

# ΕΞΑΓΩΓΗ ΕΚΔΟΣΕΩΝ ΑΠΟ ΑΡΧΕΙΟ 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

Με τον όγκο των δεδομένων στο internet να ανεβαίνει τρομακτικά και την ανάγκη πολλών συστημάτων να έχουν βάσεις δεδομένων, τα αρχεία json μπορούν να το προσφέρουν αυτό έχοντας ένα και μόνο αρχείο που περιέχει όλα τα δεδομένα που μπορεί να χρησιμοποιεί ένα σύστημα. Στην συγκεκριμένη εργασία κατασκευάστηκε ένα εργαλείο το οποίο παίρνει json αρχεία, τα επεξεργάζεται και εξάγει σε εικονική μορφή τις εκδόσεις που ακολουθούν τα json objects μέσα στο αρχείο. Για το σκοπό αυτό, χρησιμοποιήθηκαν βιβλιοθήκες για την εξαγωγή των json objects από το αρχείο, αλγόριθμοι για την επεξεργασία και την σύγκριση τους. Στην συνέχεια, υλοποιήθηκε ένα γραφικό περιβάλλον στο οποίο ο χρήστης μπορεί να εισάγει το αρχείο json που θέλει, να δει το αποτέλεσμα σε εικονική μορφή, να αποθηκεύσει το project και τέλος να ανεβάσει ένα project που ήδη υπάρχει. Τέλος, δημιουργήθηκαν δύο αρχεία html και javascript για την οπτικοποίηση των εκδόσεων του αρχείου.

**Λέξεις Κλειδιά:** json, Jackson, json objects.

# Abstract

Summary in English (up to 200 words)

**Keywords:** <keyword 1>, <keyword 2>

# Πίνακας περιεχομένων

|   |           |
|---|-----------|
| <b>Κεφάλαιο 1. Εισαγωγή.....</b>                          | <b>1</b>  |
| 1.1 Αντικείμενο της διπλωματικής.....                     | 1         |
| 1.2 Οργάνωση του τόμου .....                              | 1         |
| <b>Κεφάλαιο 2. Περιγραφή Θέματος.....</b>                 | <b>3</b>  |
| 2.1 Στόχος της εργασίας.....                              | 3         |
| 2.2 Σχετικές εργασίες και τεχνολογίες .....               | 3         |
| 2.3 Ανάλυση απαιτήσεων .....                              | 3         |
| <b>Κεφάλαιο 3. Σχεδίαση &amp; Υλοποίηση.....</b>          | <b>5</b>  |
| 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         |
| <b>Κεφάλαιο 4. Πειραματική Αξιολόγηση.....</b>            | <b>7</b>  |
| 4.1 Μεθοδολογία πειραματισμού .....                       | 7         |
| 4.2 Αναλυτική παρουσίαση αποτελεσμάτων .....              | 7         |
| <b>Κεφάλαιο 5. Επίλογος.....</b>                          | <b>11</b> |
| 5.1 Σύνοψη και συμπεράσματα.....                          | 11        |
| 5.2 Μελλοντικές επεκτάσεις .....                          | 11        |





---

# Κεφάλαιο 1. Εισαγωγή

## 1.1 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 ως 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

Ποια τα κεφάλαια / ενότητες του τόμου αυτού (1 παράγραφος ανά κεφάλαιο).

(Τα κεφάλαια πρέπει να ξεκινούν σε ΜΟΝΗ ΣΕΛΙΔΑ, κι εδώ αφήνουμε κενή σελίδα επίτηδες, ώστε το κεφάλαιο 2 να ξεκινά στη σελ. 3. Αν δεν είχαμε αφήσει την παρούσα κενή σελίδα, το κεφάλαιο θα ξεκινούσε στη σελ. 2 που είναι ζυγή.)

---

## 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, έχοντας τελειώσει η επεξεργασία του αρχείου που εισήγαγε ο χρήστης θα έχει την επιλογή να δει τα χαρακτηριστικά που παρουσιάζει η εξέλιξη του σχήματος του JSON αρχείου.

---

## 2.2 Σχετικές εργασίες και τεχνολογίες

Υπάρχουν αρκετές βιβλιοθήκες που μπορούν να βοηθήσουν κάποιον όταν θέλει να επεξεργαστεί αρχεία τύπου JSON προγραμματίζοντας σε γλώσσα java. Μερικές από αυτές είναι οι mJson, JSON.simple, JSON-P, GSON, Jackson, με τις GSON και Jackson να ναι οι πιο δημοφιλείς από αυτές, παρουσιάζοντας αρκετές ομοιότητες μεταξύ τους. Όπως θα δούμε παρακάτω 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 ασχοληθήκαμε με mJSON και JSON-P στην αρχή και μετά με τις δύο μεγαλύτερες GSON, Jackson.

### mJSON

<https://bolero.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.
- Ένας μοναδικός τύπος μεταβλητών, τα πάντα είναι Json και δεν χρειάζεται type casting.
- Γρήγορο parsing.
- Περιεκτικές μεθόδους για διάβασμα, τροποποίηση, αντιγραφή και συγχώνευση.
- Μέθοδους για έλεγχο του τύπου των τιμών και πρόσβαση στον έλεγχο τους.
- Ένα java source file για ολόκληρη την βιβλιοθήκη, χωρίς εξωτερικές εξαρτήσεις.

---

## JSONP

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

Η JSON Processing(JSON-P) είναι ένα Java API που επιτρέπει parsing ή δημιουργία json αρχείων. Είναι παρόμοιο με το JAXP, ένα Java API για επεξεργασία XML αρχείων, συμβατικό με τις κλάσεις της Java για τα δεδομένα και αρκετά εύκολα στην χρήση του. Η βιβλιοθήκη 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/>

Η Json.simple μια μικρή βιβλιοθήκη για parsing και δημιουργία json αρχείων. Προσφέρει δυνατότητες κρατώντας τη βιβλιοθήκη μικρή και γρήγορη. Μερικές δυνατότητες είναι υποστήριξη της μετατροπής των δεδομένων σε data structures της java, υψηλή απόδοση και δεν έχει εξάρτηση με εξωτερικές βιβλιοθήκες. Είναι μικρή βιβλιοθήκη οπότε όλες οι λειτουργίες της περιέχονται σ' ένα πακέτο.

---

## 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 vice versa, να διαχειρίζεται collections, generic types και nested classes. Επίσης υποστηρίζει serialization/deserialization που μπορούν εξειδικευθούν. Το serialization είναι ο τρόπος να δημιουργήσουμε Json objects μέσω Java objects και το deserialization το ανάποδο. Όσο αναφορά το custom data part επιτρέπει στον προγραμματιστή να έχει τον πλήρη έλεγχο της όλης επεξεργασίας 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 access στην GSON, την βασική κλάση για να δουλέψει κανείς με την GSON.
- `com.google.gson.annotations`: Προσφέρει annotation τύπους για χρήση με την GSON.
- `com.google.gson.reflect`: Διαχειρίζεται πληροφορίες από generic τύπους.
- `com.google.gson.stream`: Προσφέρει κλάσεις για reading/writing σε Json-encoded τιμές.

Μερικά χαρακτηριστικά της GSON είναι:

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

---

## Jackson

<https://github.com/FasterXML/jackson>

Η Jackson είναι ένας high-performance JSON processor για την Java 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 user-friendly, λόγω του αυξημένου μεγέθους της, όταν ξεκινήσαμε να διαβάζουμε ένα json file 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 user friendly. 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: Ορίζει ένα low-level streaming API και περιλαμβάνει JSON-specific implementations.
- jackson-annotations: Περιλαμβάνει standard Jackson annotations.
- Jackson-databind: Implements data-binding, υποστηρίζει streaming πακέτα και εξαρτάται από τα παραπάνω πακέτα.

Πέρα από 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 (JSON to/from POJOs).
- Υποστηρίζει Java Generics.
- Υποστηρίζει περίπλοκα αντικείμενα.
- Treebased data model.
- Mix-in Annotations.
- Polymorphic types.
- Materialized interfaces.



---

## JMH

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

Η Java Microbenchmark Harness (JMH) είναι η βιβλιοθήκη της java που χρησιμοποιήσαμε για να υλοποιήσουμε το benchmark και να πάρουμε τους χρόνους που χρειάζεται κάθε βιβλιοθήκη για να κάνει parsing στα αρχεία json που βάλαμε. Η βιβλιοθήκη έχει κάποιες ορολογίες:

- **Trial:** Το benchmark τρέχει για όσες φορές είναι το trial, τον οποίο όρο μπορούμε να τον αποκαλούμε και fork.
- **Warmup:** Για κάθε trial, 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 οι πραγματικές μετρήσεις που θα βγάλουν και τα αποτελέσματα στο τέλος που θα χρησιμοποιήσουμε.

Η βιβλιοθήκη 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, το benchmark πραγματοποιήθηκε με 2 σενάρια, στο ένα περάστηκαν μεγάλα αρχεία(25 MB) και στο δεύτερο περάστηκαν μικρά αρχεία (1 KB). Τα μεγάλα αρχεία βρίσκονται στο αρχείο photos.json και τα μικρά στο αρχείο test\_countries\_2\_entries.json. 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 των μεγάλων αρχείων φαίνονται στον πίνακα παρακάτω:

| 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

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 για τα μικρά αρχεία που φαίνονται στον παρακάτω πίνακα:

| 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

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

Αν συγκρίνουμε και τους δύο πίνακες καταλήγουμε στο συμπέρασμα ότι:

- Αν θέλουμε να δουλέψουμε με μεγάλα αρχεία η Jackson είναι η πιο κατάλληλη και να ακολουθεί η GSON.
- Αν θέλουμε να δουλέψουμε με μικρά αρχεία η JSON.simple είναι η πιο κατάλληλη 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 το parsing speed 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 η JSONP Και JSON.simple λόγω έλλειψης πηγών και αλληλεπίδρασης μεταξύ μεταξύ χρηστών στο διαδίκτυο όταν προκύπτει κάποιο πρόβλημα. Καθώς δεν έχουνε και αρκετές δυνατότητες σχετικά με περίπλοκα json objects. Συγκρίνοντας τις δύο μεγαλύτερες βιβλιοθήκες, δηλαδή την Jackson και την Gson παρατηρούμε για αρχή ότι η Jackson δεν είναι εύκολη στην αρχή και έχει αρκετά μεγάλο μέγεθος με αποτέλεσμα να έχει περισσότερες δυνατότητες. Ενώ παρουσιάζουν ομοιότητες, χρησιμοποιώντας τους όρους serialization και deserialization, η Jackson καταλήξαμε ότι υπερτερεί. Αρχικά η Gson επιστρέφει τύπους δεδομένων όπως JsonObject για τα JSON objects, JsonPrimitive για τιμές όπως String, Integer, Double, JsonArray για πίνακες και JsonNull για τιμές null ενώ η Jackson τα αναλύει λίγο περισσότερο επιστρέφοντας τύπους όπως TextNode για τιμές τύπου String, ObjectNode για JSON objects, ArrayNode για πίνακες, NullNode για τιμές null, IntNode για τιμές Integer και άλλους πολλούς τύπους. Δεδομένο που θα μας χρησιμεύσει και θα μας βοηθήσει στην λειτουργικότητα της παρούσας διπλωματικής εργασίας. Η Gson όπως είπαμε έχει μία μέθοδο για to/from Json objects ενώ η Jackson έχει δύο μεθόδους που μπορεί κάποιος να μετατρέψει Json objects σε Java objects ή το ανάποδο την mapper και jsonParser ή jsonGenerator. Η Jackson υποστηρίζει και tree model που η GSON δεν το υποστηρίζει. Τέλος, παρατηρώντας τα αποτελέσματα του benchmark καταλήγουμε ότι η Jackson είναι πιο γρήγορη από την Gson και σε μικρά αλλά και σε μεγάλα αρχεία, με βάση όλων αυτών που αναφέρθηκαν αποφασίζουμε ότι θα χρησιμοποιήσουμε την Jackson.

---

## 2.3

## Ανάλυση απαιτήσεων

### **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 επεξεργασία 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 : ShowSchemaVersions**

### Περιγραφή και Στόχος:

Στόχος της περίπτωσης χρήσης είναι η εμφάνιση στο χρήστη των διαθέσιμων εκδοχών των σχημάτων που βρέθηκαν μετά την επεξεργασία του αρχείου με τα δεδομένα.

### Ηθοποιοί:

Χρήστης

### Βασική Ροή Γεγονότων:

1. Η περίπτωση χρήσης ξεκινά εμφανίζοντας στο χρήστη τις διαθέσιμες εκδόσεις.
2. Ο χρήστης επιλέγει μία από τις διαθέσιμες εκδόσεις για οπτικοποίηση.
  - 2.1. Το σύστημα ανοίγει την εκδοσή που επέλεξε, δείχνοντας στο χρήστη το σχήμα.

---

## **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

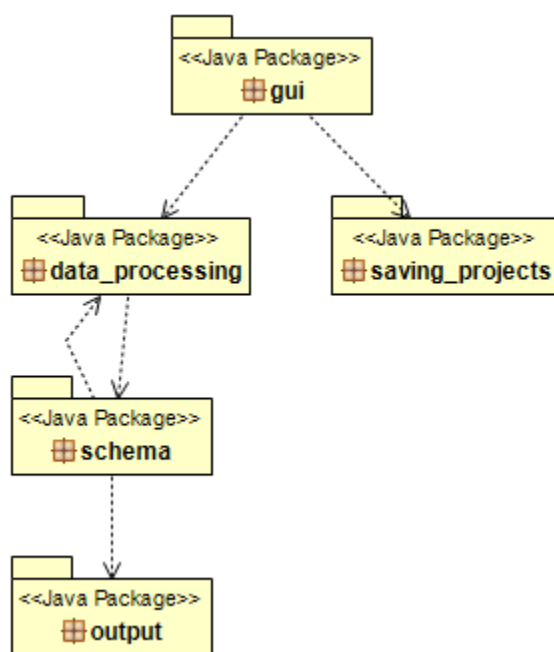
Αν πρόκειται για θέμα που προσανατολίζεται στην αλγοριθμική επίλυση ενός προβλήματος (αντί π.χ., για την κατασκευή ενός εργαλείου που παρέχει μια λειτουργικότητα) η παρούσα υποενότητα έχει τον πλέον σημαντικό ρόλο. Ενδεχομένως να χρειαστεί μία υποενότητα για κάθε ένα από τα παρακάτω.

- Τυπικός ορισμός του προβλήματος.
- Περιγραφή των χρησιμοποιούμενων δομών δεδομένων.
- Αλγόριθμος / πρωτόκολλο / μέθοδος / ... που χρησιμοποιείται για την επίλυση του προβλήματος.

---

## 3.2 Σχεδίαση και αρχιτεκτονική λογισμικού

Όπως έχει αναφερθεί ως τώρα, ο στόχος του συγκεκριμένου λογισμικού είναι η εξαγωγή σε εικονική μορφή των εκδόσεων που ακολουθούν τα αντικείμενα σ' ένα αρχείο json. 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.

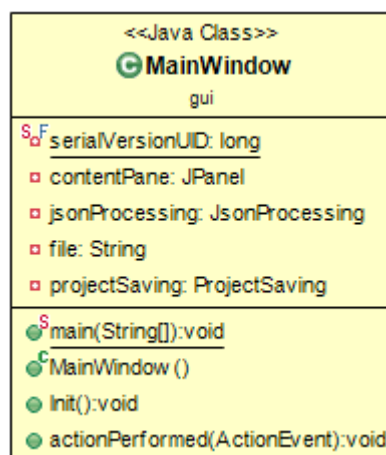


Σχήμα 3.2.1: data\_processing class diagram

### 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.

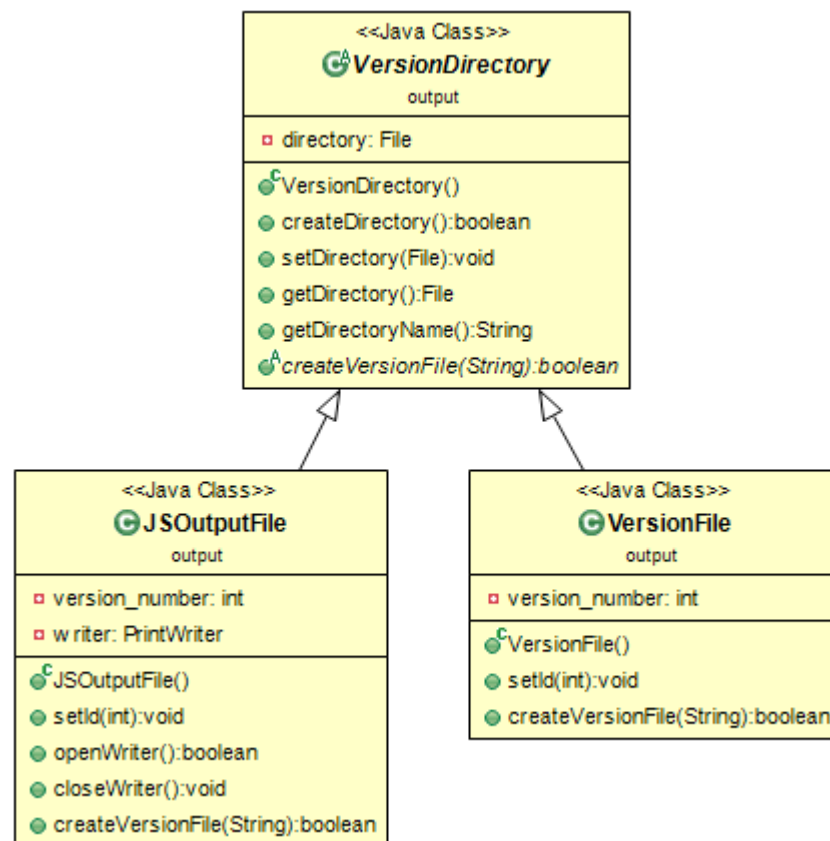
Την `ObjectNode` η οποία περιέχει τα πεδία: `objectName`, `id`, `allFields`, `objects`. Το `objectName` είναι το `key` του `JsonNode`, το `id` είναι η θέση του `JsonObject` μέσα στο αρχείο, μόνο για το `ObjectNode` που έχει `parent` ίσο με `root`, όλα τα εμφωλευμένα έχουν το ίδιο `id` με το `root` που 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`. 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`. Περιέχει και 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 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 αρχείου `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: gui class diagram**

### 3.2.2 Πακέτο gui

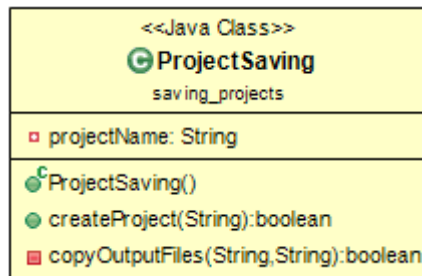
Το πακέτο gui είναι υπεύθυνο για την διεπαφή του λογισμικού με τον χρήστη και έχει την κλάση MainWindow. Δέχεται το αρχείο από τον χρήστη το επεξεργάζεται και ο χρήστης μπορεί να δει τις εκδόσεις του αρχείου, να αποθηκεύσει το project του και να ανεβάσει αν το επιθυμεί projects που ήδη υπάρχουν στο σύστημα.



Σχήμα 3.2.3: output class diagram

### 3.2.3 Πακέτο output

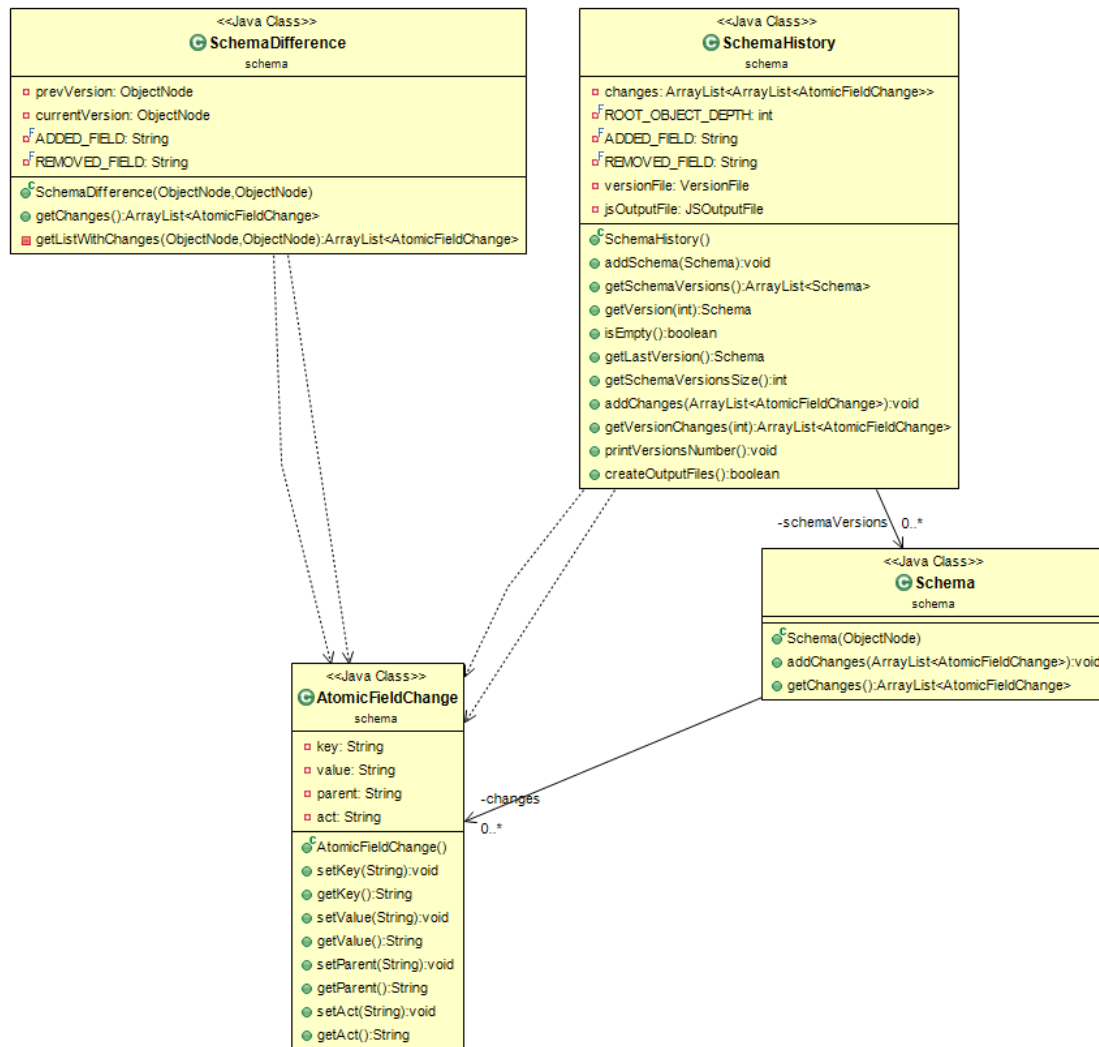
Το πακέτο output είναι υπεύθυνο για την δημιουργία του φακέλου και των αρχείων που έχουν τις εκδόσεις και περιέχει τις κλάσεις VersionDirectory, JSOutputFile και VersionFile. Η κλάση VersionDirectory δημιουργεί έναν φάκελο output ο οποίος περιέχει σε αρχεία τύπου csv τις εκδόσεις που περιέχονται στο 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 υπαρχων 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: saving\_projects 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'`. Όστε 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: schema 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, 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.



---

## 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:** Το οποίο 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 μέθοδο 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 αντικειμένων 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

### 3.4.1 Πλατφόρμες και προγραμματιστικά εργαλεία

Το συγκεκριμένο λογισμικό αναπτύχθηκε στην προγραμματική γλώσσα 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 αντικειμενοστρεφούς 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 είναι ένα αρκετά κάλο chart για οπτικοποίηση δεδομένων στο συγκεκριμένο λογισμικό 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. Το οποίο 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

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, ώστε 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. Η compareFields παίρνει σαν παραμέτρους την τελευταία μία από τις αποθηκευμένες εκδόσεις και το τρέχων json αντικείμενο και ελέγχει αν άλλαξαν τα πεδία. 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.addRow([[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.addRow(['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**

---

### 3.4.3 Χρήση του εργαλείου

## 3.5 Επεκτασιμότητα του λογισμικού

# 4 Πειραματική Αξιολόγηση

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

## 4.2 Μεθοδολογία πειραματισμού

Περιγραφή του σκοπού των πειραμάτων

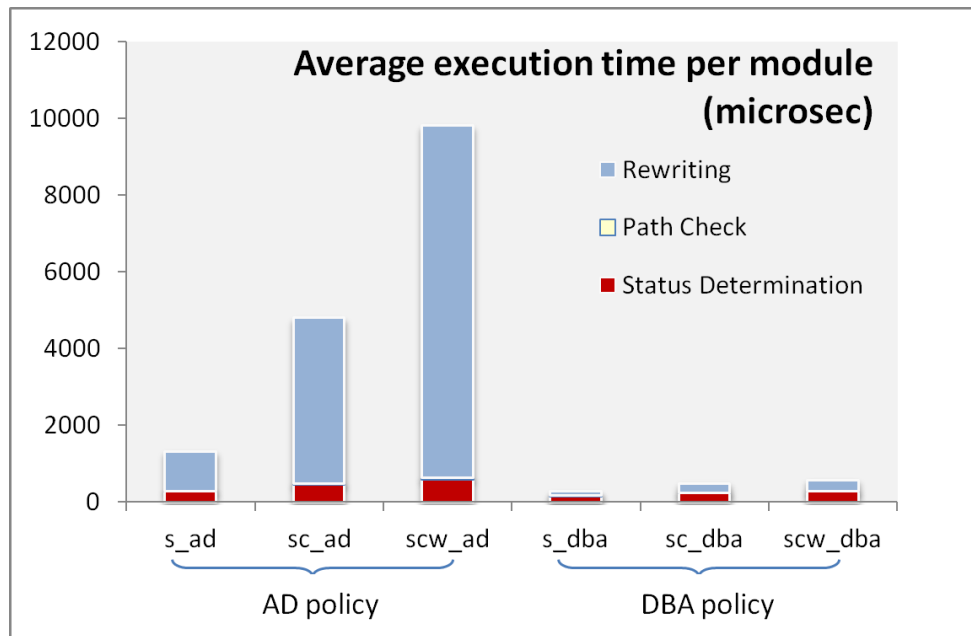
Δεδομένα που χρησιμοποιήθηκαν, αντίπαλοι αλγόριθμοι και τεχνικές (ή ότι άλλο είναι αρμόζον για την περίπτωση)

Περιγραφή του περιβάλλοντος στο οποίο διεξήχθησαν τα πειράματα.

## 4.3 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 παράγραφο τους “Keep with next” ώστε ο πίνακας να μη σπάει σε διαφορετικές σελίδες.

Breakdown of Tables over their Activity Class  
(Percentages over Total #Tables)

|           | Total<br>#Tables | Activity Class |       |        | Activity Class (%) |       |        |
|-----------|------------------|----------------|-------|--------|--------------------|-------|--------|
|           |                  | 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.

---

---

## 5 Επίλογος

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

### 5.2 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.

### 5.3 Μελλοντικές επεκτάσεις

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

---

---

# Βιβλιογραφία

- [Droe20] Michael Droettboom, Understanding JSON Schema 7.0, Space Telescope Science Institute, April 2020 .
- [Frie16] Jeff Friesen, Java XML and JSON First Edition, Jeff Friesen, 2016.
- [Peit19] Norman Peitek, Gson: Enjoy JSON (De-)Serialization in Java, Leanpub, February 2019.
- [Schi14] Herbert Schildt, Java: The Complete Rederence Ninth Edition, McGraw-Hill Education, April 2014.
- [Srip13] Sai Srinivas Sriparasa, JavaScript and JSON Essentials, Packt Publishing, October 2013.