

09.Services Web

3 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 ¹ : atome : ressource (donnée) : REST
 - orientée service ² : 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

2. https://fr.wikipedia.org/wiki/Architecture_orient%C3%A9e_services

- Services web : aux autres applications (HTTP + XML/JSON)
- Exemples :
 - Applications distribuées ³ pour l'entreprise
 - Mashups ⁴ d'applications web (exemples ⁵)
 - Applications Facebook, API Google ⁶
 - IFTTT ⁷
- 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-* ⁸ :
 - 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
 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

SOAP

- Exemple ⁹ 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 ¹⁰

REST : REpresentational State Transfer

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

REST : Interface Uniforme

Parmi les 6 contraintes ¹⁴, l'*interface uniforme* :

Tous les composants (clients, serveurs, proxies...) communiquent avec un même type d'interface générique, au lieu d'avoir une API spécifique à chaque serveur.

- L'interface uniforme est, elle-même, définie par 4 contraintes
 - Identification des *ressources* (URI)
 - Manipulation des *ressources* par des *représentations*

9. https://www.w3schools.com/xml/xml_soap.asp

10. https://www.w3schools.com/xml/xml_wsdl.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

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

14. https://fr.wikipedia.org/wiki/Representational_state_transfer

- Messages autodescriptifs
- Hypermédia comme moteur de l'état de l'application
- *Ressource* : information ou moyen d'accès
 - ex. : météo du jour, adresse ajout d'un article à un blog, ...
- *Représentation* : forme donnée à la ressource
 - ex. : page html, fichier PDF, image, flux RSS, fichier sonore, ...

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 représenter le contenu : 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	Index	GET	/blogs	Affiche la liste
	New	GET	/blog/new	Affiche formulaire création
	Create	POST	/blogs	Création en DB, puis redirection
	Show	GET	/blogs/42	Affiche le blog 42
U	Edit	GET	/blogs/42/edit	Formulaire édition blog 42
	Update	PUT	/blogs/42	MAJ en DB blog 42
D	Destroy	DELETE	/blogs/42	Suppression ne DB blog 42

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

Niveaux de maturité de Richardson ¹⁵

- 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>)

SOAP vs REST

- webservice : exposer son API en REST ou SOAP ?
- 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
 - plus lisible et plus compact
 - maintenance plus facile
 - meilleure tolérance aux pannes

Pour aller plus loin...

- Références
 - SOAP ¹⁶, WSDL ¹⁷, REST ¹⁸
 - Des services web RESTful ¹⁹, Une apologie de REST ²⁰ (recommandés)
 - REST et architectures orientées service ²¹, Présentation ROA ²²

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

16. <https://www.w3.org/TR/soap/>

17. <https://www.w3.org/2002/ws/desc/>

18. <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

19. <https://larlet.fr/david/biologeek/archives/20070629-architecture-orientee-ressource-pour-faire-des-services-web-restful/>

20. <https://web.archive.org/web/20160310205502/http://home.ccil.org/~cowan/restws.pdf>

21. <https://www.figer.com/Publications/SOA.htm>

22. <https://fr.slideshare.net/samijaber/symposium-dng-2008-roa>

- The RESTful cookbook ²³, How important is HATEOAS ²⁴ (stack overflow)
- Exemples de services web :
 - Google ²⁵, Yahoo ²⁶, Flickr ²⁷, Twitter ²⁸, Spotify ²⁹ ...
 - APIary ³⁰ : Aide au design d'une API REST
 - Tests : Postman, Hoppscotch ³¹, Ping-API ³², autres ³³
- GraphQL ³⁴
 - est destiné à devenir la prochaine évolution des apis REST utilisant JSON. Initié par Facebook, Github permet également d'en faire usage ³⁵.

Sources

-
23. <https://restcookbook.com/>
24. <https://stackoverflow.com/questions/20335967/how-useful-important-is-rest-hateoas-maturity-level-3>
25. <https://developers.google.com/products/>
26. <https://developer.yahoo.com/everything.html>
27. <https://www.flickr.com/services/api/>
28. <https://dev.twitter.com/overview/api>
29. <https://developer.spotify.com/>
30. <https://apiary.io/>
31. <https://hoppscotch.io/>
32. <https://ping-api.com/>
33. <https://testsigma.com/blog/postman-alternatives/>
34. <http://graphql.org/>
35. <https://developer.github.com/v4/>