Développement WEB PHP - Séance 2

BUT Informatique parcours DACS

Table of contents:

- 📜 | Architecture d'une application web
  - Le rôle du client
  - Front-End : Côté client
  - Back-End : Côté serveur
  - Base de données : Côté données
  - Cycle de traitement complet
  - Stack & Full-stack
- 📜 | Anatomie du protocole HTTP
  - Les composantes d'une requête HTTP
    - Ligne de requête (Request Line)
    - En-têtes de requête (Headers)
    - Corps de la requête (Body)
  - Le cycle d'une requête HTTP
  - Exemple complet d'une requête HTTP
    - Requête HTTP
    - Réponse HTTP du serveur
  - Les méthodes HTTP les plus courantes
  - En-têtes importants dans une requête HTTP
  - Exemple complet
  - Codes HTTP
  - Sécurité avec HTTPS
  - Pour aller plus loin
- 📜 | Le JSON
    - Pourquoi utiliser JSON ?
    - Structure du JSON
      - Objets
      - Tableaux
    - Types supportés
    - Exemples d'utilisation
    - ⚙️ | Encoder et décoder du JSON en PHP
      - json_encode
      - json_decode
- ⚙️ | Variables prédéfinies
    - Les superglobales
    - Les autres variables prédéfinies
- 📚 | TD 3 : Formulaires et variables prédéfinies

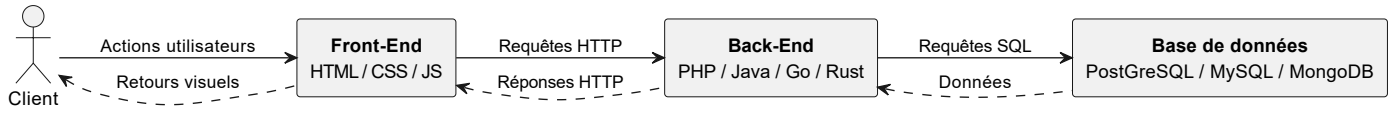
- Cloner le projet
- Faire les exercices

📜 | Architecture d'une application web

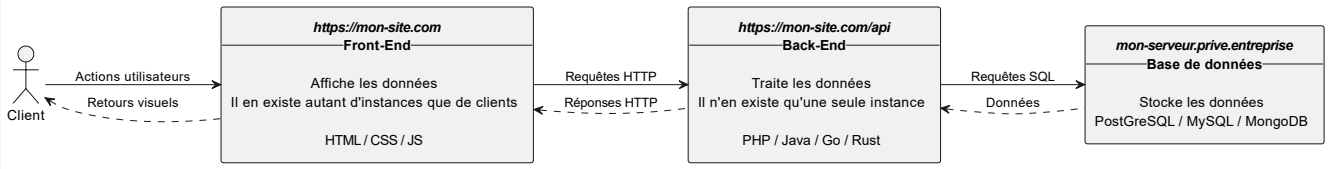
> **INFO**
>
> Cette partie n'est pas à apprendre en détails. Cependant, **la compréhension des différents termes évoqués est nécessaire**.

Dans ce cours, nous allons explorer l'architecture typique d'une application web en distinguant les rôles du **client**, du **front-end**, du **back-end**, et de la **base de données**. Chaque composant joue un rôle crucial dans le traitement des requêtes et des réponses qui permettent aux utilisateurs d'interagir avec une application web.

Voici un diagramme pour faciliter la compréhension tout au long de l'explication :



Version plus détaillée



EN ANALOGIE 🥖

Imaginons une boulangerie pour illustrer le fonctionnement d'une application web.

- Le **client** est simplement la personne qui entre dans la boulangerie.
- La **vendeuse** représente le **front-end**.
- Le **boulanger** en cuisine, correspond au **back-end**.
- Le **stock de ressources**, représente la **base de données**.

La vendeuse prend la commande du client (comme le front-end qui capture les actions utilisateurs) et, si nécessaire, envoie une demande au boulanger. Le boulanger, lui, prépare le pain ou vérifie dans le stock de ressources, pour voir s'il a les ingrédients ou les produits nécessaires. Une fois la demande traitée, le boulanger transmet le pain à la vendeuse, qui le remet au client, tout comme le front-end affiche les résultats après que le back-end a traité les données. À aucun moment le client n'a intéragit avec le boulanger ou le sock de ressources.

Le rôle du client

Le **client** représente l'utilisateur qui interagit avec une application web à l'aide d'un navigateur. Il envoie des actions, généralement sous forme de clics, soumissions de formulaires, ou autres interactions avec l'interface visuelle. Le client ne fait qu'exécuter des actions, sans avoir directement accès au traitement des données ou à la logique de l'application. Tout ce qu'il voit se limite à l'interface utilisateur visible dans le navigateur.

Front-End : Côté client

Le **front-end** désigne la partie visible de l'application avec laquelle le client interagit directement. Il est composé principalement de trois technologies :

- **HTML** : structure les pages web en définissant les éléments visibles (textes, images, boutons, etc.).
- **CSS** : stylise ces éléments pour améliorer la présentation (couleurs, polices, marges, etc.).
- **JavaScript (JS)** : ajoute de l'interactivité dynamique, par exemple, la mise à jour de contenu sans recharger la page entière.

Le front-end prend en charge les **actions de l'utilisateur**, comme un clic sur un bouton ou la soumission d'un formulaire, puis envoie ces actions au back-end sous forme de **requêtes HTTP**. Il est responsable de l'affichage des résultats et des retours visuels une fois que le back-end a traité la demande.

> **INFO**
>
> Le front-end est **obligatoire**. Sans HTML, il n'existe pas de site (et donc pas de WEB).

> **ATTENTION**
>
> Comme cité dans la séance 1, le PHP s'exécute côté serveur. Même si le PHP est utilisé pour faire du front-end, le rendu de la page se fera côté serveur, et ensuite le client réceptionne le HTML/CSS/JS créé au préalable. Cette notion est appelée **Server Side Rendering** (ou **SSR**).
>
> > **DANGER**
> >
> > **Le client ne peut pas exécuter de PHP**, il ne reçoit toujours que du HTML/CSS/JS.

Back-End : Côté serveur

Le **back-end** est la partie cachée de l'application qui s'occupe du traitement des requêtes. C'est ici que la logique métier et les règles de gestion de l'application résident. Différents langages peuvent être utilisés pour coder la logique serveur, parmi lesquels :

- **PHP** : souvent utilisé pour gérer des sites dynamiques et interagir avec des bases de données.
- **Java** : utilisé pour les applications robustes et à grande échelle.
- **Go** ou **Rust** : de plus en plus utilisés pour des raisons de performance et de sécurité.

Le back-end reçoit les **requêtes HTTP** provenant du front-end, effectue les calculs nécessaires ou exécute les règles de gestion, et peut, si nécessaire, communiquer avec une base de données pour récupérer ou stocker des informations.

> **INFO**
>
> Le back-end est **optionnel**.

Base de données : Côté données

La **base de données** est l'endroit où les informations sont stockées. Elle permet de sauvegarder des données de manière structurée, facilitant ainsi leur gestion et récupération par le back-end. Les bases de données peuvent être relationnelles ou non relationnelles, et parmi les plus courantes, nous avons :

- **PostgreSQL** : une base de données relationnelle très puissante.
- **MySQL** : une autre base de données relationnelle, très populaire pour des projets de tailles variées.
- **MongoDB** : une base de données non relationnelle (NoSQL), idéale pour stocker des documents JSON et des données semi-structurées.

Le back-end envoie des **requêtes SQL** à la base de données pour récupérer ou modifier les données en fonction des besoins de l'application. Une fois les données récupérées ou modifiées, elles sont renvoyées au back-end, qui les traite et les formate avant de les transmettre au front-end.

> **INFO**
>
> La base de données est **optionnelle**.

Cycle de traitement complet

Pour récapituler, le cycle de traitement d'une application web se déroule comme suit :

1. Le **client** effectue une action via l'interface utilisateur.
2. Le **front-end** capture cette action et envoie une **requête HTTP** au **back-end**.
3. Le **back-end** traite la requête et, si nécessaire, envoie une **requête SQL** à la **base de données** pour accéder ou modifier des informations.
4. La **base de données** renvoie les données au **back-end**, qui les formate en **réponse HTTP**.
5. Le **back-end** renvoie cette réponse au **front-end**, qui met à jour l'interface et affiche les informations pertinentes au **client**.

Ce modèle d'interaction est typique de la majorité des applications web modernes et repose sur une division claire des rôles entre les différentes couches pour une meilleure organisation et maintenabilité du code.

Stack & Full-stack

Le terme **stack WEB** correspond à l'ensemble du front-end, back-end et de la base de données (ou même l'administration de ces systèmes). Un développeur **full-stack** est développeur qui maître et utilise l'ensemble de ces aspects.

📜 | Anatomie du protocole HTTP

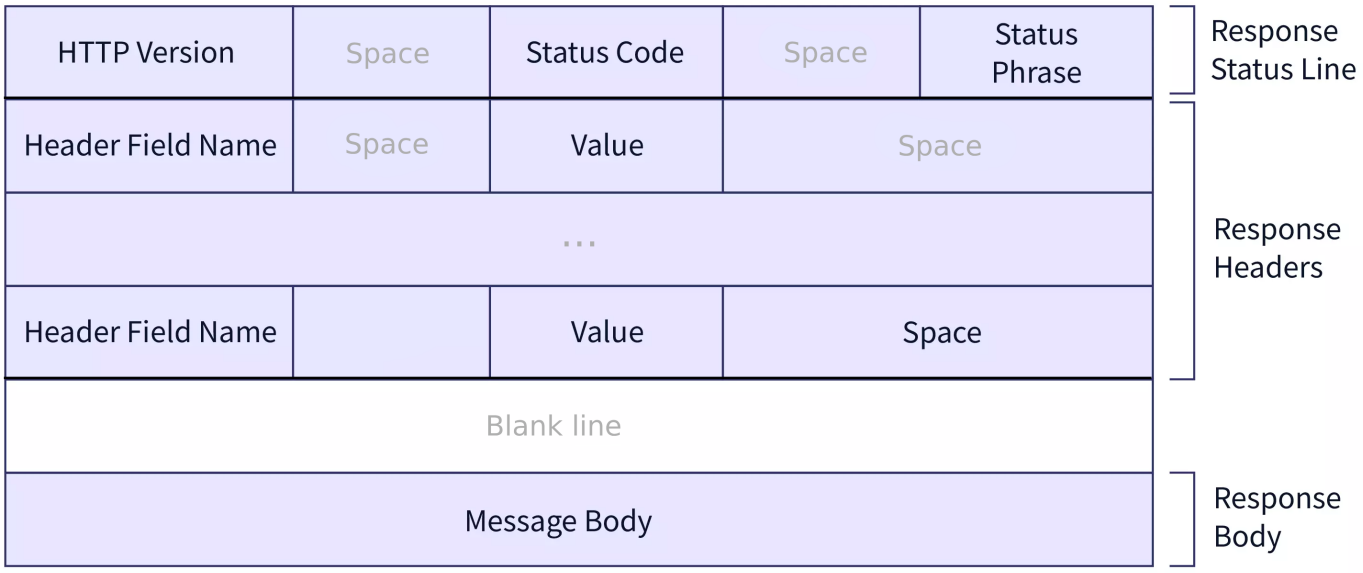
> **INFO**
>
> Cette partie n'est pas à apprendre par cœur, mais elle constitue un élément principal de la compréhension du monde du WEB. **La compréhension des différents termes évoqués est nécessaire**.

Dans le cadre du développement web, le protocole **HTTP** (HyperText Transfer Protocol) joue un rôle central en permettant la communication entre les navigateurs web (clients) et les serveurs. Lorsqu'un utilisateur accède à une page web ou interagit avec une application en ligne, une série de requêtes et réponses HTTP sont échangées entre le client et le serveur. Comprendre l'anatomie d'une requête HTTP est essentiel pour diagnostiquer des problèmes, optimiser des applications web, et développer des services API efficaces.

Les composantes d'une requête HTTP

Une requête HTTP se divise en plusieurs parties distinctes, chacune ayant un rôle spécifique.

Voici un diagramme pour faciliter la compréhension tout au long de l'explication :



Ligne de requête (Request Line)

La ligne de requête est la première ligne de la requête HTTP. Elle contient trois informations clés :

- **Méthode HTTP** : elle spécifie l'action à réaliser sur la ressource demandée. Les méthodes les plus courantes sont :
  - `GET` : pour récupérer une ressource (lecture).
  - `POST` : pour envoyer des données au serveur (souvent pour créer ou mettre à jour une ressource).
  - `PUT` : pour remplacer ou mettre à jour une ressource existante.
  - `DELETE` : pour supprimer une ressource.
  - `PATCH` : pour mettre à jour partiellement une ressource.
- **URI (Uniform Resource Identifier)** : c'est l'adresse de la ressource demandée, souvent appelée chemin ou endpoint. Par exemple, `/articles/42` fait référence à l'article numéro 42.

- **Version du protocole HTTP** : généralement `HTTP/1.1` ou `HTTP/2`, cette information précise quelle version du protocole est utilisée.

Exemple :

```
GET /articles/42 HTTP/1.1
```

En-têtes de requête (Headers)

Les en-têtes de requête fournissent des informations supplémentaires sur la requête, telles que le type de contenu envoyé, les autorisations nécessaires, ou encore des informations sur le client. Chaque en-tête est une paire clé-valeur.

Les en-têtes courants incluent :

- **Host** : l'adresse du serveur (ex : `www.example.com`).
- **User-Agent** : informations sur le client, comme le type de navigateur utilisé.
- **Accept** : spécifie le type de contenu que le client peut accepter (ex : `text/html`, `application/json`).
- **Content-Type** : indique le type de données envoyées dans le corps de la requête (utile pour `POST` ou `PUT`, ex : `application/json`).
- **Authorization** : contient les informations d'authentification.

Exemple :

```
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: application/json
```

Corps de la requête (Body)

Le corps de la requête est une partie optionnelle qui contient les données envoyées au serveur, par exemple lors d'une requête `POST` ou `PUT`. Ces données peuvent être de divers formats (JSON, XML, formulaire encodé, etc). Pour une requête `GET`, le corps de la requête est généralement absent car les données sont passées dans l'URI (via des paramètres de requête).

Exemple (dans une requête `POST`) :

```
{
  "title": "Nouvel Article",
  "content": "Contenu de l'article..."
}
```

Le cycle d'une requête HTTP

Lorsque le client (le navigateur ou une application) envoie une requête HTTP, voici les étapes principales qui se déroulent :

1. **L'utilisateur interagit avec l'interface** (par exemple en cliquant sur un lien ou un bouton de soumission de formulaire).
2. **Le navigateur envoie une requête HTTP** au serveur spécifié, contenant la ligne de requête, les en-têtes et, dans certains cas, un corps de requête.
3. **Le serveur reçoit la requête**, l'interprète, et effectue les actions nécessaires (récupérer des données depuis une base, effectuer des calculs, etc.).
4. **Le serveur envoie une réponse HTTP** au client, contenant les données demandées ou un message d’erreur si la requête ne peut être traitée.
5. **Le navigateur affiche le résultat** à l'utilisateur sous forme de page web, d'alerte, ou toute autre représentation.

Exemple complet d'une requête HTTP

Prenons un exemple où un utilisateur cherche à récupérer un article depuis un serveur via une requête `GET`.

Requête HTTP

```
GET /articles/42 HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: application/json
```

Ici, le client demande l'article numéro 42 au serveur `www.example.com`. Le client indique qu'il souhaite recevoir les données au format JSON grâce à l'en-tête `Accept`.

Réponse HTTP du serveur

En retour, le serveur pourrait répondre avec :

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 126

{
  "id": 42,
  "title": "Nouvel Article",
  "content": "Voici le contenu de l'article."
}
```

Dans cet exemple :

- Le code de statut `200 OK` signifie que la requête a réussi.
- Le type de contenu est spécifié comme `application/json`, ce qui correspond à la demande du client.
- Le corps de la réponse contient les détails de l'article en format JSON.

Les méthodes HTTP les plus courantes

Les méthodes HTTP, également appelées verbes HTTP, spécifient le type d'action à effectuer sur une ressource donnée. Voici un aperçu des méthodes les plus utilisées :

- **GET** : Utilisé pour récupérer des informations d'une ressource. Une requête `GET` ne devrait pas modifier l'état de la ressource.
- **POST** : Utilisé pour envoyer des données au serveur, par exemple pour créer une nouvelle ressource. Il est souvent utilisé pour soumettre des formulaires ou des données JSON.
- **PUT** : Utilisé pour remplacer entièrement une ressource existante avec les données fournies. Il s'agit d'une opération idempotente, ce qui signifie que l'application répétée de la requête ne modifie pas davantage la ressource.
- **PATCH** : Semblable à `PUT`, mais il est utilisé pour mettre à jour partiellement une ressource.
- **DELETE** : Utilisé pour supprimer une ressource sur le serveur.

En-têtes importants dans une requête HTTP

Certains en-têtes jouent un rôle important dans le traitement des requêtes HTTP :

- **Cache-Control** : Gère le comportement de mise en cache des ressources. Par exemple, il peut indiquer au navigateur s'il doit stocker une copie de la ressource et pendant combien de temps.
- **Authorization** : Utilisé pour transmettre des informations d'authentification, comme un jeton d'accès dans les services sécurisés.
- **Cookie** : Utilisé pour stocker et envoyer des informations liées à une session utilisateur ou à des préférences sur le site web.

Exemple complet

Voici un exemple plus complexe pour savoir à quoi peut ressembler une requête HTTP dans un cas réel :

```
POST /api/v1/users/create?notify=true&admin=false HTTP/1.1
Host: api.example.com
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.e30.m5uXnxMlfz_mxCzMzRZZHY
Content-Type: application/json
Accept: application/json
User-Agent: Firefox/1.0.0
Cache-Control: no-cache
Pragma: no-cache
Content-Length: 245

{
  "firstName": "John",
  "lastName": "Doe",
  "email": "john.doe@example.com",
  "password": "SuperSecretPassword",
  "roles": ["user", "editor"],
}
```

- **Méthode HTTP** : `POST` est utilisé ici pour créer une nouvelle ressource (un utilisateur).
- **Chemin de l'API** : `/api/v1/users/create` indique l'URL du endpoint.
  - **Paramètres de requête** : `?notify=true&admin=false` précise que l'utilisateur doit être notifié et qu'il ne sera pas créé en tant qu'administrateur.
- **En-têtes HTTP** (Headers) :
  - `Authorization: Bearer` est utilisé pour envoyer un token JWT (JSON Web Token) pour l'authentification.
  - `Content-Type: application/json` indique que le corps de la requête est au format JSON.
  - `Accept: application/json` signifie que la réponse attendue doit également être au format JSON.
  - `User-Agent` spécifie l'application qui envoie la requête.
  - `Cache-Control` et `Pragma` désactivent la mise en cache.
- **Corps de la requête** (Body) : Un JSON qui contient les informations de l'utilisateur à créer.

Cette requête va ensuite être traitée par le back-end qui disposera alors de toutes ces informations.

Codes HTTP

La réponse d'une requête arrive avec un code de retour allant de 100 à 527. Le premier chiffre est utilisé pour spécifier une des cinq catégories de réponse :

- 1XX : information
- 2XX : succès
- 3XX : redirection
- 4XX : erreur client
- 5XX : erreur serveur

Les codes les plus courants sont :

- 200 : succès de la requête
- 201 : succès avec création de ressource
- 301 et 302 : redirection, respectivement permanente et temporaire
- 401 : utilisateur non authentifié
- 403 : accès refusé
- 404 : ressource non trouvée
- 500, 502 et 503 : erreurs serveur

Voir la liste complète des codes HTTP

Sécurité avec HTTPS

HTTPS (Hypertext Transfer Protocol Secure) est la combinaison du HTTP avec une couche de chiffrement **TLS**. HTTPS permet au visiteur de vérifier l'identité du site web auquel il accède, grâce à un certificat d'authentification émis par une autorité tierce, réputée fiable.

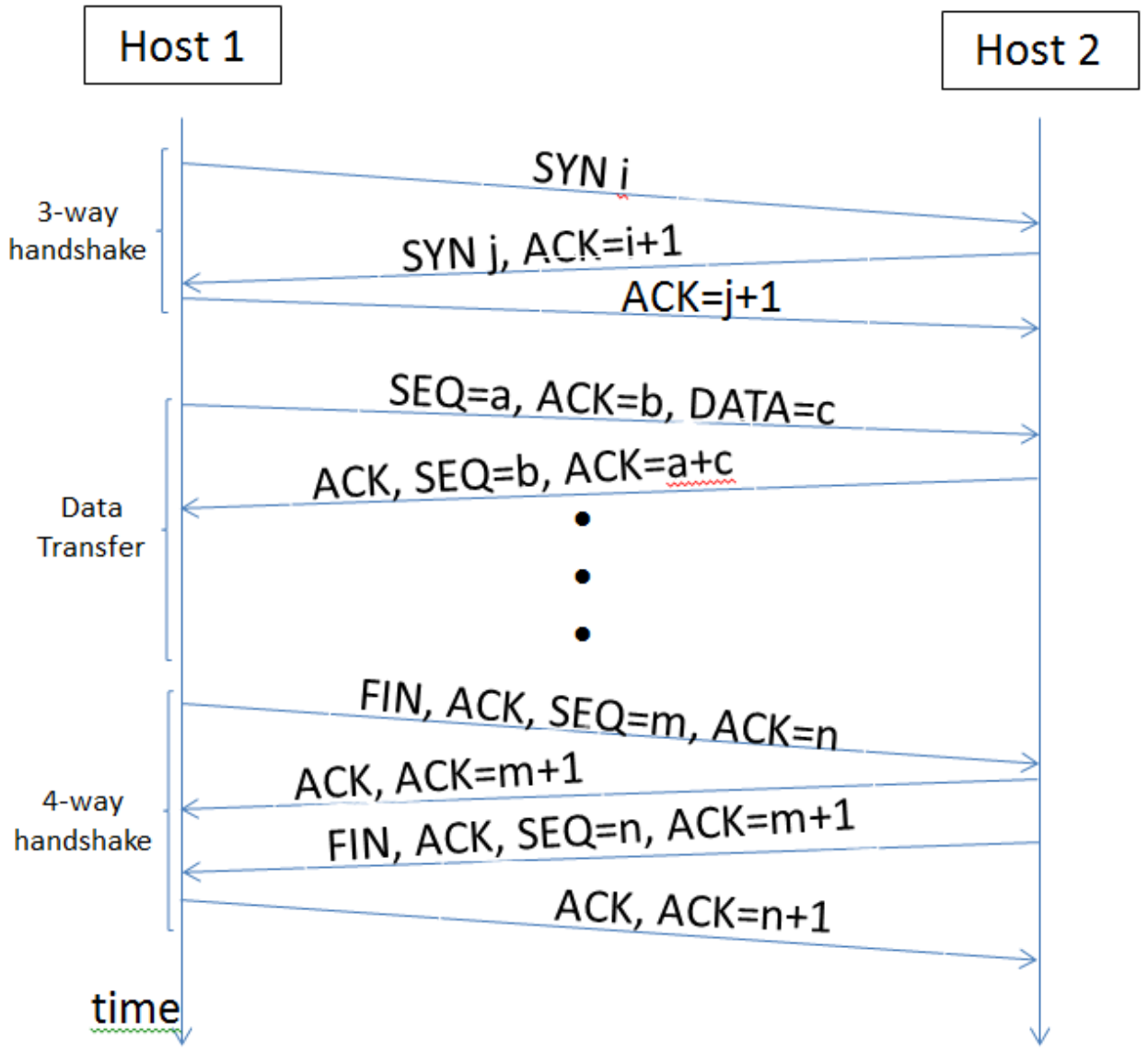
La TLS (ou SSL) fonctionne suivant un mode client-serveur. Il permet de satisfaire les objectifs de sécurité suivants :

- L'authentification du serveur
- La confidentialité des données échangées (ou session chiffrée)
- L'intégrité des données échangées
- De manière optionnelle, l'authentification du client (mais dans la réalité celle-ci est souvent assurée par la couche applicative)

Pour aller plus loin

Le protocole HTTP est un membre de la famille **TCP/IP**. TCP/IP est une famille de protocoles de communication utilisés pour connecter des systèmes informatiques dans un réseau. Il est nommé d'après deux des protocoles de la famille: Transmission Control Protocol (TCP) et Internet Protocol (IP ; oui, comme les IPv4 et IPv6).

Bien que TCP/IP et HTTP ne soient pas parfaitement identiques, le diagramme de séquence suivant permet de mieux comprendre ce qu'il se passe dernière une seule requête HTTP.



📜 | Le JSON

INFO

Cette partie n'est pas à apprendre par cœur, mais elle constitue un élément principal de la compréhension du monde du WEB. **La compréhension des différents termes évoqués est nécessaire**.

Le **JSON** (JavaScript Object Notation) est un format léger de données largement utilisé pour échanger des informations entre les systèmes, en particulier dans les applications web. Il est à la fois facile à lire pour les humains et simple à interpréter pour les machines. Le JSON est directement dérivé de la notation des objets du langage JavaScript. Ainsi le JSON est directement utilisable dans du code JavaScript et inversement. Bien qu'il soit dérivé de la syntaxe de JavaScript, il peut être utilisé avec de nombreux langages de programmation comme Python, Java, PHP, ou encore Go.

Pourquoi utiliser JSON ?

Lisibilité

Le JSON est encodé en UTF-8 (c'est-à-dire pas en bytes comme 0xf6 0x3a). Sa syntaxe claire le rend facile à lire pour les humains.

Simplicité

Moins verbeux que le XML, il est plus léger en termes de taille.

LE XML POUR LES CURIEUX

```
<?xml version="1.0" encoding="UTF-8"?>

<shiporder orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>
</shiporder>
```

Compatibilité

JSON est pris en charge nativement par la plupart des langages de programmation modernes, facilitant ainsi son adoption dans divers contextes.

Échanges avec les API

JSON est devenu le standard pour échanger des données dans les API web modernes.

Structure du JSON

Le JSON se compose de deux structures principales, les objets et les tableaux.

Objets

Les objets sont des collections de paires clé-valeur, entourées d'accolades `{}`. Chaque clé est une chaîne de caractères (texte) et est suivie d'une valeur (qui peut être de différents types).

Exemple d'un objet JSON :

```
{
  "nom": "Dupont",
  "age": 30,
  "estMembre": true
}
```

Tableaux

Les tableaux sont des listes ordonnées de valeurs, entourées de crochets `[]`. Les éléments du tableau peuvent être de différents types : objets, chaînes de caractères, nombres, booléens, ou même d'autres tableaux.

Exemple d'un tableau JSON :

```
[
  "Paris",
  "Londres",
  "Tokyo"
]
```

Types supportés

Les types de données pris en charge par JSON sont :

- **Chaînes de caractères** (string) : entourées de guillemets doubles `""` (ex : `"nom" : "Dupont"`).
- **Nombres** (int, float) : entiers ou décimaux sans guillemets (ex : `"age": 30`, `"taille": 1.80`).
- **Booléens** : valeurs logiques `true` ou `false` (ex : `"estMembre": true`).
- **Null** : pour indiquer l'absence de valeur (ex : `"adresse": null`).
- **Objets** : collections de paires clé-valeur, entourées d'accolades `{}`.
- **Tableaux** : listes ordonnées, entourées de crochets `[]`.

Exemples d'utilisation

JSON est principalement utilisé pour échanger des données entre un **client** (comme un navigateur) et un **serveur** dans des applications web. Par exemple, une API pourrait renvoyer les informations d'un utilisateur sous forme de JSON lorsqu'une requête est effectuée.

Exemple d'une réponse JSON d'une API :

```
{
  "utilisateur": {
    "id": 123,
    "nom": "Dupont",
    "email": "dupont@example.com"
  },
  "status": "success"
}
```

⚙️ | Encoder et décoder du JSON en PHP

https://www.php.net/manual/fr/ref.json.php

json_encode

Retourne la représentation JSON en string d'une valeur.

```php
$ma_var = [
    "utilisateur" => [
        "id" => 123,
        "nom" => "Dupont",
        "email" => "dupont@example.com"
    ],
    "status" => "success"
];

echo json_encode($ma_var);
```

🖥️ Sortie

```
{"utilisateur":
{"id":123,"nom":"Dupont","email":"dupont@example.com"},"status":"success"}
```

json_decode

Décode un string content JSON en une variable PHP.

```php
$mon_json = '{"utilisateur":
{"id":123,"nom":"Dupont","email":"dupont@example.com"},"status":"success"}';

$ma_var = json_decode($mon_json, true); // Le true permet de transformer
les objets en tableau associatif

var_dump($ma_var);
```

🖥️ Sortie

```
object(stdClass)#2 (2) {
  ["utilisateur"]=>
  object(stdClass)#1 (3) {
    ["id"]=>
    int(123)
    ["nom"]=>
    string(6) "Dupont"
    ["email"]=>
    string(18) "dupont@example.com"
  }
  ["status"]=>
  string(7) "success"
}
```

⚙️ | Variables prédéfinies

https://www.php.net/manual/fr/reserved.variables.php

https://www.php.net/manual/fr/language.variables.scope.php

En PHP, certaines **variables prédéfinies** sont disponibles à tout moment et facilitent la gestion des interactions entre le serveur, le client, et l'environnement d'exécution. Parmi ces variables, on trouve les **superglobales** et d'autres variables spécifiques.

Les superglobales

Les superglobales sont des tableaux associatifs qui contiennent des informations relatives à l'environnement d'exécution, aux requêtes HTTP, aux fichiers téléchargés, etc. Elles sont accessibles partout dans le script, sans besoin de les passer en argument.

$GLOBALS

Contient toutes les variables globales du script.

$_SERVER

Informations sur le serveur et l'environnement (ex. : `$_SERVER['HTTP_HOST']` pour le nom de l'hôte).

$_GET

Contient les données envoyées via une requête HTTP GET (ex. : depuis une URL).

URL : `http://example.com/page.php?nom=toto&age=30`

```
echo "Nom : " . $_GET['nom'];
echo "Âge : " . $_GET['age'];
```

🖥️ Sortie

```
Nom : toto
Âge : 30
```

$_POST

Contient les données envoyées via une requête HTTP POST (ex. : depuis un formulaire).

```
<form method="post" action="/traitement.php">
    <input type="text" name="prenom" placeholder="Votre prénom">
    <input type="submit" value="Envoyer">
</form>
```

```
// Affiche le prénom envoyé via le formulaire
echo "Prénom : " . $_POST['prenom'];
```

$_FILES

Contient les fichiers téléchargés via un formulaire.

```
<form action="/upload.php" method="post" enctype="multipart/form-
data">
    <input type="file" name="mon_fichier">
    <input type="submit" value="Uploader">
</form>
```

```
// Pas d'erreur
if ($_FILES['mon_fichier']['error'] === UPLOAD_ERR_OK) {
    $nom_du_fichier = $_FILES['mon_fichier']['name'];
    $nom_temporaire = $_FILES['mon_fichier']['tmp_name'];
    move_uploaded_file($nom_temporaire, "uploads/$nom_du_fichier");
}
// Une ou plusieurs erreurs
else {
    echo "Erreur lors du téléchargement.";
}
```

Voir la liste des erreurs d'upload de fichiers.

$_COOKIE

Contient les cookies envoyés par le client.

```
setcookie('utilisateur', 'toto', time() + 3600);

if (isset($_COOKIE['utilisateur'])) {
    echo 'Utilisateur : ' . $_COOKIE['utilisateur'];
}
```

🖥️ Sortie

```
Utilisateur : toto
```

$_SESSION

Les sessions permettent de conserver des informations d'un utilisateur entre différentes pages.

- **session_start()** : Doit être appelée pour initialiser la session.
- **session_destroy()** : Permet de détruire toutes les variables de session.

Exemple :

```
// Démarrer la session
session_start();

// Stocker des données dans la session
$_SESSION['utilisateur'] = 'toto';

// Accéder à la donnée
echo "Utilisateur : " . $_SESSION['utilisateur'];

// Détruire la session
session_destroy();
```

🖥️ Sortie

```
Utilisateur : toto
```

$_REQUEST

Combine les données de `$_GET`, `$_POST`, et `$_COOKIE`.

$_ENV

Contient les variables d'environnement du serveur.

```
echo $_ENV['HOME'];
```

🖥️ Sortie

```
/home/toto
```

Les autres variables prédéfinies

$argc

Contient le nombre d'arguments passés au script via la ligne de commande (comme en C).

$argv

Contient un tableau des arguments passés via la ligne de commande (comme en C).

📚 | TD 3 : Formulaires et variables prédéfinies

Ce TD vise à créer un système minimaliste et non-sécurisé de connexion et création de compte utilisateur. Un fichier JSON nommé `users.json` servira à stocker les utilisateurs. Les inputs HTML des formulaires ont volontairement été simplifiés afin de faciliter le processus de test de votre code (si j'avais mis des required, il aurait fallu remplir les champs à chaque fois).

Cloner le projet

Si ce n'est pas déjà fait, cloner le projet https://github.com/PHP-BUT2-DACS/TP-TD

```
git clone https://github.com/PHP-BUT2-DACS/TP-TD.git
```

Faire les exercices

1. Lancer le serveur PHP dans le dossier `/Séance 2/TD3`

```
php -S localhost:8080
```

2. Accédez à la page http://localhost:8080/login.php

3. Répondez aux exercices du TD3. Vous pouvez demander de l'aide si nécessaire.

ASTUCE

Pour mieux avancer dans le TD, il est conseillé de remplir les fichiers dans cet ordre :

1. `profile.php`
2. `logout.php`
3. `singup.php`
4. `login.php`

ASTUCE

N'oubliez pas de regarder votre fichier `users.json` après la création d'un utilisateur.