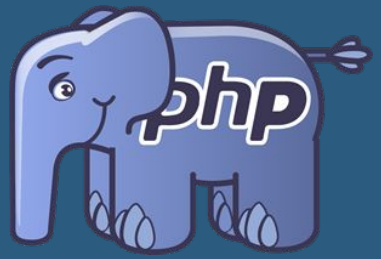
IUT de Montpellier - 2022 / 2023



Développement Web

Introduction

Enseignant :
Malo Gasquet

Déroulement du module

- Lundi 5 Décembre 2022 – TD1 – **Introduction à PHP & Persistance des données**
- Mardi 7 Décembre 2022 – TD2 – **Création d'un framework 1/2**
- Vendredi 9 Décembre 2022 – TD3 – **Création d'un framework 2/2**
- Lundi 12 Décembre 2022 – TD4 – **Evolution de l'application**
- ??? : Mise en place d'une plateforme de paiement?
- Vendredi 16 Décembre 2022 – Début du **projet**

Thème du site : Mini réseau social

Evaluation

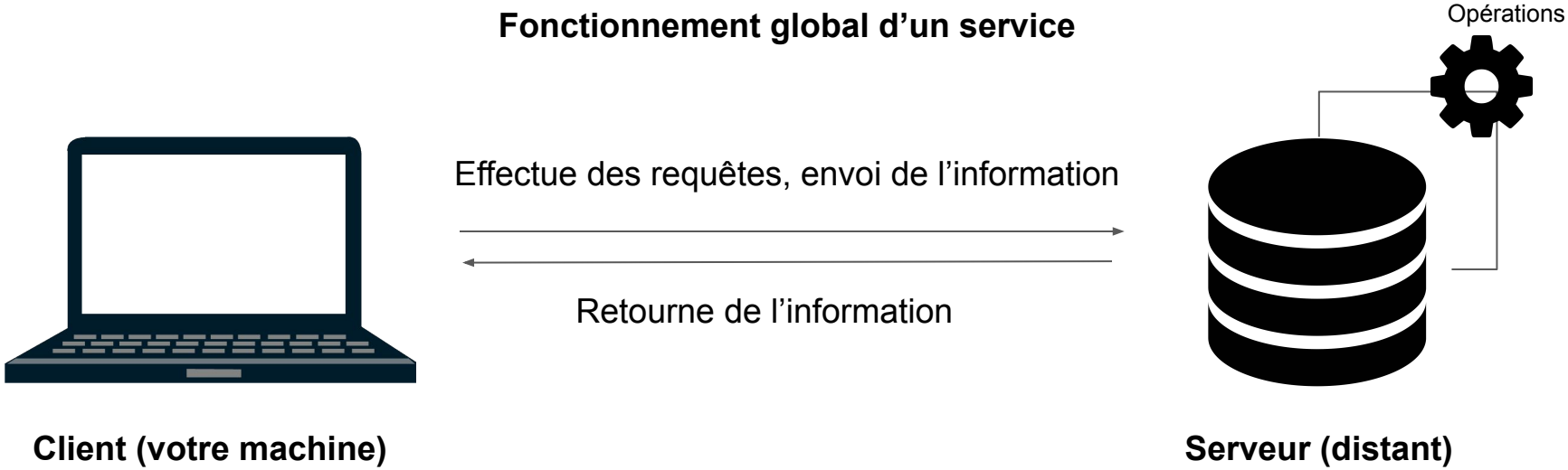
- **Note de participation** : remise de l'état d'avancement du TP en cours à la fin de chaque séance.
- **Note de projet** : projet en **binôme**, 4h de cours prévues pour le lancement.
- **Note d'examen** : prévu courant janvier, à priori.

Objectifs du module

- Découvrir (ou redécouvrir!) les bases du **langage PHP**
- Comprendre le fonctionnement d'un **framework** en détail avant d'en utiliser un (dans le cours de **Symfony**…)
- Adopter des **bonnes pratiques** de programmation (**architecture**, **design patterns**…)
- Utiliser et découvrir des **bibliothèques logicielles** externes
- Construire une **application web**
- **Dynamiser** un site web, début de mise en place d'une **API**
- Faire le lien avec le cours de **Symfony** et d'**Android**

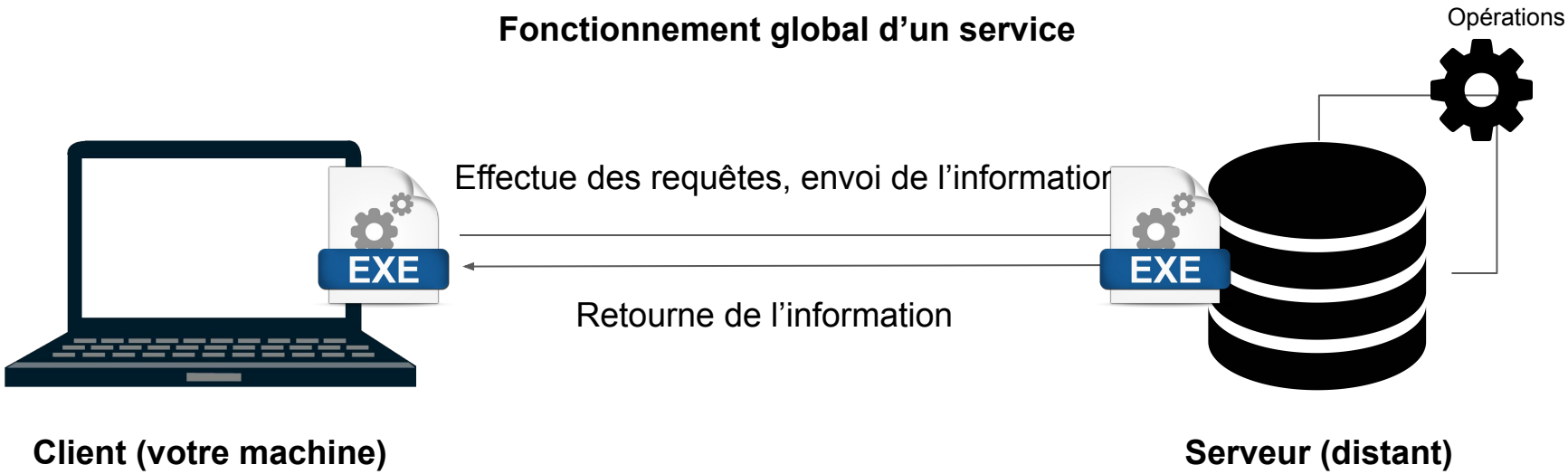
Internet et ses services - Fonctionnement global

Fonctionnement global d'un service



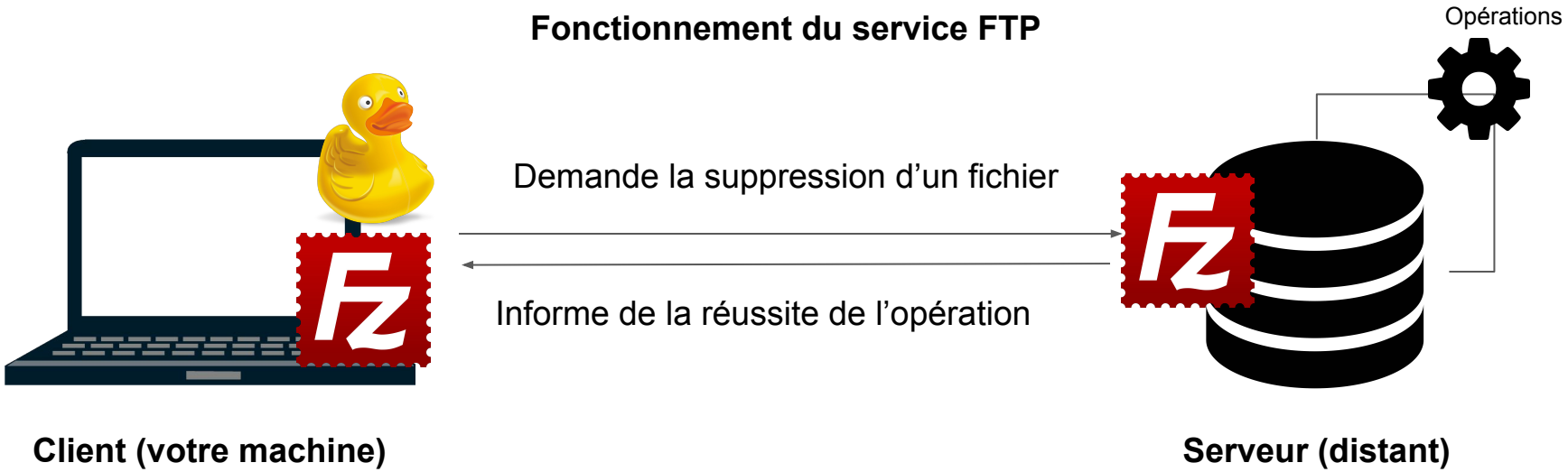
Les services sont attribués à un **port** (généralement 80 pour HTTP, 443 pour HTTPS, 20 et 21 pour FTP)
Le client et le serveur communiquent selon un **protocole** définit par le service

Internet et ses services - Fonctionnement global



Plus précisément, les services s'utilisent au travers de **programmes**.
On nomme programme "**client**" le programme s'exécutant sur l'ordinateur de l'utilisateur.
On nomme programme "**serveur**" le programme s'exécutant sur la machine jouant le rôle du serveur.
Le **port** sert à identifier vers quel **programme** rediriger l'information

Internet et ses services - Exemple - FTP



Le type de programme client **ne dépend pas** du type de programme serveur (et vice-versa).

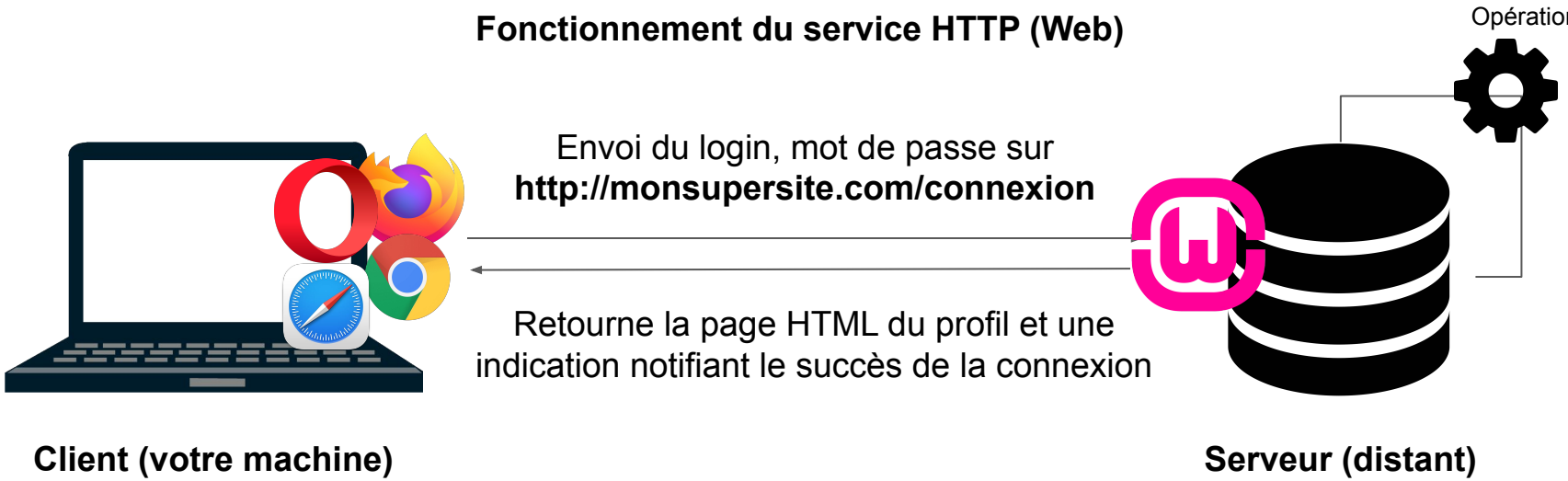
Un serveur FTP mis en place avec le programme "FileZila serveur" peut tout à fait communiquer avec un client Cyberduck, ou même en ligne de commande avec un terminal.

Internet et ses services - HTTP - Le service web!

Fonctionnement du service HTTP (Web)

Internet et ses services - HTTP - Le service web!

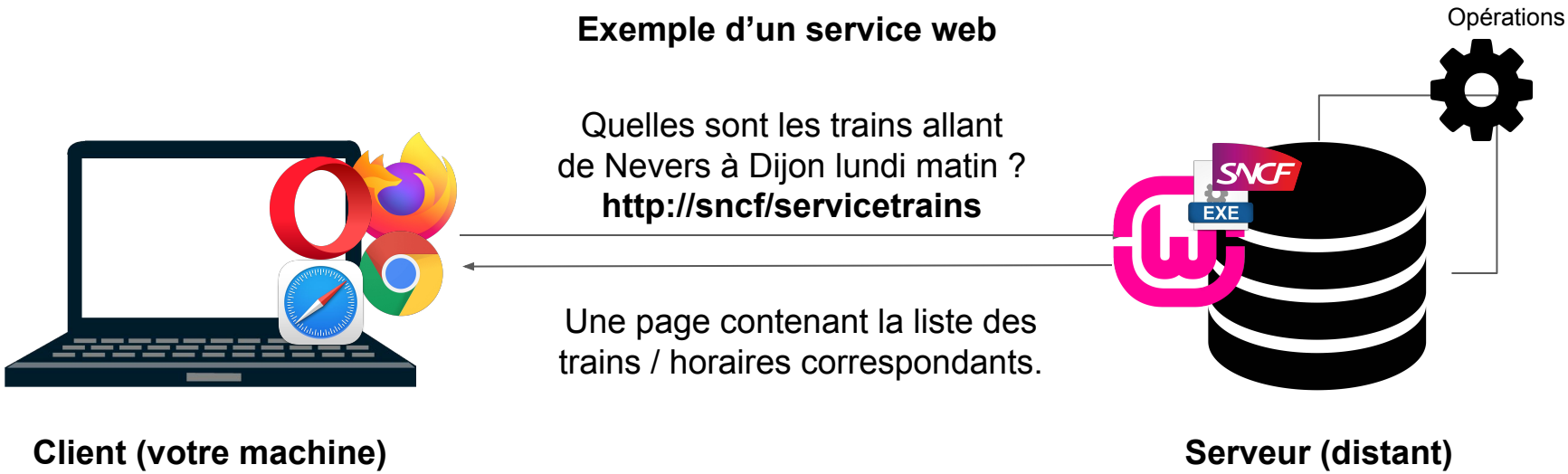
Fonctionnement du service HTTP (Web)



N'importe quel navigateur est un programme "client" qui peut communiquer avec un serveur web distant, au travers du protocole HTTP (et / ou HTTPS, si le serveur est configuré pour). Comme pour les navigateurs, il existe différents programmes de serveur web.

Le web - Les services web

Exemple d'un service web



On nomme communément un "service web" une application ayant pour but de proposer un service à l'utilisateur. Par exemple, réserver un train, une place de cinéma, utiliser un réseau social...

Le web - Un domaine en constante évolution

Au tout début le web ne servait qu'à échanger des documents "statiques"

Un site web "statique" est défini par le fait que les pages de ce site sont considérées comme des documents que l'on consulte, reliés entre eux, sans proposer de service.
Bien sûr, certains sites web possèdent des pages dynamiques et statiques (les CGU par exemples)

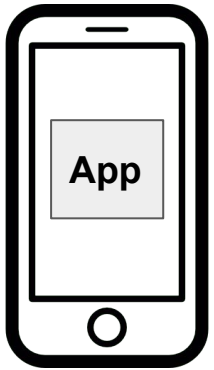
Le web - Un domaine en constante évolution

Dans les années 2000

L'émergence des technologies de programmation web (notamment **PHP**) côté serveur ont permis de passer d'un web statique à un web dynamique, et ainsi, proposer divers services web.

Le web - Un domaine en constante évolution

Client (votre machine)

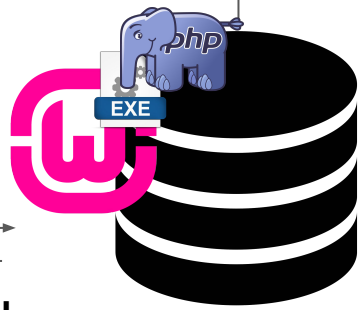


Je veux me connecter à mon espace
http://reseausocial/connexion.php

Données JSON de l'espace personnel

API

Opérations



Serveur (distant)

L'émergence de nouvelles technologies (comme les smartphones, tablettes, etc…) a amené à repenser le modèle de fonctionnement des services web.

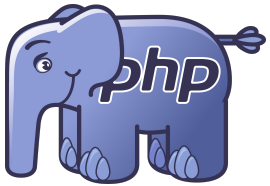
Développement web

Principalement en 1ère année d'études

Tout ce qui va être affiché, exécuté et lu par le navigateur

Front-end

Principalement en 2ème et 3ème année d'études



Tout ce qui se passe côté serveur

Back-end

Full Stack

PHP - Un peu d’histoire

- Créé par **Rasmus Lerdorf** en **1994**
- Utilisé pour afficher le nombres de personnes ayant consulté son CV sur son site web
- A la base : une librairie du **langage C** qui a peu à peu évolué
- Le but d’origine est simple : un **programme serveur** générant une **page HTML** et la renvoyant au **client** (navigateur).
- Maintenant, il peut être utilisé pour récupérer d’autres types de données, comme du **JSON** dans le cadre d’une **API**, par exemple

PHP - Le langage

- Un fichier php doit débuter par : **<?php**
- Les variables sont précédées d'un **$**
- Le langage n'est **pas typé par défaut**, mais on peut forcer le type de certaines variables dans les attributs, paramètres, retour de fonctions…
- On peut créer des **classes**, manipuler des **objets…**
- Assez similaire aux **langages orientés objets** courants (pour les mots-clés)
- Le langage peut être **permissif** sur certains aspects (ce qui peut être une bonne ou une mauvaise chose…)

PHP - Le langage - Variables

```php
<?php

$test = "hello";
$test = 5;

$date = new Date();

$nom = "Smith";
$prenom = "John";
$message = "Je m'appelle $prenom $nom";
//Concaténation avec .
$message = "Je m'appelle ". $prenom. " ". $nom;
```

PHP - Le langage - Conditions

```php
if($age > 18 && $age <21) {
    ...
}
```

PHP - Le langage - Les tableaux

```php
//Tableau vide
$tab = [];

//Tableau indexé par 0,1,2,...
$tab = [1,2,5,7];
$val = $tab[2]; //$val == 5

//Ajout à la fin du tableau
$tab[] = 8;

//Tableau multi contenu
$tab = [5, "hello", false];

//Tableau "associatif" => Les index ne sont pas forcément des nombres
(équivalent aux Map ou dictionnaires)
$tab = ["login" => "admin", "password" => "123456"];
$tab["role"] = "administrateur";
$pass = $tab["password"]; //$pass == "123456"
```

PHP - Le langage - Boucles - While

```php
$tab = [0,1,5,4,8];
$i = 0;
while($i < count($tab)) {
    $val = $tab[$i];
    $i++;
}
```

PHP - Le langage - Boucles - For

```php
$tab = [0,1,5,4,8];
for($i = 0; $i < count($tab), $i++) {
    $val = $tab[$i];
}
```

PHP - Le langage - Boucles - Foreach

```php
$tab = [0,1,5,4,8];
foreach($tab as $val) {
    ...
}
```

- **1ère itération : val == 0**
- **2ème itération : val == 1**
- **3ème itération : val == 5…**

PHP - Le langage - Boucles - Foreach

```php
$joueurs = ["bob" => ["niveau" => 5, "argent" => 1056, "alive" => true],
            "kris" => ["niveau" => 47, "argent" => 54848, "alive" => false],
            "banana" => ["niveau" => 14, "argent" => 458, "alive" => true]
           ];

foreach ($joueurs as $pseudo => $joueur) {
    $alive = $joueur["alive"];
    if($alive) {
        $message = "Joueur $pseudo : \n";
        $niveau = $joueur["niveau"];
        $argent = $joueur["argent"];
        $message .= "Niveau : $niveau\n";
        $message .= "Argent : $argent\n";
    }
}
```

PHP - Le langage - Fonctions

```php
function puissance($x, $n) {
    $result = 1;
    for($i=1;$i<=$n;$i++) {
        $result *= $x;
    }
    return $result;
}

$puiss = puissance(2, 4);
```

PHP - Le langage - Fonctions

On peut aussi forcer le type des paramètres / du retour!

```php
function puissance(int $x, int $n) : int {
    $result = 1;
    for($i=1;$i<=$n;$i++) {
        $result *= $x;
    }
    return $result;
}

$puiss = puissance(2, 4);
```

PHP - Le langage - Classes

- On peut créer des **classes**, des **classes abstraites**, des **interfaces…**
- **Visibilité** sur les attributs/méthodes similaire à Java (**public**, **private**, **protected**…) mais pas de visibilité sur les classes.
- On peut **forcer le type** d'un attribut, d'un paramètre, d'un retour de méthode…
- Certaines méthodes peuvent êtres **abstraites** (ce qui implique que la classe l'est)
- **On ne peut pas surcharger les méthodes**! (donc, un seul constructeur possible…)
- Le constructeur se définit via la méthode **__construct**

PHP - Le langage - Classes

- On peut créer des méthodes et des attributs **static** accessibles par : **NomClasse::attribut**, **NomClasse::methode()**, etc…
- On peut hériter d'une classe (**extends**), implémenter une interface (**implements**). Pas de multi-héritage (multi extends) comme en Java
- On peut utiliser le mot clé **final** sur un attribut (non réaffectable), une classe (non extensible), une méthode (non redéfinissable)
- On **instancie** un objet avec le mot clé **new**
- On peut créer des **constantes** avec le mot clé **const**
- On accède aux **méthodes** d'une instance avec la **flèche** (et pas le point)

PHP - Le langage - Classes - Exemple

```php
class Personne {
    private $nom;
    private $prenom;

    public function __construct($nom, $prenom) {
        $this->nom = $nom;
        $this->prenom = $prenom;
    }

    public function getNom() {
        return $this->nom;
    }

    public function setNom($nom) {
        $this->nom = $nom;
    }
}
```

PHP - Le langage - Classes - Exemple

```php
class Personne {
    private string $nom;
    private string $prenom;

    public function __construct(string $nom, string $prenom) {
        $this->nom = $nom;
        $this->prenom = $prenom;
    }

    public function getNom() : string {
        return $this->nom;
    }

    public function setNom(string $nom) : void {
        $this->nom = $nom;
    }
}
```

PHP - Le langage - Classes - Exemple

```php
$p = new Personne("Smith", "John");
$nom = $p->getNom();
$p->setNom("Hello there!");

//TRES PERMISSIF!
$nomClasse = "Personne";
$p2 = new $nomClasse("Smith", "John");
```

PHP - Le langage - Classes - Exemple

```php
abstract class A {
    protected $a;

    public function __construct($a) {
        $this->a = $a;
    }

    public function action() {}

    public abstract function action2();

    protected function action3() {}
}
```

```php
class B extends A {
    private $b;

    public function __construct($a, $b){
        parent::__construct($a);
        $this->b = $b;
    }

    public function action2(){}

    protected function action3() {
        parent::action3();
    }
}

$test = new B(5,7);
```

PHP - Le langage - Classes - Exemple

```php
class C {

    public static $var = "Hello";

    const TEST = 10;

}

$v = C::$var;
C::$var = 10;
$test = C::TEST;
```

PHP - Le langage - Classes - Exemple

```php
interface TestInterface {

    public function action();

}

class D implements TestInterface {

    public function action()
    {

    }
}
```

PHP - Exemple de génération d'une page HTML

```php
<?php
$nom = "Smith";
$prenom = "John";
?>

<!DOCTYPE html>
<html lang="fr">
<head>
   <meta charset="UTF-8">
   <title>Demo</title>
</head>
<body>
   <p>Je m'appelle <?php echo "$prenom $nom";?></p>
</body>
</html>
```

PHP - Exemple de génération d'une page HTML

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Demo</title>
</head>
<body>
<p>Je m'appelle John Smith</p>
</body>
</html>
```

PHP - Exemple de génération d’une page HTML

Je m'appelle John Smith

PHP - Exemple de génération d'une page HTML

```php
<?php
$john = new Personne("Smith", "John");
$kris = new Personne("GetTheBanana", "Kris");
$guy = new Personne("Tarembois", "Guy");
$tabPersonnes = [$john, $kris, $guy];
?>

<!DOCTYPE html>
<html lang="fr">
   <head>
       <meta charset="UTF-8">
       <title>Demo</title>
   </head>
   <body>
       <ul>
           <?php foreach ($tabPersonnes as $personne) { ?>
           <li><?php echo "{$personne->getPrenom()} {$personne->getNom()}";?></li>
           <?php } ?>
       </ul>
   </body>
</html>
```

PHP - Exemple de génération d'une page HTML

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Demo</title>
</head>
<body>
<ul>
    <li>John Smith</li>
    <li>Kris GetTheBanana</li>
    <li>Guy Tarembois</li>
</ul>
</body>
</html>
```

PHP - Exemple de génération d’une page HTML

- John Smith
- Kris GetTheBanana
- Guy Tarembois

PHP - Variables globales spéciales

- A chaque requête, **PHP** met à jour le contenu de certaines **variables globales** accessibles par le développeur dans n'importe script **PHP,** par exemple :
  - **$_GET** : Tableau associatif contenant les données envoyées par l'utilisateur via le query string (directement dans l'URL ou bien via un formulaire en **GET**)
  - **$_POST** : Tableau associatif contenant les données envoyées par l'utilisateur via la méthode POST (par exemple, avec un formulaire utilisant la méthode **POST**)

PHP - La méthode GET

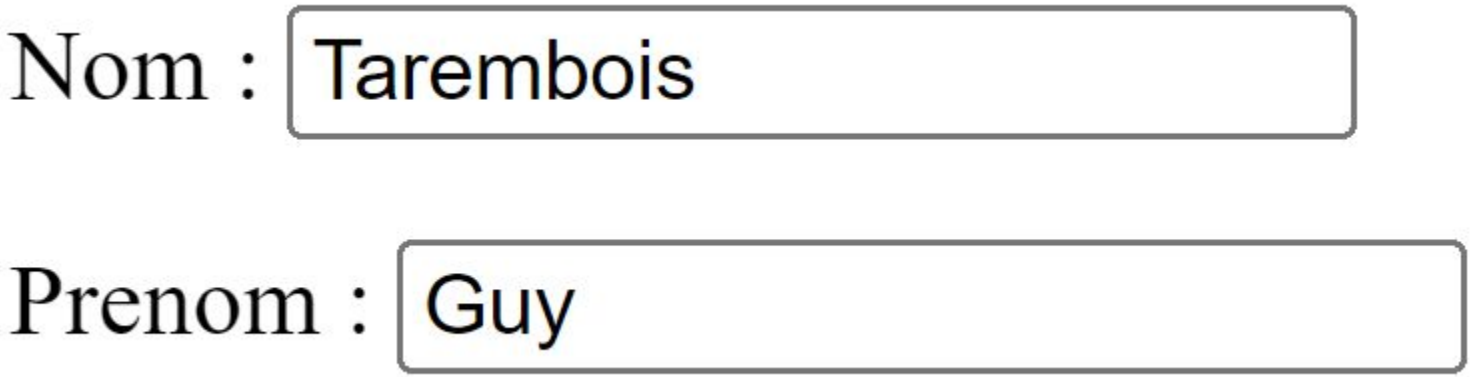
- Cette méthode d'envoi de données encode les données dans l'URL sous la forme : **cle=valeur**
- Les données sont séparées par des **&**
- La zone contenant ces données débute par un **?** et est appelée **query string**
- Le serveur récupère cette information dans la variable **$_GET**
- On ne doit pas transmettre de **données sensibles** via cet URL! (car visibles dans l'URL…)
- On peut utiliser un **formulaire HTML** paramétré avec la méthode **GET**

http://monsite.com**?nom=Smith&prenom=John**

PHP - La méthode GET

```
<html lang="fr">
<head>
   <meta charset="UTF-8">
   <title>Demo</title>
</head>
<body>
   <form method="get" action="demo.php">
       <p>
           <label for="nom">Nom : </label>
           <input id="nom" type="text" name="nom"/>
       </p>
       <p>
           <label for="prenom">Prenom : </label>
           <input id="prenom" type="text" name="prenom"/>
       </p>
       <p>
           <input type="submit" value="Envoyer">
       </p>
   </form>
</body>
</html>
```

PHP - La méthode GET



http://monsite.com/demo.php?nom=Tarembois&prenom=Guy

PHP - La méthode GET

```php
<?php
$nom = $ GET["nom"];
$prenom = $_GET["prenom"];
?>

<!DOCTYPE html>
<html lang="fr">
<head>
   <meta charset="UTF-8">
   <title>Demo</title>
</head>
<body>
   <p>Je m'appelle <?php echo "$prenom $nom";?></p>
</body>
</html>
```

PHP - La méthode GET

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Demo</title>
</head>
<body>
<p>Je m'appelle Guy Tarembois</p>
</body>
</html>
```

Je m'appelle Guy Tarembois

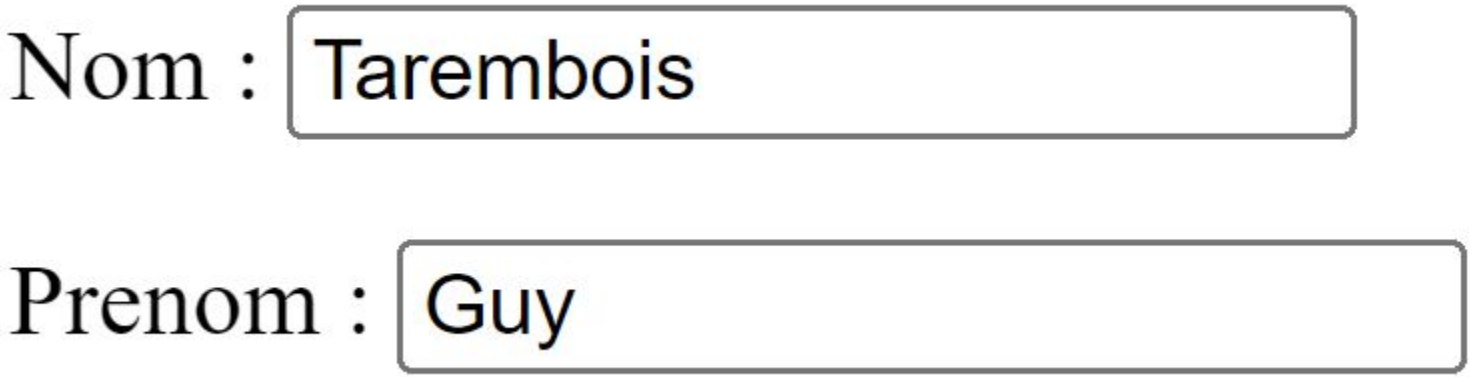
PHP - La méthode POST

- Cette méthode d'envoi de données encode les données dans le corps de la requête
- Le serveur récupère cette information dans la variable **$_POST**
- On peut transmettre des données **sensibles** via cet URL (non visibles dans l'URL…)
- On peut utiliser un **formulaire HTML** paramétré avec la méthode **POST**
- L'URL ne change pas

PHP - La méthode POST

```
<html lang="fr">
<head>
   <meta charset="UTF-8">
   <title>Demo</title>
</head>
<body>
   <form method="post" action="demo.php">
       <p>
           <label for="nom">Nom : </label>
           <input id="nom" type="text" name="nom"/>
       </p>
       <p>
           <label for="prenom">Prenom : </label>
           <input id="prenom" type="text" name="prenom"/>
       </p>
       <p>
           <input type="submit" value="Envoyer">
       </p>
   </form>
</body>
</html>
```

PHP - La méthode POST



http://monsite.com/demo.php

PHP - La méthode POST

```php
<?php
$nom = $ POST["nom"];
$prenom = $_POST["prenom"];
?>

<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Demo</title>
</head>
<body>
    <p>Je m'appelle <?php echo "$prenom $nom";?></p>
</body>
</html>
```

PHP - La méthode POST

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Demo</title>
</head>
<body>
<p>Je m'appelle Guy Tarembois</p>
</body>
</html>
```

Je m'appelle Guy Tarembois

Cookies et Session

- On peut également définir des **cookies** : données stockées du côté client (dans le navigateur) et transmises au serveur à **chaque requête**. On peut notamment y stocker des données non sensibles, comme les préférences de l'utilisateur. Ces cookies peuvent être partagés à d'autres sites (si l'utilisateur l'autorise…). Le serveur peut placer des données dans les cookies et les lire. Un cookies peut avoir une durée de vie définie.

- La variable **$_SESSION** est une autre **variable globale** contenant un **tableau associatif** stockant des données sur la session de l'utilisateur. Les données sont **persistantes** d'une requête à l'autre (on peut écrire dans ce tableau). La session a une durée et expire au bout d'un certain temps (définie par le développeur, ou bien, par exemple, quand le site est quitté / navigateur fermé…). On peut y stocker des **données sensibles**, car ces données ne sont pas visibles côté client.

Architecture d’une application

- Une **application web** et plus globalement un l**ogiciel** est organisé selon une architecture précise qui sépare de manière optimisée les classes et programmes selon leur **rôle**. Différentes architectures sont possibles, mais globalement, on retrouve toujours les mêmes types de rôle.

- Peu importe l'architecture mise en place, un logiciel est globalement constitué de **5 couches principales**.

- La couche **ihm** qui permet de gérer les différentes parties graphiques et surtout l'interaction avec l'utilisateur. Pour une application web cela va correspondre à la partie contenant les **vues**, c'est-à-dire les fichiers responsables de générer le code HTML (et également les ressources javascript, css, etc...)

- La couche **métier** qui contient le cœur de l'application, à savoir les différentes **entités** manipulées ainsi que des classes de **services** qui permettent de manipuler ces entités et d'implémenter la **partie logique** de votre application.

Architecture d’une application

- La couche **application** qui permet de faire le lien entre la couche **ihm** et la couche **métier**. Elle contient différents **controllers** dont le rôle est de gérer les **évènements** qui surviennent sur l'interface et d'envoyer des **requêtes** auprès de la couche **métier** et de transmettre les résultats obtenus à **l'ihm**. Dans une application web, les événements sont les requêtes reçues par l'application web (et ses paramètres, via l'URL). Une requête est décomposée puis la bonne méthode du controller est exécutée avec les paramètres correspondant.

- La couche **stockage** qui permet de gérer la **persistance des données** à travers une forme de stockage configurée (base de données, fichier...). Son rôle va donc être de sauvegarder et charger les données des différentes entités de la couche **métier**. Cette couche est généralement utilisée par les différents classes de services. Globalement, les interactions se déroulent dans ce sens : IHM <-> Application <-> Services <-> Stockage.

- Éventuellement, la couche **réseau** dans le cadre d'une application **client/serveur**. Cette couche va gérer la transmission des données entre deux programmes (avec des sockets, etc...). Dans une application web, il n'y a pas besoin de gérer explicitement cette couche qui est prise en charge par le protocole **HTTP** ou **HTTPS**.

Les composants essentiels d’une application web

Au delà de l’architecture, une **application web** a aussi besoin :

- D'un **point d'entrée** qui est le premier fichier exécuté lors de la réception d'une requête sur votre application. Son rôle est de récupérer les informations utiles de la requête et de la transmettre à votre application.

- D'un **routeur**, c'est-à-dire une portion de code qui associe des chemins (des **routes**) à des fonctions sur des **controllers** bien précis et permet donc d'exécuter le bon code en se basant sur les données fournies par la requête.

- D'un **résolveur d'arguments** qui permet d'extraire des données fournies dans l'URL de la route (notamment dans le cadre d’une API). A noter que cela ne concerne pas les données envoyées par les méthodes GET, POST ou autre, qui sont accessibles dans le corps de la requête.

Les frameworks

- Un **framework** est une **infrastructure logicielle** fournissant divers **outils** et composants afin de mettre en place un projet informatique. Un **framework** définit un **cadre de travail** (et donc des règles) pour le développeur afin qu'il puisse développer son application efficacement sans avoir à se soucier des diverses problématiques, notamment liées à **l'architecture**, l'accès aux services, le **routage**, le stockage, etc...

- Un **framework** permet donc **d'optimiser** la mise en place d'un projet. Le code d'un framework est **générique** et peut être théoriquement appliqué à n'importe quel projet. Il est généralement constitué de plusieurs **composants** distincts et parfois même de diverses **bibliothèques externes**.

- La différence fondamentale avec une **librairie** (ou une **API**) c'est que ce n'est pas le développeur qui se sert du code de cette ressource pour son projet mais plutôt le framework qui intègre le code créé par le développeur dans son environnement. C'est un peu comme si le développeur développait un composant qui viendrait alors s'insérer dans le **framework**.

Les frameworks

- Pour citer deux célèbres **frameworks**, on a par exemple :
    - **Symfony** (que vous utiliserez dans le prochain module web) qui est un framework PHP (français!). .Ce Framework est très vite devenu populaire à travers le monde, et il est particulièrement utilisé dans les SSII du secteur de Montpellier!
    - **Spring**, qui est un framework Java.
- Vous l'aurez compris, utiliser un **framework** permet d'optimiser au mieux le développement d'un projet en **obligeant le développeur à respecter le cadre de travail défini**. Les frameworks utilisent divers **design patterns** pour gérer l'architecture des projets et l'accès aux composants. Généralement, un framework bien construit amène naturellement le développeur à utiliser ces patterns (parfois sans s'en rendre compte!).
- L'utilisation du framework se fait quasiment en "**boîte noir**", c'est-à-dire que le développeur n'a pas besoin de savoir comment il foncitonne en interne (c'est globalement un avantage qui rend donc ces frameworks très accessibles). Néanmoins, il semble intéressant, dans le cadre de ce cours et avant que vous utilisez ce genre d'outil, de **construire votre propre framework** afin de réellement comprendre comment ils fonctionnent.

Une API Web

- Une **API** est un **service web** permettant de réaliser des actions et qui renvoie des données brutes (généralement sous le format **JSON**, mais parfois aussi en **XML**…). Généralement, on communique avec ces APIs avec les différentes méthodes HTTP : **GET, POST, PUT, PATCH** et **DELETE** en utilisant un **payload** sous le format **JSON** également.

- L'objectif est de pouvoir utiliser l'**API** avec des clients de **différentes natures** : Navigateur, application mobile, une autre **API**…Le rôle du service web n'est **pas de gérer l'interface graphique**.

- Une architecture (et des règles) particulière d'API est appelée **REST**, nous aurons l'occasion d'en reparler.

Server rendering VS API

- On a ici deux modèles qui s'opposent :
    - Le **server rendering** où le serveur (l'application web) gère **tout** : réception de la requête, traitement, et **génération de la page de résultat** (donc, toutes les couches)**.** Tout est regroupé en **une seule application**, mais il n'est pas possible (ou compliqué) d'utiliser cette application avec d'autres technologies (comme une application mobile).
    - Le modèle où l'application web côté serveur se limite à une **API** et **ne gère pas la génération des pages de résultats** (couche IHM). Diverses technologies et clients peuvent utiliser la même API (Angular, application Android, Apple, React…). Le code "**métier**" est centralisé. Il faut donc au minimum deux applications différentes (une API côté serveur et un client) pour avoir un site web, mais celui-ci devient plus facilement **adaptable** sur différents supports.