
# Projet GL _ GSON

- **Yannick Mounguengui**
- **Nouha Boukili**

**Licence 3 Info groupe 1 GL**

# 1] Présentation globale du projet

## 1.1. Utilité du projet

### . Quel projet on a choisis?

**On a choisi d'étudier la qualité logicielle du projet Gson qui se trouve dans le lien qui suit : https://github.com/google/gson**

### . Que fait le projet?

**Gson est une libraire qu'on peut ajouter aux projets java dans le but de convertir des objets Java vers leurs représentations JSON. Ce n'est pas le premier projet open-source qui s'intéresse à la conversion des annotations Java vers Json, pourtant ce projet met en valeur deux points très importants contrairement aux autres, l'utilisation des Java Generics (with the aim of reducing bugs and adding an extra layer of abstraction over types) et aussi la possibilité de convertir vers JSON sans ajouter les annotations Java dont on a pas accès à leurs code source tout le temps.**

### . Quels sont les buts du projet?

- **fournir les méthodes toJson() et fromJson() pour convertir à JSON et vice-versa.**
- **permettre les objets Java qui définis précédemment comme étant non modifiables à se convertir vers JSON.**
- **utiliser les java Generics.**
- **permettre les représentations clients aux objets**
- **prise en charge d'objets arbitrairement complexes (avec des hiérarchies d'héritage profondes et une utilisation intensive de types génériques)**

### . Comment lancer le projet?

**Comme le projet est sous forme d'une bibliothèque Java, on peut l'installer en l'ajoutant aux dépendances en deux manières :**

**façon 1 : par les dépendances du pom.xml pour un projet Maven comme suit :**

**Maven:**

```
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.10.1</version>
</dependency>
```

**façon 2 : par les dépendances Gradle :**

```
dependencies {
  implementation 'com.google.code.gson:gson:2.10.1'
}
```

## . Quel est  le résultat du projet : quelles sont les sorties du logiciel ?

**Le but principal du projet c'est pouvoir utiliser les méthodes qui convertissent au Json.**
**La classe principale à utiliser est Gson qu'on peut simplement créer en appelant new Gson(). Il existe également une classe GsonBuilder disponible qui peut être utilisée pour créer une instance Gson avec divers paramètres tels que le contrôle de version, etc.**

**L'instance Gson ne conserve aucun état lors de l'appel d'opérations JSON. Ainsi, on est libre de réutiliser le même objet pour plusieurs opérations de sérialisation et de désérialisation JSON.**

**On peut donner comme exemple d'utilisation du projet ce qui suit:**

```
// Serialization
Gson gson = new Gson();
gson.toJson(1);             // ==> 1
gson.toJson("abcd");        // ==> "abcd"
gson.toJson(new Long(10)); // ==> 10
int[] values = { 1 };
gson.toJson(values);        // ==> [1]

// Deserialization
int i = gson.fromJson("1", int.class);
Integer intObj = gson.fromJson("1", Integer.class);
Long longObj = gson.fromJson("1", Long.class);
Boolean boolObj = gson.fromJson("false", Boolean.class);
String str = gson.fromJson("\"abc\"", String.class);
String[] strArray = gson.fromJson("[\"abc\"]", String[].class);
```

## 1.2. Description du projet

### . Le readme est-il pertinent?

**Il y a bien un README dans ce projet, qui de plus est à jour (dernière modification effectuée le 6 janvier 2023 pour annoncer la sortie du gson-parent-2.10.1 ). Ce dernier présente globalement le projet sans entrer dans les détails. On peut ainsi retrouver une brève introduction pour se mettre dans le contexte, les objectifs du projet ou encore des liens utiles vers des documentations.**

### . La documentation du projet est-elle présente?

**En plus de l'extrait de documentation retrouvée dans le README, il y a à disposition une documentation supplémentaire détaillée et complète du projet dans le fichier UserGuide.md. On y retrouve également des informations sur**

- **l'installation**
- **l'utilisation : nous constatons qu'il y a suffisamment d'exemples permettant d'illustrer clairement comment utiliser Gson sans avoir besoin de consulter le code source**
- **ou encore l'utilité du projet.**

### . Les informations en terme de lancement sont-elles suffisantes ?

**Toutefois, les informations en termes d'installation et de lancement ne sont pas très explicites, on peut avoir des bugs lors de la première installation.**

# 2] Historique du projet

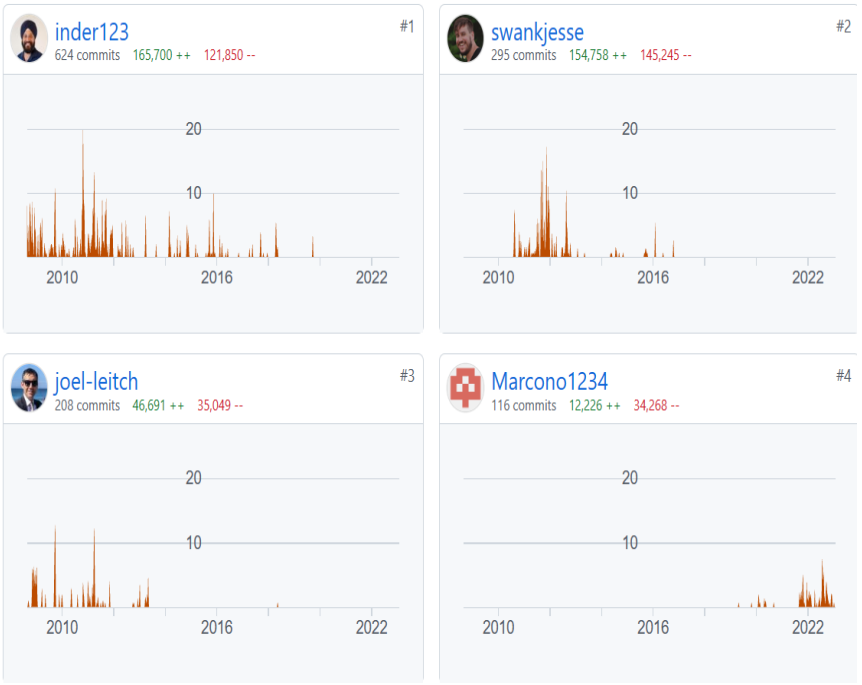
## 2.1. Analyse du git

### . Quel est le nombre de contributeurs?

**D'après le dépôt Github, 137 personnes contribuent pour l'instant au développement du projet.**

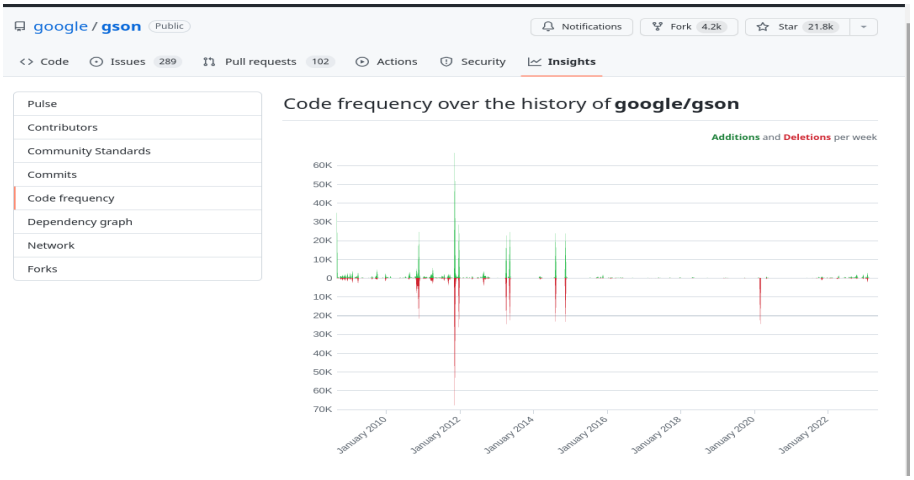
### . Est ce qu'il ont contribué de façon équilibré sur le projet? et dans le temps?

**Non, on peut constater clairement que les contributions ne sont pas équilibrées, on trouve certains qui ont fait 624 commits et d'autres qui ont fait qu'un seul.**
**Aussi, on trouve une majorité de personnes à travailler en 2010, contrairement à d'autres qui ont commencé en 2022.**
**On donne un exemple des contributions d'utilisateurs dans l'image qui suit:**




## . Le projet est-il toujours actif? l'activité est-elle répartie régulièrement sur le temps?

**Le projet est toujours actif, on trouve des modifications faites le jour même ou on vérifie, mais l'activité n'est pas régulière, on trouve que les ajouts et les suppressions en 2012 sont beaucoup plus nombreux par rapport à ces dernières années.**



### Combien de branches sont définies? et combien parmi elles sont utilisées?

**Le projet est constitué de 15 branches dont 6 sont closed et le reste est open.**

### . Le mécanisme de pull request est-il utilisé?

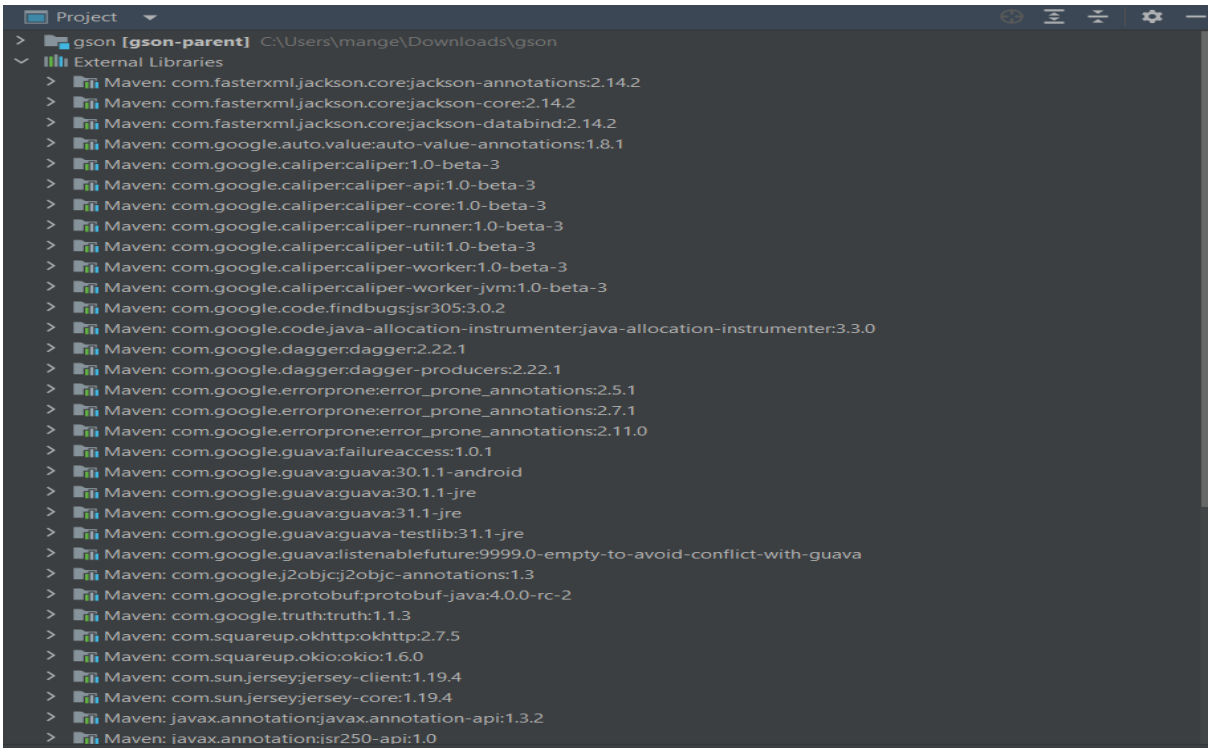
**Il y a 102 demandes de pull qui sont en attente et 627 demandes qui sont closed**

# 3] Architecture Logicielle

## 3.1. Utilisations des bibliothèques extérieurs

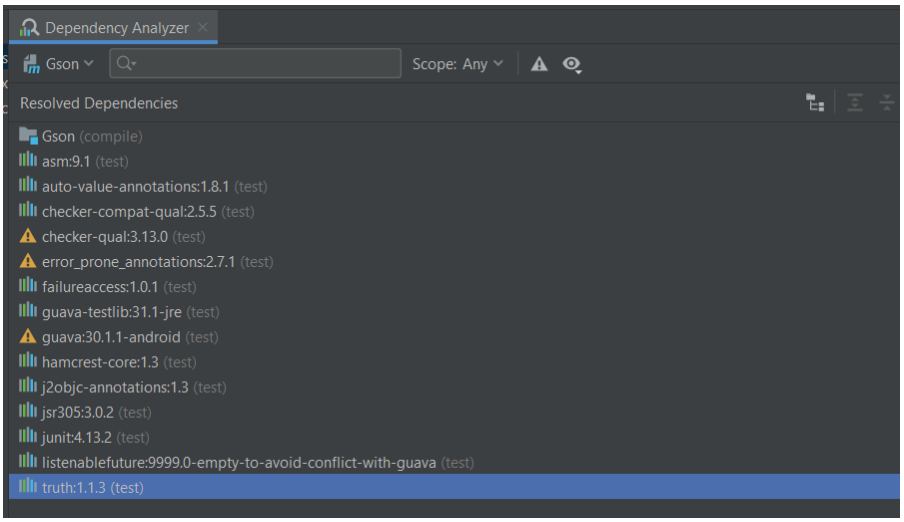
### . Quel est le nombre des bibliothèques extérieures?

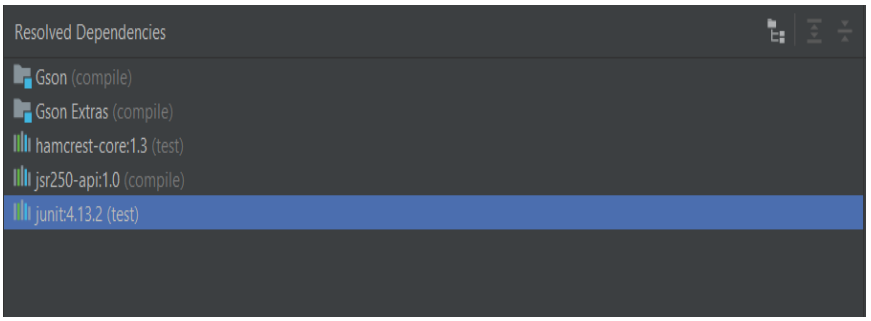
**A l'aide de intellij idea, nous avons trouvé qu'il y a 49 bibliothèques extérieures maven.**

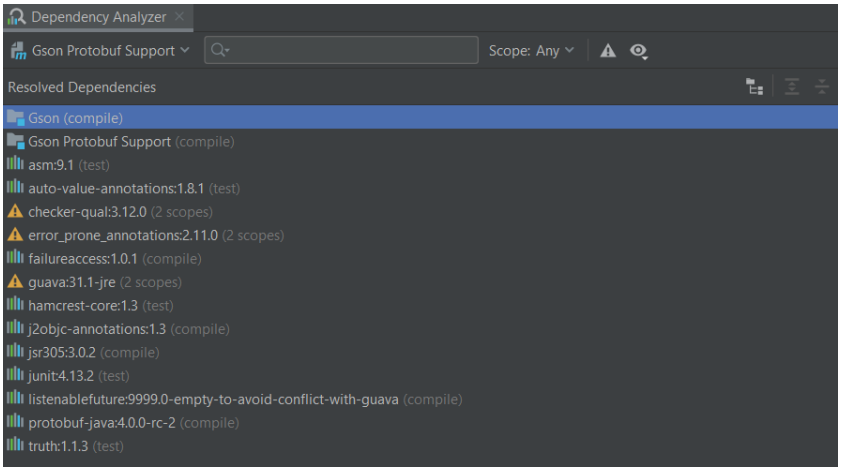


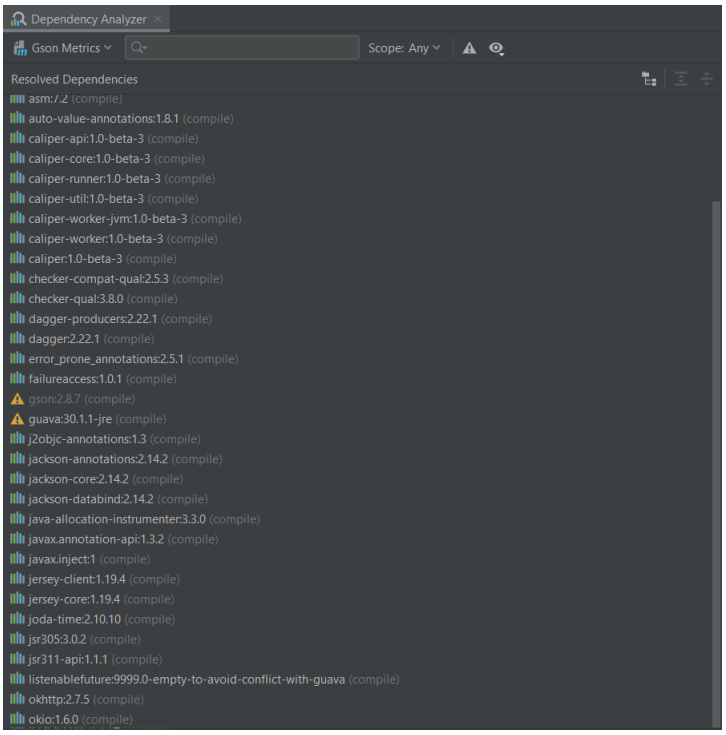
## . analyser la différence entre les bibliothèques référencées et celles utilisées : certaines dépendances sont-elles donc inutiles?

**Toutes les bibliothèques sont utilisées dans les différents dossiers du projet ainsi aucune n'est utilisée au bon fonctionnement des classes.**





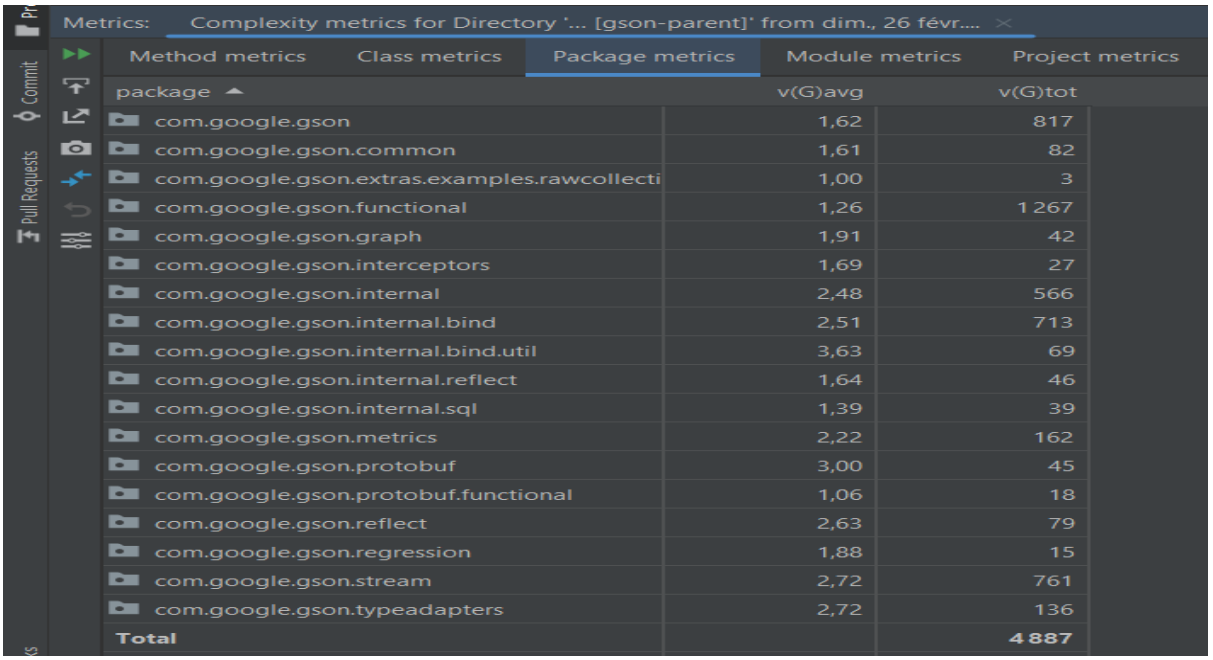




**. analyser les bibliothèques réellement utilisées : que nous apprend l'utilisation de ces bibliothèques ? A-t-on, par exemple, plusieurs bibliothèques qui servent à la même chose (par exemple plusieurs ORM, ou bibliothèques graphiques) ? Est-ce justifié ?**

## 3.2.Organisation en paquetages

**. Quel est le nombre de paquetages?**

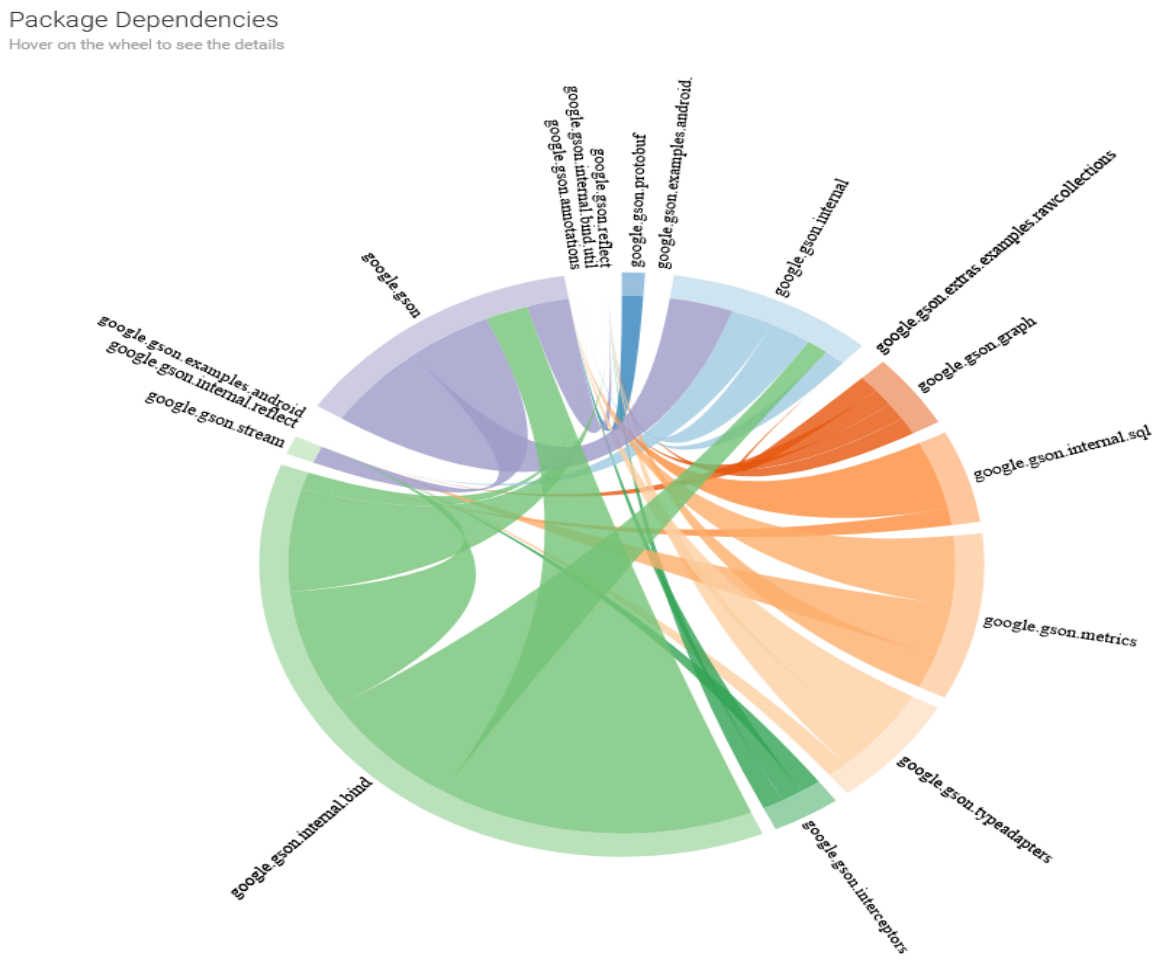
**L'analyse a montré que le nombre de paquetages de ce projet vaut 18 comme nous pouvons le voir sur l'image ci-dessous:**




| package | v(G)avg | v(G)tot |
|---|---|---|
| com.google.gson | 1,62 | 817 |
| com.google.gson.common | 1,61 | 82 |
| com.google.gson.extras.examples.rawcollecti | 1,00 | 3 |
| com.google.gson.functional | 1,26 | 1 267 |
| com.google.gson.graph | 1,91 | 42 |
| com.google.gson.interceptors | 1,69 | 27 |
| com.google.gson.internal | 2,48 | 566 |
| com.google.gson.internal.bind | 2,51 | 713 |
| com.google.gson.internal.bind.util | 3,63 | 69 |
| com.google.gson.internal.reflect | 1,64 | 46 |
| com.google.gson.internal.sql | 1,39 | 39 |
| com.google.gson.metrics | 2,22 | 162 |
| com.google.gson.protobuf | 3,00 | 45 |
| com.google.gson.protobuf.functional | 1,06 | 18 |
| com.google.gson.reflect | 2,63 | 79 |
| com.google.gson.regression | 1,88 | 15 |
| com.google.gson.stream | 2,72 | 761 |
| com.google.gson.typeadapters | 2,72 | 136 |
| **Total** | | **4 887** |

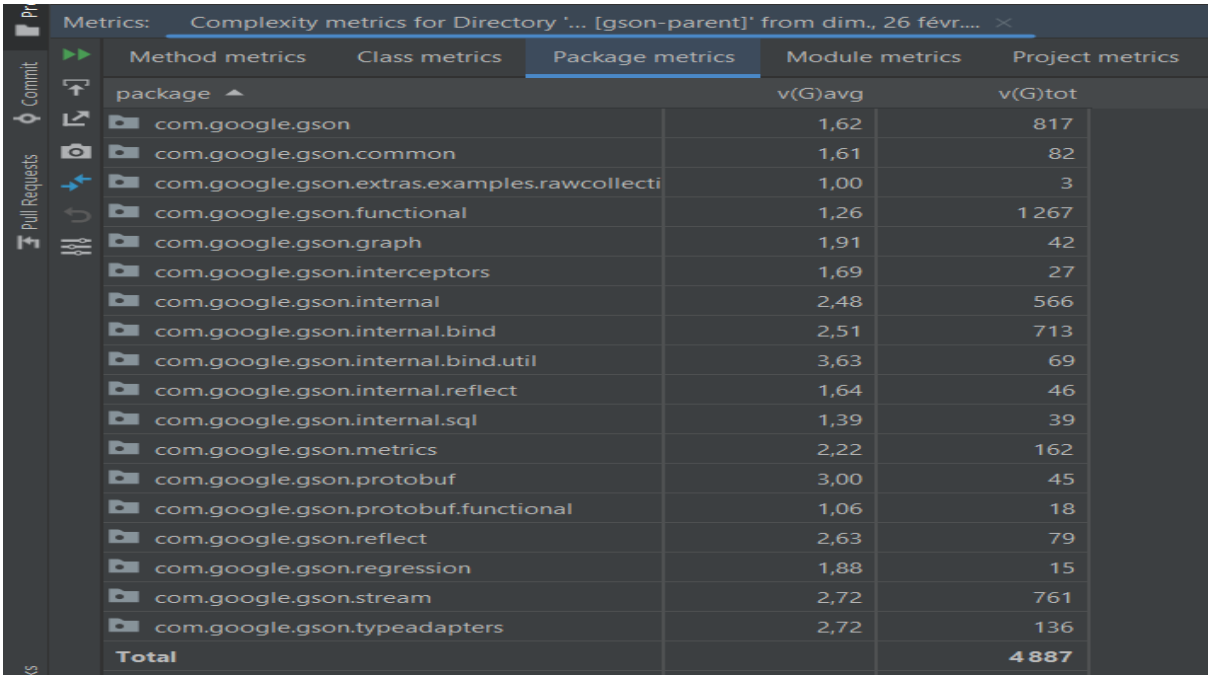
**. analyser les liens entre les paquetages : quels paquetages dépendent de quels autres ? Les paquetages sont-ils organisés en couche ?Existe-t-il des cycles entre paquetages ?**

**Il y a plusieurs paquetages qui dépendent d'autres comme le paquetage " gson.internal.bind " qui dépend beaucoup du paquetage gson,lui de même dépend beaucoup du paquetage gson.internal .Les paquetages sont organisés en plusieurs couches.Le paquetage *gson.internal.sql* est en cycle avec le paquetage *google.gson* .**




**. analyser la hiérarchie de paquetage : quel est le nombre de niveaux de paquetages ? La hiérarchie de paquetages pour les tests suit-elle la hiérarchie de paquetages des sources ? Ou ailleurs a-t-on des hiérarchies parallèles (ou presque) ? Existe-t-il des paquetages qui ne contiennent qu'un seul paquetage sans aucune classe ?**

**Le plus petit niveau de paquetage est de 2 et le plus grand est 5.Ensuite ,la hiérarchie de paquetages des classes sources et tests, nous a permis de voir que la plupart des tests suivent la hiérarchie des sources .Chaque paquetage contient au moins une classe.**




| package | v(G)avg | v(G)tot |
|---|---|---|
| com.google.gson | 1,62 | 817 |
| com.google.gson.common | 1,61 | 82 |
| com.google.gson.extras.examples.rawcollecti | 1,00 | 3 |
| com.google.gson.functional | 1,26 | 1 267 |
| com.google.gson.graph | 1,91 | 42 |
| com.google.gson.interceptors | 1,69 | 27 |
| com.google.gson.internal | 2,48 | 566 |
| com.google.gson.internal.bind | 2,51 | 713 |
| com.google.gson.internal.bind.util | 3,63 | 69 |
| com.google.gson.internal.reflect | 1,64 | 46 |
| com.google.gson.internal.sql | 1,39 | 39 |
| com.google.gson.metrics | 2,22 | 162 |
| com.google.gson.protobuf | 3,00 | 45 |
| com.google.gson.protobuf.functional | 1,06 | 18 |
| com.google.gson.reflect | 2,63 | 79 |
| com.google.gson.regression | 1,88 | 15 |
| com.google.gson.stream | 2,72 | 761 |
| com.google.gson.typeadapters | 2,72 | 136 |
| **Total** | | **4 887** |

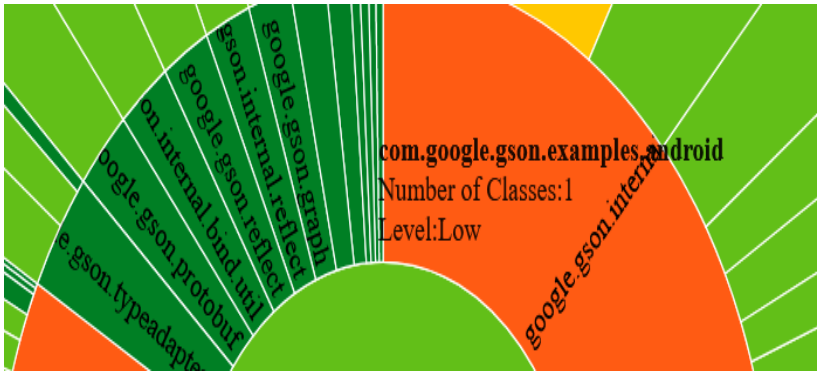
**. analyser les noms des paquetages : les noms des paquetages nous apprennent-ils des choses sur les designs patterns utilisés (MVC par exemple) ? Sur l'existence d'un lien avec une base de données ? Que nous apprennent les noms des paquetages de façon plus générale ?**

**Nous constatons que le paquetage commun à tout le projet est *com.google.gson* qui est le nom de domaine du projet. En revanche, les autres noms de paquetages sont fortement liés au domaine métier du projet. Ainsi, il est difficile d'avoir une idée précise de ce qu'ils peuvent contenir comme on le ferait par exemple pour *Model-View-Controller.* Les noms des paquetages peuvent nous fournir des informations utiles sur leur contenu,leur fonctionnalité,leurs dépendances et leur version.**

## 3.3. Répartition des classes dans les paquetages

**. Quel est le nombre minimum, maximum par paquetage et la moyenne?Également, on pourra fournir le nombre total de classes**

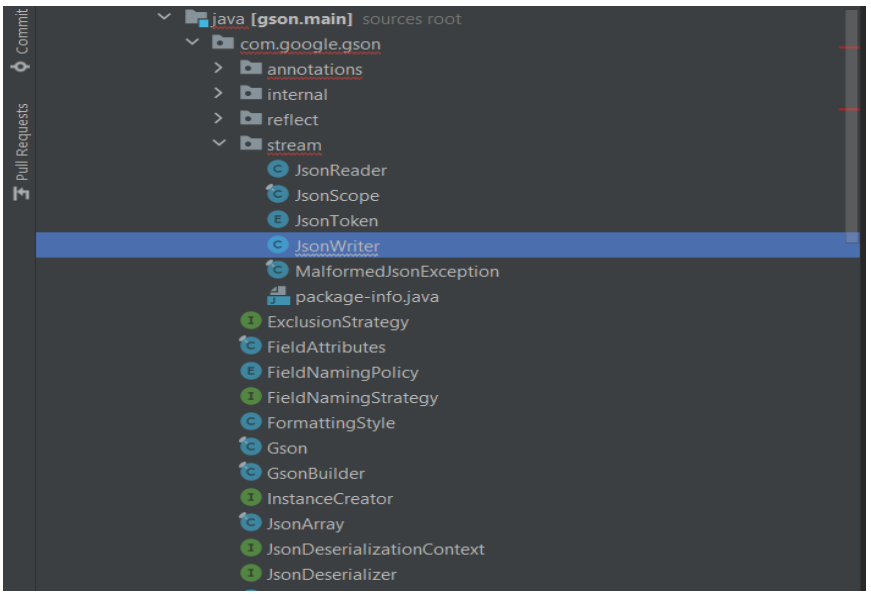
**Le minimum de classe par paquetage est de 1 et le maximum est de 25.Ainsi nous avons une moyenne de 13 classes par paquetage et un nombre total de 639 classes.**



**. analyser la répartition des classes dans les différents paquetages : toutes les classes ou presque sont-elles dans le même paquetage ?Est-ce que des paquetages non feuilles contiennent des classes ? S'il y a plusieurs hiérarchies parallèles, les paquetages qui ont le plus de classes dans une hiérarchie ont-ils aussi le plus de classes dans les autres ?**

**Presque toutes les classes sont majoritairement dans un paquetage(com.google.gson) et nous avons constaté que les paquetages non feuilles contiennent également des classes.**



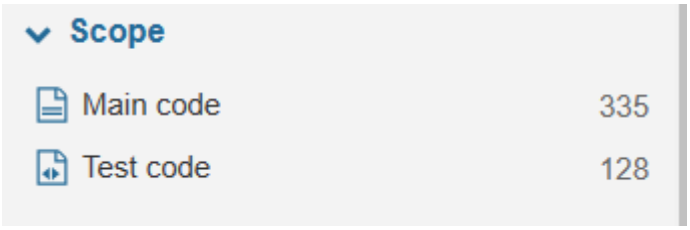

**. analyser le couplage et la cohésion au sein de quelques paquetages en particulier**

## 4] Analyse Approfondi

### 4.1. Tests

**. Combien de classes de tests existent? combien de méthodes de tests? Combien d'assertions? Que peut-on en déduire sur la structuration des tests?**

**Il existe 128 classes de tests.**




**. étudier la couverture de tests. Quel est le pourcentage de code couvert par les tests ?**

**Le pourcentage de couverture de code est de 89,9%.Il y a plusieurs parties qui ne sont pas couvertes dans le code.**

| Element | Coverage | vered Instructions | lissed Instructions | Total Instructions |
|---|---|---|---|---|
| gson | 89,9 % | 67 795 | 7 616 | 75 411 |
| src/test/java | 90,7 % | 49 120 | 5 050 | 54 170 |
| src/main/java | 87,9 % | 18 675 | 2 566 | 21 241 |

## 4.2. Commentaires

.compter le nombre de lignes de commentaire

**A l'aide de Sonarqube, on trouve qu'il y a 4372 commentaires avec un pourcentage de 28.1% de code commenté.**

| Size | |
|---|---|
| Lines of Code | 11,171 |
| Lines | 20,521 |
| Statements | 4,803 |
| Functions | 832 |
| Classes | 139 |
| Files | 101 |
| Comment Lines | 4,372 |
| Comments (%) | 28.1% |

# 5] Nettoyage de Code et Code smells.identifier des nombres magiques dans le code

## 5.2. Nombre magique

.identifier des nombres magiques dans le code

JsonReader.java

```
5 usages
209  private static final int PEEKED_DOUBLE_QUOTED_NAM
5 usages
210  private static final int PEEKED_UNQUOTED_NAME = 1
211  /** When this is returned, the integer value is s
6 usages
212  private static final int PEEKED_LONG = 15;
7 usages
213  private static final int PEEKED_NUMBER = 16;
4 usages
214  private static final int PEEKED_EOF = 17;
215
216  /* State machine when parsing numbers */
3 usages
217  private static final int NUMBER_CHAR_NONE = 0;
2 usages
218  private static final int NUMBER_CHAR_SIGN = 1;
6 usages
219  private static final int NUMBER_CHAR_DIGIT = 2;
2 usages
220  private static final int NUMBER_CHAR_DECIMAL = 3;
```

**Dans la classe JsonReader par exemple, nous avons trouvé qu'il y a des nombres magiques.On en compte 28 au total.**

.essayer de comprendre pourquoi telle valeur ou pas telle autre, en essayant de comprendre l'intention

**Il est difficile de dire à quoi cette constante peut être destinée, mais étant donné que son nom inclut le mot "DECIMAL" et que sa valeur est un entier, il est possible qu'elle soit utilisée pour définir le nombre de chiffres décimaux à utiliser lors de l'affichage ou de la manipulation de nombres à virgule flottante dans le code.**

.proposer de remplacer l'utilisation de ce nombre magique (dans le code et éventuellement dans les tests) par une constante

A FAIRE

## 5.3. Structure de code

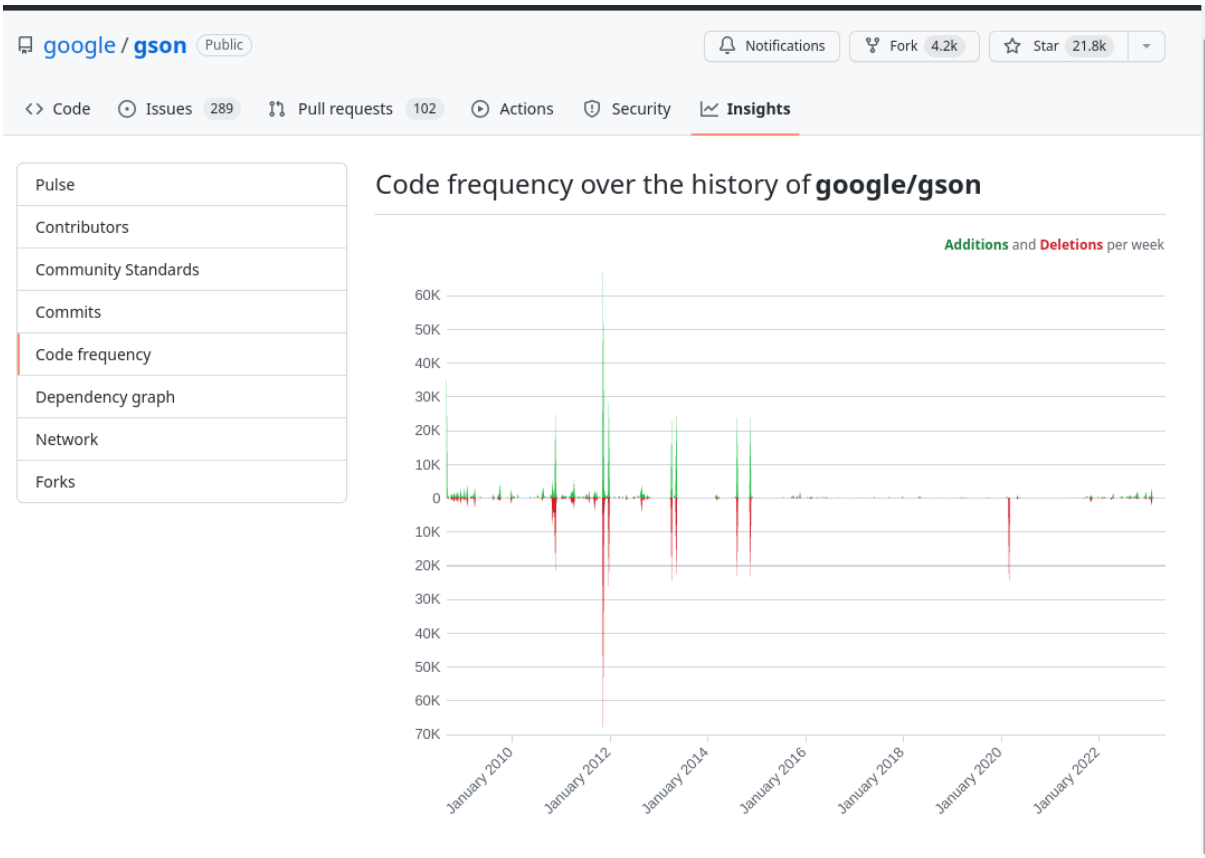
.vérifier qu'au sein des classes,  est-ce que toutes les variables d'instance sont ensemble?

Non, les variables d'instances ne sont pas toujours ensemble, on trouve la plupart du temps des variables après la déclaration de la classe, mais on trouve aussi des variables déclarés juste avant d'etre utilisés

```
47  import java.util.Locale;
48  import java.util.zip.ZipEntry;
49  import java.util.zip.ZipFile;
50
51  /**
52   * Measure Gson and Jackson parsing and binding performance.
53   *
54   * <p>This benchmark requires that ParseBenchmarkData.zip is on the classpath.
55   * That file contains Twitter feed data, which is representative of what
56   * applications will be parsing.
57   */
58  public final class ParseBenchmark {
59    @Param Document document;
60    @Param Api api;
61
62    private enum Document {
63      TWEETS(new TypeToken<List<Tweet>>() {}, new TypeReference<List<Tweet>>() {}),
```

```
174
175     public ProtoTypeAdapter build() {
176       return new ProtoTypeAdapter(enumSerialization, protoFormat, jsonFormat,
177           serializedNameExtensions, serializedEnumValueExtensions);
178     }
179   }
180
181   /**
182    * Creates a new {@link ProtoTypeAdapter} builder, defaulting enum serialization to
183    * {@link EnumSerialization#NAME} and converting field serialization from
184    * {@link CaseFormat#LOWER_UNDERSCORE} to {@link CaseFormat#LOWER_CAMEL}.
185    */
186   public static Builder newBuilder() {
187     return new Builder(EnumSerialization.NAME, CaseFormat.LOWER_UNDERSCORE, CaseFormat.LOWER_CAMEL);
188   }
189
190   private static final com.google.protobuf.Descriptors.FieldDescriptor.Type ENUM_TYPE =
191       com.google.protobuf.Descriptors.FieldDescriptor.Type.ENUM;
192
193   private static final ConcurrentMap<String, ConcurrentMap<Class<?>, Method>> mapOfMapOfMethods =
194       new MapMaker().makeMap();
195
```

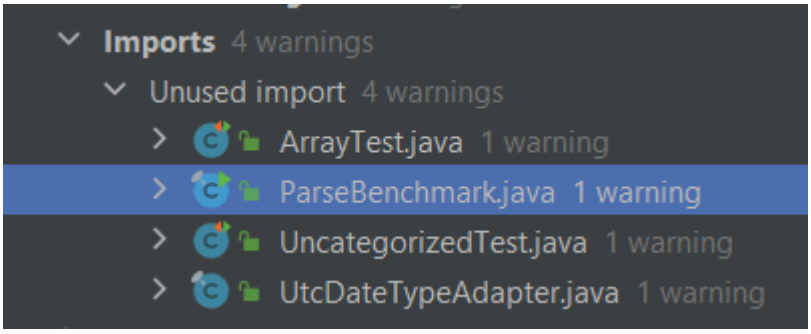
```
21    * @author Inderjeet Singh
22    */
23   public class BagOfPrimitives {
24     public static final long DEFAULT_VALUE = 0;
25     public long longValue;
26     public int intValue;
27     public boolean booleanValue;
28     public String stringValue;
29
30     public BagOfPrimitives() {
31       this(DEFAULT_VALUE, 0, false, "");
32     }
33
```




## 5.4. Code mort

### .Existe-t-il du code mort au sein du projet (rappel : code mort code pas appelé)?

**Oui il existe du code mort,  ce n'est pas du code d'une API mais c'est du code de deux classes sources java.**




## Ce code est-il testé ? maintenu ?

**Pour le cas de ParseBenchmark.java, il n'est pas testé et avec un taux technique de ration de 1.3% nous pouvons dire qu'il n'est pas maintenu.**

Technical Debt Ratio **3.1%**

| | |
|---|---|
| BagOfPrimitives.java | 2.3% |
| BagOfPrimitivesDeserializationBenchmark.java | 8.1% |
| CollectionsDeserializationBenchmark.java | 6.7% |
| NonUploadingCaliperRunner.java | 1.0% |
| ParseBenchmark.java | 1.3% |
| SerializationBenchmark.java | 3.5% |

## Ce code devrait-il être supprimé ?

**Avec un taux technique de 1.3% on peut envisager de supprimer ce code vu qu'il n'a pas une importance majeure pour le projet.**