IHM Web

Interface Homme-Machine sur le Web

Introduction

On appelle **interface homme-machine** (IHM) tout ce qui permet à une personne de communiquer avec une machine : des éléments matériels que l'on touche ou que l'on regarde, et des éléments logiciels qui réagissent à nos gestes.

Prenons un pavé tactile de portable. Le doigt glisse sur le pavé (geste de l'utilisateur), le pavé mesure ce déplacement (matériel), le système d'exploitation déplace le pointeur à l'écran (logiciel), et un appui ouvre le fichier visé (logiciel). Chaque geste produit une réponse visible : c'est ce dialogue qui fait l'interface.

Une bonne IHM part toujours des besoins de celui qui l'utilise : elle doit être claire, prévisible et facile à prendre en main. Dans ce chapitre, on s'intéresse à l'IHM que tout le monde utilise chaque jour : le navigateur Web, et la façon dont il obtient les pages qu'il affiche.

Protocole HTTP

Un **protocole** est un ensemble de règles que deux machines suivent pour se comprendre. Le protocole **HTTP** (*Hypertext Transfer Protocol*, protocole de transfert hypertexte) fixe la façon dont un **client** (le navigateur : Firefox, Chrome, Safari…) et un **serveur Web** (un programme comme Apache ou nginx, installé sur une machine connectée en permanence) échangent des messages. Ces messages sont écrits en texte ordinaire ; pour circuler sur le réseau, ils sont confiés aux protocoles TCP et IP.

Voici ce qui se passe lorsque l'on saisit l'adresse `http://www.exemple.fr/cours/http.html` :

1. le navigateur isole le **nom de domaine** `www.exemple.fr` ;
2. il demande à un serveur DNS l'**adresse IP** qui correspond à ce nom, par exemple `203.0.113.5` (le numéro d'une machine sur Internet) ;
3. il ouvre une connexion TCP avec la machine qui porte cette adresse ;
4. il envoie une **requête** HTTP qui demande la ressource `/cours/http.html` ;
5. le serveur cherche ce fichier et le renvoie dans une **réponse** HTTP ;
6. le navigateur lit le code HTML reçu et construit la page à l'écran.

En-tête HTTP d'une requête

```
GET /cours/http.html HTTP/1.1
Host: www.exemple.fr
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 14_5) Firefox/128.0
Accept: text/html
Accept-Language: fr
```

`GET` est la **méthode**, `/cours/http.html` le chemin de la ressource, `HTTP/1.1` la version du protocole. `Host` rappelle le nom de domaine, éventuellement suivi d'un **numéro de port** (80 par défaut pour HTTP, 443 pour HTTPS). `User-Agent` décrit le navigateur et le système, `Accept` le type de contenu souhaité, `Accept-Language` la langue préférée.

Les quatre méthodes à connaître :

— `GET` demande une ressource sans la modifier (clic sur un lien) ;

— `POST` envoie des données au serveur pour qu'il les traite (formulaire) ;

— `PUT` dépose une ressource sur le serveur, ou remplace celle qui existe ;

— `DELETE` demande la suppression d'une ressource.

En-tête HTTP d'une réponse

```
HTTP/1.1 200 OK
Date: Tue, 13 Oct 2026 08:15:42 GMT
Server: nginx/1.26.2
Content-Type: text/html; charset=UTF-8
Content-Length: 182
```

La première ligne est la **ligne de statut** : version du protocole, puis **code de réponse** (`200` : la ressource est jointe). Viennent ensuite les champs de l'en-tête, une ligne vide, puis le **corps** : le code HTML de la page, que le navigateur interprète pour dessiner la page.

Formulaires : GET et POST

Un **formulaire**, délimité par la balise `<form>`, envoie au serveur les informations saisies. L'attribut `action` donne l'adresse du programme du serveur qui recevra les données, l'attribut `method` la méthode HTTP : `get` (par défaut) ou `post`. Chaque champ porte un attribut `name` : c'est ce nom, associé à la valeur saisie, que le navigateur transmet.

```
<form action="/inscription" method="get">
  <label for="prenom">Prénom</label>
  <input type="text" id="prenom" name="prenom">
  <label for="age">Âge</label>
  <input type="number" id="age" name="age">
  <input type="submit" value="Envoyer">
</form>
```

Avec **GET**, les paramètres saisis (`Leo` et `16`) s'ajoutent à l'adresse, après un point d'interrogation, sous la forme `nom=valeur` séparés par `&` :

```
GET /inscription?prenom=Leo&age=16 HTTP/1.1
Host: www.exemple.fr
```

Ils restent visibles dans la barre d'adresse, l'historique et les journaux du serveur ; GET convient à une recherche ou à un filtre, qui ne modifient rien sur le serveur.

Avec **POST**, les mêmes paramètres voyagent dans le corps de la requête, après la ligne vide :

```
POST /inscription HTTP/1.1
Host: www.exemple.fr
Content-Type: application/x-www-form-urlencoded
Content-Length: 17

prenom=Leo&age=16
```

On emploie POST pour un mot de passe, une inscription ou une commande, c'est-à-dire pour les envois qui modifient des données sur le serveur. POST ne chiffre rien pour autant : seul HTTPS protège les données pendant leur transport.

Ce qui s'exécute sur le client et sur le serveur

1. le navigateur affiche la page ; un script JavaScript peut s'exécuter **sur le client**, par exemple pour vérifier qu'un champ n'est pas vide, avant tout envoi ;
2. au clic sur le bouton, le navigateur construit la requête GET ou POST et l'envoie ;
3. **sur le serveur**, un programme (en PHP ou en Python, par exemple) lit les paramètres, consulte ou modifie une base de données, puis fabrique une page HTML ;
4. le serveur renvoie cette page dans sa réponse, et le navigateur l'affiche.

Le client ne voit jamais le code du serveur, seulement la page produite ; le serveur ne voit pas ce que fait le script du client : une vérification faite en JavaScript peut être contournée, et le serveur doit refaire ses propres contrôles.

Les cookies

HTTP ne garde aucun souvenir d'une requête à l'autre. Pour reconnaître un visiteur (panier, connexion), le serveur ajoute à sa réponse un champ `Set-Cookie: session=a3f9c2`. Le navigateur **mémorise** ce cookie et le **renvoie** au même site dans chaque requête suivante, dans le champ `Cookie: session=a3f9c2`. Le cookie est stocké sur le client : on peut l'effacer dans les réglages du navigateur.

Protocole HTTPS

Les messages HTTP circulent en clair. Le protocole **HTTPS** ajoute une couche de **chiffrement** : les messages ne sont lisibles que par le client et le serveur.

1. le navigateur demande une connexion sécurisée (adresse en `https://`) ;
2. le serveur présente un **certificat**, carte d'identité numérique délivrée par un organisme de confiance, que le navigateur vérifie ;
3. client et serveur conviennent d'une **clé de chiffrement**, puis tous leurs échanges sont chiffrés.

Le certificat protège contre l'attaque de l'**intermédiaire** (*man in the middle*) ; le chiffrement rend inutilisables les données interceptées. Avant de saisir un mot de passe ou des coordonnées bancaires, on vérifie que l'adresse commence par `https://`.

Codes de réponse HTTP

Premier chiffre : 2 succès, 3 redirection, 4 erreur du client, 5 erreur du serveur.

Code	Signification
200 OK	La requête a réussi ; la ressource est dans le corps
301 Moved Permanently	Changement d'adresse définitif (champ `Location`)
302 Found	Ressource provisoirement à une autre adresse (champ `Location`)
400 Bad Request	Requête mal formée
401 Unauthorized	Ressource protégée : il faut s'identifier
403 Forbidden	Accès refusé, même après identification
404 Not Found	Aucune ressource à cette adresse
500 Internal Server Error	Erreur du serveur en préparant la réponse

Principaux champs HTTP

Champs de requête

Champ	Rôle
`Host`	Site demandé (une machine peut en héberger plusieurs)
`User-Agent`	Navigateur et système, ou nom du robot (`Googlebot`)
`Referer`	Page d'où vient le visiteur
`Cookie`	Cookies reçus de ce site, renvoyés à chaque requête

Champs de réponse

Champ	Rôle
`Content-Type`	Type MIME de la ressource (`text/html; charset=UTF-8`)
`Server`	Logiciel serveur
`Set-Cookie`	Cookie à garder et à renvoyer au même site
`Location`	Adresse de destination d'une redirection