ΕΞΑΓΩΓΗ ΚΑΙ ΟΠΤΙΚΟΠΟΙΗΣΗ ΕΚΔΟΣΕΩΝ ΤΟΥ ΣΧΗΜΑΤΟΣ ΔΕΔΟΜΕΝΩΝ ΤΥΠΟΥ JSON

Θωμάς Σιώζος

Διπλωματική Εργασία

Επιβλέπων: Π. Βασιλειάδης

Ιωάννινα, Ιούνιος 2020

ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

Ευχαριστίες

Ευχαριστίες προς κάθε ενδιαφερόμενο (προαιρετικό κεφάλαιο).

Ημερομηνία

Συγγραφέας

Περίληψη στα ελληνικά

Η παρούσα διπλωματική αφορά το σχεδιασμό και την κατασκευή ενός εργαλείου λήψης, επεξεργασίας, εξαγωγής και οπτικοποίησης της εσωτρικής δομής δεδομένων τύπου JSON. Συγκεκριμένα, με το ως άνω εργαλείο οι εκδόσεις που ακολουθούν τα JSON αντικείμενα στο αρχείο εισόδου εξάγονται και οπτικοποιούνται. Για την κατασκευή και λειτουργία του εργαλείου χρησιμοποιήθηκαν βιβλιοθήκες για την εξαγωγή των JSON αντικειμένων από το αρχείο εισόδου, αλγόριθμοι για την επεξεργασία και την σύγκριση τους. Έπειτα, δημιουργήθηκε γραφικό περιβάλλον στο οποίο ο χρήστης μπορεί να εισάγει το αρχείο JSON, να ελέγξει το αποτέλεσμα σε οπτική μορφή, να αποθηκεύσει το project και τέλος να φορτώσει ένα project που ήδη υπάρχει. Τέλος, για την οπτικοποίηση των εκδόσεων του αρχείου δημιουργήθηκαν δύο αρχεία, της μορφής HTML και Javascript. Το εργαλείο αυτό καθίσταται ιδιαίτερα χρήσιμο στην πράξη, καθώς τόσο η δραματική αύξηση των δεδομένων που διακινούνται στο διαδίκτυο, όσο και η ανάγκη της χρήσης βάσεων δεδομένων από πολλά συστήματα αντιμετωπίζεται με τη δημιουργία και χρήση των αρχείων JSON που μπορούν να περιέχουν όλα τα δεδομένα τους σε ένα αρχείο.

Λέξεις Κλειδιά: JSON, Jackson, JSON Objects, schema versions.

Abstract

This thesis concerns the design and implementation via which, a user may download, process, export and visualize the versions of the internal structure, or schema, of JSON data, stored in a single file. To design and operate the tool, libraries for extracting JSON objects from the document and algorithms for processing and comparison are used. Moreover a GUI (graphical user interface) we designed and implemented a tool that allows inserting the JSON document, controlling the result in visual form, saving the project and loading an existing project. Finally, the visualization of the documents' versions is achieved by designing two documents in html and JavaScript form. The aforementioned tool can prove to be very useful, as on the one hand, the dramatic increment of data traffic in Internet, and, on the other hand, the necessity of using databases by many systems are problems that may be solved by designing and using JSON documents that incorporate all their data in one document.

Keywords: JSON, Jackson, JSON Objects, schema versions.

Πίνακας περιεχομένων

Κεφάλαιο 1. Εισαγωγή

1.1 Αντικείμενο της διπλωματικής

Το JavaScript Object Notation (JSON) είναι ένα ανοικτό στάνταρντ για την οργάνωση των δεδομένων με τρόπο που η περιγραφή της δομής και η τιμή των δεδομένων συνυπάρχουν ταυτοχρόνως μέσα σε ένα αρχείο, και μάλιστα με τρόπο αναγνωρίσιμο από τον άνθρωπο. Ο Douglas Crockford ήταν ο πρώτος που καθόρισε και δημοσίευσε το JSON format. Τα JSON είναι ένα υποσύνολο της JavaScript και είναι σύνηθες το φαινόμενο να χρησιμοποιείται σε αυτήν και ας είναι language-independent data format. Επίσης, χρησιμοποιεί κανόνες που είναι παρόμοιοι με της γλώσσες κατηγορίας C περιέχοντας C, C++,C#, Java, JavaScript και πολλές άλλες. Ωστόσο, υπάρχει διαθέσιμος κώδικας από πολλές γλώσσες προγραμματισμού για parsing και generating. Τα JSON αρχεία αποθηκεύονται με την κατάληψη ".json". Όσο για τους τύπους δεδομένων και την σύνταξη που χρησιμοποιεί έχουμε τους βασικούς τύπους δεδομένων:

- **Number**: Είναι ένας δεκαδικός αριθμός, αλλά δεν γίνεται να χρησιμοποιεί non-numbers και NaN.
- **String**: Μια πρόταση από μηδέν ή περισσότερους χαρακτήρες Unicode, έχοντας στην αρχή και στο τέλος διπλά εισαγωγικά ("").
- **Boolean**: Μπορεί να πάρει τιμές true ή false.
- **Array**: Μια προκαθορισμένη λίστα με μηδέν ή περισσότερες τιμές που ο τύπος κάθε μία από αυτές μπορεί να ναι οτιδήποτε. Τους πίνακες τους γράφουμε με bracket στην αρχή και στο τέλος και κόμμα ανάμεσα σε κάθε στοιχείο.
- **Object**: Μια μη προκαθορισμένη συλλογή από name-value ζευγάρια όπου τα names ή αλλιώς και keys είναι strings. Τα objects γράφονται με curly brackets στην αρχή και στο τέλος, χρησιμοποιούμε κόμμα ανάμεσα από κάθε ζευγάρι name-value και ανάμεσα στο ζευγάρι άνω και κάτω τελεία (:).
- **null**: Μια κενή τιμή.

Το whitespace επιτρέπεται αλλά αγνοείται ανάμεσα ή τριγύρω από τα elements. Τέσσερεις συγκεκριμένοι χαρακτήρες θεωρούνται whitespace στα JSON που είναι οι: space, horizontal tab, line feed, carriage return. Ακολουθεί ένα παράδειγμα JSON που περιγράφει ένα άτομο.

```
{
        “firstName”: “John”,
        “lastName”: “Smith”,
        “isAlive”: true,
        “age”: 27,
        “adress”: {
                “streetAddress”: “21 2nd Street”,
                “city”: “New York”,
                “state”: “NY”,
                “postalCode”: “10021-3100”
        },
        “phoneNumbers”: [
                {
                        “type”: “home”,
                        “number”: “212 555-1234”
                },
                {
                        “type”: “office”,
                        “number”: “646 555-4567”
                },
                {
                        “type”: “mobile”,
                        “number”: “+91 9629787781”
                }
        ],
        “children”: [],
        “spouse”: null
}
```

1.2 Οργάνωση του τόμου

Η συγκεκριμένη διπλωματική εργασία αποτελείται από πέντε κεφάλαια τα οποία αναλύονται παρακάτω.

Στο κεφάλαιο 2 περιγράφονται ο στόχος της εργασίας, οι σχετικές εργασίες και τεχνολογίες σχετικά με την εξαγωγή JSON αντικειμένων στην γλώσσα προγραμματισμου που χρησιμοποιήσαμε στην εργασία, καθώς και η σύγκριση διαθέσιμων βιβλιοθηκών για την επεξεργασία JSON δεδομένων και οι λόγοι για τους οποίους τελικά επιλέξαμε τη βιβλιοθήκη Jackson για την ανάπτυξη του εργαλείου μας. Επιπλέον, στο κεφάλαιο 2 γίνεται και η ανάλυση των απαιτήσεων του συστήματος.

Στο κεφάλαιο 3 περιγράφεται η σχεδιάση και αρχιτεκτονική λογισμικού με την χρήση UML διαγραμμάτων. Στην συνέχεια, περιγράφεται ο έλεγχος που έγινε ώστε να δούμε αν μεμονωμένα σημεία του συστήματος λειτουργούν με τον τρόπο που θέλουμε. Στο τέλος του κεφαλαίου υπάρχει η περιγραφή για τις λεπτομέρεις εγκατάστασης και υλοποίησης του συστήματος.

Στο κεφάλαιο 4 περιγράφεται η μεθοδολογία πειραματισμού, τι αρχεία εισόδου βάλαμε ώστε να πάρουμε τα αποτελέσματα που θέλαμε ώστε να αναλύσουμε την επίδοση και τις πληροφορίες που υπήρχαν στα δεδομένα. Επιπλέον, υπάρχει ένα γράφημα το οποίο παρουσιάζει τα αποτελέσματα τα οποία εξήχθησαν από τα πειράματα μας.

Στο κεφάλαιο 5 περιγράφονται τα συμπεράσματα που βγάλαμε μετά τα πειράματα που έγιναν και παρουσιάζονται αναλυτικά οι μελοντικές επεκτάσεις που θα μπορούσαν να γίνουν στο λογισμικό με κάποιες αλλαγές.

Κεφάλαιο 2. Περιγραφή Θέματος

2.1 Στόχος της εργασίας

Ο στόχος της παρούσας διπλωματικής εργασίας είναι να κατασκευαστεί ένα σύστημα το οποίο θα επιτρέπει στον χρήστη την εισαγωγή ενός αρχείου δεδομένων τύπου JSON με αποτέλεσμα να προβάλλονται οπτικοποιημένα το schema των δεδομένων, καθώς και οι αλλαγές που πραγματοποιήθηκαν στη διάρκεια του οποίου οι διαχειριστές των δεδομένων είτε μπορεί να πρόσθεσαν, να αφαίρεσαν τιμές ή να μετέβαλλαν το τύπο των τιμών.

Συγκεκριμένα οι απαιτήσεις του συστήματος οργανώνονται ως εξής:

- Θα πρέπει ο χρήστης να μπορεί να φορτώσει στο σύστημα το αρχείο δεδομένων τύπου JSON που θέλει να εισάγει ώστε να ξεκινήσει η διαδικασία εξαγωγής του σχήματος των δεδομένων. Αντικείμενο της διπλωματικής είναι η εξαγωγή του σχήματος δεδομένων, καθώς και των πολλαπλών εκδόσεων αν υπάρχουν μέσα στον χρόνο.
- Ο χρήστης θα μπορεί να δει τα αποτελέσματα των σχημάτων δεδομένων και των διαφορετικών εκδόσεων.
- Τέλος, τελειώνοντας την επεξεργασία του αρχείου που εισήχθη από τον χρήστη, θα παρέχεται η επιλογή να δει τα χαρακτηριστικά που παρουσιάζει η εξέλιξη του σχήματος του JSON αρχείου.

2.2 Σχετικές εργασίες και τεχνολογίες

Υπάρχουν αρκετές βιβλιοθήκες που αποτελούν εργαλεία για χρήστες που επιθυμούν να επεξεργαστούν αρχεία τύπου JSON προγραμματίζοντας σε γλώσσα java. Οι πιο δημοφιλείς από αυτές είναι οι mJson. JSON.simple, JSON-P, GSON, Jackson, με τις GSON και Jackson και παρουσιάζουν αρκετές ομοιότητες μεταξύ τους. Όπως θα δούμε παρακάτωμ η mJSON είναι «μικρή» βιβλιοθήκη και η GSON και Jackson οι βιβλιοθήκες με τις περισσότερες λειτουργίες. Οι JSON.simple και JSON-P έχουν παρόμοια λειτουργικότητα, οπότε αρχικά ασχοληθήκαμε με mJSON και JSON-P και έπειτα με τις δύο μεγαλύτερες GSON, Jackson.

2.2.1 mJson

https://bolerio.github.io/mjson/

Η mJson είναι μία αρκετά μικρή βιβλιοθήκη με πολύ συνοπτικό API. Σε αυτήν την βιβλιοθήκη υπάρχει μια κλάση με όλες τις μεθόδους που μπορεί κάποιος να χρησιμοποιήσει. Σε αντίθεση με άλλες JSON βιβλιοθήκες τις οποίες θα δούμε παρακάτω, επικεντρώνεται στο να διαχειρίζεται τα JSON structures στην Java, υποστηρίζοντας parsing από JSON αντικείμενα σε Java αντικείμενα και το ανάποδο. Ο κύριος στόχος αυτής της βιβλιοθήκης είναι να επιτρέπει στον προγραμματιστή να δουλεύει με JSON στην Java όπως δουλεύει κανείς σε JavaScript. Μερικά χαρακτηριστικά της είναι:

- Πλήρης υποστήριξη του JSON Schema Draft 4 validation, ένα πρότυπο-δημιουργία του IETF- το οποίο παρέχει ένα σχήμα για την επαλήθευση ενός JSON αντικειμένου που περιέχει τον αναμενόμενο τύπο εντός του σχήματος.
- ‘Ενας μοναδικός τύπος μεταβλητών, τα πάντα είναι JSON και δεν χρειάζεται type casting.
- Γρήγορο parsing.
- Περιεκτικές μεθόδους για διάβασμα, τροποποίηση, αντιγραφή και συγχώνευση.
- Μέθοδούς για έλεγχο του τύπου των τιμών και πρόσβαση στον έλεγχο τους.
- Ένα java source file για ολόκληρη την βιβλιοθήκη, χωρίς εξωτερικές εξαρτήσεις.

2.2.2 JSON-P

https://javaee.github.io/jsonp/

Το JSON Processing(JSON-P) είναι ένα Java API που επιτρέπει parsing ή δημιουργία JSON αρχείων. Το JSON-P είναι παρόμοιο με το JAXP, ένα Java API για επεξεργασία XML αρχείων. Το JSON-P είναι συμβατικό με τις κλάσεις της Java για τα δεδομένα και αρκετά εύκολο στην χρήση του. Το JSON-P έρχεται με τρία πακέτα:

- **javax.json**: Περιέχει ένα API για να επεξεργαζόμαστε πίνακες και αντικείμενα. Μπορούμε να πάρουμε τα περιεχόμενα από πίνακες ή αντικείμενα και να τα επεξεργαστούμε.
- **javax.json.spi**: Περιέχει ένα Service Provider Interface (SPI) το οποίο επιτρέπει στον προγραμματιστή να φτιάξει δικά του εργαλεία για την επεξεργασία των JSON αντικειμένων πέρα από defaults που προσφέρει από μόνη της η βιβλιοθήκη.
- **javax.json.stream:** Περιέχει stream για την ανάγνωση/δημιουργία JSON αρχείων. Στην ανάγνωση δεν επιτρέπει επεξεργασία των δεδομένων και μόνο ανάγνωση προς μία κατεύθυνση. Στην δημιουργία φτιάχνει ζευγάρια από key : value και πίνακες με τις τιμές τους και κόμμα ανάμεσα.

2.2.3 JSON.simple

https://code.google.com/archive/p/json-simple/

Το Json.simple συγκαταλέγεται στα toolkits και χρησιμοποιείται για διάβασμα και δημιουργία JSON αρχείων. Το Json.simple είναι μικρό, αρά και γρήγορο όπως θα δούμε παρακάτω στα πειράματα. Μερικές δυνατότητες είναι υποστήριξη της μετατροπής των δεδομένων σε data structures της java, υψηλή απόδοση και δεν εξαρτάται από εξωτερικές βιβλιοθήκες. Στο Json.simple περιέχονται δύο πακέτα:

- **org.json.simple:** Περιέχει τις κλάσεις που είναι υπεύθυνες για την διαχείριση της μετατροπής των JSON nodes σε data structures της Java.
- **org.json.simple.parser:** Περιέχει τις κλάσεις που βοηθούν στην ανάλυση των JSON αντικειμένων.

2.2.4 GSON

https://github.com/google/gson

Η GSON είναι μια Java-based βιβλιοθήκη την οποία δημιούργησε η Google για προσωπική της χρήση και μετά την παραχώρησε για δημόσια χρήση. Η GSON χρησιμοποιείται για να μετατρέψει ένα JSON αντικείμενο σε ένα ισοδύναμο αντικείμενο της Java και το ανάποδο. Υποστηρίζει απλές μεθόδους όπως toJson() και fromJson() για να μετατρέπει ο προγραμματιστής Java αντικείμενα σε JSON αντικείμενα και το αντίστροφο, να διαχειρίζεται collections, generic types και nested classes. Επίσης, υποστηρίζει serialization/deserialization. Το serialization είναι ο τρόπος να δημιουργήσουμε JSON αντικείμενα μέσω Java αντικειμένων και το deserialization το ανάποδο. Στην περίπτωση που γνωρίζουμε από πριν τα πεδία που υπάρχουν στα δεδομένα των JSON αντικειμένων, χωρίς αυτά να αλλάζουν μέσα στον χρόνο, η GSON επιτρέπει στον προγραμματιστή να έχει τον πλήρη έλεγχο της όλης επεξεργασίας των JSON αντικειμένων. Αυτό σημαίνει, ότι δίνεται η δυνατότητα ο προγραμματιστής να χρησιμποποιήσει POJO κλάσεις για την χαρτογράφηση των JSON αντικειμένων. Ένα απλό deserialization παράδειγμα είναι όταν έχουμε JSON αντικείμενα με πεδία name, last name, age τότε ο προγραμματιστής μπορεί να φτιάξει μια κλάση Person με τα πεδία αυτά ώστε όταν περνάει το JSON αντικείμενο να χρησιμοποιείται η κλάση Person και να αποθηκεύονται αυτόματα στα πεδία οι τιμές.

Στην GSON θα βρούμε τα εξής πακέτα:

- com.google.gson: Προσφέρει πρόσβαση στην GSON, την βασική κλάση για να δουλέψει κάποιος με την βιβλιοθήκη.
- com.google.gson.annotations: Προσφέρει annotation τύπους προς χρήση.
- com.google.gson.reflect: Διαχειρίζεται πληροφορίες από generic τύπους.
- com.google.gson.stream: Προσφέρει κλάσεις για διάβασμα/γράψιμο από/σε JSON τύπους δεδομένων.

Μερικά χαρακτηριστικά της GSON είναι:

- Προσφέρει απλές μεθόδους toJson() και fromJson() για να μετατρέψουμε Java αντικείμενα σε JSON και το ανάποδο.
- Deserialization/Serialization.
- Υποστηρίζει Java Generics.
- Υποστηρίζει περίπλοκα αντικείμενα.

2.2.5 Jackson

https://github.com/FasterXML/jackson

Η Jackson είναι μια βιβλιοθήκη της Java με υψηλή απόδοση στο να διαχειρίζεται JSON δεδομένα και θεωρείται και η πιο διάσημη βιβλιοθήκη σύμφωνα με την χρήση στο Github. Η Jackson ενώ χρησιμοποιεί διαφορετικούς όρους για να ονομάσει τις λειτουργίες της παρουσιάζει πολλές ομοιότητες με την GSON. Στη διάρκεια των δοκιμών που εκτελέσαμε, παρατηρήσαμε ότι η Jackson δεν είναι τόσο φιλική, λόγω του αυξημένου μεγέθους της, όταν ξεκινήσαμε να διαβάζουμε ένα JSON αρχείο προσπαθώντας να μετατρέψουμε τα JSON αντικείμενα σε Java αντικείμενα δυσκολευτήκαμε να κατανοήσουμε τον τρόπο λειτουργίας της βιβλιοθήκης. Σε αντίθεση με την GSON η οποία είναι φιλική. Στην Jackson χρησιμοποιούνται και εδώ οι όροι serialization/deserialization που εξηγήθηκαν παραπάνω στην GSON τι ακριβώς είναι. Στην Jackson θα βρούμε τα εξής πακέτα:

- jackson-core: Περιλαμβάνει λειτουργίες για διάβασμα/γράψιμο από/σε JSON αντικείμενα.
- jackson-annotations: Περιλαμβάνει βασικά Jackson annotations.
- Jackson-databind: Εδώ βρίσκονται οι λειτουργίες για serialization/deserialization.

Πέρα απο τα πακέτα που χρησιμοποιούνται αυτούσια από την βιβλιοθήκη, έχουμε και εξωτερικά πακέτα που μπορεί κάποιος να χρησιμοποιήσει για να δουλέψει σε δεδομένα τύπου Avro, BSON, CBOR, CSV, Smile, Protobuf, XML, YAML και πολλά άλλα.

Μερικά χαρακτηριστικά της Jackson είναι:

- Χρησιμοποιεί δύο μεθόδους για serialization/deserialization όπως Object Mapper, JsonParser, JsonGenerator.
- Data-binding.
- Υποστηρίζει Java Generics.
- Υποστηρίζει περίπλοκα αντικείμενα.
- Υποστηρίζει δέντρα.
- Annotations.
- Java Generics.

2.2.6 Java Microbenchmark Harness(JMH)

https://openjdk.java.net/projects/code-tools/jmh/

Το Java Microbenchmark Harness (JMH) είναι το εργαλείο της java που χρησιμοποιήσαμε για να υλοποιήσουμε το benchmark και να πάρουμε τους χρόνους που χρειάζεται κάθε βιβλιοθήκη για να διαβάσει δεδομένα τύπου JSON, που βρίσκονται μέσα σε αρχεία εισόδου. Το εργαλείο JMH έχει κάποιες ορολογίες:

- **Trial:** Είναι οι φορές που θα εκτελεστεί το benchmark πριν ξεκινήσει την λειτουργία του ώστε να βγάλει πιο σωστά αποτελέσματα, τον όρο μπορούμε να τον αποκαλούμε και fork.
- **Warmup:** Για κάθε δοκιμή, ένας αριθμός από επαναλήψεις έχει οριστεί ως warmups. Αυτό είναι σημαντικό για να αποφύγουμε αυξομειώσεις ή παραλλαγές όταν ξεκινήσουμε να τρέχουμε τις πραγματικές μετρήσεις.
- **Iteration:** Είναι ο αριθμός που θα εκτελεστούν οι πραγματικές μετρήσεις που θα βγάλουν και τα αποτελέσματα στο τέλος που θα χρησιμοποιήσουμε.

Το εργαλείο μας επιτρέπει να δουλέψουμε με κάποια annotations που προσφέρει, τα οποία είναι:

- **Throughput:** Είναι για να μετρήσουμε πόσες φορές εκτελείται μία μέθοδος σε συγκεκριμένο χρονικό διάστημα. Η πιο δημοφιλής λειτουργία που χρησιμοποιείται είναι η AverageTime.
- **OutputTimeUnit:** Είναι για να καθορίσουμε σε τι μονάδα μέτρησης του χρόνου θα βγουν τα αποτελέσματα.
- **Benchmark:** Βάζουμε benchmark annotation στον κώδικα ή την μέθοδο που θα τρέξει το benchmark. Θα πρέπει να είναι public και οι παράμετροι να είναι κλάσεις της JMH όπως State, Control ή Blackhole.
- **Fork:** Εξηγήσαμε παραπάνω τι εννοούμε με τον όρο fork, οπότε αν για παράδειγμα γράψουμε @Fork(value = 5, warmups = 2) σημαίνει ότι έχουμε 5 forks και σε κάθε μία από αυτές 2 warmups.
- **Measurement:** Χρησιμοποιείται για να εισάγουμε τα χαρακτηριστικά του benchmark. Επιτρέπει να ορίσουμε πόσες φορές θα τρέξουν οι μετρήσεις και για πόσο χρόνο η κάθε μια. Για παράδειγμα @Measurement(iterations = 3, time = 1000, timeUnit = TimeUnit.MILLISECONDS), έχουμε 3 επαναλήψεις, κάθε μία από αυτές θα τρέξει για 1000 millisecond (1 second). Η προκαθορισμένη τιμή του timeUnit είναι τα seconds.
- **Warmup:** Αν γράψουμε @Warmup(iterations = 3, time =2) εννοούμε ότι θα κάνει 3 επαναλήψεις για 2 seconds η κάθε μια.

2.2.7 Benchmark

Για να αποφασίσουμε ποια βιβλιοθήκη θα χρησιμοποιήσουμε για το σκοπό της διπλωματικής εργασίας, οργανώσαμε μια συγκριτική μελέτη(benchmark). Πραγματοποιήθηκε με 2 σενάρια, στο ένα περάστηκαν μεγάλα αρχεία(25 MB) και στο δεύτερο περάστηκαν μικρά αρχεία (1 KB). Τα αρχεία υπάρχουν μέσα στον φάκελο του "benchmark". Για κάθε βιβλιοθήκη χρησιμοποιήσαμε BenchmarkMode = AverageTime, OutputTimeUnit = Timeunit.MILLISECONDS, Warmup = (iterations = 5, time = 5) και Measurement = (iterations = 10, time = 5, timeUnit = TimeUnit.MILLISECONDS). Τα αποτελέσματα του πειράματος με αρχείο εισόδου το μεγάλο σε μέγεθος αρχείο φαίνονται στον πίνακα 2.2.1.

BIG FILE (25.256kb) (ms/op)				
	JSON.simple	JSONP	GSON	JACKSON
1	608.630	918.254	469.102	406.364
2	629.594	895.896	458.992	397.598
3	601.786	898.090	453.495	396.582
4	624.479	896.872	463.046	401.167
5	630.198	897.469	456.440	394.900
6	613.333	897.282	460.155	393.910
7	615.867	901.147	456.915	393.270
8	625.719	894.348	465.858	398.416
9	673.665	894.297	453.392	399.134
10	607.742	896.939	472.911	395.026
AVERAGE	623.101	899.059	461.031	397.637

Πίνακας: 2.2.1. Αποτελέσματα πειράματος με μέγεθος αρχείου εισόδου 25.256kb

Όπως παρατηρούμε στον πίνακα 2.2.1 υπάρχουν μεγάλες διαφορές, εκτός από την GSON και την Jackson οι οποίες είναι πολύ κοντά, με την Jackson να είναι πιο γρήγορη.

Ας περάσουμε στα αποτελέσματα του πειράματος με αρχείο εισόδου το μικρό σε μέγεθος αρχείο που φαίνονται στον παρακάτω πίνακα 2.2.2.

SMALL FILE (1kb) (ms/op)				
	JSON.simple	JSONP	GSON	JACKSON
1	0.147	0.289	0.84	0.202
2	0.133	0.225	0.489	0.194
3	0.127	0.269	0.278	0.254
4	0.127	0.230	0.220	0.227
5	0.211	0.298	0.264	0.199
6	0.146	0.208	0.282	0.209
7	0.131	0.204	0.170	0.210
8	0.178	0.206	0.170	0.241
9	0.151	0.345	0.293	0.187
10	0.207	0.191	0.168	0.242
AVERAGE	0.156	0.246	0.252	0.217

Πίνακας: 2.2.2. Αποτελέσματα πειράματος με μέγεθος αρχείου εισόδου 1kb

Όπως παρατηρούμε στον πίνακα 2.2.2 ο νικητής για τα μικρά αρχεία είναι το JSON.simple toolkit και ακολουθεί η Jackson με το JSON-P API μετά και την GSON τελευταία.

Αν συγκρίνουμε και τους δύο πίνακες καταλήγουμε στο συμπέρασμα ότι:

- Αν θέλουμε να δουλέψουμε με μεγάλα σε μέγεθος αρχεία η Jackson είναι η πιο κατάλληλη και να ακολουθεί η GSON.
- Αν θέλουμε να δουλέψουμε με μικρά σε μέγεθος αρχεία το JSON.simple είναι το πιο κατάλληλο και φαίνεται ότι η Jackson έρχεται δεύτερη που φαίνεται αρκετά ενδιαφέρον αν αναλογιστούμε το μέγεθος της βιβλιοθήκης.

  Προφανώς όμως όταν θέλουμε να διαλέξουμε ποια βιβλιοθήκη θέλουμε να χρησιμοποιήσουμε δεν παίζει ρόλο μόνο ο χρόνος διαβάσματος των JSON δεδομένων, αλλά περισσότερα χαρακτηριστικά που είδαμε παραπάνω και θα αναλύσουμε παρακάτω.

2.2.8 Συμπέρασμα Σύγκρισης Βιβλιοθηκών

Ας δούμε για αρχή έναν περιληπτικό πίνακα για τις βιβλιοθήκες:

	JSON.simple	JSONP	GSON	Jackson
Easy-To-Use	√	√	√	
Conventional	√	√	√	√
Features			√	√
Community			√	√
Support Complex Objects			√	√

Πίνακας: 2.2.3

Αρχικά απορρίπτουμε την mJson επειδή δεν μπορεί να διαχειριστεί αρχεία με πολλά JSON αντικείμενα και έλλειψη στοίχισης, δηλαδή σε κάθε γραμμή του αρχείου υπάρχει ένα JSON αντικείμενο, το οποίο συμβαίνει στην πλειονότητα αλλά όχι σε όλα τα αρχεία. Παρατηρώντας τον πίνακα 2.2.3 βλέπουμε ότι το JSONP και JSON.simple πάσχουν λόγω έλλειψης πηγών και αλληλεπίδρασης μεταξύ μεταξύ χρηστών στο διαδίκτυο όταν προκύπτει κάποιο πρόβλημα, και, δεν έχουν αρκετές δυνατότητες σχετικά με περίπλοκα JSON αντικείμενα. Συγκρίνοντας τις δύο μεγαλύτερες βιβλιοθήκες, δηλαδή την Jackson και την GSON:

Τα θετικά της Jackson:

- Περισσότερες δυνατότητες.
- Serialization/deserialization.
- Καλύτερη ανάλυση στους τύπους δεδομένων σε ένα JSON αντικείμενο, όπως:
    - **TextNode:** Τιμές τύπου String.
    - **ArrayNode:** Για τους πίνακες.
    - **IntNode, DoubleNode κτλ:** Για τιμές που υποστηρίζονται και από την Java. Δεδομένο που θα μας χρησιμεύσει και θα μας βοηθήσει στην λειτουργικότητα της παρούσας διπλωματικής εργασίας.
    - **NullNode:** Για τιμές null.
- Δύο τρόπους για διάβασμα/γράψιμο από/σε JSON αντικείμενα.
- Υποστηρίζει δέντρα.
- Πιο γρήγορη και στα δύο μεγέθη αρχείων που εκχωρήθηκαν στα πειράματα που έγιναν.
- Community.
- Υποστηρίζει περίπλοκα JSON αντικείμενα.

Το μόνο αρνητικό που μπορούμε να σχολιάσουμε για την Jackson είναι ότι δεν είναι φιλική στην αρχή της χρήσης της.

Τα θετικά της Gson:

- Serialization/deserialization.
- Οι τύποι των τιμών επιστρέφονται ως:
    - **JsonPrimitive:** Για τα δεδομένα τύπου ακέραιοι, αλφαριθμητικά, πίνακες και άλλα.
    - **JsonNull:** Για τιμές null.
- Community.
- Υποστηρίζει περίπλοκα JSON αντικείμενα.

Τα αρνητικά της:

- Οι τύποι των τιμών δεν αναλύονται όπως την Jackson.
- Τελευταία στα πειράματα, με βάση τον χρόνο που χρειάζεται για να διαβάσει μικρά σε μέγεθος αρχεία που περιέχουν δεδομένα τύπου JSON.
- Δεν υποστηρίζει δέντρα.
- Μιά μέθοδος για διάβασμα/γράψιμο από/σε JSON αντικείμενα.

Στη βάση όλων αυτών που αναφέρθηκαν αποφασίζουμε ότι θα χρησιμοποιήσουμε την Jackson.

2.3 Ανάλυση απαιτήσεων

Στην παρούσα ενότητα περιγράφονται οι απαιτήσεις του συστήματος που κατασκευάσαμε.

Use Case: CreateNewProject

Περιγραφή και Στόχος:

Στόχος της περίπτωσης χρήσης είναι η δημιουργία από το χρήστη ενός νέου project.

Ηθοποιοί:

Χρήστης

Βασική Ροή Γεγονότων:

1. Η περίπτωση χρήσης ξεκινά όταν ο χρήστης πληκτρολογεί στην αντίστοιχη φόρμα το όνομα του project που επιθυμεί.

2. Αν το όνομα του project δεν υπάρχει.

    3.1. Το σύστημα προχωρά στην δημιουργία ενός φακέλου με το όνομα του project που εισήγαγε.

3. Αν το όνομα του project υπάρχει ήδη.

    4.1.1. Το σύστημα εμφανίζει μήνυμα λάθους.

    4.1.2. Το σύστημα ζητάει από το χρήστη διαφορετικό όνομα για το project που επιθυμεί να δημιουργήσει.

Use Case: DeleteExistingProject

Περιγραφή και Στόχος:

Στόχος της περίπτωσης χρήσης είναι η διαγραφή από το χρήστη ενός project που υπάρχει ήδη.

Ηθοποιοί:

Χρήστης

Βασική Ροή Γεγονότων:

1. Η περίπτωση χρήσης ξεκινά όταν ο χρήστης επιλέξει να διαγράψει κάποιο από τα ήδη υπάρχοντα project.

2. Το σύστημα εμφανίζει στον χρήστη τα πιθανά project για διγραφή.

3. Ο χρήστης επιλέγει project που επιθυμεί να διαγράψει.

Use Case : LoadData

Περιγραφή και Στόχος:

Στόχος της περίπτωσης χρήσης είναι η εισαγωγή από το χρήστη του αρχείου με τα δεδομένα τύπου JSON.

Ηθοποιοί:

Χρήστης

Βασική Ροή Γεγονότων:

1. Η περίπτωση χρήσης ξεκινά όταν ο χρήστης πληκτρολογεί στην αντίστοιχη φόρμα το όνομα του αρχείου με τα δεδομένα που επιθυμεί να εισάγει προς επεξεργασία.

2. Αν το αρχείο υπάρχει:

    2.1. Το σύστημα προχωρά στην επξεργασία των δεδομένων, δηλαδή στον έλεγχο των JSON αντικειμένων για διαφορές και εξαγωγή των εκδόσεων.

3. Αν το αρχείο δεν υπάρχει.

    3.1. Το σύστημα εμφανίζει μήνυμα λάθους.

3.2. Το σύστημα ζητάει από το χρήστη εκ νέου αρχείο που υπάρχει.

Use Case : SaveProject

Περιγραφή και Στόχος:

Στόχος της περίπτωσης χρήσης είναι η αποθήκευση του project.

Ηθοποιοί:

Χρήστης

Βασική Ροή Γεγονότων:

1. Η περίπτωση χρήσης ξεκινά όταν ο χρήστης αποφασίσει να αποθηκεύσει το project.

2. Το σύστημα στον φάκελο που έφτιαξε με το όνομα που επέλεξε ο χρήστης όταν δημιούργησε νέο project, αποθηκεύει το αρχείο εισόδου με τα δεδομένα, τις διαφορετικές εκδόσεις σχημάτων που βρέθηκαν και αρχείο με τα χαρακτηριστικά εξέλιξης του σχήματος .

3. Η περίπτωση χρήσης τερματίζεται.

Use Case : ViewSchemaFeatures

Περιγραφή και Στόχος:

Στόχος της περίπτωσης χρήσης είναι η εμφανιστεί στο χρήστη των χαρακτηριστικών που παρουσιάζει η εξέλιξη του σχήματος.

Ηθοποιοί:

Χρήστης

Βασική Ροή Γεγονότων:

1. Η περίπτωση χρήσης ξεκινά όταν ο χρήστης επιλέξει να δει τα χαρακτηριστικά που εμφανίζει η εξέλιξη του σχήματος.

2. Αν υπάρχει εξέλιξη στο σχήμα.

2.1. Το σύστημα εμφανίζει στο χρήστη σε κείμενο τα χαρακτηριστικά της εξέλιξης.

3. Αν δεν υπάρχει εξέλιξη στο σχήμα.

3.1. Το σύστημα εμφανίζει μήνυμα ότι δεν υπήρχαν διαφορετικές εκδοχές στο σχήμα.

Κεφάλαιο 3. Σχεδίαση & Υλοποίηση

3.1 Ορισμός προβλήματος και αλγόριθμοι επίλυσης

Στην παρούσα ενότητα περιγράφονται ο τύπος των αλλαγών του σχήματος δεδομένων τύπου JSON, ώστε να καταχωρηθεί ως έκδοση.

Πρωτού ξεκινήσουμε καλό θα ήταν να αναφέρουμε πότε ένα πεδίο άλλαξε σε σχέση με το προηγούμενο JSON αντικείμενο. Υπάρχουν 3 περιπτώσεις που το σύστημα αντιλαμβάνεται μια αλλαγή.

- **Αφαίρεση πεδίου:** Αν το πεδίο αφαιρέθηκε από το JSON αντικείμενο σε σχέση με την προηγούμενη έκδοση.
- **Πρόσθεση πεδίου:** Αν το πεδίο προστέθηκε στο JSON αντικείμενο σε σχέση με την προηγούμενη έκδοση.
- **Αλλαγή τύπου τιμής**: Αν η τιμή ενός πεδίου άλλαξε τύπο.

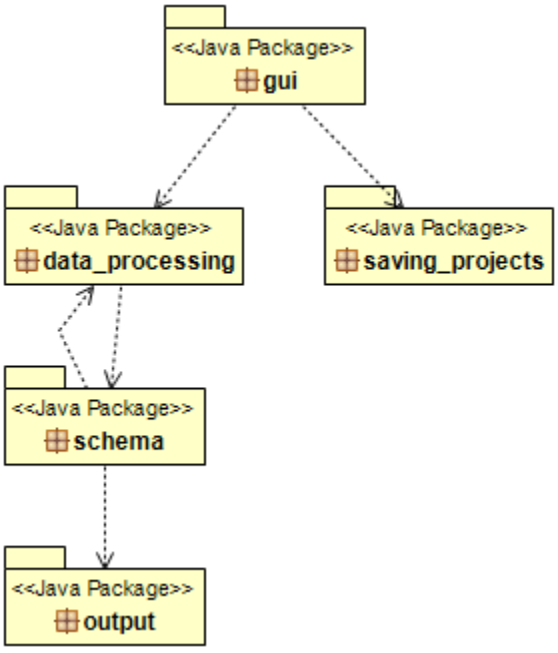
Ένα πεδίο λέγεται Node. Οι τύποι που χρησιμοποιούνται περισσότερο και θα αναφέρουμε παρακάτω στην βιβλιοθήκη που διαλέξαμε χρησιμοποιούν τους ορισμούς:

- **IntNode:** Όταν ο τύπος της τιμής του πεδίου είναι ακέραιος αριθμός.
- **DoubleNode:** Όταν ο τύπος της τιμής του πεδίου είναι ίδιος με αυτόν όταν μιλάμε για Double μεταβλητές στην Java.
- **ArrayNode:** Όταν ο τύπος της τιμής του πεδίου είναι ένας πίνακας ο οποίος μπορεί να περιέχει πεδία με οποιονδήποτε τύπο τιμής υποστηρίζεται.
- **ObjectNode:** Όταν ο τύπος της τιμής του πεδίου είναι JSON αντικείμενο, το βασικό JSON αντικείμενο και τα εμφωλευμένα JSON αντικείμενα έχουν τέτοιο τυπο.

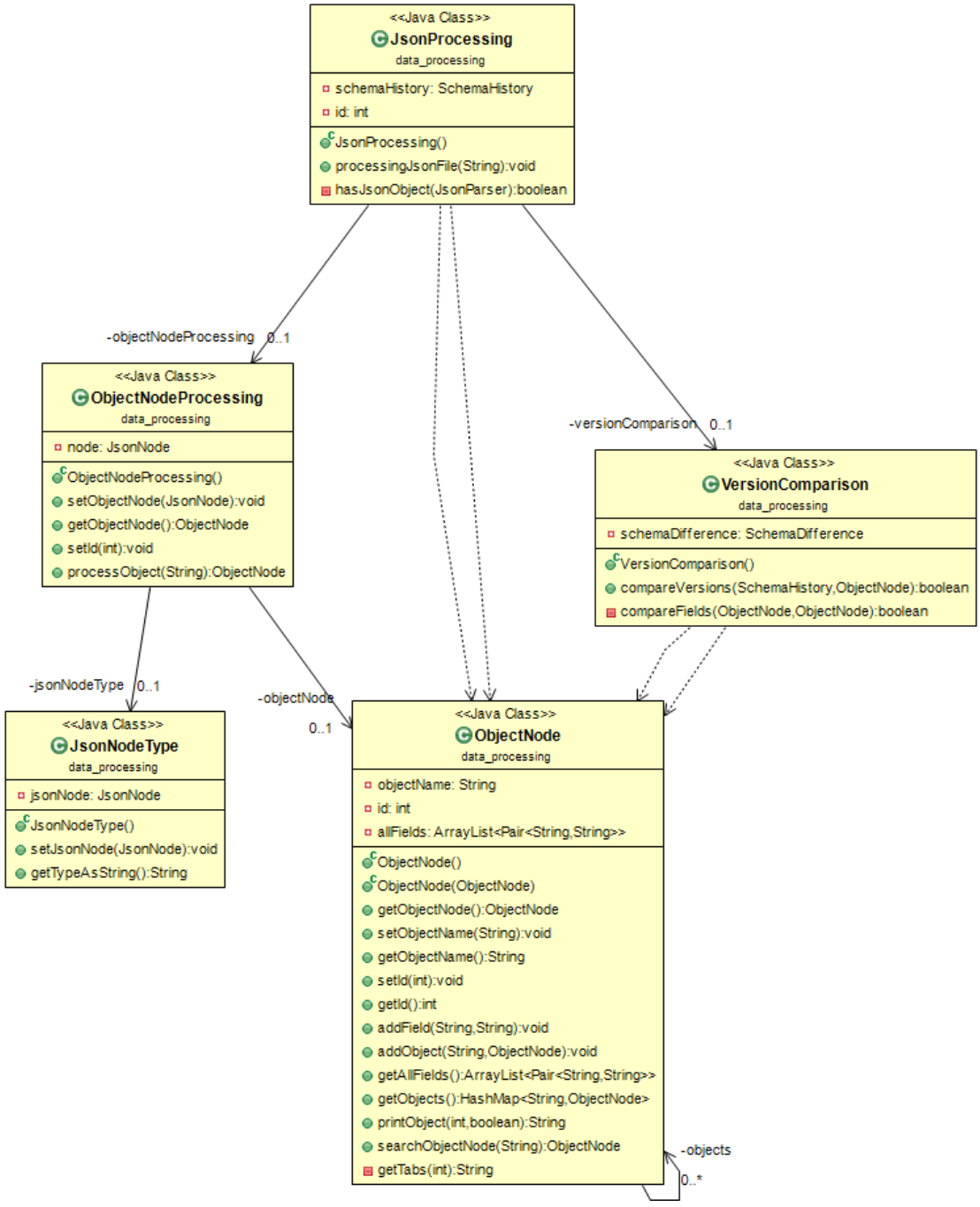
- **NullNode:** Όταν ο τύπος της τιμής του πεδίου είναι Null.

3.2 Σχεδίαση και αρχιτεκτονική λογισμικού

Όπως έχει αναφερθεί ως τώρα, ο στόχος του συγκεκριμένου λογισμικού είναι η εξαγωγή των εκδόσεων που ακολουθούν τα JSON αντικείμενα και η οπτικοποίηση αυτών των εκδόσεων. Για την υλοποίηση του λογισμικού σχεδιάστηκαν και υλοποιήθηκαν οι κατάλληλες κλάσεις οι οποίες χωρίστηκαν σε πέντε πακέτα: data_processing, gui, output, saving_projects, schema και το πακέτο με τα tests που δεν θα αναλύσουμε στο συγκεκριμένο κεφάλαιο.



Σχήμα 3.2.1. Διάγραμμα Πακέτων.

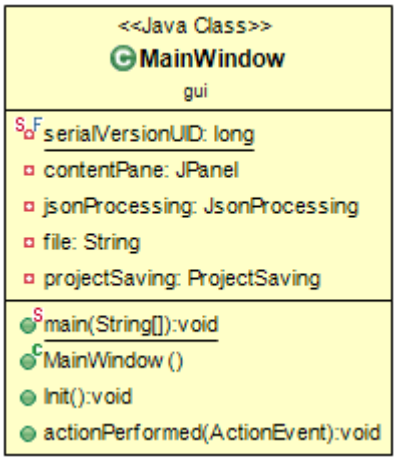


Σχήμα 3.2.1: For the data_processing package Class diagram

3.2.1 Πακέτο data_processing.

Το συγκεκριμένο πακέτο περιέχει τις κλάσεις που χρησιμοποιήθηκαν για να διαβάσει το αρχείο, να συγκρίνει τις εκδόσεις των αντικειμένων μέσα στο αρχείο JSON και να αποθηκεύσει σε κατάλληλες δομές τα δεδομένα που χρειάζεται το λογισμικό. Το πακέτο

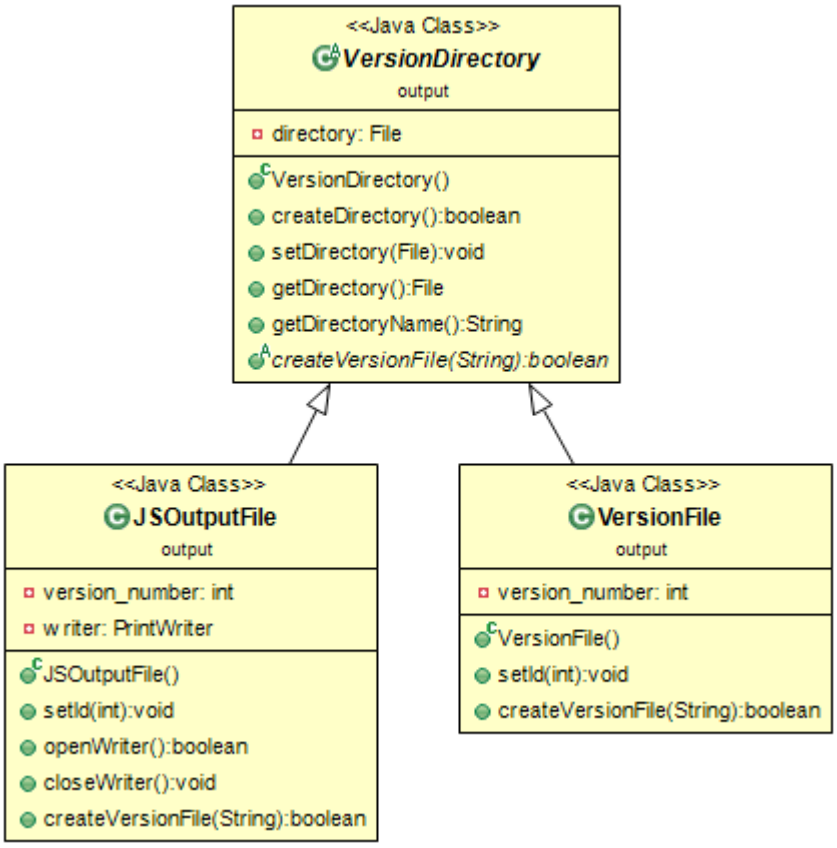
αρχικά περιέχει την κλάση JsonNodeType η οποία δέχεται σαν όρισμα την τιμή του JsonNode και επιστρέφει με την μέθοδο getTypeAsString σε String μορφή τι τύπος είναι. Το πακέτο περιέχει την κλάση ObjectNode η οποία περιέχει τα πεδία: objectName, id, allFields, objects. Το objectName είναι το key του JsonNode, το id είναι η θέση του JsonObject μέσα στο αρχείο, μόνο για το ObjectNode που έχει parent ίσο με root, όλα τα εμφωλευμένα έχουν το ίδιο id με το root που ανήκουν. Το allFields είναι μια λίστα με String, String που περιέχει όλα τα JsonNodes του JsonObject με την μορφή key : value type. Η μεταβλήτη objects είναι hashmap με το κλειδί να είναι String που περιέχει το key και η τιμή να είναι ObjectNode το οποίο περιέχει το εμφωλευμένο JsonObject. Εκτός από τις getters, setters μεθόδους, η κλάση περιέχει και μια μέθοδο printObject που επιστρέφει σε String μορφή το JsonObject αλλά πλέον αντί για key : value έχει key : value type. Το πακέτο περιέχει επίσης την κλάση ObjectNodeProcessing η οποία είναι υπεύθυνη για την επεξεργασία του JsonObject. Η κλάση αυτή περιέχει την μέθοδο processObject, η οποία παίρνει σαν String τον parent του JsonObject και διαβάζει το JsonObject, στο οποίο αν βρεθεί εμφωλευμένο JsonObject θα καλέσει αναδρομικά τον εαυτό της και θα επιστρέφει μεταβλητή της κλάσης ObjectNode που θα περιέχει μέσα όλα τα δεδομένα που θέλουμε για το JsonObject. Το πακέτο επίσης, περιέχει την κλάση VersionComparisson η οποία έχει τις μεθόδους compareVersions, που υλοποιεί την σύγκριση ανάμεσα στο JsonObject που τρέχει εκείνη την στιγμή και τα JsonObject που αποθηκεύτηκαν ως εκδόσεις σε κατάλληλές δομές που θα εξηγήσουμε παρακάτω. Στην κλάση VarsionComparison υπάρχει και η μέθοδος compareFields που συγκρίνει αν προστέθηκε ή αφαιρέθηκε κάποιο JsonNode, αλλά και αν άλλαξε ο τύπος της τιμής. Τέλος, το πακέτο περιέχει την κλάση JsonProcessing η οποία υλοποιεί το διάβασμα του αρχείο JSON και την εξαγωγή των JsonObjects από αυτό με την βοήθεια της βιβλιοθήκης Jackson που χρησιμοποιούμε. Έπειτα, συγκρίνει τις εκδόσεις και φτιάχνει το ιστορικό εκδόσεων που θα χρειαστεί για να αναπαραστήσουμε τις εκδόσεις μας στο μέλλον.



Σχήμα 3.2.2: For the gui package Class diagram

3.2.2 Πακέτο gui

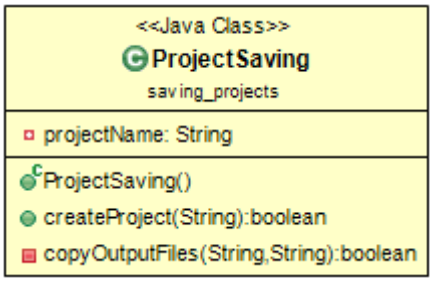
Το πακέτο gui είναι υπεύθυνο για την διεπαφή του λογισμικού με τον χρήστη και έχει την κλάση MainWindow. Δέχεται το αρχείο από τον χρήστη το επεξεργάζεται και ο χρήστης μπορεί να δει τις εκδόσεις του αρχείο, να αποθηκεύσει το project του και να ανεβάσει αν το επιθυμεί projects που ήδη υπάρχουν στο σύστημα.



Σχήμα 3.2.3: For the output package Class diagram

3.2.3 Πακέτο output

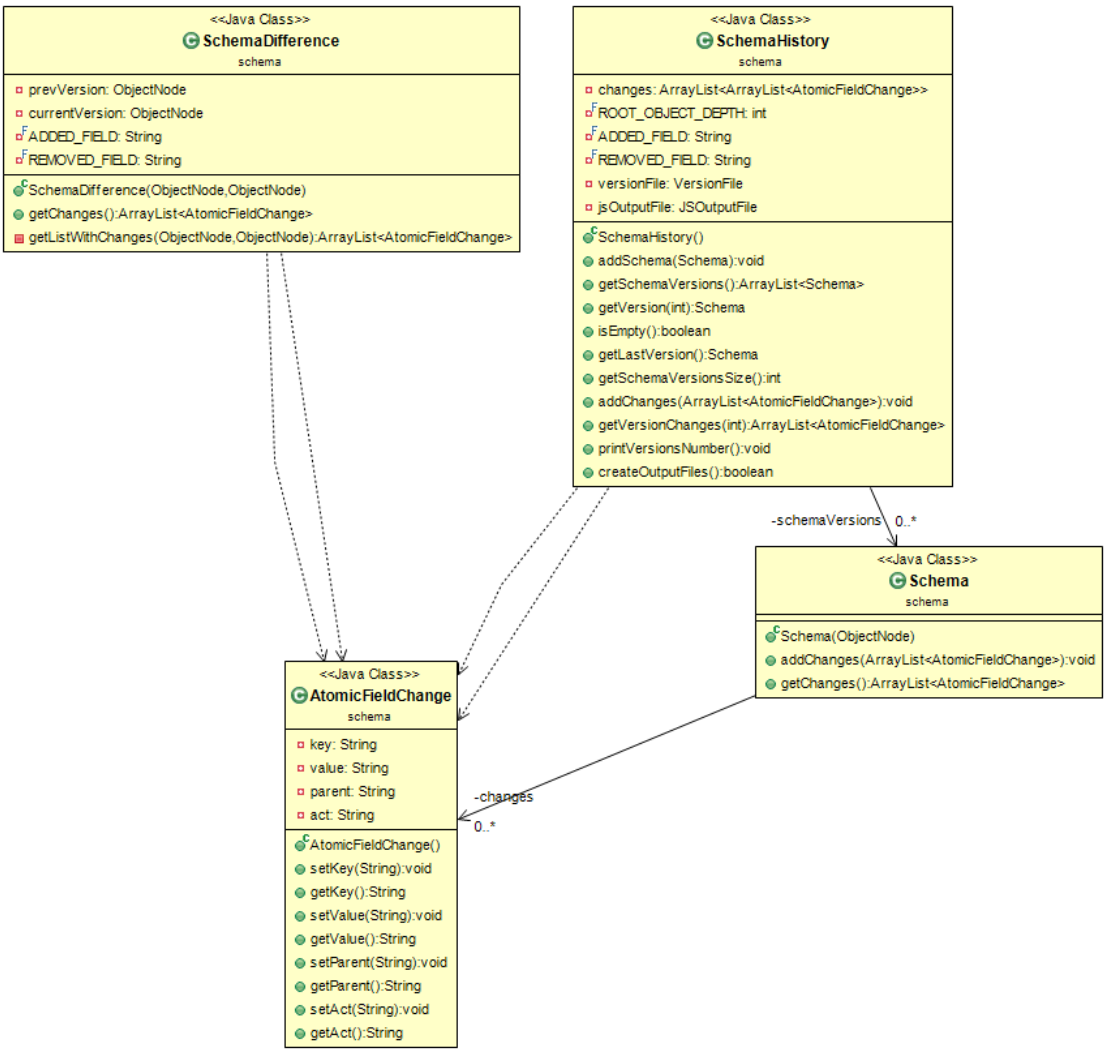
Το πακέτο output είναι υπεύθυνο για την δημιουργία του φακέλου και των αρχείων που έχουν τις εκδόσεις και περιέχει τις κλάσεις VersionDirectory, JSOutputFile και VersionFile. Η κλάση VersionDirectory δημιουργεί έναν φάκελο output ο όποιος περιέχει σε αρχεία τύπου csv τις εκδόσεις που περιέχονται στο αρχείο και στην αρχή του κάθε αρχείου τις αλλαγές που έγιναν από την τελευταία έκδοση που προστέθηκε. Την κλάση VersionFile που είναι υπεύθυνη για την δημιουργία του κάθε csv αρχείου και η ονομασία του κάθε αρχείου προκύπτει από την μέτρηση των ήδη υπάρχων εκδόσεων, δηλαδή 'version_' και ο αριθμός της έκδοσης. Η κλάση JSOutputFile υλοποιεί την δημιουργία του javascript αρχείου που έχει μέσα τις εκδόσεις όλες σε JSON μεταβλητές που υποστηρίζει η javascript και όνομα 'data_' συν ο αριθμός της έκδοσης, τέλος ένα data_counter για να ξέρουμε όταν εμφανίζουμε τις εκδόσεις πόσες εκδόσεις έχουμε.



Σχήμα 3.2.4: For the saving_projects package Class diagram

3.2.4 Πακέτο saving_projects

Το πακέτο saving_projects υλοποιεί την αποθήκευση του project και περιέχει την κλάση ProjectSaving η οποία δημιουργεί έναν φάκελο Saving Projects αν το λογισμικό τρέχει για πρώτη φορά ώστε μέσα σε αυτόν να βρίσκονται όλα τα project που θα αποθηκεύσει ο χρήστης. Στην συνέχεια, δημιουργεί έναν φάκελο με το όνομα που έδωσε ο χρήστης για το project του με τον φάκελο output μέσα σε αυτόν και τα αρχεία 'index.html' και 'view.js'. Ώστε να μην πειράζουμε τα δύο αυτά αρχεία κάθε φορά ανάλογα με το όνομα που έδωσε ο χρήστης για project και να χρησιμοποιούμε πάντα τον φάκελο Output.



Σχήμα 3.2.5: For the schema package Class diagram

3.2.5 Πακέτο schema

Το πακέτο schema είναι υπεύθυνο για τον έλεγχο και την δημιουργία των εκδόσεων που ακολουθούν τα JsonObject του αρχείου που καταχώρησε ο χρήστης. Περιέχει τις κλάσεις SchemaHistory, SchemaDifference, Schema, AtomicFieldChange. Την AtomicFieldChange η οποία περιέχει τέσσερα πεδία τα key, value, parent, act. Το πεδίο key και το πεδίο value αναφέρονται στα key : value του JsonNode αντίστοιχα, το parent στο JsonObject που ανήκει αυτό το JsonNode, αν είναι το αρχικό JsonObject θα έχει τιμή root αλλιώς το όνομα του JsonObject που ανήκει. Το πεδίο act παίρνει τιμές '+' ή '-' για το αν προστέθηκε ή αφαιρέθηκε το συγκεκριμένο JsonNode. Την κλάση Schema οποία κληρονομεί από την ObjectNode όλα τα δεδομένα και προσθέτει και τις αλλαγές στα JsonNodes. Την κλάση SchemaDifference η οποία είναι υπεύθυνη για την εύρεση των αλλαγών που έγιναν στις

εκδόσεις. Τέλος, την κλάση SchemaHistory η οποία περιέχει κατάλληλα πεδία που χρησιμεύουν στην αποθήκευση της έκδοσης στο λογισμικό.

3.2.6 Αρχείο index.html

Το αρχείο index είναι υπεύθυνο για την οπτικοποίηση των εκδόσεων που βρέθηκαν. Χρησιμοποιεί την βιβλιοθήκη της Google, την Google Charts που περιέχει διάφορους τρόπους για οπτικοποίηση. Στο συγκεκριμένο λογισμικό χρησιμοποιήσαμε το OrgChart επειδή φαίνεται να είναι το πιο κατάλληλο ώστε να μπορεί ο χρήστης να διακρίνει τις διαφορές ανάμεσα στις εκδόσεις.

3.2.7 Αρχείο view.js

Το αρχείο view υλοποιεί τον πίνακα για το OrgChart (https://developers.google.com/chart/interactive/docs/gallery/orgchart), διαβάζει το αρχείο versions.js που έβγαλε το λογισμικό χρησιμοποιώντας τις τεχνικές που εξηγήσαμε παραπάνω και επεξεργάζεται τα δεδομένα. Χρησιμοποιεί την συνάρτηση setDataTable έτσι ώστε να δημιουργήσει τον πίνακα για το OrgChart και αναδρομικά καλεί τον εαυτό της αν πρόκειται για εμφωλευμένα JSON αντικείμενα.

3.3 Σχεδίαση και αποτελέσματα ελέγχου του λογισμικού

Για τον έλεγχο του εργαλείου χρησιμοποιήθηκαν Junit Test Cases και συγκεκριμένα η πέμπτη έκδοση της Junit βιβλιοθήκης, στα οποία ελέγχουμε τις βασικές κλάσεις και τις μεθόδους τους. Τα tests βρίσκονται μέσα στον κώδικα στο πακέτο “tests” και τα JSON αρχεία που ελέγχονται βρίσκονται στον φάκελο “tests”. Έχουμε τα παρακάτω tests:

- **AtomicFieldChangeTest:** Ελέγχει την κλάση AtomicFieldChange για το αν ενημερώνεται σωστά για το αν προστέθηκε ή αφαιρέθηκε ένα πεδίο.
- **JsonNodeTypeTest:** Ελέγχει την κλάση JsonNodeType και συγκεκριμένα την μεθοδό της, getTypeAsString(), για το αν επιστρέφει σωστά τον τύπο της τιμής του πεδίου.
- **ObjectNodeProcessingTest:** Ελέγχει την κλάση ObjectNodeProcessing και την μέθοδο searchObject της κλάσης ObjectNode. Συγκεκριμένα, ελέγχει αν η

processObject μέθοδος της ObjectNodeProcessing επιστρέφει σωστά το JSON αντικειμένων που πήρε σαν είσοδο και στην μέθοδο searchObject ελέγχει αν το JSON αντικείμενο που ψάχνουμε υπάρχει ή όχι μέσα στην δομή δεδομένων που έχουμε.

- **ObjectNodeTest:** Ελέγχει την κλάση ObjectNode αν αποθηκεύει σωστά τα JSON αντικειμένα σε κατάλληλες δομές που έχουμε ορίσει μέσα στην κλάση.
- **SchemaDifferenceTest:** Ελέγχει την κλάση SchemaDifference για το αν βρίσκει σωστά τις αλλαγές που έγιναν μεταξύ των JSON αντικειμένων, ώστε να αποθηκευτούν σωστά οι διαφορετικές εκδόσεις.
- **VersionComparisonChangeNestedObjectTest:** Ελέγχει την κλάση VersionComparison για το αν μπορεί να ξεχωρίσει τις αλλαγές που γίνονται σε ένα εμφωλευμένο JSON αντικείμενο. Παίρνει σαν είσοδο το αρχείο version_comparison_nested_object.json το οποίο περιέχει δύο JSON αντικείμενα και το πεδίο “address” είναι εμφωλευμένο JSON αντικείμενο. Στο πρώτο αντικείμενο έχει το πεδίο του number με τύπο “IntNode” και στο δεύτερο αντικείμενο με τύπο “StringNode”, οπότε ελέγχουμε αν η κλάση συμπεριφέρεται σωστά σε αυτό το σενάριο.
- **VersionComparisonChangesTest:** Ελέγχει την κλάση VersionComparison, σε αυτή την περίπτωση για το αν μπορεί να ξεχωρίσει τις αλλαγές στο αρχικό JSON object. Παίρνει ως είσοδο το αρχείο version_comparison_different_object.json και την τιμή του πεδίου kids την αλλάζουμε από “DoubleNode” σε “IntNode”.
- **VersionComparisonNotChangesTest:** Ελέγχει την κλάση VersionComparison με την διαφορά ότι σε αυτήν την περίπτωση δεν άλλαξαν καθόλου τα JSON αντικείμενα. Παίρνει σαν είσοδο το αρχείο version_comparison_same_object.json το οποίο περιέχει δύο JSON αντικείμενα, τα οποία διατηρούν ίδιο το schema τους χωρίς να αλλάξουν τιμές.

3.4 Λεπτομέρειες εγκατάστασης και υλοποίησης

Στην παρούσα ενότητα περιγράφονται οι πλατφόρμες που χρησιμοποιήθηκαν για την ανάπτυξη του λογισμικού και οι λεπτομέρειες υλοποίησης.

3.4.1 Πλατφόρμες και προγραμματιστικά εργαλεία

Το συγκεκριμένο λογισμικό αναπτύχθηκε στην γλώσσα προγραμματισμού Java (https://www.java.com/en/) η οποία είναι μία αντικειμενοστρεφής γλώσσα προγραμματισμού υψηλού επιπέδου με απλή σύνταξη που περιέχει πάρα πολλές βιβλιοθήκες που μπορούν να βοηθήσουν στην ανάπτυξη λογισμικών. Η Java λόγω της αντικειμενοστρεφής φύσης της μπορεί να προσφέρει αντικείμενα που δημιουργούνται πολλές φορές μέσα στο λογισμικό και επικοινωνούν με τα υπόλοιπα αντικείμενα. Χρησιμοποιείται πάρα πολύ όταν θέλουμε να φτιάξουμε εφαρμογές για κινητά που χρησιμοποιούν Android, χρηματοοικονομικά και εμπορικά λογισμικά καθώς και προγράμματα που χειρίζονται αρκετά μεγάλα δεδομένα. Τέλος, προσφέρει ασφάλεια, παράλληλο προγραμματισμό και έχει πολύ μεγάλο και αξιόλογο community λόγω της απήχησης που έχει. Για την οπτικοποίηση στο κομμάτι του λογισμικού που ο χρήστης μπορεί να εισάγει αρχείο, να αποθηκεύσει το project, να δει τα αποτελέσματα ή να ανεβάσει project που υπάρχει ήδη χρησιμοποιήθηκε η Swing η οποία είναι ελαφριά και εύκολη προς την χρήση. Στο κομμάτι της οπτικοποίησης που ο χρήστης μπορεί να δει τα αποτελέσματα δηλαδή τις εκδόσεις που έβγαλε το λογισμικό χρησιμοποιήθηκε η γλώσσα σήμανσης HTML (https://html.com/) η οποία είναι αρκετά φιλική και υποστηρίζεται σχεδόν από όλους τους browsers. Επίσης, χρησιμοποιήσαμε την γλώσσα προγραμματισμού JavaScript (https://www.javascript.com/) κατάλληλη για Web Development, προσφέρει ταχύτητα, απλότητα και είναι αρκετά δημοφιλής πράγμα που σημαίνει ότι και αυτή έχει αρκετά μεγάλο community. Τέλος, για να οπτικοποιήσουμε τα αποτελέσματα χρησιμοποιήσαμε το εργαλείο Google Charts (https://developers.google.com/chart) της Google (https://www.google.com/) και συγκεκριμένα το Organization Chart. Το εργαλείο προσφέρει αρκετά charts τα οποία μπορούν να βοηθήσουν στην οπτικοποίηση δεδομένων, αλλά ήμασταν ανάμεσα στο Treemaps ή το Organization chart. Ενώ το Treemaps θα ήταν εφικτό να χρησιμοποιηθεί για την οπτικοποίηση των εκδόσεων, δεδομένου ότι οι εμφωλευμένες περιοχες θα μπορούσαν να βοηθήσουν στην οπτικοποίηση των πεδίων ή εμφωλευμένων JSON αντικειμένων, δεν θα μπορούσε ο χρήστης να διακρίνει εύκολα τις αλλαγές που έγιναν στις εκδόσεις, οπότε καταλήξαμε στο Organization Chart.

Το περιβάλλον ανάπτυξης που χρησιμοποιήθηκε για την συγγραφή του κώδικα είναι το eclipse (https://www.eclipse.org/) το οποίο είναι ένα πάρα πολυ διαδεδομένο δωρεάν εργαλείο. Το eclipse προσφέρει μια πληθώρα από plug-ins που βοηθάνε στην ανάπτυξη λογισμικού και έρχεται με χαρακτηριστικά όπως git version control, eclipse marketplace,

debug, xml editing, grandle support κ.α. Τέλος, στο eclipse υπάρχει και το ObjectAid (https://www.objectaid.com/) που μας βοήθησε να απεικονίσουμε τα UML διαγράμματα λαμβάνοντας τις απαραίτητες λεπτομέρειες των κλάσεων που έχουμε δημιουργήσει.

Το πρόγραμμα που χρησιμοποιήθηκε για την συγγραφή του κώδικα σε HTML και JavaScript είναι το Atom (https://atom.io/). Το Atom εκτός από αρκετά όμορφο UI έχει επίσης πάρα πολλά και χρήσιμα plug-ins, ειδικά για web development, TypeScript editor, σύνδεση με github, cross platform, μεγάλο community και είναι δωρεάν.

3.4.2 Λεπτομέρειες υλοποίησης

3.4.2.1 Υλοποίηση κώδικα

Αρχικά, για την εκτέλεση του λογισμικού ο χρήστης πρέπει να επιλέξει αρχείο εισόδου από το γραφικό περιβάλλον που εμφανίζεται, το οποίο πρέπει να είναι JSON αρχείο. Για την επεξεργασία του αρχείου που εισήγαγε ο χρήστης, και για να μπορέσουμε να οπτικοποιήσουμε τις εκδόσεις των JSON αντικειμένων που βρίσκονται μέσα σε αυτό, είναι υπεύθυνες οι κλάσεις:

- **JsonNodeType:** Για την επιστροφή του τύπου της τιμής του πεδίου.
- **JsonProcessing:** Η μέθοδος processingJsonFile θα πάρει το αρχείο εισόδου σαν παράμετρο και θα το επεξεργαστεί με την βοήθεια της process που θα πάρει σαν παράμετρο έναν JsonParser για να παίρνουμε ένα συγκεκριμένο JSON αντικείμενο. Στα σχήματα 3.4.2.1 και 3.4.2.2 φαίνονται οι μέθοδοι.

```
public void processingJsonFile(String file) {
    JsonFactory factory = new JsonFactory();
    JsonParser parser = null;
    boolean json_array = true;
    try {
        parser = factory.createParser(new File(file));
    } catch (IOException e) {
        System.out.println("IO Exception in JsonParser...");
        e.printStackTrace();
    } catch (NullPointerException e) {
        System.out.println("Can't open this file...");
        e.printStackTrace();
    }
    if (parser != null) {
        while(hasJsonObject(parser)) {
            json_array = false;
            process(parser);
        }
        if (json_array == true) {
            JsonToken token = null;
            while(true) {
                try {
                    token = parser.nextToken();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
                if (!JsonToken.START_OBJECT.equals(token)) break;
                if (token == null) break;
                process(parser);
            }
        }
    }
    System.out.println("Total json objects : " + id);
    schemaHistory.printVersionsNumber();
    if (schemaHistory.createOutputFiles()) {
        System.out.println("All Version Files Created Successfully...");
    } else {
        System.out.println("An error occurred while "
                + "creating version files...");
    }
}
```

Σχήμα 3.4.2.1. Επεξεργασία του JSON αρχείου

```
private void process(JsonParser parser) {
    parser.setCodec(new ObjectMapper());
    JsonNode jsonNode = null;
    try {
        jsonNode = parser.readValueAsTree();
    } catch (IOException e) {
        System.out.println("IO Exception in JsonNode...");
        e.printStackTrace();
    }
    objectNodeProcessing = new ObjectNodeProcessing();
    objectNodeProcessing.setObjectNode(jsonNode);
    objectNodeProcessing.setId(id);
    ObjectNode currentObject = objectNodeProcessing
            .processObject("root");
    versionComparison.compareVersions(schemaHistory, currentObject);
    id++;
}
```

Σχήμα 3.4.2.2. Σύγκριση JSON αντικειμένων

- **ObjectNode:** Υπεύθυνη για την κατάλληλη αποθήκευση σε δομές των JSON αντικειμένων.
- **ObjectNodeProcessing:** Βοηθάει στην επεξεργασία των JSON αντικειμένων και την αποθήκευση τους. Στα σχήματα 3.4.2.3 φαίνεται η μέθοδος processObject που υλοποιεί την επεξεργασία και αποθήκευση, η οποία παίρνει σαν όρισμα τον πατέρα που βρίσκεται το JSON αντικείμενο.

```
public ObjectNode processObject(String parent) {
    Iterator<Map.Entry<String, JsonNode>> objectIterator = node.fields();
    Map.Entry<String, JsonNode> nextField;
    objectNode.setObjectName(parent);
    while (objectIterator.hasNext()) {
        nextField = objectIterator.next();
        jsonNodeType.setJsonNode(nextField.getValue());
        if (jsonNodeType.getTypeAsString().equals("ObjectNode")) {
            objectNode.addField(nextField.getKey(),
                    jsonNodeType.getTypeAsString());
            ObjectNodeProcessing object = new ObjectNodeProcessing();
            object.setObjectNode(nextField.getValue());
            object.processObject(parent + "/" + nextField.getKey());
            objectNode.addObject(nextField.getKey(),
                    object.getObjectNode());
        } else {
            objectNode.addField(nextField.getKey(),
                    jsonNodeType.getTypeAsString());
        }
    }
    return objectNode;
}
```

Σχήμα 3.4.2.3. Επεξεργασία και αποθήκευση των JSON αντικειμένων

- **VersionComparison:** Περιέχει τις μεθόδους compareVersion και compareFields οι οποίες υλοποιούν την σύγκριση των JSON αντικειμένων. Συγκεκριμένα, η compareVersions παίρνει σαν παραμέτρους την λίστα με τις εκδόσεις που έχουν ήδη αποθηκευτεί και το JSON αντικείμενο που διαβάζει το λογισμικό εκείνη την στιγμή και τα συγκρίνει με την βοήθεια της compareFields, με σκοπό να αποφασίσει αν η έκδοση που επεξεργαζόμαστε εκείνη την στιγμή έχει εμφανιστεί ξανά ή όχι. Η compareFields παίρνει σαν παραμέτρους μία από τις αποθηκευμένες εκδόσεις και την τρέχουσα έκδοση, ώστε να ελέγξει αν υπάρχουν διαφορές μεταξύ τους. Στα σχήματα 3.4.2.4 και 3.4.2.5 φαίνονται οι μέθοδοι.

```
public boolean compareVersions(SchemaHistory schemaHistory,
                ObjectNode currentVersion) {
    int count_dif_versions = 0;
    if (schemaHistory.isEmpty()) {
        Schema schema = new Schema(currentVersion);
        schemaHistory.addSchema(schema);
        return true;
    }
    for (Schema version : schemaHistory.getSchemaVersions()) {
        if (compareFields(version.getObjectNode(), currentVersion) == false) {
            count_dif_versions++;
        }
    }
    if (count_dif_versions == schemaHistory.getSchemaVersions().size()) {
        if (schemaHistory.getSchemaVersionsSize() >= 1) {
            schemaDifference =
                    new SchemaDifference(schemaHistory.getLastVersion()
                            .getObjectNode(),currentVersion);
            Schema schema = new Schema(currentVersion);
            schema.addChanges(schemaDifference.getChanges());
            schemaHistory.addSchema(schema);
        }
        return true;
    }
    return false;
}
```

Σχήμα 3.4.2.4. Σύγκριση εκδόσεων

```
private boolean compareFields(ObjectNode version,
        ObjectNode currentVersion) {
    int count_same_fields = 0;
    for (Pair<String, String> currentVersionField :
            currentVersion.getAllFields()) {
        if (version.getAllFields().contains(currentVersionField)) {
            count_same_fields++;
            if (currentVersionField.getValue().equals("ObjectNode")) {
                try {
                    if (compareFields(version.searchObjectNode
                            (currentVersionField.getKey()),
                            currentVersion.searchObjectNode
                            (currentVersionField.getKey())) == false ) {
                        return false;
                    }
                } catch(NullPointerException e) {
                    System.out.println("Class: VersionComparison, " +
                                "can't find object " +
                                "with this name...");
                }
            }
        } else {
            return false;
        }
    }
    if (count_same_fields == version.getAllFields().size()) return true;
    return false;
}
```

Σχήμα 3.4.2.5. Σύγκριση πεδίων.

Για την οπτικοποίηση των αποτελεσμάτων έχουμε το αρχείο ”index.html” το οποίο εισάγει το εργαλείο της Google, το αρχείο με τις εκδόσεις “versions.js” και το αρχείο της JavaScript “view.js”. Στο αρχείο view.js έχουμε την συνάρτηση setDataTable η οποία παίρνει σαν παραμέτρους την έκδοση, τον πίνακα που θα φτιάξουμε με αυτά και τον πατέρα των JSON αντικειμένων κάθε φορά. Την συνάρτηση drawChart η οποία διαβάζει και επεξεργάζεται τα δεδομένα από το αρχείο “versions.js”, φτιάχνει στοιχεία “div” για την HTML και εμφανίζει σε καθένα από αυτά και μια έκδοση με την βοήθεια του Organization Chart. Βλέπουμε και τις δύο συναρτήσεις στα σχήματα 3.4.2.6 και 3.4.2.7.

```
function setDataTable(version, data, father) {
  var name = null;
  for (var key in version) {
    if (version.hasOwnProperty(key)) {
      name = key + " : " + version[key];
      data.addRows([[name, father]]);
      if (version[key].constructor == objectConstructor) {
        var nested = version[key];
        setDataTable(nested, data, name);
      }
    }
  }
```

Σχήμα 3.4.2.6. Αποθήκευση δομής οπτικοποίησης

```
function drawChart() {
  var versions = new Array(data_counter);
  for (var i = 0; i <= data_counter; i++) {
    var data = new google.visualization.DataTable();
    data.addColumn('string', 'Field');
    data.addColumn('string', 'Father');
    data.addRows([['root', '']]);
    setDataTable(window["data_" + i], data, "root");
    versions[i] = data;
    var newDiv = document.createElement("div");
    newDiv.style.height = "100px";
    newDiv.style.textAlign = "center";
    newDiv.innerHTML = "Version " + (i + 1);
    newDiv.style.fontSize = "50px";
    document.body.appendChild(newDiv);
    var newDiv = document.createElement("div");
    newDiv.setAttribute("id", "version_" + i);
    document.body.appendChild(newDiv);
  }
  for (var i = 0; i <= data_counter; i++) {
    var chart = new google.visualization.OrgChart(document.getElementById("version_" + i));
    chart.draw(versions[i], {'allowHtml' : true})
  }
}
```

Σχήμα 3.4.2.7. Οπτικοποίηση εκδόσεων

3.4.2.2 Αρχεία JSON

Το σύστημα μπορεί να δεχθεί αρχεία που είναι υποχρεωτικά τύπου JSON, τα οποία περιέχουν ένα ή περισσότερα JSON αντικείμενα. Αυτό σημαίνει ότι η δομή του αρχείου θα ξεκινάει με JSON αντικείμενο δηλάδη με "{" τα πεδία του αντικειμένου και κλείνει με "}", και στην συνέχεια μπορεί να περιέχει και αλλά τέτοια αντικείμενα χωρίς κόμμα μεταξύ τους. Καθώς ψάχναμε δεδομένα που θα μπορούσαν να χρησιμοποιηθούν ως αρχεία εισόδου, είδαμε ότι πολλά αρχεία αντί να έχουν την δομή όπως εξηγήσαμε παραπάνω, πολλά JSON αντικείμενα είχαν εν συνεχεία JSON αντικείμενα μέσα σε ArrayNode (JSON πίνακα). Έτσι, το σύστημα μπορεί να δεχθεί και αρχεία που τα JSON αντικείμενα βρίσκονται σε ArrayNode, δηλαδή να ξεκινάει με "[" να συνεχίζει με JSON αντικείμενα χωρισμένα μεταξύ τους με κόμμα και στο τέλος να κλείνει με "]".

3.4.2.3 Αρχεία Εξόδου

Το σύστημα μόλις τελειώσει με την επεξεργασία του αρχείου εισόδου, δημιουργεί έναν φάκελο "output" ο οποίος περιέχει μέσα τις εκδόσεις μεμονομένες σε αρχεία τύπου "csv" και αρχείο με τις εκδόσεις όλες μαζί για να χρησιμοποιηθούν στο αρχείο JavaScript για την οπτικοποίηση των αποτελεσμάτων.

3.4.2.3.1 Αρχεία CSV

Τα αρχεία csv δημιουργούνται από το σύστημα μέσα στον φάκελο "output", με ονομασία "version_" συν τον αύξων αριθμό κάθε έκδοσης. Στην αρχή του αρχείου υπάρχει η πληροφορία για το πότε προστέθηκε η έκδοση στο ιστορικό εκδόσεων του συστήματος, παράδειγμα αν η έκδοση προστέθηκε στο δεύτερο JSON αντικείμενο τότε στην αρχή γράφει "Version added at: 2". Στην συνέχεια, το αρχείο έχει τις αλλαγές που έγιναν σε σχέση με την προηγούμενη έκδοση, το μονοπάτι του πεδίου που άλλαξε και τι αλλαγή έγινε. Στο τέλος του αρχείου υπάρχει η έκδοση, η οποία είναι γραμμένη με την μορφή ενός JSON αντικειμένου με την μόνη διαφορά ότι αντί για την τιμή του πεδίου υπάρχει ο τύπος της τιμής του πεδίου.

3.4.2.3.2 Αρχείο Εκδόσεων

Το συγκεκριμένο αρχείο είναι τύπου JavaScript και δημιουργήθηκε από το σύστημα για να παρέχει στο αρχείο JavaScript "view.js" όλες τις πληροφορίες για τις εκδόσεις που βρέθηκαν στο αρχείο εισόδου. Οι εκδόσεις γράφονται ως JSON αντικείμενα όπως

ακριβώς και στα “csv” αρχεία και καταχωρούνται σε μεταβλητές τις JavaScript οι οποίες έχουν όνομα “data_” συν τον αύξων αριθμό της έκδοσης, στο τέλος του αντικειμένου μέτα το “}” χρειάζεται το σύμβολο “;”. Στο τέλος του αρχείου υπάρχει μια μεταβλητή “data_counter” η οποία μας βοηθάει στην οπτικοποίηση να γνωρίζουμε πόσες εκδόσεις βρήκε το σύστημα. Το αρχείο δημιουργήθηκε με αυτή την δομή λόγω του ότι η JavaScript δεν έχει πρόσβαση “εύκολα” σε τοπικά αρχεία, έτσι εμείς κάναμε το αρχείου τύπου JavaScript για να μπορεί να “βλέπει” και να έχει πρόσβαση στις μεταβλητές που περιέχουν τις εκδόσεις.

3.4.2.4 Αποθήκευση Project

Το σύστημα επιτρέπει στον χρήστη να αποθηκεύσει στο σύστημα το project που δημιούγησε. Αυτό γίνεται όταν ο χρήστης πατήσει το κουμπί “Save Project”, το σύστημα ζητάει από τον χρήστη ένα όνομα που επιθυμεί να αποθηκευτεί το project του. Τα σύστημα αν είναι η πρώτη φορά που ο χρήστης αποθηκεύει ένα από τα projects του, δημιουργεί τον φάκελο “Saving Projects”. Μέσα στον φάκελο “Saving Projects” δημιουργεί έναν φάκελο με το όνομα που επέλεξε ο χρήστης και αντιγράφει τον φάκελο “output” που δημιούργησε το σύστημα και τα αρχεία “index.html” και “view.js”, έτσι ώστε όταν ο χρήστης θελήσει να ξαναδεί το project να μπορεί να το κάνει. Η κλάση που είναι υπεύθυνη για την αποθήκευσή του project είναι η “ProjectSaving” με τις μεθόδους που φαίνονται στα σχήματα 3.4.2.4 και 3.4.2.5.

```
public boolean createProject(String name) {
    projectName = name;
    File directory = new File("Saving Projects/" + projectName);
    System.out.println("Creating directory: Saving Projects/"
                       + directory.getName());
    boolean result = false;
    try {
        if (!directory.exists()) {
            result = directory.mkdir();
            System.out.println(directory.getName() +
                    " directory created" + " successfully...");
        }
    } catch (SecurityException e) {
        System.out.println("Directory: " + directory.getName()
                    + ", didn't create...");
    }
    directory = new File("Saving Projects/" + projectName + "/output");
    try {
        if (!directory.exists()) {
            result = directory.mkdir();
            System.out.println(directory.getName() + " directory created"
                    + " successfully...");
        }
    } catch (SecurityException e) {
        System.out.println("Directory: " + directory.getName()
                    + ", didn't create...");
    }
    result = copyOutputFiles("output", "Saving Projects/" + projectName +
             "/output/");
    result = copyOutputFiles("index.html", "Saving Projects/" + projectName +
             "/index.html");
    result = copyOutputFiles("view.js", "Saving Projects/" + projectName +
             "/view.js");
    return result;
}
```

Σχήμα 3.4.2.4. Αποθήκευση Project

```
private boolean copyOutputFiles(String source, String destination) {
    File srcFile = new File(source);
    File destFile = new File(destination);
    try {
        if (srcFile.isDirectory()) {
            FileUtils.copyDirectory(srcFile, destFile);
        } else {
            FileUtils.copyFile(srcFile, destFile);
        }
        return true;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return false;
}
```

Σχήμα 3.4.2.5. Αντιγραφή αρχείων και φακέλων εξόδου

3.4.2.5 Άνοιγμα Αποθηκευμένου Project

Το σύστημα επιτρέπει στον χρήστη να ανοίξει ένα project το οποίο έχει αποθηκεύσει το παρελθόν. Όταν ο χρήστης πατήσει το κουμπί “Load Project” το σύστημα του εμφανίζει μήνυμα ότι το αρχείο που πρέπει να διαλέξει είναι το “index.html” το οποίο βρίσκεται στον φάκελο του project που αποθήκευσε. Στην συνέχεια το σύστημα του εμφανίζει έναν File Explorer για να διαλέξει το αρχείο που επιθυμεί.

3.5 Επεκτασιμότητα του λογισμικού

Σε περίπτωση που κάποιος επιθυμεί να χρησιμοποιήσει το εργαλείο και θα ήθελε να πραγματοποιήσει κάποια αλλαγή ή επέκταση, στην υποενότητα αυτή αναφέρονται πληροφορίες που θα τον βοηθήσουν.

Σε περίπτωση που θελήσουμε να επεκτείνουμε το εργαλείο ώστε να χρησιμοποιήσουμε ως αρχεία εισόδου αρχεία που έχουν άλλο τύπο εκτός από JSON θα πρέπει να κάνει αλλαγή στην κλάση “MainWindow” που βρίσκεται στο πακέτο “gui” και συγκεκριμένα στο σημείο μέσα στην μέθοδο “actionPerformed” που αναφέρεται στην επιλογή “Upload File”.

Σε περίπτωση που θελήσουμε να επεκτείνουμε το εργαλείο ώστε να αλλάξουμε τον τρόπο με τον οποίο αποθηκεύονται οι εκδόσεις ή οι αλλάγες θα πρέπει να πάει στην κλάση “SchemaHistory”, στην μέθοδο “createOutputFiles” και να αλλάξει τον τρόπο με τον οποίο αποθηκεύονται να δεδομένα μέσα στο κάθε αρχείο.

Κεφάλαιο 4. Πειραματική Αξιολόγηση

4.1 Μεθοδολογία πειραματισμού

Στην ενότητα αυτή θα περιγράψουμε πειράματα αξιολόγησης, στα οποία χρησιμοποιήθηκαν δεδομένα με διάφορα μεγέθη με στόχο να ελέγξουμε πόση ώρα θα χρειαστεί το σύστημα να επεξεργαστεί και να βρει τις εκδόσεις σε κάθε αρχείο. Συγκεκριμένα, επεξεργαστήκαμε τα αρχεία για να εντοπίσουμε εκδόσεις, τα οποία μπορούν να βρεθούν στον φάκελο “data”, τα οποία έχουν δεδομένα που είναι μέσα σε ArrayNode ή εκτός όπως εξηγήσαμε παραπάνω. Τα αρχεία συγκεντρώθηκαν είτε από διάφορα datasets στο Github (https://github.com/), είτε σε διάφορες σελίδες οι οποίες συλλέγουν δεδομένα σε αρχεία τύπου JSON όπως για παράδειγμα το yelp (https://www.yelp.ie/dublin). Τα αρχεία που χρησιμοποιήσαμε στα πειράματα φαίνονται στον πίνακα 4.2.1.

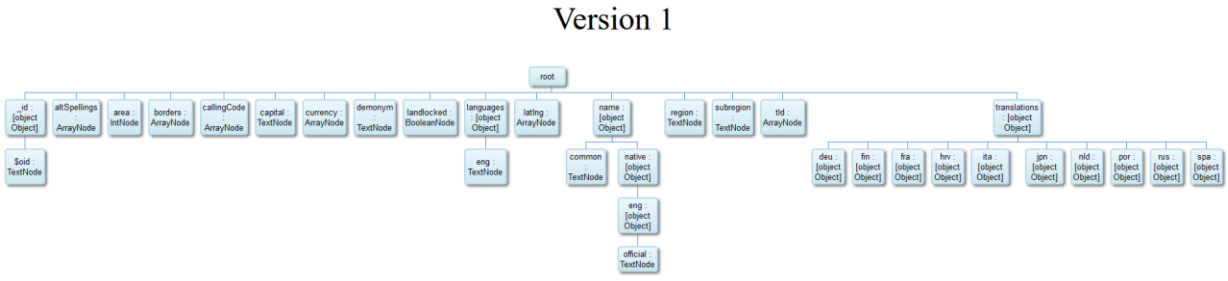
4.2 Αναλυτική παρουσίαση αποτελεσμάτων

‘Υστερα από την εκτέλεση του συστήματος με είσοδο τα αρχεία που διαλέξαμε για τα πειράματα, βγάλαμε τα αποτελέσματα που μπορείτε να δείτε στον πίνακα 4.2.1.

File(.json)	File Size(KB)	Execution Time	Total JSON Objects	Total Versions
test_countries_2_same_entries	3	0.001	2	1
test_countries_2_entries	3	0.001	2	2
countries_small	422	0.051	248	159
profiles	456	0.049	1515	11
countries_big	2.313	0.111	21.640	3
cards	2.690	0.640	9.059	464
city_inspections	23.842	0.587	81.047	6
photo	25.060	0.509	200.000	1
tip	238.805	3.571	1.223.094	1
review	5.222.145	35.914	6.685.900	1

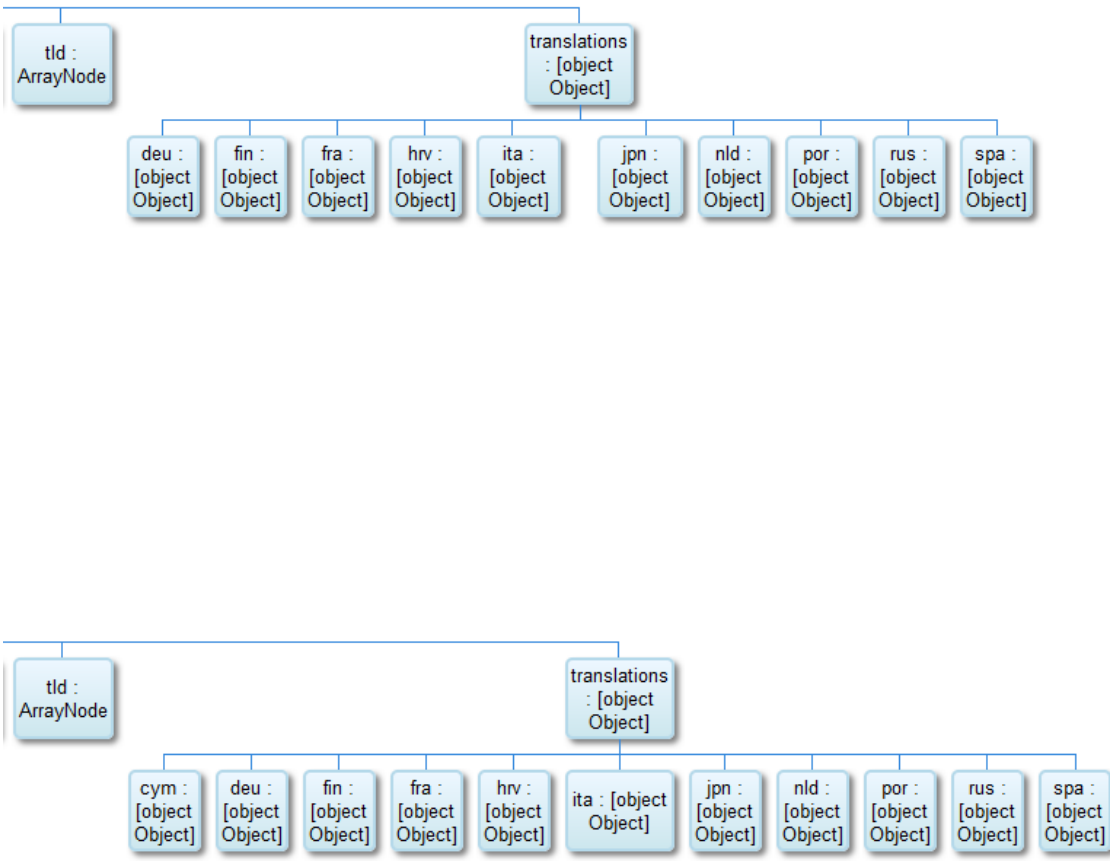
Πίνακας 4.2.1. Αποτελέσματα πειραμάτων

Αναλύοντας τον πίνακα 4.2.1 μπορεί κάποιος να προσέξει ότι όσο αυξανόταν το μέγεθος των αρχείων τόσο αυξανόταν και ο χρόνος που χρειαζόταν το σύστημα να επεξεργαστεί τα αρχεία και να εξάγει τις εκδόσεις. Τα δύο πρώτα αρχεία, "test_countries_2_same_entries" και "test_countries_2_entries" είναι σχεδόν ίδια, με την μόνη διαφορά να βρίσκεται στο πεδίο "cym" που είναι τύπου ObjectNode και βρίσκεται εμφωλευμένο μέσα στο JSON αντικείμενο "translations". Αν παρατηρούμε τις εικόνες 4.2.2 και 4.2.3, στην εικόνα 4.2.2 βλέπουμε ότι το πρώτο αρχείο περιέχει δύο JSON αντικείμενα αλλά έχει μόνο μία έκδοση.



Εικόνα 4.2.2. Οπτικοποίηση εκδόσεων για το αρχείο "test_countries_2_same_entries"

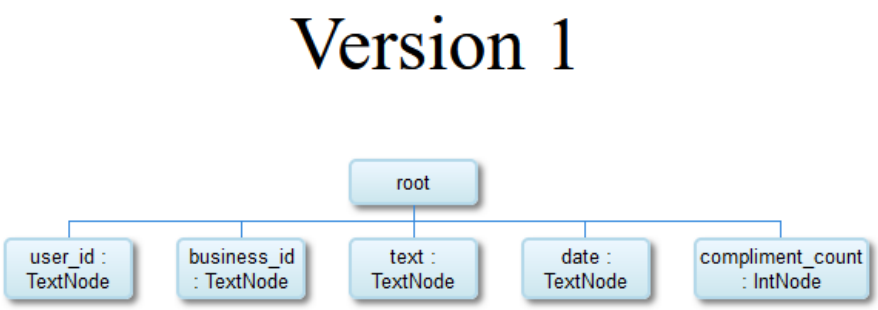
Στην εικόνα 4.2.3 βλέπουμε ότι το δεύτερο αρχείο το οποίο περιέχει και αυτό δύο JSON αντικείμενα για τον λόγο που εξηγήσαμε παραπάνω έβγαλε σωστά δύο εκδόσεις. Στην εικόνα φαίνεται μόνο το ObjectNode “translations” για να φαίνεται η διαφορά.



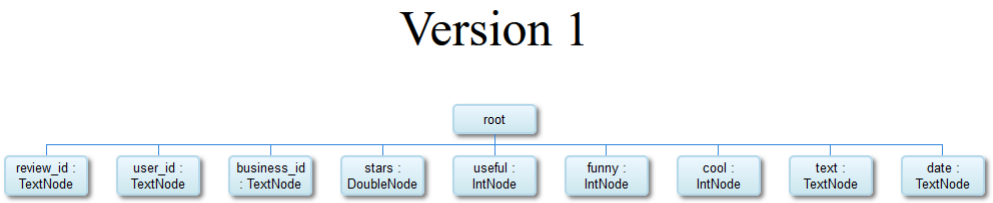
Εικόνα 4.2.3. Τμήμα της οπτικοποίησης εκδόσεων για το αρχείο “test_countries_2_entries”

Στην συνέχεια των πειραμάτων πήραμε το αρχείο “countries_small” το οποίο είχε 248 JSON αντικείμενα και έβγαλε 159 εκδόσεις, το οποίο δεν έχει καλή αναλογία με το πόσο άλλαξαν τα αντικείμενα μέσα στον χρόνο στο συγκεκριμένο αρχείο. Σε αντίθεση το αρχείο “profiles” που έχει λίγο μεγαλύτερο μέγεθος αλλά, συνολικά 1515 JSON αντικείμενα αρκετά μεγάλη διαφορά με το “countries_small” που έχει συνολικά 248, έβγαλε 11 εκδόσεις. Μεγαλώνοντας το μέγεθος των αρχείων στα πειράματα

παρατηρούμε ότι στα αρχεία countries_big και cards που έχουν περίπου ίδιο μέγεθος υπάρχει πάλι μεγάλη διαφορά στον αριθμό JSON αντικειμένων και εκδόσεων που βρέθηκαν σε αυτά τα αρχεία. Στο αρχείο "cards" τα JSON αντικείμενα βρίσκονται μέσα σε ArrayNode και το σύστημα βρίσκει 464 εκδόσεις σε 9.059 JSON αντικείμενα. Ενώ, στο αρχείο "countries_big" βρίσκει 3 εκδόσεις σε 21.640 JSON αντικείμενα, αρκετά μεγάλη διαφορά στην αναλογία JSON αντικεμενα με εκδόσεις που βρέθηκαν σε αυτά τα δύο αρχεία. Τέλος, στα τέσσερα τελευταία αρχεία έχει αυξηθεί σημαντικά το μέγεθος τους, με το τελευταίο αρχείο να φτάνει ως 5.22.145 KB και το σύστημα να χρειάζεται 35.914 δευτερόλεπτα να το επεξεργαστεί. Αν παρατηρήσει κανείς ενώ στα προηγούμενα αρχεία οι εκδόσεις αλλάζαν αρκετά στα μεγάλα αρχεία τα JSON αντικείμενα μέσα σε αυτά ακολουθούν μία και μόνο έκδοση εκτός από το "city_inspections" που έχει 6 εκδόσεις. Στην εικόνα 4.2.3 και 4.2.4 φαίνονται οι εκδόσεις που έβγαλαν τα αρχεία "tip" και "review" αντίστοιχα.

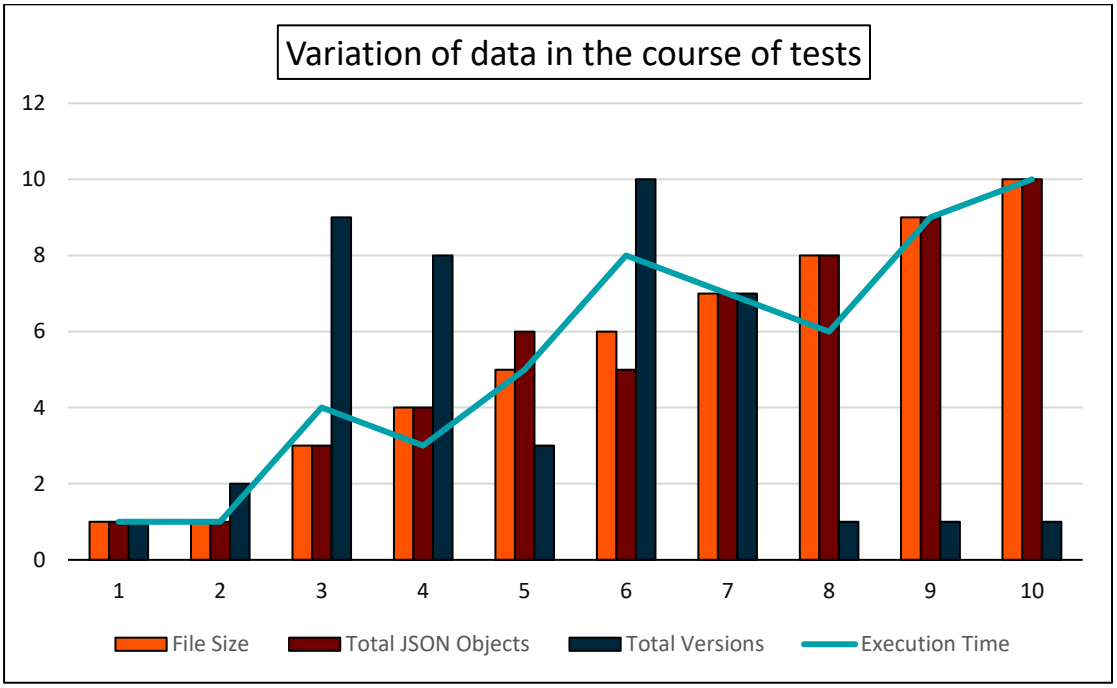


Εικόνα 4.2.3. Οπτικοποίηση εκδόσεων για το αρχείο "tip"



Εικόνα 4.2.4. Οπτικοποίηση εκδόσεων για το αρχείο "review"

4.2.1 Ανάλυση Αποτελεσμάτων



Εικόνα 4.2.5. Μεταβολή μεγέθους αρχείου, συνολικών JSON αντικειμένων, εκδόσεων στην πάροδο των πειραμάτων

Τα αρχεία που χρησιμοποιήθηκαν ως αρχεία εισόδου στα πειράματα, είχαν αρκετά μεγάλη διαφορά στις τιμές μεταξύ τους όπως φαίνεται στον Πίνακα 4.2.1. Εν συνεπεία, αποφασίσαμε για λόγους ευκολίας να προχωρήσουμε σε κανονικοποίηση των τιμών σε κλίμακα 1-10. Πιο συγκεκριμένα:

- **File Size:** Το μέγεθος των αρχείων μεγενθύνεται όσο αυξάνεται ο αριθμός του πειράματος.
- **Total JSON Objects:** Τα συνολικά JSON αντικείμενα μέσα στα αρχεία, αυξομειώνονται. Αναλυτικότερα, μέχρι το πέμπτο πείραμα παρατηρούμε αύξηση, ενώ μεταξύ πέμπτου και έκτου πειράματος υπάρχει μείωση. Στην συνέχεια συνεχίζεται η αύξηση από το έκτο πείραμα εως το τέλος.
- **Total Versions:** Οι εκδόσεις που βρέθηκαν από το σύστημα, παρατηρούμε ότι δεν ακολουθούν συγκεκριμένο ρυθμό.
- **Execution Time:** Ο χρόνος εκτέλεσης της επεξεργασίας των πειραμάτων, εναλλάσει τον ρυθμό του στα πρώτα έξι πειράματα ανάλογα με τις τιμές των υπόλοιπων κατηγοριών. Εν αντιθέση, από το έβδομο πείραμα και μετά εξαρτάται από το μέγεθος του αρχείου και τα συνολικά JSON αντικείμενα που βρίσκονται σε αυτά.

4.2.1.1 Συμπέρασμα Ανάλυσης Αποτελεσμάτων

Συμπερασματικά, παρατηρήθηκε ότι διατηρώντας τον αριθμό των εκδόσεων που βρέθηκαν, ο χρόνος εκτέλεσης αυξάνεται σε συνάρτηση με το μέγεθος του αρχείου και το συνολικό αριθμό των JSON αντικειμένων που περιέχονται σε αυτά. Αντιθέτως, όσο ο αριθμός των εκδόσεων δεν ακολουθεί σταθερό ρυθμό, οδηγούμαστε στο συμπέρασμα ότι, ο χρόνος εκτέλεσης εξαρτάται και από τις εκδόσεις. Ειδικότερα, στα πειράματα 3, 4 που αυξήθηκε το μέγεθος του αρχείο και τα συνολικά JSON αντικείμενα αλλά, ο αριθμός των εκδόσεων μειώθηκε, τότε ο ρυθμός του χρόνου εκτέλεσης οδηγήθηκε σε μείωση. Επιπλέον, στα πειράματα 6–8 παρατηρούμε ότι ο χρόνος εκτέλεσης ακολουθεί την ίδια πορεία με τα πειράματα 3,4. Δηλαδή, καθώς αυξάνεται το μέγεθος του αρχείου και ο αριθμός των JSON αντικειμένων, ο χρόνος εκτέλεσης μειώνεται όπως και οι εκδόσεις. Εν κατακλείδι, ο χρόνος εκτέλεσης εξαρτάται και από τις τρεις παραμέτρους.

Κεφάλαιο 5. Επίλογος

5.1 Σύνοψη και συμπεράσματα

Στόχος της παρούσας διπλωματικής εργασίας ήταν να κατασκευαστεί ένα σύστημα το οποίο θα επιτρέπει στον χρήστη την εισαγωγή ενός αρχείου δεδομένων τύπου JSON με αποτέλεσμα να προβάλλονται οπτικοποιημένα το schema των δεδομένων, καθώς και οι αλλαγές που πραγματοποιήθηκαν στη διάρκεια του οποίου οι διαχειριστές των δεδομένων είτε μπορεί να πρόσθεσαν, να αφαίρεσαν τιμές ή να μετέβαλλαν το τύπο των τιμών.

Τελικά υλοποιήθηκε ένα σύστημα που επιτρέπει στο χρήστη:

- Να μπορεί να φορτώσει στο σύστημα το αρχείο δεδομένων τύπου JSON που θέλει να εισάγει ώστε να ξεκινήσει η διαδικασία εξαγωγής του σχήματος των δεδομένων.
- Να δει τις εκδόσεις του σχήματος δεδομένων τύπου JSON που βρήκε το σύστημα.
- Να δει τα χαρακτηριστικά που παρουσιάζει η εξέλιξη του σχήματος δεδομένων τύπου JSON.

5.2 Μελλοντικές επεκτάσεις

Υπάρχει μια μεγάλη λίστα από πράγματα τα οποία κάποιος θα μπορούσε να προσθέσει στο μέλλον, τα οποία περιγράφονται παρακάτω στις επόμενες παραγράφους.

Επιπλέον τύπος αρχείων εισόδου. Τα αρχεία εισόδου υποχρεωτικά είναι ".json", με το σύστημα να μην δέχεται σαν αρχεία εισόδου αρχεία που έχουν άλλο τύπο. Αυτή η επέκταση θα βοηθούσε στο να μπορεί ο χρήστης να εισάγει περισσότερα αρχεία και να μην τον υποχρεώνει να είναι τύπου JSON.

Αναζήτηση κλειδιών. Όπως συμβαίνει και στις βάσεις υπάρχουν τα πρωτεύοντα κλειδία τα οποία περιέχουν μοναδικές τιμές και δεν μπορούν να είναι Null. Το σύστημα μετά το διάβασμα των JSON αντικειμένων θα μπορούσε να βρίσκει τα πεδία που θα μπορούσαν να χρησιμοποιηθούν σαν πρωτεύοντα κλειδιά σε μια βάση δεδομένων. Αυτή η επέκταση θα μπορούσε να βοηθήσει στην μεταφορά ενός JSON αρχείου σε μια βάση δεδομένων.

Δημιουργία βάσεις δεδομένων. Με την βοήθεια της προηγούμενης επέκτασης θα μπορούσε κάποιος να δημιουργήσει μια βάση δεδομένων με τα δεδομένα που υπάρχουν στα JSON αρχεία. Αυτή η επέκταση θα βοηθούσε στο να μπορεί κάποιος να αποθηκεύσει τις πληροφορίες των JSON αντικειμένων σε μια βάση δεδομένων, ώστε να είναι πιο "σίγουρος" για την ασφάλεια των δεδομένων.

Δημιουργία αρχείων βάσει έκδοσης. Μπόρει κάποιος να δημιουργήσει αρχεία τα οποία αποθηκεύουν τα JSON αντικείμενα με βάση την έκδοση που ακολουθούν. Αυτή η επέκταση έχει ως αποτέλεσμα κάποιος να χωρίζει το αρχείο εισόδου σε μικρότερα και να ξεχωρίζει τις εκδόσεις μέσα σε αυτά, έτσι ώστε κάθε μικρότερο (ή ίσο αν υπάρχει μία και μόνο έκδοση) αρχείο σε σχέση με το αρχείο εισόδου να περιέχει μόνο τα JSON αντικείμενα που ακολουθούν μια συγκεκριμένη έκδοση.

Επιλογή έκδοσης. Με βάση την προηγούμενη επέκταση μπορούμε να δώσουμε την δυνατότητα στον χρήστη στο μέλλον να μπορεί να επιλέξει αυτός με βάση τις εκδόσεις που έβγαλε το σύστημα, ποια έκδοση θέλει ακριβώς να αποθηκευτεί και σε ποιο αρχείο. Έτσι, το σύστημα θα παίρνει την επιλογή του χρήστη και θα ξεχωρίζει από το αρχικό αρχείο εισόδου τα JSON αντικείμενα που ακολουθούν την έκδοση που επέλεξε ο χρήστης. Με αυτή την επέκταση ο χρήστης θα έχει την δυνατότητα να μπορεί να επιλέξει και να ξεχωρίσει την έκδοση που χρειάζεται. Για παράδειγμα αν υπάρχει ένα αρχείο που έγιναν αλλάγες μέσα στον χρόνο στα JSON αντικείμενα και ο χρήστης θέλει να κρατήσει την νεότερη έκδοση, θα επιλέξει την τελευταία και θα την καταχωρήσει σε ένα αρχείο. Σε συνδυασμό με την επέκταση "Αναζήτησης κλειδιών" και "Δημιουργία βάσεις δεδομένων" ο χρήστης θα μπορεί να δημιουργεί μια βάση δεδομένων με τα δεδομένα των JSON αντικειμένων που ακολουθούν μόνο την τελευταία και νεότερη έκδοση.

Αναζήτηση έκδοσης στα ArrayNode. Το σύστημα βρίσκει μόνο την έκδοση στα JSON αντικείμενα είτε το βασικό είτε τα εμφωλευμένα, έτσι θα μπορούσε κάποιος να ψάχνει και τις εκδόσεις και τις αλλαγές που έγιναν στα πεδία που είναι τύπου ArrayNode. Με αυτή την επέκταση θα μπορούσε κάποιος να έχει περισσότερο έλεγχο στα JSON αντικείμενα που υπάρχουν στο αρχείο.

Διαφορετικός τρόπος οπτικοποίησης. Οι εκδόσεις μπορούν να οπτικοποιηθούν με διαφορετικούς τρόπους. Όπως αναφέραμε στο κεφάλαιο 3 και συγκεκριμένα στην υποενότητα "3.4.2.1 Υλοποίηση Κώδικα" οι εκδόσεις θα μπορούσαν να οπτικοποιηθούν και με Treemaps Chart, αλλά ο χρήστης δεν θα μπορούσε να διακρίνει εύκολα τις αλλαγές στις εκδόσεις. Με την ίδια λογική μπορεί κάποιος στο μέλλον να αλλάξει τον τρόπο με τον οποίο οπτικοποιούνται οι εκδόσεις.

Βιβλιογραφία

[BBC+99] P.A. Bernstein, T. Bergstraesser, J. Carlson, S. Pal, P. Sanders, D. Shutt. Microsoft Repository Version 2 and the Open Information Model. Information Systems 24(2), pp. 71-98, 1999.

[BaCR94] V.R. Basili, G.Caldiera, H.D. Rombach. The Goal Question Metric Approach. Encyclopedia of Software Engineering, pp. 528-532, John Wiley & Sons, Inc, 1994. Also available at http://www.cs.umd.edu/users/basili/papers.html

[Dean97] E.B. Dean. Quality Functional Deployment from the Perspective of Competitive Advantage. Available at http://mijuno.larc.nasa.gov/dfc/qfd.html

[JJQV98] M. Jarke, M.A. Jeusfeld, C. Quix, P. Vassiliadis. Architecture and quality in data warehouses. In Proc. 10th Conference on Advanced Information Systems Engineering (CAiSE '98), pp. 93-113, Pisa, Italy, June 1998.

[JaVa97] M. Jarke, Y. Vassiliou. Foundations of data warehouse quality – a review of the DWQ project. In Proc. 2nd Intl. Conference Information Quality (IQ-97), pp. 299-313, Cambridge, Mass., USA, June 1997.

[Orr98] K. Orr. Data quality and systems theory. In Communications of the ACM, 41(2), pp. 54-57, Feb. 1998.