

# Technologies et langages pour le Web

Coté serveur  
Z225DM06

Yannick Loiseau

Université Clermont-Auvergne

Licence Informatique 2<sup>e</sup> année  
Version du 13 mars 2023

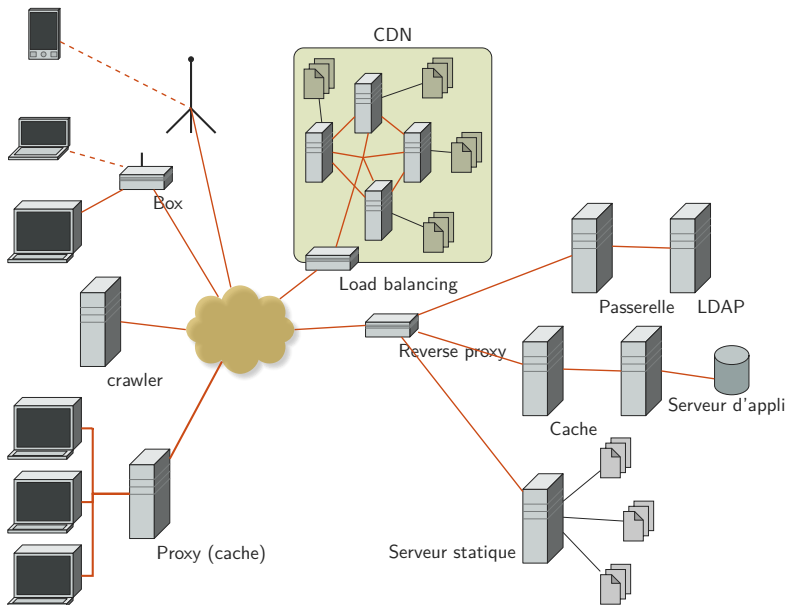
Yannick Loiseau

- ▶ `yannick.loiseau@uca.fr`
- ▶ bureau D008 (bâtiment ISIMA)

## Technologies et langages pour le Web Serveur

- ▶ CM : 15h
- ▶ TP : 15h

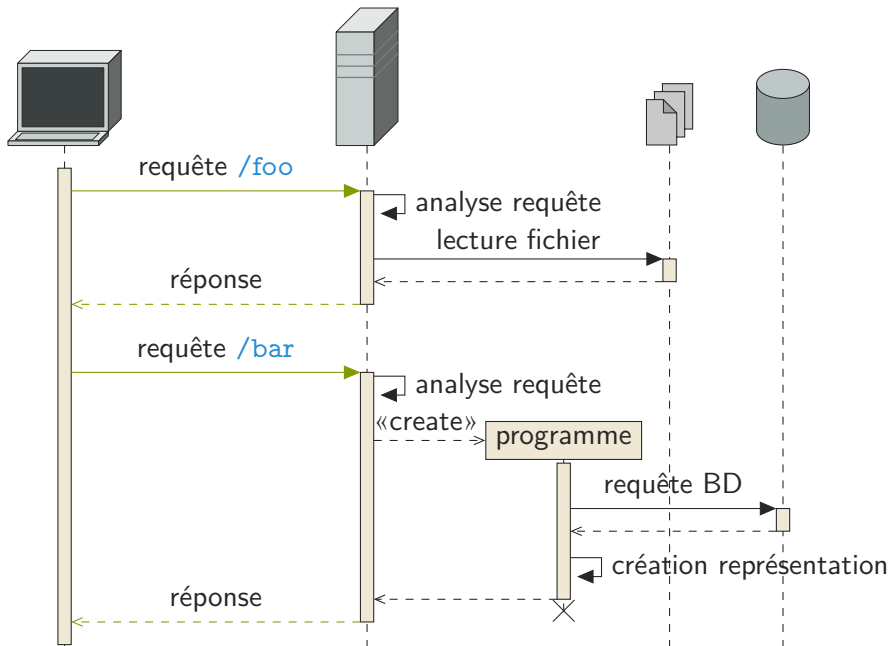
Vidéos...



Hypertext Transfert Protocol : (RFC 2616)

RFC 7230, RFC 7231, RFC 7232, RFC 7233, RFC 7234, RFC 7235

Manipulation des ressources transfert des représentations de l'état



# Client/Serveur

- ▶ séparation des responsabilités
- ▶ Données – (Traitements) – Présentation
- ▶ portabilité de l'interface
- ▶ passage à l'échelle du serveur
- ▶ évolution indépendante

# Sans état

- ▶ requête suffisante pour répondre
- ▶ pas de contexte client sur le serveur
- ▶  $\Rightarrow$  transparent pour les intermédiaires
- ▶ état de l'application sur le client



# Sans état

## Avantages

- ▶ visibilité : monitoring → requête (intermédiaires)
- ▶ fiabilité : reprise sur panne plus simple
- ▶ évolutivité (échelle) :
  - ▶ pas d'état  $\Rightarrow$  moins de ressource
  - ▶ implém. plus simple
- ▶ possibilité d'intermédiaires (couches)

# Sans état

## Inconvénients

- ▶ consistance : pas de contrôle serveur → implém. / comportement clients
- ▶ performances : répétition d'informations

# Caches

- ▶ réponse cachable
- ▶ sémantique claire

# Caches

↘ interactions  $\Rightarrow$  ↗ performances

- ▶ efficacité
- ▶ échelle
- ▶ latence
- ▶ robustesse

↘ fiabilité

# Interface uniforme

- découplage service ↔ implémentation
- évolution indépendante

## Contraintes (ReST)

- identification des ressources
- manipulation via représentations
- messages auto-descriptifs
- état conduit par hypermédia (HATEOAS)

# Couches

- ▶ système en couches hiérarchiques
- ▶  $\Rightarrow$  intermédiaires
- ▶ visibilité limitée
- ▶ indépendance
- ▶ passage à l'échelle

interface uniforme + couches  $\Rightarrow$  *pipe and filter*

composition des composants

flot de données

start line

en-têtes

ligne vide  
corps

version, type de message,...
Nom : valeur Nom : valeur ...
...



# En-têtes généraux

- ▶ Cache-Control
- ▶ Connection : close, keep-alive
- ▶ Date
- ▶ Pragma
- ▶ Via : intermédiaires
- ▶ ...

## Requête

start line : verbe identifiant version

### Exemple

```
GET /foo HTTP/1.1
```

# Manipulation de ressources

## RFC 7231

- ▶ création : **POST**
- ▶ lecture : **GET** (**HEAD**)
- ▶ mise à jour : **PUT** – **PATCH** (RFC 5789)
- ▶ suppression : **DELETE**

# Propriétés des méthodes

- ▶ idempotence
- ▶ sans effet de bord

## Définition (Idempotence)

$n$  requêtes  $\equiv$  1 requête

## Définition (sans effet de bord)

pas de modification de l'état de la ressource

# Propriétés des méthodes

- ▶ indépendance des intermédiaires
- ▶ « cachabilité »
- ▶ reprise sur erreur
- ▶ automatisation / responsabilité

GET /articles/1234?action=delete

# Propriétés des méthodes

Méthode	Sans effet de bord	Idempotente
GET	✓	✓
HEAD	✓	✓
PUT	✗	✓
DELETE	✗	✓
POST	✗	✗
PATCH	✗	✗

# Propriétés des méthodes

À propos du POST

Pas de sémantique bien définie

- ▶ POSTa : append (commentaire, message de forum)
- ▶ POSTp : process (traitement quelconque)

RFC 2310 : en-tête **Safe** : (yes|no)

# Méta informations, diagnostic et connexion

- ▶ **OPTIONS** : méta-informations
- ▶ **CONNECT** : tunnel
- ▶ **TRACE** : echo



## Réponse

start line : version status message

## Exemple

HTTP/1.1 200 OK

5 catégories :

- ▶ 1xx : Informations (100 [Continue](#), 101 [Switching Protocols](#))
- ▶ 2xx : Succès (200 [OK](#), 201 [Created](#), 204 [No Content](#))
- ▶ 3xx : Redirection ou pas de contenu (301 [Moved Permanently](#), 304 [Not Modified](#))
- ▶ 4xx : Erreur du client (404 [Not Found](#), 401 [Unauthorized](#), 418 [I'm a teapot](#)<sup>1</sup>)
- ▶ 5xx : Erreur du serveur (500 [Internal Server Error](#), 504 [Gateway Timeout](#))

<http://www.iana.org/assignments/http-status-codes/>

---

1. HTCPCP : RFC 2324

## 2xx : Succès

```
GET /example HTTP/1.1  
Host : www.example.com  
User-Agent : Mozilla/5.0  
[...]
```

```
HTTP/1.1 200 OK  
Date : Sat, 22 Dec 2012 00 :00 :00 GMT  
Server : Apache  
Content-Type : text/html  
Content-Length : 1270  
[...]
```

## 2xx : Succès

POST /articles/ HTTP/1.1

Host : www.example.com

Content-Type : text/plain

Content-Length : 13

Hello World !

HTTP/1.1 201 Created

Date : Sat, 22 Dec 2012 00 :00 :00 GMT

Location : /articles/42

GET /articles/42 HTTP/1.1

Host : www.example.com

HTTP/1.1 200 OK

Date : Sat, 22 Dec 2012 00 :00 :05 GMT

Content-Type : text/plain

Content-Length : 13

Hello World !

## 2xx : Succès

POST /articles/42 HTTP/1.1

Host : www.example.com

Content-Type : text/plain

Content-Length : 6

Salut

HTTP/1.1 200 OK

Date : Sat, 22 Dec 2012 00 :00 :20 GMT

Content-Type : text/plain

Content-Length : 19

Hello World !

Salut

## 2xx : Succès

POST /calculator HTTP/1.1

Host : www.example.com

*[donnees a traiter]*

HTTP/1.1 202 Accepted

Date : Sat, 22 Dec 2012 00 :00 :00 GMT

Location : /calculator/status/42

## 2xx : Succès

POST /search HTTP/1.1

Host : www.example.com

*[requete complexe (sparql, graphql, ...)]*

HTTP/1.1 303 See Other

Date : Sat, 22 Dec 2012 00 :00 :00 GMT

Location : /articles/42

GET /articles/42 HTTP/1.1

HTTP/1.1 200 OK

Date : Sat, 22 Dec 2012 00 :00 :02 GMT

Content-Type : text/plain

Content-Length : 19

Hello World !

Salut

## 2xx : Succès

DELETE /articles/42 HTTP/1.1

Host : www.example.com

HTTP/1.1 204 No Content

Date : Sat, 22 Dec 2012 00 :01 :00 GMT

GET /articles/42 HTTP/1.1

Host : www.example.com

HTTP/1.1 410 Gone

Date : Sat, 22 Dec 2012 00 :01 :02 GMT



## 4xx : Erreur du client

```
PATCH /articles/42 HTTP/1.1
```

```
Host : www.example.com
```

```
[...]
```

```
HTTP/1.1 405 Method Not Allowed
```

```
Date : Sat, 22 Dec 2012 00 :01 :00 GMT
```

```
Allow : OPTIONS, GET, HEAD, POST, DELETE
```

```
Content-Length : 0
```

## 4xx : Erreur du client

```
PUT /articles/42 HTTP/1.1
Host : www.example.com
Content-Type : application/vnd.ms-excel
[...]
```

```
HTTP/1.1 415 Unsupported Media Type
Date : Sat, 22 Dec 2012 00 :01 :00 GMT
Content-Length : 0
```

## 4xx : Erreur du client

PUT /articles/42 HTTP/1.1

Host : www.example.com

Content-Type : text/plain

[...]

HTTP/1.1 428 Precondition Required

Date : Sat, 22 Dec 2012 00 :01 :00 GMT

Content-Length : 0

au delà des fichiers statiques

- ▶ passerelle
- ▶ calcul
- ▶ manipulation de ressources
- ▶ portail
- ▶ ...

⇒ code coté serveur

## procédure

- ▶ entrée : requête
- ▶ sortie : réponse

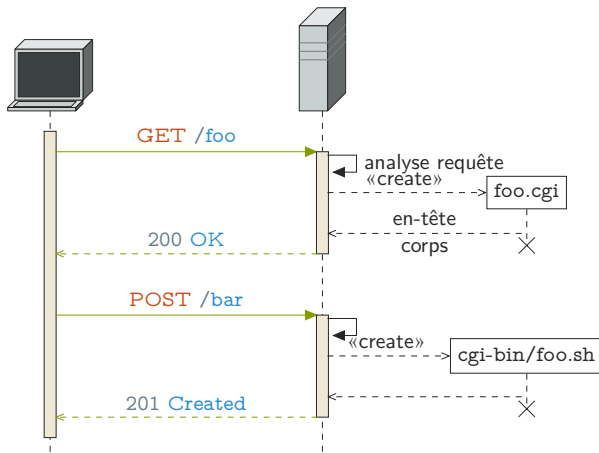
architecture n-tiers

- ▶ V1.0 1993 NCSA
- ▶ V1.1 1995 → 2004 RFC 3875



programme exécutable

- ▶ autonome
- ▶ n'importe quel langage (script)



configuration serveur  $\Rightarrow$  correspondance uri + méthode + en-têtes  $\mapsto$  programme

## programme

- ▶ test méthode
- ▶ test chemin
- ▶ test en-têtes
- ▶ lecture et analyse du corps
- ▶ génération des en-têtes
- ▶ génération du corps

# Variables d'environnement

- ▶ `GATEWAY_INTERFACE` : CGI/version
- ▶ `SERVER_NAME`
- ▶ `SERVER_PROTOCOL`
- ▶ `SERVER_PORT`
- ▶ `REQUEST_METHOD`
- ▶ `PATH_INFO` : chemin relatif au script CGI
- ▶ `SCRIPT_NAME` :
- ▶ `QUERY_STRING` : application/x-www-form-urlencoded brut
- ▶ `REMOTE_ADDR` : IP du client
- ▶ `AUTH_TYPE`
- ▶ `REMOTE_USER`
- ▶ `CONTENT_TYPE` : type média du contenu de la requête
- ▶ `CONTENT_LENGTH`
- ▶ `HTTP_ACCEPT` : type média demandés par le client
- ▶ `HTTP_*` : tous les en-têtes HTTP

GET /cgi-bin/foo.sh/bar/baz?a=1&b=2 HTTP/1.1

SERVER\_PROTOCOL : HTTP/1.1

REQUEST\_METHOD : GET

PATH\_INFO : /bar/baz

SCRIPT\_NAME : /cgi-script/foo.sh

QUERY\_STRING : a=1&b=2

CONTENT\_LENGTH : 0

- ▶ Corps → stdin
- ▶ Réponse → stdout
  - ▶ en-têtes + ligne vide + corps
  - ▶ en-tête spécial [Status](#)

```
#!/bin/sh

if [ $REQUEST_METHOD != "GET" ] ; then
    echo "Content-Type : text/plain"
    echo "Status : 405 NotAllowed"
    echo "Allow : GET"
    echo ""
    echo "Method not Allowed"
    exit 0
fi

echo "Content-Type : plain/text"
echo ""
echo "Hello world!"
```



## bibliothèques

- ▶ accès env.
- ▶ parsing *query string* et formulaire (application/x-www-form-urlencoded)
- ▶ lecture stdin
- ▶ codes erreurs
- ▶ gestion en-têtes (échapement et encodage des valeurs)
- ▶ génération du corps, échapement html, ...



- ▶ executable « normal »  $\Rightarrow \forall$  techno. serveur
- ▶ autonome
- ▶ tests
- ▶ déploiement
- ▶ gestion droits (suexec)
- ▶ isolation

**X**

- ▶ 1 req.  $\Rightarrow$  1 sous-processus
- ▶ pas d'état partagé :
  - ✓ HTTP (sans état)
  - X** connexion BD, cache interne, ...

CGI → script ⇒ interprété  
facile à déployer

interpréteur → serveur

## Exemple (Apache)

- ▶ mod\_perl
- ▶ mod\_php
- ▶ mod\_python, mod\_wsgi
- ▶ mod\_lua
- ▶ mod\_mono
- ▶ mod\_ruby
- ▶ mod\_tcl
- ▶ ...

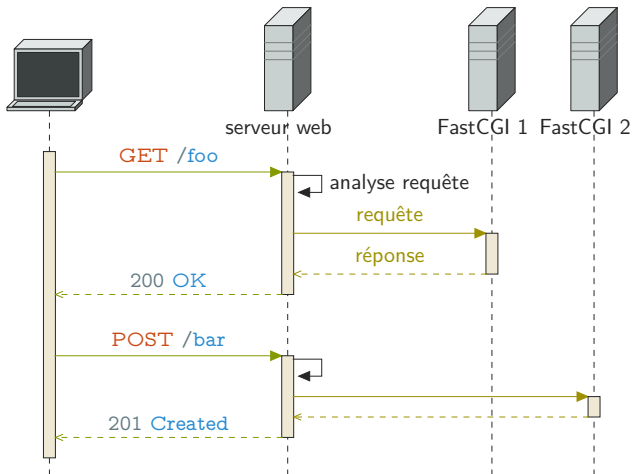
- ▶  $\equiv$  CGI
- ▶ ✓ pas de sous-processus
- ▶ ✓ accès direct aux variables
- ▶ fonctions util.
- ▶ interprété
- ▶ ✗ pas d'isolation

- ▶ 1996 Open Market
- ▶ <http://www.fastcgi.com/devkit/doc/fcgi-spec.html>

# Principe

- ▶  $\equiv$  CGI
- ▶ daemon
- ▶ communication  $\rightarrow$  socket (protocole spécifique)





# Avantages

- ▶ pas de processus  $\Rightarrow$  performances
- ▶ isolation  $\Rightarrow$  sécurité
- ▶ réutilisation (cache, connexions BD)
- ▶ répartition charge
- ▶ n'importe quel langage et serveur

# Outils

- ▶ fcgiwrap
- ▶ spawn-fcgi

# Outils

aussi SCGI, AJP

- ▶ conteneur
- ▶  $\cong$  FastCGI
- ▶  $\cong$  embarqué

langage spécifique  $\Rightarrow$  accès facilité aux paramètres

- ▶ entrée : objet requête
- ▶ sortie : objet réponse

communication HTTP

## Exemple

- ▶ Java : Tomcat, GlassFish, Jetty, WebSphere, WildFly (JBoss)
- ▶ Javascript : Node.js
- ▶ .Net : IIS
- ▶ Python : Gunicorn, Paste, Tornado, uWSGI, Zope
- ▶ Ruby : Passenger, Mongrel, Unicorn



## API spécifique

- ▶ Rack (ruby)
- ▶ WSGI (python)
- ▶ servlet, JSP, JAX-(R/W)S (java)
- ▶ ...

≈ lourd

- ▶  $\cong$  serveur d'application
- ▶ léger

- serveur autonome
- embarqué
- reverse proxy
- répartition
- SOA

⇒ micro-services

- ▶ application : sans état
- ▶ maintenu par le client  $\Rightarrow$  toutes infos : dans la requête
- ▶ spécifique ?
  - $\Rightarrow$  cookies

# Principe

## Définition (Cookie)

- ▶ information serveur : clé → valeur + méta
- ▶ spécifiques à l'application
- ▶ stockée sur le client
- ▶ renvoyée à chaque requête
- ▶ opaque pour le client

RFC 6265

## Exemple

préférences utilisateur (**Set-Cookie** : **theme=dark**)

# Principe

## En-têtes

- ▶ **Set-Cookie** dans les réponses
- ▶ **Cookie** dans les requêtes

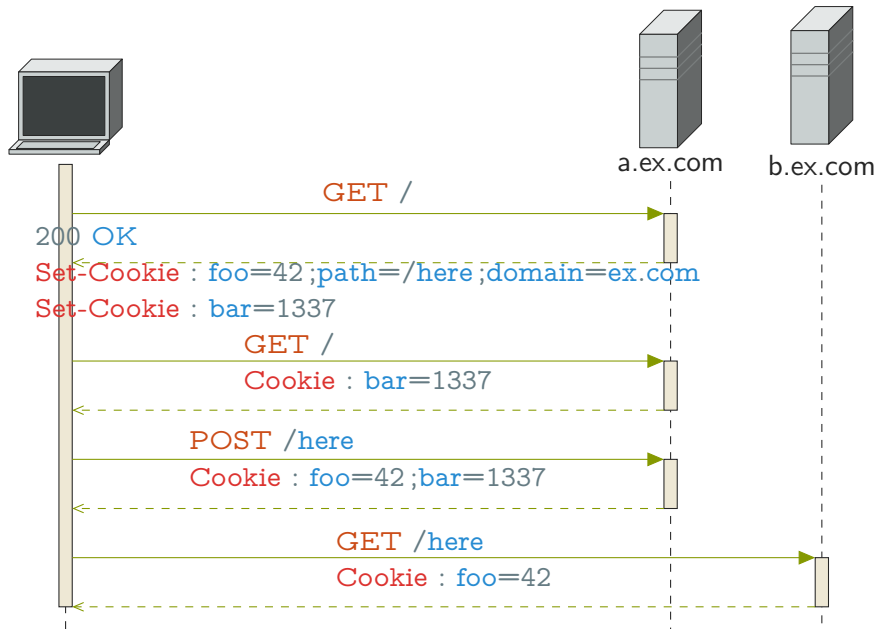
# Principe

## Méta-données

### champ de validité

- ▶ serveur : [domain](#)
- ▶ chemin : [path](#)
- ▶ durée : [expires](#), [max-age](#)
- ▶ protocole : [secure](#), [httponly](#) (pas API JS)





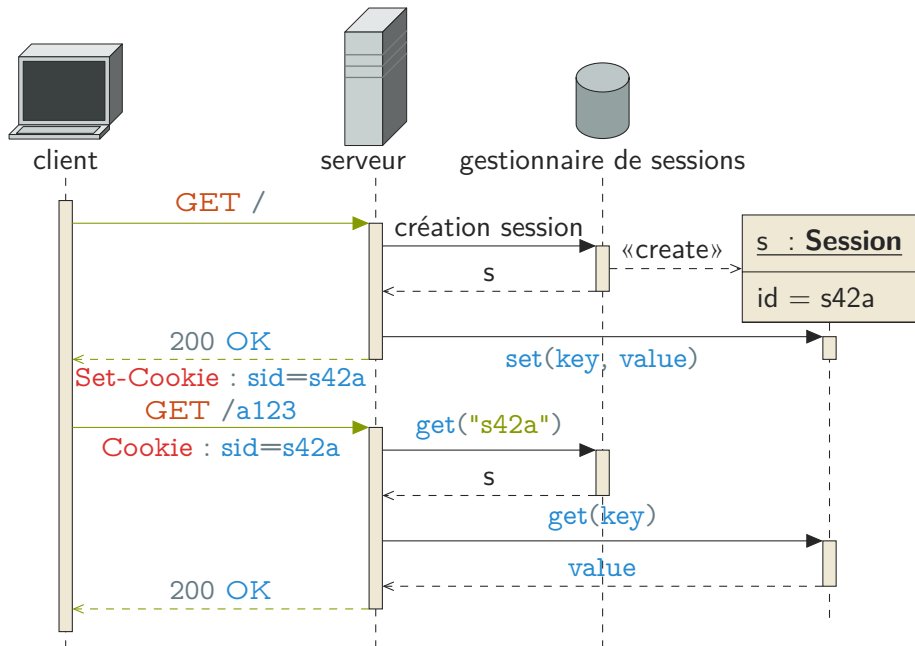
# Sessions

## Définition

- ▶ structure de donnée
- ▶ mémoire serveur
- ▶ identifiant  $\mapsto$  clés/valeurs  $\Rightarrow$  cookie

## Exemple





authentification, préférences, panier de commande, ...



# Sessions

## Problèmes

⇒ état coté serveur

- ▶  intermédiaires
- ▶  cache
- ▶  répartition de charge
- ▶  reprise sur panne

solution ?

- ▶ persistant entre sessions
- ▶ partagé entre serveurs

≡ ressource non référençable (pas d'url) ⇒ non manipulable

## Exemple

commande sur différents terminaux

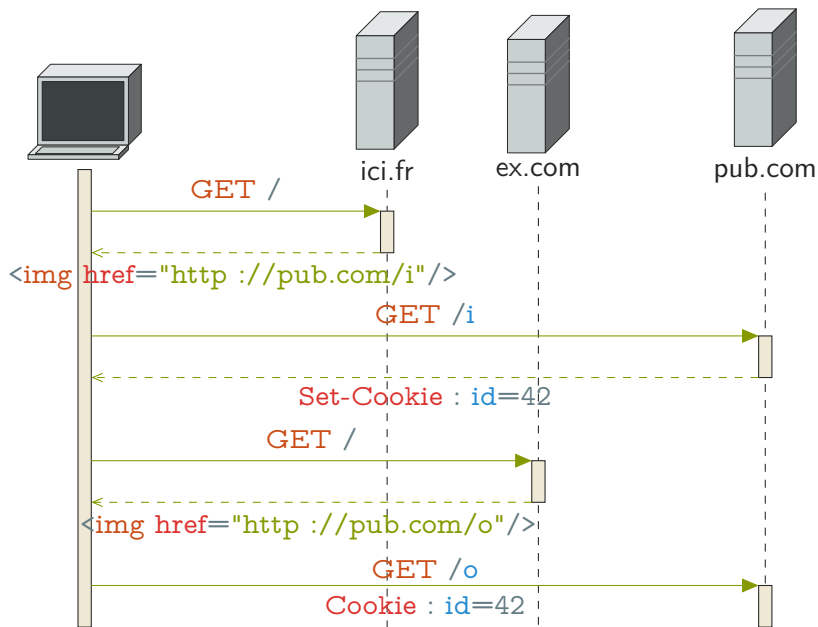
# Cookies tiers

## Définition (Cookie tier)

cookie dans la réponse d'un serveur tiers (pub, image)



vie privée  $\Rightarrow$  blocages, suppression



# Sécurité

- ▶ autorité globale (si authentication) : cross-site request forgery (CSRF)
- ▶ données en clair : session hijacking, replay  $\Rightarrow$  horodatage + chiffrement + signature
- ▶ faible intégrité :  $\neq$  hosts, ports, chemins

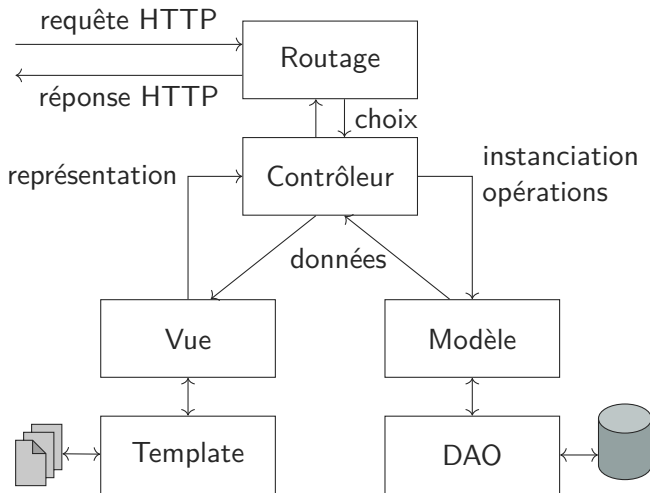
## Modèle – Vue – Contrôleur

Modèle : données et règles métier

Vue : présentation et formatage

Contrôleur : interactions et connexion M–V





- ▶ Hierarchical MVC
- ▶ Presentation – Abstraction – Control
- ▶ Model – View – Adapter
- ▶ Model – View – Presenter
- ▶ Model – View – View Model (MVVM)

# Frameworks

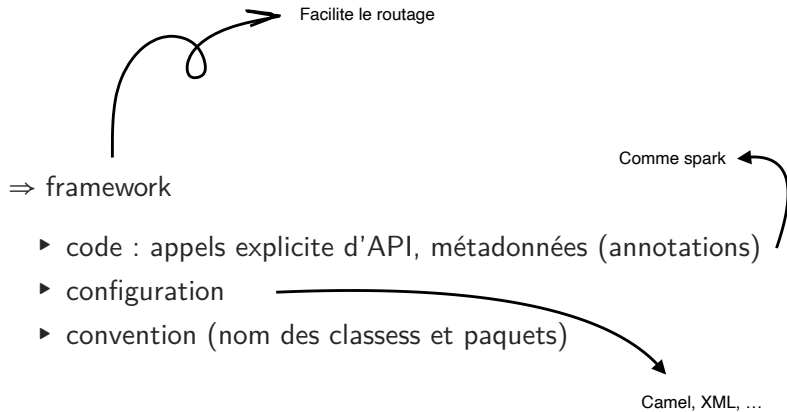
## Micro-frameworks

- ▶ Sinatra (Ruby)
- ▶ Flask (Python)
- ▶ Spark (Java)
- ▶ ...

[https://en.wikipedia.org/wiki/Category:Web\\_frameworks](https://en.wikipedia.org/wiki/Category:Web_frameworks)

## orchestration

- ▶ url (en-têtes) → choix modèle instantiation
- ▶ verbe (en-têtes) → opération
- ▶ extraction information (modèle de vue ?)
- ▶ en-têtes (url) → choix vue
- ▶ génération réponse
- ▶ capture exceptions → code erreur HTTP



## Jax-RS (Jersey)

```
@Path("hello/{name}")
public class HelloResource {
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String sayHello(@PathParam("name") String name) {
        return String.format("Hello %s!", name);
    }
}
```



On fait le lien avec le lien "hello/{name}"

Produit du texte brut



Projet

## Spark

```
public class App {  
    public static void main(String[] args) {  
        get("hello/ :name",  
            (req, res) -> String.format("Hello %s!", req.params(" :name")));  
    }  
}
```

- règles métier
- accès aux données  $\Rightarrow$  persistance



On sépare la vue et le contrôleur, à l'instar de l'encapsulation en POO

pas de dépendance :

- contrôleur
- vue

Un objet ne peut pas être stocké en SQL dans une BDD

## Définition (Impedance Mismatch)

différence de concepts entre modèles objet et relationnel



- types scalaires vs. références
- (multi)set vs. graphe
- interface uniforme (CRUD) vs. interface spécifique
- déclaratif vs. impératif
- égalité vs. identité
- encapsulation, sous-typage, polymorphisme
- transactions

Un objet qui encapsule toutes les données dans des objets

**DAO** Data Access Object


⇒ encapsule l'accès aux données

Structure de données qui encapsule les données de la BDD

**DTO** Data Transfert Object


structure, modèle




Déconseillé d'importer comme ça

`java.sql.*`

```
class UserDAO {  
    public User getUserById(int userId) {  
        PreparedStatement stm = connection.prepareStatement("select name, age  
            from users where id = ?");  
        stm.setInt(1, userId);  
        ResultSet rs = stmt.executeQuery();  
        rs.first();  
        User u = new User(userId);  
        u.setName(rs.getString(1));  
        u.setAge(rs.getInt(2));  
        return u;  
    }  
}
```





"SQL to Object"


`org.sql2o.*`

```
class UserDao {  
    public User getUserById(int userId) {  
        return connection.createQuery("select id, name, age from users where id =  
            :id")  
            .addParameter("id", userId)  
            .executeAndFetch(User.class);  
    }  
}
```

**ORM** (*Object-Relational Mapper*)

```
String name = User.getById(1).getName();
```

```
User u = User.getById(42);  
u.setName("Zaphod");  
u.save();
```




Enregistre dans la BDD




On le configure, et le reste est fait tout seul

« magie noire »



- configuration (xml)
- code : annotations, redéfinition, interface
- convention



Hibernate : le plus connu des ORM

"La classe User est liée à la table users"

```
<hibernate-mapping package="my.application">
  <class name="User" table="users">
    <id name="id" column="id">
      <generator class="increment"/>
    </id>
    <property name="name" column="name"/>
    <property name="age" column="age"/>
  </class>
</hibernate-mapping>
```

"On lie le name de la  
classe à la colonne name  
de la table"

Persistante

javax.persistence.\*



```
@Entity
@Table(name="users")
public class User {
    private Integer id;
    private String name;
    private Integer age;

    @Id
    @GeneratedValue(generator="increment")
    @GenericGenerator(name="increment", strategy = "increment")
    public Long getId() {
        return id;
    }
    private void setId(Long id) {
        this.id = id;
    }

    @Column(name="name")
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

Auto-incrément de l'id

Il s'inspire de ce schémas pour  
générer les requêtes SQL

- ▶  many to many  $\Rightarrow$  jointures
- ▶  synchronisation

## Définition (Template)

- ▶ génération de texte
- ▶ patron → forme générale
- ▶ moteur → données

HTML notamment



# Illimité



Même langage pour définir la template et pour l'executer

langage complet (hôte)




"Là j'écrit du PHP, là j'écrit du HTML"  
On doit le dire explicitement

- traitement quelconque
- ⚠ peut modifier le modèle



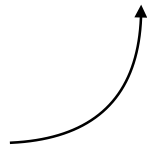
Le script peut modifier la source

## Limité



Langage pour le patron (ex : EJS) qui permet de lire des valeurs

langage spécifique

- ▶ valeur en lecture
  - ▶ pas d'effet de bord
- 

Si la vue n'a pas besoin de la données, on l'appelle pas

*pull* : vue (template) → données (modèle)  
⇒ à la demande

*push* : contrôleur (vue) → template  
⇒ a priori



Les données sont lues à priori



- ▶ pipeline
- ▶ callback
- ▶ reverse callback    != Callback :)

- ▶ mélange code/template → changement de contexte
- ▶ → appel de code natif
- ▶ *pull*

## Exemple

*server page* (ASP, JSP, PHP)

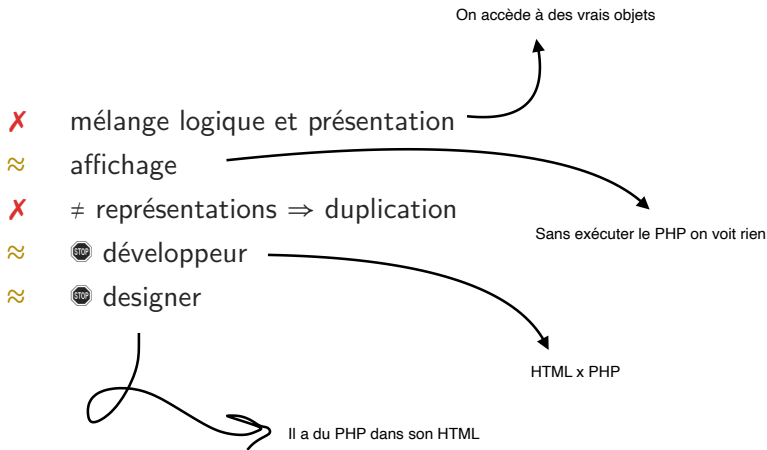
“Là c’est du code, là c’est pas du code”

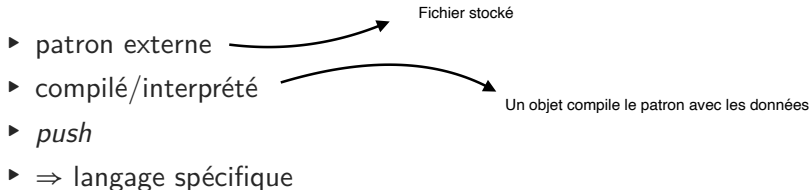
The diagram consists of a hand-drawn loop starting from the 'pull' item in the list, going up and then right to point at the quote. A second arrow starts from the 'server page' text and points up and right to the same quote. A third arrow starts from the 'server page' text and points up and left to the 'pull' item.

```
<?php
$data = DAO.getObject($GET["id"]);
?>
<h2><?=$data->getTitle() ?></h2>
<ul>
<?foreach ($data->getElements() as $elt) { ?>
    <li><a href='<?=$elt->url ?>'><?=$elt->name ?></a></li>
<?} ?>
</ul>
```

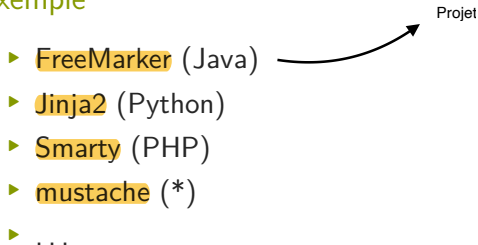
```
<?php
$data = DAO.getObject($GET["id"]);

echo "<h2>" . $data->getTitle() . "</h2>";
echo "<ul>";
foreach ($data->getElements() as $elt) {
    echo "<li><a href='" . $elt->url . "'>";
    echo $elt->name;
    echo "</a></li>";
}
echo "</ul>";
?>
```

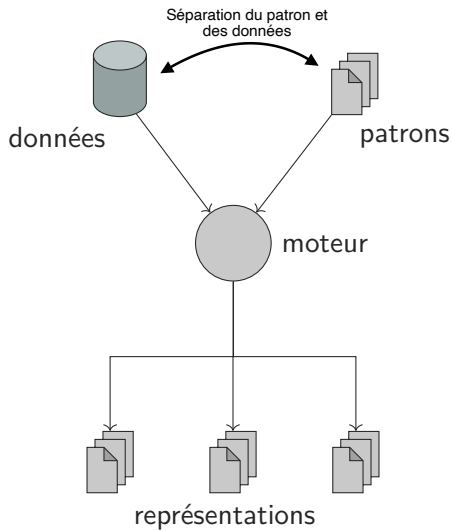




## Exemple



[https://en.wikipedia.org/wiki/Template\\_engine\\_%28web%29](https://en.wikipedia.org/wiki/Template_engine_%28web%29)



```
<h2>{{title}}</h2>
{% if elements %}
<ul>
    {%- for name, url in elements %}
    <li><a href="{{url}}">{{name}}</a></li>
    {%- endfor %}
</ul>
{%-else%}
<p>Rien...</p>
{%endif%}
```

Jinja, comme du python

```
<table><caption>{{title}}</caption>
  <thead><tr><th>Name</th><th>URL</th></tr></thead>
  <tbody>
    {%- for name, url in elements %}
    <tr><td>{{name}}</td>
      <td><a href="{{url}}">{{url}}</a></td></tr>
    {%- endfor %}
  </tbody>
</table>
```

```
from jinja2 import Environment, FileSystemLoader
templates = './listings/templates/'
env = Environment(loader=FileSystemLoader(templates))
data = [
    {
        "title" : "Mon super titre",
        "elements" : [
            ("nom1", "http ://url1.example.com/"),
            ("nom2", "http ://url2.example.com/")
        ]
    },
    {
        "title" : "Autre titre",
        "elements" : []
    }
]

for t in ['list', 'table'] :
    with file(templates + t + 'out.html', 'w') as out :
        for d in data :
            out.write(env.get_template(t + ".html").render(d))
```



```
<h2>Mon super titre</h2>
```

```
<ul>
```

```
  <li><a href="http ://url1.example.com/">nom1</a></li>
```

```
  <li><a href="http ://url2.example.com/">nom2</a></li>
```

```
</ul>
```

```
<h2>Autre titre</h2>
```

```
<p>Rien...</p>
```

```
<table><caption>Mon super titre</caption>
```

```
  <thead><tr><th>Name</th><th>URL</th></tr></thead>
```

```
  <tbody>
```

```
    <tr><td>nom1</td>
```

```
      <td><a
```

```
        href="http ://url1.example.com/">http ://url1.example.com/</a></td></tr>
```

```
    <tr><td>nom2</td>
```

```
      <td><a
```

```
        href="http ://url2.example.com/">http ://url2.example.com/</a></td></tr>
```

```
  </tbody>
```

```
</table>
```

```
<table><caption>Autre titre</caption>
```

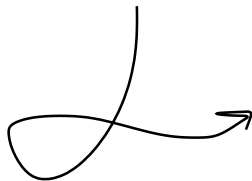
```
  <thead><tr><th>Name</th><th>URL</th></tr></thead>
```

```
  <tbody>
```

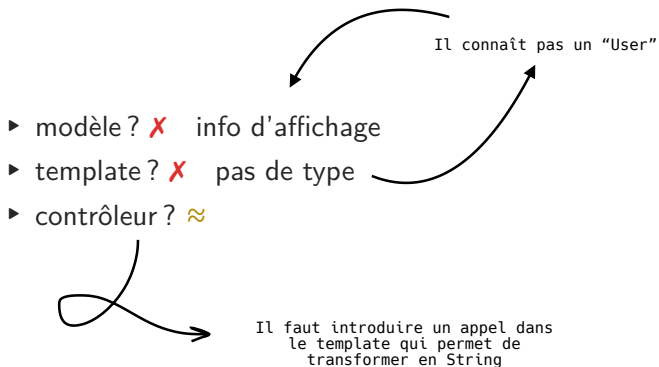
- ✓ sépare logique et présentation
  - ✓ affichage
  - ≈ duplication de la logique → présentation
  - ✓ ✌ développeur
  - ✓ ✌ designer
- }

# Problème

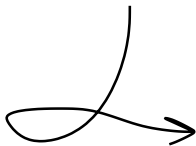
conversion valeurs → chaînes date, valeurs numériques, etc.



Un float ?? Quelle précision



appel (implicite) à `toString` (ou équivalent)



Pour dire "je veux n décimales"

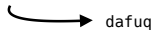
"Comment je dois faire le rendu des nombres"



MVCR : renderer

contrôleur / moteur de template

⇒ i18n



dafuq

```
def human_date(val) :  
    return ...  
  
def machine_date(val) :  
    return ...  
  
env = jinja2.Environment(...)  
env.filters['hdate'] = human_date  
env.filters['mdate'] = machine_date
```

Jinja

```
<time datetime="{{begin|mdate}}">{{begin|hdate}}</time>
```



Le renderer

internet international  $\Rightarrow \neq$  langues, cultures

- ▶ symboles
- ▶ représentations
- ▶ codes

$\neq$  représentations et rendus

$\Rightarrow$  négociation + méta-données d'encodage

$\Rightarrow$  négociation + méta-données de langue (RFC 1766)



## locale (POSIX)

- ▶ langue
- ▶ sens d'écriture
- ▶ ordre alphabétique
- ▶ formatage des nombre (1,000.5 vs. 1 000,5)
- ▶ formatage de date (2010-10-01, 01/10/10)
- ▶ formatage des monnaies (\$3.5 vs. 3,5 €)
- ▶ règles typographiques (« » vs. "" , listes, ident)




i18n → adaptable

- ▶ design + code
- ▶ prise en compte des locales

L10n → adaptation

- ▶ traduction
- ▶ locales
- ▶ styles
- ▶ 1 fois par langue !

- ▶ caractère : abstrait
- ▶ jeu de caractère (charmap) / page de code (code page) :  
caractère  $\leftrightarrow$  code numérique
- ▶ encodage(charset) : représentation de ce code (nb de bits, ...)
- ▶ fonte : caractère  $\leftrightarrow$  glyphe

- ▶  système de fichier (nom)
- ▶  flux (réseau)
- ▶  base de données

toute la chaîne doit soit :

- ▶ être homogène (unicode)
- ▶ gérer l'encodage à chaque niveau  $\Rightarrow$  connaître celui des autres

# Unicode

- ▶ jeu de caractère
- ▶ encodage (UTF)

+1 000 000

U+000000 → U+10FFFD

## encodage

- ▶ UTF-8
- ▶ UTF-16
- ▶ UTF-32



## multi-octets

- ▶ petit-boutiste (little endian)
- ▶ gros-boutiste (big endian)

⇒ BOM (byte order mark) : U+FEFF

é : U+00E9

latin1 (ISO 8859-1) : E9

UTF-8 : 2 octets : C3 A9

UTF-16 : 2 octets (plus le BOM) : FF FE | E9 00

UTF-32 : 4 octets (plus le BOM) : FF FE 00 00 | E9 00 00 00

é utf-8 → latin1 « Ã© »

latin1 : C3 → « Ã » A9 → « © »

Ω « U+03A9 GREEK CAPITAL LETTER OMEGA » :

UTF-8 : CE A9

UTF-16 : FF FE | A9 03

UTF-32 : FF FE 00 00 | A9 03 00 00

þ « U+1E55 LATIN SMALL LETTER P WITH ACUTE »

UTF-8 : E1 B9 95 (55 → U)

UTF-16 : FF FE | 55 1E

UTF-32 : FF FE 00 00 | 55 1E 00 00

ℳ « U+1D4B4 MATHEMATICAL SCRIPT CAPITAL Y »

UTF-8 : 4 octets : F0 9D 92 B4

UTF-16 :  $2 \times 2$  octets + BOM D835 DCB4 →  
FF FE | 35 D8 | B4 DC

UTF-32 : 4 octets (plus le BOM) : FF FE 00 00 | B4 D4 01 00

## Diacritiques : é

- ▶ U+00E9
- ▶ U+0065 U+0301

## Emoji ZWJ : Deaf Woman Medium-Dark Skin Tone

- ▶ U+1F9CF Deaf Person
- ▶ U+1F3FE Medium-Dark Skin Tone
- ▶ U+200D Zero Width Joiner
- ▶ U+2640 Female Sign
- ▶ U+FE0F Variation Selector-16

- ▶ identification
- ▶ authentification
- ▶ autorisation

## Définition (Identification)

- ▶ associer une identité/un identifiant à l'utilisateur
- ▶ pas forcément un compte
- ▶ pas forcément identité physique

## Exemple

- ▶ recherches récentes → recommandations
- ▶ caddie
- ▶ statistiques comportementales
- ▶ corrélation de requêtes
- ▶ **contrôle d'accès**
- ▶ ...



# Identification

- ▶ identifiant fourni (**Authorization**, **From**)
- ▶ cookie (uuid)  
**Set-Cookie** :  
`user=1f1d86ca-7080-403b-89e9-bc649dd7887e;Expires=Tue, 01-Jan-2036 08:00:01 GMT`
- ▶ certificat
- ▶ implicite
- ▶ ...

## Définition (Authentification)

garantir l'identité

### Exemple

- ▶ mot de passe (HTTP, sessions)
- ▶ jeton
- ▶ certificat / clé
- ▶ tiers de confiance (OAuth, OpenID, ...)

## Définition (Autorisation)

contrôle d'accès : défini ce que l'utilisateur peut faire

⇒ ACL (*access control list*)

groupe, rôle (RBAC)

attributs (ville, service, etc.), règles métier (ABAC), ...

- ▶ applicatif / métier
- ▶ HTTP (frontal)

403 Forbidden

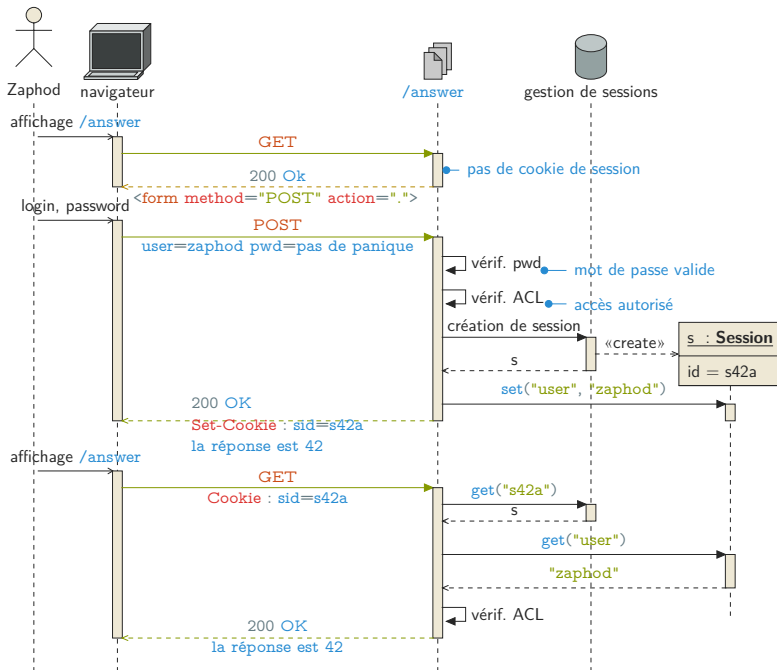
Conf. Apache (fichiers statiques, reverse proxy, ...) :

```
<Location /foo>  
  <Limit GET>  
    Require valid-user  
  </Limit>  
  <Limit POST>  
    Require group editors  
  </Limit>  
  <Limit DELETE>  
    Require user zaphod trillian  
  </Limit>  
</Location>
```

<https://httpd.apache.org/docs/2.4/fr/howto/access.html>

- ▶ ad hoc
- ▶ code ressource
- ▶ état coté serveur → session

Version simple



```
GET /answer HTTP/1.1
```

```
Host : h2g2.com
```

```
HTTP/1.1 200 OK
```

```
Content-Type : text/html; charset=UTF-8
```

```
...
```

```
<form action="/answer" method="POST">
```

```
  <input name="u" placeholder="Identifiant" type="text" />
```

```
  <input name="p" placeholder="Mot de passe" type="password"/>
```

```
</form>
```

```
...
```



POST /answer HTTP/1.1

Host : h2g2.com

u=zaphod&p=pas%20de%20panique

HTTP/1.1 200 Ok

Set-Cookie : sid=s42a ; path=/ ; domain=.h2g2.com ; Secure ; HTTPOnly

Content-Type : text/plain

Content-Length : 31

Hello Zaphod, the answer is 42 !

GET /answer HTTP/1.1

Host : h2g2.com

Cookie : sid=s42a

HTTP/1.1 200 Ok

Set-Cookie : sid=s42a ; path=/ ; domain=.h2g2.com ; Secure ; HTTPOnly

Content-Type : text/plain

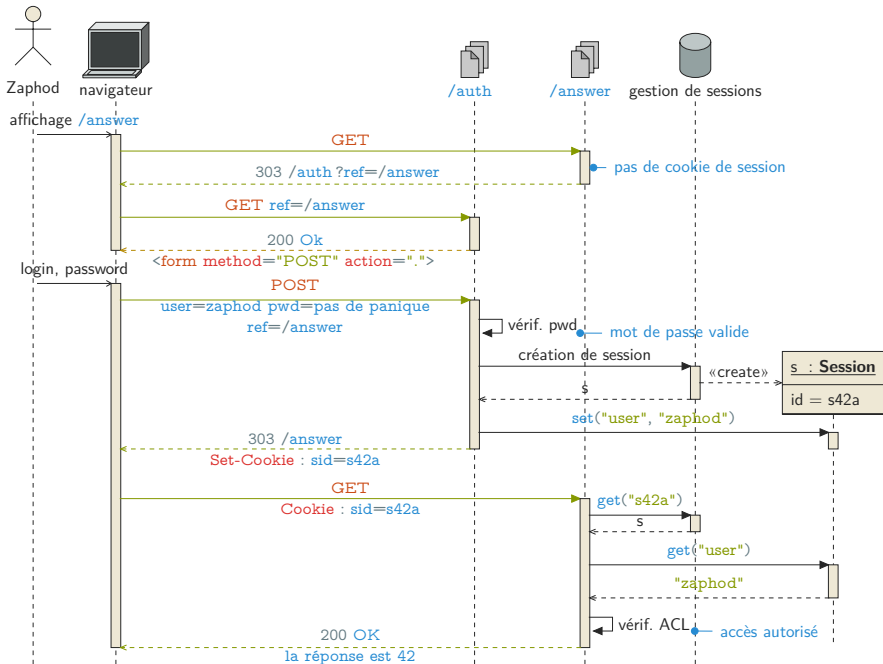
Content-Length : 31

Hello Zaphod, the answer is 42 !

# Version simple

- ▶ ✓ interaction simple
- ▶ ≈ gestion pour toutes les ressources
- ▶ ✗ 200 **Ok** mais pas contenu
- ▶ ✗ « gaspille » **POST** (ou surcharge)
- ▶ ✗ ...

Version élaborée



GET /answer HTTP/1.1

Host : h2g2.com

HTTP/1.1 303 See Other

Location : /auth?ref=%2Fanswer

GET /auth?ref=%2Fanswer HTTP/1.1

Host : h2g2.com

```
HTTP/1.1 200 OK
```

```
Content-Type : text/html; charset=UTF-8
```

```
...
```

```
<form action="/auth" method="POST">
```

```
  <input name="u" placeholder="Identifiant" type="text" />
```

```
  <input name="p" placeholder="Mot de passe" type="password"/>
```

```
  <input name="r" type="hidden" value="/answer"
```

```
</form>
```

```
...
```

```
POST /auth HTTP/1.1
```

```
Host : h2g2.com
```

```
u=zaphod&p=pas%20de%20panique&r=%20Fanswer
```

HTTP/1.1 200 OK

Content-Type : text/html ; charset=UTF-8

...

```
<form action="/auth?ref=%2Fanwser" method="POST">
```

```
  <input name="u" placeholder="Identifiant" type="text" />
```

```
  <input name="p" placeholder="Mot de passe" type="password"/>
```

```
</form>
```

...

POST /auth?ref=%2Fanswer HTTP/1.1

Host : h2g2.com

u=zaphod&p=pas%20de%20panique



HTTP/1.1 303 See Other

Location : /answer

Set-Cookie : sid=s42a ; path=/ ; domain=.h2g2.com ; Secure ; HTTPOnly

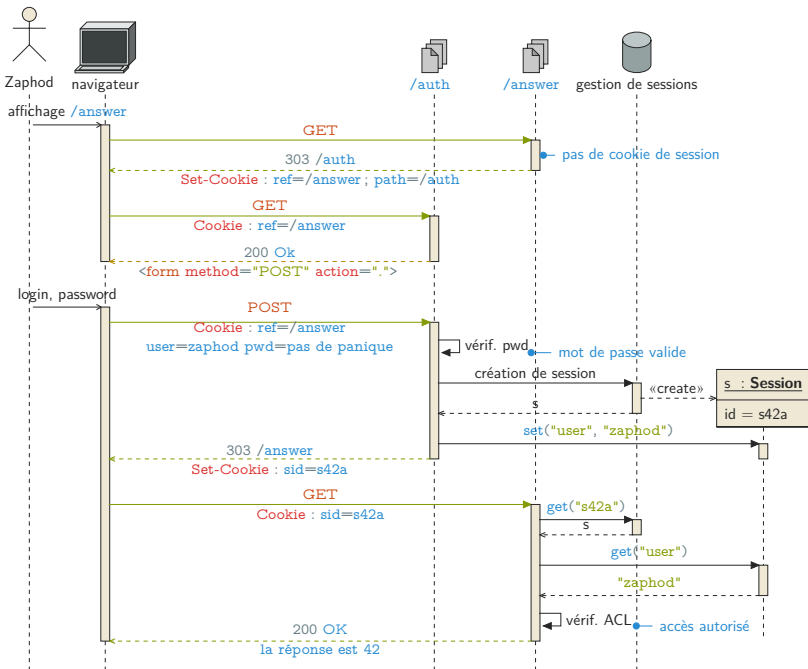
GET /answer HTTP/1.1

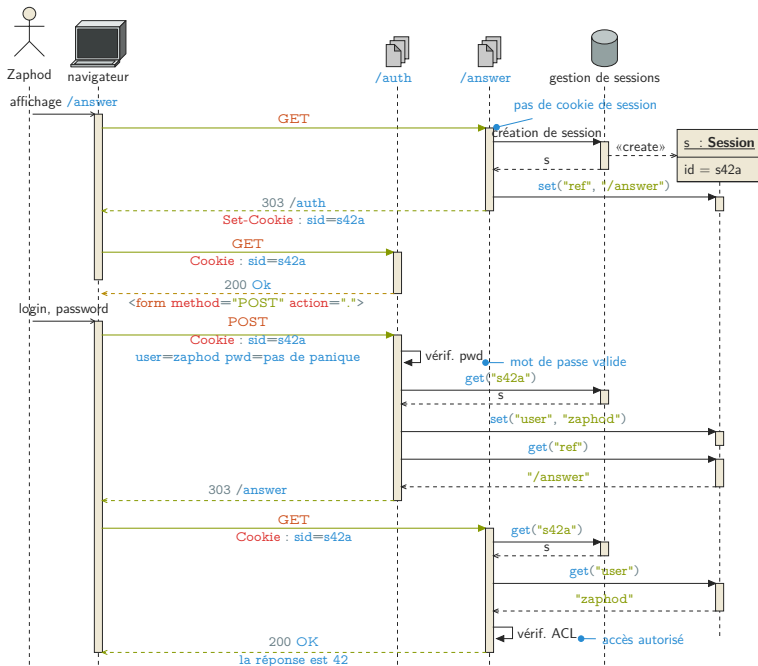
Host : h2g2.com

Cookie : sid=s42a

# Variantes

- ▶ origine dans un cookie
- ▶ session immédiate





# Formulaire et session

- ▶ ✓ formulaire personnalisé
- ▶ ✓ page oubli / création
- ▶ ≈ authentification persistante
- ▶ ✗ automatisation
- ▶ ✗ cookie
- ▶ ✗ clair  $\Rightarrow$  TLS
- ▶ ✗ vol de session (e.g. Wireshark, FireSheep)

## RFC 7235

- ▶ standard
- ▶ générique
- ▶ sans état
- ▶ HTTP : (rev. prox., app. serv.)

- ▶ **WWW-Authenticate** : type, realm
- ▶ **Authorization** : type, jeton
- ▶ 401 **Unauthorized**

# Types

- ▶ Basic : clair (RFC 7617)
- ▶ Digest : chiffré (RFC 7616)
- ▶ Bearer : opaque (RFC 6750)
- ▶ autres...

<http://www.iana.org/assignments/http-authschemes/>



# Basic

Jetton : `base64(user + ' ' + password)`

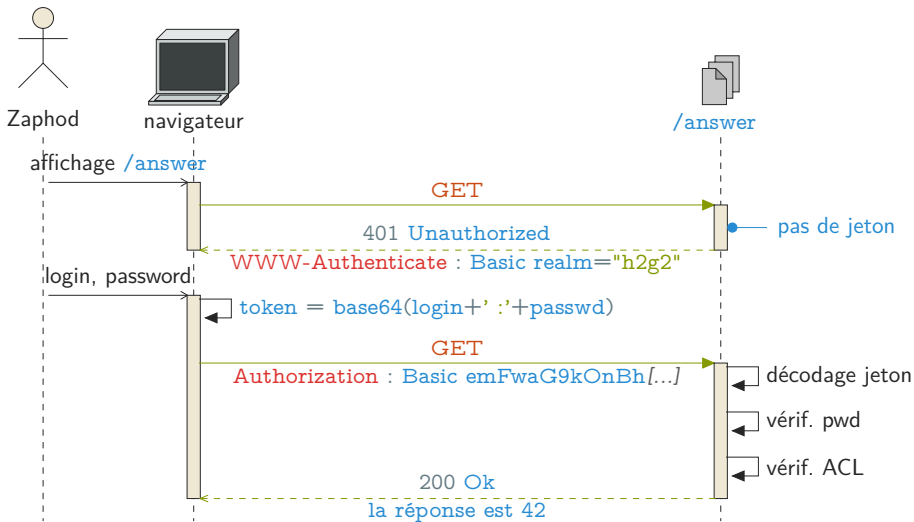
## Exemple

**WWW-Authenticate** : Basic realm="h2g2"

- ▶ user : `zaphod`
- ▶ password : `pas de panique`
- ▶ `base64("zaphod :pas de panique")` →  
`emFwaG9kOnBhcyBkZSBwYW5pcXVl`

**Authorization** : Basic `emFwaG9kOnBhcyBkZSBwYW5pcXVl`

# Basic




# Digest

- ▶ similaire Basic
- ▶ `nonce`
- ▶ `response` : md5
  - ▶ user
  - ▶ password
  - ▶ method
  - ▶ URI
  - ▶ nonce
  - ▶ compteur, ...

- ▶ ✓ sans état, générique
- ▶ ✓ automatisation (robots, .netrc)
- ▶ ✓ sur un frontal (rev. proxy)
- ▶ ≈ oubli / création → page 401/403 personnalisée
- ▶ ✗ « formulaire » natif

localhost:8080/protected

**Authentication requise**

 Le site http://localhost:8080 demande un nom d'utilisateur et un mot de passe. Le site indique : « here and now »

Utilisateur :

Mot de passe :

localhost:8080/protected

**Ouvrir une session**

http://localhost:8080

Nom d'utilisateur

Mot de passe

- ▶ authentification ad-hoc
- ▶ spécifique à une application

- ▶ **X** don't reinvent the wheel
- ▶ **X** don't repeat yourself
- ▶ **X** intégration, portail, SSO
- ▶ **X** sécurité
- ▶ ...

legacy

mot de passe

- salé
- haché

 jamais en clair

## haché (fonction de hachage crypto)

- ▶ « unique »
- ▶ non réversible
- ▶ ❌ MD5
- ▶ ≈ SHA-1
- ▶ ✔ SHA-256



## Exemple

« pas de panique »

► SHA1 : `be38f00728a47a1c205f6b06946339085b026722`

► SHA256 :

`e211cdd14efdeef401940873725896a9aef63fbb5fcb19600bdcbac11807`

 attaque par dictionnaire / rainbow table

sel

- ▶ chaîne quelconque
- ▶ appli ou utilisateur
- ▶ stockée à part
- ▶ ajoutée au mot de passe avant hachage

## Exemple (SHA-1)

- ▶ « pas de panique » →  
be38f00728a47a1c205f6b06946339085b026722
- ▶ + « \$marvin ! »
- ▶ « pas de panique\$marvin ! » →  
3e525b6df88d2eaae52af08ea435134663507368

⇒ bcrypt

# Vérification

## Exemple

- ▶ user : `zaphod`, password : `pas de panique`  
(HTTP Basic, formulaire HTML, ...)
- ▶ `h = select passHash from users where login = 'zaphod'`
- ▶ `assert h == sha1("pas de panique$marvin")!`

- ▶ SSL : Secure Socket Layer (RFC 6101)
- ▶ TLS : Transport Layer Security (RFC 5246)
- ▶ https : (RFC 7230, RFC 7231)

entre TCP et protocole applicatif (HTTP)

# HTTP

RFC 2817, RFC 2818

- ▶ 101 Switching Protocols
- ▶ 426 Upgrade Required
- ▶ Upgrade

forcer le chiffrement



GET / HTTP/1.1

Host : [www.example.com](http://www.example.com)

Upgrade : TLS/1.2

Connection : upgrade

HTTP/1.1 101 Switching Protocols

Upgrade : TLS/1.2, HTTP/1.1

Connection : upgrade

## chiffrement asymétrique

- authentification du serveur
- chiffrement des communications
- intégrité des données
- authentification du client

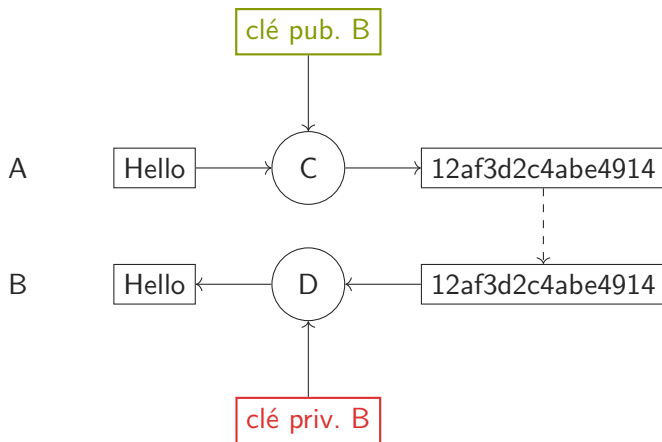
# Principe

2 clés :

- ▶ clé publique : peut être divulguée
- ▶ clé privée : gardée secrète (mot de passe)

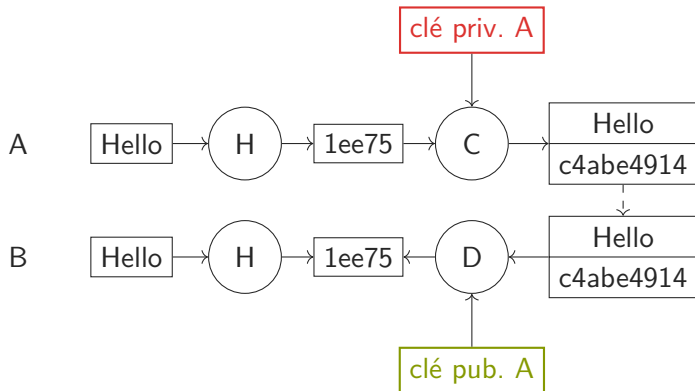
# Chiffrement

garantir le destinataire

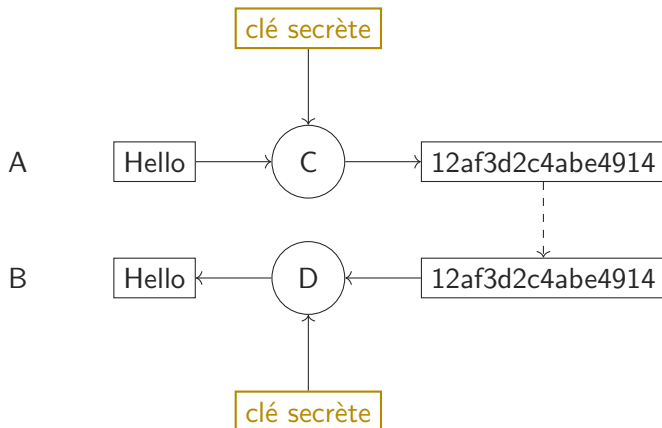


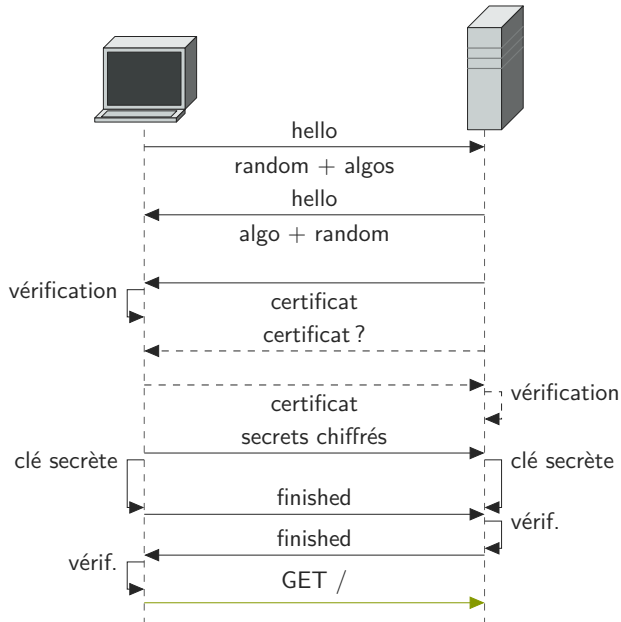
# Signature

garantir l'expéditeur / contenu



# Chiffrement symétrique





certificat : X.509

- clé publique
- + méta-données (host, utilisation)
- signé par CA (certification authority)

→ PKI (Public Key Infrastructure)



- ▶ ≈ certificats auto-signés
- ▶ ≈ intermédiaires : [CONNECT](#)
- ▶ ≈ virtual host
- ▶ ≈ mixed content