Découverte des clients REST: implémenter un client REST en Java

Rémi SHARROCK, Télécom ParisTech

# ​1​ Eléments introductifs

Rappelez-vous que “REST (Representational State Transfer) est un style d'architecture définissant un ensemble de contraintes et de propriétés basées sur le protocole HTTP. Les services web conformes au style d'architecture REST, aussi appelés services Web RESTful, établissent une interopérabilité entre les ordinateurs sur Internet. Les services web REST permettent aux systèmes effectuant des requêtes d'accéder et de manipuler des représentations textuelles de ressources web à travers un jeu d'opérations uniformes et prédéfinies sans état.” <https://fr.wikipedia.org/wiki/Representational_state_transfer>

Pour pouvoir utiliser REST Il faut d’abord parler à un serveur HTTP avec un Client HTTP.

En Java, il existe plusieurs solutions:

* La bonne vieille connection HTTP bas niveau de Java, inclus dans Java de base depuis Java 1.1: <https://docs.oracle.com/javase/8/docs/api/java/net/HttpURLConnection.html>
* Le client HTTP officiel de Java est encore en cours d’incubation (pour Java version 9). Attention de ne pas utiliser ce client en état d’incubation! Il faudra attendre peut-être la prochaine version de Java? <https://docs.oracle.com/javase/9/docs/api/jdk/incubator/http/HttpClient.html>   
  Note du site: “Incubating Feature. Will be removed in a future release.”
* Le client HTTP Apache, attention difficilement compatible avec Android!  
  <https://hc.apache.org/httpcomponents-client-ga/>   
  Argumentaire du site : “Although the java.net package provides basic functionality for accessing resources via HTTP, it doesn't provide the full flexibility or functionality needed by many applications. HttpClient seeks to fill this void by providing an efficient, up-to-date, and feature-rich package implementing the client side of the most recent HTTP standards and recommendations.”
* Le client HTTP OkHttp, compatible Android  
  <http://square.github.io/okhttp/>   
  “OkHttp perseveres when the network is troublesome: it will silently recover from common connection problems. If your service has multiple IP addresses OkHttp will attempt alternate addresses if the first connect fails. This is necessary for IPv4+IPv6 and for services hosted in redundant data centers. OkHttp initiates new connections with modern TLS features (SNI, ALPN), and falls back to TLS 1.0 if the handshake fails.”
* Le client HTTP de Google, compatible Android mais qui utilise un transporteur HTTP tiers, tel que celui d’apache…  
  <https://developers.google.com/api-client-library/java/google-http-java-client/>   
  “The HTTP library supports Java 5 (or higher) standard (SE) and enterprise (EE), Android 1.5 (or higher), and Google App Engine.”
* D’autres listes:   
  <https://github.com/akullpp/awesome-java#http-clients> <https://dzone.com/articles/high-concurrency-http-clients-on-the-jvm>

## ​1.1​ Tester des clients HTTP

### ​1.1.1​ Introduction

Comment tester un client HTTP ? Avec un serveur HTTP.

Deux solutions simples: un serveur HTTP de test distant ou un serveur HTTP de test local.

Serveur HTTP de test distant: <https://httpbin.org/>

Serveur HTTP de test local (à installer en local): <http://www.mock-server.com/>

Nous allons tester des clients HTTP en java en utilisant le serveur HTTP de test distant <https://httpbin.org/>

**Créez un fichier Java différent pour chaque question: vous allez en créer beaucoup.**

### ​1.1.2​ HttpURLConnection

#### ​1.1.2.1​ Une connection simple

Utilisez le tutoriel

<https://alvinalexander.com/blog/post/java/simple-https-example>

En changeant le serveur par <https://httpbin.org/ip> pour obtenir votre adresse IP publique au format JSON.  
Rappelez-vous le format JSON : <https://fr.wikipedia.org/wiki/JavaScript_Object_Notation>

#### ​1.1.2.2​ Un GET avec une clé valeur

Utilisez le tutoriel <https://alvinalexander.com/blog/post/java/how-open-url-read-contents-httpurl-connection-java>

Faites une requête avec la méthode GET en envoyant la clé “message” et la valeur “bonjour”. L’URL doit être de la forme

|  |
| --- |
| <https://httpbin.org/get?message=bonjour> |

Vérifiez que vous obtenez dans la réponse JSON

|  |
| --- |
| "args":{"message":"bonjour"} |

#### ​1.1.2.3​ Un GET avec plusieurs “clé-valeur” et l’encodage d’URL

Faites une requête avec la méthode GET en envoyant les clé-valeur suivants:

|  |  |
| --- | --- |
| message1 | bonjour |
| message2 | au revoir |
| mess age 3 | special characters : " ' / \ % ~ ! @ # $ % ^ & \* ( ) etc... |
| m e s s a g e 4 | C’est l’été à Paris! |

Attention de bien encoder le caractère espace dans la valeur du message 2, à encoder la clé “mess age 3” avec les espaces, à encoder tous les caractères spéciaux de la valeur du “mess age 3” ainsi qu’à encoder la clé “m e s s a g e 4” et les caractères accentués correspondants, et **DE NE PAS ENCODER le & entre les paramètres et le reste de l’URL**.

Aidez-vous du tutoriel <https://alvinalexander.com/blog/post/java/how-encode-java-string-send-web-server-safe-url> et <https://www.url-encode-decode.com/>

L’URL résultante devrait être

|  |
| --- |
| [https://httpbin.org/get?message1=bonjour&message2=au+revoir&mess+age+3=special+characters+%3A+%22+%27+%2F+%5C+%25+%7E+%21+%40+%23+%24+%25+%5E+%26+\*+%28+%29+etc...&m+e+s+s+a+g+e+4=C%27est+l%27%C3%A9t%C3%A9+%C3%A0+Paris%21](https://httpbin.org/get?message1=bonjour&message2=au+revoir&mess+age+3=special+characters+%3A+%22+%27+%2F+%5C+%25+%7E+%21+%40+%23+%24+%25+%5E+%26+*+%28+%29+etc...&m+e+s+s+a+g+e+4=C%27est+l%27%C3%A9t%C3%A9+%C3%A0+Paris%21) |

Vérifiez que vous obtenez dans la réponse JSON (indépendamment de l’ordre des paramètres, mais vérifiez bien caractère par caractère!)

|  |
| --- |
| "args":{"m e s s a g e 4":"C'est l'\u00e9t\u00e9 \u00e0 Paris!","mess age 3":"special characters: \" ' / \\ % ~ ! @ # $ % ^ & \* ( ) etc...","message1":"bonjour","message2":"au revoir"} |

Remarquez qu’il faut ajouter le caractère d’échappement \ pour \” et \\ pour échapper les caractères “ et \ en Java (dans une String).

Solution à demander à Rémi SHARROCK: fichier JDKGet.java

#### ​1.1.2.4​ Un POST avec clé valeur

Maintenant, en utilisant la méthode POST envoyez les même “clé-valeurs” que la question précédente, également encodées. Aidez-vous du tutoriel <https://www.mkyong.com/java/how-to-send-http-request-getpost-in-java/>

Remarquez que la méthode connect() est optionnelle (même dans les cas précédents cf <https://stackoverflow.com/questions/2792705/httpurlconnection-is-it-necessary-to-call-connect> )

Vérifiez que vous obtenez dans la réponse JSON

|  |
| --- |
| "args":{},"data":"","files":{} |

Vérifiez également que vous obtenez dans la réponse JSON (indépendamment de l’ordre des paramètres):

|  |
| --- |
| "form":{"m e s s a g e 4":"C'est l'\u00e9t\u00e9 \u00e0 Paris!","mess age 3":"special characters: \" ' / \\ % ~ ! @ # $ % ^ & \* ( ) etc...","message1":"bonjour","message2":"au revoir"} |

Enfin, remarquez que le type de contenu dans la réponse JSON est de type

|  |
| --- |
| "Content-Type":"application/x-www-form-urlencoded" |

Solution à demander à Rémi SHARROCK: fichier JDKPost.java

#### ​1.1.2.5​ Un POST en multipart/form-data au lieu de application/x-www-form-urlencoded

Lisez la différence ici <https://stackoverflow.com/questions/4007969/application-x-www-form-urlencoded-or-multipart-form-data/4073451#4073451>

Au lieu d’encoder des données en “URLEncoding” pour les envoyer en POST, nous allons envoyer plusieurs types de données sans les encoder, en multipart/form-data.

Les données que nous voulons envoyer sont:

* une “clé-valeur” simple: “param” valeur “value”
* un fichier texte encodé avec un “charset” UTF-8 contenant   
  C'est un fichier encodé en UTF-8!
* un fichier binaire contenant la valeur de l’entier 99.
* Pour créer les fichiers, vous pouvez utiliser le code suivant:

|  |
| --- |
| //création du fichier texte  Writer out = new BufferedWriter(new OutputStreamWriter(  new FileOutputStream("file.txt"), "UTF-8"));  try {  out.write("C'est un fichier encodé en UTF-8!");  } finally {  out.close();  }  //fin création du fichier texte  //création du fichier binaire  int i = 99;  DataOutputStream os = new DataOutputStream(new FileOutputStream("file.bin"));  os.writeInt(i);  os.close();  //fin création du fichier binaire |

Utilisez la partie “Upload files” de ce tutoriel: <https://stackoverflow.com/a/2793153>

Vous devriez obtenir dans la réponse JSON du serveur la partie suivante:

|  |
| --- |
| "args":{},"data":"","files":{"binaryFile":"\u0000\u0000\u0000c","textFile":"C'est un fichier encod\u00e9 en UTF-8!"},"form":{"param":"value"} |

Vérifiez bien que ce sont les entrées files et form qui sont remplies. L’entrée data est vide.

Remarquez que le binaryFile est affiché en UTF-8 octet par octet, l’entier 99 étant codé sur 4 octets, les trois premiers octets sont nuls et le dernier correspond à la lettre c :

|  |
| --- |
| \u0000\u0000\u0000c |

​Solution à demander à Rémi SHARROCK: fichier JDKPostFiles.java

#### ​1.1.2.6​ Envoi par un PUT

Utilisez la méthode PUT pour envoyer la valeur de l’entier 42

L’URL est <https://httpbin.org/put>

La réponse doit contenir:

|  |
| --- |
| "args":{},"data":"\u0000\u0000\u0000c","files":{},"form":{} |

Vérifiez bien que c’est l’entrée data qui contient les données!

​Solution à demander à Rémi SHARROCK: fichier JDKPut.java

### ​1.1.3​ Client Apache HttpClient

Installez de la manière que vous préférez les bibliothèques pour faire fonctionner le client Apache HttpClient (Maven est recommandé).

#### ​1.1.3.1​ Une connection simple

Utilisez l’exemple simple <https://hc.apache.org/httpcomponents-client-ga/httpclient/examples/org/apache/http/examples/client/ClientWithResponseHandler.java>

En changeant le serveur par <https://httpbin.org/ip> pour obtenir votre adresse IP publique au format JSON.

La sortie devrait être

|  |
| --- |
| Executing request GET http://httpbin.org/ip HTTP/1.1  ----------------------------------------  {"origin":"176.159.23.2"} |

#### ​1.1.3.2​ Un GET avec une clé valeur

Faites une requête avec la méthode GET en envoyant la clé “message” et la valeur “bonjour”. L’URL doit être de la forme

|  |
| --- |
| <https://httpbin.org/get?message=bonjour> |

La sortie devrait être

|  |
| --- |
| Executing request GET http://httpbin.org/get?message=bonjour HTTP/1.1  ----------------------------------------  {"args":{"message":"bonjour"},"headers":{"Accept-Encoding":"gzip,deflate","Connection":"close","Host":"httpbin.org","User-Agent":"Apache-HttpClient/4.5.5 (Java/9.0.4)"},"origin":"176.159.23.2","url":"http://httpbin.org/get?message=bonjour"} |

#### ​1.1.3.3​ Un GET avec plusieurs “clé-valeur” et l’encodage d’URL

Faites une requête avec la méthode GET en envoyant les clé-valeur suivants:

|  |  |
| --- | --- |
| message1 | bonjour |
| message2 | au revoir |
| mess age 3 | special characters : " ' / \ % ~ ! @ # $ % ^ & \* ( ) etc... |
| m e s s a g e 4 | C’est l’été à Paris! |

Attention de bien encoder le caractère espace dans la valeur du message 2, à encoder la clé “mess age 3” avec les espaces, à encoder tous les caractères spéciaux de la valeur du “mess age 3” ainsi qu’à encoder la clé “m e s s a g e 4” et les caractères accentués correspondants, et DE NE PAS ENCODER le & entre paramètres et le reste de l’URL.

Aidez-vous du tutoriel <https://alvinalexander.com/blog/post/java/how-encode-java-string-send-web-server-safe-url> et <https://www.url-encode-decode.com/>

L’URL résultante devrait être [https://httpbin.org/get?message1=bonjour&message2=au+revoir&mess+age+3=special+characters+%3A+%22+%27+%2F+%5C+%25+%7E+%21+%40+%23+%24+%25+%5E+%26+\*+%28+%29+etc...&m+e+s+s+a+g+e+4=C%27est+l%27%C3%A9t%C3%A9+%C3%A0+Paris%21](https://httpbin.org/get?message1=bonjour&message2=au+revoir&mess+age+3=special+characters+%3A+%22+%27+%2F+%5C+%25+%7E+%21+%40+%23+%24+%25+%5E+%26+*+%28+%29+etc...&m+e+s+s+a+g+e+4=C%27est+l%27%C3%A9t%C3%A9+%C3%A0+Paris%21)

Vérifiez que vous obtenez la réponse suivante

|  |
| --- |
| Executing request GET http://httpbin.org/get?mess+age+3=special+characters+%3A+%22+%27+%2F+%5C+%25+%7E+%21+%40+%23+%24+%25+%5E+%26+\*+%28+%29+etc...&m+e+s+s+a+g+e+4=C%27est+l%27%C3%A9t%C3%A9+%C3%A0+Paris%21&message2=au+revoir&message1=bonjour HTTP/1.1  ----------------------------------------  {"args":{"m e s s a g e 4":"C'est l'\u00e9t\u00e9 \u00e0 Paris!","mess age 3":"special characters: \" ' / \\ % ~ ! @ # $ % ^ & \* ( ) etc...","message1":"bonjour","message2":"au revoir"},"headers":{"Accept-Encoding":"gzip,deflate","Connection":"close","Host":"httpbin.org","User-Agent":"Apache-HttpClient/4.5.5 (Java/9.0.4)"},"origin":"176.159.23.2","url":"http://httpbin.org/get?mess+age+3=special+character%3A+\"+'+%2F+\\+%25+~+!+%40+%23+%24+%25+^+%26+\*+(+)+etc...&m+e+s+s+a+g+e+4=C'est+l'\u00e9t\u00e9+\u00e0+Paris!&message2=au+revoir&message1=bonjour"} |

Solution à demander à Rémi SHARROCK: fichier ApacheGet.java

#### ​1.1.3.4​ Un POST avec clé valeur

Maintenant, en utilisant la méthode POST envoyez les même “clé-valeurs” que la question précédente, également encodées. Aidez-vous des tutoriels

<https://hc.apache.org/httpcomponents-client-ga/quickstart.html> et <https://hc.apache.org/httpcomponents-client-ga/httpclient/examples/org/apache/http/examples/client/ClientWithResponseHandler.java>

Vérifiez que vous obtenez la réponse

|  |
| --- |
| {"args":{},"data":"","files":{},"form":{"m e s s a g e 4":"C'est l'\ufffdt\ufffd \ufffd Paris!","mess age 3":"special characters : \" ' / \\ % ~ ! @ # $ % ^ & \* ( ) etc...","message1":"bonjour","message2":"au revoir"},"headers":{"Accept-Encoding":"gzip,deflate","Connection":"close","Content-Length":"187","Content-Type":"application/x-www-form-urlencoded","Host":"httpbin.org","User-Agent":"Apache-HttpClient/4.5.5 (Java/9.0.4)"},"json":null,"origin":"176.159.23.2","url":"http://httpbin.org/post"} |

Solution à demander à Rémi SHARROCK: fichier ApachePost.java

#### ​1.1.3.5​ Un POST en multipart/form-data au lieu de application/x-www-form-urlencoded

Pour cette question, la réponse est fournie en Annexe 1.

En utilisant le tutoriel <http://www.baeldung.com/httpclient-multipart-upload>

et la solution complète du tutoriel ici : <https://github.com/eugenp/tutorials/blob/master/httpclient/src/test/java/org/baeldung/httpclient/HttpClientMultipartLiveTest.java>

Nous voulons envoyer:

* un fichier texte encodé avec un “charset” UTF-8 contenant   
  C'est un fichier encodé en UTF-8!
* deux fichiers binaires: l’un contenant la valeur de l’entier 99 et l’autre 98 **mais en simulant un contenu de fichier ZIP**.
* Pour créer les fichiers, vous pouvez vous inspirer du code suivant:

|  |
| --- |
| new File("uploads").mkdirs();  //création du fichier texte  static final String *TEXTFILENAME* = "temp.txt";  Writer out = new BufferedWriter(new OutputStreamWriter(  new FileOutputStream("uploads/"+*TEXTFILENAME*), "UTF-8"));  try {  out.write("C'est un fichier encodé en UTF-8!");  } finally {  out.close();  }  //fin création du fichier texte  //création du fichier binaire  static final String *FILE1\_BIN* = "file1.bin";  static final String *FILE2\_BIN* = "file2.bin";  int i = 99;  DataOutputStream os = new DataOutputStream(new FileOutputStream("uploads/"+*FILE1\_BIN*));  os.writeInt(i);  os.close();  i = 98;  os = new DataOutputStream(new FileOutputStream("uploads/"+*FILE2\_BIN*));  os.writeInt(i);  os.close();  //fin création du fichier binaire |

Installez les bibliothèques nécessaires pour faire fonctionner le code, notamment httpmime et junit (si vous voulez tester)

Solution à demander à Rémi SHARROCK: fichier ApachePostFiles.java

#### ​1.1.3.6​ Envoi par un PUT

Utilisez la méthode PUT pour envoyer une String contenant le JSON suivant:

|  |
| --- |
| {"id":"123", "name":"Remi Sharrock"} |

L’URL est <https://httpbin.org/put>

Utilisez le tutoriel

<https://www.javaquery.com/2016/01/apache-httpput-example.html>

La réponse doit être::

|  |
| --- |
| Response code: 200  Response: {"args":{},"data":"{\"id\":\"123\", \"name\":\"Remi Sharrock\"}","files":{},"form":{},"headers":{"Accept-Encoding":"gzip,deflate","Connection":"close","Content-Length":"36","Content-Type":"text/plain; charset=UTF-8","Host":"httpbin.org","User-Agent":"Apache-HttpClient/4.5.5 (Java/9.0.4)"},"json":{"id":"123","name":"Remi Sharrock"},"origin":"176.159.23.2","url":"https://httpbin.org/put"} |

Vérifiez bien que des \ supplémentaires ont été ajoutés avant les “ dans la réponse

Solution à demander à Rémi SHARROCK: fichier ApachePut.java

### ​1.1.4​ OkHttp

#### ​1.1.4.1​ Un get simple

Installez la bibliothèque OkHttp pour votre projet.

En utilisant les exemples sur <http://square.github.io/okhttp/> , faites un GET avec <https://httpbin.org/ip>

#### ​1.1.4.2​ Un get avec paramètres

faites un GET avec des paramètres encodés (noms et valeurs) avec <https://httpbin.org/get>

Solution à demander à Rémi SHARROCK: fichier OkHttpGet.java

#### ​1.1.4.3​ Un post d’un contenu JSON

De même, faites un POST d’un contenu JSON en utilisant <https://httpbin.org/post>

Solution à demander à Rémi SHARROCK: fichier OkHttpPost.java

#### ​1.1.4.4​ Un post d’un fichier

en vous aidant de la page <https://github.com/square/okhttp/wiki/Recipes> faites un post d’un fichier contenant du Markdown sur <https://httpbin.org/post> . Par exemple en prenant le fichier <https://raw.githubusercontent.com/square/okhttp/master/README.md>

Utilisez le mime-type (ou type de média) text/x-markdown; charset=utf-8

Solution à demander à Rémi SHARROCK: fichier OkHttpPostFiles.java

### ​1.1.5​ Google Http-client

Nous n’allons pas tester la bibliothèque Google http-client mais vous pouvez regarder l’exemple d’appel de l’API de Dailymotion <https://developers.google.com/api-client-library/java/google-http-java-client/samples>

## ​1.2​ Traiter le JSON en Java

### ​1.2.1​ Introduction

Nous avons vu beaucoup de réponses au format JSON qu’il faudra ensuite traiter pour extraire les informations que nous voulons.

Il existe plusieurs bibliothèques pour traiter du JSON en Java. Voici un extrait de <https://blog.takipi.com/the-ultimate-json-library-json-simple-vs-gson-vs-jackson-vs-json/> :

* Yidong Fang’s JSON.simple (<https://github.com/fangyidong/json-simple> ). JSON.simple is a Java toolkit for encoding and decoding JSON text. It’s meant to be a lightweight and simple library that still performs at a high level.
* Google’s GSON (<https://github.com/google/gson> ). GSON is a Java library that converts Java Objects into JSON and vice versa. It provides the added benefit of full support for Java Generics, and it doesn’t require you to annotate your classes. Not needing to add annotations makes for simpler implementation and can even be a requirement if you don’t have access to your source code.
* FasterXML’s Jackson Project (<https://github.com/FasterXML/jackson> ). Jackson is a group of data processing tools highlighted by its streaming JSON parser and generator library. Designed for Java, it can also handle other non-JSON encodings. It’s the most popular JSON parser, according to our findings on Github usages.
* Oracle’s JSONP (<https://jsonp.java.net/> ). JSONP (JSON Processing) is a Java API for JSON processing, namely around consuming and producing streaming JSON text. It’s the open source reference implementation of JSR353.

D’autres sur <https://stackoverflow.com/a/31743324>

La plus connue et la plus utilisée est Jackson et la deuxième GSON. Jackson est utilisée dans beaucoup d’autres frameworks de plus haut niveau comme Jersey, Apache CXF, RESTEasy, Restlet et Spring.

### ​1.2.2​ Sérialiser, désérialiser

Une méthode classique consiste à obtenir une équivalence JSON <---> objets Java.

**Désérialiser**: faire une conversion de JSON vers un ensemble d’objets Java.

**Sérialiser**: faire une conversion d’un ensemble d’objets Java en JSON.

Voici comment sérialiser et désérialiser du JSON en Java avec GSON et Jackson: <http://www.baeldung.com/jackson-vs-gson>

#### ​1.2.2.1​ Désérialiser une réponse

Désérialisez la réponse JSON de <https://httpbin.org/ip> en méthode GET dans une classe permettant de manipuler votre adresse ip avec HttpURLConnection et GSON.

|  |
| --- |
| public class IP {  private String origin;  public IP(String origin) {  this.origin = origin;  }  public String getOrigin() {  return origin;  }  public void setOrigin(String origin) {  this.origin = origin;  }  @Override  public String toString() {  return "IP{" +  "origin='" + origin + '\'' +  '}';  }  } |

Solution à demander à Rémi SHARROCK: fichiers IP.java et GetIP.java

#### ​1.2.2.2​ Autres combinaisons possibles

Faites de même avec différentes combinaisons si vous le souhaitez, voici toutes les combinaisons possibles:

* HttpURLConnection/GSON
* HttpURLConnection/Jackson
* OkHttp/Jackson
* OkHttp/GSON
* Apache HttpClient/GSON
* Apache HttpClient/Jackson

#### ​1.2.2.3​ Désérialiser avec des clés ayant des caractères spéciaux (espaces etc.)

Faites la désérialisation HttpURLConnection/GSON (ou autre combinaison) de la réponse à la méthode GET de:

|  |
| --- |
| [https://httpbin.org/get?message1=bonjour&message2=au+revoir&mess+age+3=special+character%3A+%22+%27+%2F+%5C+%25+%7E+%21+%40+%23+%24+%25+%5E+%26+\*+%28+%29+etc...&m+e+s+s+a+g+e+4=C%27est+l%27%C3%A9t%C3%A9+%C3%A0+Paris%21](https://httpbin.org/get?message1=bonjour&message2=au+revoir&mess+age+3=special+character%3A+%22+%27+%2F+%5C+%25+%7E+%21+%40+%23+%24+%25+%5E+%26+*+%28+%29+etc...&m+e+s+s+a+g+e+4=C%27est+l%27%C3%A9t%C3%A9+%C3%A0+Paris%21) |

Pour cela vous devez utiliser les deux classes suivantes (exemple avec GSON):

|  |
| --- |
| public class ArgsWithMessages {  public ArgsWithMessages(Messages args) {  this.args = args;  }  private Messages args;  @Override  public String toString() {  return "ArgsWithMessages{" +  "args=" + args +  '}';  }  public Messages getArgs() {  return args;  }  public void setArgs(Messages args) {  this.args = args;  }  } |

Ainsi que

|  |
| --- |
| import com.google.gson.annotations.SerializedName;  public class Messages {  @SerializedName("message1")  private String message1;  @SerializedName("message2")  private String message2;  @SerializedName("mess age 3")  private String message3;  @SerializedName("m e s s a g e 4")  private String message4;  @Override  public String toString() {  return "Messages{" +  "message1='" + message1 + '\'' +  ", message2='" + message2 + '\'' +  ", message3='" + message3 + '\'' +  ", message4='" + message4 + '\'' +  '}';  }  public String getMessage1() {  return message1;  }  public void setMessage1(String message1) {  this.message1 = message1;  }  public String getMessage2() {  return message2;  }  public void setMessage2(String message2) {  this.message2 = message2;  }  public String getMessage3() {  return message3;  }  public void setMessage3(String message3) {  this.message3 = message3;  }  public String getMessage4() {  return message4;  }  public void setMessage4(String message4) {  this.message4 = message4;  }  public Messages(String message1, String message2, String message3, String message4) {  this.message1 = message1;  this.message2 = message2;  this.message3 = message3;  this.message4 = message4;  }  } |

Solution à demander à Rémi SHARROCK: fichiers Messages.java et GetMessages.java et ArgsWithMessages.java

# ​2​ Bonnes pratiques REST

[​https://blog.mwaysolutions.com/2014/06/05/10-best-practices-for-better-restful-api/](https://blog.mwaysolutions.com/2014/06/05/10-best-practices-for-better-restful-api/)

<https://code-maze.com/top-rest-api-best-practices/>

# ​3​ Projets pour tester vos clients REST/JSON:

## ​3.1​ Quizz musical Deezer

Créez un quizz musical qui utilise l’API Deezer.

Votre programme doit être en ligne de commande, pas besoin d’interface graphique.

Choisissez une playlist sur Deezer.  
Prenez une chanson au hasard, jouez le son. L’utilisateur doit deviner le nom de l’artiste et de la chanson pendant la lecture du son. Il perd s’il ne trouve pas à la fin du son.  
Le son suivant est joué et le jeu se termine quand toutes les chansons de la playlist ont été jouées.

Pour comparer la réponse du joueur avec la véritable réponse, utilisez de la logique floue (Fuzzy logic). Une implémentation connue pour vous aider à faire cela est disponible ici : <https://github.com/xdrop/fuzzywuzzy>

Pour lire un fichier mp3, vous pouvez utiliser Media et MediaPlayer, attention ceci n’est compatible qu’avec une version du JDK fournie par Oracle (OpenJDK n’est pas compatible).

|  |
| --- |
| Media hit = new Media("https://cdns-preview-4.dzcdn.net/stream/c-4c01676b7ddb90d97976d4dfea1b15de-4.mp3");  MediaPlayer mediaPlayer = new MediaPlayer(hit);  mediaPlayer.play(); |

## ​3.2​ Horaires temps réel du RER B

Affichez les horaires en temps réel du passage du RER B à la station cité universitaire. Utilisez l’API temps réel du STIF. <https://api-lab-trone-stif.opendata.stif.info>

Deux sources sont disponibles: vianavigo et ile-de-france mobilités. Vianavigo renvoie du JSON et ile-de-france mobilités renvoie du XML.

# ​4​ Annexes

## ​4.1​ Annexe 1

|  |
| --- |
| import org.apache.http.HttpEntity;  import org.apache.http.client.methods.CloseableHttpResponse;  import org.apache.http.client.methods.HttpPost;  import org.apache.http.entity.ContentType;  import org.apache.http.entity.mime.HttpMultipartMode;  import org.apache.http.entity.mime.MultipartEntityBuilder;  import org.apache.http.entity.mime.content.FileBody;  import org.apache.http.entity.mime.content.StringBody;  import org.apache.http.impl.client.CloseableHttpClient;  import org.apache.http.impl.client.HttpClientBuilder;  import org.junit.After;  import org.junit.Before;  import org.junit.Test;  import java.io.\*;  import java.util.logging.Level;  import java.util.logging.Logger;  import static org.junit.Assert.*assertTrue*;  public class ApachePostFiles {  private static final String *SERVER* = "https://httpbin.org/post";  private static final String *TEXTFILENAME* = "temp.txt";  private static final String *FILE1\_BIN* = "file1.bin";  private static final String *FILE2\_BIN* = "file2.bin";  private static final Logger *LOGGER* = Logger.*getLogger*("ApachePostFilesFiles");  private CloseableHttpClient client;  private HttpPost post;  private BufferedReader rd;  private CloseableHttpResponse response;  @Before  public final void before() throws Exception {  client = HttpClientBuilder.*create*()  .build();  post = new HttpPost(*SERVER*);  new File("uploads").mkdirs();  //création du fichier texte  Writer out = new BufferedWriter(new OutputStreamWriter(  new FileOutputStream("uploads/"+*TEXTFILENAME*), "UTF-8"));  try {  out.write("C'est un fichier encodé en UTF-8!");  } finally {  out.close();  }  //fin création du fichier texte  //création du fichier binaire  int i = 99;  DataOutputStream os = new DataOutputStream(new FileOutputStream("uploads/"+*FILE1\_BIN*));  os.writeInt(i);  os.close();  i = 98;  os = new DataOutputStream(new FileOutputStream("uploads/"+*FILE2\_BIN*));  os.writeInt(i);  os.close();  //fin création du fichier binaire  }  @After  public final void after() throws IllegalStateException, IOException {  post.completed();  try {  client.close();  } catch (final IOException e1) {  *LOGGER*.log(Level.*SEVERE*, e1.getMessage(), e1);  throw e1;  }  try {  rd.close();  } catch (final IOException e) {  *LOGGER*.log(Level.*SEVERE*, e.getMessage(), e);  throw e;  }  }  // tests  @Test  public final void givenFileandMultipleTextParts\_whenUploadwithAddPart\_thenNoExceptions() throws IOException {  final File file = new File("uploads/" + *TEXTFILENAME*);  final FileBody fileBody = new FileBody(file, ContentType.*DEFAULT\_BINARY*);  final StringBody stringBody1 = new StringBody("This is message 1", ContentType.*MULTIPART\_FORM\_DATA*);  final StringBody stringBody2 = new StringBody("This is message 2", ContentType.*MULTIPART\_FORM\_DATA*);  //  final MultipartEntityBuilder builder = MultipartEntityBuilder.*create*();  builder.setMode(HttpMultipartMode.*BROWSER\_COMPATIBLE*);  builder.addPart("textfile", fileBody);  builder.addPart("text1", stringBody1);  builder.addPart("text2", stringBody2);  final HttpEntity entity = builder.build();  //  post.setEntity(entity);  response = client.execute(post);  System.*out*.println("Status: "+response.getStatusLine()  .getStatusCode());  final String responseString = getContent();  final String contentTypeInHeader = getContentTypeHeader();  // assertTrue(responseString.contains("Content-Type: multipart/form-data;"));  *assertTrue*(contentTypeInHeader.contains("Content-Type: multipart/form-data;"));  System.*out*.println(responseString);  System.*out*.println("POST Content Type: " + contentTypeInHeader);  }  @Test  public final void givenFileandTextPart\_whenUploadwithAddBinaryBodyandAddTextBody\_ThenNoExeption() throws IOException {  final File file = new File("uploads/" + *TEXTFILENAME*);  final String message = "This is a multipart post";  final MultipartEntityBuilder builder = MultipartEntityBuilder.*create*();  builder.setMode(HttpMultipartMode.*BROWSER\_COMPATIBLE*);  builder.addBinaryBody("textfile", file, ContentType.*DEFAULT\_BINARY*, *TEXTFILENAME*);  builder.addTextBody("text", message, ContentType.*DEFAULT\_BINARY*);  final HttpEntity entity = builder.build();  post.setEntity(entity);  response = client.execute(post);  System.*out*.println("Status: "+response.getStatusLine()  .getStatusCode());  final String responseString = getContent();  final String contentTypeInHeader = getContentTypeHeader();  // assertTrue(responseString.contains("Content-Type: multipart/form-data;"));  *assertTrue*(contentTypeInHeader.contains("Content-Type: multipart/form-data;"));  System.*out*.println(responseString);  System.*out*.println("POST Content Type: " + contentTypeInHeader);  }  @Test  public final void givenFileAndInputStreamandText\_whenUploadwithAddBinaryBodyandAddTextBody\_ThenNoException() throws IOException {  final InputStream inputStream = new FileInputStream("uploads/" + *FILE2\_BIN*);  final File file = new File("uploads/" + *FILE1\_BIN*);  final String message = "This is a multipart post";  final MultipartEntityBuilder builder = MultipartEntityBuilder.*create*();  builder.setMode(HttpMultipartMode.*BROWSER\_COMPATIBLE*);  builder.addBinaryBody("file1", file, ContentType.*DEFAULT\_BINARY*, *FILE1\_BIN*);  builder.addBinaryBody("file2", inputStream, ContentType.*create*("application/zip"), *FILE2\_BIN*);  builder.addTextBody("text", message, ContentType.*TEXT\_PLAIN*);  final HttpEntity entity = builder.build();  post.setEntity(entity);  response = client.execute(post);  System.*out*.println("Status: "+response.getStatusLine()  .getStatusCode());  final String responseString = getContent();  final String contentTypeInHeader = getContentTypeHeader();  // assertTrue(responseString.contains("Content-Type: multipart/form-data;"));  *assertTrue*(contentTypeInHeader.contains("Content-Type: multipart/form-data;"));  System.*out*.println(responseString);  System.*out*.println("POST Content Type: " + contentTypeInHeader);  inputStream.close();  }  @Test  public final void givenCharArrayandText\_whenUploadwithAddBinaryBodyandAddTextBody\_ThenNoException() throws IOException {  final String message = "This is a multipart post";  final byte[] bytes = "binary code".getBytes();  final MultipartEntityBuilder builder = MultipartEntityBuilder.*create*();  builder.setMode(HttpMultipartMode.*BROWSER\_COMPATIBLE*);  builder.addBinaryBody("textfile", bytes, ContentType.*DEFAULT\_BINARY*, *TEXTFILENAME*);  builder.addTextBody("text", message, ContentType.*TEXT\_PLAIN*);  final HttpEntity entity = builder.build();  post.setEntity(entity);  response = client.execute(post);  System.*out*.println("Status: "+response.getStatusLine()  .getStatusCode());  final String responseString = getContent();  final String contentTypeInHeader = getContentTypeHeader();  // assertTrue(responseString.contains("Content-Type: multipart/form-data;"));  *assertTrue*(contentTypeInHeader.contains("Content-Type: multipart/form-data;"));  System.*out*.println(responseString);  System.*out*.println("POST Content Type: " + contentTypeInHeader);  }  // UTIL  private String getContent() throws IOException {  rd = new BufferedReader(new InputStreamReader(response.getEntity()  .getContent()));  String body;  StringBuilder content = new StringBuilder();  while ((body = rd.readLine()) != null) {  content.append(body).append("\n");  }  return content.toString().trim();  }  private String getContentTypeHeader() {  return post.getEntity()  .getContentType()  .toString();  }  } |

## ​4.2​ Annexe 2: ressources complémentaires

<https://alvinalexander.com/java/java-apache-httpclient-restful-client-examples>

<https://stackoverflow.com/questions/221442/rest-clients-for-java?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa>

<https://stackoverflow.com/questions/42199614/jersey-rest-client-with-apache-http-client-4-5-vs-retrofit?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa>

<https://stackoverflow.com/questions/8267928/android-rest-client-sample?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa>

<http://www.baeldung.com/google-http-client>

<https://github.com/square/okhttp/wiki/Recipes>