Le web, comment ça marche ?

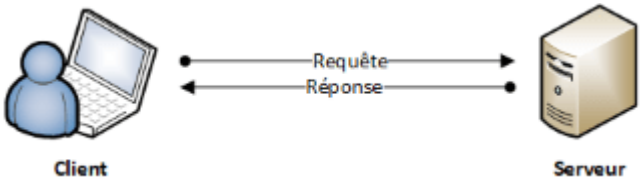
Table des matières

1. L'architecture client-serveur

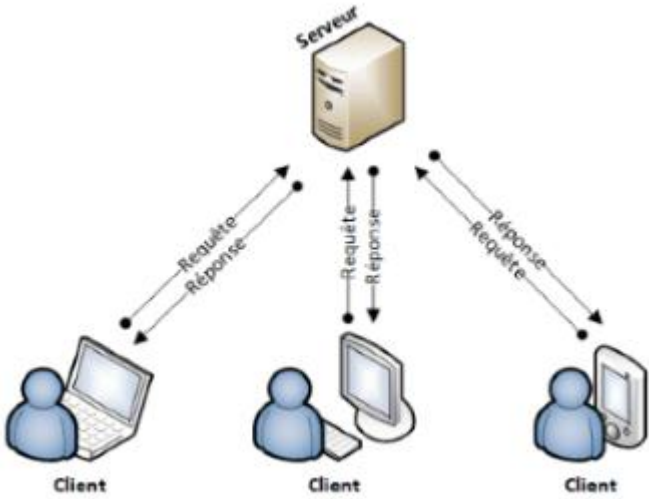
L'architecture client-serveur décrit le fonctionnement du **processus** permettant d'afficher une page web sur un navigateur.

Elle met en œuvre deux acteurs :

1. Le **client**, c'est le navigateur. Il va recevoir du **serveur** le contenu HTML à interpréter, il va le traiter, et afficher à l'écran le résultat de ce traitement : le site internet. Le **client** peut également effectuer des demandes au **serveur**, on parle de **requêtes** HTTP.
2. Le **serveur**, lui, remplit deux missions : le **stockage** (fichiers PHP / HTML, fichiers images et divers, base de données…) et l'interprétation requêtes HTTP.



Il est tout à fait possible d'avoir plusieurs clients pour un seul serveur :



Et même plusieurs serveurs si on découpe les rôles du serveur en plusieurs serveurs :

2. Un site, c'est quoi ?

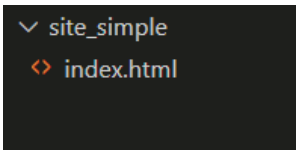
La définition la plus simple que l'on peut donner d'un site web serait la suivante :

C'est un ensemble de fichiers stockés sur un serveur dont l'interprétation permet de renvoyer un document HTML utilisable par le client (le navigateur).

Ces fichiers sur le serveur peuvent interagir entre eux, permettre d'effectuer des opérations complexes comme échanger avec la base de données, effectuer des appels à d'autre sites via API. Traiter ces opérations est le rôle du **serveur**.

Prenons un exemple simple : un site avec une seule page html. Mon site consistera donc en un seul fichier html, que je vais nommer index.html car le mot clé **index** est le nom de fichier par défaut que le serveur va nous renvoyer. **HTML** veut dire HyperText Markup Language.

Voici donc mon architecture :



Et le contenu de mon fichier :

site_simple > index.html > link

```html
<html>
    <head>
        <meta charset="utf-8">
    </head>
    <body>
        <header>
            <div class="monContenu">
                <h1>Voici un titre dans une balise H1</h1>
                <p>Et un bout de texte dans une balise p</p>
            </div>
        </header>
        <p>Ici on peut bien écrire ce qu'on veut</p>
    </body>
</html>
```

Le résultat affiché est le suivant :

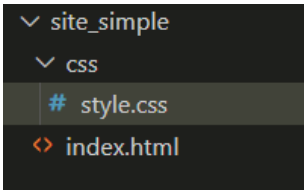
Voici un titre dans une balise H1

Et un bout de texte dans une balise p

Ici on peut bien écrire ce qu'on veut

On voit ici que le fichier a été interprété par le navigateur pour afficher mon texte dans des balises respectives, en leur appliquant une mise en forme par défaut. Parlons de mise en forme justement : On va créer un autre fichier, mais cette fois un fichier CSS, pour styliser un peu notre page. CSS veut dire **Cascading StyleSheet**.

Par soucis de clarté, on va ranger ce fichier CSS dans un **sous-dossier** « css ». En effet, il ne sera peut-être pas le seul fichier CSS que nous allons créer, il vaut donc mieux les ranger correctement dès le début. Voici notre nouvelle arborescence :



Le contenu du fichier CSS en question :

Et le résultat :

Voici un titre dans une balise H1

Et un bout de texte dans une balise p

Ici on peut bien écrire ce qu'on veut

On a bien un style qui s'est appliqué sur notre page, qui est donc le résultat de l'interprétation de deux fichiers. Comment est-ce possible ? Grâce à cette instruction supplémentaire que l'on rajoute dans notre index.html, ligne 3 :

site_simple > index.html > html

```html
<html>
    <head>
        <link href="/css/style.css" rel="stylesheet"/>
        <meta charset="utf-8">
    </head>
    <body>
        <header>
            <div class="monContenu">
                <h1>Voici un titre dans une balise H1</h1>
                <p>Et un bout de texte dans une balise p</p>
            </div>
        </header>
        <p>Ici on peut bien écrire ce qu'on veut</p>
    </body>
</html>
```

Cette instruction est interprétée par le navigateur qui va savoir lire le chemin du fichier CSS et aller le chercher pour l'intégrer à notre page.

Dernier exemple d'une interprétation, mais cette fois côté serveur : nous allons modifier notre fichier index.html et le renommer **index.php**. Et dans ce fichier, nous allons écrire ce code en PHP dans lequel on déclare une variable avec **$** et où l'on affiche un texte avec **echo** :

site_simple > index.php

```php
<html>
    <head>
        <link href="/css/style.css" rel="stylesheet"/>
        <meta charset="utf-8">
    </head>
    <body>
        <header>
            <div class="monContenu">
                <h1>Voici un titre dans une balise H1</h1>
                <p>Et un bout de texte dans une balise p</p>
            </div>
        </header>
        <p>Ici on peut bien écrire ce qu'on veut</p>
        <p>Et là on va afficher bonjour autant de fois que ma variable $nbBonjour le dit.</p>
        <?php
        $nbBonjour = 10;
        for($i=1; $i<=$nbBonjour; $i++) {
            echo "Bonjour N°" . $i . "<br>";
        }
        ?>
    </body>
</html>
```

Dans notre condition, on a trois paramètres

- **$i = 1** est notre point de départ : on donne à notre **index $i** une valeur de 1
- **$i <= $nbBonjour** est notre **condition**, la partie « **tant que** »
- **$i++** dit à notre boucle d'incrémenter notre inde x **$i** à chaque tout de boucle

Voici le résultat :

Voici un titre dans une balise H1

Et un bout de texte dans une balise p

Ici on peut bien écrire ce qu'on veut

Et là on va afficher bonjour autant de fois que ma variable $nbBonjour le dit.

Bonjour N°1
Bonjour N°2
Bonjour N°3
Bonjour N°4
Bonjour N°5
Bonjour N°6
Bonjour N°7
Bonjour N°8
Bonjour N°9
Bonjour N°10

3. Avant de se lancer

Avant de se lancer dans l'installation de **WAMP** et la configuration de son **environnement de développement**, quelques astuces, rappels et définitions :

1. La structure d'une page HTML obéit à plusieurs règles définies. La première d'entre elle est **l'encapsulation**. Quand on ouvre une balise, il faut la refermer au bon endroit, et ce même si l'on ouvre d'autres balises à l'intérieur. Par exemple, si j'ouvre une balise **<body>** et que DANS CETTE BALISE, j'ouvre une balise <p>, alors il faut que je ferme ma balise <p> AVANT de fermer ma balise <body>. On ferme une balise on la précédant d'un slash. Exemple :

```
<header>
    <div class="monContenu">
        <h1>Voici un titre dans une balise H1</h1>
        <p>Et un bout de texte dans une balise p</p>
    </div>
</header>
```

2. Le meilleur moyen de ne pas se perdre en appliquant ce principe d'**encapsulation** est de veiller à l'**indentation** de notre code. L'indentation consiste à décaler vers la droite le contenu d'une balise afin de faciliter la **lisibilité** du code. On peut voir dans l'exemple du dessus qu'il est facile de tracer un trait vertical entre le début et la fin de la balise <header> : et c'est bien le but !

3. Enfin, par facilité de langage, on appelle le document HTML affiché par le navigateur le DOM (Document Objet Model)

4. En PHP, on n'est pas obligés de typer les variables, sauf si l'on veut ajouter une sécurité supplémentaire. Pour l'instant, contentons-nous de les déclarer avec **$**

5. En PHP, on peut afficher n'importe quelle chaîne de caractère en utilisant **echo.**

6. On peut également utiliser la **concaténation** de chaînes de caractères en fermant les guillemets et en utilisant le **point .**

4. WAMP, c’est quoi ?

WAMP est un acronyme qui décrit les 4 technologies mise à l’œuvre dans l’environnement **local** de développement : Windows, Apache, MySQL et PHP.

Windows c’est simplement pour décrire l’environnement dans lequel WAMP est fait pour fonctionner.

Apache décrit le type de serveur Web sur le modèle duquel on va se baser. Apache est un des plus courants, avec Nginx.

MySQL est notre moteur de base de données

Et enfin PHP est notre langage de programmation principal.

WAMP va donc nous fournir un environnement de développement complet : Un serveur Apache avec PHP et MySQL d’installés sous Windows. Pas besoin d’entrer plus dans les détails pour l’instant, pratiquons !

5. Installation de WAMP Server

Télécharger le fichier sur : https://sourceforge.net/projects/wampserver/

Installer les packages Visual Studio C++ : https://github.com/abbodi1406/vcredist/releases

Rédémarrer le PC

Lancer Wamp, puis clic GAUCHE sur le menu contextuel > Apache > httpd.conf, modifier le DocumentRoot et le Directory path pour pointer vers votre répertoire de DEV. Je vous invite à faire un dossier bien accessible, épinglez-le à votre accès rapide, et rangez-le bien car nous y ferons certainement plusieurs projets.

Faire pareil dans httpd-vhosts.conf

On peut tester d'écrire en créant un fichier index.html dans le répertoire voulu, il devrait s'ouvrir sur l'adresse localhost ou 127.0.0.1

6. Pratique

1. Reproduire le fichier index.html
2. Reproduire le fichier style.css et l'inclure dans le fichier html
3. Changer l'extension de index.html en index.php et lui faire répéter bonjour.
4. Corriger le fichier Exemple 4
5. Corriger le fichier Exemple 5
6. Rajouter un titre <h2> dans le fichier, et lui définir un style
7. Isoler les styles appliqués aux deux titres dans un autre fichier titres.css, et l'inclure dans le fichier index.php
8. Créer une variable PHP $titre1, et une variable PHP $titre2. On déclarera ces variables comme contenant les chaînes de caractère des deux titres. Afficher les variables directement au lieu de la chaîne de caractères.

site_simple > index.php

```php
<?php
$titre1 = "Voici un titre dans une balise H1";
$titre2 = "Mon titre 2";
?>
<html>
```

```
<h1><?= $titre1 ?></h1>
<h2><?= $titre2 ?></h2>
```

9. Créer une variable **$nombre** et lui attribuer une valeur. Afficher dans la page le nombre suivi de « est pair » ou « est impair », en utilisant une structure **if / else**. Le modulo se note %, et si on est bien en modulo 2 alors on obtient un reste égal à 0. Alors on note $nombre % 2 == 0

```php
$nombre = 3;
if($nombre % 2 == 0) {
    echo $nombre . " est pair";
} else {
    echo $nombre . " est impair";
}
```

10. Utiliser une condition ternaire pour afficher ou non la classe « nombrePair » sur une div, en calculant le modulo 2 de $nombre.

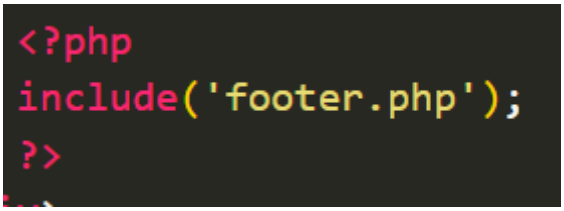
```php
echo ($nombre % 2 == 0) ? $nombre . " est pair" : $nombre . " est impair";
```

11. Prévoir un style CSS pour la classe nombrePair : texte en rouge et taille de police à 20px. Le nombre affiché ne doit prendre ce style que s'il est pair

```php
for($i = 1; $i <= 10; $i++) {
    $className = "bloc_couleur";
    if($i % 2 != 0) {
        // $className = $className . " rouge";
        $className .= " rouge";
    }

    ?>
    <div class="<?= $className ?>">
        Bloc N°<?= $i ?>
    </div>
    <?php
}
?>
```

12. Utiliser la syntaxe **include** de PHP pour mettre le footer dans un autre fichier HTML et l'inclure dans index.php



7. Glossaire

- **Client**
  - Le client est le navigateur, qui va permettre à l'utilisateur d'envoyer une requête HTTP au serveur, mais également d'interpréter la réponse du serveur
- **Serveur**
  - Le serveur stocke les fichiers du site ainsi que les données. Il est interrogé par le client sous forme de requête HTTP, et renvoie le HTML produit au navigateur
- **Navigateur**
  - Le navigateur est l'interprète côté client. Il reçoit le DOM (HTML) ainsi que le CSS et le Javascript, et interprète le tout pour afficher la page telle que l'utilisateur la visualise.
- **Requête http**
  - Une requête HTTP est une instruction normée envoyée au serveur, par exemple « Obtenir page Catalogue de produits ». Elles sont de 4 types : **POST** ou **PUT** pour l'envoi de données, **GET** pour la récupération de données, et **DELETE**. Dans le web, les plus courantes sont POST et GET
- **Arborescence**
  - On parle d'arborescence pour décrire l'agencement des dossiers, sous-dossiers et fichier dans un site web. Il faut visualiser ça comme les branche d'un arbre découlant du tronc commun : chaque branche doit représenter une fonction précise. Certains frameworks utilisent une arborescence **normée**, c'est-à-dire que vous n'aurez pas le choix d'arranger les fichiers comme il le requiert, mais dans un premier temps, tâchons simplement d'être organisés.
- **Itération**
  - Une itération est simplement en tour de boucle. Si on demande à une boucle d'aller de 1 à 10, alors il y aura une **itération** de 1, puis une **itération** de deux etc… Jusqu'à 10. Donc dix itérations au total. Pour décrire un algorithme, on parle parfois de « **L'itération en cours** », c'est-à-dire là où on en est dans la boucle. Par exemple si je veux aller de 1 à 10 mais afficher « À demain » à la troisième itération, je pourrais tester **l'itération en cours** pour savoir si c'est la troisième ou pas.
- **Encapsulation**
  - L'encapsulation du code décrit l'idée que chaque balise doit être fermée uniquement lorsque tout son contenu a également été fermé. L'encapsulation définit l'ordre et la clarté du code.
- **Indentation**
  - L'indentation est l'action de décaler vers la droite les divers éléments du code pour en faciliter l'organisation et la lisibilité. En PHP **l'indentation n'est pas interprétée**. Cela n'a donc aucune utilité du point de vue du serveur ou du navigateur, en revanche pour le développeur, un code non indenté est indéchiffrable. Note : il existe certains langages qui prennent en compte l'indentation, comme les fichiers YAML.
- **Concaténation**
  - La concaténation est l'action de prendre deux chaînes de caractères différentes et de les assembler pour en faire une chaîne de caractère plus grande. Par exemple la chaîne « Bonjour » et la chaîne « tout le monde » peuvent être concaténée pour

obtenir « Bonjour tout le monde ». En PHP, on utilise le caractère **point** pour concaténer, mais en Javascript ça sera le caractère **+**.

- **Condition (structure IF/ELSE)**
  - Une condition aussi basique qu'essentielle. On la note

```
if(ma condition) {
    Faire mon action
} else {
    Faire une autre action
}
```

- **Boucle (Structure FOR)**
  - La boucle **for** permet d'**itérer** un nombre défini de fois : On lui donne trois paramètres séparés par des points virgules.
    - Le premier est notre **index**, on lui donne une valeur de départ, le plus souvent 0 ou 1
    - Le deuxième est une limite, on dit à la boucle de continuer à itérer tant que la condition est **vraie**
    - Le troisième est la description de notre incrémentation : En général on va de 1 en 1 mais des exceptions peuvent se produire.
      Exemple :

```
$nbBonjour = 10;
for($i=1; $i<=$nbBonjour; $i++) {
    echo "Bonjour N°" . $i . "<br>";
}
```