

Initiation Back-end

Algorithmes & serveurs

C'EST PARTI ! →



Samih HABBANI



Initiation Back-end



Introduction au terminal de commande

Les lignes de commande via le **cmd** sur Windows, ou le **terminal** sur Mac vous permettront de communiquer avec votre ordinateur pour **exécuter des tâches** diverses à l'aide de commandes pré-définies que nous découvrirons dans les prochaines parties.



Différences entre Client/Serveur, Front/Back

Vous découvrirez dans les prochaines parties la différence entre les notions "**client**" et "**serveur**", souvent associées aux termes "**front-end**" et "**back-end**".



Initiation au PHP

C'est alors que vous concevrez vos **premiers algorithmes en PHP** en voyant les concepts clés de ce langage back-end.



Initiation Back-end



Introduction aux librairies PHP avec Composer

Pour accélérer les développements, vous découvrirez comment installer des librairies externes via l'outil **Composer** en ligne de commande.



Introduction aux API et à GuzzleHttp

Vous découvriez finalement que sont les API et notamment comment utiliser l'API **GuzzleHttp** qui est une bibliothèque PHP populaire utilisée pour effectuer des requêtes HTTP.



Exercices pratiques

Vous devrez finalement réaliser un projet avec les notions abordées durant la semaine.

INITIATION BACKEND

PARTIE 1

Initiation au terminal

- Qu'est-ce qu'un terminal de commande?
- Les commandes de base
- Exercice pratique

PARTIE 2

Client/server

- Client vs Serveur
- Front-end vs Back-end
- Illustrations
- Exercice pratique

PARTIE 3

Initiation PHP

- L'histoire de PHP
- Syntaxe de base
- Fonctions
- Exercice pratique

INITIATION BACKEND

PARTIE 4

Librairies PHP

- Qu'est-ce que Composer?
- Installation et utilisation
- Exercice pratique

PARTIE 5

Introduction aux API

- Qu'est-ce qu'une API?
- Fonctionnement
- Exemples d'API
- Les API privées

PARTIE 6

Guzzle HTTP

- Présentation de GuzzleHttp
- Installation et configuration
- Exemple pratique : API météo
- Lien entre PHP & JS
- Le format JSON
- Synchrone vs Asynchrone
- Exercice pratique
- Rendu final

INITIATION BACKEND

⚠️ Rappel sur l'utilisation de l'IA ⚠️

L'utilisation de l'IA pour générer du contenu (texte, code, image, etc) que vous présentez comme vôtre dans un livrable est à la fois un délit de « **Plagiat** » et un délit de « **Fraude à un examen** » (contrôle continu).

Interdit

Sauf autorisation exceptionnelle et écrite

- Générer du contenu par IA et l'intégrer dans votre rendu pour évaluation

Autoriser

- Utiliser l'IA pour apprendre ou pour vous accompagner dans vos réflexions

Attention

- Ne pas prendre les informations fournies par une IA comme forcément vrai, revérifiez !

PARTIE 1

INTRODUCTION AU TERMINAL DE COMMANDE

Le terminal de commande est **un outil essentiel pour interagir directement avec un système d'exploitation**. Il permet d'exécuter des commandes, de gérer des fichiers et d'automatiser des tâches.

Cette introduction présente ses principales fonctionnalités pour vous aider à l'utiliser efficacement.



QU'EST-CE QU'UN TERMINAL DE COMMANDE ?



COMMANDES DE BASE



EXERCICE PRATIQUE

INTRODUCTION AU TERMINAL DE COMMANDE

PARTIE 1

QU'EST-CE QU'UN TERMINAL DE COMMANDE ?

Définition et utilité

Un terminal de commande est un outil qui **permet d'interagir avec un système d'exploitation** en utilisant des lignes de commande textuelles.

C'est une **interface en mode texte** où l'utilisateur peut saisir des commandes pour **exécuter des actions spécifiques**, comme naviguer dans les fichiers, installer des logiciels ou gérer des processus.

Utilité

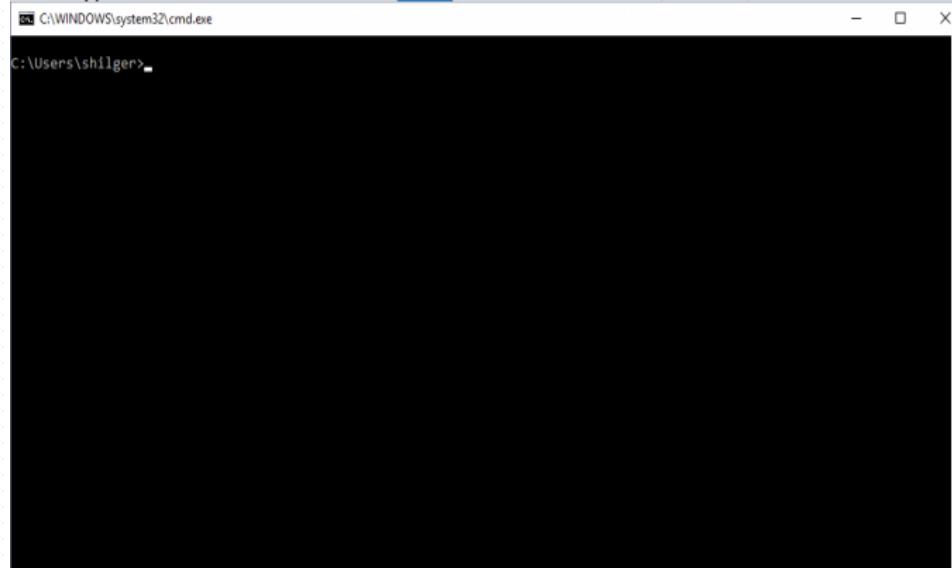
- **Contrôle total** : Accéder et gérer directement les ressources du système.
- **Automatisation** : Exécuter des scripts pour automatiser des tâches répétitives.
- **Efficacité** : Gagner du temps grâce à des commandes rapides et précises.
- **Développement** : Utilisé pour compiler des programmes, gérer des versions et configurer des environnements de développement.

QU'EST-CE QU'UN TERMINAL DE COMMANDE?

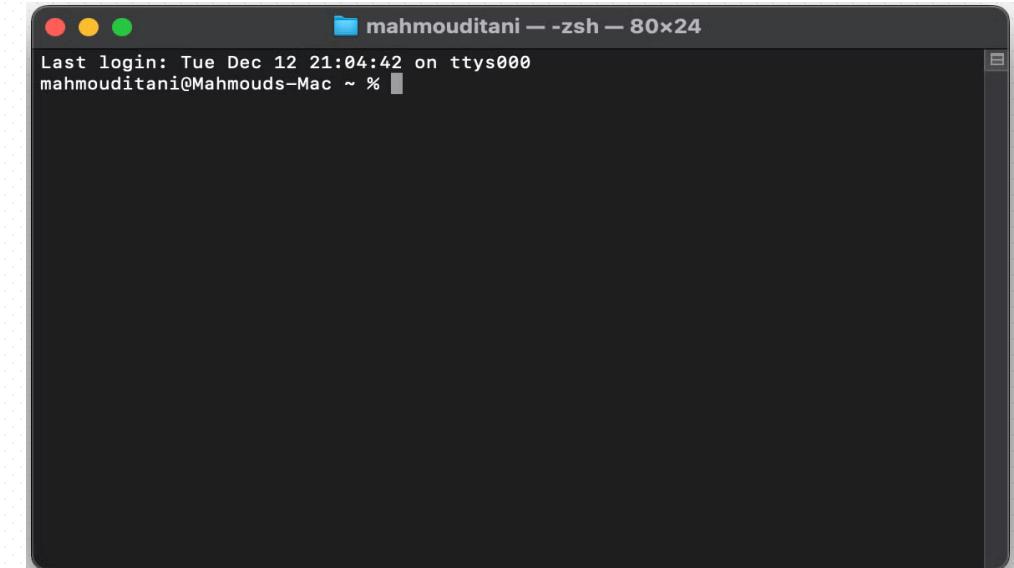
PARTIE 1

Illustrations

Command prompt sur Windows (CMD)



Terminal sur Mac



Différence entre terminal et interface graphique

Terminal

- **Basé sur du texte**, il utilise des lignes de commande pour interagir avec le système.
- Offre un **contrôle précis et puissant**, idéal pour les développeurs et administrateurs système.
- Permet **l'automatisation** des tâches grâce à des scripts.
- Nécessite de connaître des **commandes spécifiques**.

Interface graphique

- **Basée sur des éléments visuels** comme des boutons, des icônes et des menus.
- **Plus intuitive et accessible** pour les débutants.
- **Moins flexible** pour les tâches avancées et l'automatisation.
- Peut être **plus gourmande en ressources** système.

INTRODUCTION AU
TERMINAL DE COMMANDE

PARTIE 2

LES COMMANDES DE BASE

COMMANDES DE BASE

Les commandes à savoir pour la navigation

Action	Windows (Powershell/CMD)	MAC (Terminal)
Afficher le répertoire actuel	cd ou cd .	pwd
Changer de répertoire	cd nomDossier	cd nomDossier
Revenir au répertoire parent	cd ..	cd ..
Lister les fichiers	dir	ls

COMMANDES DE BASE

Les commandes à savoir pour la gestion des fichiers

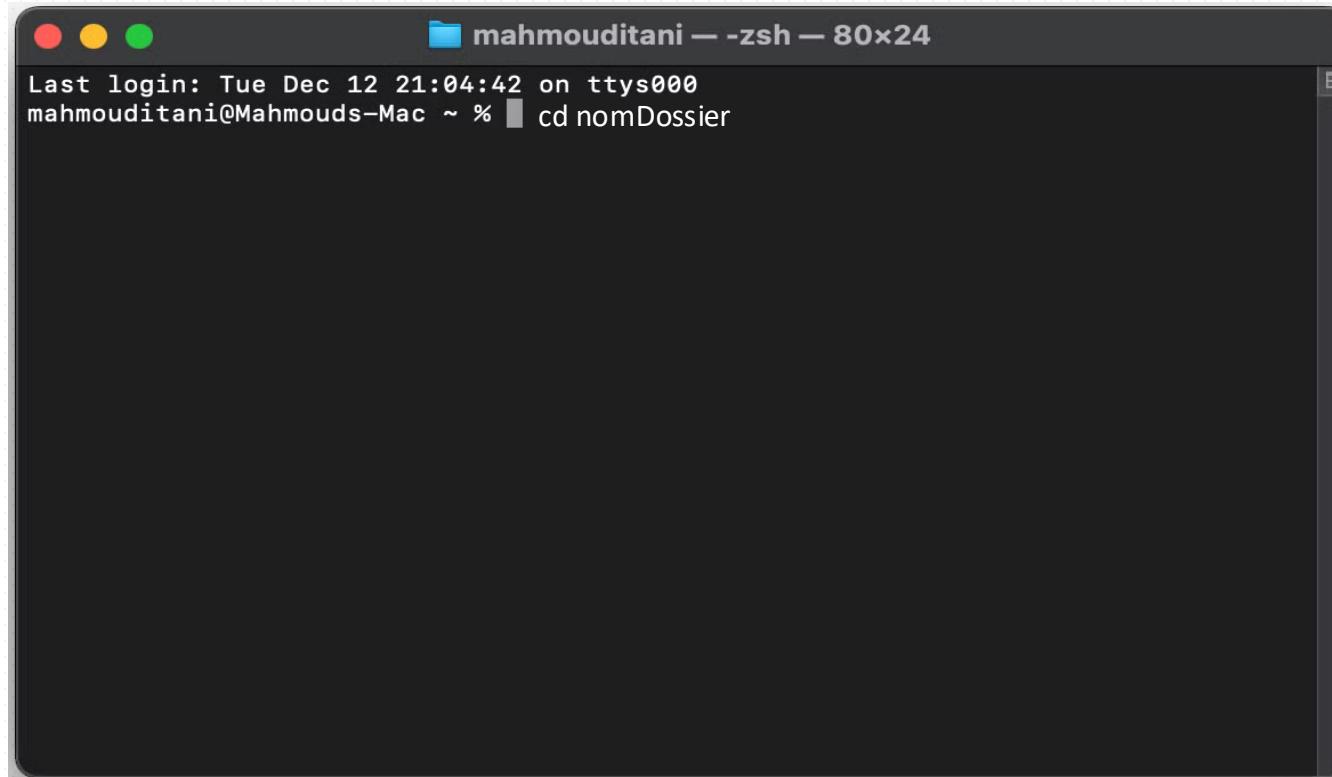
Action	Windows (Powershell/CMD)	MAC (Terminal)
Créer un fichier vide	type nul > fichier.txt	touch fichier.txt
Créer un dossier	mkdir nomDossier	mkdir nomDossier
Supprimer un fichier	del fichier.txt	rm fichier.txt
Supprimer un dossier	rmdir /s /q nomDossier	rm -r nomDossier
Copier un fichier ou un dossier	copy fichier.txt copie.txt	cp fichier.txt copie.txt
Déplacer ou renommer un fichier	move fichier.txt nouveau.tct	mv fichier.txt nouveau.txt

COMMANDES DE BASE

Les commandes à savoir pour le contrôle des processus

Action	Windows (Powershell/CMD)	MAC (Terminal)
Afficher les processus actifs	cd ou cd .	pwd
Arrêter un processus (par PID)	cd nomDossier	cd nomDossier
Forcer l'arrêt d'un processus	cd ..	cd ..

Mise en pratique



```
mahmouditani — -zsh — 80x24
Last login: Tue Dec 12 21:04:42 on ttys000
mahmouditani@Mahmouds-Mac ~ % cd nomDossier
```

INTRODUCTION AU TERMINAL DE COMMANDE

PARTIE 3

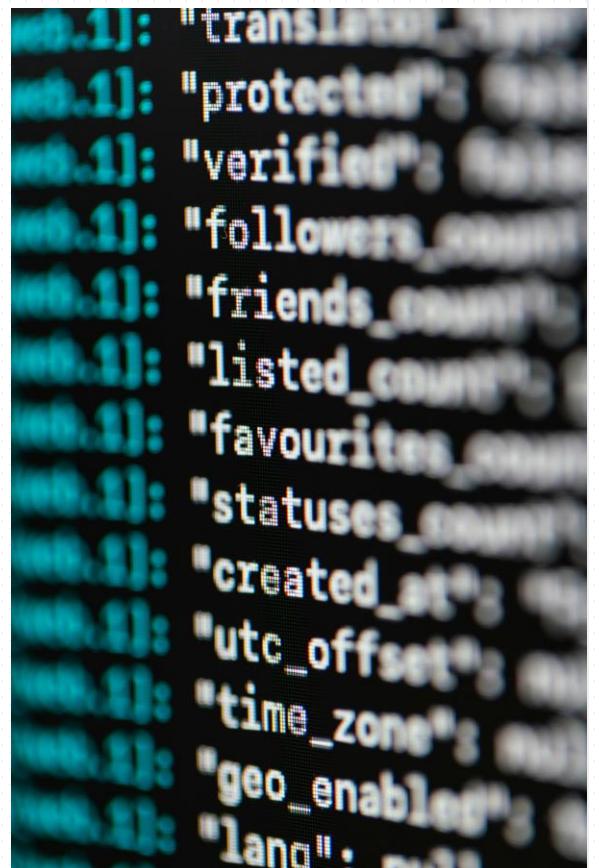
MISE EN PRATIQUE

MISE EN PRATIQUE

Exercice – Créer et naviguer dans une arborescence de projet.

Consignes :

- **Créer un dossier de projet nommé MonProjet.**
- **Se déplacer** dans ce dossier.
- **Créer une structure de sous-dossiers** avec les dossiers suivants :
- **src** (code source)
- **assets** (images et fichiers CSS)
- **docs** (documentation)
- **Créer un fichier** dans chaque dossier :
- Un fichier **index.html** dans **src**.
- Un fichier **style.css** dans **assets**.
- **Vérifier la structure** de l'arborescence créée.
- **Explorer la navigation** en se déplaçant entre les répertoires et en revenant au répertoire parent.



PARTIE 2

Differences Client/Serveur et Front/Back

Dans le développement web, il est essentiel de comprendre les rôles et les interactions entre les différentes parties d'une application. Deux distinctions fondamentales structurent cette architecture : **Client/Serveur** et **Front/Back**.

Ces concepts définissent la manière dont les données sont traitées, affichées et échangées entre les utilisateurs et le système. Cette section explore ces différences pour mieux saisir le fonctionnement et l'organisation des applications modernes.



CLIENTS vs SERVEUR



FRONT-END vs BACK-END



ILLUSTRATIONS



EXERCICE PRATIQUE

DIFFÉRENCES ENTRE CLIENT/SERVEUR & FRONT/BACK

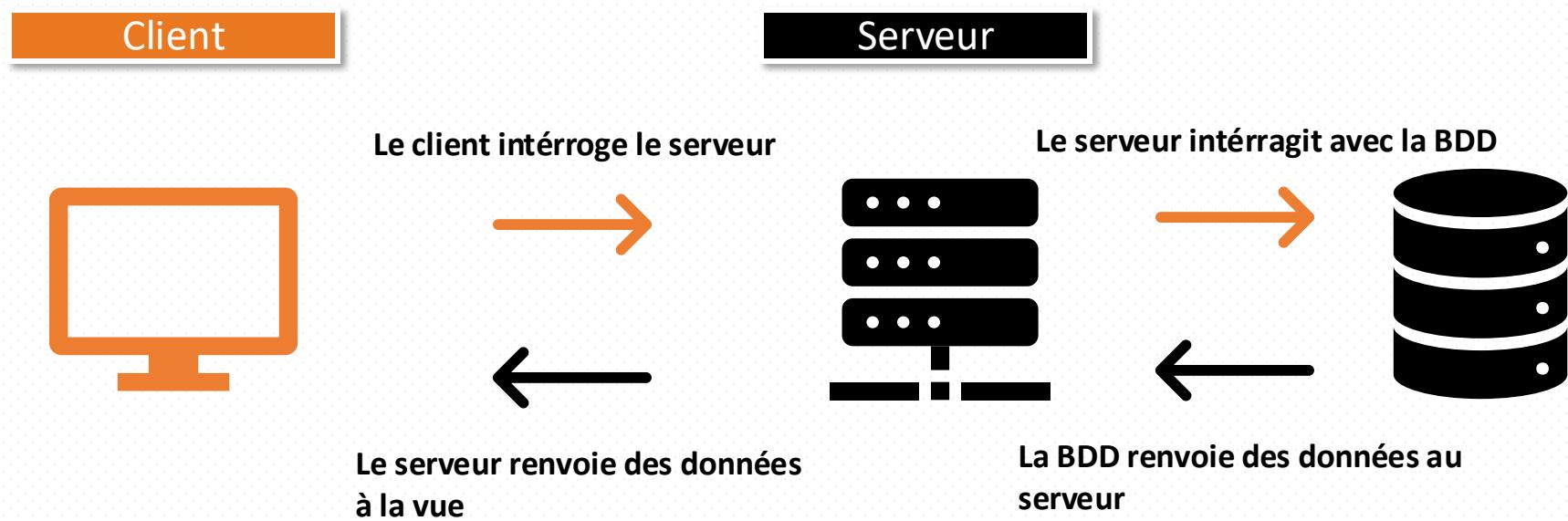
PARTIE 1

CLIENT VS SERVEUR

CLIENT VS SERVEUR

Qu'est-ce qu'un client?

Un client est un dispositif ou une application qui **envoie des requêtes à un serveur** pour accéder à des services, des données ou des ressources. Il peut s'agir d'un **navigateur web**, d'une **application mobile** ou d'un **logiciel installé** sur un ordinateur. Le client est **responsable de l'affichage des informations** et de l'**interaction avec l'utilisateur**. Il **communique avec le serveur via des protocoles réseau**, comme **HTTP**, pour récupérer ou envoyer des données.



CLIENT VS SERVEUR

Qu'est-ce qu'un serveur?

Un serveur est un **ordinateur ou un logiciel** conçu pour fournir des **services**, des **ressources** ou des données à d'autres appareils, appelés clients. Il **répond aux requêtes des clients** en exécutant des actions spécifiques, comme l'**envoi de pages web**, le **stockage de fichiers** ou la **gestion de bases de données**. Les serveurs fonctionnent en continu pour garantir un accès constant aux informations et services qu'ils hébergent.

Exemples de serveurs connus :

- **Apache HTTP Server** – Très utilisé pour héberger des sites web.
- **Nginx** – Performant pour la gestion de trafic élevé.
- **Microsoft IIS** – Serveur web de Microsoft pour Windows.



DIFFÉRENCES ENTRE CLIENT/SERVEUR & FRONT/BACK

PARTIE 2

FRONT VS BACK

FRONT END VS BACKEND

Front-end ?

Le **front-end** correspond à la partie visible et interactive d'une application, celle avec laquelle l'utilisateur interagit directement. Il est exécuté dans le navigateur et utilise principalement :

- **Langages** : HTML, CSS, JavaScript.
- **Frameworks et bibliothèques** : React, Angular, Vue.js...

Exemples :

- Boutons, formulaires, menus, et mises en page.
- Affichage des données récupérées depuis le backend.

HTML

CSS

JS

FRONT END VS BACKEND

Back-end ?

Le **backend** gère la logique métier, les bases de données et les interactions avec le serveur. Il traite les requêtes envoyées par le front-end et renvoie les réponses appropriées.

Technologies utilisées :

- **Langages** : PHP, Python, Java, Node.js, Ruby.
- **Bases de données** : MySQL, PostgreSQL, MongoDB.
- **Frameworks** : Laravel, Django, Express.js.

Exemples :

- Authentification des utilisateurs.
- Gestion des bases de données et des API.
- Traitement et validation des données.

PHP

PYTHON

RUBY

DIFFÉRENCES ENTRE CLIENT/SERVEUR & FRONT/BACK

PARTIE 3

ILLUSTRATIONS

CLIENT VS SERVEUR

En résumé



Front-end (Côté client)

- Partie visible et interactive de l'application.
- Exécuté dans le navigateur.
- **Langages** : HTML, CSS, JavaScript.
- **Frameworks** : React, Angular, Vue.js.
- **Rôle** : Affichage des données, gestion de l'interface utilisateur (UI) et interactions.



Back-end (Côté serveur)

- Partie invisible qui gère la logique, les données et les requêtes.
- Exécuté sur un serveur.
- **Langages** : PHP, Python, Java, Node.js, Ruby.
- **Bases de données** : MySQL, PostgreSQL, MongoDB.
- **Rôle** : Traitement des données, authentification, gestion des API et stockage des

DIFFÉRENCES ENTRE CLIENT/SERVEUR & FRONT/BACK

PARTIE 4

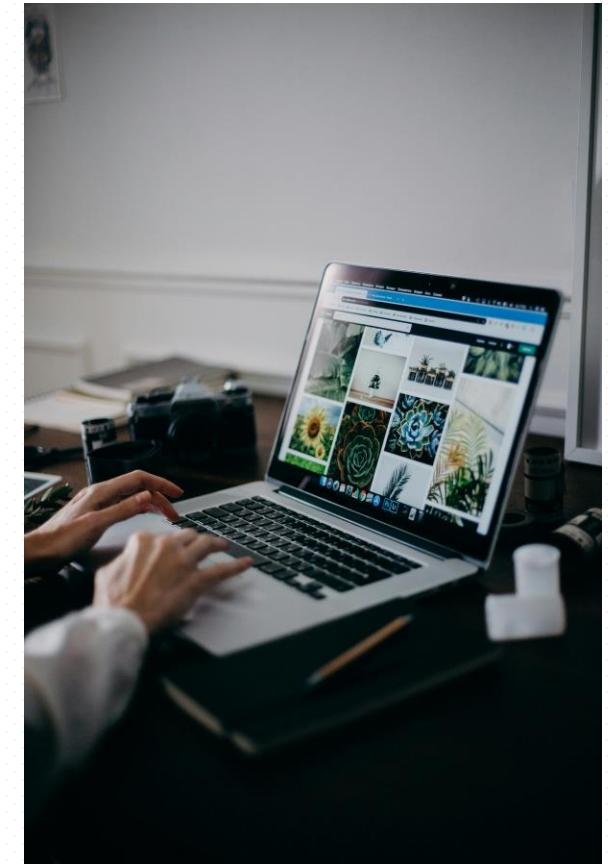
MISE EN PRATIQUE

MISE EN PRATIQUE

Exercice – Identifier les rôles dans une application web

Consignes :

- **Analysez une application web** (par exemple, un site e-commerce ou un blog).
- **Identifiez les éléments visibles** par l'utilisateur et précisez s'ils relèvent du **frontend**.
- **Repérez les actions en arrière-plan** (traitement des données, gestion des utilisateurs, stockage des informations) et associez-les au **backend**.
- **Classez les technologies** utilisées pour chaque partie (frontend et backend).



PARTIE 3

INTRODUCTION AU LANGAGE PHP

PHP est un **langage de programmation côté serveur** utilisé pour créer des pages web dynamiques. Il s'intègre facilement avec HTML et les bases de données, ce qui en fait un choix populaire pour le développement web. Dans les prochaines parties vous découvrirez les bases de PHP et son utilité dans la création de sites web modernes.



HISTOIRE DE PHP



SYNTAXE DE BASE



FONCTIONS



EXERCICE PRATIQUE

INTRODUCTION AU LANGAGE PHP

PARTIE 1

HISTOIRE DE PHP

HISTOIRE DE PHP

Son histoire

PHP (Hypertext Preprocessor) est un langage de programmation côté serveur largement utilisé pour le développement web. Crée en **1995**, il est conçu pour générer des pages web dynamiques et interactives. Facile à apprendre et flexible, PHP s'intègre parfaitement avec **HTML** et les bases de données comme **MySQL**.

Ce langage est particulièrement apprécié pour la création de sites web, d'applications e-commerce et de systèmes de gestion de contenu tels que WordPress et **+80% des sites web sur internet utiliseraient ce langage.**

Ses versions :

- **1995** - PHP 1.0
- **1997** - PHP 3.0
- **2000** - PHP 4.0
- **2004** - PHP 5.0
- **2015** - PHP 7.0
- **2020** - PHP 8.0

HISTOIRE DE PHP

Versions majeures et évolutions

1 PHP 3.0 (1997)

- Première version officielle.
- Ajout du support des bases de données et des fonctionnalités orientées objet.

2 PHP 4.0 (2000)

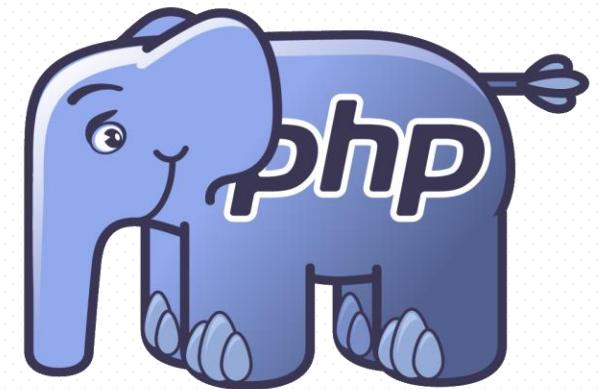
- Introduction du moteur **Zend** pour de meilleures performances.
- Gestion des sessions et amélioration des formulaires.

3 PHP 5.0 (2004)

- Amélioration de la programmation orientée objet (POO).
- Introduction de **PDO** pour la gestion des bases de données.
- Support des exceptions et des interfaces.

4 PHP 7.0 (2015)

- Optimisation des performances et réduction de la consommation mémoire.
- Introduction des **types scalaires** et du **retour typé**.
- Suppression des anciennes fonctionnalités obsolètes.



HISTOIRE DE PHP

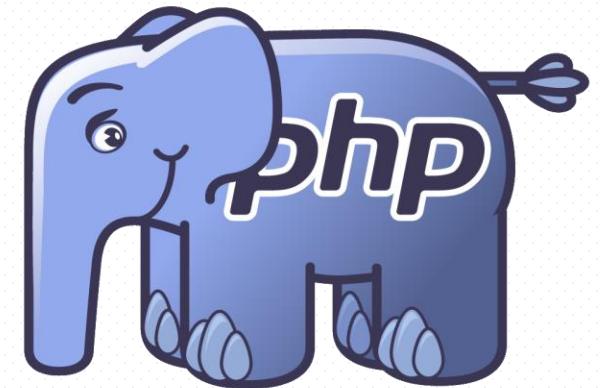
Versions majeures et évolutions

5 PHP 8.0 (2020)

- **Améliorations des performances** grâce au compilateur JIT (Just-In-Time).
- **Types d'union** permettant de spécifier plusieurs types possibles pour un argument ou un retour.
- **Attributs (annotations)** pour remplacer les commentaires PHPDoc par des métadonnées natives.
- **Expressions match**, une alternative plus concise et sécurisée à switch.
- **Opérateur nullsafe (? ->)** pour éviter les erreurs sur les propriétés ou méthodes nulles.
- **Arguments nommés**, offrant plus de flexibilité dans l'ordre et l'omission des paramètres.
- **Validation plus stricte des types** pour renforcer la sécurité et la robustesse du code.

Pour plus d'informations sur les variables :

<https://www.php.net/manual/fr/faq.general.php>



INTRODUCTION AU LANGAGE PHP

PARTIE 2

SYNTAXE DE BASE

SYNTAXE DE BASE

Les Variables

Une variable est un **espace mémoire permettant de stocker des données**, comme du texte ou des nombres. En PHP, une variable commence toujours par le symbole \$.

Types de données :

PHP prend en charge plusieurs types de données :

- **Chaîne de caractères (string)** pour du texte.
- **Entiers (integer)** et **décimaux (float)** pour les nombres.
- **Booléens (boolean)** pour vrai ou faux.

```
$nom = "Jean"; // Chaîne de caractères  
$age = 25;      // Nombre entier
```

Opérateurs :

PHP utilise également des opérateurs pour effectuer des calculs ou des comparaisons :

- **Arithmétiques** : +, -, *, / pour les calculs.
- **Comparaison** : ==, !=, <, > pour vérifier des conditions.
- **Logiques** : && (ET), || (OU), ! (NON) pour combiner des tests.

Pour plus d'informations sur les variables :

<https://www.php.net/manual/en/language.variables.php>

SYNTAXE DE BASE

Guillemets simples vs Guillemets doubles

En PHP, les guillemets simples (' ') et doubles (" ") servent à délimiter des chaînes de caractères, mais ils ne se comportent pas de la même manière.

- **Guillemets simples (')** : Ce sont les plus basiques. Ce que tu écris à l'intérieur est pris **tel quel**. Par exemple :

```
echo 'Salut $nom'; // Affiche : Salut $nom
```

- **Guillemets doubles ("")** : Ils sont plus **flexibles**. Ils permettent d'interpréter les variables et d'exécuter des séquences spéciales comme les sauts de ligne (\n) à l'intérieur.
Exemple :

```
$nom = "Alice";  
echo "Salut $nom"; // Affiche : Salut Alice
```

SYNTAXE DE BASE

Les instructions d'affichage

L'affichage des données est une étape essentielle en PHP pour vérifier les résultats d'un traitement ou afficher des informations à l'utilisateur.

PHP propose **plusieurs fonctions pour afficher des valeurs**, chacune ayant ses particularités.

1. echo : Affichage rapide

L'instruction echo est la méthode la plus courante pour afficher du texte ou des variables. Elle est rapide, accepte plusieurs arguments et est idéale pour afficher des données simples.

```
echo "Bonjour !";
echo "Nom : ", "Jean";
```

2. print : Retour de valeur

Similaire à echo, mais retourne toujours une valeur (1), ce qui permet de l'utiliser dans des expressions. Il est légèrement plus lent mais utile dans certains cas.

```
print "Bienvenue !";
$result = print "Test"; // Retourne 1.
```

SYNTAXE DE BASE

Les Boucles

Les boucles permettent de **répéter un bloc de code tant qu'une condition est respectée**. Elles sont utiles pour automatiser des tâches répétitives.

La boucle **while()**, va répéter le code qu'elle contient, tant que sa condition est vraie.

La boucle **for()**, initie un compteur, qu'elle incrémente à chaque itération, jusqu'à ce que la condition soit remplie.

Les instructions **Break** (Arrêt) et **Continue** (Retour au début) peuvent être utiliser dans une boucle si nécessaire.

Attention à bien spécifier une condition atteignable ou cela peut faire planter votre programme.

Pour plus d'informations sur les boucles :

<https://www.php.net/manual/en/control-structures.for.php>

```
for ($i = 1; $i <= 5; $i++) {  
    echo $i . " ";  
}
```

```
$i = 1;  
while ($i <= 5) {  
    echo $i . " ";  
    $i++;  
}
```

SYNTAXE DE BASE

Les Conditions

Les conditions permettent d'**exécuter des blocs de code spécifiques en fonction des situations.**

On utilise pour cela les opérateurs « if », «else if » et « else ».

On peut utiliser également l'instruction **Switch Case** pour exécuter du code en fonction de la valeur d'une variable.

Ce code vérifie si une personne est majeure ou mineure.

```
$age = 18;  
if ($age >= 18) {  
    echo "Majeur";  
} else {  
    echo "Mineur";  
}
```

Ce code teste plusieurs cas possibles et agit en fonction du jour.

```
$jour = "lundi";  
switch ($jour) {  
    case "lundi":  
        echo "Début de semaine.";  
        break;  
    case "vendredi":  
        echo "Fin de semaine.";  
        break;  
    default:  
        echo "Jour classique.";  
}
```

SYNTAXE DE BASE

Les opérateurs logiques

Les **opérateurs de comparaison** sont essentiels dans les **conditions** pour évaluer des expressions logiques et **exécuter des blocs de code** en fonction du résultat (vrai ou faux).

Opérateur	Signification	Exemple	Résultat
<code>==</code>	Egal à	<code>\$a == \$b</code>	Vrai si \$a est égal à \$b
<code>!=</code>	Différent de	<code>\$a != \$b</code>	Vrai si \$a est différent de \$b
<code><</code>	Inférieur à	<code>\$a < \$b</code>	Vrai si \$a est inférieur à \$b
<code>></code>	Supérieur à	<code>\$a > \$b</code>	Vrai si \$a est supérieur à \$b
<code><=</code>	Inférieur ou égal à	<code>\$a <= \$b</code>	Vrai si \$a est inférieur ou égal à \$b
<code>>=</code>	Supérieur ou égal à	<code>\$a >= \$b</code>	Vrai si \$a est supérieur ou égal à \$b

SYNTAXE DE BASE

Les opérateurs logiques - exemples

Les opérateurs `>=`, `<` et `&&` permettent d'évaluer ici plusieurs critères pour exécuter le code sous condition :

```
if ($note >= 10 && $note < 15) {
    echo "Passable";
} elseif ($note >= 15) {
    echo "Bien";
} else {
    echo "Échec";
}
```

Pour plus d'informations sur les opérateurs logiques en PHP :

<https://www.php.net/manual/fr/language.operators.logical.php>

SYNTAXE DE BASE

Les Tableaux

Les tableaux permettent de stocker plusieurs variables dans une même variable. On va pouvoir ensuite ajouter des valeurs dans le tableau et accéder aux différentes valeurs qu'il contient.

nomDuTableau[0] permet d'accéder à la première valeur du tableau.

On peut utiliser **foreach()** pour effectuer des actions sur chaque valeur du tableau en PHP.

Pour plus d'informations sur les tableaux en PHP:

<https://www.php.net/manual/fr/language.types.array.php>

```
$fruits = ["Pomme", "Banane", "Orange"];
echo $fruits[1]; // Affiche "Banane"
```

```
foreach ($fruits as $index => $fruit) {
    echo "Index $index : $fruit <br>";
}
```

SYNTAXE DE BASE

Les instructions d'affichage adaptées aux tableaux

3. print_r : Affichage des structures complexes

Cette fonction est utilisée pour afficher des tableaux ou des objets dans un format lisible. Elle est utile pour le débogage.

```
$arr = [1, 2, 3];
print_r($arr);
```

4. var_dump : Informations détaillées

Contrairement à print_r, la fonction var_dump fournit des informations détaillées sur la structure et le type des variables.

```
$val = 42;
var_dump($val);
```

SYNTAXE DE BASE

Les Tableaux associatifs

Un tableau associatif **utilise des index personnalisés pour organiser ses éléments**, ce qui facilite leur identification. Ces tableaux sont idéaux pour organiser des informations complexes et les manipuler facilement.

Pour plus d'informations sur les tableaux en PHP:

<https://www.php.net/manual/fr/language.types.array.php>

```
$personne = [  
    "nom" => "Jean",  
    "age" => 25,  
    "ville" => "Paris"  
];  
echo $personne["nom"]; // Affiche "Jean"
```

SYNTAXE DE BASE

Les Tableaux multidimensionnels

En PHP, un tableau multidimensionnel est un **tableau qui contient un ou plusieurs tableaux en tant qu'éléments**. Il permet de stocker des données sous forme de structure à plusieurs niveaux.

On accède aux valeurs en utilisant plusieurs indices, comme **\$tableau[0][1]**, ce qui facilite l'organisation et la manipulation des données hiérarchiques.

```
$etudiants = [
    ["Nom" => "Alice", "Age" => 20, "Note" => 15],
    ["Nom" => "Bob", "Age" => 22, "Note" => 17],
    ["Nom" => "Clara", "Age" => 21, "Note" => 16]
];

// Affichage d'un élément spécifique
echo $etudiants[1]["Nom"]; // Affiche "Bob"
```

Pour plus d'informations sur les tableaux en PHP:

<https://www.php.net/manual/fr/language.types.array.php>

INTRODUCTION AU LANGAGE PHP

PARTIE 3

Les Fonctions

LES FONCTIONS

Les Fonctions prédefinies - définition et appel

Les fonctions sont des **blocs de code réutilisables** conçus **pour effectuer une tâche spécifique**. En PHP, elles permettent de structurer, d'optimiser et de rendre le code plus lisible. Une fonction peut recevoir des **paramètres** (informations en entrée) et retourner un **résultat**. Il existe des fonctions pré-définies disponibles dans le langage php et des fonctions utilisateur créées par l'utilisateur lui-même.

Par exemple, la **fonction native if()** sert à vérifier une condition. Les données ajoutées entre les parenthèses d'une fonction sont appelées "**Paramètres**".

Il existe évidemment une multitude de fonctions prédefinies en PHP.

Voici des exemples de fonctions pré-définies en PHP :

Pour plus d'informations sur les fonctions prédefinies en PHP:

<https://www.php.net/manual/fr/indexes.functions.php>

```
$texte = "Bonjour";
echo strlen($texte); // Affiche 7

$texte = "bonjour";
echo strtoupper($texte); // Affiche "BONJOUR"

$notes = [15, 18, 12];
echo array_sum($notes); // Affiche 45
```

LES FONCTIONS

Les Fonctions utilisateur - définition et appel

Les fonctions sont des **blocs de code réutilisables conçus pour effectuer une tâche spécifique**. En PHP, elles permettent de structurer, d'optimiser et de rendre le code plus lisible. Une fonction peut recevoir des **paramètres** (informations en entrée) et retourner un **résultat**.

Une fonction est créée avec le mot-clé **function**, suivie de son nom et de parenthèses.

Attention !

Toute déclaration de variable au sein d'une fonction n'existera qu'au sein de celle-ci.

- Définition

```
function nomDeLaFonction() {  
    // Instructions  
}
```

- Appel

```
function saluer() {  
    echo "Bonjour !";  
}  
  
saluer(); // Affiche "Bonjour !"
```

LES FONCTIONS

Les Fonctions utilisateur - paramètres

Un paramètre est une **variable définie dans les parenthèses lors de la déclaration d'une fonction**. Ces variables reçoivent les valeurs fournies lors de l'appel de la fonction.

Vous pouvez avoir plusieurs paramètres dans une fonction.

Paramètres avec valeur par défaut :

On peut attribuer une valeur par défaut à un paramètre. Si aucune valeur n'est fournie lors de l'appel, cette valeur sera utilisée.

- Paramètre avec valeur par défaut

```
function direBonjour($nom = "Invité") {
    echo "Bonjour, $nom !";
}

direBonjour();          // Affiche : Bonjour, Invité !
direBonjour("Marie");  // Affiche : Bonjour, Marie !
```

- Définition

```
function nomFonction($param1, $param2) {
    // Code utilisant les paramètres
}
```

- Appel

```
function saluer($nom) {
    echo "Bonjour, $nom !";
}

saluer("Alice"); // Affiche : Bonjour, Alice !
saluer("Paul"); // Affiche : Bonjour, Paul !
```

LES FONCTIONS

Les Fonctions utilisateur - return

Les fonctions qui retournent une valeur permettent d'envoyer un résultat au programme qui les appelle. **Ce résultat peut ensuite être utilisé, stocké ou manipulé.** Ces fonctions sont particulièrement utiles pour effectuer des calculs ou transformer des données.

Définition et Syntaxe :

Une fonction utilise le mot-clé **return** pour renvoyer une valeur. Une fois que la commande **return** est exécutée, l'exécution de la fonction s'arrête.

- **Définition**

```
function nomFonction($param1, $param2) {  
    // Traitement  
    return $resultat;  
}
```

- **Retourne une chaîne de caractères**

```
function saluer($nom) {  
    return "Bonjour, $nom !";  
}  
echo saluer("Alice"); // Affiche "Bonjour, Alice !"
```

LES FONCTIONS

Les Fonctions utilisateur - résumé

- Une fonction commence par le mot-clé **function**.
- Elle peut accepter des **paramètres** pour recevoir des données.
- Elle peut utiliser **return** pour renvoyer un résultat.
- Les fonctions rendent le code **réutilisable** et **plus organisé**.

Mise en pratique

- Créez une fonction qui calcule le **carré d'un nombre et retourne le résultat**.
- Ajoutez un **deuxième paramètre pour multiplier le carré par un autre nombre**.
- Affichez le résultat final.



INTRODUCTION AU LANGAGE PHP

PARTIE 4

Mise en pratique

MISE EN PRATIQUE

Exercice pratique

- Crée une fonction appelée **calculerMoyenne** qui :
- Accepte trois nombres en paramètres.
- Retourne leur moyenne.
- Crée une deuxième fonction appelée **afficherResultat** qui :
- Accepte un nom et une moyenne en paramètres.
- Affiche un message indiquant si la moyenne est suffisante (≥ 10) ou insuffisante (< 10).
- Appelle les deux fonctions avec des exemples :
- Calculer la moyenne de trois notes.
- Afficher le résultat pour un étudiant nommé "Alice".

Objectifs :

- ❖ Utiliser des paramètres pour passer des valeurs aux fonctions.
- ❖ Retourner des résultats avec le mot-clé **return**.
- ❖ Utiliser des conditions pour afficher un message personnalisé.



PARTIE 4

INSTALLER UN LIBRAIRIE PHP AVEC COMPOSER

Dans le développement PHP moderne, l'utilisation de bibliothèques externes permet de **gagner du temps et d'ajouter des fonctionnalités avancées**.

Composer simplifie l'installation et la gestion de ces bibliothèques en PHP, rendant les projets plus modulaires et faciles à maintenir.



QU'EST-CE QU'UNE
LIBRAIRIE?



QU'EST-CE QUE
COMPOSER?



INSTALLATION ET
UTILISATION



EXERCICE PRATIQUE

INSTALLER UNE LIBRAIRIE PHP AVEC COMPOSER

PARTIE 1

Qu'est-ce que Composer?

QU'EST-CE QUE COMPOSER?

Qu'est-ce qu'une librairie ?

Une **librairie** en PHP est un ensemble de fichiers contenant des **fonctions**, des **classes** ou des **modules** préécrits, conçus pour accomplir des tâches spécifiques. Elle permet de réutiliser du code existant pour accélérer le développement et éviter de tout créer de zéro.

Utilité :

- **Simplifie le développement** en fournissant des fonctionnalités prêtes à l'emploi.
- **Facilite la gestion des tâches complexes** (ex. : gestion des bases de données, envoi d'e-mails, intégration d'API).
- **Améliore la maintenabilité et la lisibilité** du code.

QU'EST-CE QUE COMPOSER?

Exemples de librairies PHP

1 **GuzzleHttp** : Gestion des requêtes HTTP

- Effectue des requêtes HTTP pour interagir avec des API.

2 **Carbon** : Manipulation des chaînes de caractères

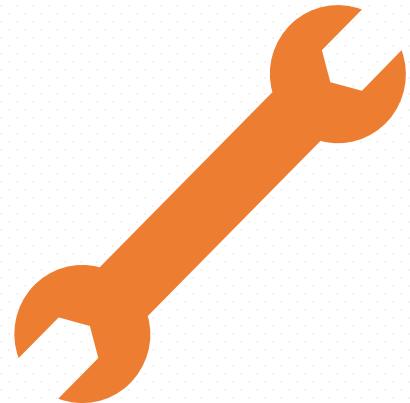
- Gérer et manipuler les dates et heures.

3 **Doctrine** : Gestion des bases de données :

- Utilisé pour l'ORM (Mapping Objet-Relationnel).

4 **Respect/Validation** : Crédation et validation de formulaires

- Valider des données.



QU'EST-CE QUE COMPOSER?

Qu'est-ce que Composer?

Composer est un **gestionnaire de dépendances** pour PHP. Il permet d'ajouter, de gérer et de mettre à jour facilement des **librairies** et des **packages** dans un projet.

Utilité:

- **Simplifie l'intégration** de bibliothèques tierces.
- **Automatise l'installation** des dépendances et leurs mises à jour.
- Génère un fichier **autoload** pour inclure automatiquement les classes.

Comment ça fonctionne ?

Composer télécharge les bibliothèques spécifiées dans un projet et gère leurs versions pour garantir la compatibilité. Il utilise un fichier appelé **composer.json** pour définir les dépendances du projet.



QU'EST-CE QUE COMPOSER?

Installation et utilisation

- **Windows** : Téléchargez et exéquez le programme d'installation depuis getcomposer.org.
- **Mac/Linux** : Utilisez cette commande dans le terminal :

```
curl -sS https://getcomposer.org/installer | php  
mv composer.phar /usr/local/bin/composer
```

Pour plus d'informations sur composer :

<https://getcomposer.org/>

QU'EST-CE QUE COMPOSER?

Les commandes principales

Les commandes sont à lancer depuis votre command prompt (CMD ou Powershell) sur Windows, ou votre terminal sur Mac.

- **Créer un nouveau projet**

```
composer init
```

- **Ajouter une dépendance**

```
composer require guzzlehttp/guzzle
```

- **Mettre à jour une dépendance**

```
composer update
```

- **Installer les dépendances d'un projet existant**

```
composer install
```

QU'EST-CE QUE COMPOSER?

Le fichier **composer.json**

Le fichier **composer.json** est le cœur d'un projet PHP utilisant **Composer**.

Il sert à :

- **Définir les dépendances** nécessaires au projet (librairies).
- **Gérer les versions** compatibles des dépendances.
- **Décrire les métadonnées** du projet (nom, description, auteur).
- **Configurer l'autoloading** pour charger automatiquement les classes.

Pourquoi est-il important?

- Assure la **compatibilité** des versions avec les bibliothèques installées.
- Simplifie la **réPLICATION DU PROJET** sur d'autres machines grâce à un suivi des dépendances.
- Permet une **maintenance facile** en mettant à jour automatiquement les dépendances.

QU'EST-CE QUE COMPOSER?

La composition du fichier composer.json

- 1** **name:** Nom du projet
- 2** **description:** Brève description du projet
- 3** **authors:** Informations sur l'auteur du projet
- 4** **require :** Liste des bibliothèques et leurs versions nécessaires.
- 5** **Autoload :** Configuration pour le chargement automatique des classes selon l'arborescence.

```
{  
    "name": "monprojet/exemple",  
    "description": "Un projet d'exemple avec Composer",  
    "authors": [  
        {  
            "name": "Jean Dupont",  
            "email": "jean.dupont@email.com"  
        }  
    ],  
    "require": {  
        "guzzlehttp/guzzle": "^7.0"  
    },  
    "autoload": {  
        "psr-4": {  
            "App\\": "src/"  
        }  
    }  
}
```

INSTALLER UNE LIBRAIRIE PHP AVEC COMPOSER

PARTIE 2

Mise en pratique

MISE EN PRATIQUE

Mise en pratique – Installer une librairie avec Composer

- Initialiser un projet Composer.
- Installer une librairie dans le projet.
- Vérifier l'installation des fichiers nécessaires.
- Créer un fichier PHP pour tester la librairie installée.
- Utiliser la librairie pour exécuter une action simple.

Objectifs :

- Apprendre à initialiser un projet avec Composer.
- Installer et gérer des dépendances dans un projet PHP.
- Mettre en pratique l'utilisation d'une librairie externe.



PARTIE 5

INTRODUCTION AUX API

Dans un monde connecté, les applications doivent souvent échanger des informations et interagir avec d'autres services.

Les **API** sont devenues indispensables pour faciliter ces échanges et permettre aux systèmes de travailler ensemble efficacement.



QU'EST-CE QU'UNE API?



FONCTIONNEMENT



EXEMPLES

INTRODUCTION AUX API

PARTIE 1

Qu'est-ce qu'une API?

QU'EST-CE QU'UNE API?

Définition

Une **API** ou Interface de Programmation d'Applications, est un ensemble de règles et d'outils permettant à des logiciels de **communiquer entre eux**. Elle définit la manière dont une application peut **demander et recevoir des données** ou des services d'une autre application.

Exemple :

- Une API météo permet à un site web d'afficher les prévisions en récupérant des données d'un service externe.



QU'EST-CE QU'UNE API?

Vous utilisez les API tous les jours !

Par exemple lorsque vous vous connectez sur votre application préférée, on vous propose souvent ceci :



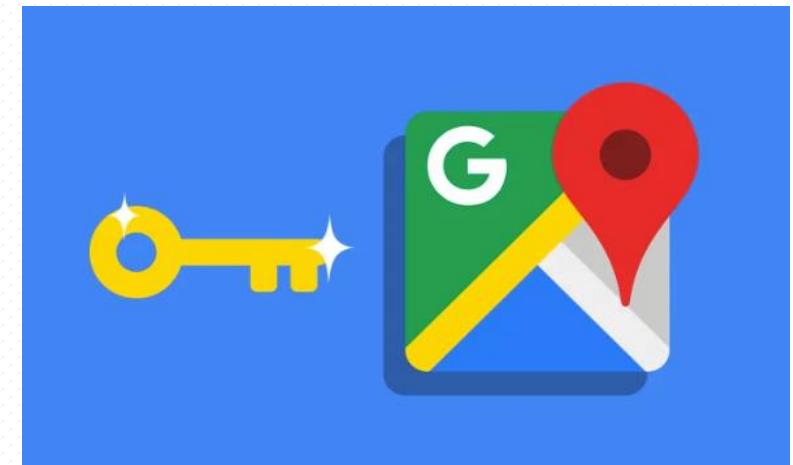
Les développeurs d'API créent des méthodes standardisées, réutilisables et documentées pour permettre à d'autres développeur d'accéder à des données spécifiques lors de la construction d'applications.

QU'EST-CE QU'UNE API?

API publique vs API privée

API publique :

- **Accessible à tous** les développeurs.
- Utilisée pour offrir des services ouverts comme des informations météo, des cartes ou des paiements en ligne.
- **Exemples :**
 - **Google Maps API** – Intégration de cartes interactives.
 - **OpenWeatherMap API** – Données météorologiques.
 - **PokeAPI** – Données sur les pokémon.



QU'EST-CE QU'UNE API?

API publique vs API privée

API privée :

- **Accessible uniquement en interne** par une organisation ou une équipe.
- Utilisée pour connecter des systèmes internes ou automatiser des processus au sein d'une entreprise.
- **Exemple** : Une API utilisée par une application de gestion interne pour accéder aux bases de données des employés.
- **Exemple** :
 - **The Movie Database** – Intégration d'informaiton sur des films/séries

INTRODUCTION AUX API

PARTIE 2

Fonctionnement

FONCTIONNEMENT

Fonctionnement

Une API utilise le protocole **HTTP** pour échanger des données entre un **client** (navigateur ou application) et un **serveur**.

Les méthodes principales :

- **GET** : Récupérer des données.

Exemple : Demander la liste des utilisateurs.

- **POST** : Envoyer des données.

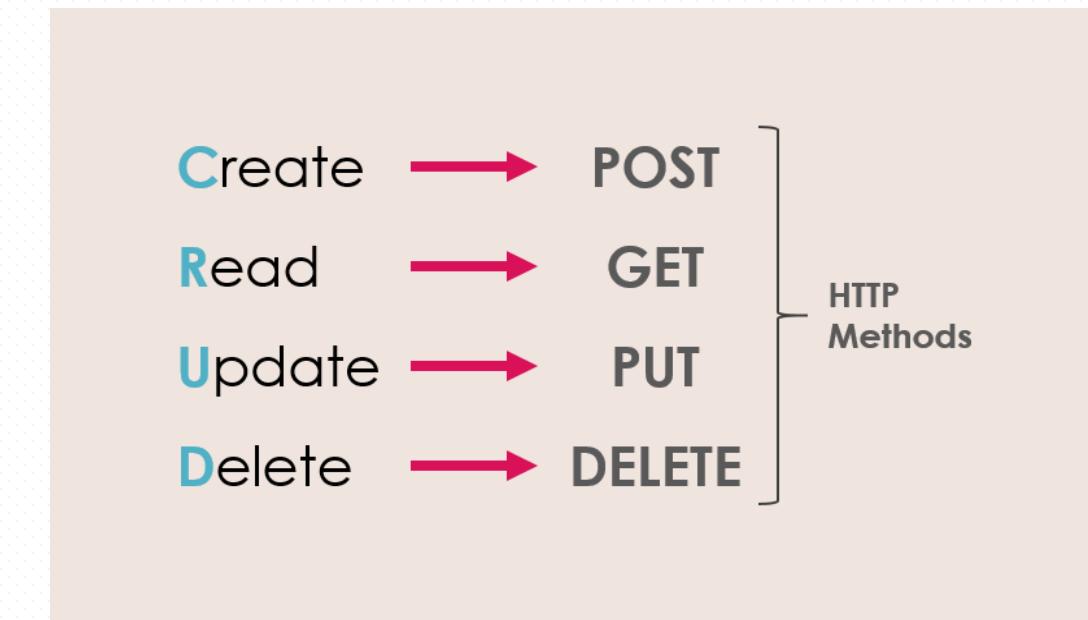
Exemple : Ajouter un nouvel utilisateur.

- **PUT** : Mettre à jour des données existantes.

Exemple : Modifier les informations d'un utilisateur.

- **DELETE** : Supprimer des données.

Exemple : Supprimer un utilisateur.



FONCTIONNEMENT

Mise en pratique

Par exemple, de quelle méthode s'agit-il ? :

- Pierrick souhaite s'inscrire sur mon site, il envoie ses informations pour qu'elles puissent être stockées.
- Pierrick a tapé trop vite et s'est trompé dans son adresse, il envoie de nouvelles informations pour mettre à jour son lieu d'habitation.
- Pierrick consulte sa page profil pour vérifier que tout est bon, il récupère ses informations pour les afficher sur le site.
- Pierrick n'aime finalement pas ce site, il décide de supprimer son compte, et donc ses informations.

FONCTIONNEMENT

Les types de données renvoyées par une API

Les API peuvent renvoyer différents types de données en fonction des besoins et du contexte d'utilisation. Ces formats permettent aux applications de communiquer efficacement et de partager des informations structurées ou non structurées. Parmi les formats les plus courants, on trouve **JSON** et **XML**, souvent utilisés pour les échanges de données, mais aussi des réponses en **texte brut**, **HTML** ou encore des **fichiers** comme des images ou des PDF.

Utilisés pour structurer les données :

- JSON (JavaScript Object Notation)
- XML (Extensible Markup Language)

Utilisés pour des messages courts :

- Texte Brut (Plain Text)
- HTML (HyperText Markup Language)

Utilisés pour renvoyés des contenus multimédias :

- Fichiers (Images, PDF, etc.)

FONCTIONNEMENT

Les codes de réponse HTTP

Les codes de réponse indiquent le **résultat de la requête** effectuée auprès de l'API.

Les méthodes principales :

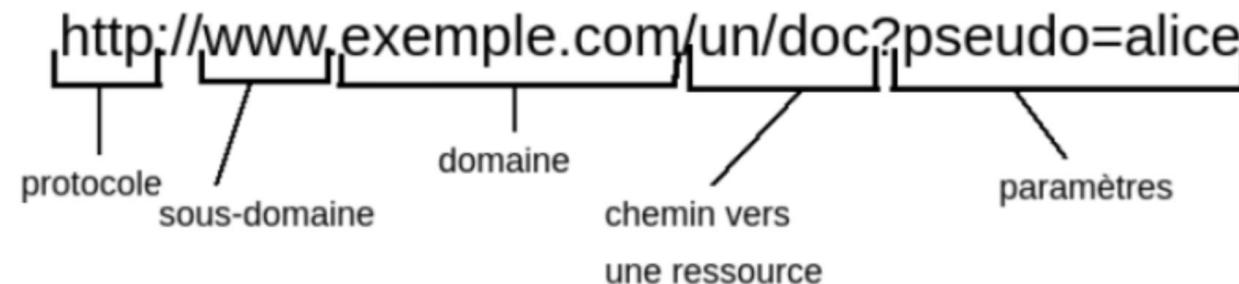
- **200 – OK** : La requête a réussi.
- **201 – Created** : Une nouvelle ressource a été créée.
- **400 – Bad Request** : La requête est invalide ou mal formulée.
- **401 – Unauthorized** : Accès refusé (problème d'authentification).
- **404 – Not Found** : La ressource demandée est introuvable.
- **500 – Internal Server Error** : Erreur interne sur le serveur.



FONCTIONNEMENT

Comment fonctionne une URL?

Vous connaissez probablement la base d'une URL :



- **son protocole**
- **son sous-domaine**
- **son domaine**
- **son chemin**

Mais savez-vous qu'on peut envoyer également des informations complémentaires ?

FONCTIONNEMENT

Les paramètres d'URL

Pour préciser certaines informations qui pourraient être variables, **on peut placer des paramètres dans l'url.**

Vous utilisez ces paramètres tous les jours sans vous rendre compte lorsque vous allez sur n'importe quel site internet.

Lorsque vous effectuez une recherche sur X par exemple :

```
https://x.com/search?q=OnePiece
```

INTRODUCTION AUX API

PARTIE 3

Exemple

EXEMPLES

OpenWeather API : Données Météorologiques

OpenWeather API est une API qui fournit des informations météorologiques en temps réel et des prévisions.

Exemple de requête :

```
GET https://api.openweathermap.org/data/2.5/weather?q=Nice&appid=VOTRE_CLE_API
```

Réponse JSON :

```
{
    "weather": [
        {
            "description": "ciel dégagé",
            "icon": "01d"
        }
    ],
    "main": {
        "temp": 22.5,
        "humidity": 60
    }
}
```

PARTIE 4

Les API privées

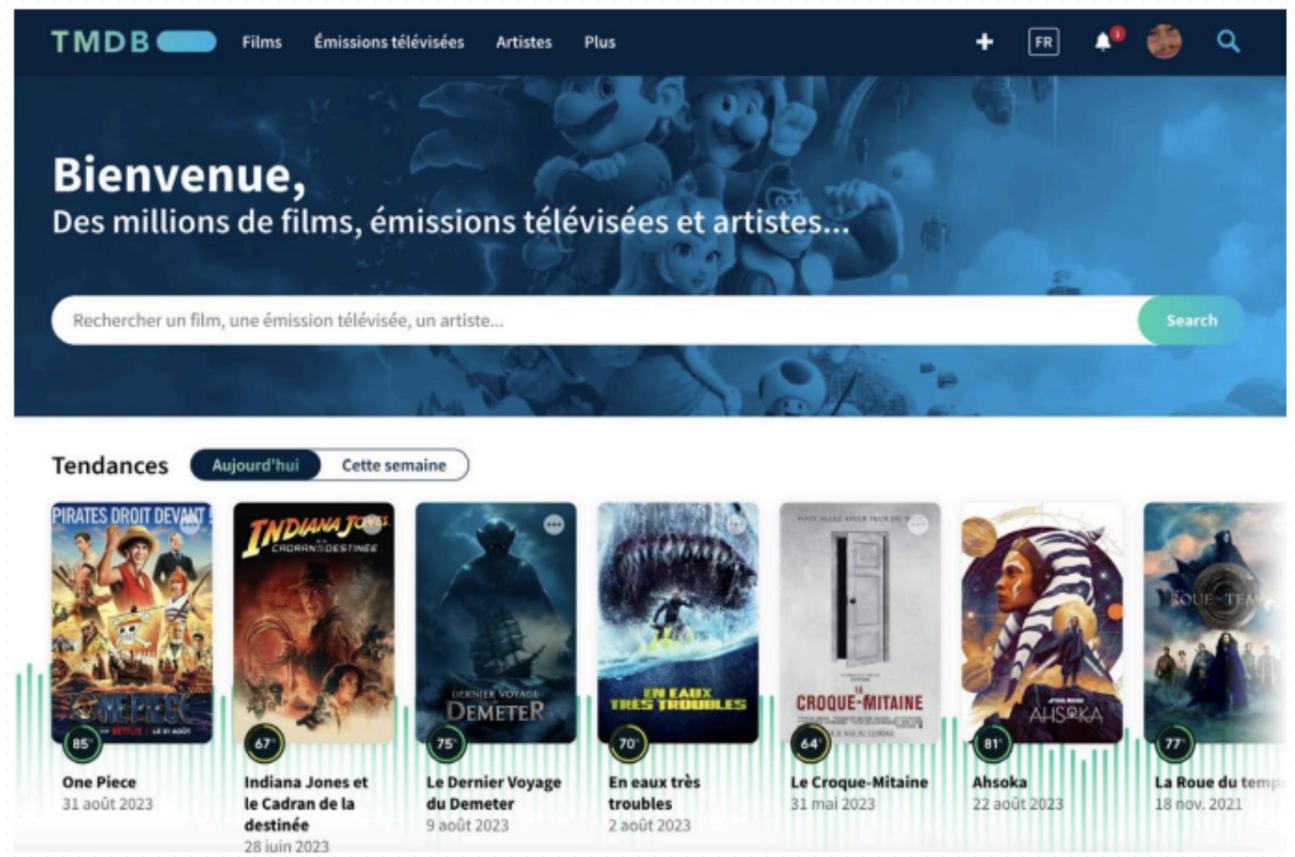
EXEMPLES

TheMovieDatabase

Une API privée **utilise donc un token**, qui est une sorte de clé d'accès qui ressemble à un code wifi beaucoup trop long.

Ce token permet de s'authentifier, et ce besoin d'authentification est souvent lié au nombre limité de requêtes autorisées.

On va utiliser l'API de **TheMovieDatabase** pour notre exemple.



EXEMPLES

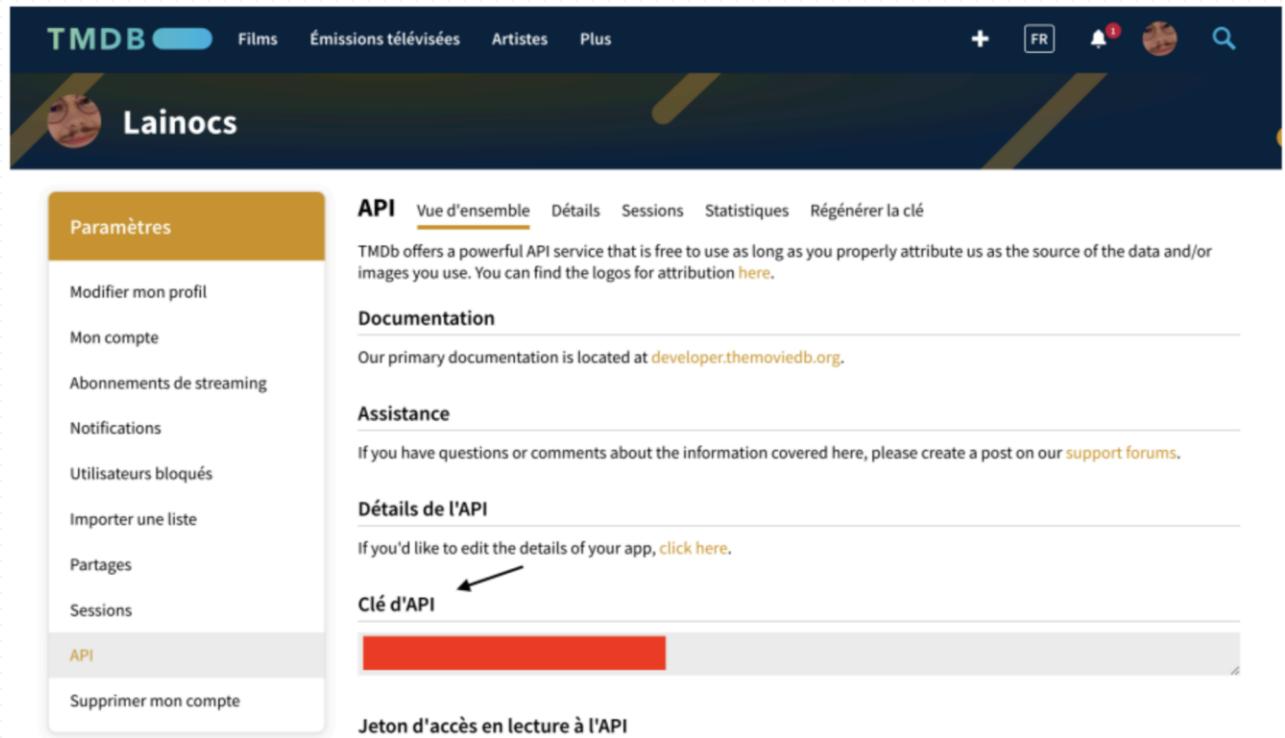
TheMovieDatabase

Pour cela, on va créer un compte TMDB.
Une fois le compte créé, la clé est disponible
dans les paramètres de l'utilisateur.

The Movie Database

On va voir comment l'utiliser
de manière sécurisée.

(On ne va pas la publier sur
GitHub pour éviter que d'autres
utilisent nos crédits)



The screenshot shows the TMDB API key page. At the top, there's a navigation bar with links for Films, Émissions télévisées, Artistes, and Plus. On the right side of the header are icons for adding a new account, switching to French (FR), notifications, profile picture, and search. Below the header, the user's profile picture and name 'Lainocs' are displayed. The main content area has a sidebar titled 'Paramètres' containing links for Modifier mon profil, Mon compte, Abonnements de streaming, Notifications, Utilisateurs bloqués, Importer une liste, Partages, Sessions, API (which is highlighted in yellow), and Supprimer mon compte. To the right of the sidebar, under the 'API' heading, there's a sub-section titled 'Vue d'ensemble' which includes links for Détails, Sessions, Statistiques, and Régénérer la clé. Below this, there's a paragraph about TMDb's API service and a link to developer.themoviedb.org. Further down are sections for Documentation, Assistance, and Détails de l'API, each with its own descriptive text and links. A red arrow points to the 'Clé d'API' section, which contains a large redacted API key.

EXEMPLES

Utilisation de TheMovieDatabase

Pour utiliser cette API, on va devoir **placer le token en paramètre d'URL**.

```
https://api.themoviedb.org/3/movie/1895?api_key=f28lia5fe6aod0mn128o51c22178f344
```

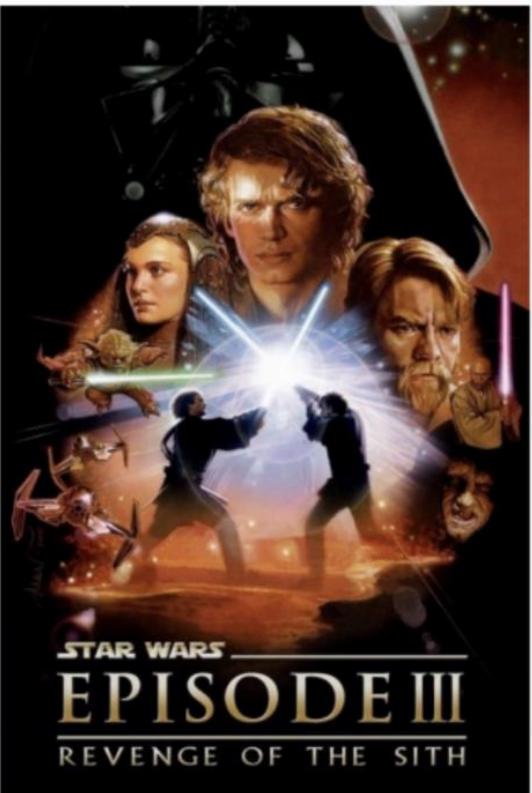
On retrouve tout d'abord dans le chemin la version de l'API, le type de contenu qu'on souhaite rechercher, puis l'identifiant lié au film.

Ensuite on précise notre token avec le paramètre « **api_key** » pour s'authentifier et accéder à l'API.

EXEMPLES

Appel de l'API TheMovieDatabase depuis notre code

Star Wars: Episode III - Revenge of the Sith



```
function fetchStarWars3() {
    return fetch(
        'https://api.themoviedb.org/3/movie/1895?api_key=f28lia5fe6aod0mnl28o51c22178f34'
    )
        .then((response) => response.json())
}

async function displayStarWars3() {
    const data = await fetchStarWars3()
    document.getElementById("star-wars-3").innerHTML =
        <h1>${data.title}</h1>
        
    }

    displayStarWars3()
```

EXEMPLES

Appel de l'API TheMovieDatabase depuis notre code

Cette API propose également un 2ème paramètre qui pourrait nous intéresser. On peut **changer la langue des informations récupérées** ! Pour ça, il faut simplement rajouter le paramètre « language ». On sépare 2 paramètres par un « & ».

```
https://api.themoviedb.org/3/movie/1895?api_key=f28lia5fe6aod0mn128o51c22178f344&language=fr-FR
```

Et on se retrouve maintenant avec toutes les informations traduites en français !
Plein d'autres possibilités sont disponibles.

Vous pouvez essayer librement tout ce qui est proposé sur l'API en cliquant ici [API The Movie Database.](#)

PARTIE 6

CONSOMMATION D'UNE API EXTERNE AVEC GUZZLEHTTP

Dans le développement web, il est souvent nécessaire de récupérer ou d'envoyer des données vers des services externes.

Cette section explore comment consommer une **API externe** en utilisant des outils adaptés pour gérer efficacement les requêtes et les réponses HTTP, par exemple via GuzzleHTTP.



PRÉSENTATION DE
GUZZLEHTTP



INSTALLATION ET
CONFIGURATION



EXEMPLE PRATIQUE:
CONSOMMER UNE API
MÉTÉO



LIEN ENTRE PHP ET JS



EXERCICE PRATIQUE

CONSOMMATION D'UNE API EXTERNE : GUZZLE HTTP

PARTIE 1

Guzzle HTTP

CONSOMMATION D'UNE API EXTERNE : GUZZLE HTTP

Qu'est-ce que Guzzle HTTP?

GuzzleHTTP est une bibliothèque PHP moderne et flexible conçue pour **simplifier l'envoi de requêtes HTTP** et l'intégration d'API externes. Elle permet de **récupérer ou d'envoyer des données rapidement** tout en gérant les réponses, les erreurs et les redirections.

Utilisée dans de nombreux projets, **GuzzleHTTP** facilite l'automatisation des échanges avec des services web et s'adapte aussi bien aux requêtes simples qu'aux appels asynchrones plus complexes.

Cas d'utilisation :

- 1** Consommer une API externe
- 2** Envoyer des données
- 3** Gérer des fichiers
- 4** Automatiser des tâches



CONSOMMATION D'UNE API EXTERNE : GUZZLE HTTP

PARTIE 2

Installation et configuration

CONSOMMATION D'UNE API EXTERNE : GUZZLE HTTP

Via composer

Pour installer **GuzzleHTTP**, vous devez utiliser **Composer**, le gestionnaire de dépendances PHP.

Etapes :

- Ouvrez un terminal dans votre projet.
- Exécutez la commande suivante :

```
composer require guzzlehttp/guzzle
```

Résultat :

- **GuzzleHTTP** sera téléchargé dans le dossier **vendor/**.
- Un fichier **composer.json** sera mis à jour avec la dépendance ajoutée.

CONSOMMATION D'UNE API EXTERNE : GUZZLE HTTP

Exemple de configuration et d'utilisation simple

- Consommation de l'API Jsonplaceholder :

```
require 'vendor/autoload.php'; // Inclure l'autoloader de Composer

use GuzzleHttp\Client;

// Créer un client HTTP
$client = new Client();

// Faire une requête GET
$response = $client->request('GET', 'https://jsonplaceholder.typicode.com/posts/1');

// Obtenir le code de statut HTTP
$statusCode = $response->getStatusCode(); // 200

// Obtenir le corps de la réponse
$body = $response->getBody(); // JSON

// Convertir le JSON en tableau PHP
$data = json_decode($body, true);

// Afficher les données
echo "Titre : " . $data['title'] . "\n";
echo "Contenu : " . $data['body'] . "\n";
```

CONSOMMATION D'UNE API EXTERNE : GUZZLE HTTP

PARTIE 3

Exemple pratique : API météo

INSTALLER UNE LIBRAIRIE PHP AVEC COMPOSER

Mise en pratique – Consommer une API météo

Créer un script PHP :

```
<?php require 'vendor/autoload.php'; // Charger les dépendances

use GuzzleHttp\Client;

// Créer un client HTTP
$client = new Client();

try {
    // Envoyer une requête GET vers l'API météo
    $response = $client->request('GET', 'https://api.openweathermap.org/data/2.5/weather', [
        'query' => [
            'q' => 'Nice', // Ville
            'appid' => 'VOTRE_CLE_API', // Clé API
            'units' => 'metric', // Unités en Celsius
            'lang' => 'fr' // Langue en français
        ]
    ]);

    // Décoder la réponse JSON
    $data = json_decode($response->getBody(), true);

    // Afficher les informations météo
    echo "Météo à " . $data['name'] . " : " . $data['weather'][0]['description'] . "<br>";
    echo "Température : " . $data['main']['temp'] . "°C<br>";
    echo "Humidité : " . $data['main']['humidity'] . "%<br>";
} catch (Exception $e) {
    echo "Erreur : " . $e->getMessage();
}
```

INSTALLER UNE LIBRAIRIE PHP AVEC COMPOSER

Mise en pratique – Consommer une API météo

Résultat attendu :

Météo à Nice : ciel dégagé
Température : 22°C
Humidité : 60%

CONSOMMATION D'UNE API EXTERNE : GUZZLE HTTP

PARTIE 4

Lien entre PHP et JS

LIEN ENTRE PHP ET JAVASCRIPT

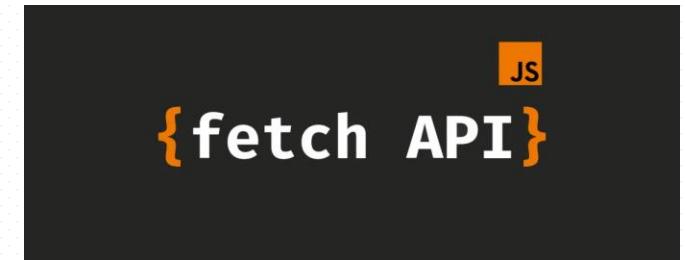
Communication

PHP et JavaScript peuvent communiquer pour **échanger des données** et créer des applications dynamiques. Cette interaction se fait principalement via des **requêtes HTTP** envoyées par JavaScript et traitées par PHP côté serveur.

L'**API Fetch** est une **méthode moderne et simplifiée pour effectuer des requêtes HTTP en JavaScript**. Elle est utilisée pour récupérer des données depuis un serveur ou envoyer des informations à un serveur, souvent dans des applications web interactives.

Avantages de Fetch :

- Syntaxe plus simple que l'ancienne méthode **XMLHttpRequest**.
- Support des **promesses**, facilitant la gestion des requêtes asynchrones.
- Compatible avec **JSON**, le format de données le plus courant pour les API.



LIEN ENTRE PHP ET JAVASCRIPT

Communication

On va utiliser la méthode fetch en 2 parties :

- Syntaxe de base de fetch() : La fonction **fetch()** prend l'URL de la ressource à récupérer comme argument et renvoie une promesse.
- Pour gérer les réponses : La méthode **.then()** sur la promesse renvoyée par **fetch()**.

Syntaxe de base de fetch() pour consommer l'API PokeAPI :

```
fetch('https://pokeapi.co/api/v2/pokemon/ditto')
```

LIEN ENTRE PHP ET JAVASCRIPT

Consommer l'API OpenWeather avec Fetch et PHP

Côté serveur : Créer un fichier php (weather.php)

```
<?php
    // Remplacez par votre clé API OpenWeather
    $apiKey = 'VOTRE_CLE_API';
    $city = $_GET['city'] ?? 'Nice'; // Paramètre par défaut
    $url = "https://api.openweathermap.org/data/2.5/weather?q=$city&appid=$apiKey&units=metric&lang=fr";

    // Faire la requête à l'API OpenWeather
    $response = file_get_contents($url);
    if ($response === FALSE) {
        http_response_code(500);
        echo json_encode(['error' => 'Erreur lors de la récupération des données.']);
        exit;
    }

    // Retourner la réponse JSON
    header('Content-Type: application/json');
    echo $response;

?>
```

LIEN ENTRE PHP ET JAVASCRIPT

Consommer l'API OpenWeather avec Fetch et PHP

Côté client: Créer un fichier html avec Javascript intégré

```
<body>
    <h1>Météo Actuelle</h1>
    <input type="text" id="city" placeholder="Entrez une ville" />
    <button id="getWeather">Voir la météo</button>
    <div id="weather"></div>

    <script>
        document.getElementById('getWeather').addEventListener('click', () => {
            const city = document.getElementById('city').value || 'Nice';
            const url = `weather.php?city=${city}`; // Appel vers le script PHP

            fetch(url) // Requête HTTP vers PHP
                .then(response => {
                    if (!response.ok) {
                        throw new Error('Erreur réseau ou API');
                    }
                    return response.json(); // Convertir la réponse JSON
                })
                .then(data => {
                    // Afficher les informations météo
                    const weatherDiv = document.getElementById('weather');
                    weatherDiv.innerHTML =
                        `
                            <p><strong>Ville :</strong> ${data.name}</p>
                            <p><strong>Météo :</strong> ${data.weather[0].description}</p>
                            <p><strong>Température :</strong> ${data.main.temp}°C</p>
                            <p><strong>Humidité :</strong> ${data.main.humidity}%</p>
                        `;
                })
                .catch(error => {
                    console.error('Erreur :', error);
                    document.getElementById('weather').innerText = "Erreur lors de la récupération des données.";
                });
        });
    </script>
</body>
```

LIEN ENTRE PHP ET JAVASCRIPT

Explications

PHP :

- Récupère la ville envoyée en paramètre via **GET**.
- Fait la requête vers l'API OpenWeather.
- Retourne les données météo au format **JSON**.

JavaScript (Fetch) :

- Envoie une requête vers le fichier PHP avec la ville comme paramètre (**query string**).
- Récupère la réponse au format **JSON** et affiche les informations météo.

Communication :

- **PHP agit comme un intermédiaire** entre le client (JavaScript) et l'API externe (OpenWeather).
- **Fetch API** facilite l'envoi et la récupération des données sans recharger la page.



LIEN ENTRE PHP ET JAVASCRIPT

POSTMAN pour consommer un URL

On va utiliser ici l'**API Pokeapi**, qui regroupe l'intégralité des Pokemons, des lieux, des jeux, des baies... allant de la première à la dernière génération.

Pour vérifier si l'URL fonctionne, on peut utiliser l'application **Postman**.

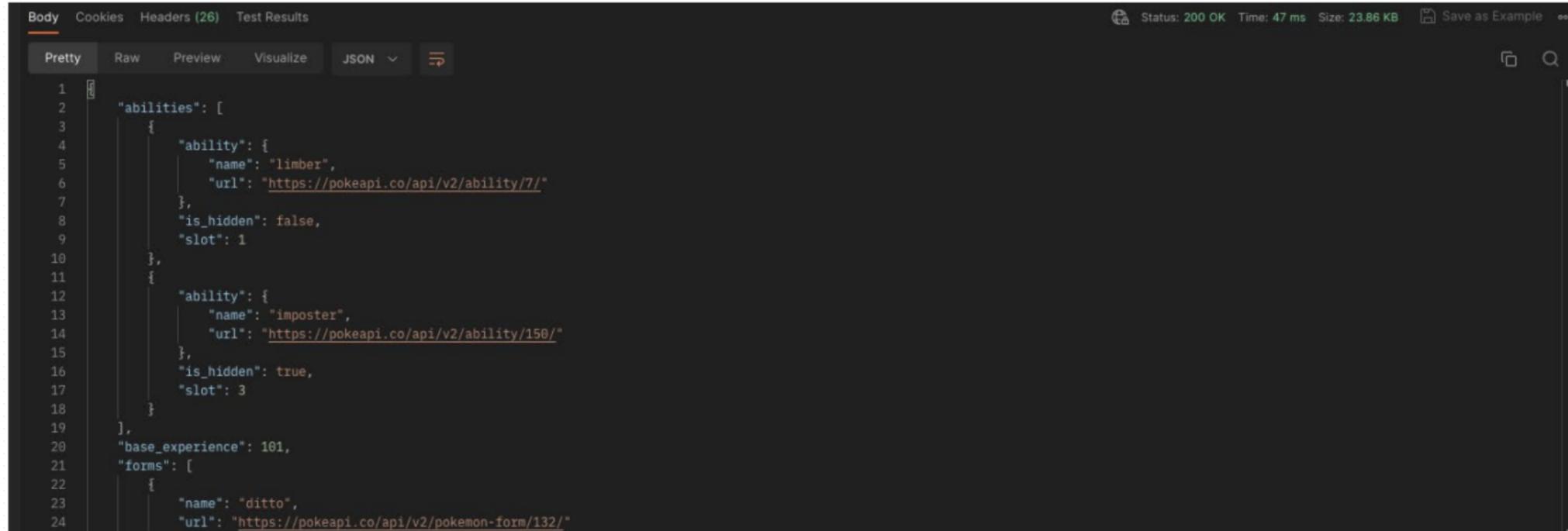
Ici on veut récupérer des informations, on va donc utiliser la méthode GET :

The screenshot shows the Postman application interface. At the top, there is a header bar with the URL `https://pokeapi.co/api/v2/pokemon/ditto`. Below the header, a dropdown menu shows "GET" is selected. To the right of the URL input field is a "Send" button. The main workspace has a sidebar on the left with methods: GET, POST, PUT, PATCH, DELETE, HEAD, and OPTIONS. The "GET" method is highlighted with a blue border. Below the sidebar, there are tabs for Headers (9), Body, Pre-request Script, Tests, and Settings. The Headers tab is active. The Body tab shows a green dot icon. The Pre-request Script, Tests, and Settings tabs are inactive. Below these tabs is a table with columns for Value and Description. The table has two rows, both of which are empty. At the bottom of the interface, there is a status bar showing "Status: 200 OK Time: 47 ms Size: 23.86 KB".

LIEN ENTRE PHP ET JAVASCRIPT

POSTMAN pour consommer un URL

Lorsque l'on appuie sur le bouton « Send », on envoie une requête.
La réponse s'affiche en bas de cette manière :



The screenshot shows the Postman interface with a successful API call. The top navigation bar includes 'Body', 'Cookies', 'Headers (26)', 'Test Results', and tabs for 'Pretty' (selected), 'Raw', 'Preview', 'Visualize', and 'JSON'. The status bar at the bottom right shows 'Status: 200 OK', 'Time: 47 ms', 'Size: 23.86 KB', and 'Save as Example'. The main content area displays a JSON response with line numbers from 1 to 24. The response object contains 'abilities', 'base_experience', and 'forms' arrays.

```
1 "abilities": [
2   {
3     "ability": {
4       "name": "limber",
5       "url": "https://pokeapi.co/api/v2/ability/7/"
6     },
7     "is_hidden": false,
8     "slot": 1
9   },
10  {
11    "ability": {
12      "name": "imposter",
13      "url": "https://pokeapi.co/api/v2/ability/150/"
14    },
15    "is_hidden": true,
16    "slot": 3
17  }
18 ],
19 "base_experience": 101,
20 "forms": [
21   {
22     "name": "ditto",
23     "url": "https://pokeapi.co/api/v2/pokemon-form/132/"
```

CONSOMMATION D'UNE API EXTERNE : GUZZLE HTTP

PARTIE 5

Le format JSON

LIEN ENTRE PHP ET JAVASCRIPT

L'objet JSON

Le JSON est un format simple et léger utilisé pour **échanger des données entre différents programmes, notamment entre PHP et Javascript.**

Il ressemble beaucoup à une liste d'informations organisées.

Imaginons que l'on souhaite partager facilement des informations d'utilisateur, on va l'écrire de cette façon :

C'est sous ce format que nous recevons les réponses d'API.

```
[  
  {  
    "name": "Nicolas",  
    "age": "23",  
    "city": "Nanterre"  
  },  
  {  
    "name": "Alexis",  
    "age": "27",  
    "city": "Montreuil"  
  }  
]
```

LIEN ENTRE PHP ET JAVASCRIPT

Mise en pratique

Maintenant que l'on sait appeler une API et que nous avons compris l'API fetch, appelons l'API pokeAPI. On va rajouter le `.then()` pour traiter la donnée.

```
fetch('https://pokeapi.co/api/v2/pokemon/1002')

  .then((response) => response.json())

  .then((data) => {
    console.log(data)
  })
```

Dans un premier temps, **on va transformer la réponse en format JSON** pour l'exploiter. Ensuite, pour vérifier, on va faire un `console.log()` de data.

LIEN ENTRE PHP ET JAVASCRIPT

Mise en pratique

Si vous voyez des informations en format JSON dans la console de votre navigateur, tout est bon !

Ici on retrouve donc plusieurs informations liées au Pokémons Ditto (Métamorph) : ses capacités, sa taille, son poids, des photos de lui...

```
index.html:16
▼ {abilities: Array(2), base_experience: 101, forms: Array(1), game_indices: Array(20), height: 3, ...} ⓘ
  ► abilities: (2) [{}]
    base_experience: 101
  ► forms: [{}]
  ► game_indices: (20) [{}]
    height: 3
  ► held_items: (2) [{}]
    id: 132
    is_default: true
    location_area_encounters: "https://pokeapi.co/api/v2/pokemon/132/encounters"
  ► moves: [{}]
    name: "ditto"
    order: 214
  ► past_types: []
  ► species: {name: 'ditto', url: 'https://pokeapi.co/api/v2/pokemon-species/132'}
  ► sprites: {back_default: 'https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/132/back.png', back_shiny: 'https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/132/back_shiny.png', front_default: 'https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/132/normal.png', front_shiny: 'https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/132/shiny.png', front_thin: 'https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/132/thin.png'}
  ► stats: (6) [{}]
    weight: 40
  ► types: [{}]
```

LIEN ENTRE PHP ET JAVASCRIPT

Mise en pratique

On veut maintenant afficher les informations de notre pokemon directement sur notre page.

Pour ça, on va procéder d'une manière un peu différente.
On va séparer la requête à l'API et l'affichage dans 2 fonctions différentes.

- Une fonction **fetchPokemon()** qui va return la réponse de la requête API.
- Une fonction **displayPokemon()** qui va afficher sur notre page.

```
function fetchPokemon(pokemon) {  
  return fetch('https://pokeapi.co/api/v2/pokemon/' + pokemon)  
    .then((response) => response.json())  
}
```

```
function displayPokemon(pokemon) {  
  const data = fetchPokemon(pokemon)  
  document.getElementById("pokemon").innerHTML = `  
    <h1>${data.name}</h1>  
      
  `;  
}
```

CONSOMMATION D'UNE API EXTERNE : GUZZLE HTTP

PARTIE 6

Synchrone/Asynchrone

LIEN ENTRE PHP ET JAVASCRIPT

Synchrone vs Asynchrone

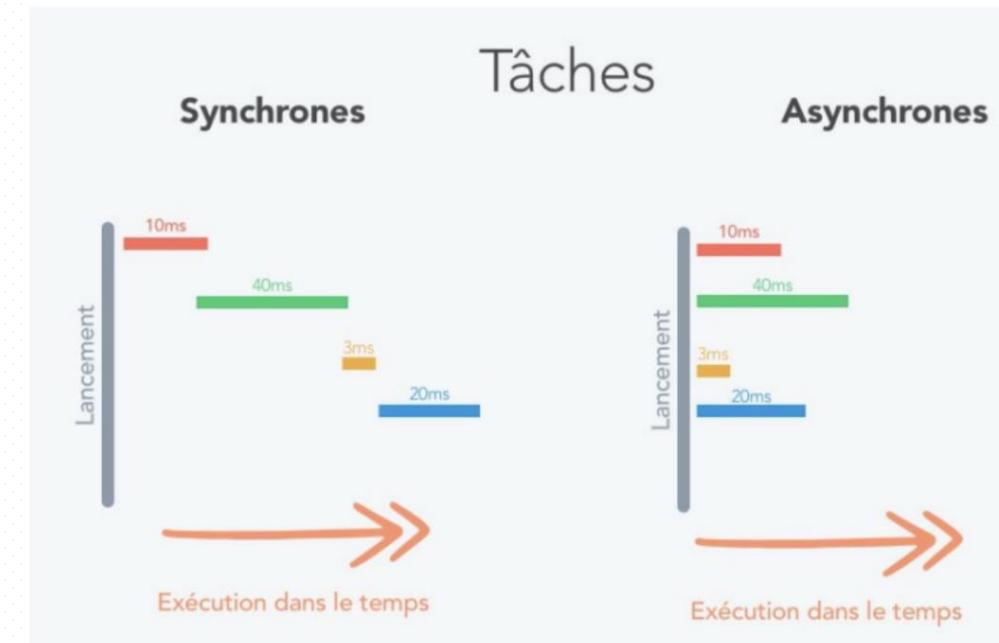
Le javascript fonctionne de manière dite « synchrone ».

Cela signifie que **le code est exécuté de manière séquentielle**, avec une instruction à la fois en suivant l'ordre d'écriture.

Chaque instruction doit se terminer avant que la suivante puisse commencer.

Mais il est possible de faire de l'**asynchrone**.

Cela permet aux instructions lentes ou bloquantes de s'exécuter en arrière -plan tout en permettant au reste du code de s'exécuter.



LIEN ENTRE PHP ET JAVASCRIPT

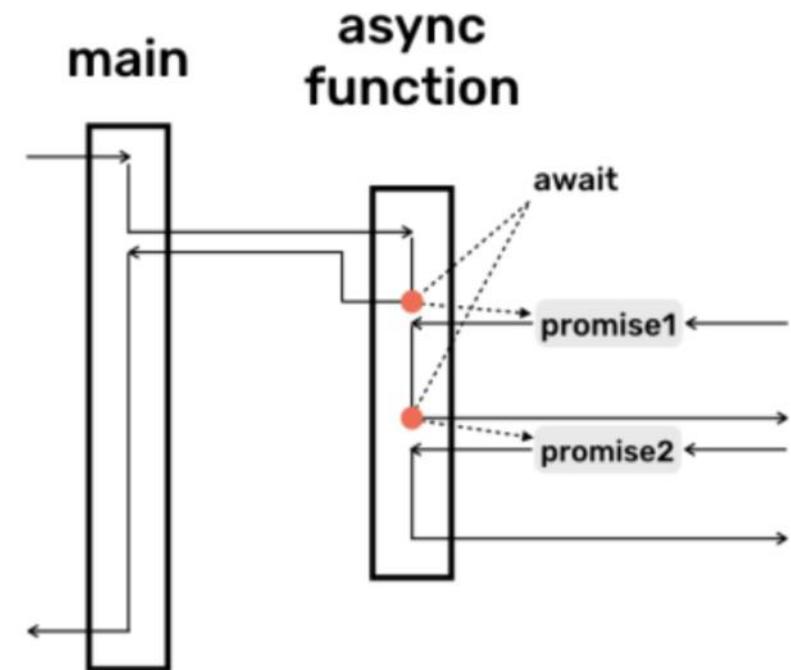
Synchrone vs Asynchrone

D'accord mais pourquoi nous avons besoin de ça dans notre cas ?

Et bien puisque la fonction **fetch()** est une fonction asynchrone, ce qui veut dire qu'elle n'attendra pas la réponse pour passer à la suite.

On utilise donc **async / await** qui sont des directives qui spécifient le fonctionnement des instructions :

- **async** force la fonction à retourner une promesse*.
- **await** capture une promesse en cours de traitement et bloque l'exécution en attendant la résolution de cette dernière.



LIEN ENTRE PHP ET JAVASCRIPT

Synchrone vs Asynchrone

La fonction **displayPokemon()** est passée en **asynchrone**.

Ensuite on demande à la fonction **fetchPokemon()** d'attendre la réponse avant de l'afficher.

```
async function displayPokemon(pokemon) {
    const data = await fetchPokemon(pokemon)
    document.getElementById("pokemon").innerHTML =
        <h1>${data.name}</h1>
        
    }
}
```

CONSOMMATION D'UNE API EXTERNE : GUZZLE HTTP

PARTIE 7

Exercice pratique

EXERCICE PRATIQUE

• Exercice – Créez une petite application PHP qui consomme une API

Préparer l'environnement :

- Créez un nouveau projet PHP avec un fichier principal.
- Installez **Composer** et ajoutez la bibliothèque **GuzzleHTTP**.

Créer un script PHP :

- Envoyez une requête HTTP vers une **API publique** (au choix, par exemple, OpenWeather ou les API proposées par votre projet d'AXE).
- Récupérez les données au format **JSON**.

Traiter et afficher les données :

- Décodez la réponse JSON.
- Affichez des informations spécifiques dans un format lisible (par exemple, température, description météo).

Bonus

- Permettez à l'utilisateur de **saisir une ville** pour récupérer la météo en temps réel.
- Gérez les erreurs de connexion ou de réponse.



CONSOMMATION D'UNE API EXTERNE : GUZZLE HTTP

PARTIE 8

RENDU FINAL

INITIATION BACKEND

Rappel des Rendus

■ Exercice pratique - créer une fonction utilisateur (Diapo 54) : 4 points

- Crée une fonction « calculateMoyenne » qui retourne la moyenne de trois nombres, puis une fonction « afficherResultat » qui prend un nom et une moyenne, affichant si elle est suffisante (≥ 10) ou insuffisante (< 10)

■ Exercice – Identifier les rôles dans une application web (Diapo 29) : 2 points

- Analysez une application web en identifiant les éléments visibles par l'utilisateur (frontend) et les actions en arrière-plan (backend), puis associez les technologies utilisées à chaque partie.

■ Créer une petite application PHP qui consomme une API (Diapo 114) : 4 points

- Créez un projet PHP avec Composer et GuzzleHTTP, envoyez une requête à une API publique (ex: OpenWeather ou l'API de votre choix), récupérez et affichez des données JSON (ex: température, météo), et en bonus, permettez à l'utilisateur de choisir une ville et gérez les erreurs.

INITIATION BACKEND

Rappel des Rendus

■ Rendu final : 10 points

- Implémenter et appeler une API sur votre projet d'Axe (celle de votre choix) pour afficher les données de votre choix, en passant par un appel asynchrone avec Javascript.

Attention aux MALUS :



- Code non ou trop peu commenté
- Nomenclatures fantaisistes des fichiers, variables, fonctions
- Dossier de rendu mal rangé
- Fonctionnalités ou fichiers demandés mal indiqués faisant perdre du temps lors des corrections
- GitHub en privé ou lien erroné

INITIATION BACKEND

Comment me rendre votre travail?

■ Consignes

- Déposer tous vos rendus dans **un seul et unique repository Github** en public et mettre le lien sur votre **Moodle** mais m'envoyer également ce **lien par email** que je vous transmettrai en classe.

Attention :

- Votre lien Github devra prendre la forme suivante : NOM_PRENOM_GROUPE_MATIERE

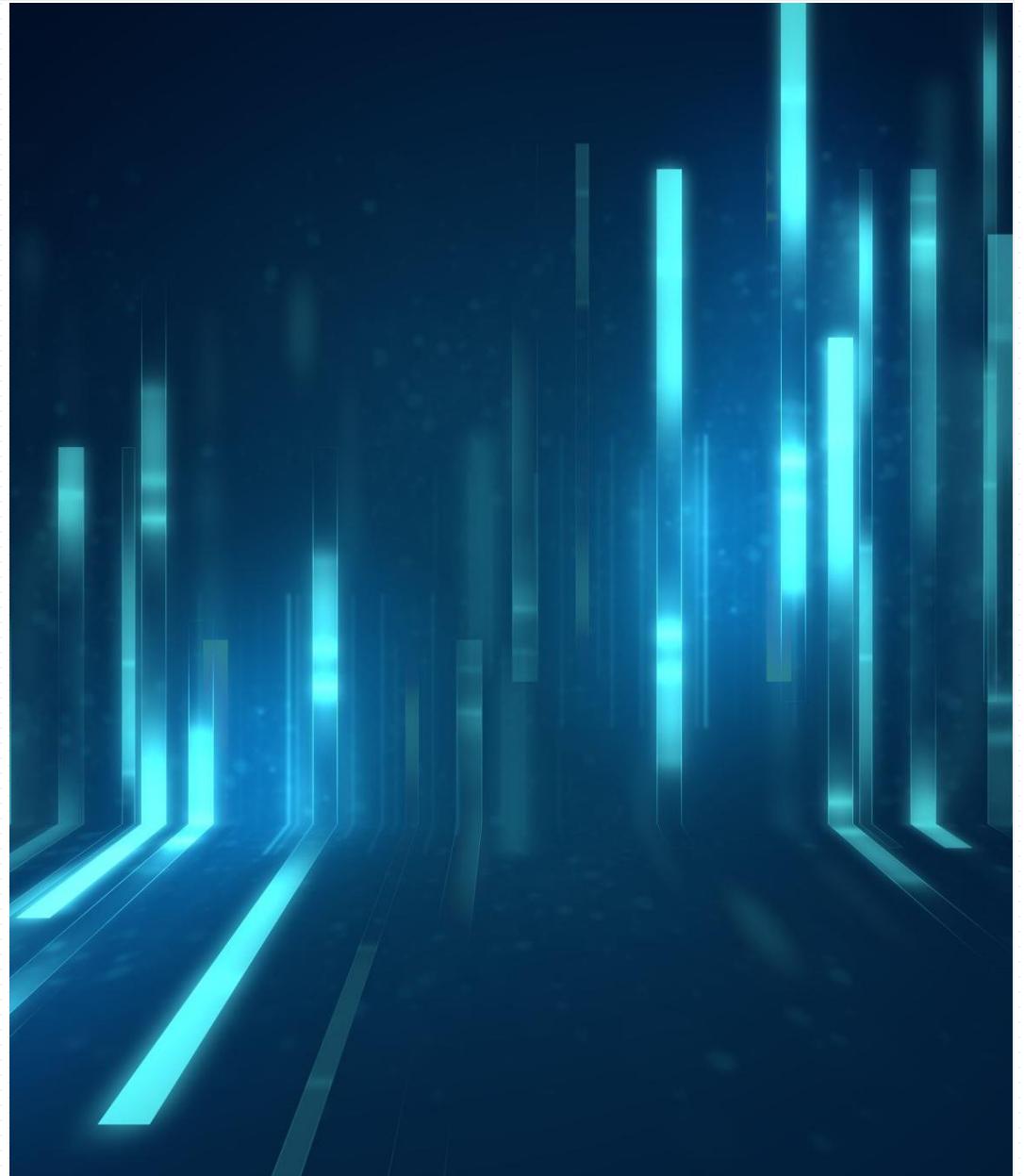
SOURCES

Liste des sources

-  **Guzzle HTTP**
<https://docs.guzzlephp.org/en/stable/>

-  **PHP – documentation officielle**
<https://www.php.net/>

-  **API fetch**
https://developer.mozilla.org/fr/docs/Web/API/Fetch_API



SOURCES

Liste des sources



Composer

<https://getcomposer.org/>



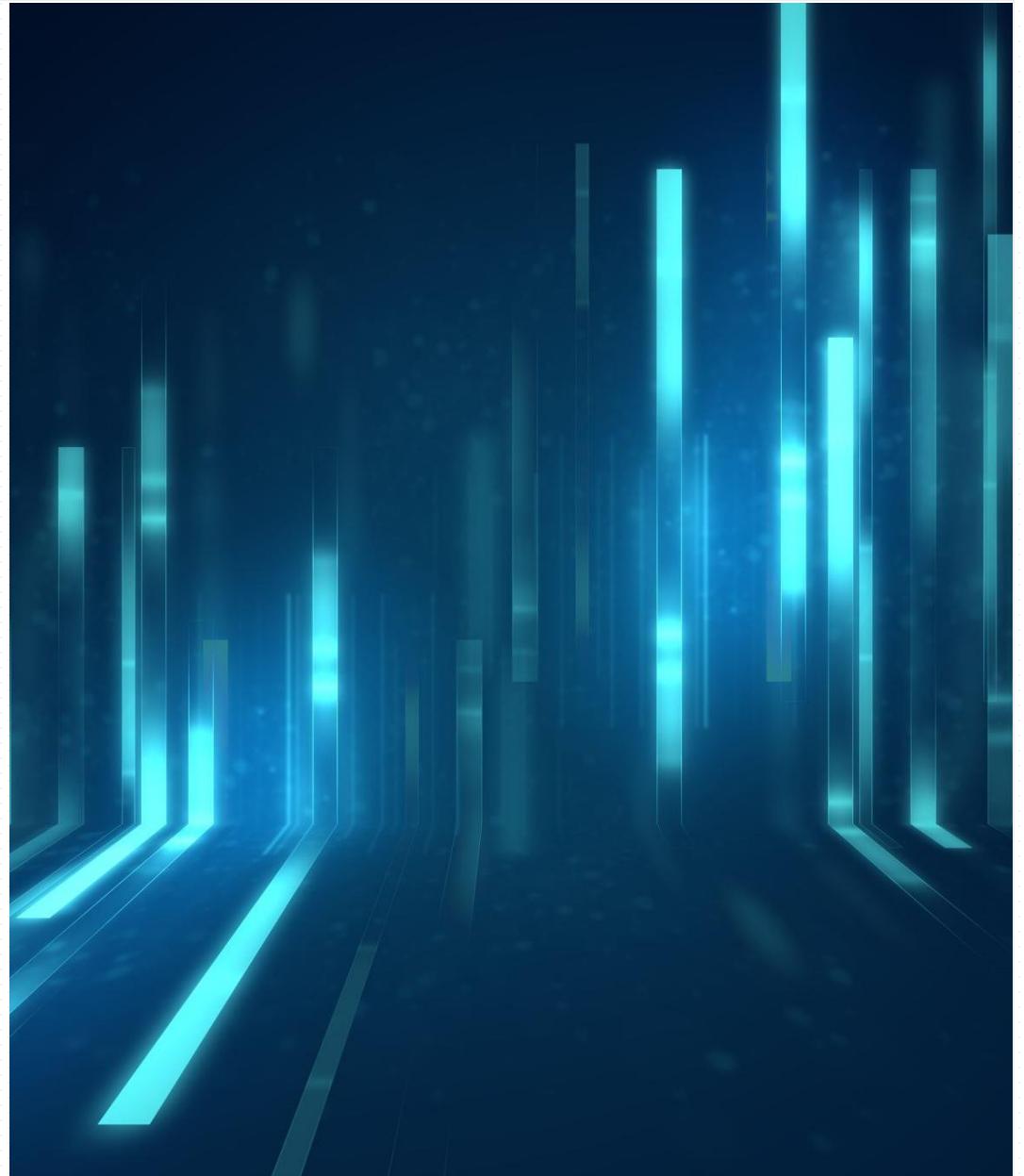
OpenWeatherMap

<https://openweathermap.org/>



JSON

https://developer.mozilla.org/fr/docs/Learn_web_development/Core/Scripting/JSON



SOURCES

Liste des sources



PokeAPI

<https://pokeapi.co/>



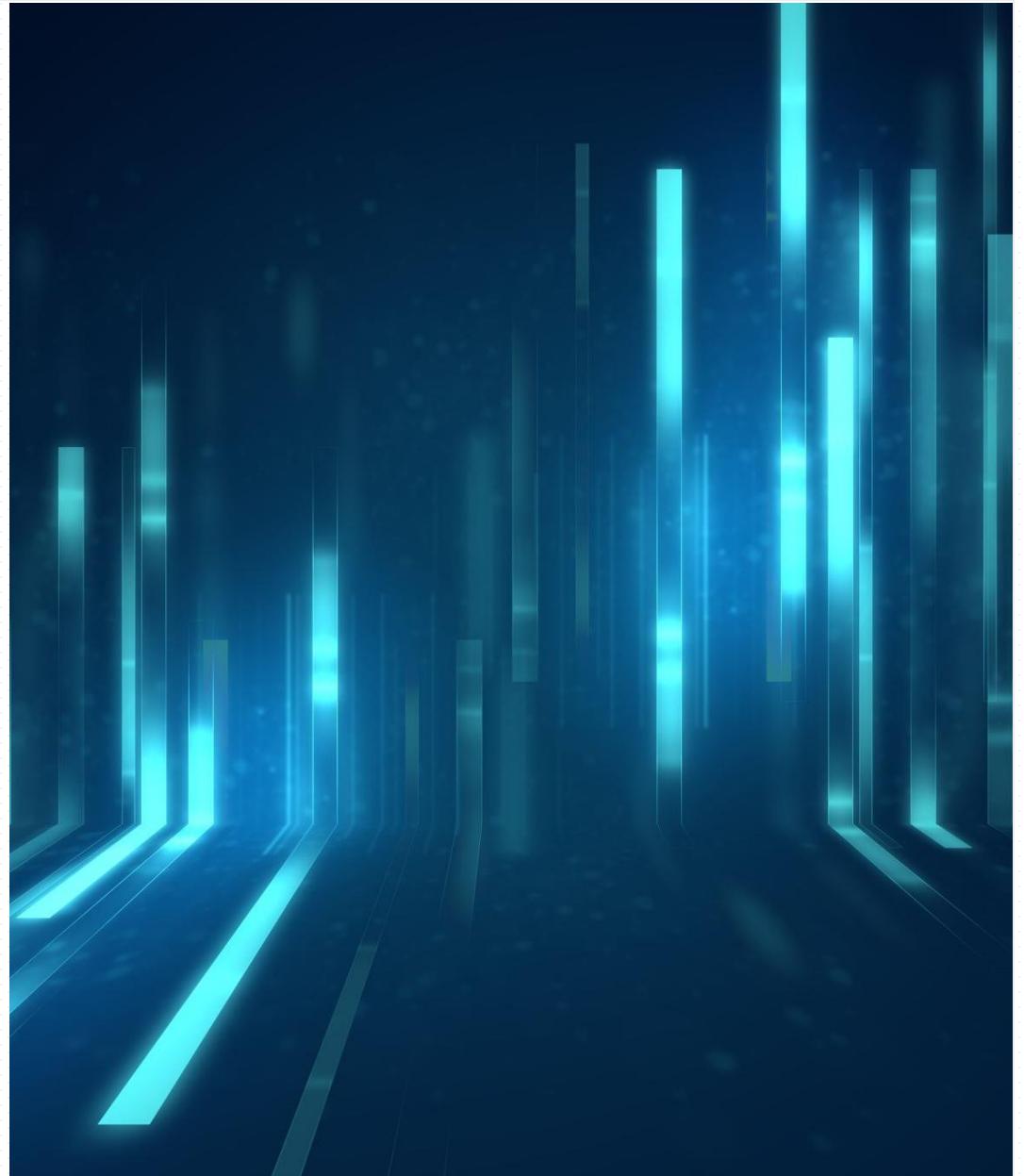
OpenWeatherMap

<https://www.themoviedb.org/?language=fr>



JSON placeholder

<https://jsonplaceholder.typicode.com/>



DES QUESTIONS ?

Me contacter



samih@academiews.fr



07 82 14 81 41



MERCI POUR VÔTRE ATTENTION

Bonne révision !

