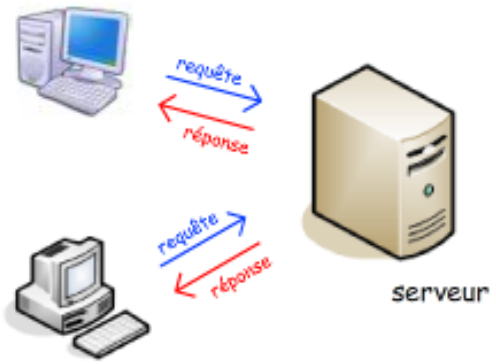
Fiche de révisions - Architecture client-serveur

Rédigé par : Jimmy Paquereau

1. Introduction

1.1. Architecture client-serveur : cas général



De manière simpliste, **un serveur est une machine informatisée quelconque fournissant un ou plusieurs services à des consommateurs**. **Un client est alors une tierce machine informatisée quelconque pouvant consommer ces services**.

Soyons à présent plus exact. Dans une architecture client-serveur, un serveur est un logiciel ou un ensemble de logiciels installés sur une machine informatisée, et ce logiciel fournit un ensemble de services disponibles sur un réseau. **Les machines informatisées connectées au réseau s'appellent des hôtes**. Ce réseau peut être un simple **réseau local** (appelé **LAN**, Local Area Network), **internet**. Il existe divers types et diverses topologies de réseaux. Sur un LAN, les machines sont de nos jours reliées entre elles par un ou plusieurs **switch** (= **commutateur**). Les machines sont reliées à internet par un **routeur**, et le plus souvent même par un routeur/NAT. Afin d'accéder aux services proposés par un serveur, les hôtes ont besoin de logiciels, appelé clients, capable de communiquer avec ces services.

Afin de discuter entre eux, clients et serveur utilisent des protocoles. **Un protocole définit un ensemble de règle à adopter en vue d'échanger des informations. Il définit en particulier le format des informations échangées, qu'on appelle messages**. En règle générale, parmi ces messages, **on distingue les requêtes et les réponses**. Le client est celui qui demande (envoi de requêtes) et le serveur celui qui répond (envoi de réponses). Finalement, les échanges entre clients et serveur sont portés par un protocole.

Il existe de nombreux protocoles afin de répondre à la variété des services dont nos machines informatisées (ordinateurs, imprimantes, téléphones, fax, switch, routeur…) peuvent avoir besoin :
o **FTP**, SFTP : protocoles destinés au stockage, à l'échange, au partage de fichiers (exemple de client FTP : FileZilla) ;
o **SMTP**, POP, IMAP : protocoles destinés aux échanges relatifs aux boîtes mails (exemple de client SMTP : MS Outlook, exemple de serveur SMTP : MS Exchange) ;
o **HTTP** : protocole destiné aux échanges de données et ressources hétérogènes au travers d'intranet (exemple de client http : Chrome, exemples de serveur web : Apache, Tomcat, JBOSS…) ;
o **HTTPS** : protocole destiné à sécuriser les échanges HTTP (c'est ce qu'on appelle une surcouche, en l'occurrence un protocole qui vient par-dessus un protocole) ;
o TCP/IP, DHCP, TELNET, NTP, SNMP, SVN, GIT…

Quoiqu'il en soit, dès qu'il est question d'architecture client-serveur, il est question de **partage de ressources** !

Un détail nous a échappé. Lorsque l'on demande un service, à savoir lorsque l'on envoie une requête, il faut pouvoir identifié non seulement le **serveur physique** (la machine) auquel on s'adresse, mais encore le **serveur/service logiciel**. Nous le savons, une **adresse IP** (exemple : 34.52.12.16) publique ou un **nom de domaine** (exemple : youtube.com) permet d'identifier une machine, sur un réseau local ou sur internet. Mais un serveur physique peut fournir plusieurs services : il peut être à la fois un serveur web, FTP, GIT et SMTP, ce

qui est très loin d'être rare. Aussi, à chaque serveur logiciel, on fait correspondre un **numéro de port**. Le couple **adresse ip** + **port** ou celui **nom de domaine + port** permet d'identifier le service (serveur physique + serveur/service logiciel).

1.2. Architecture client-serveur : cas d'une architecture web

Désormais, nous allons étudier la notion d'architecture client-serveur au travers de la notion d'architecture web et d'application web client/serveur.

1.2.1. Application client-serveur

Qu'est-ce qu'une **application client-serveur** ? Un logiciel réalisé en architecture client-serveur est un programme qui s'exécute en partie côté client (à savoir du côté de l'utilisateur du logiciel) et en partie côté serveur. Dans le cadre d'une application web, le programme s'exécute en partie sur le navigateur de l'utilisateur (côté client) et en partie côté serveur web. Dans une architecture web, les clients, ce sont les navigateurs. Le serveur, c'est le serveur web, lequel fournit diverses ressources.

1.2.2. Client web : le navigateur

Qu'est-ce qu'un **navigateur** ? Un « vrai » navigateur (chrome, internet explorer…) est un logiciel particulièrement complexe qui assure les fonctionnalités suivantes :

o **client HTTP** : un navigateur est capable de discuter avec un serveur web au moyen du protocole HTTP. Il est ainsi capable de demander des ressources et d'envoyer des données via ce protocole. On parle de **requêtes HTTP**. HTTP est un protocole à part entière. Il ne se résume pas à l'utilisation de simples URL telles que http://www.google.com/ ;

o **moteur de rendu** : un navigateur est capable d'effectuer le rendu de pages web, c'est-à-dire de calculer un affichage complexe, à partir de la description de ces pages. Le code HTML décrit la structure d'une page. Le CSS décrit la mise en forme de la page. D'autres langages viennent éventuellement s'additionner aux deux autres : SVG (description de dessins 2D), flash (en voie de disparition)… ;

o **interpréteur JavaScript** : un navigateur est capable d'interpréter (= exécuter) le langage de programmation appelé JavaScript, devenu depuis maintenant des années un langage de programmation à part entière ;

o les navigateurs récents intègrent de nombreuses autres fonctionnalités telles que : moteur de rendus 3D pour l'affichage de scènes 3D (standard WebGL), moteur de rendu PDF permettant l'affichage de document en format PDF…

1.2.3. Serveur web

De même qu'il existe de multiples navigateurs, ils existent de multiples serveurs web : Apache, Tomcat, JBOSS… Un serveur web est généralement accessible par défaut via le port 80. S'il n'est pas démarré sur le port 80, mais par exemple le port 8080, il est nécessaire de le préciser dans les URL côté client (exemple : http://www.google.com:8080/).

Qu'est-ce qu'un **serveur web** ? Un serveur web implémente le protocole HTTP et est capable de fournir des réponses HTTP. Sur un pareil serveur, on peut déployer des applications web : un site internet, un intranet, un site e-commerce… Un serveur web permet en général de développer des pages web dans un langage de programmation donné, interprété ou exécuté côté serveur. Ce langage de programmation est communément du PHP ou encore du Java/JEE, on parle alors respectivement de serveur web PHP et de serveur web Java/JEE. Mais il existe encore un serveur web .NET (Microsoft) voire un serveur web JavaScript (NodeJS).

1.2.4. Protocole HTTP

Qu'est-ce que le **protocole HTTP** ? Tout d'abord, c'est un protocole. Un protocole, c'est une façon de discuter en respectant un format d'échange donné compris par les intéressés. Les convenances humaines : bonjour, au revoir…, constituent en quelque sorte un protocole. HTTP (et **HTTPS**, version sécurisée du protocole HTTP) fonctionne sur le principe des requêtes/réponses :
o le client HTTP envoie une requête HTTP au serveur web ;
o en réponse à la requête HTTP, le serveur web retourne une réponse HTTP au client ;
o le protocole HTTP décrit le format des messages (requête HTTP et réponse HTTP) à utiliser par les clients HTTP et les serveurs web.

Qu'est-ce qu'une **requête HTTP** ? Une requête HTTP est un message divisé en deux parties : les en-têtes (les **headers**) et le corps (le **body**). C'est une demande que l'on formule auprès d'un serveur web. Cette demande vise à appeler un service web ou encore à récupérer une ressource quelconque (image, vidéo, html…). Une telle requête comporte diverses informations, en voici quelques-unes :

o une **URL**, laquelle permet d'identifier un service à appeler (traitement) ou une ressource à récupérer (fichier PDF, page HTML pure, image…) ;

Exemple d'URL
https://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol
- www.wikipedia.org (≈ wikipedia.org) est ce qu'on appelle le **nom de domaine** ;
- fr.wikipedia.org est ce qu'on appelle est un sous-domaine (de wikipedia.org) ;
- /wiki/Hypertext_Transfer_Protocol est ce qu'on appelle une URI

o une **méthode** qui détermine si oui ou non des données sont transmises et comment. Les méthodes les plus courantes sont les méthodes **GET**, **POST**, **DELETE** et **PUT**.

o aucun à plusieurs **paramètres**. On distingue entre autre les paramètres **GET** et les paramètres **POST**. Ce sont des données transmises dans la requête. Les paramètres GET figurent directement dans l'URL. Les paramètres POST sont « encapsulés » dans le corps de la requête. C'est typiquement au moyen de paramètres POST que des formulaires sont soumis, envoyés, à un serveur web.

Exemple d'URL comportant des paramètres GET
https://www.google.fr/search?q=http&oq=http&aqs=chrome..69i57j69i60l5.3269j0j9
&sourceid=chrome&es_sm=93&ie=UTF-8

Signification :
- c'est l'URL qui est transmise à un serveur web de Google lorsque vous effectuez la recherche « http » sur le moteur de recherche de Google ;
- on retrouve un nom de domaine : www.google.fr, ainsi qu'une URI : /search ;
- après le caractère « ? », on trouve ce qu'on appelle une *query string* (chaîne de requête) dans laquelle figurent les fameux paramètres GET. Un paramètre a un nom et une valeur. Ici les paramètres sont les suivants :
o les paramètres **q** et **oq** (q pour « query ») qui ont pour valeur **http** : ce paramètre indique vraisemblablement au moteur de recherche hébergé sur un serveur web de Google la recherche que vous effectuez ;
o les paramètres **aqs**, **es_sm** et **ie**, qui ont pour valeur respective **chrome..69i57j69i60l5.3269j0j9**, **93** et **UTF-8**, et lesquels ne nous intéressent guère ;
o le paramètre **sourceid**, ayant ici pour valeur **chrome**, lequel indique à partir de quel navigateur la requête a été envoyée.

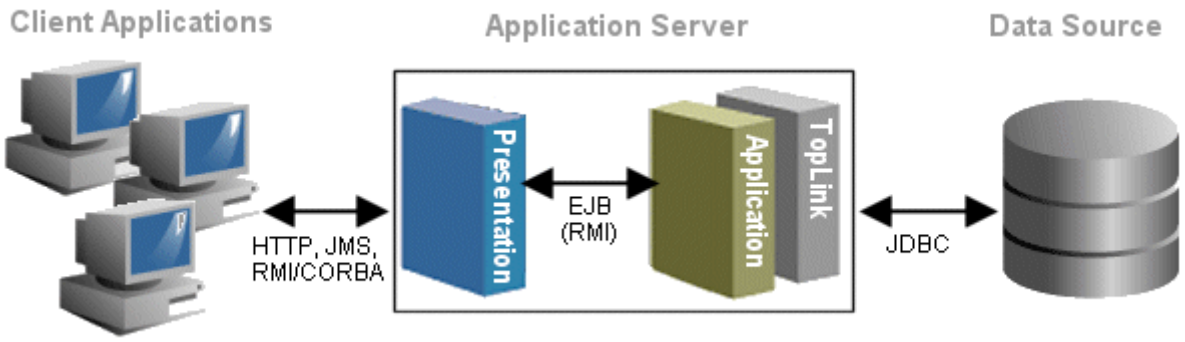
Quelles informations comporte une **réponse HTTP** ? Une réponse HTTP est également divisée en **headers** et **body**. Il s'agit d'un message retourné à un client HTTP en réponse à sa requête. Le corps de la réponse représente un contenu quelconque : un résultat, une page web, une ressource (musique, image, feuille de style…). Une telle réponse comporte également de multiples informations, en voici quelques-unes :

o un **code de statut** (*status code*) indiquant le type de réponse fourni. Il existe des codes d'erreurs tels que la célèbre **erreur 404** (page non trouvée), l'**erreur 500** (erreur serveur), des codes de succès, de redirection… Et chaque code a une signification qui lui est propre ;
o un **type MIME** (type de contenu, appelé *content type*) qui indique au client HTTP le type de contenu encapsulé dans le corps de la page (exemples : music/mp3, video/mp4, image/jpeg…) ;
o un **corps** (le *body*) contenant la ressource, les données de la réponse.

1.2.5. Système de Gestion de Base de Données (SGBD)

Une architecture juste client-serveur est ce qu'on appelle une **architecture 2-tiers**. Le premier tiers, ce sont les clients. Le deuxième, c'est le serveur.

Or, très couramment, une application client-serveur va manipuler une base de données logée sur un SGBD (en PHP, MySQL par exemple), le SGBD n'étant pas nécessairement installé sur le même serveur physique. On parle alors d'architecture 3-tiers, que l'on représentée ainsi :



Clients et application serveur discutent entre eux via un protocole, le protocole HTTP. Application serveur et SGBD discutent entre eux via un autre protocole, JDBC par exemple (autre exemple : ODBC, ADO…).

1.3. Architectures N-tiers

En pratique, la réalité est bien souvent plus complexe, et on parlera **d'architecture N-tiers**, soit autant de tiers qui, pour une architecture donnée, discutent entre eux. En effet, il est très courant d'avoir besoin de services fournis par de multiples serveurs. On délègue par exemple la gestion des authentifications à un annuaire LDAP sur un serveur LDAP, la récupération d'informations cartographiques à Goolge Maps ou autre, l'envoi de mails à un serveur OVH…

Egalement, des raisons que nous étudierons dans un prochain chapitre peuvent nous pousser à adopter une architecture plus complexe :
- la **tolérance aux pannes**, à savoir la capacité tout à la fois de ne pas perdre ses informations en cas de panne et d'assurer la **continuité de service**. Diverses notions font alors leur apparition : les notions de **sauvegarde**, de **stratégie de sauvegarde** et de **réplication**, mais encore les concepts de **répartition** (***load balancing***) ;
- la **scalabilité**, c'est-à-dire la capacité à supporter la **montée en charge**, à savoir la capacité pour un serveur ou un logiciel à supporter des connexions nombreuses, un trafic élevé. Le concept de *load balancing* fait encore une fois son apparition.

Nous étudierons ces concepts dans un chapitre prochain.

1.4. Intérêt

L'intérêt d'une architecture client-serveur, ou plus généralement d'une architecture N-tiers, réside dans ce que celle-ci permet de partager des ressources. En cela, elle facilite le travail collaboratif. Plus encore, une telle architecture permet de centraliser des services et, ainsi, d'éviter la prolifération de logiciels potentiellement hétérogènes sur les machines d'un réseau. Une telle prolifération s'avèrerait particulièrement difficile à maintenir et à faire évoluer. Autrement dit, cela facilite encore la maintenance et l'évolution du système d'information.

2. Application web : généralités

De Youtube à Linkedin en passant par les ENT ou Amazon, nous utilisons désormais des applications web tous les jours. Toutes ces applications sont des logiciels en architecture client-serveur, voire en architecture N-tiers. Ici, nous allons nous intéresser à ces applications, à leur fonctionnement et à quelques-uns des technologies qu'elles utilisent inévitablement ou utilisent potentiellement.

Cette partie est une première approche, une initiation. En pratique, d'autres concepts sont nécessaires en vue de développer une application client-serveur, web ou non web, digne de ce nom.

Une application web comporte vulgairement une **partie client** et une **partie serveur**. On parle également de contenus statiques et de contenus dynamiques :
o les contenus statiques sont les ressources que le serveur web peut retourner sans intervention particulière, plus exactement sans qu'il n'ait à exécuter ou interpréter ces contenus : les images, les vidéos, les pages HTML, les feuilles de style CSS, les programmes JavaScript… Tous ces contenus, ces ressources, sont traités par le navigateur (**côté client**). Le serveur se contente d'en assurer la transmission via le protocole HTTP, ce qui n'est déjà pas rien en soi ;
o les contenus dynamiques sont les programmes interprétés ou exécutés **côté serveur**. Sur un serveur web PHP, il s'agira des fichiers PHP (extension : .php). Le PHP est un langage interprété. Il est interprété tel quel. Sur un serveur web Java/JEE, il s'agira d'un « exécutable » Java (extension : .war en web, .jar en non web). Le Java est un langage hybride. Le Java est exécuté après avoir été compilé dans un langage proche du binaire (le *ByteCode*). Le programme retourne potentiellement tous types de contenus/ressources : une page HTML, des données XML, un PDF, des données JSON…

Remarque : une simple erreur dans un code source (HTML, CSS, JavaScript, PHP…) et votre programme ne fonctionne potentiellement plus en tout ou partie. Ne pensez pas qu'un ordinateur est là pour penser pour vous lorsqu'il interprète ou exécute un programme. Un ordinateur ne pense pas. Vous êtes là pour cela.

3. Application web : technologies

3.1. Le HTML

Le HTML est un **langage de description**. Sa version actuelle est le HTML5. Il ne s'agit pas d'un langage de programmation ! Un fichier HTML (extension : .html typiquement) est qualifié de page HTML. Il permet de décrire la structure d'une page web. C'est un **langage balisé**, un langage à balises, au même titre que l'est le XML. Son unité élémentaire est la **balise**, aussi appelée **marqueur** (de l'anglais *marker*). Une balise a un nom

et aucun ou plusieurs **attributs**. Une page HTML prend la forme d'un **arbre**, ou chaque élément s'appelle un **nœud**.

Il existe des **balises ouvrantes** (exemple :) et des **balises fermantes** (exemple). Une balise peut aussi être **à la fois ouvrante et fermante** (exemple : <input/>). Le couple balise ouvrante + balise fermant constitue un nœud.

Structure d'une page HTML 5 :

Code HTML	Explications

`<!doctype html>`
`<html lang="fr">`
`  <head>`
`  <meta charset="utf-8">`
`  <title>Titre de la page</title>`
`  <link`
`  rel="stylesheet"`
`  href="style.css"`
`  />`
`  <script src="script.js"></script>`
`  </head>`
`  <body>`
`  ...`
`  <!-- Le reste du contenu -->`
`  ...`
`  </body>`
`</html>`
	o l'annotation < !doctype html> indique que la page est en HTML5 ;
	o le nœud <html></html> délimite la page HTML et l'attribut lang permet de préciser la langue[1] dans laquelle la page web est écrite ;
	o la nœud <head></head> délimite l'en-tête de la page, à savoir un ensemble d'informations non affichées au sein de la page web ;
	o la balise <meta/> permet de préciser une métadonnée (donnée connexe), en l'occurrence le *charset*, i.e. le jeu de caractères utilisé[2] ;
	o le nœud <title></title> contient le titre de la page, à savoir le titre apparaissant dans l'onglet sur votre navigateur ;
	o la balise <link> permet entre autre d'importer une feuille de style CSS ;
	o la balise <script> permet entre autre d'importer un script JavaScript qui est exécuté par le navigateur ;
	o le nœud <body></body> délimite le corps, le contenu apparent d'une page web ;
	o < !-- Le reste du contenu --> est un commentaire. C'est la syntaxe à respecter pour insérer des commentaires en HTML.

[1] Tous les codes de langue sont normalisés : norme ISO-639-1

[2] Un jeu de caractères est une façon d'encoder les caractères textuels. De nos jours, le format standard et à privilégier est utf8. Il en existe d'autres : ASCII, ANSI…

Vous comprendrez dès lors qu'il importe, afin de ne pas se perdre dans son code, de **bien l'indenter** (les fameux décalages).

3.2. Le CSS

Le CSS est aussi un langage de description, et n'est donc pas un langage de programmation. Un fichier CSS (extension : .css) est qualifié de **feuille de style**. Même s'il est possible d'intégrer du CSS directement au sein d'une page HTML, cette pratique est dépréciée et à proscrire. Le CSS permet de décrire la mise en forme d'une page HTML. Le CSS introduit la notion de sélecteur (**sélecteur CSS**). Un sélecteur CSS permet de « sélectionner » un nœud d'une page HTML. Une fois ce nœud sélectionné, on en décrit la mise en forme à partir de **propriétés CSS** (styles CSS).

Ci-après, quelques exemples simples d'utilisation du CSS.

Quelques exemples élémentaires :

Code HTML

```
< !-- Un bloc html -->
<div class="carre-rouge"></div>
```

Résultat

Mise en forme CSS

```
/**
 * Sélection d'un nœud HTML
 * en fonction de sa classe de style
 **/
.carre-rouge{
   background: rgb(255,0,0);
   color: rgb(255,255,255);
   border: solid 1px #000000;
   height: 10px;
   width: 10px;
}
```

Code HTML

```
< !-- Un champ text html -->
<input id="champText" type="text">
```

Résultat

Mise en forme CSS

```
/**
 * Sélection d'un nœud HTML
 * en fonction de son id
 **/
#champText{
   border: solid 1px #dcdcdc;
   border-radius: 3px;
   background: #f5f5f5;
   color: #444444;
}
```

On aura à peu près compris la signification de chacune des propriétés CSS : border, background, color…
Ce qui reste potentiellement obscur, ce sont les éléments suivants :
o **px** : c'est un nombre de pixel, à savoir un nombre de point à l'écran. C'est une unité permettant de définir une distance à l'écran (longueur, largeur, épaisseur…). Il existe d'autres unités, dont % et em, souvent privilégiées ;
o #444444, #f5f5f5, rgb(255,0,0)… : ce sont des codes couleurs en composantes, i.e. des couleurs définies en fonction de leur quantité de rouge (*red*), de vert (*green*) et de bleu (*blue*). La valeur de chaque composante varie entre 0 et 255. rgb(255,0,0) signifie qu'il n'y a que du rouge. Le code #ffaa00 peut être décomposé en trois nombres hexadécimaux : ff, aa et 00, qui valent respectivement 255, 170 et 0. Cela nous donne une couleur composée de beaucoup de rouge, pas mal de vert et pas de bleu. #ffaa00 correspond donc à un orange.

3.3. Le JavaScript

Le JavaScript est cette fois-ci un langage de programmation. Il est interprété côté client, c'est-à-dire interprété par le navigateur. Ici, nous ne parlerons que de JavaScript en l'utilisant avec jQuery. jQuery est une librairie facilitant l'utilisation du JavaScript. Bien qu'il soit possible d'intégrer du JavaScript directement dans une page HTML, c'est une pratique à proscrire. Un programme JavaScript est normalement développé dans un fichier JavaScript (extension : .js).

JavaScript est devenu un langage de programmation à part entière. On y retrouve les notions habituelles de la programmation procédurales : variables, branchement conditionnels SI/SINON SI/SINON, boucles POUR, TANTQUE… De fait, JavaScript dépasse le cadre de la programmation procédurale. C'est un langage dit à prototypes. Il fait presque partie de ce qu'on appelle les langages orientés objets.

Par ailleurs, JavaScript permet entre autre de manipuler les éléments d'une page HTML. Il permet par exemple de changer de changer le contenu HTML d'un nœud d'une page HTML, d'ajouter des attributs à un nœud, de supprimer un nœud, de modifier les mises en forme d'un nœud. Il permet d'exécuter des traitements différés, à intervalles réguliers et permet bien d'autres choses.

On notera également que JavaScript est un langage laxiste dans le sens en outre où il est faiblement typé. En effet, lorsqu'on déclare une variable en JavaScript, il n'est pas utile d'en préciser le type (entier, chaîne de caractères…). L'interpréteur se débrouille.

jQuery est là pour nous faciliter la vie. Il intègre la même notion de sélecteur que celle utilisée en CSS. On peut ainsi sélectionner et manipuler facilement des éléments d'une page HTML.

Ci-après, un exemple élémentaire :

Code HTML

```
< !-- Un bloc html -->
<div class="carre-rouge"></div>
```

Résultat

Notre carré de tout à l'heure clignote toutes les 1000 millisecondes (=1sec).

Explication : toutes les 1sec, on appelle la fonction clignoter. Si la variable affiche valait true, elle vaut false, si elle valait false, elle vaut true. Par conséquent, un coup on affiche, un coup on cache.
Le caractère « ! » retourne la négation d'une variable (la négation de « oui », c'est « non », et inversement).

Algorithme JavaScript/jQuery

```
// Procédure faisant clignoter le carré
function clignoter(affiche){
   // sélectionne le nœud carre-rouge
   var carre = $(".carre-rouge");
   // si affiché, on cache le carré
   if(affiche){
      carre.hide();
   // sinon on affiche le carré
   }else{
      carre.show();
   }
   // appelle une fonction (anonyme)
   // après un délai de 1000ms
   setTimeout(function(){
      clignoter(!affiche);
   }, 1000);
}
```

3.4. Le PHP

PHP est un langage de programmation des plus classiques et tout particulièrement utilisé dans le monde du web. De même que JavaScript est équipé de multiples librairies telles que jQuery (on parle de *frameworks*), PHP est en pratique utilisé à l'aide de librairies. Son concurrent principal est Java/JEE.

Avec PHP, on se heurte cette fois-ci à un véritable langage de programmation. Il fait clairement partie des langages de programmation orientés objets. En PHP, on peut en quelque sorte « tout faire » : générés du PDF, envoyer des mails, interagir avec une base de données, retourner une page HTML contenant des données en provenance d'une base de données…

En PHP, on peut bien entendu utiliser des boucles, des variables, des conditions… Lui aussi est un langage faiblement typé où il n'est pas nécessaire de préciser le type des variables. C'est un langage interprété. Il est interprété par les serveurs web utilisant un interpréteur PHP.

A la page suivante, on trouvera un exemple d'algorithme rédigé en PHP.

Exemple d'utilisation de PHP : construction d'une liste de produits, retournée au navigateur, qui peut l'afficher.

```php
<?php
// déclaration d'un utilisateur et d'un mot de passe ayant accès à la base de
// données locale « test »
$user = 'root';
$password = '';
// récupération d'une connexion à la base de données « test »
$connection = new PDO('mysql:host=localhost;dbname=test', $user, $password );
// récupération d'une liste de produits
$produits = $connection->query('SELECT * FROM Produit';
// parcours de la liste des produits (se lit : pour chaque produits produit)
foreach($produits as $i => produit) {
   // écrit la référence et la désignation d'un produit
   echo 'Produit n°' . $produit->reference . ' : ' . $produit->designation;
   // écrit un retour à la ligne HTML
   echo '<br/>';
}
// ferme la connexion
$dbh = null;
```