

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
- 📄 | Anatomie du protocole HTTP
 - Les composantes d'une requête HTTP
 - Ligne de requête (Request Line)
 - En-têtes de requête (Request Headers)
 - Corps de la requête (Request 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

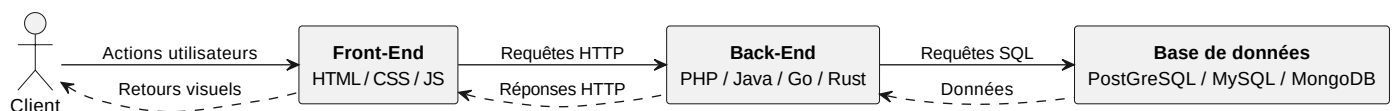
📄 | Architecture d'une application web

! INFO

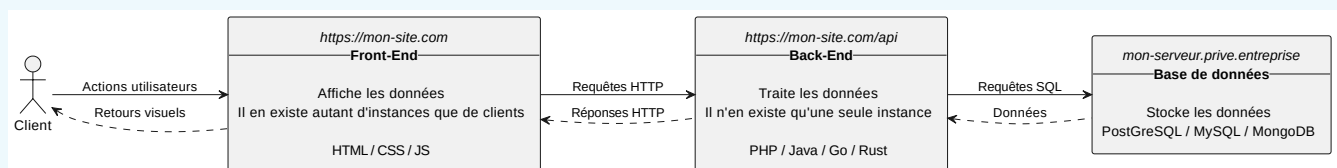
Cette partie n'est pas à apprendre en détails. Cependant, la compréhension des différents termes évoqués est nécessaires.

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



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).

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.

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 **boulangier** 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 **boulangier**. Le boulangier, 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 boulangier 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 interagit avec le boulangier ou le stock de ressources.



| 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.

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 :

HTTP Version	Space	Status Code	Space	Status Phrase	Response Status Line
Header Field Name	Space	Value	Space		
...					Response Headers
Header Field Name		Value	Space		
Blank line					Response Body
Message Body					

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 (Request 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 (Request 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.