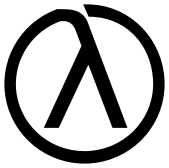

Une brève histoire du développement d'applications web

Le (world wide) web est né en 1990 à Genève dans les laboratoires du CERN. Sa fonction première était de permettre la diffusion et la consultation de documents hypertexte. La façon la plus simple de concevoir ces derniers était de les stocker dans des fichiers statiques[1] sur des serveurs (serveurs HTTP) capables de les diffuser à la demande (requêtes HTTP) auprès de postes clients équipés d'un logiciel adéquat (un navigateur).

Mais assez vite il s'est agit d'imaginer comment on pouvait rendre dynamiques ces documents, ou au moins des parties de ces documents. Imaginons par exemple un document qui soit un rapport incorporant des tableaux de données statistiques. On conçoit aisément qu'il serait utile que ces données soient systématiquement les dernières données à jour, collectées directement dans une base de données.

Une fois cette dimension dynamique introduite, il n'y plus qu'un pas à effectuer pour passer de la notion de document à celle d'application. Truffez votre document d'éléments dynamiques, certains pouvant être des menus présentant des fonctions, ajoutez-y des éléments de formulaires pour effectuer des saisies de données ainsi qu'un mécanisme pour que des informations puissent circuler non plus seulement du serveur vers le navigateur mais également, au moins occasionnellement, du navigateur vers le serveur et vous vous retrouvez finalement avec une application.

1. Première ère : l'interface CGI

Dans le milieu des années 1990, Internet commence à s'ouvrir au grand public et aux entreprises, après avoir été réservé aux militaires (américains), auxquels se sont ensuite ajoutés les universitaires (du monde occidental). Cette ouverture a inévitablement fait naître un marché pour des applications web.

Les premières applications web, celles de la fin des années 1990, s'articulaient autour de deux apports techniques :

- L'adjonction au langage HTML d'éléments permettant de réaliser des formulaires de saisie.
- L'emploi par les serveurs HTTP de la « Common Gateway Interface » (**CGI**).

Cette dernière technologie consiste à faire en sorte que le serveur HTTP, à la réception d'une requête, exécute un programme et en capture la sortie standard pour la rerouter vers la réponse à la requête HTTP. Il devient donc possible d'écrire des programmes qui produisent des interfaces d'utilisation en écrivant du HTML sur la sortie standard.

---

1. Petite remarque au passage, si vous êtes en train de lire le présent texte dans sa version en ligne, c'est précisément un fichier statique qui vous été servi exactement dans l'esprit originel du web.

Cette technologie procède d'une **exécution côté serveur**. Le client est constitué d'un simple navigateur générique dont la seule capacité est d'afficher le rendu du HTML et d'envoyer des requêtes HTTP au serveur.

Tout langage[2] capable d'écrire sur la sortie standard peut potentiellement être utilisé pour réaliser une telle application. Mais comme il s'agit de générer du code HTML, donc des chaînes de caractères, les langages les mieux dotés en la matière auront tendance à être favorisés. C'est pourquoi cette période fût l'âge d'or du langage Perl.

2. Deuxième ère : Dynamic HTML et la guerre des navigateurs

Au fur et à mesure de l'augmentation du nombre d'utilisateurs des applications web, l'approche CGI à commencé à révéler quelques faiblesses en matière de performances. Le nœud du problème est que chaque requête HTTP y donne lieu à l'exécution d'un programme, ce qui au niveau du système d'exploitation du serveur se traduit par le démarrage d'un processus avec le temps d'initialisation et l'empreinte mémoire que cela implique.

On est donc sur un schéma dans lequel N requêtes HTTP provoquent, à leur réception par le serveur, la mise en route de N processus. Quand N peut atteindre des ordres de grandeurs en dizaines de milliers, en centaines de milliers ou en millions, cela pose un problème évident (qui plus est aux machines de l'époque).

Parallèlement à cela, cette période (nous sommes alors dans les années 2000) connaît également l'équipement d'un nombre croissant de foyers avec des ordinateurs de plus en plus puissants. Ces ordinateurs, désormais capables de faire tourner des jeux vidéos en trois dimensions, ne font qu'une bouchée d'un programme comme un navigateur.

La situation qui commence alors à se dessiner est donc celle dans laquelle on a d'un côté des serveurs qui croulent sous la charge de travail et d'un autre côté des postes clients dont la puissance croissante est complètement sous utilisée par le navigateur dans le cadre d'une application web. La tentation est donc naturellement grande de chercher des moyens de déplacer une partie de la charge de travail du serveur vers le poste client.

C'est là qu'intervient le langage **Javascript**. Il s'agit d'un langage interprété, dont un interpréteur est désormais embarqué dans les navigateurs et qui est capable de manipuler la structure du document HTML affiché au moyen de son « Document Object Model » (**DOM**).

Il devient alors possible d'alléger le programme côté serveur en lui faisant générer des scripts Javascript capables de créer procéduralement le contenu à afficher. Le serveur se trouve alors libéré de toute une partie de la charge qui lui incombait auparavant.

Par exemple, au lieu de générer les dizaines ou centaines de balises HTML correspondant à un tableau de résultats à afficher, le script peut se charger de générer ces balises côté client à partir d'une expression beaucoup plus concise de ces résultats.

---

2. À titre d'exemple, l'auteur de ces mots a conçu et réalisé sa toute première application web CGI à titre professionnel en langage C++, qui n'est pourtant généralement pas considéré comme un « langage du web ».

Ce mélange de HTML et de Javascript le manipulant est alors dénommé **Dynamic HTML**.

3. Troisième ère : les serveurs de pages

Nous sommes désormais dans le milieu des années 2000 et une part importante des applications web reposent désormais sur la capacité des navigateurs à exécuter vite et bien des scripts Javascript. Précédemment, dans les années 1990, le marché des navigateurs était dominé par Netscape Navigator, mais désormais la guerre des navigateurs bat son plein entre ce dernier et Internet Explorer de Microsoft.

Seulement cette guerre d'innovation se déroule en dehors de tout cadre commun de normalisation, aussi bien en ce qui concerne les balises HTML que le langage Javascript. Il en résulte qu'au fur et à mesure que sortent les versions successives de l'un et l'autre de ces navigateurs (plus quelques autres moins répandus), les incompatibilités se multiplient.

Pour les développeurs d'applications web, il devient alors de plus en plus difficile de garantir leur bon fonctionnement sur tous les navigateurs du marché. L'idée que l'on peut maîtriser le fonctionnement du côté serveur et non du côté client prend de plus en plus de valeur à leurs yeux et il devient de nouveau séduisant de rapatrier un maximum (voire la totalité) des traitements côté serveur.

Mais les problèmes de performances de l'époque des programmes CGI ne sont pas oubliés. L'on s'emploie donc à y remédier en cherchant le moyen de faire en sorte que plusieurs requêtes HTTP successives puisse être traitées par un seul et même processus au lieu que chacune d'elles nécessite le sien.

C'est dans cette démarche que naissent différentes technologies de **serveurs de pages**. L'on peut notamment citer les langages **PHP**, **JSP** ou encore **ASP**.

Il s'agit là de langages interprétés qui génèrent **côté serveur** le code HTML à fournir au client en réponse à une requête HTTP. Comme cela était auparavant le cas avec les programmes CGI. Mais à la différence de ces derniers, chacun de ces scripts ne constitue pas un programme distinct du point de vue du système d'exploitation du serveur, nécessitant son propre processus.

Du point de vue du système d'exploitation du serveur, le seul programme exécuté dans son processus est désormais l'interpréteur du langage. Chaque requête HTTP y est généralement traitée dans un fil d'exécution interne au processus (un « thread »), ce qui est moins coûteux en ressources qu'un processus à part entière.

4. Quatrième ère : l'avènement d'AJAX

En 2005, Google (déjà en tête sur le marché des moteurs de recherche) frappe un énorme coup avec son service Google Maps. Ce logiciel frappe les esprits de tous les développeurs web du monde en redéfinissant ce qu'il est possible de faire sous forme d'application web.

L'année suivante, la firme ré-itère avec Google Docs, une application web qui vient chatouiller le monopole de Microsoft Office en matière de logiciels de bureautique.

Ces logiciels ont en commun de proposer des interfaces d'utilisation qui s'affranchissent totalement des limites que l'on supposait jusque là être celles des applications web : essentiellement de simples formulaires de saisie et, si on consentait à vraiment faire des efforts, quelques graphiques générés sous forme d'images JPEG. Ils ont également en commun de faire un usage intensif de la technologie AJAX.

La technologie AJAX, pour « Asynchronous Javascript And Xml », repose sur la possibilité pour du code Javascript d'effectuer une requête HTTP et d'en recevoir le résultat pour le traiter lui-même et sans que cela ne remplace le document en cours. Son principe d'utilisation consiste à ce que le script contacte le serveur pour obtenir des données (initialement dans un format XML, mais désormais le format JSON est le plus utilisé) et à les traiter pour modifier le DOM du document en cours.

Il s'agit donc, en réponse à une action de l'utilisateur, de remplacer de façon pertinente une partie du document affiché tout en évitant son rechargement total. Cela permet donc de fluidifier le comportement de l'application et son ressenti par l'utilisateur.

On notera cependant que pour la mise en œuvre de cette technologie, l'emploi côté client du langage Javascript devient incontournable, en totale contradiction avec la philosophie des technologies de serveurs de pages. Cette période voit la naissance de développements hybrides d'applications reposant pour partie sur des technologies ancrées du côté du serveur (les technologies de serveurs de pages) et pour partie sur des technologies ancrées du côté du client (AJAX). Des technologies comme JSF ou ASP.NET tentent, avec des résultats plus ou moins heureux, d'en faire une synthèse. Mais du côté des développeurs web, l'ont sent bien que l'état de l'art du métier est en train de se chercher.

Mais parallèlement à cela, la guerre des navigateurs se termine, du moins pour ce qui est de sa forme la plus sauvage. Les pratiques déloyales et illégales de Microsoft, en dépit des condamnations judiciaires qu'elles suscitent contre la firme, viennent à bout de la société Netscape (dont le code, libéré dans un dernier acte désespéré, renaîtra plus tard sous la forme du navigateur Firefox).

Désormais, la concurrence à Internet Explorer, avec Google Chrome et Firefox, s'organise dans des conditions plus respectueuses des intérêts des utilisateurs qu'à l'ère de Netscape. Elle s'accompagne dorénavant d'un effort de normalisation du langage Javascript, à travers la norme ECMAscript.

5. Cinquième ère : HTML5 et le grand retour de l'architecture client / serveur

Tous ces efforts de normalisation aboutissent, à partir de 2014, à la norme HTML5.

Il s'agit d'une nouvelle version majeure du langage, dont la plupart des nouveautés sont destinées à faciliter le développement d'applications web (là où il s'agissait auparavant avant tout d'un langage de définition de documents hypertexte avec seulement quelques bricoles permettant la réalisation d'applications).

Cette norme ne concerne en fait pas le seul langage de balises HTML. Elle intègre également les dernières évolutions officielles de Javascript (ECMAscript) et le langage de mise en forme CSS3.

Le « nouveau » modèle de développement qui constitue l'état de l'art accompagnant cette nouvelle mouture ressemble en fait furieusement à un grand retour du modèle client serveur des années 1980. Il s'agit d'écrire un client en HTML, Javascript et CSS qui communique en AJAX (ou plutôt en AJAJ, le format d'échange privilégié étant désormais JSON) avec des services web.

Ces services web, à la différence de ce qui se faisait avec les serveurs de pages, ne réponde pas par du code HTML correspondant à des écrans ou des portions d'écrans à afficher. Au lieu de cela, les réponse sont constituées de données brutes (généralement en JSON, donc). Idéalement, les interfaces d'utilisation de ces services peuvent être décrites avec la norme Swagger.

Si le HTML, le Javascript et le CSS s'imposent côté client, les services côté serveur peuvent être implémentés dans toutes une variété de langages différents (souvent C# ou Java ; personnellement je recommande évidemment Haskell, car c'est un domaine dans lequel il brille particulièrement).

On refait donc en HTML, Javascript et CSS sur du HTTP ce que dans les années 1980 l'on faisait typiquement en C++ sur des sockets TCP/IP. Mais, en termes de compétences techniques, cela se révèle beaucoup plus abordable aujourd'hui que cela l'était à l'époque.

6. La prochaine ère ?

Notre actuelle ère du développement web, la cinquième dans l'énumération du présent article, est marquée par un paradoxe : les concepts n'ont jamais été aussi matures, mais les technologies disponibles pour les mettre en œuvre sont extrêmement diverses et se succèdent à un tel rythme que cela pose un problème de pérennité du code qui les emploie. Et encore plus paradoxalement, c'est le côté client, avec son HTML5 unique et incontournable, qui est plus touché par ce phénomène que le côté serveur avec la multitude de langages qu'il est possible d'y employer.

Les « frameworks » constituant des surcouches à HTML5 pour construire des interfaces utilisateurs pullulent. Certains ont vécu environ deux ans avant d'être supplantés par d'autres, dont on ne sait pas combien de temps ils tiendront à leur tour.

Peut être que la prochaine ère découlera d'un assainissement de cette situation. Mais ce ne serait là pas nécessairement une révolution suffisante au niveau des concepts pour que l'on puisse considérer que cela constituerait un changement d'ère.

Une autre possibilité, plus radicale, serait une atténuation de la frontière qui sépare le client du serveur.

Il est classique de faire des analogies entre le domaine de l'architecture des logiciels et celui de la cuisine italienne. Si l'on qualifie votre logiciel de plat de spaghettis, c'est peu flatteur, car cela signifie que tout y est emmêlé et en désordre. Ordonnez un peu mieux votre logiciels en couches dont chacune a un rôle bien défini et ne communique qu'avec les couches qui lui sont immédiatement connexes et l'on comparera cette fois ci votre logiciel à un plat de lasagnes. Enfin, il y a toujours l'idée que les logiciels pourraient être organisés comme des raviolis, c'est à dire avec des composants autonomes et complets pour la fonction qu'ils ont à remplir et qui collaborent les uns avec les autres sur un pied d'égalité.

Le modèle actuel du client et du serveur envisagé comme des couches séparées relève plutôt du plat de lasagnes (indépendamment du fait que chacune de ces deux couches peut elle même adopter une architecture organisée en (sous) couches). On pourrait imaginer un avenir dans lequel le web ressemblerait plus à un plat de raviolis, avec des navigateurs qui se comporteraient également comme des serveurs et s’échangeraient des données de pair à pair.

La défiance vis à vis d’un Internet trop centralisé (Facebook, Google, Amazon et autres siphons à données personnelles) et la technologie de la blockchain pourraient y contribuer.