Services web RESt - Swagger / Swagger-ui

Il est possible de paramétrer Swagger via le fichier de configuration Spring "*/WEB-INF/beans.xml*" en déclarant :

- un bean faisant référence à la classe org.apache.cxf.jaxrs.swagger.Swagger2Feature
  Exemple :

```
<bean id="swagger2Feature" class="org.apache.cxf.jaxrs.swagger.Swagger2Feature" lazy-init="true">
      <property name="basePath" value="/jaxrs/ws"/>
      <property name="resourcePackage" value="hardis.jaxrs"/>
      <property name="contact" value="hotline.adelia@hardis.fr" />
      <property name="title" value="REST Services List" />
      <property name="description" value="REST services documentation" />
      <property name="version" value="2.1" />
      <property name="termsOfServiceUrl" value="http://www.hardis-group.com" />
   </bean>
```

Le bean *Swagger2Feature* doit être référencé au niveau du point d'accès (jaxrs:server) pour activer la production du document *swagger.json* (utilisé par la suite par *Swagger-ui* pour afficher les informations/documentation de vos APis).

```
<jaxrs:server id="rest" address="/" transportId="http://cxf.apache.org/transports/http">
    <jaxrs:features>
      <ref bean="swagger2Feature" />
    </jaxrs:features>
  </jaxrs:server>
```

Détail des propriétés de la feature Swagger2Feature

Name	Description	Default
basePath	the context root path+	null
contact	the contact information+	"users@cxf.apache.org"
description	the description+	"The Application"
filterClass	a security filter+	null
host	the host and port+	null
ignoreRoutes	excludes specific paths when scanning all resources (see scanAllResources)++	null
license	the license+	"Apache 2.0 License"
licenceUrl	the license URL+	"http://www.apache.org/licenses/LICENSE-2.0.html"
prettyPrint	when generating swagger.json, pretty-print the json document+	false
resourcePackage	a list of comma separated package names where resources must be scanned+	a list of service classes configured at the endpoint
runAsFilter	runs the feature as a filter	false
scan	generates the swagger documentation+	true
scanAllResources	scans all resources including non-annotated JAX-RS resources++	false
schemes	the protocol schemes+	null
termsOfServiceUrl	the terms of service URL+	null
title	the title+	"Sample REST Application"
version	the version+	"1.0.0"

Remarques :

- La propriété *resourcePackage* permet de filtrer les ressources (services) scannées et par extension les ressources (services) ajoutées au swagger.json.
- Les propriétés *scanAllResources* et *ignoreRoutes* fonctionnent de pair. Si *scanAllResources* a la valeur *true*, il est alors possible d'exclure les ressources (services) des chemins (paths) précisés.
- La propriété basePath n'est plus utile car calculée automatiquement.

La déclaration d'un bean faisant référence à la classe *com.hardis.adelia.webservice.SwaggJaxbInit* permet de modifier les propriétés utile à la sérialisation /désérialisation utile à la production du document swagger.json (s'appuie sur fasterXML/Jackson).

Exemple :

```
<bean id="SwaggJaxbInit" class="com.hardis.adelia.webservice.SwaggJaxbInit" init-method="init" >
    <property name="jaxbMapper" value="true"/>
    <property name="serializeWRAP_ROOT_VALUE" value="false"/>
    <property name="serializeWRITE_DATES_AS_TIMESTAMPS" value="true"/>
    <property name="serializeWRITE_DATE_KEYS_AS_TIMESTAMPS" value="false"/>
    <property name="serializeWRITE_CHAR_ARRAYS_AS_JSON_ARRAYS" value="false"/>
    <property name="serializeWRITE_EMPTY_JSON_ARRAYS" value="true"/>
    <property name="serializeWRITE_SINGLE_ELEM_ARRAYS_UNWRAPPED" value="false"/>
    <property name="deserializeUSE_JAVA_ARRAY_FOR_JSON_ARRAY" value="false"/>
    <property name="deserializeACCEPT_SINGLE_VALUE_AS_ARRAY" value="false"/>
    <property name="deserializeUNWRAP_ROOT_VALUE" value="false"/>
    <property name="deserializeUNWRAP_SINGLE_VALUE_ARRAYS" value="false"/>
    <property name="deserializeFAIL_ON_IGNORED_PROPERTIES" value="false"/>
    <property name="deserializeFAIL_ON_UNKNOWN_PROPERTIES" value="true"/>
  </bean>
```

Les détails sur les propriétés se trouvent :
https://github.com/FasterXML/jackson-databind/wiki/Serialization-features
https://github.com/FasterXML/jackson-databind/wiki/Deserialization-Feature

(V13 PTF9)

La version V13 PTF09 intègre les librairies :

- CXF 3.1.6
- swagger.core 1.5.9
- swagger.ui 2.1.4

(V14 PTF00)

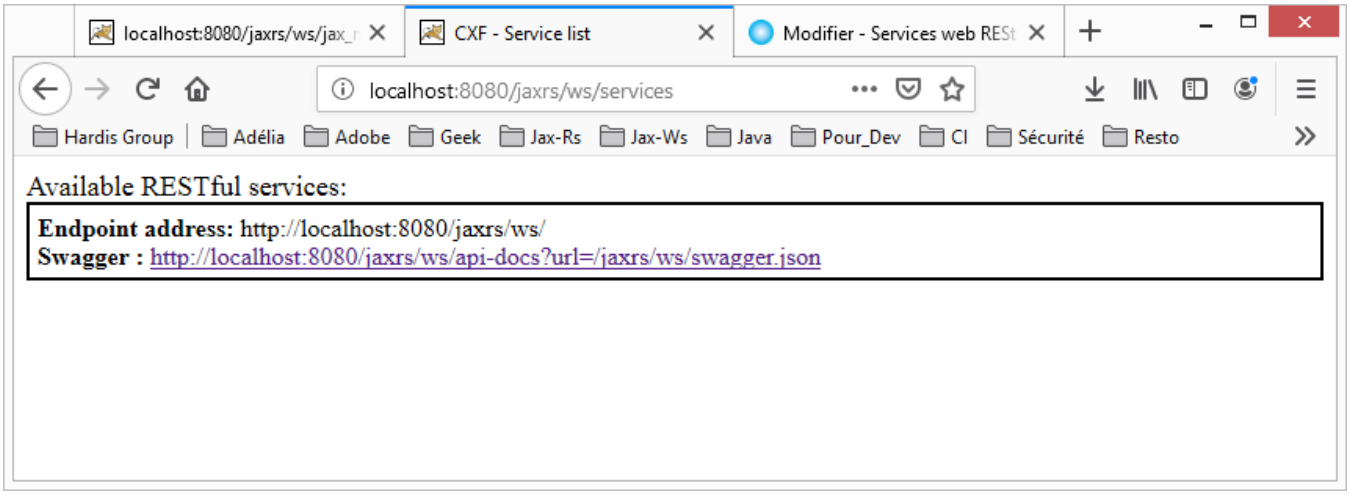
La version V14 PTF00 intègre les librairies :

- CXF 3.1.17
- swagger.core 1.5.18
- swagger.ui 3.13.0

Par défaut tous les éléments de swagger-ui sont désormais intégrés dans le .jar *swagger-ui-3.13.0.jar*.
La présence dans le bean *Swagger2Feature* (Cf. *beans.xml*) de la propriété *supportSwaggerUi* avec la valeur fixée à *true* permet de rendre opérationnel *S wagger-ui* sans avoir à décompacter l'archive/jar dans le dossier de l'application web.

L'accès à *Swagger-ui* pour les différents points d'accès (*jaxrs:server*) est disponible via l'URL : http(s)://*host*:*port*/*ContextRoot*/*CXFServletMapping*/**services**

Exemple :



Il est possible de paramétrer la page de *Swagger-ui* atteinte par le lien présenté ci-dessus en ajoutant des paramètres à l'url http(s)://*host*:*port*/*ContextRoot*/*CXFServletMapping*/**api-docs**.
Par défaut le lien proposé ajoute le paramètre *url* pour localiser le fichier swagger.json en entrée. **(?url=jaxrs/ws/swagger.json)**. D'autres paramètres peuvent être ajoutés permettant par exemple de trier les méthodes ou les tags.
Tous les paramètres sont listés dans la page suivante : https://github.com/swagger-api/swagger-ui/blob/master/docs/usage/configuration.md
Certaines options de configuration ne peuvent pas être fixées à l'aide de paramètres ; il faut alors modifier le fichier *index.html* inclus dans l'archive swagger-ui-3.x.y.jar.

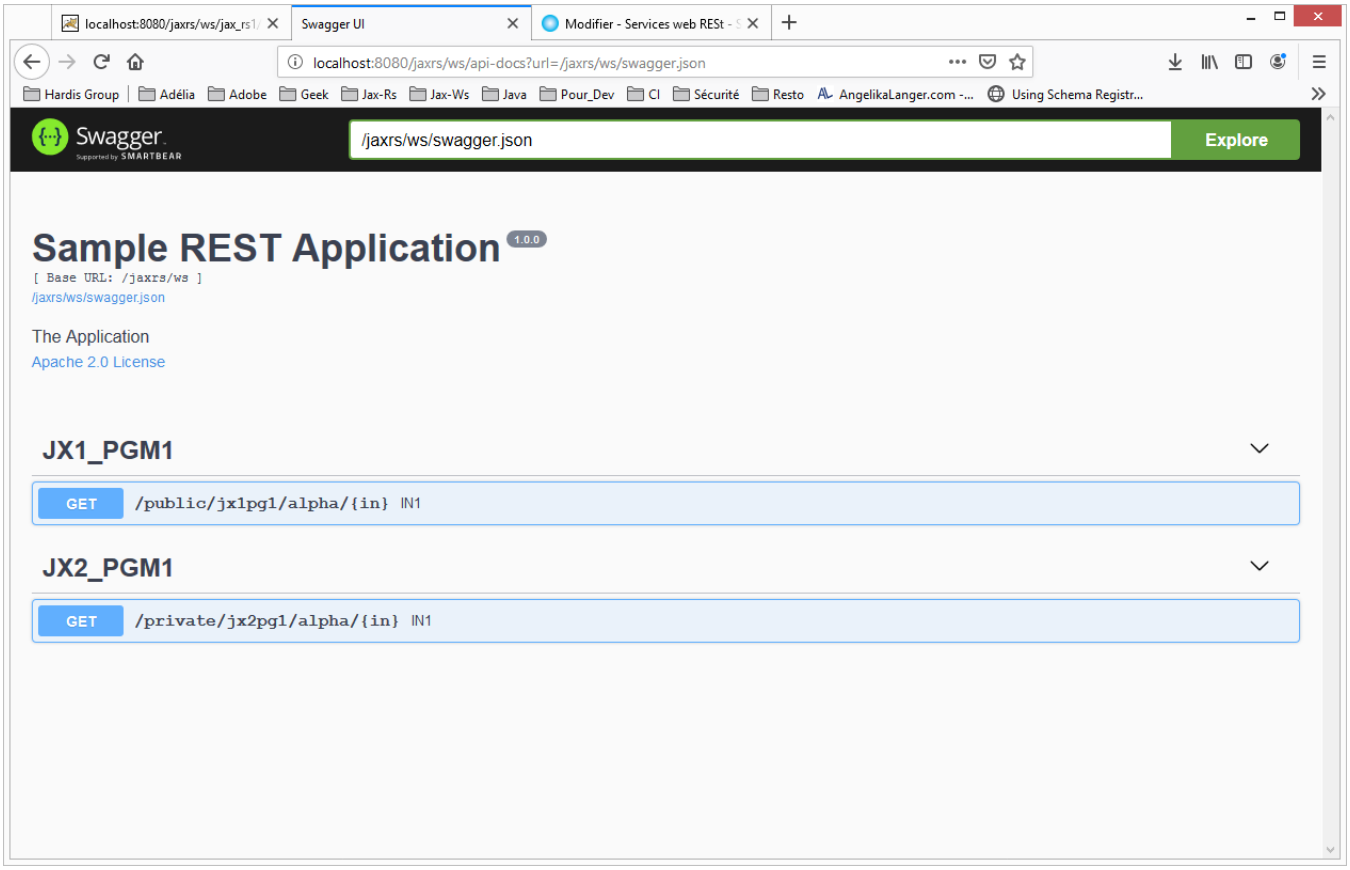
La modification du fichier *index.html* permet également d'avoir recours à l'utilisation des objets *requestInterceptor* et *responseInterceptor* lors de la création de l'objet *SwaggerUIBundle*. Ces *"interceptor*" permettent respectivement de modifier la requête adressée au serveur et de modifier la réponse retournée par le serveur avant son affichage.
Le *requestInterceptor* peut par exemple être utilisé pour ajouter un header HTTP à la requête (exemple : récupérer un jeton JWT passé dans URL pour l'injecter dans le header *Authorization*).
Le *responseInterceptor* peut par exemple modifier le swagger.json retourné afin d'effectuer un filtrage de ressources.

Exemple (*index.html*; ajout dans le header *Authorization* du jeton passé dans l'URL via un paramètre nommé *apiKey*:

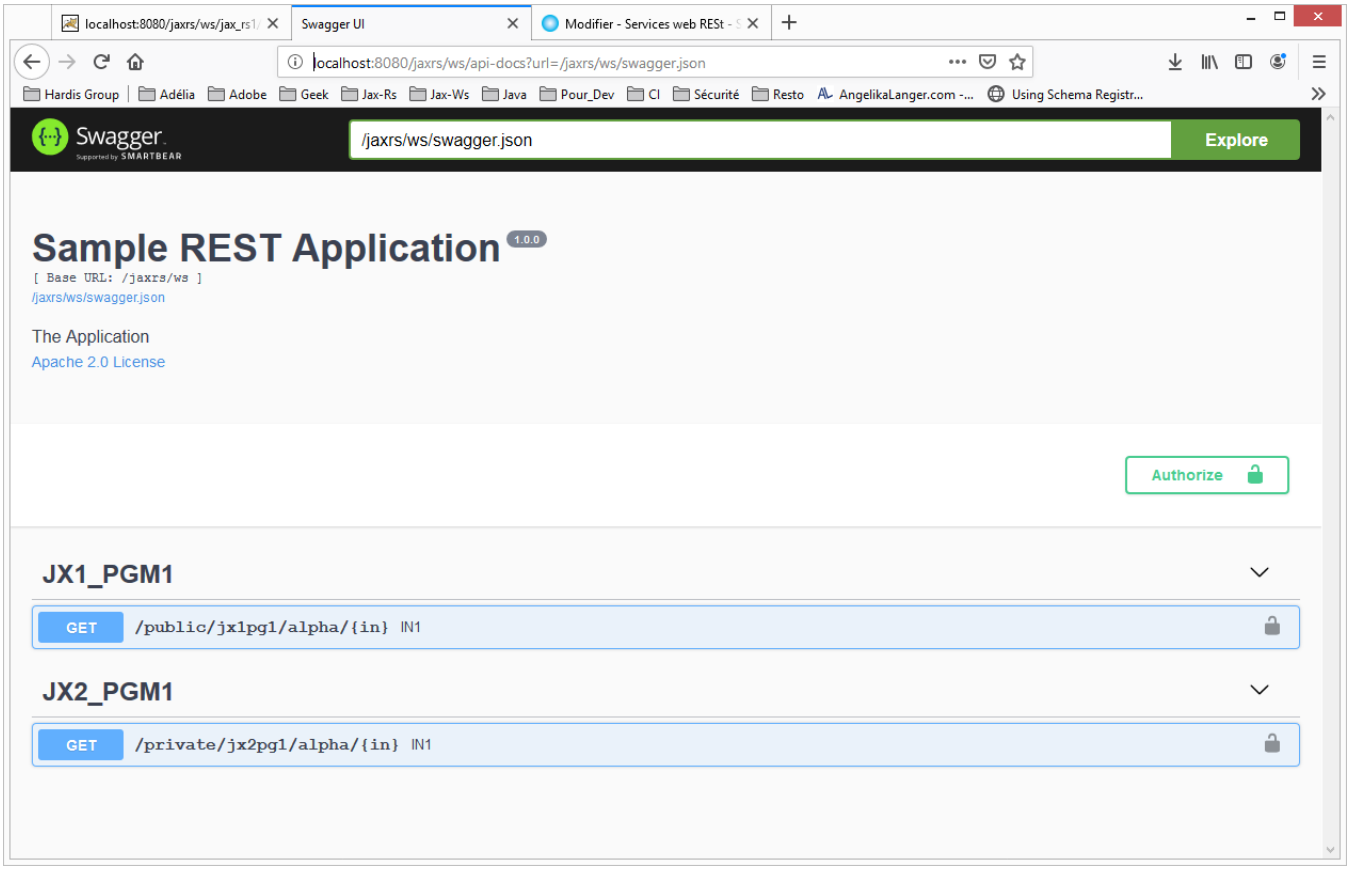
index.html

```
<script>
  function getUrlParameter(name) {
         name = name.replace(/[\[]/, '\\[').replace(/[\]]/, '\\]');
         var regex = new RegExp('[\\?&]' + name + '=([^&#]*)');
         var results = regex.exec(location.search);
         return results === null ? '' : decodeURIComponent(results[1].replace(/\+/g, ' '));
       };

     var apiKey = getUrlParameter('apiKey');

     window.onload = function() {
     // Begin Swagger UI call region
     const ui = SwaggerUIBundle({
       url: "/jaxrs/ws/swagger.json",
       dom_id: '#swagger-ui',
       deepLinking: true,
       validatorUrl:null,
       presets: [
         SwaggerUIBundle.presets.apis,
         SwaggerUIStandalonePreset
       ],
       plugins: [
         SwaggerUIBundle.plugins.DownloadUrl
       ],
       layout: "StandaloneLayout",

       requestInterceptor: function(request) {
       request.headers.Authorization = "JWT " + apiKey;
       return request;
       }
     })
```

En cliquant sur le lien proposé un peu plus haut (http://localhost:8080/jaxrs/ws/api-docs?url=/jaxrs/ws/swagger.json), la page suivante s'affiche :



L'exemple de modification du fichier *index.html* vu ci-dessus propose une solution pour automatiser le passage d'un jeton JWT au header *Authorization* à la requête.
La gestion du passage du jeton ou des informations d'une authentification de type HTTP Basic peut également se faire via l'interface de Swagger-ui.
L'interface de Swagger-ui s'adapte au contenu retourné par le serveur dans le swagger.json ; si ce dernier contient des informations de type*SwaggerSecuri tyDefinitions* alors l'interface offre un bouton [Authorize] permettant de saisir les informations d'authentification.

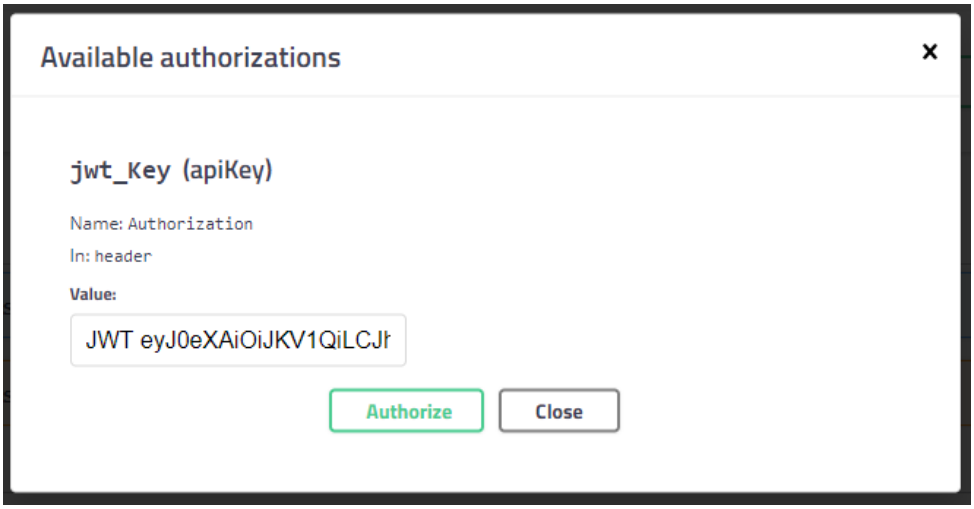


Accès à une ressource sécurisée par un jeton JWT
Pour que Swagger-ui offre la possibilité de saisir des informations d'authentification, il faut ajouter le package *com.hardis.adelia.webservice* à la propriété *r esourcePackage* du bean *Swagger2Feature* (le package *com.hardis.adelia.webservice* contient un service factice qui permet d'ajouter une information de type *SwaggerSecurityDefinitions* dans le document swagger.json produit, information dont le contenu est ajustable en fonction de la propriété (*jaxrs: properties*) *SwaggerSecurityDefinitions* du point d'accès (*jaxrs:server*).
La propriété en question permet de choisir le type d'informations saisissables (jeton JWT et/ou HTTP Basic) via l'interface de *Swagger-ui*. Si aucune propriété n'est fixée alors, par défaut, l'interface propose un champ pour saisir un jeton JWT.

Pour déclarer de façon explicite les informations en lien avec un jeton JWT, il faut ajouter une entrée de type "jwt_key" avec la classe d'implémentation *io. swagger.models.auth.ApiKeyAuthDefinition* et les propriétés *name* et *in* prenant respectivement les valeurs *Authorization* et *HEADER*.
Exemple :

```
<bean id="swagger2Feature" class="org.apache.cxf.jaxrs.swagger.Swagger2Feature" lazy-init="true">
<!-- ... -->
      <property name="resourcePackage" value="com.hardis.adelia.webservice,hardis.fr" />
<!-- ... -->
</bean>


<jaxrs:server id="RestAdelia" address="/" transportId="http://cxf.apache.org/transports/http">
        <jaxrs:features>
            <ref bean="swagger2Feature" />
        </jaxrs:features>
    <jaxrs:properties>
                <entry key="SwaggerSecurityDefinitions">
                <map>
                    <entry key="jwt_Key">
                        <bean class="io.swagger.models.auth.ApiKeyAuthDefinition">
                            <property name="name" value="Authorization"/>
                                        <property name="in">
                                <value type="io.swagger.models.auth.In">HEADER</value>
                                        </property>
                        </bean>
                    </entry>
                </map>
         </entry>
        </jaxrs:properties>
</jaxrs:server>
```

La configuration proposée ci-dessus a pour effet de modifier le *swagger.json* produit ; Swagger-ui adapte alors son interface en proposant un bouton [Authorize] (et des pictogrammes "cadenas" pour chaque méthode) ouvrant une fenêtre dans laquelle il est possible de saisir le jeton JWT. Par la suite ce jeton sera ajouté au header *Authorization* des requêtes d'accès aux ressources.



Accès à une ressource sécurisée par une authentification HTTP Basic

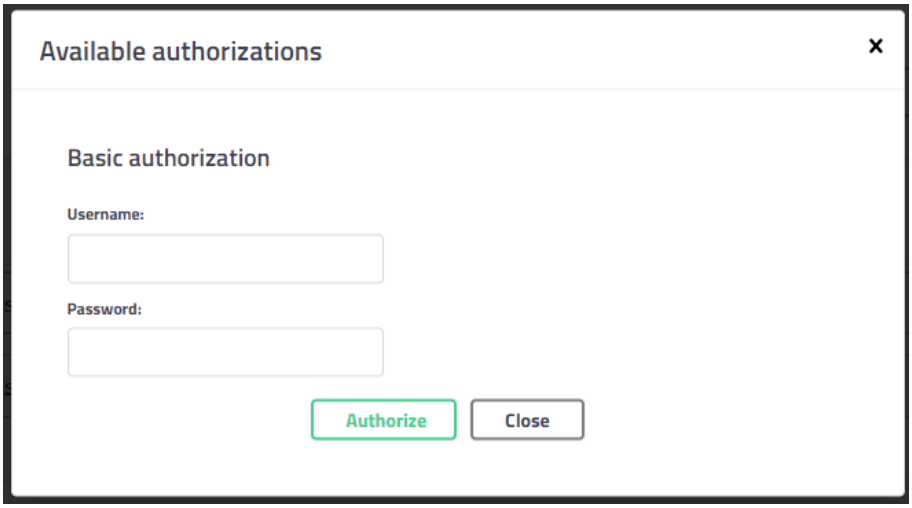
Pour que *Swagger-ui* offre la possibilité de saisir des informations d'authentification, il faut ajouter le package *com.hardis.adelia.webservice* à la propriété *r esourcePackage* du bean *Swagger2Feature*.
Pour choisir le type d'informations saisissables (jeton JWT et/ou HTTP Basic), il faut ajouter une propriété *SwaggerSecurityDefinitions* au *jaxrs:server*.

Pour des informations en lien avec une authentification HTTP Basic, il faut ajouter une entrée de type "basic_key" avec la classe d'implémentation *io. swagger.models.auth.BasicAuthDefinition*

Exemple :

```
<bean id="swagger2Feature" class="org.apache.cxf.jaxrs.swagger.Swagger2Feature" lazy-init="true">
<!-- ... -->
      <property name="resourcePackage" value="com.hardis.adelia.webservice,hardis.fr" />
<!-- ... -->
</bean>


<jaxrs:server id="RestAdelia" address="/" transportId="http://cxf.apache.org/transports/http">
        <jaxrs:features>
            <ref bean="swagger2Feature" />
        </jaxrs:features>
   <jaxrs:properties>
                <entry key="SwaggerSecurityDefinitions">
                <map>
                    <entry key="basic_Key">
                                        <bean class="io.swagger.models.auth.BasicAuthDefinition"
/>
                                </entry>
                   </map>
         </entry>
        </jaxrs:properties>
</jaxrs:server>
```

La configuration proposée ci-dessus a pour effet de modifier le *swagger.json* produit ; *Swagger-ui* adapte alors son interface en proposant un bouton [Authorize] (et des pictogrammes "cadenas" pour chaque méthode) ouvrant une fenêtre dans laquelle il est possible de saisir un *Username* et un *Password* pour alimenter le header *Authorization* des requêtes d'accès aux ressources.



Points d'accès multiples avec des contextes Swagger spécifiques
Par défaut Swagger (via la feature Swagger2Feature) gère un contexte unique. Le document swagger.json produit prend en compte l'ensemble des ressources des différents points d'accès.
Il est possible d'associer un contexte Swagger spécifique à un point d'accès à l'aide des propriétés *usePathBasedConfig* et *scan* du bean *Swagger2Feature*.

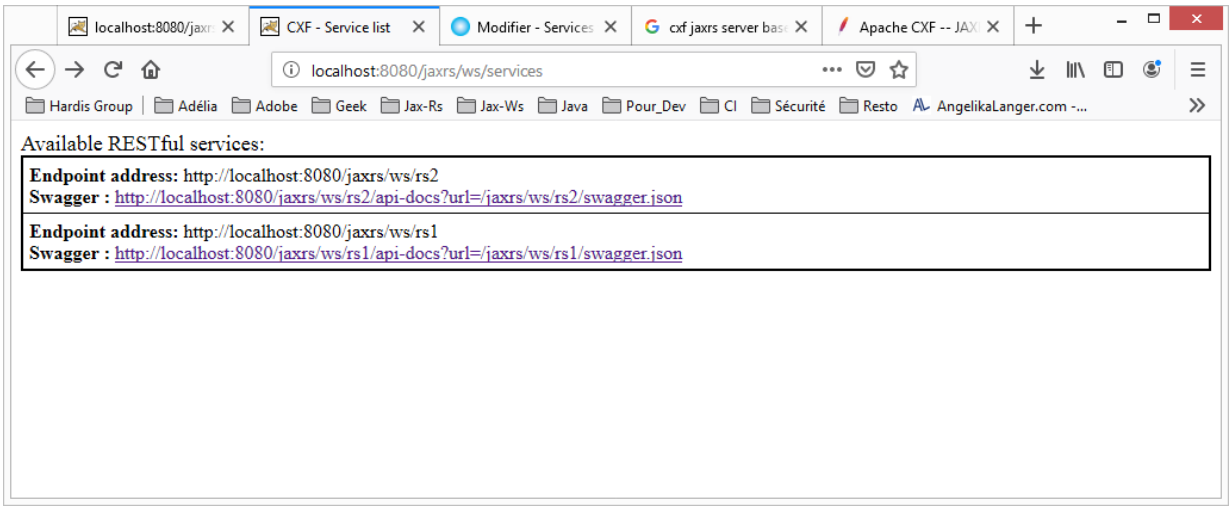
Exemple

beans.xml

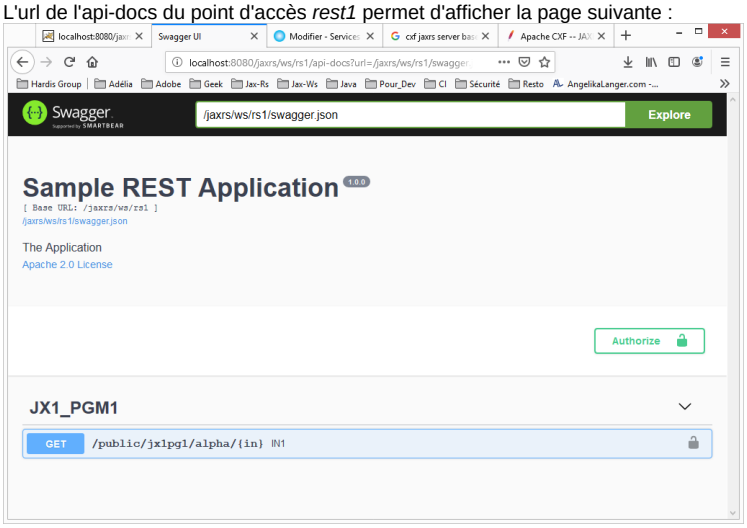
```
<bean id="swagger2Feature" class="org.apache.cxf.jaxrs.swagger.Swagger2Feature" lazy-init="true">
    <property name="supportSwaggerUi" value="true"/>
    <property name="usePathBasedConfig" value="true"/>
    <property name="scan" value="false"/>
        <!-- ...  -->
  </bean>


<jaxrs:server id="rest1" address="/rs1" transportId="http://cxf.apache.org/transports/http" basePackages="com.
hardis.adelia.webservice,hardis.jaxrs1">
  <jaxrs:features>
      <ref bean="swagger2Feature" />
  </jaxrs:features>
</jaxrs:server>

<jaxrs:server id="rest2" address="/rs2" transportId="http://cxf.apache.org/transports/http" basePackages="com.
hardis.adelia.webservice,hardis.jaxrs2">
  <jaxrs:features>
    <ref bean="swagger2Feature" />
  </jaxrs:features>
</jaxrs:server>
```

Le fichier *beans.xml* ci-dessus déclare 2 points d'accès (*rest1* et *rest2*). Les *<jaxrs:server>*utilisent l'attrbut *basePackages* pour détecter automatiquement les ressources et providers.
Remarque : Dans les cas précédents; la détection était réalisée à l'aide de la balise *<context:component-scan base-package="package1, package2, ..."/>*.
Chaque point d'accès fait référence à la feature *Swagger2Feature* fixant la propriété *usePathBasedConfig* à *true* et la propriété *scan* à *false*.

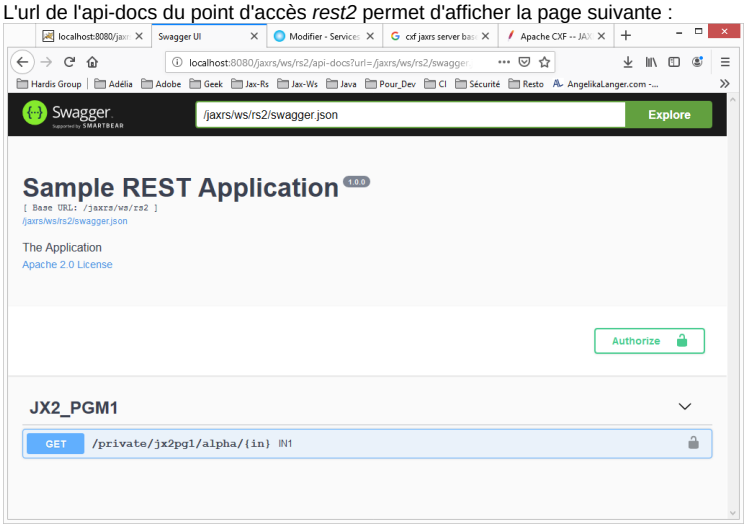
Les points d'accès exposés sont les suivants :



L'url de l'api-docs du point d'accès *rest1* permet d'afficher la page suivante :

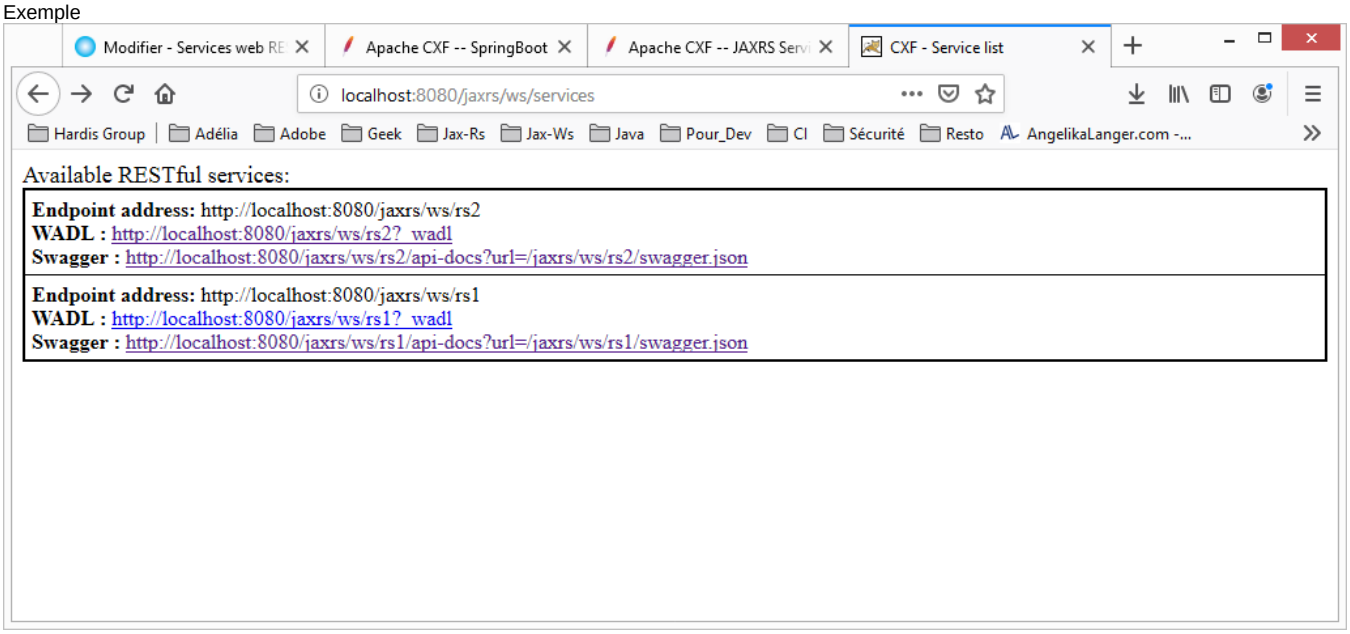


L'url de l'api-docs du point d'accès *rest2* permet d'afficher la page suivante :



WADL
Pour obtenir une description des services au format *WADL*, il faut déployer la librairie **cxf-rt-rs-service-description-3.x.y.jar** dans WEB-INF/lib de la webapp.
L'URL d'accès aux endpoints proposera alors des liens pour récupérer la description des services au format WADL.

Exemple



L'URL d'accès au WADL au point d'accès rs1 permet de récupérer le document suivant :

rs1.xml

```
<application xmlns="http://wadl.dev.java.net/2009/02" xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <grammars/>
    <resources base="http://localhost:8080/jaxrs/ws/rs1">
        <resource path="/public/jx1pg1">
            <resource path="/alpha/{in}">
                <param name="in" style="template" type="xs:string"/>
                <method name="GET">
                    <request/>
                    <response>
                        <representation mediaType="text/plain"/>
                    </response>
                </method>
            </resource>
        </resource>
        <resource path="/swagger.{type:json|yaml}">
            <param name="type" style="template" type="xs:string"/>
            <method name="GET">
                <request/>
                <response>
                    <representation mediaType="application/json"/>
                    <representation mediaType="application/yaml"/>
                </response>
            </method>
        </resource>
        <resource path="/api-docs">
            <resource path="/{resource:.*}">
                <param name="resource" style="template" type="xs:string"/>
                <method name="GET">
                    <request/>
                    <response>
                        <representation mediaType="*/*"/>
                    </response>
                </method>
            </resource>
        </resource>
    </resources>
</application>
```

(V14 PTF01)

La version V14 PTF01 intègre les librairies :

- CXF 3.1.18
- swagger.core 1.5.22
- swagger.ui 3.20.9

(V14 PTF02)

La version V14 PTF02 intègre les librairies :

- CXF 3.3.4
- swagger.core 1.6.0
- swagger.ui 3.24.3

(V14 PTF04)

La version V14 PTF04 intègre les librairies :

- CXF 3.3.8
- swagger.core 1.6.2
- swagger.ui 3.38.0

(V14 PTF07)

La version V14 PTF07 intègre les librairies :

- CXF 3.3.13
- swagger.core 1.6.6
- swagger.ui 3.52.5

(V14 PTF08 Fix02)

Evolution **_swag_response**
Possibilité de spécifier :

- le nom d'une variable résultat pour préciser le schéma/modèle de la réponse (Note : par défaut c'est celui de la variable paramètre portant la définition *o[Content]*) pour le code HTTP donné
- *NONE pour ne pas associer de schéma/modèle pour le code HTTP donné

SW_CONFIGURER *OPERATION _WS_REST_SWAG_RESPONSE 'ResultVarName:CodeHttp:Description' le schéma/modèle est celui ResultVarName
SW_CONFIGURER *OPERATION _WS_REST_SWAG_RESPONSE '*NONE:CodeHttp:Description' aucun schéma/modèle
SW_CONFIGURER *OPERATION _WS_REST_SWAG_RESPONSE 'CodeHttp:Description' le schéma/modèle est celui de la variable paramètre définie en o[Content]

(V14 PTF09)

La version V14 PTF09 intègre les librairies :

- CXF 3.4.10
- swagger.core 2.2.10
- swagger.ui 4.18.2

Abandon de la spécification Swagger2(OpenApi2) au profit d'OpenApi3.
Les beans *Swagger2Feature, Swagger2FeatureExtended, SwaggJaxbInit* sont dépréciés et doivent être supprimés du fichier *beans.xml*.
Il faut désormais utiliser l'unique bean *com.hardis.adelia.webservice.OpenApiFeatureExtended*. Ce composant hérite du composant OpenApiFeature dont les propriétés sont détaillées ici : https://cxf.apache.org/docs/openapifeature.html

Exemple :

beans.xml

```
<!-- Liste des ressources REST : définition des packages à scanner -->
<!-- Note : le package com.hardis.adelia.webservice doit toujours figurer dans la liste : -->
<!-- - Permet de définir le serveur par défaut à partir du basePath -->
<!-- - Permet de définir un SecurityScheme (JWT) pour assurer une authentification via un jeton JWT -->
<context:component-scan base-package="com.hardis.adelia.webservice,local.test"/>

<!-- Configuration du bean openApiFeature -->
<!-- basePath : <ContextRoot>/<CXFServletMapping> -->
<bean id="openApiFeature" class="com.hardis.adelia.webservice.OpenApiFeatureExtented" lazy-init="true">
        <property name="basePath" value="/jaxrs/ws" />
         <property name="swguicfg_core_queryConfigEnabled" value="false" />
    <property name="title" value="REST API - OPENAPIFEATURE" />
           <property name="description" value="REST Api 2.0 OpenApiFeature description list" />
           <property name="version" value="2.0" />
          <property name="supportSwaggerUi" value="true"/>
</bean>

<jaxrs:server id="rest" address="/" transportId="http://cxf.apache.org/transports/http">
        <jaxrs:features>
                <!-- déclaration de l'utilisation de l'openApiFeature -->
                <ref bean="openApiFeature" />
        </jaxrs:features>
</jaxrs:server>
```

IMPORTANT :

- **OpenApi3** a son propre jeu d'annotations aussi est-il nécessaire de regénérer les programmes de services web REST et les classes associées pour bénéficier de la documentation SwaggerUi.

- En cas d'utiliation combinée de **swaggerUi** et de l'authentification **JWT** - il faut ajouter à l'attribut *jwtSwaggerURI* du bean *jwtTokenConfiguration* la séquence */ws/openapi**
  Exemple :

  Beans.xml - jwtTokenConfiguration - OpenApi

```
<bean id="jwtTokenConfiguration" class="com.hardis.adelia.webservice.JwtTokenConfiguration">
        <property name="jwtBasePath" value="/ws/*" />
        <property name="jwtSwaggerURI" value="/ws/api-docs;/ws/swagger*;/ws/openapi*"/>
        ...
</bean>
```

- La version embarquée de swaggerui (swagger-ui-4.18.2.jar) ne permet plus par défaut de passer des éléments de configuration de swaggerUI via des paramètres d'url. Seul le paramètre *url* est accepté sous condition de fixer la propriété *swguicfg_core_queryConfigEnabled* à la valeur *false* dans le bean *openApiFeature*.
  Exemple :
  http://localhost:8081/adelrestapp/ws/api-docs?**url=/adelrestapp/ws/openapi.json**

  Pour passer des éléments de configuration il faut modifer le fichier ***swagger-initializer.js*** *(swagger-ui-4.18.2.jar-->/META-INF/resources/webjars /swagger-ui/4.18.2),*
  - soit en ajoutant au *SwaggerUIBundle* l'attribut *queryConfigEnabled:true* puis en passant les paramètres via l'URL
    Exemple :
    window.ui = SwaggerUIBundle({
    url: "https://petstore.swagger.io/v2/swagger.json",
    dom_id: '#swagger-ui',
    deepLinking: true,
    queryConfigEnabled:true,
    presets: [...

    http://localhost:8081/adelrestapp/ws/api-docs?**url=/adelrestapp/ws/openapi.json&docExpansion=none&tagsSorter=alpha&displayRequestDuration=true&filter=true&syntaxHighlight=false&defaultModelsExpandDepth=-1**
  - soit en ajoutant directement au *SwaggerUIBundle* l'ensemble des attributs voulus
    Exemple:
    window.ui = SwaggerUIBundle({
    url: "https://petstore.swagger.io/v2/swagger.json",
    dom_id: '#swagger-ui',
    deepLinking: true,
    defaultModelsExpandDepth:-1,
    displayRequestDuration:true,
    docExpansion:'none',
    filter:true,
    tagsSorter:'alpha',
    syntaxHighlight:false,
    presets: [...

- La propriété **resourcePackages** (permettant de filtrer via des noms des packages les apis affichées par swagger-ui) doit être utilisée conjointement avec la propriété **scan.** Pour activer le filtrage il faut désormais fixer la propriété **scan** à la valeur *false ;* dans le cas contraire tous les packages listés dans l'attribut *base-package* de l'élément **component-scan** sont systématiquement pris en compte.

- La présence du package *com.hardis.adelia.webservice* dans l'attribut *base-package* de l'élément *component-scan* permet d'enrichir automatiquement le fichier *openapi.json* construit - notamment pour donner la possibilité d'une authentification via un jeton JWT.

  <context:component-scan *base-package*="**com.hardis.adelia.webservice**,..."/>

  Il est possible de définir ses propres modes d'authentification (via des SecurityScheme) en ajoutant des propriétés au *jaxrs:server*
  Exemple :

```
<bean id="openApiFeature" class="com.hardis.adelia.webservice.OpenApiFeatureExtended" lazy-init="true">
<!-- ... -->
</bean>

<jaxrs:server id="RestAdelia" address="/" transportId="http://cxf.apache.org/transports/http">
        <jaxrs:features>
            <ref bean="openApiFeature" />
        </jaxrs:features>
    <jaxrs:properties>
                <entry key="SwaggerSecurityDefinitions">
                <map>
                           <entry key="jwt_Key">
                             <bean class="io.swagger.v3.oas.models.security.SecurityScheme" >
                                <property name="name" value="jwt_Key"/>
                                <property name="type">
                                          <value type="io.swagger.v3.oas.models.security.SecurityScheme.Type"
>HTTP</value>
                                </property>
                                               <property name="scheme" value="bearer"/>
                                                <property name="bearerFormat" value="JWT"/>
                            </bean>
                    </entry>
                    <entry key="basic_Key">
                              <bean class="io.swagger.v3.oas.models.security.SecurityScheme" >
                                               <property name="name" value="basic_Key"/>
                                               <property name="type">
                                        <value type="io.swagger.v3.oas.models.security.SecurityScheme.Type"
>HTTP</value>
                                </property>
                                               <property name="scheme" value="basic"/>
                                          </bean>
                    </entry>
           </map>
       </entry>
        </jaxrs:properties>
</jaxrs:server>
```

Ajout d'annotations permettant à swagger-ui d'afficher la définition Adélia des champs alpha et numérique :
maxLength donne la longeur maximale (longueur Adélia) d'un champ alphanumérique
minimum et *maximum* donnent respectivement la borne inférieure et la borne supérieure d'un champ numérique selon sa définition Adélia.
Mise en garde : Swagger-UI n'affiche correctement que les entiers compris entre `-9007199254740991` et `9007199254740991`. Les entiers en dehors de cette plage ainsi que certains nombres décimaux peuvent faire l'objet d'un arrondissement.

(V14 PTF10)

La version V14 PTF010 intègre les librairies :

- CXF 3.4.10
- swagger.core 2.2.19
- swagger.ui 5.10.3
- springframework : 5.3.31