ARCHITECTURE APPLICATION WEB CLASSIQUE

FRONTEND → Affichage, UI, UX

FRONTEND / CLIENT

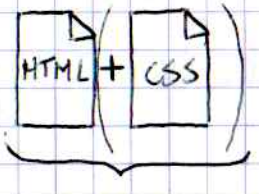
(dans le navigateur)

- HTML
- CSS
- (Parfois, un tout petit peu de JS)

- Le client affiche le HTML et CSS que le serveur lui envoie
- Le client est "stupide" : il ne fait qu'afficher le HTML et CSS que le serveur lui envoie.
- Le client doit constamment faire appel au serveur (pour passer d'une page à une autre par exemple : le client doit demander au serveur de lui envoyer le HTML et CSS de la nouvelle page)

Requête HTTP

(ex : GET http://hello.com/home/)



HTML (+ CSS) de la page http://hello.com/home/

BACKEND / SERVEUR

→ logique, algorithmes

- PHP (+ Symfony) ou Python (+ Django)
- Le serveur fait tout (routage, traitement, interaction avec la BDD, calcul du HTML + CSS à envoyer au client...)
  - Les "philosophies" du backend et du frontend sont mélangées dans le serveur !

ROUTEUR (PHP)
- Traduit une requête HTTP en une action que doit réaliser le serveur

Fais ça !

TRAITEMENT
- Algorithmes
- Logique de l'application

OK

HTML + CSS

SQL

BDD (base de données)

* client = application qui tourne dans le navigateur de l'utilisateur.

- INCONVÉNIENTS : • Le serveur fait tout • Besoin d'interroger le serveur très souvent • Les "philosophies" du backend et du frontend sont mélangées dans le serveur (difficile à maintenir) • Pas très flexible ! • Peu "moderne".

ARCHITECTURE SPA (SINGLE PAGE APPLICATION)

FRONTEND / CLIENT

- HTML
- CSS
- Beaucoup de JS

- Le client possède tout le HTML + CSS de toutes les pages de l'application
  - HTML + CSS envoyés en une fois par le serveur/API lors de la 1e connexion.

UTILISATEUR → (1*) → ROUTEUR (JS)

ROUTEUR (JS) → OK → TRAITEMENT

HTML + CSS → TRAITEMENT

HTML à trous (à compléter) → TRAITEMENT

TRAITEMENT :
- "Calcul" de l'affichage.
- Complète le HTML fourni par le routeur avec les infos récupérée auprès du serveur / API

- Le client est plus "intelligent" : il gère tout seul la construction du HTML des pages. Il est moins dépendant au serveur/API, et concentre tout ce qui tourne autour de l'affichage, UI, UX, etc...

"J'ai besoin de connaître ma position !"
(GET http://google-api/geo/where-am-i/)

API Google Maps
- Géolocalisation

OK : "T'es ici !"
JSON :
{"latitude" : "48°",
"longitude" : "2°"}

"Trouve-moi les restos autour d'ici"
JSON :
{"latitude" : "48°",
"longitude" : "2°"}

(GET http://opcat-api.fr/restos-autour-de-moi)

JSON :
{"L'en K" : {
  "adresse" : "...",
  "horaires" : "..."
},
"MaDo" : {
  "adresse" : "...",
  "horaires" : "...",
  ...
},
}

OK.

On peut faire appel à des API pleins d'API (remplacent les plugins WordPress)

BACKEND / SERVEUR / API SPEAT

- PHP (+ Symfony)
- Ne s'occupe que de la logique de l'application (interroger la BDD, etc...)
- On lui demande des services via des requêtes HTTP (avec des arguments au mini en données au format JSON)
- Repond au client avec des données au format JSON

BDD SPEAT

LES ÉCHANGES ENTRE LE FRONTEND ET LE BACKEND NE SE FONT QU'EN JSON.

*1 : "Je veux afficher la page qui me montre tous les restos autour de moi !"
(requête HTTP : GET http://pwa.opcat.fr/restos-autour-de-moi/)