

**ΕΞΑΓΩΓΗ ΚΑΙ ΟΠΤΙΚΟΠΟΙΗΣΗ ΕΚΔΟΣΕΩΝ ΤΟΥ ΣΧΗΜΑΤΟΣ ΔΕΔΟΜΕΝΩΝ ΤΥΠΟΥ JSON**

**Θωμάς Σιώζος**

**Διπλωματική Εργασία**

Επιβλέπων: Π. Βασιλειάδης

Ιωάννινα, Ιούνιος 2020

**Τμήμα Μηχ. Η/Υ & Πληροφορικής**

**Πανεπιστήμιο Ιωαννίνων**

**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 objects στο αρχείο εισόδου εξάγονται και οπτικοποιούνται. Για την κατασκευή και λειτουργία του εργαλείου χρησιμοποιήθηκαν βιβλιοθήκες για την εξαγωγή των json objects 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

1.1 Αντικείμενο της διπλωματικής 1

1.2 Οργάνωση του τόμου 1

Κεφάλαιο 2. Περιγραφή Θέματος 3

2.1 Στόχος της εργασίας 3

2.2 Σχετικές εργασίες και τεχνολογίες 3

2.3 Ανάλυση απαιτήσεων 3

Κεφάλαιο 3. Σχεδίαση & Υλοποίηση 5

3.1 Ορισμός προβλήματος και αλγόριθμοι επίλυσης 5

3.2 Σχεδίαση και αρχιτεκτονική λογισμικού 5

3.3 Σχεδίαση και αποτελέσματα ελέγχου του λογισμικού 6

3.4 Λεπτομέρειες εγκατάστασης και υλοποίησης 6

3.5 Επεκτασιμότητα του λογισμικού 6

Κεφάλαιο 4. Πειραματική Αξιολόγηση 7

4.1 Μεθοδολογία πειραματισμού 7

4.2 Αναλυτική παρουσίαση αποτελεσμάτων 7

Κεφάλαιο 5. Επίλογος 11

5.1 Σύνοψη και συμπεράσματα 11

5.2 Μελλοντικές επεκτάσεις 11

# Εισαγωγή

## Αντικείμενο της διπλωματικής

Το JavaScript Object Notation (JSON) είναι ένα αρχείο που ακολουθεί το πρωτόκολλο “open standard” (Είναι διαθέσιμο δημόσια) και είναι γραμμένο σε γλώσσα ανθρώπου. Τα JSON δημιουργήθηκαν στην ανάγκη για real-time server-to-browser πρωτόκολλο επικοινωνίας χωρίς να χρησιμοποιεί plugins όπως Flash ή Java applets στις αρχές του 2000. 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.

To 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

}

## 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.

# Περιγραφή Θέματος

## 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.

## 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.

### mJson

[**https://bolerio.github.io/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 objects σε Java objects 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 object 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.

### JSON-P

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

Το JSΟΝ 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 αρχείων 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.

### JSON.simple

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

Το Json.simple συγκαταλέγεται στα toolkits και χρησιμοποιείται για parsing 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 objects.

### GSON

[**https://github.com/google/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 string σε ένα ισοδύναμο Java object και το ανάποδο. Υποστηρίζει απλές μεθόδους όπως toJson() και fromJson() για να μετατρέπει ο προγραμματιστής Java objects σε Json objects και το αντίστροφο, να διαχειρίζεται collections, generic types και nested classes. Επίσης, υποστηρίζει serialization/deserialization. Το serialization είναι ο τρόπος να δημιουργήσουμε Json objects μέσω Java objects και το deserialization το ανάποδο. Στην περίπτωση που γνωρίζουμε από πριν τα πεδία που υπάρχουν στα δεδομένα των Json objects, χωρίς αυτά να αλλάζουν μέσα στον χρόνο, η GSON επιτρέπει στον προγραμματιστή να έχει τον πλήρη έλεγχο της όλης επεξεργασίας των Json objects. Αυτό σημαίνει, ότι δίνεται η δυνατότητα ο προγραμματιστής να χρησιμποποιήσει POJO κλάσεις για την χαρτογράφηση των Json objects. Ένα απλό deserialization παράδειγμα είναι όταν έχουμε Json objects με πεδία name, last name, age τότε ο προγραμματιστής μπορεί να φτιάξει μια κλάση Person με τα πεδία αυτά ώστε όταν περνάει το Json object 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 objects σε Json και το ανάποδο.
* Deserialization/Serialization.
* Υποστηρίζει Java Generics.
* Υποστηρίζει περίπλοκα αντικείμενα.

### Jackson

[**https://github.com/FasterXML/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 objects σε Java objects 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 objects.
* 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.

### 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.

### Benchmark

Για να αποφασίσουμε ποια βιβλιοθήκη θα χρησιμοποιήσουμε για το σκοπό της διπλωματικής εργασίας, oργανώσαμε μια συγκριτική μελέτη(benchmark). Πραγματοποιήθηκε με 2 σενάρια, στο ένα περάστηκαν μεγάλα αρχεία(25 MB) και στο δεύτερο περάστηκαν μικρά αρχεία (1 ΚΒ). 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 τo 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 Jacκson 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.

### Conclusion

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 objects και έλλειψη στοίχισης, δηλαδή σε κάθε γραμμή του αρχείου υπάρχει ένα Json object, 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 objects. 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 object, 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 objects.
* Υποστηρίζει δέντρα.
* Πιο γρήγορη και στα δύο μεγέθη αρχείων που εκχωρήθηκαν στα πειράματα που έγιναν.
* Community.
* Υποστηρίζει περίπλοκα Json αντικείμενα.

Τo 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.

## 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 objects 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.

# Σχεδίαση & Υλοποίηση

## 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.

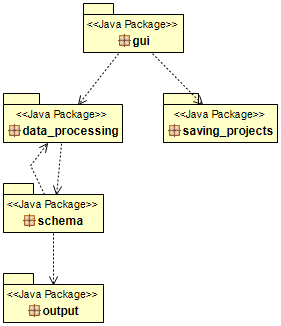
* **Αφαίρεση πεδίου:** Αν το πεδίο αφαιρέθηκε από το json αντικείμενο σε σχέση με την προηγούμενη έκδοση.
* **Πρόσθεση πεδίου:** Αν το πεδίο προστέθηκε στο json αντικείμενο σε σχέση με την προηγούμενη έκδοση.
* **Αλλαγή τύπου τιμής**: Αν η τιμή ενός πεδίου άλλαξε τύπο.

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

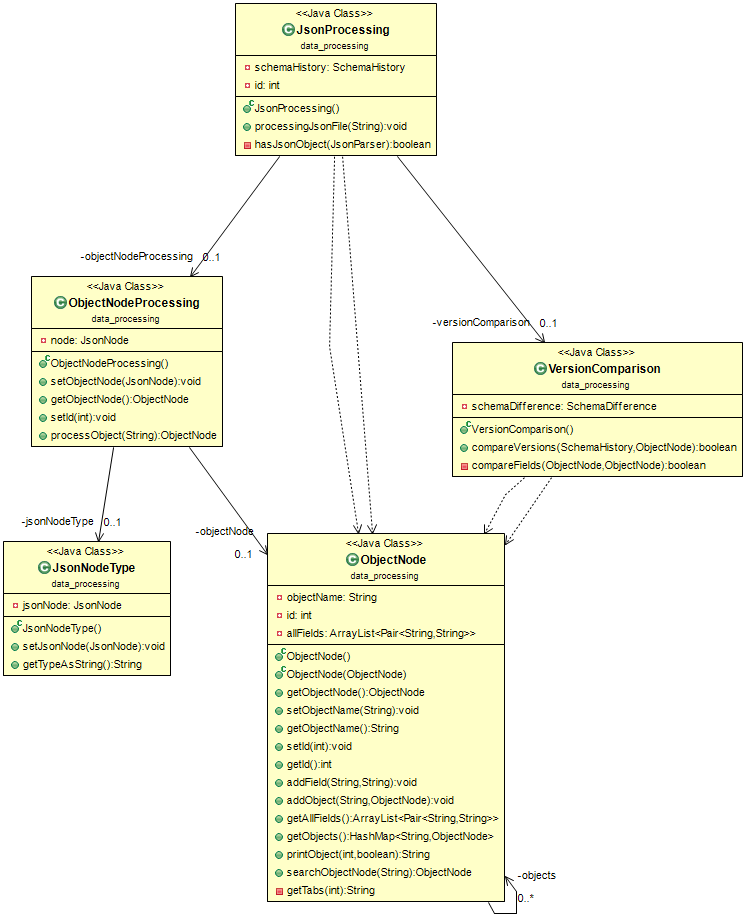
* **IntNode:** Όταν ο τύπος της τιμής του πεδίου είναι ακέραιος αριθμός.
* **DoubleNode:** Όταν ο τύπος της τιμής του πεδίου είναι ίδιος με αυτόν όταν μιλάμε για Double μεταβλητές στην Java.
* **ArrayNode:** Όταν ο τύπος της τιμής του πεδίου είναι ένας πίνακας ο οποίος μπορεί να περιέχει πεδία με οποιονδήποτε 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.

## 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.



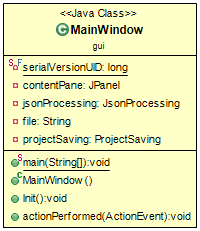
**Σχήμα 3.2.1. Διάγραμμα Πακέτων.**



**Σχήμα 3.2.1: For the data\_processing package Class diagram**

### Πακέτο data\_processing.

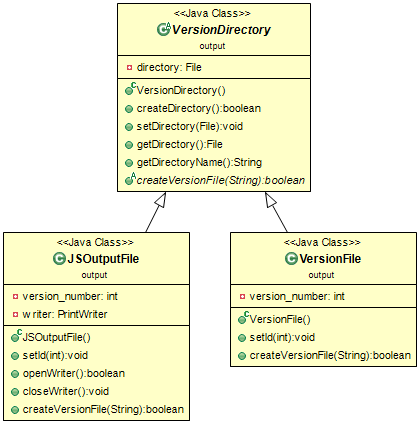
Το συγκεκριμένο πακέτο περιέχει τις κλάσεις που χρησιμοποιήθηκαν για να διαβάσει το αρχείο, να συγκρίνει τις εκδόσεις των αντικειμένων μέσα στο αρχείο json και να αποθηκεύσει σε κατάλληλες δομές τα δεδομένα που χρειάζεται το λογισμικό. Το πακέτο αρχικά περιέχει την κλάση JsonNodeType η οποία δέχεται 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 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**

### Πακέτο gui

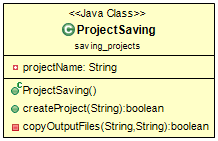
Το πακέτο gui είναι υπεύθυνο για την διεπαφή του λογισμικού με τον χρήστη και έχει την κλάση MainWindow. Δέχεται το αρχείο από τον χρήστη το επεξεργάζεται και ο χρήστης μπορεί να δει τις εκδόσεις του αρχείο, να αποθηκεύσει το project του και να ανεβάσει αν το επιθυμεί projects που ήδη υπάρχουν στο σύστημα.



**Σχήμα 3.2.3: For the output package Class diagram**

### Πακέτο output

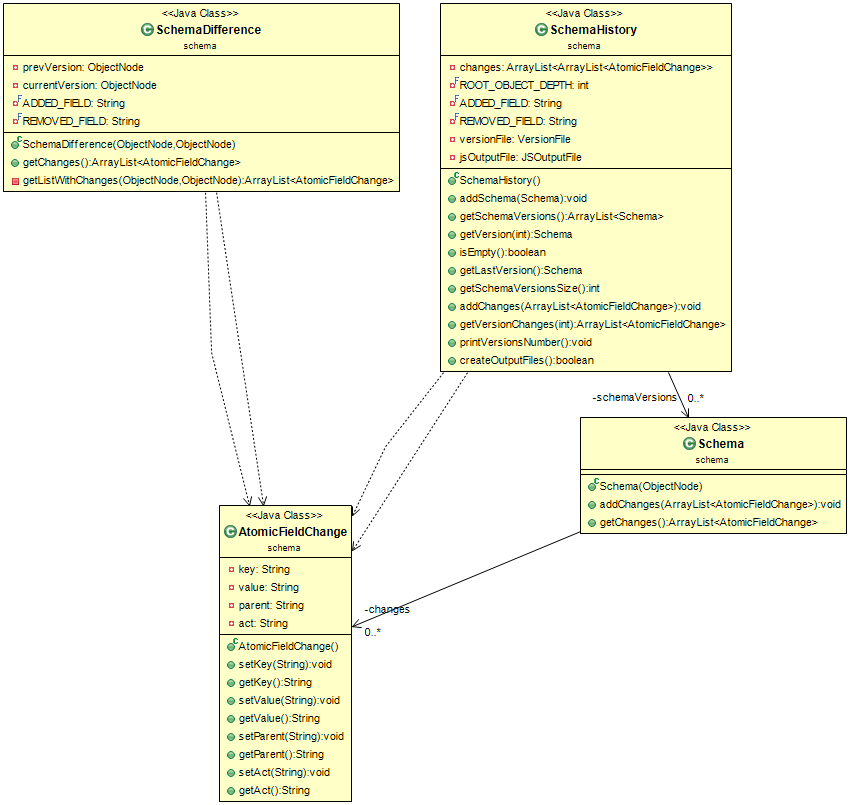
Το πακέτο output είναι υπεύθυνο για την δημιουργία του φακέλου και των αρχείων που έχουν τις εκδόσεις και περιέχει τις κλάσεις VersionDirectory, JSOutputFile και VersionFile. Η κλάση VersionDirectory δημιουργεί έναν φάκελο output ο όποιος περιέχει σε αρχεία τύπου csv τις εκδόσεις που περιέχονται στο 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**

### 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**

### 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.

### 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 τo OrgChart επειδή φαίνεται να είναι το πιο κατάλληλο ώστε να μπορεί ο χρήστης να διακρίνει τις διαφορές ανάμεσα στις εκδόσεις.

### 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 objects.

## 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.

## Λεπτομέρειες εγκατάστασης και υλοποίησης

Στην ενότητα αυτή περιγράφονται τα χαρακτηριστικά της συγκεκριμένης υλοποίησης, όπως η πλατφόρμα ανάπτυξης και εκτέλεσης, τα προγραμματιστικά εργαλεία, οι απαιτήσεις της εφαρμογής σε hardware, κ.λ.π.

Είναι σημαντικό να παρατεθούν με συγκροτημένο τρόπο οι απαραίτητες ρυθμίσεις ώστε το πρόγραμμα να εγκατασταθεί και να εκτελείται σωστά. Αυτό δεν αφορά μόνο τις ρυθμίσεις του προγραμματιστικού περιβάλλοντος, αλλά π.χ., και τι πρέπει να μπει σε κάποια αρχεία αρχικοποίησης του συστήματος κ.ο.κ.

## Επεκτασιμότητα του λογισμικού

Όταν σχεδιάζουμε το λογισμικό σκεφτόμαστε και πώς θα επεκταθεί και συντηρηθεί στο μέλλον. Άρα κάνουμε και μια λίστα από πιθανές επεκτάσεις. Στην ενότητα αυτή περιγράφονται τα σημεία του κώδικα τα οποία θα χρειαστεί να αλλαχθούν σε μελλοντικές τέτοιες επεκτάσεις. Επίσης σημειώνουμε τα σημεία που έχουν hard-coded λειτουργικότητες που θα πρέπει να συντηρηθούν στο μέλλον.

# Πειραματική Αξιολόγηση

Στην ενότητα αυτή παρουσιάζουμε αναλυτικά την πειραματική αξιολόγηση της μεθόδου μας.

## Μεθοδολογία πειραματισμού

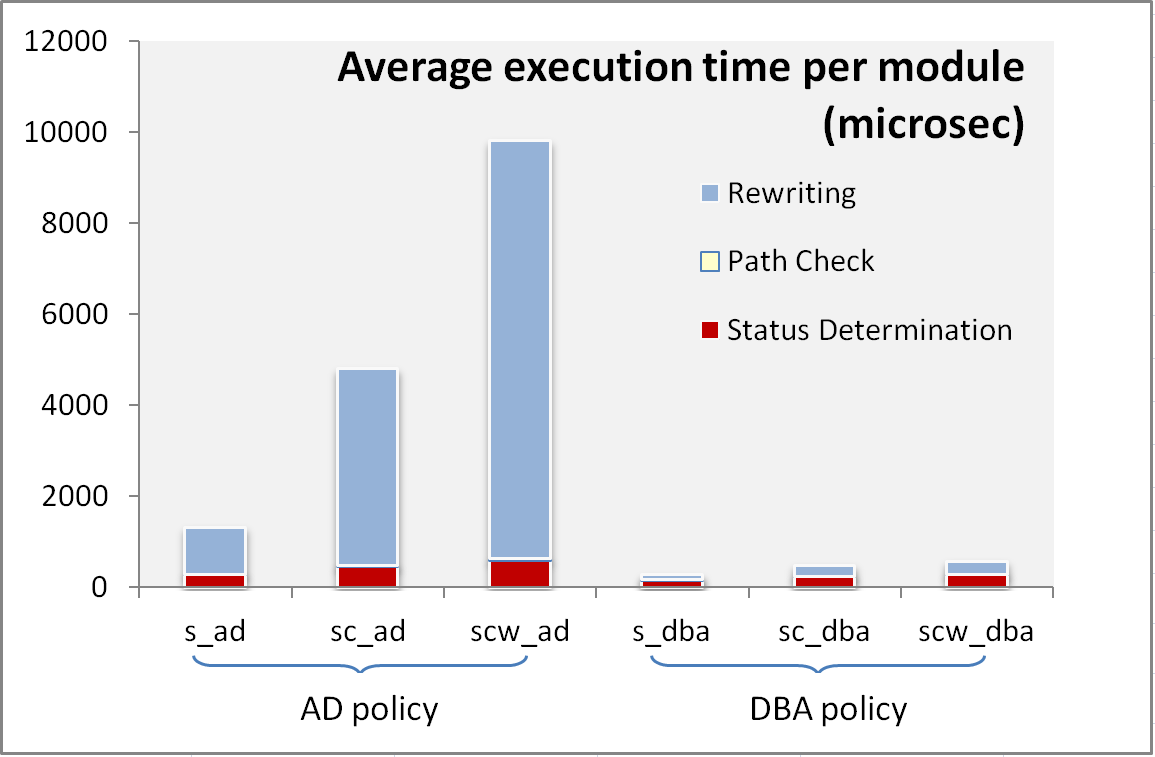
Περιγραφή του σκοπού των πειραμάτων

Δεδομένα που χρησιμοποιήθηκαν, αντίπαλοι αλγόριθμοι και τεχνικές (ή ότι άλλο είναι αρμόζον για την περίσταση)

Περιγραφή του περιβάλλοντος στο οποίο διεξήχθησαν τα πειράματα.

## Αναλυτική παρουσίαση αποτελεσμάτων

Για κάθε παράμετρο που μας ενδιαφέρει να μετρήσουμε, αναλύουμε τα αποτελέσματα του σχετικού πειράματος. Περιγράφουμε τι μεταβάλλαμε, τι μετρήσαμε και τι συμπεράσματα βγαίνουν από το πείραμα. Συνιστάται η παράθεση διαγραμμάτων και δεδομένων από τα διεξαχθέντα πειράματα και οπωσδήποτε ο σχετικός σχολιασμός.



Εικόνα 4.1 Χρόνος εκτέλεσης (microsec) ως συνάρτηση (α) της κατανομής αρμοδιοτήτων και (β) της πολιτικής διαχείρισης της διάδοσης αλλαγών. Εσωτερικά σε κάθε στήλη αναπαρίστανται(i) ο χρόνος επανεγγραφής, (ii) ο χρόνος ελέγχου μονοπατιών διάδοσης και (iii) ο χρόνος αποτίμησης της κατάστασης κόμβων.

Η γενική ιδέα είναι ότι περιγράφουμε πώς μεταβάλλονται οι τιμές στον κάθετο άξονα, σε σχέση με τις τιμές στον οριζόντιο άξονα ή/και τις διάφορες τεχνικές που εμπλέκονται στο πείραμα. Πάντα αναφέρουμε τις μονάδες. Η ασφαλής λύση είναι να χρησιμοποιήσετε bar charts. Το παράδειγμα στην Εικόνα 4.1 που σας δίνεται είναι όσο πιο πολύπλοκο θα μπορούσε να είναι.

Για τους πίνακες: στο οπτικό αποτέλεσμα, ελαχιστοποιήστε τα pixels που δεν είναι περιεχόμενο. Αυτό αφορά κυρίως τα borders αλλά και το χρώμα στο background (το πολύ πολύ να μπει ένα απαλό φόντο σε όλο τον πίνακα, όπως π.χ., έχουμε στην Εικόνα 2.1). Η έμφαση πρέπει να είναι στο να αναδειχθεί το περιεχόμενο του πίνακα. Προσέξτε, οι γραμμές του πίνακα, να έχουν στην παράγραφό τους “Keep with next” ώστε ο πίνακας να μη σπάει σε διαφορετικές σελίδες.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Breakdown of Tables over their Activity Class  (Percentages over Total #Tables) | | | | | | | |
|  |  | Activity Class | | | Activity Class (%) | | |
|  | Total  #Tables | RIGID | QUIET | ACTIVE | RIGID | QUIET | ACTIVE |
| Atlas | 88 | 18 | 43 | 27 | ***20%*** | **49%** | 31% |
| BioSQL | 45 | 16 | 13 | 16 | **36%** | ***29%*** | **36%** |
| Castor | 91 | 57 | 31 | 3 | **63%** | 34% | ***3%*** |
| SlashCode | 68 | 15 | 38 | 15 | ***22%*** | **56%** | ***22%*** |
| Zabbix | 56 | 23 | 30 | 3 | 41% | **54%** | ***5%*** |

Πίνακας 4.1 Κατανομή πινάκων σε διαφορετικές κλάσεις δραστηριότητας σε απόλυτες τιμές και ποσοστά (για κάθε σύνολο δεδομένων, η μέγιστη τιμή με **κόκκινα έντονα** γράμματα και η ελάχιστη με μπλε πλαγιαστά)

Συνιστάται η συντήρηση ενός ΚΑΘΑΡΟΥ spreadsheet, είτε συνολικά για την εργασία, είτε ανά πείραμα, και η παράδοσή του στο τέλος, μαζί με τον κώδικα, τα δεδομένα εισόδου και εξόδου, και την παρούσα αναφορά.

# Επίλογος

Στην ενότητα αυτή συνοψίζουμε τη συνεισφορά και τα αποτελέσματα της εργασίας και παραθέτουμε σκέψεις για μελλοντικές επεκτάσεις της.

## Σύνοψη και συμπεράσματα

Στην ενότητα αυτή συνοψίζουμε τα αποτελέσματα τις διπλωματικής (αντιγράφουμε την ενότητα 1.1 χρησιμοποιώντας αόριστο αντί για μέλλοντα χρόνο) και περιγράφουμε λίγο πιο αναλυτικά και τα όποια συμπεράσματα εξάγαμε.

## Μελλοντικές επεκτάσεις

Στην ενότητα αυτή δίνουμε μια λίστα με πράγματα που έχει νόημα / ενδιαφέρον να φτιαχτούν στο μέλλον.

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. |