Web Client

VAN DE MERGHEL Robin

2023

Table des matières

1 Notions de base

1.1 Différence entre Net et Web

1.1.1 Le Net

Le **Net** (⇔ Internet) est l'ensemble des connections entre les machines qui forme un imense réseau (Internet). Le **Web** c'est l'ensemble des ressources qui sont distribuées sur internet (sites, images, documents, …). Le **Web** est fait pour être évolutif (on peut améliorer les façons de communiquer par exemple).

1.1.2 Client vs Serveur

La différence **Client / Serveur** : Le client c'est toi, ton entourage, etc… Vous demandez aux serveurs des données (comme un site, ou la dernière vidéo de untel). Les serveurs c'est ce qui stock et traîte les données / requêtes des clients. Ex : les **Serveurs de Google** répondent à ta recherche

1.2 Côté serveur

1.2.1 Protocole

Un **protocole** (+ de la culture G, mais peut être utile pour rep), un ensemble de règles prédéfinies qui permettent à deux élément d'un réseaux (ex : Toi et ta box wifi) de discuter (quelques protocoles qu'on a vu en cours, à connaître de nom : **RSS, TCP, IP, HTTP(s)**)

1.2.2 Site web

Un **Site Web** n'est *pas* une machine. Si je reprends google en exemple, google n'a pas que une machine pour leur site. Donc un site c'est **l'ensemble des ressources** (page HTML, CSS, images, documents PDF, …)

1.2.3 Deux types de serveurs : statiques, et dynamiques

Ajout sur les **Serveurs** : Le **Serveur** va aussi gérer les ressources (supprimer, créer, envoyer, …), et on va distinguer : **Serveurs statiques** (page statique, genre présentation d'une personne), et **Serveurs dynamiques** (avec page de connection, calculs par du javascript, …)

1.2.4 Le cache

Le **Cache** est une méthode en informatique super utile pour accélérer les choses. Le **Cache** c'est un stockage temporaire des données (ex : les images d'un site web). Le **Cache** permet de ne pas avoir à télécharger les données à chaque fois, et donc d'accélérer le chargement des pages.

Il permet aussi si le serveur ne fonctionne plus, de pouvoir quand même afficher les données (ex : si le serveur de google tombe, on peut quand même afficher certaines pages de google).

1.2.5 Le Proxy

Le **Proxy** se trouve entre le **Client** et le **Serveur** du côté du **Client**. Il permet de faire des choses comme :

- **Cacher** (not hide, to cache) les données (comme le **Cache**)
- **Chiffrer** les données (pour les protéger)
- **Filtrer** les données (pour les protéger, exemple : les collèges et lycées qui bloquent certains sites)
- **Accélérer** les données (pour les accélérer, exemple : les serveurs de google qui sont répartis dans le monde entier)

1.2.6 Le Reverse Proxy

Le **Reverse Proxy** est un **Proxy** qui se trouve entre le **Client** et le **Serveur**, mais cette fois-ci, du côté du **Serveur**. Il permet de faire des choses comme :

- **Relayer** les données (pour les accélérer, exemple : les serveurs de google qui sont répartis dans le monde entier)
- **Chiffrer** les données (pour les protéger)

1.2.7 CDN (Content Delivery Network)

Le **CDN** (Content Delivery Network) est un **Reverse Proxy** qui permet de faire des choses comme :

- **Mirrorring** (copie) des données (pour sécuriser les données, exemple : les serveurs de google qui sont répartis dans le monde entier, si un serveur tombe, on peut quand même afficher les données)
- **Accélérer** les données (il y a des serveurs procches de toi, donc plus rapide)

1.2.8 Hypermédia

L'**Hypermédia** c'est un ensemble de données qui sont liées entre elles. Ex : un site web, un document PDF, …

Ils ont une adresse : **URI** (Uniform Resource Identifier) qui permet de les identifier.

1.2.9 URI

L'**URI** (Uniform Resource Identifier) c'est l'identifiant d'une ressource. Il y a deux types d'**URI** :

- **URL** (Uniform Resource Locator) : l'adresse d'une ressource (ex : https://google.com)
- **URN** (Uniform Resource Name) : le nom d'une ressource (ex : urn:isbn:0451450523)

Un **URI** n'est pas clair : c'est le serveur qui lui donne un sens. Ex : `http://youtube.com/watch?v=j3VzaBcoDAI` c'est pas clair pour un humain, mais le serveur va comprendre que c'est une vidéo youtube qui a pour nom `XXXX`…

1.2.10 DNS (Domain Name System)

Le **DNS** (Domain Name System) est un service qui permet de convertir un **URI** en une adresse IP (ex : `google.com` => `142.250.201.164`). C'est un service qui est ultra important, car sans ça, on ne pourrait pas accéder aux sites web facilement (tu devrais apprendre les adresses IP par coeur).

1.2.11 Protocole HTTP

1.2.11.1 Présentation générale Le **Protocole HTTP** (HyperText Transfer Protocol) est un protocole qui permet de communiquer entre un **Client** et un **Serveur**.

Comme tout les protocoles, il y a la même composition :

- Le **schéma** (ex : `http`, `https`, `ftp`, …), c'est le nom du protocole
- Le **hôte** (ex : `google.com`, `github.com`, …), c'est l'adresse du serveur
- Le **port** (ex : `80`, `443`, `21`, …), c'est le port du serveur
- Le **chemin** (ex : `/`, `/search`, `/user`, …), c'est le chemin de la ressource
- Les **paramètres** (ex : `?q=hello`, `?id=1`, …), c'est les paramètres de la requête
- Les **fragments** (ex : `#hello`, `#world`, …), c'est les fragments de la requête

Exemple d'**URI** :

- `http://google.com`
- `https://google.com/search?q=hello`

Le protocol HTTP est divisé en trois parties :

- **start-line** : la première ligne de la requête (ex : `GET / HTTP/1.1`), permet d'identifier la version du protocole, le type de requête, et le chemin de la ressource
- **headers** : les lignes suivantes (ex : `Host: google.com`), permettent d'envoyer des informations supplémentaires
- **body** : le corps de la requête (ex : `q=hello`), permet d'envoyer des données supplémentaires

1.2.11.2 Idempotence

L'**Idempotence** c'est la capacité d'un protocole à être répété sans changer le résultat. Ex : `GET` est idempotent, `POST` non.

En reformulant, si je fais 300 fois la même requête `GET`, le résultat sera le même. Si je fais 300 fois la même requête `POST`, le résultat sera différent.

1.2.11.3 Effet de bord

L'**Effet de bord** c'est la capacité d'un protocole à modifier le serveur. Ex : `GET` n'a pas d'**Effet de bord**, `POST` oui.

Si je fais une requête `GET`, le serveur ne va pas changer. Si je fais une requête `POST`, le serveur va changer (créer un utilisateur, …).

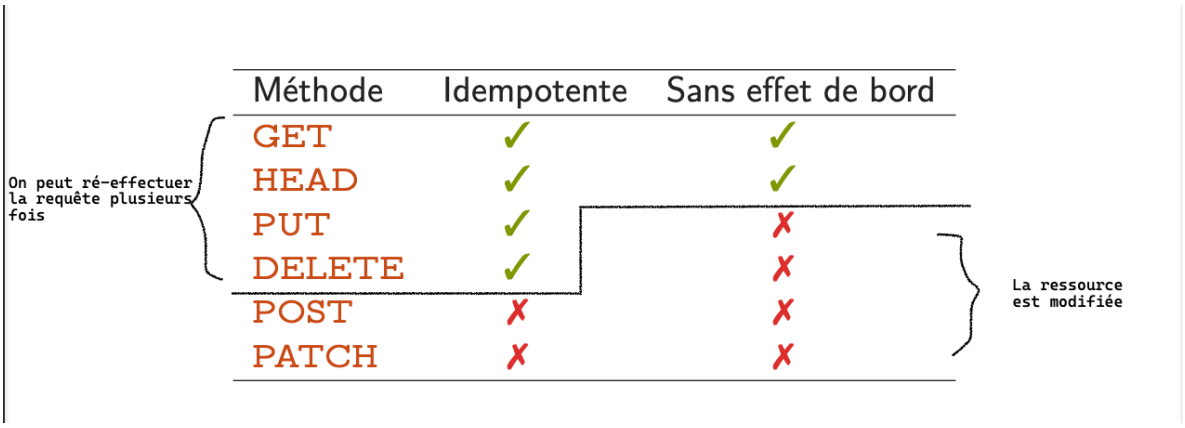

Figure 1: Effet de bord

1.2.11.4 Les erreurs

Les erreurs HTTP sont des codes qui permettent de savoir si la requête a fonctionné ou non. Ex : `200` c'est que la requête a fonctionné, `404` c'est que la ressource n'a pas été trouvée.

Il y a 5 catégories d'erreurs :

- **1xx** : Information
- **2xx** : Succès
- **3xx** : Redirection
- **4xx** : Erreur du client
- **5xx** : Erreur du serveur

1.3 Côté client

1.3.1 Un agent

Un **Agent** c'est ce qui permet à un **Client** d'intéragir avec le serveur : navigateurs web (Chrome, firefox, …), robots (pour référencer les sites par exemple), agrégateurs (des outils pour écouter les protocoles)

1.3.2 Clients lourds, légers, riches

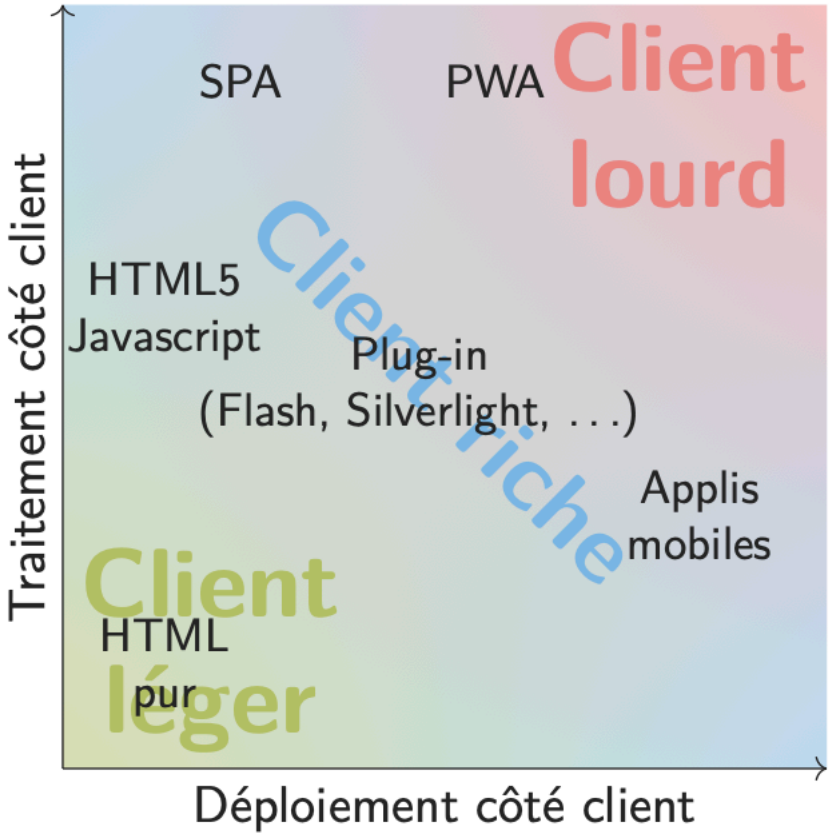

Figure 2: Clients lourds, légers, riches

2 Le CSS

2.1 Présentation générale

Le **CSS** (Cascading Style Sheets) est un langage qui permet de définir le style d'une page web. Il est composé de **règles** qui sont composées de :

- **sélecteurs** : qui permettent de sélectionner les éléments (ex : `h1`, `p`, `#id`, `.class`, …)
- **propriétés** : qui permettent de définir le style (ex : `color`, `background-color`, `font-size`, …)
- **valeurs** : qui permettent de définir la valeur de la propriété (ex : `red`, `blue`, `12px`, …)

2.2 Les sélecteurs

Les principaux **sélecteurs** sont :

- Les différentes balises HTML avec juste leur nom (ex : `h1`, `p`, ..., j'en énumère après)
- Les classes avec un `.` devant (ex : `.salut`, `.hello`, ...)
- Les id avec un `#` devant (ex : `#salut`, `#hello`, ...)
- Les attributs avec un `[` devant (ex : `input[type="text"]`, `input[type="password"]`, ...)

2.3 Différences entre `id` et `class`

Les **id** et les **classes** sont des **sélecteurs** qui permettent de sélectionner des éléments. La différence entre les deux c'est que les **id** sont uniques, alors que les **classes** ne le sont pas.

Je peux dire "Il y a des cellules d'un tableau", et "Il y a la 4e cellule en partant du haut". D'un côté je parle de plusieurs cellules, de l'autre je parle d'une cellule en particulier. C'est la même chose pour les **id** et les **classes** : les **id** c'est pour une seule chose, les **classes** c'est pour plusieurs choses.

Note : Qu'on utilise une `class` ou un `id` c'est une question de goût, mais il faut rester cohérent dans son code.

2.4 Les différentes balises HTML

Les grandes balises utiles en HTML sont :

- `h1` à `h6` : les titres
- `p` : les paragraphes
- `div` : les divisions (pour faire des blocs)
- `span` : les spans (pour faire des éléments en ligne)
- `ul` : les listes
- `li` : les éléments de liste
- `a` : les liens
- `img` : les images
- `form` : les formulaires
- `input` : les champs de saisie
  - `input[type="text"]` : les champs de texte
  - `input[type="password"]` : les champs de mot de passe
  - `input[type="submit"]` : les boutons de validation
  - `input[type="button"]` : les boutons radio
  - ...
- `button` : les boutons
- `table` : les tableaux
- `tr` : les lignes de tableau
- `td` : les cellules de tableau
- `thead` : l'entête du tableau
- `tbody` : le corps du tableau
- `label` : les labels
- `svg` : les images vectorielles

2.5 Les propriétés

Les principales **propriétés** sont :

2.5.1 Les unités

- `px` : pixels
- `em` : taille de la police
- `rem` : taille de la police de base
- `%` : pourcentage

Des fois pour le `body` (le parent de tout les éléments de la page) dire "Je prends `100%` de la page" ne marche pas, il faut dire "Je prends 100% de l'écran".

Pour cela, on utilise `vh` (pourcentage de la hauteur de l'écran) et `vw` (pourcentage de la largeur de l'écran). Le `v` vient de `viewport` (la fenêtre du navigateur).

Donc à l'instar de `100%` on peut dire `100vh` et `100vw`.

2.5.2 Les couleurs

- `color` : la couleur du texte
- `background-color` : la couleur de fond
- `border-color` : la couleur de la bordure

2.5.3 Les polices

- `font-family` : la police (ex : `Arial`, `Helvetica`, `Times New Roman`, …)
- `font-size` : la taille de la police (ex : `12px`, `1em`, `1rem`, …)
- `font-weight` : l'épaisseur de la police (ex : `normal`, `bold`, `bolder`, …)
- `font-style` : l'italique de la police (ex : `normal`, `italic`, `oblique`, …)
- `text-align` : l'alignement du texte (ex : `left`, `right`, `center`, `justify`, …)
- `text-decoration` : le soulignement du texte (ex : `none`, `underline`, `overline`, `line-through`, …)
- `text-transform` : la transformation du texte (majuscules, minuscules, …)
- `line-height` : la hauteur de ligne
- `letter-spacing` : l'espacement des lettres
- `word-spacing` : l'espacement des mots

2.5.4 Les marges

- `margin` : la marge extérieure (ex : `10px`, `1em`, `1rem`, …)
  - `margin-top` : la marge extérieure du haut
  - `margin-right` : la marge extérieure de droite
  - …
- `padding` : la marge intérieure (ex : `10px`, `1em`, `1rem`, …)
  - `padding-top` : la marge intérieure du haut
  - `padding-right` : la marge intérieure de droite
  - …

La différence entre les marges et les paddings c'est que les marges sont extérieures, alors que les paddings sont intérieures.

Je le vois perso comme ça :

- Les marges, c'est je pousse mes voisins pour avoir plus de place dans un canapé (le canapé c'est le bloc qui contient l'élément)
- Les paddings c'est je m'installe en prenant de la place dans un canapé

2.5.5 Les bordures

- `border` : la bordure (ex : `1px solid black`, `1px solid red`, …)
  - `border: [largeur] [style] [couleur]`
  - `border-top` : la bordure du haut
  - `border-right` : la bordure de droite
  - …
- `border-radius` : le rayon des coins (ex : `10px`, `1em`, `1rem`, …, `24px` c'est nickel pour des boutons souvent)

2.5.6 Les dimensions

- `width` : la largeur (ex : `100px`, `10em`, `10rem`, …)
- `height` : la hauteur (ex : `100px`, `10em`, `10rem`, …)
  - `max-width` : la largeur maximale (ex : `100px`, `10em`, `10rem`, …)
  - `min-width` : la largeur minimale (ex : `100px`, `10em`, `10rem`, …)
  - …

2.5.7 Les positions

- `position` : la position (ex : `static`, `relative`, `absolute`, `fixed`, …)
- `top` : la position du haut (ex : `100px`, `10em`, `10rem`, …)
- `right` : la position de droite (ex : `100px`, `10em`, `10rem`, …)
- …
- `z-index` : l'index de profondeur (ex : `1`, `2`, `3`, …), c'est comme des couches, plus c'est grand plus c'est en avant
- `float` : le flottage (ex : `left`, `right`, `none`, …)

Les positions sont très importantes, je conseille de bien les comprendre.

- `block` : par défaut, les éléments sont en `block`, c'est à dire qu'ils prennent toute la largeur disponible
- `relative` : l'élément est positionné par rapport à lui-même
- `absolute` : en gros se positionne par rapport à son parent via des coordonnées, et il ignore ses frères et sœurs
- `fixed` : se positionne par rapport à la fenêtre du navigateur, et il ignore ses frères et sœurs (comme `absolute` mais par rapport à la fenêtre du navigateur)

2.5.8 Les affichages (TRÈS IMPORTANT)

Je ne vais parler que de `display` ici, car vraiment c'est à maîtriser absolument.

C'est aussi ce qui permet de centrer un élément de façon propre, et de faire des mises en page complexes.

Par défaut, il y a `block`, par défaut, c'est à dire qu'ils prennent toute la largeur disponible

Les affichages modernes sont les suivants :

2.5.8.1 `Grid`

`Grid` est un affichage qui permet de découper une section en grille, et de placer des éléments dans cette grille.

C'est très pratique pour faire des mises en page complexes, surtout avec le **responsive**, et c'est très simple à utiliser :

```
.container { /* On veut faire une grille de 3 colonnes et 3 lignes */
  display: grid;
```

```
  grid-template-columns: 1fr 1fr 1fr; /* 3 colonnes de même taille */
  grid-template-rows: 1fr 1fr 1fr; /* 3 lignes de même taille */
}
```

Un **excellent** cours sur grid est disponible sur CSS-Tricks.

2.5.8.2 `Flex`

`Flex` est un affichage qui permet de placer des éléments les uns à côté des autres, et de les aligner.

Comme sont nom l'indice, c'est très pratique pour faire des mises en page flexibles, surtout avec le **responsive**, et c'est très simple à utiliser :

```
.container { /* On veut faire une grille de 3 colonnes et 3 lignes */
  display: flex;
  flex-direction: row; /* Par défaut, c'est row, mais on peut mettre column */
  justify-content: center; /* Alignement horizontal */
  align-items: center; /* Alignement vertical */
}
```

Un **excellent** cours sur flex est disponible sur CSS-Tricks.

2.5.9 Exemple

Un petit exemple de ce que l'on peut faire avec CSS (`#Autopub`) ici (Code source disponible ici).