

7. De HTTP à REST

Jean-Luc Falcone

Août 2019

Serveurs de fichiers (1989-)

- **HTTP 0.9**: Premier protocole documenté (1991)
- Web purement statique (*Read-Only*)
- Les URLS correspondent directement à des chemins

This request consists of the word "GET", a space, the document address, [...]

The response to a simple GET request is a message in hypertext mark-up language (HTML). This is a byte stream of ASCII characters.

Common Gateway Interface (1993-)

- L'interface CGI permet au serveur d'appeler un **executable** ou un **script**
- Série de conventions
- Le serveur devient dynamique (*Read-Write*)
- Début des applications webs

CGI en C

```
int main(void)
{
    char *data;
    long m,n;
    printf("%s%c%c\n",
        "Content-Type:text/html;charset=iso-8859-1",13,10);
    printf("<TITLE>Multiplication results</TITLE>\n");
    printf("<H3>Multiplication results</H3>\n");
    data = getenv("QUERY_STRING");
    if(data == NULL)
        printf("<P>Error! Error in passing data from form.");
    else if(sscanf(data,"m=%ld&n=%ld",&m,&n)!=2)
        printf("<P>Error! Invalid data. Data must be numeric.");
    else
        printf("<P>The product of %ld and %ld is %ld.",m,n,m*n);
    return 0;
}
```

PHP (1994-)

- *Personal Home Page*, puis *PHP Hypertext Processor*
- Spécifications en 2014 !
- Page HTML avec des éléments dynamiques interprétés par le serveur

Exemple PHP

```
<html>
  <head>
    <title>PHP Test</title>
  </head>
  <body>
    <?php
if(strpos($_SERVER['HTTP_USER_AGENT'],'MSIE') !== FALSE) {
    echo 'You are using Internet Explorer.<br />';
}
?>
  </body>
</html>
```

Ajax (1999-)

- Acronyme: *Asynchronous JavaScript And XML*
- Extension propriétaire de MS-IE5, rapidement adoptée
- Requêtes asynchrones dont le résultat permet de modifier la page **sans la recharger**
- Meilleures applications Web (par exemple GMail)
- Pas seulement XML (HTML, json, texte, etc.)

Exemple AJAX (côté client)

```
function reqListener () {  
    console.log(this.responseText);  
}
```

```
var oReq = new XMLHttpRequest();  
oReq.addEventListener("load", reqListener);  
oReq.open("GET", "http://www.example.org/example.txt");  
oReq.send();
```


Server Push (2000)

Notifications depuis le serveur

2000 Comet

2006 Server Sent Event

2011 WebSocket

Tendances actuelles

- Toutes ces approches existent toujours !

Tendances actuelles

- Toutes ces approches existent toujours !
- Approche contemporaine:
 - Oublier la notion de page
 - Utiliser un framework pour gérer le client (par exemple *Angular*)
 - Utiliser un serveur web pour servir directement les données

HyperText Transfer Protocol

- Requête – Réponse
- Couche applicative (utilise TCP)
- Utilise des URLs pour identifier les ressources
- Protocole texte en clair (utiliser https)

Requête

Verb que faire avec la ressources

Path identifie la resource

Version indentifie le protocole (optionnel)

Headers méta-données (clés-valeurs)

Body contenu (optionnel)

Requête

Verb que faire avec la ressources

Path identifie la resource

Version indentifie le protocole (optionnel)

Headers méta-données (clés-valeurs)

Body contenu (optionnel)

```
GET /index.html HTTP/1.1
```

```
Host: www.google.ch
```

Path

- Partie de l'URL sans le nom de domaine
- Peut contenir une liste de paramètres de requête

Request parameters

`http://example.com/the/page?id=12&debug=false`

- Path: `/the/page`
- Requests Parameters:

`id 12`

`debug false`

Verbes

GET Obtiens une ressources (safe)

HEAD Comme GET mais seulement métadonnées (safe)

POST Complète une resource existantes (body)

PUT Ajoute ou modifie une ressource (body,idempotent)

DELETE Efface une ressource (idempotent)

Il existe d'autres verbers moins utilisés

Request headers

Métadonnées de la requête. Surtout utilisés pour négocier les formats (codage, langue, compression). Par exemple:

Accept Format acceptable (text/plain)

Accept-Language Langue (fr q=1.0, en; q=0.5)

User-Agent description du client (Mozilla/5.0 (X11; Linux
x86_64; rv:12.0) Gecko/20100101
Firefox/12.0)

Response

La réponse est composée de:

Status Code de retour

Headers Metadonnées

Contenu Resource envoyée

Code de retour

Quelques exemples:

200 OK Succès

201 Created Ressource créée

206 Partial Content Seulement une partie du contenu est envoyé

301 Moved Permanently La nouvelle URL figure dans le contenu

400 Bad request Requête mal formée

404 Not found La ressource n'existe pas

418 I'm a teapot La ressource est une théière

451 Unavailable For Legal Reasons La ressources est censurée

500 Internal Server Error Erreur du côté du serveur

501 Not implemented Pratique durant le développement

Response Header

Le serveur donne des métadonnées décrivant le format de retour, la politique de cache, etc.:

Cache-Control Politique de cache (max-age=3600)

Content-Language Langue du contenu (de)

Content-Type Format du contenu (text/html; charset=utf-8)

Date Date de la ressource (Tue, 15 Nov 1994 08:12:31 GMT)

Expires Expiration de la ressource (Mon, 31 Dec 2018 23:59:59 GMT)

REpresentational State Transfer (REST)

- Série de conventions et contraintes pour définir des service web
- Basé sur des opérations prédéfinies et sans état (**stateless**)
- Introduit en 2000 dans la thèse de doctorat de Roy Fielding

Contraintes

- Architecture Client-Serveur (interface vs. données)
- Protocole sans état
- Caches explicites
- Interface uniforme d'accès
- Système en couches
- Code à la demande

Bénéfices

- Encapsulation
- Interface simple
- Performant
- Scalable

Éléments

Ressource Entité conceptuelle

Identifiant URL

Representation JSON, JPEG, HTML, ...

Repr. Metadata type, last-modified, ...

HTTP: URL

Les URLs représentent des ressources:

- `http://example.org/blog/post/123`
- `http://example.org/blog/post/123/comment/2`

ou des collections de ressources:

- `http://example.org/blog/posts/`
- `http://example.org/blog/post/123/comments/`

HTTP: Verbes

Verbe	Ressource	Collection
GET	Obtient la res.	Liste les res.
PUT	Crée ou remplace une res.	Remplace une collection
POST		Crée une nouvelle res.
DELETE	Efface la ressource	Efface toute la collection

Par exemple:

- GET /blog/post/123 obtient l'article 123
- GET /blog/post/123/comments listes les commentaires de l'article 123
- DELETE /blog/post/123/comments/2 efface le commentaire 2 de l'article 123
- POST /blog/posts publie un nouvel article (numméroté par le serveur)
- PUT /blog/post/123 publie une version corrigée de l'article 123.

Bénéfices

- Portable (ubiquité du web)
- Interface simple
- Performant
- Scalable
- Tolérance à la panne

JSON

- JavaScript Object Notation (JSON)
- Format de sérialisation de données en texte.
- Lisible par un humain
- Syntaxe légère, sous-ensemble de JS
- Parseurs dans presque tous les langages

Grammaire

object

{ }

{ members }

members

pair

pair , members

pair

string : value

array

[]

[elements]

elements

value

value , elements

value

string

number

object

array

true

false

null

Exemple

```
{
  "name": {
    "firstnames": [
      "Alice",
      "Pleasance"
    ],
    "surname": "Hargreaves",
    "born": "Lidell"
  },
  "birthday": {
    "date": "1852-05-04",
    "place": "Oxford"
  }
}
```

En javascript

Initialement les strings JSON étaient directement converties en JavaScript à l'aide la fonction `eval`. En raison des problèmes de sécurité, on utilisera plutôt les méthodes de l'objet JSON.

```
> o
{ foo: 123, bar: true }
> let s = JSON.stringify(o);
> s
'{"foo":123,"bar":true}'
> JSON.parse(s);
{ foo: 123, bar: true }
```


En Python

```
>>> import json
>>> str = "{\"foo\":123,\"bar\":true}"
>>> json.loads(str)
{'bar': True, 'foo': 123}
```

En Java (avec Jackson)

```
public class FooBar {  
    public int foo;  
    public boolean bar;  
}
```

```
ObjectMapper om = new ObjectMapper();  
String str = "{\"foo\":123,\"bar\":true}"  
FooBar fb = mapper.readValue( str, FooBar.class);
```