MODULE WS

WSDL et SOAP

1.

Description du service

Description d’un service

- **Objectif** :
  - Décrire d’un point de vue informatique ce que fait le service

Description d’un service

- La description d’un service doit comporter :
  - La Définition du service
  - Les types de données utilisés notamment dans le cas de types complexes
  - Les opérations utilisables
  - Le protocole utilisé pour le transport
  - L'adresse d'appel

Description d'un service

- Définition en **WSDL** (Web Service Description Language)
  - Métalangage basé sur XML permettant de décrire en détail le service web
  - WSDL est normalisé (W3C)
  - Fichier WSDL disponible sur le serveur pour permettre aux clients de s'informer sur les services web disponibles

Description d'un service

- Ça ressemble à quoi un fichier **WSDL** ?
  - Un bloc de **définition**

```
<definitions name = "HelloService"
  targetNamespace = "http://www.m2i.fr/wsdl/HelloService.wsdl"
  xmlns = "http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap = "http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns = "http://www.m2i.fr/wsdl/HelloService.wsdl"
  xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
```

Description d'un service

- Ça ressemble à quoi un fichier **WSDL** ?
  - Des types de **données**

```
<message name = "SayHelloRequest">
    <part name = "firstName" type = "xsd:string"/>
  </message>

  <message name = "SayHelloResponse">
    <part name = "greeting" type = "xsd:string"/>
  </message>
```

Description d'un service

- Ça ressemble à quoi un fichier **WSDL** ?
  - Des **opérations**

```
<portType name = "Hello_PortType">
    <operation name = "sayHello">
      <input message = "tns:SayHelloRequest"/>
      <output message = "tns:SayHelloResponse"/>
    </operation>
  </portType>
```

Description d'un service

- Ça ressemble à quoi un fichier **WSDL** ?
  - Le **binding**

```
<binding name = "Hello_Binding" type = "tns:Hello_PortType">
    <soap:binding style = "rpc"
      transport = "http://schemas.xmlsoap.org/soap/http"/>
    <operation name = "sayHello">
      <soap:operation soapAction = "sayHello"/>
```

Description d'un service

- Ça ressemble à quoi un fichier **WSDL** ?
  - Le **binding**

```
<input>
  <soap:body
    encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"
    namespace = "urn:examples:helloservice"
    use = "encoded"/>
</input>
```

Description d'un service

- Ça ressemble à quoi un fichier **WSDL** ?
  - Le **binding**

```
      <output>
        <soap:body
          encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"
          namespace = "urn:examples:helloservice"
          use = "encoded"/>
      </output>
    </operation>
  </binding>
```

Description d'un service

- Ça ressemble à quoi un fichier **WSDL** ?
  - Le **point d'accès** au service

```
<service name = "Hello_Service">
   <documentation>WSDL File for HelloService</documentation>
   <port binding = "tns:Hello_Binding" name = "Hello_Port">
      <soap:address
         location = "http://www.examples.com/SayHello/" />
   </port>
</service>
```

2.

Protocoles et Bindings

Liaison au protocole (binding)

- Un message SOAP est lié au protocole sur lequel il se déploie
- SOAP se déploie au-dessus de HTTP mais cela ne signifie pas qu'il remplace ou se substitue à quoi que ce soit dans la sémantique du protocole, mais plutôt qu'il en hérite
- La documentation se concentre sur HTTP (protocole le plus utilisé)
- Il est possible d'utiliser SOAP avec d'autres protocoles que HTTP (SMTP, FTP, …)

Liaison au protocole (binding)

- La définition d'une liaison HTTP concerne trois parties :
  - La requête HTTP
  - La réponse HTTP
  - Le cadre d'extension HTTP
- Dans tous les cas, le *media type* "text/xml" doit être utilisé lors de l'encapsulation de messages SOAP dans les échanges HTTP

Liaison au protocole (binding)

- Requête HTTP, grande majorité des liaisons avec la méthode de requête HTTP POST

- En-tête de requête HTTP SOAPAction (SOAPAction HTTP request header) peut être utilisé afin d'indiquer la cible de requête SOAP HTTP

- La valeur que doit renseigner un tel champ représente l'URI de la cible

Liaison au protocole (binding)

- Une valeur de chaîne vide signifie que la cible du message SOAP est fournie par l'URI de la requête HTTP

- L'absence de valeur indique qu'il n'y a pas de cible explicite du message

Liaison au protocole (binding)

- Exemple d'en-tête de requête SOAP HTTP

```
POST /soap/servlet/rpcrouter HTTP/1.1
Host: localhost
Content-Type: text/xml; charset="utf-8"
Content-Length: 345
SOAPAction: "http://electrocommerce.org/">
<env:Envelope xmlns:env="http://www.w3.org/2001/06/soap-envelope">
...
</env:Envelope>
```

Liaison au protocole (binding)

- Réponse HTTP : SOAP suit la sémantique des codes de statut HTTP pour communiquer des informations de statut sur HTTP :
  - Code de statut 2xx = requête du client incluant le composant SOAP a été reçu avec succès, correctement interprété, accepté, …
  - Si une erreur se produit pendant le traitement de la requête, le serveur SOAP HTTP doit renvoyer une réponse HTTP 500 et inclure un élément SOAP **fault** dans le message SOAP de retour

Liaison au protocole (binding)

- Exemple de réponse SOAP :

```
HTTP/1.1 200 OK
Content-Type: text/xml charset="utf-8"
Content-Length: 323
<env:Envelope
xmlns:env="http://www.w3.org/2001/06/soap-envelope">
...
</env:Envelope>
```

3.

Structure d’un message

Structure et propriétés du WSDL

- Utilisation des éléments principaux du XML
  - types : définition des types de messages
  - messages : description des données à transmettre
  - interface : opération abstraite décrivant le communication entre le serveur et le client
  - binding : informations sur le protocole de transport utilisé
  - endpoint : informations sur l'interface de communication (URI)
  - service : point d'accès du service web

Structure et propriétés du WSDL

- Exemple de fichier WSDL
  - https://graphical.weather.gov/xml/SOAP_server/ndfdXMLserver.php?wsdl

4.

SoapUI

SoapUI

- **SoapUI** est une application open source permettant le test de services web

- https://www.soapui.org/

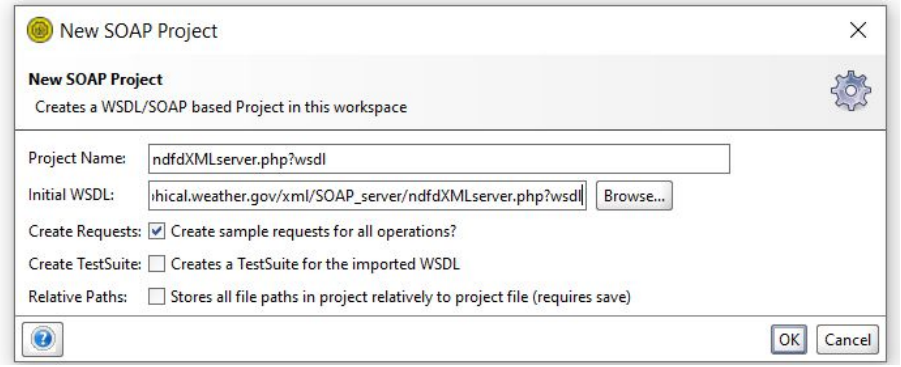
- https://www.soapui.org/downloads/soapui/

SoapUI

- **Exercice 1 :** inspecter et invoquer un service web
  - Construire un projet SoapUI à partir de la description WSDL d'un service web SOAP
  - Invoquer des opérations d'un service web SOAP
    - https://graphical.weather.gov/xml/SOAP_server/ndfdXMLserver.php?wsdl
    - https://graphical.weather.gov/xml/#use_it

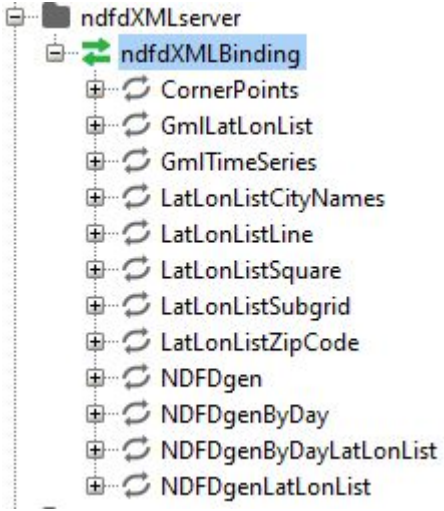
SoapUI

- **Exercice 1 :** inspecter et invoquer un service web
  - Construire un projet SoapUI à partir de la description WSDL d'un service web SOAP
    - File □ New SOAP Project



SoapUI

- **Exercice 1 :** inspecter et invoquer un service web
  - Construire un projet SoapUI à partir de la description WSDL d'un service web SOAP
    - Liste des opérations :



SoapUI

- **Exercice 1 :** inspecter et invoquer un service web
  - Invoquer des opérations d'un service web SOAP
    - Sélectionner l'opération ***LatLonListZipCode***
    - Ouvrir l'objet *Request 1*
    - Saisir la valeur « 10001 » dans le paramètre ZipCodeList

```
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.
   <soapenv:Header/>
   <soapenv:Body>
      <ndf:LatLonListZipCode soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <zipCodeList xsi:type="xsd:string">10001</zipCodeList>
      </ndf:LatLonListZipCode>
   </soapenv:Body>
</soapenv:Envelope>
```

SoapUI

- **Exercice 1 :** inspecter et invoquer un service web
  - Invoquer des opérations d'un service web SOAP
    - Exécuter la requête et vérifier le résultat
      - 40.7198,-73.993

SoapUI

- **Exercice 1 :** inspecter et invoquer un service web
  - Invoquer des opérations d'un service web SOAP
    - Sélectionner l'opération ***NDFDgenByDay*** et ouvrir l'objet *Request 1*
    - Saisir les valeurs suivantes :

```
<latitude xsi:type="xsd:decimal">40.7198</latitude>
<longitude xsi:type="xsd:decimal">-73.993</longitude>
<startDate xsi:type="xsd:date">2020-12-08</startDate>
<numDays xsi:type="xsd:integer">1</numDays>
<Unit xsi:type="xsd:string">m</Unit>
<format xsi:type="xsd:string">24 hourly</format>
```

SoapUI

- **Exercice 1 :** inspecter et invoquer un service web
  - Invoquer des opérations d'un service web SOAP
    - Exécuter la requête et vérifier le résultat
      - Prévision météo pour la journée et la ville demandées

```
<temperature type="maximum" units="Celsius" time-layout="k-p24
  <name>Daily Maximum Temperature</name>
  <value>4</value>
</temperature>
<temperature type="minimum" units="Celsius" time-layout="k-p24
  <name>Daily Minimum Temperature</name>
  <value>-2</value>
</temperature>
<probability-of-precipitation type="12 hour" units="percent" t
  <name>12 Hourly Probability of Precipitation</name>
  <value>2</value>
  <value>2</value>
</probability-of-precipitation>
```

SoapUI

- **Exercice 1 :** inspecter et invoquer un service web
  - Invoquer des opérations d'un service web SOAP
    - Sélectionner l'opération ***NDFDgenByDayLatLonList*** et ouvrir l'objet *Request 1*
    - L'objectif est de récupérer les prévisions météo pour la ville de New York (on a déjà les coordonnées) et pour la ville de Beverly Hills (zipcode = 90210)
    - Le paramètre ListLatLon prend une liste de coordonnées
      - Latitude et Longitude séparées par une virgule
      - Chaque couple de coordonnées séparée par un espace

SoapUI

- **Exercice 1 :** inspecter et invoquer un service web
  - Invoquer des opérations d'un service web SOAP
    - Exécuter la requête et vérifier le résultat
      - Quelle température max à Beverly Hills ?

5.

Contenu d’un message

Contenu d’un message SOAP

- Enveloppe (SOAP Envelope) définit le contenu du message
  - Header (SOAP Header) optionnel (par exemple pour les autorisations)
  - Body (SOAP Body) informations sur la requête ou la réponse
- Attachements optionnels

Contenu d’un message SOAP

- Exemple de message :

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
        <env:Header>
                <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
                 <n:priority>1</n:priority>  </n:alertcontrol>
        </env:Header>
        <env:Body>
                <m:alert xmlns:m="http://example.org/alert">
                 <m:msg>Example message</m:msg>
                </m:alert>
        </env:Body>
</env:Envelope>
```

6.

Code first et Contract First

Code First

- Approche traditionnelle qui consiste à développer le code du service **en premier**
  - Etape 1 : Développement du service
  - Etape 2 : Génération de la documentation du service

Contract First

- Approche différente qui consiste à concevoir le contrat du service **en premier** (avant d'écrire le code)
  - Etape 1 : conception du service
  - Etape 2 : Génération du code à partir de la description du service

Quelle approche choisir ?

- Approche **Contract First**
  - Si l'expérience développeur (DX) compte
  - Pour fournir des API stratégiques
  - Pour assurer une bonne communication

Quelle approche choisir ?

- Approche **Code First**
  - Si besoin d'une livraison rapide
  - Pour développer des API internes

7.

Génération d’artefacts

Génération d'artefacts à partir du WSDL

- Outils disponibles pour générer les artefacts à partir du fichier WSDL (par exemple en Java)

- Artefacts générés :

  - Interface SEI (Service endpoint interface)
  - Classe de service
  - Classe d'exception mappée à partir de la classe wsdl:fault
  - Bean de réponse dérivé de la réponse wsdl:message
  - Valeurs générées de type JAXB (Java Architecture for XML Binding) (classes Java mappées à partir de types de schéma XML)

8.

Les normes d’interopérabilité WS-I

La norme WS-I

- WS-I = Web Services Interoperability
- A l'origine : consortium industriel pour la promotion de l'interopérabilité entre plateformes
  - Rédaction des spécifications des services web WS-*
- Aujourd'hui : norme OASIS (Organization for the Advencement of Structures Information Standards)

Les profils WS-I

- Un profil = un ensemble de contraintes rattachées à la rédaction d'une spécification
  - WS-I Basic Profile
  - WS-I Basic Security Profile
  - Simple Soap Binding Profile

Les profils WS-I

- WS-I Basic Profile
  - Profil de base WS-I
  - Le profil utilise le langage de description des services Web (WSDL) pour activer la description des services sous forme d'ensembles de points de terminaison fonctionnant sur les messages
  - WSI-BP définit un ensemble beaucoup plus restreint de services valides que le schéma WSDL ou SOAP complet
  - WSI-BP restreint généralement la spécification SOAP

Les profils WS-I

- WS-I Basic Security Profile
  - Profil axé sur la sécurité

Les profils WS-I

- Simple SOAP Binding Profile
  - Profil de support pour le profil de base WS-I
  - Définit la manière dont les documents WSDL doivent lier les opérations à un protocole de transport SOAP spécifique

MODULE WS

JAX-WS

JAX-WS

- JAX-WS = API normalisée permettant de créer et de consommer des services Web SOAP en Java
- SOAP est lourd en XML, il est donc préférable de l'utiliser avec des outils / frameworks
- JAX-WS est un framework qui simplifie l'utilisation de SOAP
- JAX-WS fait partie du standard Java

1.

Annotations standards

Annotations JAX-WS

- @WebService
  - Marque une classe Java comme implémentant une interface de service web ou un service web
- @WebMethod
  - Indique une opération du service web

Annotations JAX-WS

- @WebParam
  - Mapping d’un paramètre vers le WSDL
- @WebResult
  - Mapping d’une valeur de retour vers le WSDL

2.

Implémentations : Metro, CXF

Implémentations JAX-WS

- Pour pouvoir utiliser la spécification JAX-WS, il faut une implémentation !

- Différentes implémentations sont disponibles :
  - Metro : projet GlassFish
  - CXF : projet Apache

Implémentation Metro

- **Metro**
  - **com.sun.xml.ws:jaxws-ri** : implémentation JAX-WS proposée par Metro

Implémentation Metro

- **Metro**
  - **Génération automatique du wsdl à l'exécution**
  - Possibilité de générer physiquement le fichier wsdl avec l'utilitaire fourni par Metro : **wsgen**
  - Possibilité de générer les classes Java à partir d'un wsdl : **wsimport**

Implémentation Metro

- **Mise en pratique**
  - Projet Maven avec Java
    - Module « wsdlToJava » : génération du code source à partir d'un WSDL
    - Module « contractFirst » : Implémentation à partir de « wsdlToJava »
    - Module « codeFirst » : création du code source puis génération du WSDL
    - Module « tomcatServer » : déploiement sur un serveur d'application Tomcat
    - Module « soapClient » : développement d'un client

Implémentation Metro

- **Mise en pratique**
  - Projet Maven avec Java
    - Création d'un projet Maven « chapeau » de type « pom »
    - Création d'un module Maven « wsdlToJava »
    - Création d'un module Maven « contractFirst »
    - Création d'un module Maven « codeFirst »
    - Création d'un module Maven « tomcatServer »

Implémentation Metro

- **Mise en pratique – Etape 1**
  - Module « wsdlToJava » : génération du code source à partir d'un WSDL
    - Récupération d'un fichier wsdl et de son xsd
    - Construction du projet avec génération du code à partir du wsdl

Implémentation Metro

- **Mise en pratique – Etape 2**
  - Module « contractFirst » : implémentation à partir du code généré
    - Dépendance « wsdlToJava »
    - Création d'une classe d'implémentation pour le web service

Implémentation Metro

- **Mise en pratique – Etape 2**
  - Module « contractFirst » : implémentation à partir du code généré
    - Création d'une classe Server pour déployer le web service en test
      - Endpoint de publication : http://localhost:9990/CustomerService

Implémentation Metro

- **Mise en pratique – Etape 2**
  - Module « contractFirst » : implémentation à partir du code généré
    - Test avec SoapUI
      - Vérifier la publication du WSDL : http://localhost:9990/CustomerService?wsdl
      - Créer un projet SoapUi et tester le service

SoapUI

- **Exercice SoapUI 2 :** construire une suite de tests simple (Test Suite)
  - New TestSuite
    - Création d'un TestCase avec possibilité d'ajouter des pas

Implémentation Metro

- **Mise en pratique – Etape 3**
  - Module « codeFirst » : création du code source puis génération du WSDL
  - Test avec SoapUI

Implémentation Metro

- **Mise en pratique – Etape 3**
  - Module « codeFirst » : création du code source puis génération du WSDL
    - Créer deux classes « model » : Order et Customer
    - Créer une classe d'implémentation pour les commandes
      - *OrderServiceImpl*
        - 1 méthode de récupération d'une commande par son identifiant retourne un objet de type Order
        - 1 méthode de construction d'un objet Order pour la réponse

Implémentation Metro

- **Mise en pratique – Etape 3**
  - Module « codeFirst » : création du code source puis génération du WSDL
    - Création d'une classe ServerApp pour déployer le web service en test
      - Endpoint de publication : http://localhost:9980/OrderService

Implémentation Metro

- **Mise en pratique – Etape 3**
  - Module « codeFirst » : création du code source puis génération du WSDL
    - Test avec SoapUI
      - Vérifier la publication du WSDL : http://localhost:9980/OrderService?wsdl
      - Créer un projet SoapUi et tester le service

Implémentation Metro

- **Mise en pratique – Etape 4**
  - Module « tomcatServer » : déploiement sur un serveur d'application Tomcat
  - Test avec SoapUI
  - **Prérequis** : Avoir un serveur Tomcat sur son poste

Implémentation Metro

- **Mise en pratique – Etape 4**
  - Module « tomcatServer » : déploiement sur un serveur d'application Tomcat
    - Ajout de dépendances au fichier pom.xml car Tomcat n'embarque pas nativement d'implémentation pour JAX-WS
    - Dépendance « wsdlToJava »

Implémentation Metro

- **Mise en pratique – Etape 4**
  - Module « tomcatServer » : déploiement sur un serveur d'application Tomcat
    - Création d'une classe d'implémentation pour le web service : *CustomerServiceImpl*
    - Fichier *web.xml* □ Descripteur de la servlet web service
    - Fichier *sun-jaxws.xml* □ Descripteur du *endpoint* pour notre service

Implémentation Metro

- **Mise en pratique – Etape 4**
  - Module « tomcatServer » : déploiement sur un serveur d'application Tomcat
    - Déploiement du war sur Tomcat
    - Accès au WSDL : http://localhost:8080/tomcatServer/CustomerWS?wsdl

Implémentation Metro

- **Mise en pratique – Etape 4**
  - Module « tomcatServer » : déploiement sur un serveur d'application Tomcat
    - Test avec SoapUI
      - Créer un projet SoapUI et tester le service

Implémentation Metro

- **Mise en pratique – Etape 5**
  - Module « soapClient » : développement d'un client
  - Test unitaire avec JUnit
  - Test avec SoapUI
    - Utilisation du serveur
    - Utilisation d'un bouchon généré avec SoapUI

Implémentation Metro

- **Mise en pratique – Etape 5**
  - Module « soapClient » : développement d'un client
    - Ajouts au fichier pom.xml
      - Junit
      - wsimport

Implémentation Metro

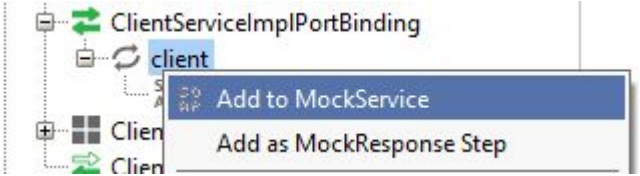
- **Mise en pratique – Etape 5**
  - Module « soapClient » : développement d'un client
    - Vérifier la génération du code
    - Créer une classe pour le client WS : *CustomerServiceClient*
      - Gérer l'URL du WS en attribut
      - Créer une méthode pour appeler le WS

Implémentation Metro

- **Mise en pratique – Etape 5**
  - Module « soapClient » : développement d'un client
    - Créer une classe de test avec Junit pour tester le client WS : *CustomerServiceClientTest*
      - Positionnement de l'URL du WS
      - Méthode de test pour client trouvé
      - Méthode de test pour cas où on ne passe pas de paramètre (fault)

SoapUI

- **Exercice 3 :** Simuler un service web SOAP à partir de sa description WSDL
  - A partir du projet SoapUI *CustomerService*
    - New SOAP MockService
    - Sélectionner le service ***customer***, faire clic droit, puis « Add to MockService »



SoapUI

- **Exercice 3 :** Simuler un service web SOAP à partir de sa description WSDL
  - A partir du projet SoapUI *CustomerService*
    - Possibilité de « bouchonner » la réponse
    - Clic droit sur la réponse créée, puis *Open Request*
      - *Regarder l'URL positionnée dans la requête*
    - Démarrer le bouchon : clic droit sur le *CustomerMockService* et « Start Minimized »

SoapUI

- **Exercice 3 :** Simuler un service web SOAP à partir de sa description WSDL
  - A partir du projet SoapUI *CustomerService*
    - Exécuter la requête et vérifier que la réponse « bouchon » est retournée !

SoapUI

- **Exercice 3 :** Simuler un service web SOAP à partir de sa description WSDL

  - Pour aller plus loin : mock dynamique !

  https://www.soapui.org/docs/soap-mocking/creating-dynamic-mockservices/

Implémentation CXF

- **CXF**
  - **org.apache.cxf:cxf-rt-frontend-jaxws** : implémentation JAX-WS proposée par CXF
  - **cxf-rt-transports-http-jetty** : définition des types Java nécessaires pour ne pas utiliser de conteneurs de servlet

Implémentation CXF

- **CXF**
  - **Génération automatique du wsdl à l'exécution**
  - Possibilité de générer physiquement le fichier wsdl avec l'utilitaire fourni par CXF : **java2WS**
  - Possibilité de générer les classes Java à partir d'un wsdl : **wsdl2java**

Implémentation CXF

- **Mise en pratique**
  - Projet Maven avec Java
  - Créer le WS pour récupérer une commande à l'image de ce qui a été fait avec Metro en « code first » (mais cette fois-ci avec CXF)
    - Lancement avec plugin Maven (exec-maven-plugin ; goal = exec:java)
    - Test avec SoapUI

Implémentation CXF

- **Mise en pratique**
  - Génération du WSDL
    - Utilisation de java2WS de CXF
    - Plugin Maven
      - https://cxf.apache.org/docs/maven-java2ws-plugin.html

3.

Sérialisation avec JAXB

JAXB

- **JAXB** = Java Architecture for XML Binding
  - Objectif : faciliter la manipulation d'un document XML en générant un ensemble de classes qui fournissent un niveau d'abstraction
  - JAXB fournit un outil qui analyse un schéma XML et génère à partir de ce dernier un ensemble de classes qui vont encapsuler les traitements de manipulation du document

JAXB

- **JAXB** = Java Architecture for XML Binding
  - JAX-WS utilise JAXB en interne comme couche de liaison pour convertir des objets Java vers et depuis XML

JAXB

- **JAXB** = Java Architecture for XML Binding
  - Les classes générées avec JAX-WS comporte des annotations JAXB
    - @XmlRootElement
    - @XmlElement
    - @XmlSeeAlso
    - @XmlType
    - @XmlAccessorType
    - @XmlSchemaType
    - @XmlEnumValue
    - @XmlEnum

4.

Asynchronisme

Asynchronisme

- **Problème de HTTP : protocole de « Best effort »**
  - Pas de qualité de service (on ne peut pas avoir la certitude qu'un message est arrivé à destination)
- Architecture **asynchrone** :
  - On reste sur un mécanisme requête/réponse en synchrone
  - Mais la réponse est un acquittement sur la réception de la requête ou simplement l'envoi de la requête

Asynchronisme

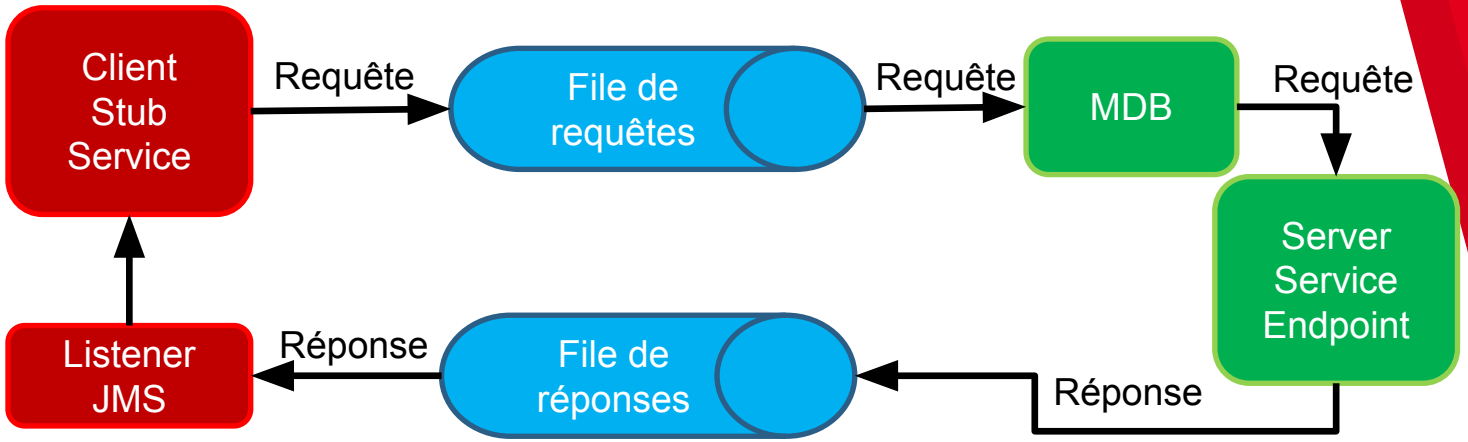
- Architecture **asynchrone** :
  - Scénarios possibles pour la suite :
    - 1/ Le client met à disposition un service web pour être notifié par le serveur de la disponibilité du résultat
      - Le client peut alors invoquer un service de récupération
    - 2/ Le client requête plus tard après un laps de temps, et avec par exemple un identifiant récupéré dans la réponse synchrone du premier appel
  - Peut s'appuyer sur un MOM (par exemple JMS)

5.

Services asynchrones avec JMS

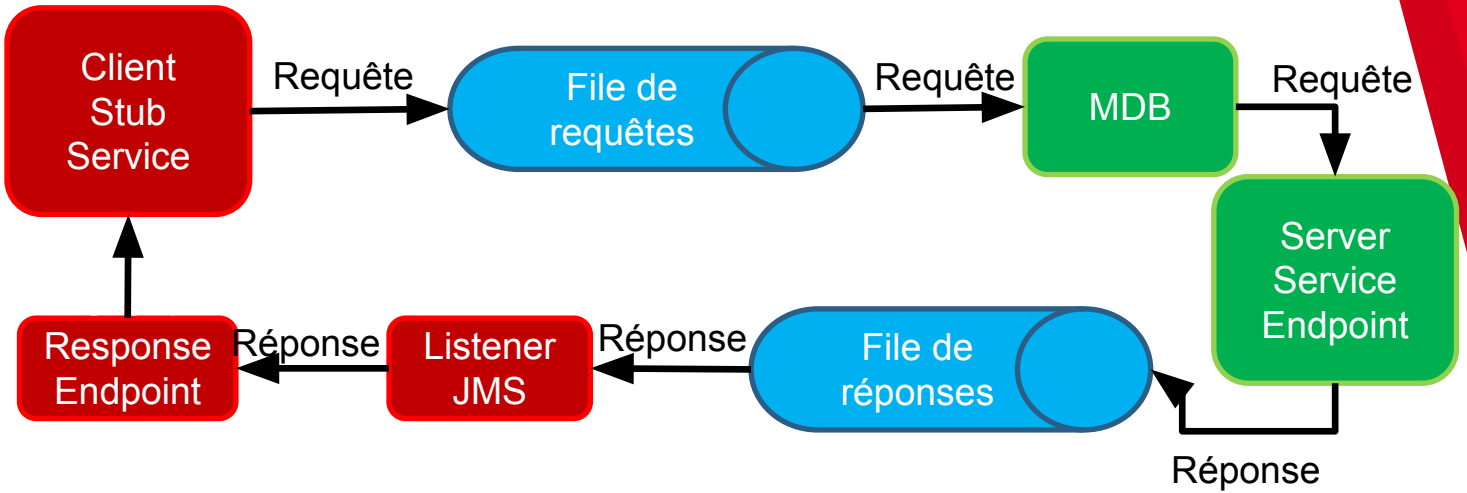
Service synchrone avec JMS

- **Flux d'échange synchrone avec JMS**



Service asynchrone avec JMS

- **Flux d'échange asynchrone avec JMS**



6.

Optimisation des échanges

Optimisation des échanges

- Il peut être intéressant d'optimiser le trafic SOAP au niveau du transport :
  - Des boîtiers accélérateurs peuvent être insérés devant les serveurs afin de compresser les flux XML dans un format binaire bien moins volumineux
  - Cette opération est triviale, puisque le boîtier se contente de détecter le support de gzip sur le client et compresse ensuite le flux XML sortant dans ce format
  - Ce type d'opération s'applique aussi bien aux flux XML que HTML

7.

MTOM et Fast InfoSet

MTOM

- **MTOM** = Message Transmission Optimization Mechanism (W3C)
- Méthode d'envoi de données binaires par services Web (pièces jointes, attachements)
- **MTOM** est habituellement utilisé avec XOP (XML-binary Optimized Packaging)

MTOM

- Efficacité de MTOM = taille des messages envoyés dans le flux
- Puisque SOAP utilise le XML, les données binaires doivent être encodées en texte, généralement du « Base64 », qui a la particularité d'augmenter la taille des données binaires de 33%
- MTOM propose un moyen d'envoyer ces données dans sa forme originale, ce qui évite une augmentation de la taille due à l'encodage du texte

MTOM

- **MTOM** doit être activé sur la classe d'implémentation du service
  - Annotation @MTOM
  - 2 paramètres :
    - **enabled** : activation de MTOM (booléen)
    - **threshold** : seuil pour codage XOP (entier >= 0)

MTOM

- **MTOM** peut être utilisé également côté client pour l'upload
  - Activation de MTOM sur le binding (*setMTOMEnabled(true)*)

```
// Activation de MTOM sur le client
BindingProvider bp = (BindingProvider) server;
SOAPBinding binding = (SOAPBinding) bp.getBinding();
binding.setMTOMEnabled(true);
```

MTOM

- **Test avec SoapUI**
  - https://www.soapui.org/docs/soap-and-wsdl/attachments/
  - Création d'un projet pour upload de fichier

Fast Infoset

- **Fast Infoset** = spécification d'un XML binarisé
  - Format de codage binaire pour l'ensemble des informations XML comme alternative au format de document XML
  - Sérialisation plus efficace que le XML basé sur du texte
- Fast Infoset vise à optimiser à la fois la taille du document et les performances de traitement

Fast Infoset

- **Fast Infoset** = spécification d'un XML binarisé
  - Propriété sur le client

```
getRequestContext().put(com.sun.xml.ws.client.ContentNegociation.P
ROPERTY, « pessimistic »);

Map<String, Object> ctxt =
((BindingProvider)proxy).getRequestContext();
ctxt.put(JAXWSProperties.CONTENT_NEGOTIATION_PROPERTY,
"pessimistic");
```