Jakarta RESTful Web Services (JAX-RS)

Emmanuel BRUNO
Université de Toulon, LIS UMR CNRS 7020
emmanuel.bruno@univ-tln.fr

2026-11-01

Table des matières

1 Une application REST minimale 1
2 Une application REST plus complète 4
2.1 Chemins et Verbes 4
2.2 Les paramètres simples 5

Java propose un standard appelé Jakarta RESTful Web Services pour construire efficacement des serveurs et des clients REST. Jersey est l'implantation de référence.

Ce cours présente la version 3.1, la version actuelle de la spécification est la 4.0

1 Une application REST minimale

Pour créer des applications REST en Java, il faut utiliser une implantation de Jakarta RESTful Web Services. Par exemple, il est possible d'utiliser le framework Jersey qui est l'implantation de référence de la spécification JAX-RS. Dans un premier temps nous allons étudier une application minimale qui s'appuie sur Jersey intégré dans un serveur Web en Java Grizzly.

Dans la partie pratique, nous utiliserons une autre approche pour créer des applications REST en Java en utilisant le framework Quarkus.

L'archetype maven suivant permet de créer un projet de base dans le répertoire `/home/jovyan/work/src/samples/jaxrs/myresource`.

```
%%shell
mkdir -p /home/jovyan/work/src/samples/jaxrs
cd /home/jovyan/work/src/samples/jaxrs
rm -rf /home/jovyan/work/src/samples/jaxrs/myresource

mvn archetype:generate --batch-mode --no-transfer-progress --quiet \
  -DarchetypeGroupId=org.glassfish.jersey.archetypes \
```

```
  -DarchetypeArtifactId=jersey-quickstart-grizzly2 \
  -DarchetypeVersion=3.1.10 \
  -DgroupId=fr.univtln.bruno.demos.jaxrs \
  -DartifactId=myresource
```

Le serveur peut être compilé puis exécuté

```
cd /home/jovyan/work/src/samples/jaxrs/myresource
mvn package && mvn exec:java
```

Il est maintenant possible d'accéder à la ressource en ligne de commande à partir de l'adresse http://localhost:8080/myapp.

```
%%shell
cd /home/jovyan/work/src/samples/jaxrs/myresource
mvn --quiet --ntp package
nohup mvn --quiet --ntp exec:java &
```

```
Picked up JAVA_TOOL_OPTIONS: -XX:MaxRAMPercentage=25
Nov 01, 2026 12:20:11 PM
org.glassfish.jersey.message.internal.MessagingBinders$EnabledProvidersBinder
bindToBinder
WARNING: A class jakarta.activation.DataSource for a default provider
MessageBodyWriter<jakarta.activation.DataSource> was not found. The provider is
not available.
Nov 01, 2026 12:20:11 PM org.glassfish.jersey.server.wadl.WadlFeature configure
WARNING: JAX-B API not found . WADL feature is disabled.
Nov 01, 2026 12:20:11 PM org.glassfish.grizzly.http.server.NetworkListener start
INFO: Started listener bound to [localhost:8080]
Nov 01, 2026 12:20:11 PM org.glassfish.grizzly.http.server.HttpServer start
INFO: [HttpServer] Started.
Nov 01, 2026 12:20:12 PM org.glassfish.grizzly.http.server.NetworkListener
shutdownNow
INFO: Stopped listener bound to [localhost:8080]
```

Une application REST JAX-RS est construite autour de deux notions principales l'Application (le serveur) et les Ressources. Une instance d'une ressource est créée pour répondre à chaque requête et détruite ensuite. Elle peut donc être utilisée comme une sorte de singleton. Une ressource est définie annotant une classe, un ou plusieurs de ses superclasses (y compris abstraites) ou l'une de ses interfaces.

Dans l'exemple, la classe `fr.univtln.bruno.demos.jaxrs.Main` démarre le serveur et paramètre les packages où le framework va chercher des ressources comme le montre la méthode ci-dessous.

```
/**
 * Starts Grizzly HTTP server exposing JAX-RS resources defined in this application.
 * @return Grizzly HTTP server.
 */
public static HttpServer startServer() {
    // create a resource config that scans for JAX-RS resources and providers
    // in fr.univtln.bruno.demos.jaxrs package
                            final       ResourceConfig       rc       =       new ResourceConfig().packages("fr.univtln.bruno.demos.jaxrs");
    // create and start a new instance of grizzly http server
    // exposing the Jersey application at BASE_URI
    return GrizzlyHttpServerFactory.createHttpServer(URI.create(BASE_URI), rc);
}
```

La classe `fr.univtln.bruno.demos.jaxrs.MyResource` présente le fonctionnement minimal d'une ressource. La classe est annotée avec `@Path(...)` pour indiquer le chemin à ajouter à l'URL correspondant à cette ressource. D'une façon générale, les méthodes sont annotées avec `@POST`, `@GET`, `@PUT`,`@DELETE`, ... pour indiquer le type de verbe HTTP associé.

Les méthodes peuvent être annotées avec `@Produces` qui indique le ou les types MIME dans lequel le résultat peut être fourni : `@Produces("text/plain")`, `@Produces("application/json")`, ... Il est possible d'indiquer plusieurs types avec `@Produces({"application/json", "application/xml"})`. Il existe aussi des constantes équivalentes `MediaType.TEXT_PLAIN`. Une valeur par défaut de `@Produces` peut être indiquée en annotant la classe.

```
/**
 * Method handling HTTP GET requests. The returned object will be sent
 * to the client as "text/plain" media type.
 *
 * @return String that will be returned as a text/plain response.
 */
@GET
@Produces(MediaType.TEXT_PLAIN)
public String getIt() {
    return "Got it!";
}
```

Le client peut indiquer le type demandé parmi l'un des « produces » avec la valeur `Content-Type:` de l'entête de la requête.

La commande suivante exécute une requête GET sur l'URL d'une ressource et affiche le résultat en-tête compris. (Le serveur doit être lancé).

2 Une application REST plus complète

Pour la suite nous allons étudier en détail l'application https://github.com/ebpro/sample-jaxrs.

Nov 01, 2026 12:20:20 PM org.glassfish.jersey.message.internal.MessagingBinders$EnabledProvidersBinder bindToBinder WARNING: A class jakarta.activation.DataSource for a default provider MessageBodyWriter<jakarta.activation.DataSource> was not found. The p

```
EvaluationTimeoutException: Evaluator timed out after 60 seconds while executing:
'cellMagic(new String(Base64.getDecoder().decode("c2hlbGw=")),List.of(),new String(Base64.getDecod
|  cellMagic(new String(Base64.getDecoder().decode("c2hlbGw=")),List.of(),new String(Base64.getDec
Evaluation timed out after 60 seconds.
```

L'application peut être compilée et exécutée avec maven. Cela lance le serveur (vous pourrez l'arrêter avec ctrl-c dans le terminal). Cela compile, exécute les tests unitaires, package et exécute les tests d'intégration (en lançant le serveur REST et en exécutant de vraies requêtes).

```
cd /home/jovyan/work/materials/github/ebpro/sample-jakarta-rest-full && \
    mvn clean verify &&
    mvn exec:java
```

La classe `fr.univtln.bruno.samples.jaxrs.server.BiblioServer` paramètre Jersey, démarre Grizzly et attend un CTRL-C pour arrêter le serveur.

La classe `fr.univtln.bruno.samples.jaxrs.model.LibraryModel` définit le modèle de donnée (Une bibliothèque qui est une facade pour gérer des Auteurs et des Livres.)

Les classes `fr.univtln.bruno.samples.jaxrs.resources.LibraryResource` et `fr.univtln.bruno.samples.jaxrs.resources.AuthorResource` définissent des ressources REST.

2.1 Chemins et Verbes

La méthode `sayHello()` reprend l'exemple précédent.

Error: failed to read file `/home/jovyan/work/materials/github/ebpro/sample-jakarta-rest-full/src/main/java/fr/univtln/bruno/samples/jaxrs/resources/LibraryResource.java`: /home/jovyan/work/materials/github/ebpro/

sample-jakarta-rest-full/src/main/java/fr/univtln/bruno/samples/jaxrs/resources/LibraryResource.java

D'autres verbe peuvent être utilisé sur le même chemin. Une méthode peut aussi être annotées avec `@Path` pour définir le chemin associé à cette méthode.

La méthode `init()` est un simple `PUT` sans paramètre qui initialise la bibliothèque avec deux auteurs.

Error: failed to read file `/home/jovyan/work/materials/github/ebpro/sample-jakarta-rest-full/src/main/java/fr/univtln/bruno/samples/jaxrs/resources/LibraryResource.java`: /home/jovyan/work/materials/github/ebpro/sample-jakarta-rest-full/src/main/java/fr/univtln/bruno/samples/jaxrs/resources/LibraryResource.java

2.2 Les paramètres simples

JAX-RS permet d'extraire automatiquement des valeurs de paramètres depuis le chemin de la ressources, les paramêtres de la requête ou l'entête http. Ces valeurs peuvent alors être « injectées » (affectée par annotation aux paramètres des méthodes REST).

L'annotation `@PathParam` permet d'injecter les valeurs provenant des URL comme des paramètres.

La méthode `getAuthor(@PathParam("id") final long id)` ci dessous-s'exécute lors d'un `GET` sur un chemin de forme `@Path("author/{id}")`. `id` est est un pas de chemin quelconque qui sera extrait, converti en long et injecté grâce à `@PathParam` dans le paramètre `id` de la fonction. Il est possible d'indiquer une expression régulière pour contraindre la forme du pas par exemple `@Path("authors/{id: [0-9]+}")`.

Le `@Produces` sur la classe indique que du XML ou du JSON peuvent être produits. Les méthodes REST retournent instance de la classe Response qui représente une réponse HTTP. Cette classe propose un builder pour construire manuellement. Cependant, JAX-RS permet de construite automatiquement ces réponses si le type de retour peut être transformé en un contenu de réponse (une entité http) par une implantation de l'interface `MessageBodyWriter`. Les implantations de JAX-RS en fournissent généralement par défaut par exemple pour String voire pour XML ou JSON (via les mécanismes de marshalling qui seront étudiés en détail plus tard).

Error: failed to read file `/home/jovyan/work/materials/github/ebpro/sample-jakarta-rest-full/src/main/java/fr/univtln/bruno/samples/jaxrs/resources/AuthorResource.java`: /home/jovyan/work/materials/github/ebpro/sample-jakarta-rest-full/src/main/java/fr/univtln/bruno/samples/jaxrs/resources/AuthorResource.java

Get author 1 in JSON :

La requête suivante reprend la précédente et demande du XML. JAX-RS va chercher automatiquement des classes (MessageBodyWriter et Reader) pour créer le bon format. Ces classes peuvent construites explicitement mais des extensions peuvent être ajoutées pour produire les types classiques par annotations des entités (cf. le pom.xml et les annotations de la classe `BiblioModel.Auteur`) : jersey-media-jaxb pour XML et jersey-media-json-jackson pour JSON. Jackson n'est pas l'implantatation pas défaut mais elle est plus efficace et plus configurable.

Get author 2 in XML :

Les collections classiques sont supportés. Notez qu'ici les collections eclipse sont utilisées en particulier celles pour les primitifs et qu'elles sont supportées par Jackson.

Get authors in XML :

Error: failed to read file `/home/jovyan/work/materials/github/ebpro/sample-jakarta-rest-full/src/main/java/fr/univtln/bruno/samples/jaxrs/resources/AuthorResource.java`: /home/jovyan/work/materials/github/ebpro/sample-jakarta-rest-full/src/main/java/fr/univtln/bruno/samples/jaxrs/resources/AuthorResource.java

Get authors in JSON

D'une façon similaire les annotations `@HeaderParam` et `@QueryParam` permettent d'extraire des valeurs de l'entête ou des paramètres de la requête http. La méthode suivante permet de construire un filtre pour des requêtes complexe. L'utilisation d'un chemin différent (« filter ») n'est utile que pour l'exemple dans une application réelle il n'y aura qu'un seul GET.

Error: failed to read file `/home/jovyan/work/materials/github/ebpro/sample-jakarta-rest-full/src/main/java/fr/univtln/bruno/samples/jaxrs/resources/AuthorResource.java`: /home/jovyan/work/materials/github/ebpro/sample-jakarta-rest-full/src/main/java/fr/univtln/bruno/samples/jaxrs/resources/AuthorResource.java

Pour simplifier le traitement des paramètres JAX-RX propose l'annotation `@BeanParam` qui permet de créer un instance d'une classe à partir des paramètres extraits. Pour cela, les propriétés de la classe peuvent être annotées pour indiquer les paramêtres correspondants.

L'exemple suivant montre comment l'utiliser pour mettre en place la pagination qui est essentielle quand le volume des données peut être important. Là aussi le chemin spécifique (« page ») n'est là que pour l'exemple.

Error: failed to read file `/home/jovyan/work/materials/github/ebpro/sample-jakarta-rest-full/src/main/java/fr/univtln/bruno/samples/`

`jaxrs/resources/PaginationInfo.java`: /home/jovyan/work/materials/github/ebpro/sample-jakarta-rest-full/src/main/java/fr/univtln/bruno/samples/jaxrs/resources/PaginationInfo.java

Error: failed to read file `/home/jovyan/work/materials/github/ebpro/sample-jakarta-rest-full/src/main/java/fr/univtln/bruno/samples/jaxrs/resources/AuthorResource.java`: /home/jovyan/work/materials/github/ebpro/sample-jakarta-rest-full/src/main/java/fr/univtln/bruno/samples/jaxrs/resources/AuthorResource.java

L'appel suivant de l'API génère alétoirement 100 auteurs.

On peut alors demander la page 3 (de taille 4).

L'appel suivant de l'API remet uniquement deux auteurs dans la base de données.