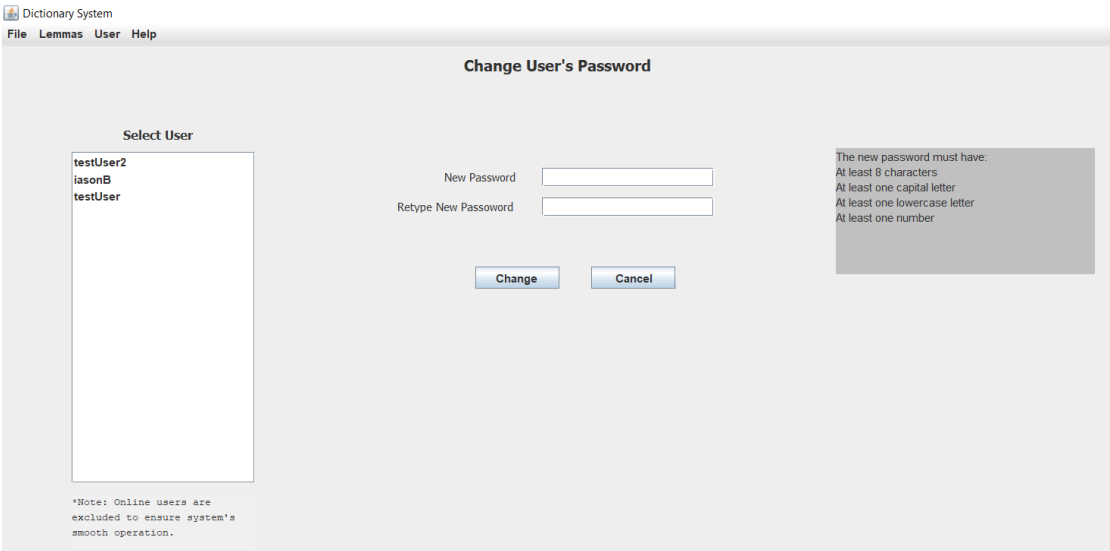


Εικόνα 29: Το username χρησιμοποιείται.



Εικόνα 30:Καταχώρηση νέου χρήστη με επιτυχία

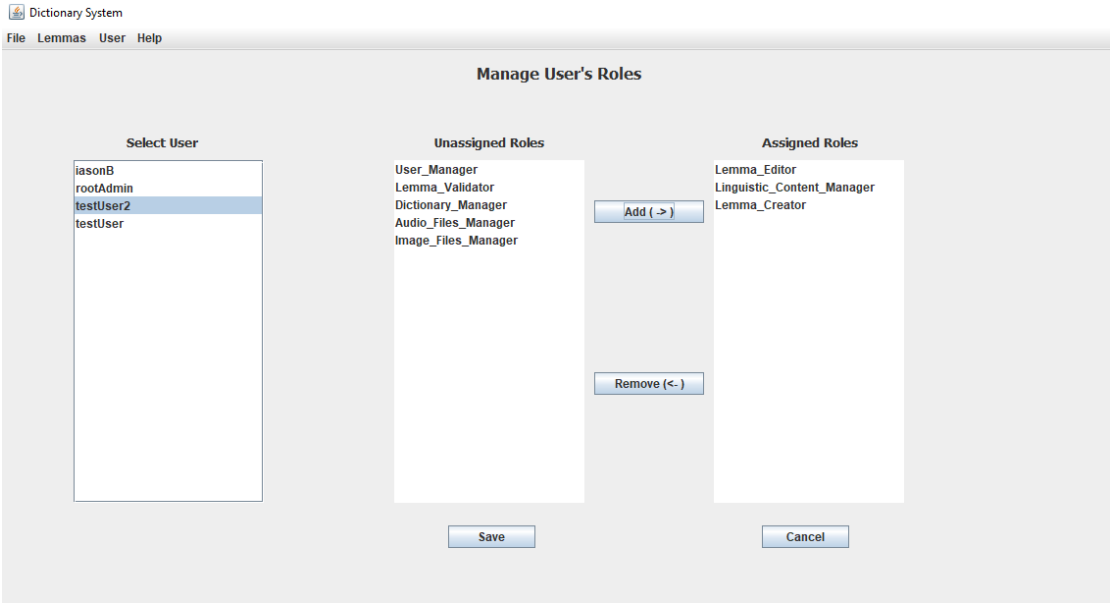
Αλλαγή κωδικού χρήστη



Εικόνα 31:Αλλαγή κωδικού χρήστη

Από αυτό το παράθυρο οι διαχειριστές χρηστών έχουν την δυνατότητα να αλλάξουν τον κωδικό από κάποιον χρήστη. Επιτρέπεται η αλλαγή μόνο των χρηστών που δεν είναι ενεργοί στο σύστημα, διαφορετικά θα δημιουργηθεί πρόβλημα στην ταυτοποίηση του επόμενου αιτήματος των ενεργών χρηστών. Ισχύουν οι απαιτήσεις κωδικού πρόσβασης.

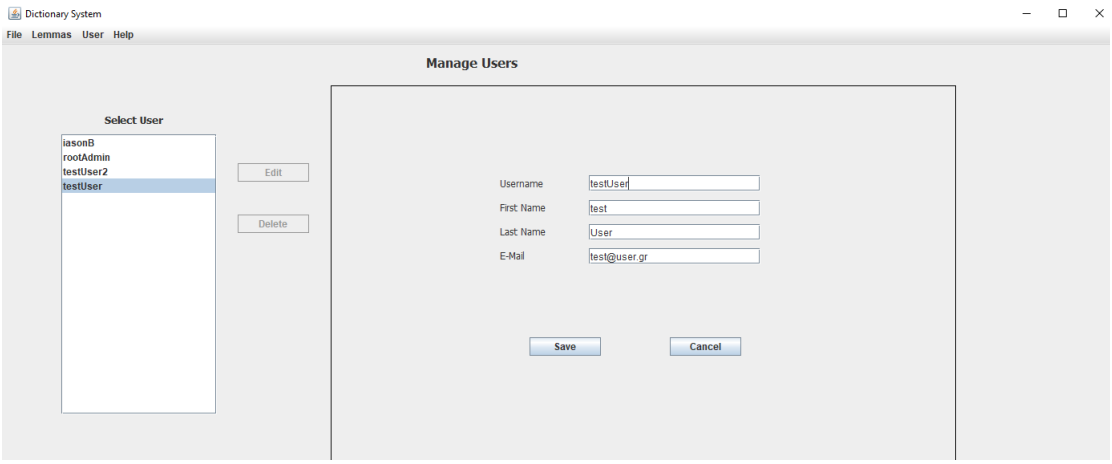
Διαχείριση ρόλων χρήστη



Εικόνα 32:Διαχείρηση ρόλων χρήστη

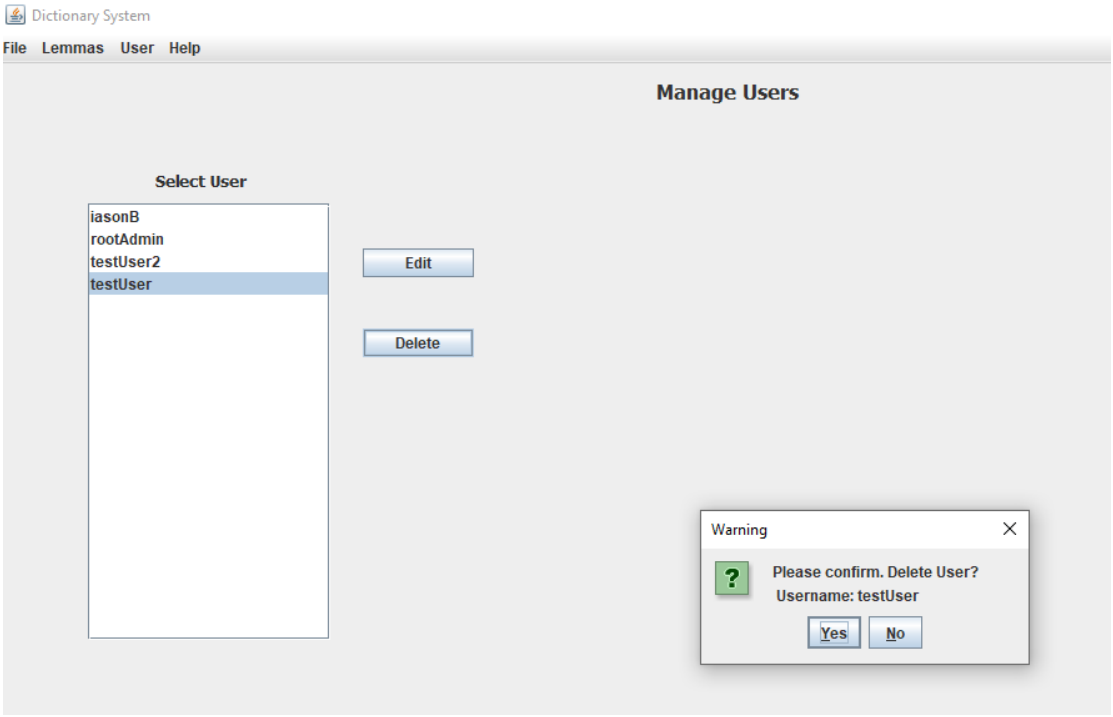
Σε αυτό το παράθυρο ο διαχειριστής χρηστών επιλέγει έναν χρήστη από την λίστα με τους διαθέσιμους χρήστες και φορτώνονται ποιοι ρόλοι του έχουν ανατεθεί και ποιοι και ποιοι όχι. Έχει τη δυνατότητα να αφαιρέσει και να προσθέσει ρόλους στον χρήστη.

Διαχείριση Χρηστών



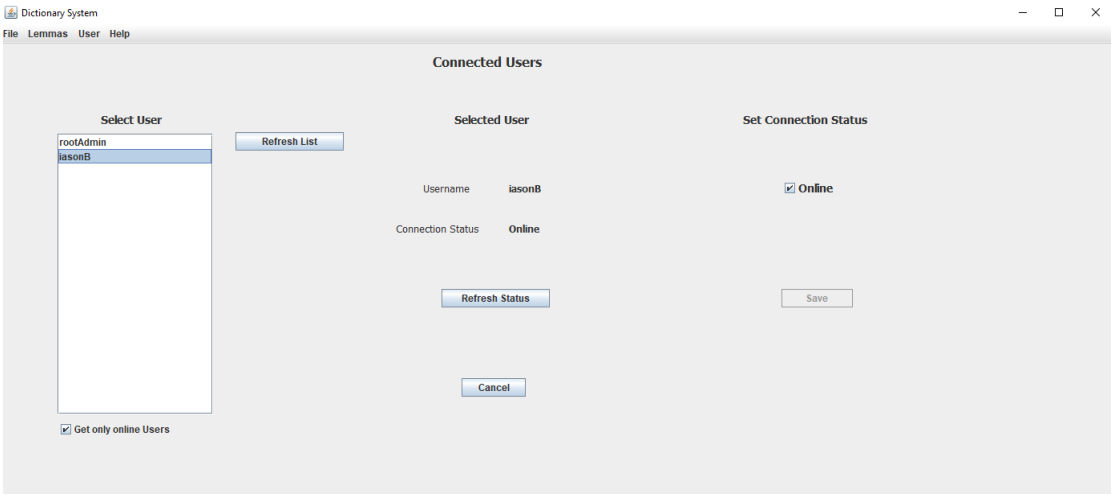
Εικόνα 33: Διαχείριση χρηστών

Σε αυτό το παράθυρο δίνεται η δυνατότητα στο διαχειριστή να αλλάξει τις πληροφορίες ενός χρήστη και να διαγράψει ένα χρήστη. Κατά τη διαγραφή χρήστη εμφανίζεται σχετικό μήνυμα επιβεβαίωσης.



Εικόνα 34: Μήνυμα επιβεβαίωσης διαγραφής χρήστη

Συνδεδεμένοι χρήστες



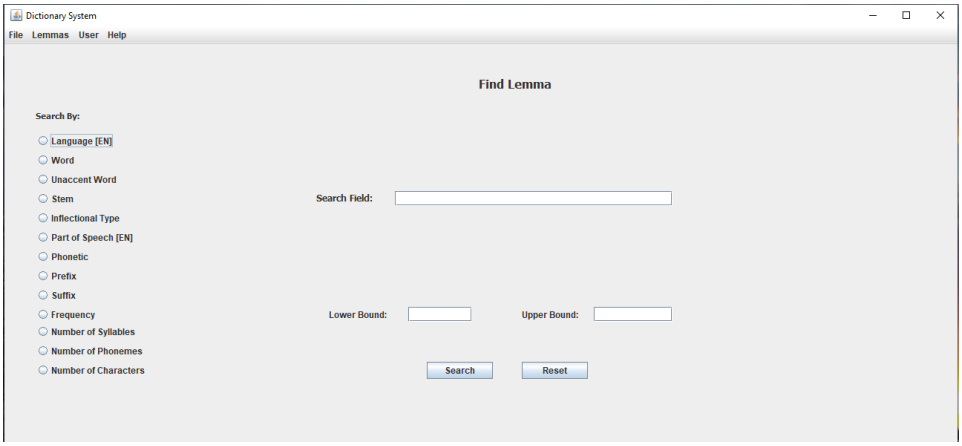
Εικόνα 35: Παράθυρο εύρεσης και διαχείρισης συνδεδεμένων χρηστών

Στο παράθυρο οι διαχειριστές μπορούν να δουν ποιοι χρήστες είναι συνδεδεμένοι και να αλλάξουν την κατάσταση σύνδεσης τους.

2.6.2 Τμήμα Λημμάτων

2.6.2.1 Εύρεση Λημμάτων και Διαγραφή Λημμάτων

Εύρεση Λημμάτων



Εικόνα 36: Αναζήτηση λημμάτων

Μετά την επιτυχή ταυτοποίηση ενός χρήστη, ο χρήστης βλέπει την παραπάνω εικόνα. Εκεί χρησιμοποιώντας το πεδίο αναζήτησης μπορεί να ψάξει ανάλογα με το

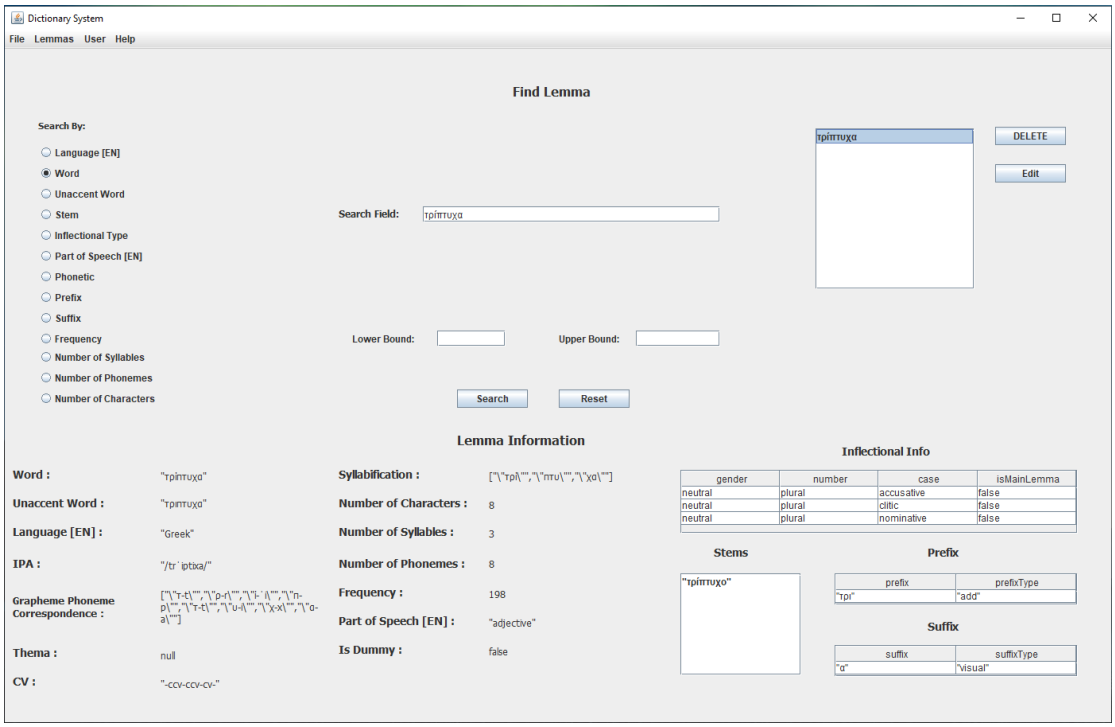
κριτήριο που έχει επιλέξει. Εάν δεν επιλέξει κανένα κριτήριο τότε η εφαρμογή θα του φέρει όλα τα λήμματα που υπάρχουν στην βάση δεδομένων, σε αυτή τη λειτουργία έχουν πρόσβαση μόνο οι χρήστες με τον ρόλο 'Dictionary Manager'. Αναλυτικότερα τα κριτήρια αναζήτησης είναι:

- **Γλώσσα (Αγγλικά)**: Ο χρήστης εισάγει μία γλώσσα (στα Αγγλικά), τώρα το σύστημα υποστηρίζει Ελληνικά και Αγγλικά, αλλά υπάρχουν λήμματα μόνο για την Ελληνική γλώσσα. Με την αναζήτηση αυτή θα δει όλα τα λήμματα που είναι καταχωρημένα και ανήκουν στην γλώσσα που έχει πληκτρολογήσει.
- **Λέξης**: Ο χρήστης εισάγει μια τονισμένη λέξη και η εφαρμογή φέρνει τα αντίστοιχα αποτελέσματα από το σύστημα.
- **Μη τονισμένη Λέξη**: Αφορά την αναζήτηση λέξεων χωρίς να βάλει ο χρήστης τονισμό. Αυτή η λειτουργία θα φανεί ιδιαίτερα χρήσιμη σε μελλοντική υλοποίηση μηχανής αναζήτησης.
- **Stem**: Ο χρήστης εισάγει κάποια λέξη (τονισμένη) και το σύστημα θα του επιστρέψει όλες τις λέξεις που έχουν αυτή τη λέξη σαν αναφερόμενη λέξη.
- **Κλητική Πληροφορία**: Για αυτή την αναζήτηση ο χρήστης πρέπει να εισάγει σε μορφή JSON Object την επιθυμητή κλητική πληροφορία και το σύστημα θα του παρουσιάσει όλες τις λέξεις που έχουν αυτήν την κλητική πληροφορία.
- **Μέρος του λόγου (Αγγλικά)**: Ο χρήστης εισάγει το μέρος του λόγου στην Αγγλική γλώσσα και το σύστημα θα του παρουσιάσει όλες τις λέξεις που είναι αυτό το μέρος του λόγου.
- **Φωνητική γραφή**: Ο χρήστης κάνει αναζήτηση με την γραφή με την οποία παράγεται η προφορά της λέξης (πρέπει να είναι ανάμεσα σε '/'), για παράδειγμα /ðasˈaci/ η φωνητική γραφή για τη λέξη «δασάκι»
- **Πρόθεμα**: Το σύστημα θα επιστρέψει τις λέξεις που έχουν σαν πρόθεμα την εισαγωγή του χρήστη.
- **Κατάληξη**: Το σύστημα θα επιστρέψει τις λέξεις που έχουν σαν κατάληξη την εισαγωγή του χρήστη.
- **Συχνότητα λέξης**: Μπορεί να γίνει αναζήτηση για λέξεις που έχουν την συγκεκριμένη συχνότητα που εισάγει ο χρήστης. Δίνεται και η δυνατότητα

αναζήτησης εύρους συχνότητας συμπληρώνοντας κατάλληλα τα πεδία Lower Bound και Upper Bound.

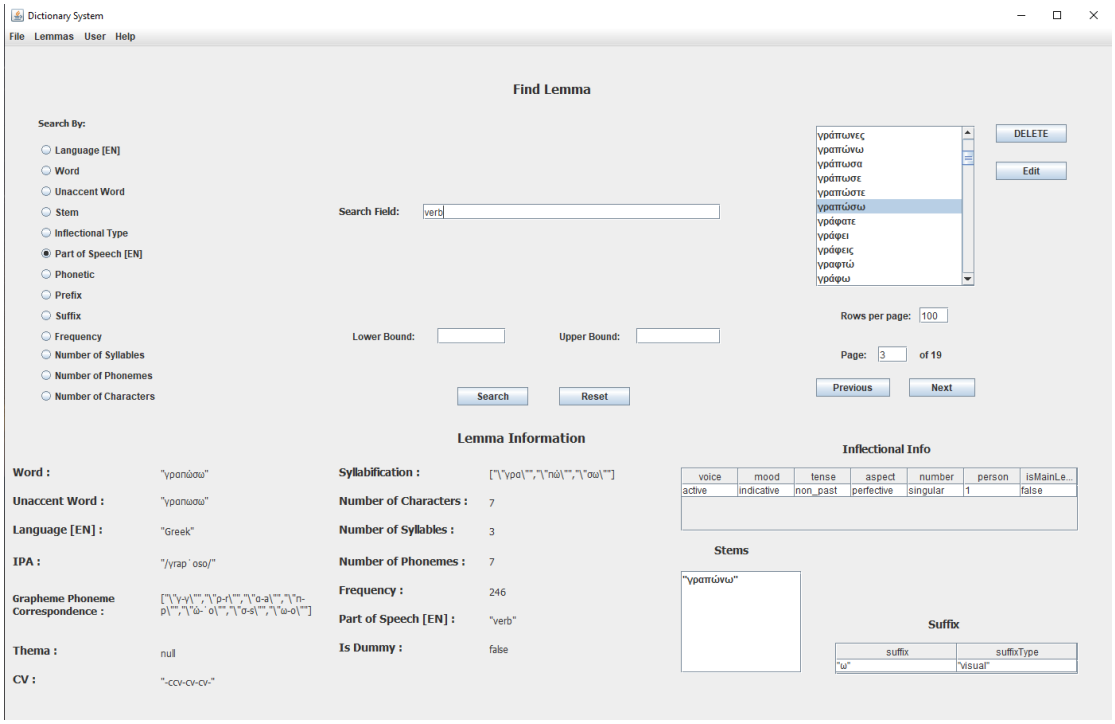
- **Αριθμού συλλαβών**
- **Αριθμού φωνηέντων**
- **Αριθμού γραμμάτων της λέξης**

Εύρεση Λημμάτων βάσει λέξης



Εικόνα 37: Αναζήτηση της λέξης 'τρίπτυχα'.

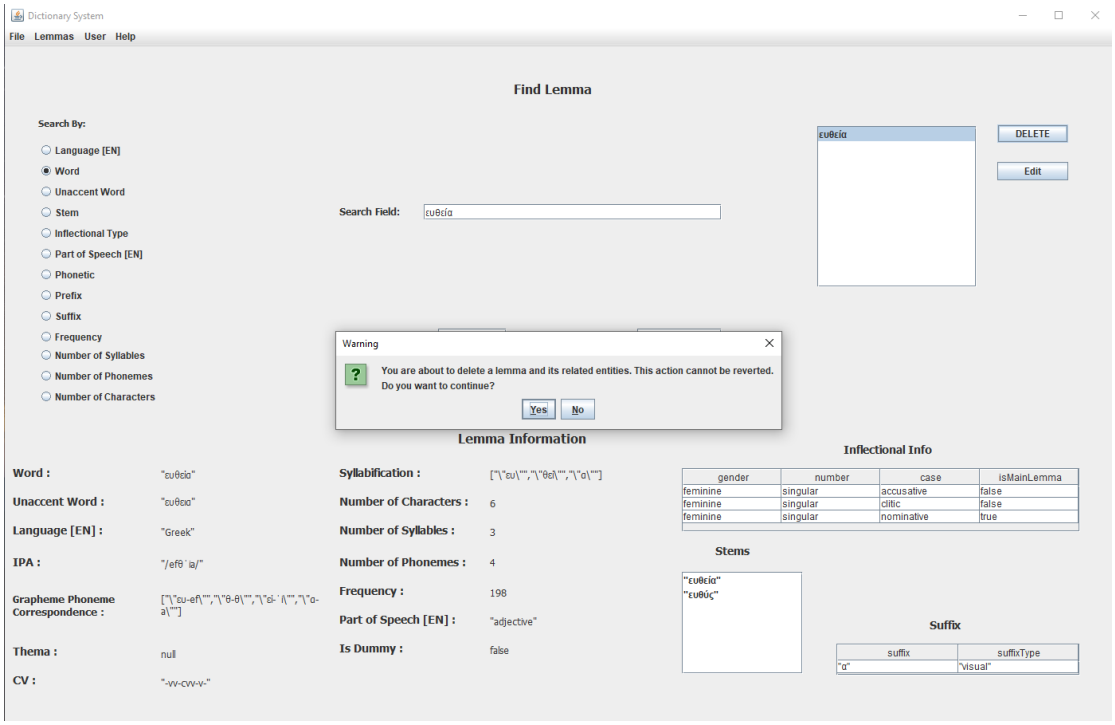
Εύρεση Λημμάτων Βάσει μέρους του λόγου



Εικόνα 38: Αναζήτηση λέξεων που είναι ρήματα.

Διαγραφή λημμάτων

Ο χρήστης μπορεί να διαγράψει ένα λήμμα που έχει επιλέξει με χρήση του αντίστοιχου κουμπιού. Πριν γίνει τη διαγραφή το σύστημα επιβεβαιώνει με το ν χρήστη καθώς πρόκειται για μη αναστρέψιμη διαδικασία.



Εικόνα 39: Διαγραφή λέξης και μήνυμα επιβεβαίωσης διαγραφής.

3 Προτάσεις για το μέλλον

Η λεπτομέρεια με την οποία έχει σχεδιαστεί στο σύστημα υποστηρίζει περισσότερες λειτουργίες από αυτές που είδαμε. Κάποιες από αυτές είναι:

- Λειτουργία εισαγωγής και εξαγωγής: Πρόκειται για μία χρήσιμη λειτουργία η οποία θα επιτρέπει στον Dictionary Manager να κάνει εξαγωγή λεξικών και να τα προσθέτει στο κύριο σύστημα. Οι εξαγωγές αυτές μπορούν να αποτελούν και διαδικασία εφεδρικού αρχείου. Αντίστοιχα η εφαρμογή μπορεί να δίνει τη δυνατότητα στο χρήστη να φορτώνει μαζικά στο σύστημα τις εξαγωγές που αναφέραμε νωρίτερα.
- Περιβάλλον διαχείρισης πρόσθετων πληροφοριών: Θα είναι το γραφικό περιβάλλον το οποίο θα δίνει την δυνατότητα στο χρήστη να διαχειριστεί τις «περιφερειακές» πληροφορίες ενός λήμματος όπως είναι το μέρος του λόγου, πρόθεμα/κατάληξη, διαθέσιμες γλώσσες κ.α.

- Λειτουργία διαχείρισης αρχείων: Για την διαχείριση αρχείων εικόνας και ήχου κρίνεται απαραίτητη υλοποίηση δομών που να υποστηρίζουν την λειτουργία αυτή. Κατόπιν ο χρήστης μέσα από κατάλληλη υλοποίηση στο γραφικό περιβάλλον θα μπορεί να προσθέσει, να αφαιρέσει αρχεία και να τα συνδέσει με το λήμμα με το οποίο αντιστοιχούν.

Μετά την προσθήκη των παραπάνω λειτουργιών στο κορμό του συστήματος, υπάρχουν οι απαραίτητες δομές για την υλοποίηση μιας έξυπνης μηχανής αναζήτησης λημμάτων. Μέσω της οποίας ο χρήστης θα μπορεί να κάνει πιο πολύπλοκες αναζητήσεις στο λεξικό.

4 Appendix

4.1 Βιβλιογραφία

Atlassian, n.d. Better code with Bitbucket: 4 starting steps [WWW Document]. Bitbucket. URL https://bitbucket.org/product/guides/basics/four-starting-steps (accessed 1.30.22).

Busbee, K.L., 2018. Integrated Development Environment.

Coronel, C., Morris, S., 2019. DATABASE SYSTEMS Design, Implementation & Management. CENGAGE, p. 37.

Creative Commons Αναφορά-Παρόμοια Διανομή 4.0 — CC BY-SA 4.0 [WWW Document], n.d. URL https://creativecommons.org/licenses/by-sa/4.0/legalcode.el (accessed 2.12.22).

ELG - Greek Annotated Dictionary with Morphological and Linguistic Features [WWW Document], n.d. . Eur. Lang. Grid. URL https://live.european-language-grid.eu/catalogue/lcr/7360 (accessed 2.11.22).

Elmasri, R., Navathe, S., 2007. Fundamentals of database systems, 5th ed. ed. Pearson/Addison Wesley, Boston.

Features - IntelliJ IDEA [WWW Document], n.d. . JetBrains. URL https://www.jetbrains.com/idea/features/ (accessed 1.31.22).

Git [WWW Document], n.d. URL https://git-scm.com/ (accessed 2.11.22).

Graham, S., n.d. Web Services Architecture 41.

Gson User Guide - gson [WWW Document], n.d. URL https://sites.google.com/site/gson/gson-user-guide#TOC-Overview (accessed 2.11.22).

Guindon, C., n.d. Eclipse desktop & web IDEs | The Eclipse Foundation [WWW Document]. URL https://www.eclipse.org/ide/ (accessed 1.30.22).

Hofmann, M., O'Mahony, M., 2003. A framework to utilise urban bus data for advanced data analysis.

HTTP response status codes - HTTP | MDN [WWW Document], n.d. URL https://developer.mozilla.org/en-US/docs/Web/HTTP/Status (accessed 2.12.22).

Indexes [WWW Document], 2016. . PostgreSQL Doc. URL https://www.postgresql.org/docs/9.1/indexes.html (accessed 2.11.22).

Java | Definition & Facts | Britannica [WWW Document], n.d. URL https://www.britannica.com/technology/Java-computer-programming-language (accessed 1.30.22).

Java Release History - Dev.java [WWW Document], n.d. . Devjava Destin. Java Dev. URL https://dev.java/download/releases/ (accessed 1.30.22).

JavaBeans(TM) Specification 1.01 Final Release [WWW Document], n.d. URL https://download.oracle.com/otndocs/jcp/7224-javabeans-1.01-fr-spec-oth-JSpec/ (accessed 2.11.22).

Maven – Introduction [WWW Document], n.d. URL https://maven.apache.org/what-is-maven.html (accessed 2.12.22).

Maven – Welcome to Apache Maven [WWW Document], n.d. URL https://maven.apache.org/ (accessed 2.12.22).

Meier, A., Kaufmann, M., 2019. SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management. Springer Fachmedien Wiesbaden, Wiesbaden. https://doi.org/10.1007/978-3-658-24549-8

PostgreSQL: About [WWW Document], n.d. URL https://www.postgresql.org/about/ (accessed 2.11.22).

Schildt, H., 2019. Java The Complete Referene Eleventh Edition, 11th Edition. ed. McGraw-Hill Education.

Silva, R.A. e, n.d. I. Overview [WWW Document]. URL https://pgmodeler.io/support/docs/i-overview?v=0.9.4 (accessed 1.30.22).

Spring Framework 1.0 Final Released [WWW Document], 15:00:00.0. URL https://spring.io/blog/2004/03/24/spring-framework-1-0-final-released (accessed 2.11.22).

Spring Framework Overview [WWW Document], n.d. URL

https://docs.spring.io/spring-framework/docs/current/reference/html/overview.html#overview (accessed 2.11.22).

TOP IDE Top Integrated Development Environment index [WWW Document], n.d. URL https://pypl.github.io/IDE.html (accessed 1.30.22).

What is a Relational Database? – Amazon Web Services (AWS) [WWW Document], n.d. . Amaz. Web Serv. Inc. URL https://aws.amazon.com/relational-database/ (accessed 1.29.22).

What is PostgreSQL? – Amazon Web Services [WWW Document], n.d. . Amaz. Web Serv. Inc. URL https://aws.amazon.com/rds/postgresql/what-is-postgresql/ (accessed 2.11.22).

Wren, J., n.d. WindowBuilder | The Eclipse Foundation [WWW Document]. URL https://www.eclipse.org/windowbuilder/ (accessed 2.11.22).

Your relational data. Objectively. - Hibernate ORM [WWW Document], n.d. . Hibernate. URL https://hibernate.org/orm/ (accessed 2.12.22).

Κατσαρός, Γ., 2012. Τεχνολογίες διαχείρισης υπολογιστικών υποδομών και εφαρμογών σε υπηρεσιοστρεφείς αρχιτεκτονικές και περιβάλλοντα Νεφών. Ε.Μ.Π., Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Αθήνα.

Στάικου, Τ., Παλλαδινού, Ν., 2015. Οι νέες δυνατότητες της Java 8. ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΔΥΤΙΚΗΣ ΕΛΛΑΔΑΣ, ΣΧΟΛΗ ΔΙΟΙΚΗΣΗ ΚΑΙ ΟΙΚΟΝΟΜΙΑΣ, Πάτρα.

4.2 API documentation

4.2.1 User Services

4.2.1.1 Log In

Description	Login of user
Method	POST
Endpoint	http://localhost:8080/internalAPI/user/login
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	-
Success Code	**200**
Success Response	{ “**user**” : [User JSON]}
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the user is not authenticated **406** – in case the user is already logged in **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes:	

4.2.1.2 Log Out

Description	Logout of user
Method	PUT
Endpoint	http://localhost:8080/internalAPI/user/logout
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	-
Body	{ “**user**” : [User JSON]}
Success Code	**200**
Success Response	-
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Not checking if the user is already logged out.* *Only Requesting users and admins have access to this service.*	

4.2.1.3 Register User

Description	Register New User
Method	PUT
Endpoint	http://localhost:8080/internalAPI/user/register
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json

Body	{
	"**username**":"",
	"**password**":"",
	"**firstName**":"",
	"**lastName**":"",
	"**email**":""
	}

Success Code	**200**
Success Response	{ “**user**” : User JSON}
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case of username already exists, the password is invalid or any of the rest of the provided info is invalid. **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Does not return the register user.* *Only admins have access to this service*	

4.2.1.4 Delete User

Description	Delete a User
Method	DELETE
Endpoint	http://localhost:8080/internalAPI/user/deleteUser
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	-

Body	{
	"**id**" :
	}

Success Code	**200**
Success Response	-

Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case username is blank or missing **500** – in case of an internal error or username not in database
Error Response	**error** (String): Information about error
Notes: *Admins can delete any user. Any user can delete himself.*	

4.2.1.5 Get Assigned Roles of User

Description	Get a Assigned roles of a user
Method	GET
Endpoint	http://localhost:8080/internalAPI/user/getAssignedRoles
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**user**” : User JSON}
Success Code	**200** **204 No Content –** in case user has no assigned roles (No Response)
Success Response	{“**roles**”:[Role JSON, Role JSON ,…]}
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Admins can see any user’s assigned roles.* *A User can see the assigned roles to him.*	

4.2.1.6 Assign Role to User

Description	Assign a Role to a User
Method	PUT
Endpoint	http://localhost:8080/internalAPI/user/assignRole
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””

Body Type	Content-Type: application/json
Response Type	-
Body	{“**user**” : User JSON , “**role**” : Role JSON }
Success Code	**200**
Success Response	-
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **409** – in case user already has the role **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Only Admins have access to this service.*	

4.2.1.7 Remove Role from User

Description	Remove a Role from a User
Method	DELETE
Endpoint	http://localhost:8080/internalAPI/user/removeRole
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	-
Body	{“**user**” : User JSON , “**role**” : Role JSON }
Success Code	**200**
Success Response	-
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Only Admins have access to this service.*	

4.2.1.8 Get Connected Users

Description	Get the connected users
Method	GET
Endpoint	http://localhost:8080/internalAPI/user/getConnectedUsers
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	-
Success Code	**200**
Success Response	{ “connectedUsers”: [User JOSN , User JSON, …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Only Admin Users have access to this service.*	

4.2.1.9 Set Online status of a User

Description	Sets the Online status of a user
Method	PUT
Endpoint	http://localhost:8080/internalAPI/user/setOnlineStatus
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{“**user**”: , “**online**” : }
Success Code	**200**
Success Response	-
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error

Notes: *Only User Admins have access to this service.*

4.2.1.10 Edit User

Description	Edit a User
Method	POST
Endpoint	http://localhost:8080/internalAPI/user/editUser
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**user**” : User JSON}
Success Code	**200**
Success Response	{ “**user**” : User JSON}
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **404** – in case there is no User matching the given User username **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users can edit their basic info and Admin can edit their and other’s basic info.* *Changeable Information: First and Last Name and E-mail.*	

4.2.1.11 Change Password

Description	Change the password of the given User
Method	POST
Endpoint	http://localhost:8080/internalAPI/user/changePassword
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**user**” : User JSON “**newPassword**” : “” }
Success Code	**200**
Success Response	{ “**user**” : User JSON}
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **404** – in case there is no User matching the given User username **406** – in case any of the mandatory values is blank or the new password is invalid. **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Any user can change his password, only Admins can change other User’s password.*	

4.2.1.12 Get User by ID

Description	Get a User entry bu its ID
Method	GET
Endpoint	http://localhost:8080/internalAPI/user/getUser
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**id**”: }
Success Code	**200**
Success Response	{ “**user**” : User JSON}
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request

	500 – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Only admins have access to this service.*	

4.2.1.13 Get Users

Description	Get all Database Users
Method	GET
Endpoint	http://localhost:8080/internalAPI/user/getUsers
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	-
Success Code	**200**
Success Response	{ “**databaseUsers**”: [User JOSN , User JSON, …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Only admins have access to this service.*	

4.2.1.14 Get Roles

Description	Get all Roles from the Database
Method	GET
Endpoint	http://localhost:8080/internalAPI/user/getRoles
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	-
Success Code	**200**
Success Response	{“**roles**”:[Role JSON, Role JSON ,…]}
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated

	403 – in case that the request user is not authorized for the request **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Only admins have access to this service.*	

4.2.2 Lemma Services

4.2.2.1 Get From Id

Description	Get lemma by Id
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getFromID
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**id**” : “” }
Success Code	**200**
Success Response	{ “lemma” : {Lemma Object} }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *All users have access to this service*	

4.2.2.2 Get All Lemmas

Description	Get all Lemmas
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getAll
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**pageable**” : {Pageable Object} }
Success Code	**200**
Success Response	{“**lemmas**” : [{Lemma Object}, {Lemma Object}, …]}

Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with 'Dictionary_Manager' role have access to this service. (Heavy duty service)*	

4.2.2.3 Initialize Pageable of Get All Lemmas

Description	Initialize Pagebale of Get All Lemmas
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getAll/initializePageable
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	-
Success Code	**200**
Success Response	{ "**pageable**" : {Pageable Object} }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with 'Dictionary_Manager' role have access to this service.*	

4.2.2.4 Get Lemmas By Word

Description	Get by word
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getByWord
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ "**word**" : "" }

Success Code	**200**
Success Response	{“**lemmas**” : [{Lemma Object}, … , {Lemma Object} }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *All users have access to this service*	

4.2.2.5 Get Lemmas by Unaccent Word

Description	Get by unaccent word
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getByUnaccentWord
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**wordUnaccent**” : “” }
Success Code	**200**
Success Response	{“**lemmas**” :[{Lemma Object}, {Lemma Object} …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *All users have access to this service* *Different Lemmas may have the same unaccent word (e.g. Αγγείο)*	

4.2.2.6 Get Lemmas by Stem

Description	Get by Stem
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getByStem
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””

Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ "**stem**" : "stemWord" } OR {"**stem**" : {Lemma Object} }
Success Code	**200**
Success Response	{"**lemmas**" :[{Lemma Object}, {Lemma Object} …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *All users have access to this service* *Stem can be either a word or a Lemma JsonObject*	

4.2.2.7 Get Stems given Lemma and Inflectional Type

Description	Get Stems
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getStems
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ "**lemma**" : {Lemma Object}, "**inflectionalType**" : {Inflectional Type Object} }
Success Code	**200**
Success Response	{"**stems**" :[{Lemma Object}, {Lemma Object} …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *All users have access to this service*	

4.2.2.8 Get All possible Stems

Description	Get Stems
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getPossibleStems
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**lemma**” : {Lemma Object} }
Success Code	**200**
Success Response	{“**stem**” :[{Lemma Object}, {Lemma Object} …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *All users have access to this service*	

4.2.2.9 Get Inflectional Types of a given Lemma

Description	Get Inflectional Types
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getInflectionalTypes
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**lemma**” : {Lemma Object} }
Success Code	**200**
Success Response	{ “**inflectionalTypes**” :[{Inflectional Type Object}, {Inflectional Type Object} …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *All users have access to this service*	

4.2.2.10 Get Inflectional Info of a given Lemma

Description	Get Inflectional Info of Lemma
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getInflectionalInfoOfLemma
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**lemma**” : {Lemma Object} }
Success Code	**200**
Success Response	{ “**inflectionalInfo**” :[{ “key1” : “value1”}, …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *All users have access to this service*	

4.2.2.11 Get Main Lemmas

Description	Get Main Lemmas
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getMainLemmas
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**pageable**” : {Pageable Object} }
Success Code	**200**
Success Response	{“**lemmas**” :[{Lemma Object}, {Lemma Object} …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error

Notes: *Users with 'Dictionary_Manager' role have access to this service.*

4.2.2.12 Initialize Pageable of Get Main Lemmas

Description	Initialize Pagebale of Get Main Lemmas
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getMainLemmas/initializePageable
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	-
Success Code	**200**
Success Response	{ "**pageable**" : {Pageable Object} }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with 'Dictionary_Manager' role have access to this service.*	

4.2.2.13 Get Lemmas by Inflectional Type

Description	Get By Inflectional Type
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getByInflectionalInfo
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json

Body	{ “**inflectionalType**” : {Inflectional Type} , “**pageable**” : {Pageable Object} }
Success Code	**200**
Success Response	{“**lemmas**” :[{Lemma Object}, {Lemma Object} …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with ‘Dictionary_Manager’ role have access to this service.*	

4.2.2.14 Initialize Pageable of Get Lemmas By Inflectional Type

Description	Initialize Pagebale of Get By Inflectional Type
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getByInflectionalInfo/initializePageable
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**inflectionalType**” : {Inflectional Type} }
Success Code	**200**
Success Response	{ “**pageable**” : {Pageable Object} }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with ‘Dictionary_Manager’ role have access to this service.*	

4.2.2.15 Get Dummy Lemmas

Description	Get Dummy Lemmas

Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getDummyLemmas
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**pageable**” : {Pageable Object} }
Success Code	**200**
Success Response	{ “**lemmas**” :[{Lemma Object}, {Lemma Object} …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with ‘Dictionary_Manager’ role have access to this service.*	

4.2.2.16 Initialize Pageable of Get Dummy Lemmas

Description	Initialize Pagebale of Get Dummy Lemmas
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getDummyLemmas/initializePageable
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	-
Success Code	**200**
Success Response	{ “**pageable**” : {Pageable Object} }

Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with 'Dictionary_Manager' role have access to this service.*	

4.2.2.17 Get Lemmas By Part of Speech

Description	Get by Part of Speech
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getByPoS
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ "**pageable**" : {Pageable Object} , "**partOfSpeech**" : {Part of Speech} }
Success Code	**200**
Success Response	{ "**lemmas**" :[{Lemma Object}, {Lemma Object} …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with 'Dictionary_Manager' role have access to this service.*	

4.2.2.18 Initialize Pageable of Get Dummy Lemmas

Description	Initialize Pagebale of Get Dummy Lemmas
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getByPoS/initializePageable
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json

Response Type	Content-Type: application/json
Body	-
Success Code	**200**
Success Response	{ "**pageable**" : {Pageable Object} }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with 'Dictionary_Manager' role have access to this service.*	

4.2.2.19 Get Lemmas by Phonetic

Description	Get by phonetic
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getByPhonetic
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ "**phonetic**" : "" }
Success Code	**200**
Success Response	{ "**lemmas**" :[{Lemma Object}, {Lemma Object} …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *All users have access to this service*	

4.2.2.20 Get Lemmas By Suffix

Description	Get by suffix
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getBySuffix
URL Parameters	-

HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**suffix**” : “”,“**pageable**” : {Pageable Object} }
Success Code	**200**
Success Response	{ “**lemmas**” :[{Lemma Object}, {Lemma Object} …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with ‘Dictionary_Manager’ role have access to this service.*	

4.2.2.21 Initialize Get Lemmas by Suffix pageable

Description	Initialize get by suffix pageable
Method	
Endpoint	http://localhost:8080/internalAPI/lemma/getBySuffix/initializePageable
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**suffix**” : “” }
Success Code	**200**
Success Response	{ “**pageable**” : {Pageable Object} }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error

Notes: *Users with 'Dictionary_Manager' role have access to this service.*

4.2.2.22 Get Suffixes of Lemma

Description	Get suffixes
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getSuffixes
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ "**lemma**" : {Lemma Object} }
Success Code	**200**
Success Response	{ "**suffixes**" :[{"suffix": "", "suffixType" : ""}, …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *All users have access to this service*	

4.2.2.23 Get Lemmas By Prefix

Description	Get by prefix
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getByPrefix
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ "**prefix**" : "","**pageable**" : {Pageable Object} }
Success Code	**200**
Success Response	{ "**lemmas**" :[{Lemma Object}, {Lemma Object} …] }

Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with 'Dictionary_Manager' role have access to this service.*	

4.2.2.24 Initialize Get Lemmas by Prefix pageable

Description	Initialize by prefix pageable
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getByPrefix/initializePageable
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ "**prefix**" : "" }
Success Code	**200**
Success Response	{ "**pageable**" : {Pageable Object} }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with 'Dictionary_Manager' role have access to this service.*	

4.2.2.25 Get Prefixes of Lemma

Description	Get Prefixes
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getPrefixes
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**lemma**” : {Lemma Object} }
Success Code	**200**
Success Response	{ “**prefixes**” :[{“prefix”: “”, “prefixType” : “”}, …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *All users have access to this service*	

4.2.2.26 Get Lemmas By Frequency Range

Description	Get by frequency range
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getByFrequencyRange
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**lowerBound**” : int , “**upperBound**” : int, “**pageable**” : {Pageable Object} }
Success Code	**200**
Success Response	{ “**lemmas**” :[{Lemma Object}, {Lemma Object} …] }

Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with 'Dictionary_Manager' role have access to this service.*	

4.2.2.27 Initialize Get by Frequency Range pageable

Description	Initialize by frequency range pageable
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/ getByFrequencyRange/initializePageable
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ "**lowerBound**" : int , "**upperBound**" : int }
Success Code	**200**
Success Response	{ "**pageable**" : {Pageable Object} }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with 'Dictionary_Manager' role have access to this service.*	

4.2.2.28 Get Lemmas By Frequency

Description	Get by frequency
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getByFrequency
URL Parameters	-

HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**frequency**” : int ,“**pageable**” : {Pageable Object} }
Success Code	**200**
Success Response	{ “**lemmas**” :[{Lemma Object}, {Lemma Object} …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with ‘Dictionary_Manager’ role have access to this service.*	

4.2.2.29 Initialize Get Lemmas by Frequency pageable

Description	Initialize by frequency pageable
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getByFrequency/initializePageable
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**frequency**” : int }
Success Code	**200**
Success Response	{ “**pageable**” : {Pageable Object} }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error

Error Response	**error** (String): Information about error
Notes: *Users with 'Dictionary_Manager' role have access to this service.*	

4.2.2.30 Get Lemmas By Number of Syllables

Description	Get by syllables number
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getBySyllablesNum
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ "**number**" : 0 ,"**pageable**" : {Pageable Object} }
Success Code	**200**
Success Response	{ "**lemmas**" :[{Lemma Object}, {Lemma Object} …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with 'Dictionary_Manager' role have access to this service.*	

4.2.2.31 Initialize pageable of Get By Number of Syllables

Description	Initialize get by syllables number pageable
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/ getBySyllablesNum/initializePageable
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ "**number**" : 0 }
Success Code	**200**

Success Response	{ “**lemmas**” :[{Lemma Object}, {Lemma Object} …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with ‘Dictionary_Manager’ role have access to this service.*	

4.2.2.32 Get By Number of Phonemes

Description	Get by phonemes number
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getByPhonemesNum
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**number**” : 0 ,“**pageable**” : {Pageable Object} }
Success Code	**200**
Success Response	{ “**lemmas**” :[{Lemma Object}, {Lemma Object} …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with ‘Dictionary_Manager’ role have access to this service.*	

4.2.2.33 Initialize pageable of Get By Number of Phonemes

Description	Initialize get by phonemes number pageable
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/ getByPhonemesNum/initializePageable
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**number**” : 0 }
Success Code	**200**
Success Response	{ “**lemmas**” :[{Lemma Object}, {Lemma Object} …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with ‘Dictionary_Manager’ role have access to this service.*	

4.2.2.34 Get By Number of Characters

Description	Get by characters number
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getByNumberOfCharacters
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**number**” : 0 ,“**pageable**” : {Pageable Object} }
Success Code	**200**
Success Response	{ “**lemmas**” :[{Lemma Object}, {Lemma Object} …] }

Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with 'Dictionary_Manager' role have access to this service.*	

4.2.2.35 Initialize pageable of Get By Number of Characters

Description	Initialize get by characters number pageable
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/ getByNumberOfCharacters/initializePageable
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ "**number**" : 0 }
Success Code	**200**
Success Response	{ "**lemmas**" :[{Lemma Object}, {Lemma Object} …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with 'Dictionary_Manager' role have access to this service.*	

4.2.2.36 Get Lemmas By Language

Description	Get by language
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getByLanguage
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""

Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**language**” : “Greek” ,“**pageable**” : {Pageable Object} }
Success Code	**200**
Success Response	{ “**lemmas**” :[{Lemma Object}, {Lemma Object} …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with ‘Dictionary_Manager’ role have access to this service.*	

4.2.2.37 Initialize Get By Language pageable

Description	Initialize by language
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getByLanguage/initializePageable
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**language**” : “Greek” }
Success Code	**200**
Success Response	{ “**pageable**” : {Pageable Object} }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error

Error Response	**error** (String): Information about error
Notes: *Users with 'Dictionary_Manager' role have access to this service.*	

4.2.2.38 Get Part of Speech entities

Description	Get POS entities
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getPOS
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	-
Success Code	**200**
Success Response	{"**posEntities**" : [{Part of Speech Object}…]}
Error Code	**401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with one of the 'Lemma_Editor', 'Linguistic_Content_Manager' or 'Lemma_Creator' role have access to this service.*	

4.2.2.39 Get Prefixes

Description	Get Prefixes
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getPrefix
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ "**prefix**": "" }
Success Code	**200**
Success Response	{"**prefixPair**: [{Prefix Pair Object},…] }

Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with one of the 'Lemma_Editor', 'Linguistic_Content_Manager' or 'Lemma_Creator' role have access to this service.*	

4.2.2.40 Get Suffixes

Description	Get Suffixes
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getSuffix
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ "**suffix**": "" }
Success Code	**200**
Success Response	{"**suffixPair**: [{Suffix Pair Object}, …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with one of the 'Lemma_Editor', 'Linguistic_Content_Manager' or 'Lemma_Creator' role have access to this service.*	

4.2.2.41 Get Inflectional Info of Inflectional Type

Description	Get inflectional info of type
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getInflectionalInfoOfType
URL Parameters	-

HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**inflectionalType**”: {Inflectional Type Object} }
Success Code	**200**
Success Response	{ “**inflectionalPairs”**: { “key1” : “value1”, … } }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with one of the ‘Lemma_Editor’, ‘Linguistic_Content_Manager’ or ‘Lemma_Creator’ role have access to this service.*	

4.2.2.42 Get Lemmas by Inflectional Info Values

Description	Get By Inflectional Info Values
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/getByInflectionalInfoValues
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**inflectionalPairs”**: { “key1” : “value1”, … } , “**pageable**” : {Pageable Object} }
Success Code	**200**
Success Response	{“**lemmas**” :[{Lemma Object}, {Lemma Object} …] }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error

Notes: *Users with 'Dictionary_Manager' role have access to this service.*

4.2.2.43 Initialize Pageable of Get Lemmas By Inflectional Info Values

Description	Initialize Pagebale of Get By Inflectional Info Values
Method	GET
Endpoint	http://localhost:8080/internalAPI/lemma/ getByInflectionalInfoValues/initializePageable
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ "**inflectionalPairs"**: { "key1" : "value1", … } }
Success Code	**200**
Success Response	{ "**pageable**" : {Pageable Object} }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with 'Dictionary_Manager' role have access to this service.*	

4.2.2.44 Insert Lemma

Description	insertLemma
Method	PUT
Endpoint	http://localhost:8080/internalAPI/lemma/insertLemma
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ "**word**" : "" , "**wordUnaccent**" : "", "**language**" : {Language Object} }
Success Code	**200**

Success Response	{ “**lemma**” : {Lemma Object}}
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with ‘Lemma Creator’ roles have access to this service.* *The combination of word, phonetic and part of speech is unique.*	

4.2.2.45 Insert Inflectional Type

Description	Insert Inflectional Type
Method	PUT
Endpoint	http://localhost:8080/internalAPI/lemma/insertInflectionalType
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**inflectionalPairs**”: {“**key1**”:”value”, … “**key**” : “value”}, “**language**” : “”}
Success Code	**200**
Success Response	{“**inflectionalTypes**” : {Inflectional Type Object} }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with one of the ‘Lemma_Editor’, ‘Linguistic_Content_Manager’ or ‘Lemma_Creator’ role have access to this service.* *Language: Greek, English*	

4.2.2.46 Insert Suffix Pair

Description	Insert Suffix Pair
Method	PUT
Endpoint	http://localhost:8080/internalAPI/lemma/insertSuffix
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**suffix**” : “”, “**suffixType**” : “”, “**language**” : {Language Object}}
Success Code	**200**
Success Response	{ “**suffixPair**” : {Suffix Pair Object} }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with one of the ‘Linguistic_Content_Manager’ role have access to this service.*	

4.2.2.47 Insert Prefix Pair

Description	Insert Prefix Pair
Method	PUT
Endpoint	http://localhost:8080/internalAPI/lemma/insertPrefix
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**prefix**” : “”, “**prefixType**” : “”, “**language**” : {Language Object}}
Success Code	**200**
Success Response	{ “**prefixPair**” : {Prefix Pair Object} }

Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with one of the 'Linguistic_Content_Manager' role have access to this service.*	

4.2.2.48 Insert Stem with Inflectional Info Type

Description	Insert Stem with Inflectional info type
Method	PUT
Endpoint	http://localhost:8080/internalAPI/lemma/insertInflInfoStem
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ "**lemma**" : {Lemma Object}, "**inflectionalType**" : {Inflectional Type Object}, "**stem**" : {Lemma Object}, "**isMain**" : Boolean }
Success Code	**200**
Success Response	{ "**inflectionalTypeMapping**" : {Lemma-Inflectional-Stem map Object} }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with 'Dictionary_Manager', 'Lemma Editor' roles have access to this service.*	

4.2.2.49 Update Lemma

Description	Update lemma
Method	POST
Endpoint	http://localhost:8080/internalAPI/lemma/updateLemma
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**lemma**” : {Lemma Object} }
Success Code	**200**
Success Response	{ “**lemma**” : {Lemma Object} }
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with ‘Lemma Editor’ roles have access to this service.*	

4.2.2.50 Remove Lemma by ID

Description	Remove by id
Method	DELETE
Endpoint	http://localhost:8080/internalAPI/lemma/removeByID
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**id**” : 0 }
Success Code	**200**
Success Response	-
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error

Error Response	**error** (String): Information about error
Notes: *Users with 'Dictionary_Manager' role have access to this service.*	

4.2.2.51 Delete Lemma-Inflectional Type-Stem pairing

Description	Delete Lemma-Inflectional mapping
Method	DELETE
Endpoint	http://localhost:8080/internalAPI/lemma/deleteLemmaInflectionalTypeStem
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ "**inflectionalTypeMapping**" : {Map Lemma Inflectional Type Object} }
Success Code	**200**
Success Response	-
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with 'Dictionary_Manager' role have access to this service.*	

4.2.2.52 Delete Lemma

Description	Delete Lemma
Method	DELETE
Endpoint	http://localhost:8080/internalAPI/lemma
URL Parameters	-
HTTP Headers	Authentication: "**username**" :"", "**password**":""
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json

Body	{ “**lemma**” : {Lemma Object} }
Success Code	**200**
Success Response	-
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with ‘Dictionary_Manager’ role have access to this service.*	

4.2.2.53 Delete Prefix

Description	Delete Prefix
Method	DELETE
Endpoint	http://localhost:8080/internalAPI/lemma/deletePrefix
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**prefixPair**” : {Prefix Pair Object} }
Success Code	**200**
Success Response	-
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with ‘Linguistic_Content_Manager’ role have access to this service.*	

4.2.2.54 Delete Suffix

Description	Delete Suffix
Method	DELETE
Endpoint	http://localhost:8080/internalAPI/lemma/suffixPair
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	{ “**suffixPair**” : {Suffix Pair Object} }
Success Code	**200**
Success Response	-
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank **500** – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *Users with ‘Linguistic_Content_Manager’ role have access to this service.*	

4.2.3 API Appendix

4.2.3.1 Prototype

Description	
Method	
Endpoint	http://localhost:8080/internalAPI/lemma/
URL Parameters	-
HTTP Headers	Authentication: ”**username**” :””, ”**password**”:””
Body Type	Content-Type: application/json
Response Type	Content-Type: application/json
Body	
Success Code	**200**
Success Response	
Error Code	**400** – in case that the format of the request body is not valid or mandatory values are missing **401** – in case that the request user is not authenticated **403** – in case that the request user is not authorized for the request **406** – in case any of the mandatory values is blank

	500 – in case of an internal error
Error Response	**error** (String): Information about error
Notes: *All users have access to this service* *Users with 'Dictionary_Manager' role have access to this service.*	

4.2.3.2 User JSON

User (JsonObject)
id (BigInteger): the id **username** (String): the username of the user **password** (Integer): the password of the user **firstName** (String): the First Name of the user **lastName** (String): the Last Name of the user **emai** (String)l: E-mail of the user **online** (Boolean): the online status of the user **lastLogin** (TimeStamp): the last time the user Logged In

4.2.3.3 Role JSON

Role (JsonObject)
id (Small Integer): the id **role** (String): The name of the role. **Comment** (String): Comment on the role.

4.2.3.4 Lemma Object

Lemma (JsonObject)
id (Big Integer): the id **word** (String): lemmas word **wordUnaccent** (String): word without accent {**Language Object**} **gpc** (String Array): grapheme – phoneme correspondence **ipa** (String): the phonetic pronounce of the word, written on international phonetic alphabet **cv** (String): consonant-verb sequence **syllabification** (String Array): syllables of the word **numOfCharacters** (Small Integer): number of the characters the word has **numOfSyllables** (Small Integer): number of syllables the word has **numOfPhonemes** (Small Integer): number of phonem es the word has **frequency** (Small Integer): how frequent is the word

{**Part of Speech Object**} **dummyData** (Boolean): dummy lemma declaration

4.2.3.5 Language Object

Language (JsonObject)
id (Small Integer): the id **language** (String): name of the language

4.2.3.6 Part of Speech Object

Part of Speech (JsonObject)
id (Big Integer): the id **pos** (String): name of the part of speech

4.2.3.7 Inflectional Type Object

Inflectional Type (JsonObject)
id (Big Integer): the id **partOfSpeech**: {**Part of Speech Object**} {**Language Object**}

4.2.3.8 Pageable Object

Pageable (JsonObject)
startingIndex (int): the index of the first item of the query **pageSize** (int): number of items each page will contain (matches the number of items the query will return) **currentPage** (int): indicator of the current page **numOfValues** (int): total values that the query returns **totalPages** (int): Number of pages that the query breaks down to, regarding page size and number of values

4.2.3.9 Lemma-Inflectional Type-Stem Mapping object

MapLemmaInflectionalType (JsonObject)
lemmaId {Lemma Object} : the lemma entity of interest **inflectionalType** {Inflectional Type Object} : The inflectional type that matches the lemma of interest **stemId** {Lemma Object} : the stem entity that matches the lemma and inflectional type of interest **isMain** (Boolean) : if the above combination regards a main lemma

4.2.3.10 Suffix Pair Object

Suffix Pair (JsonObject)
id (Short Integer): the id **suffix**: {Suffix Object} **type**: {SuffixType Object}

4.2.3.11 Suffix Object

Suffix (JsonObject)
id (Short Integer): the id **suffix** (String): the suffix **language**: {Language Object}

4.2.3.12 SuffixType Object

SuffixType (JsonObject)
id (Short Integer): the id **code (**String**)**: description regarding the paired suffix

4.2.3.13 Prefix Pair Object

Prefix Pair (JsonObject)
id (Short Integer): the id **prefix**: {Prefix Object} **type**: {PrefixType Object}

4.2.3.14 Prefix Object

Prefix (JsonObject)
id (Short Integer): the id **prefix** (String): the prefix **language**: {Language Object}

4.2.3.15 PrefixType Object

PrefixType (JsonObject)
id (Short Integer): the id **code (**String**)**: description regarding the paired prefix