

# 09.Services Web

10 décembre 2025

## Développement web il3

### Service web / A.P.I.

HE-Arc (DGR) 2025

## Applications distribuées

- Motivation : répartir l'exécution sur plusieurs machines
  - Principe : Les composants/services communiquent par le réseau
  - Problèmes : Hétérogénéité systèmes, langages, ...
  - Solution : Protocole générique, abstraction différences
  - Exemples : RPC, RMI (java), CORBA, DCOM (MS)
- Utiliser les technologies du web, comme HTTP et XML :
  - indépendantes de la plateforme, éprouvées, largement utilisées
- Système distribué importance de l'architecture :
  - orientée ressource <sup>1</sup> : atome : ressource (donnée) : REST
  - orientée service <sup>2</sup> : atome : service (traitement) : RPC (SOAP)

### Service web / API

- 2 visions :
  - Utiliser les technos web pour développer des applis distribuées
  - Accès pour une application aux services offerts aux humains
- Service web = webapp pour une autre application :
  - Webapps : pour humains, via un navigateur (HTTP + HTML)

---

1. [https://en.wikipedia.org/wiki/Resource-oriented\\_architecture](https://en.wikipedia.org/wiki/Resource-oriented_architecture)

2. [https://fr.wikipedia.org/wiki/Architecture\\_orient%C3%A9e\\_services](https://fr.wikipedia.org/wiki/Architecture_orient%C3%A9e_services)

- Services web : aux autres applications (HTTP + XML/JSON)
- Exemples :
  - Applications distribuées<sup>3</sup> pour l'entreprise
  - Mashups<sup>4</sup> d'applications web (exemples<sup>5</sup>)
  - Applications Facebook, API Google<sup>6</sup>
  - IFTTT<sup>7</sup>
- Consommer un service web ≠ Créer un service web

## SOAP

- AVANT : Simple Object Access Protocol (obsolète)
- Evolution de XML-RPC, format XML d'envoi de messages
- Architecture Orientée Service (SOA)
- Indépendant du langage et de la plateforme
- Recommandation du w3c depuis 2003
- SOAP = abus de langage, service web WS-\* est plus exact
- Spécifications WS-\*<sup>8</sup> :
  - spécifications liées aux différents aspects des services web
  - pour déployer un WS : au minimum SOAP + WSDL + UDDI

## SOAP

- Structure d'un message SOAP
  - Enveloppe, Entête, Corps, Erreurs
- Squelette :

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Header> ... </soap:Header>
  <soap:Body> ...
    <soap:Fault> ... </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

- 
3. [https://upload.wikimedia.org/wikipedia/commons/3/3f/Concept\\_WS.jpg](https://upload.wikimedia.org/wikipedia/commons/3/3f/Concept_WS.jpg)
  4. [https://en.wikipedia.org/wiki/Mashup\\_\(web\\_application\\_hybrid\)](https://en.wikipedia.org/wiki/Mashup_(web_application_hybrid))
  5. <https://science.howstuffworks.com/innovation/repurposed-inventions/5-web-mashups.htm>
  6. <https://developers.google.com/apis-explorer/>
  7. <https://ifttt.com/>
  8. [https://en.wikipedia.org/wiki/List\\_of\\_web\\_service\\_specifications](https://en.wikipedia.org/wiki/List_of_web_service_specifications)

## SOAP

- Exemple <sup>9</sup> requête/réponse
- Créer un service web WS (SOAP) nécessite WSDL et UDDI :
  - SOAP : Echange de messages XML sur le réseau
  - WSDL : Web Service Description Language
  - UDDI : Universal Description, Discovery and Integration
- WSDL : Description des interfaces des web services
- UDDI : Découverte et inscription aux services web
  - annuaire d'informations sur les services web
  - annuaire d'interfaces de services web décrites en WSDL
- Tutorial WSDL/UDDI w3schools <sup>10</sup>

## REST : REpresentational State Transfer

- Style d'architecture sur lequel a été bâti le web
- Architecture Orientée Ressource (ROA)
- Chapitre 5 de la thèse <sup>11</sup> de Roy T. Fielding <sup>12</sup> (fr <sup>13</sup>), 2000
- Architecture REST est définie par 6 contraintes architecturales :
  - Client / Serveur
  - Sans état (pour le serveur)
  - Avec mise en cache
  - En couches
  - Avec code à la demande (optionnel : envoi de JS au client)
  - **Interface uniforme**

## REST : Interface Uniforme

- Parmi les 6 contraintes <sup>14</sup>, l'*interface uniforme* :
  - Les composants (clients, serveurs, proxies...) communiquent avec une interface générique (et non une API spécifique à chaque serveur) : HTTP (verbes, URI, codes de réponse, entêtes, ...)
- L'*interface uniforme* est, elle-même, définie par 4 contraintes
  - Identification des *ressources* (URI)

---

9. [https://www.w3schools.com/xml/xml\\_soap.asp](https://www.w3schools.com/xml/xml_soap.asp)

10. [https://www.w3schools.com/xml/xml\\_wsd.asp](https://www.w3schools.com/xml/xml_wsd.asp)

11. <https://web.archive.org/web/20251011130223/https://ics.uci.edu/~fielding/pubs/dissertation/top.htm>

12. [https://fr.wikipedia.org/wiki/Roy\\_Fielding](https://fr.wikipedia.org/wiki/Roy_Fielding)

13. <https://opikanoba.org/tr/fielding/rest/>

14. [https://fr.wikipedia.org/wiki/Representational\\_state\\_transfer](https://fr.wikipedia.org/wiki/Representational_state_transfer)

- Manipulation des *ressources* par des *représentations* (MIME types <sup>15</sup>)
- Messages autodescriptifs (entêtes/réponses HTTP)
- Hypermédia comme moteur de l'état de l'application (liens)
- *Ressource* : information ou moyen d'accès (abstrait)
  - ex. : météo du jour, adresse, article à un blog, ...
- *Représentation* : forme donnée à la ressource
  - ex. : text/html, application/pdf, image/png, application/rss+xml, audio/mpeg, ...

## REST concrètement

- Principes
  - Identifier les ressources avec des URI (noms)
  - Actions déterminées par des méthodes HTTP (verbes)
    - GET : READ (sûre)
    - POST : CREATE
    - PUT, PATCH : UPDATE (idempotente)
    - DELETE : DELETE (idempotente)
  - Les liens hypertextes permettent de changer d'état : navigation
  - Les types MIME déterminent la représentation de la ressource
- Rappel
  - Sûreté : Etat de la ressource (contenu) inchangé
  - Idempotence : plusieurs appels donnent le même résultat

## REST : exemples

- L'appel d'une ressource avec des verbes différents produira un résultat différent :

Effet	Route	Verbe	URI (ressource)	Description
C R	Index	GET	/blogs	Affiche la liste
	New	GET	/blog/new	Affiche formulaire création
	Create	POST	/blogs	Création en DB, puis redirection
U	Show	GET	/blogs/42	Affiche le blog 42
	Edit	GET	/blogs/42/edit	Formulaire édition blog 42
U	Update	PUT	/blogs/42	MAJ en DB blog 42
D	Destroy	DELETE	/blogs/42	Suppression ne DB blog 42

- Laravel, Django, Rails, ... sont RESTful!

---

15. <https://mimetype.io/all-types>

## Niveaux de maturité de Richardson <sup>16</sup>

- 0 : Plain Old Xml (POX)
  - Utilisation de HTTP pour faire du RPC
- 1 : Ressources
  - Ressources identifiées par URI
- 2 : Verbes HTTP
  - Respect des propriétés des verbes HTTP
- 3 : Hypertext As The Engine Of Application State (HATEOAS)
  - Les états suivants sont documentés dans la réponse (<link>)

## REpresentational State Transfer

- **Representational**
  - On ne manipule pas directement les ressources, mais des *représentations* de ces ressources
- **State**
  - L'état de l'application côté **client** qui *représente* une ressource
  - Changement d'état en suivant des liens hypertextes (requêtes HTTP)
- **Transfer**
  - Chaque réponse aux requêtes **transfère** une nouvelle représentation
- REST
  - Client/Serveur
  - L'état de l'application est gérée par le client
  - Ressources manipulées avec des représentations (MIME Types <sup>17</sup>)
  - Hypermedia (liens) pour changer d'état

## SOAP vs REST

- SOAP (WS-\*)
  - hérité du monde de l'entreprise
  - plus de code pour manipuler la requête et générer la réponse
  - plus flexible, extensible (namespace)
  - valider requêtes depuis WDSL
  - nécessité d'un framework (ex : nuSOAP en PHP)
- REST
  - hérité du web
  - plus facile et rapide à utiliser

---

16. <https://martinfowler.com/articles/richardsonMaturityModel.html>

17. <https://mimetype.io/all-types>

- plus lisible et plus compact
- maintenance plus facile
- meilleure tolérance aux pannes

## Pour aller plus loin...

- Références
  - SOAP <sup>18</sup>, WSDL <sup>19</sup>, REST <sup>20</sup>
  - Des services web RESTful <sup>21</sup>, Une apologie de REST <sup>22</sup> (recommandés)
  - REST et architectures orientées service <sup>23</sup>, Présentation ROA <sup>24</sup>
  - The RESTful cookbook <sup>25</sup>, How important is HATEOAS <sup>26</sup> (stack overflow)
- Exemples de services web :
  - Google <sup>27</sup>, Yahoo <sup>28</sup>, Flickr <sup>29</sup>, Twitter <sup>30</sup>, Spotify <sup>31</sup>...
  - APIary <sup>32</sup> : Aide au design d'une API REST
  - Tests : Postman, Hoppscotch <sup>33</sup>, Ping-API <sup>34</sup>, autres <sup>35</sup>
- GraphQL <sup>36</sup>
  - Requête envoyée à un endpoint unique, le client décrit la forme de la réponse (JSON)
  - Initiative Facebook, ex : Github <sup>37</sup>.

## Sources

- 
18. <https://www.w3.org/TR/soap/>
  19. <https://www.w3.org/2002/ws/desc/>
  20. <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
  21. <https://larlet.fr/david/biologeeek/archives/20070629-architecture-orientee-ressource-pour-faire-des-services-web-restful/>
  22. <https://web.archive.org/web/20160310205502/http://home.ccil.org/~cowan/restws.pdf>
  23. <https://www.figer.com/Publications/SOA.htm>
  24. <https://fr.slideshare.net/samijaber/symposium-dng-2008-roa>
  25. <https://restcookbook.com/>
  26. <https://stackoverflow.com/questions/20335967/how-useful-important-is-rest-hateoas-maturity-level-3>
  27. <https://developers.google.com/products/>
  28. <https://developer.yahoo.com/everything.html>
  29. <https://www.flickr.com/services/api/>
  30. <https://dev.twitter.com/overview/api>
  31. <https://developer.spotify.com/>
  32. <https://apiary.io/>
  33. <https://hoppscotch.io/>
  34. <https://ping-api.com/>
  35. <https://testsigma.com/blog/postman-alternatives/>
  36. <http://graphql.org/>
  37. <https://docs.github.com/fr/graphql>