État des lieux du développement web

Depuis sa création, dans les années 1990, jusqu'à nos jours, le Web a énormément évolué.

Initialement conçu pour partager des documents statiques sur Internet, on peut aujourd'hui utiliser de vraies applications directement depuis notre navigateur. Utilisation sans connexion, notifications *push*, interactions avec le *device* (accès à l'orientation, accès à la caméra), etc., la majorité des fonctionnalités disponibles historiquement sur les applications clients lourds le sont maintenant en Web.

Pour cela, les moteurs **JavaScript** des navigateurs et les API **HTML** ont dû évoluer. Les navigateurs proposent maintenant des moteurs extrêmement optimisés permettant d'exécuter du code JavaScript de manière rapide même sur des *devices* bas de gamme. Les API HTML sont devenues plus complètes, notamment avec l'arrivée de **HTML 5** qui a apporté énormément (stockage dans le navigateur, animations **CSS 3**, etc.). Ceci permet de proposer beaucoup plus de fonctionnalités.

Les outils ont également changé. JavaScript étant devenu un langage prédominant, alors qu'avant il n'était utilisé qu'avec parcimonie, de nombreux frameworks et IDE ont vu le jour afin de proposer une meilleure expérience de développement. **Node.js**, créé en 2009, permet d'exécuter du code JavaScript en dehors d'un navigateur et est vite devenu un outil indispensable dans le développement *front*. Des frameworks comme **React**, **Angular**, **Vue.js** et bien d'autres ont été créés afin de faciliter la création d'applications web.

Angular

Angular est un framework permettant de créer des applications clientes web. Plus qu'un framework, Angular se présente même comme une plateforme de développement permettant de créer des applications web et mobiles.

Aujourd'hui, la version d'Angular est la 4. Les propos de ce livre se basent sur cette version, et quelques changements pourraient survenir sur les versions suivantes.

Ce framework s'appuie sur plusieurs principes présentés dans les sections suivantes.

1. Organisation par composants

L'organisation d'une application Angular se fait par composants. Un composant correspond à un élément réutilisable, indépendant et responsable d'une seule action métier.

De cette manière, une application sera faite de l'assemblage d'un ensemble de composants. Cela permet une meilleure organisation, une meilleure testabilité et donc une meilleure maintenabilité.

2. TypeScript

Bien qu'il soit possible de développer son application en pur JavaScript, Angular est écrit en **TypeScript** et il est conseillé de faire son développement d'applications également en TypeScript.

Ce langage, développé par **Microsoft**, *open source* et se transcompilant en JavaScript, apporte du typage et une phase de compilation au développement front. Ces deux éléments assurent une meilleure stabilité du code puisqu'il est vérifié à la compilation.

3. Les spécifications ES6

EcmaScript 6, la spécification JavaScript en cours de support par les navigateurs, apporte un ensemble d'éléments : mot-clé `let`, template de chaîne de caractères, paramètres par défaut, gestion des modules, etc.

Angular a basé son architecture sur ces spécifications. C'est notamment le cas pour l'utilisation des modules ES6, à ne pas confondre avec les modules Angular, qui permettent de déclarer des éléments puis de les importer.

4. DOM Virtuel

L'une des tâches les plus impactantes en termes de performances dans le développement *front* est la manipulation du **DOM** (DOM signifie *Document Object Model* et représente la structure d'une page HTML sous la forme d'un arbre). Chaque opération visant à interagir, que ce soit en lecture ou en écriture, avec les nœuds HTML est coûteuse et doit être minimisée au maximum.

La librairie React a introduit une nouvelle façon de gérer cette problématique : le **DOM Virtuel**. Angular utilise ce mécanisme.

L'idée du DOM Virtuel est d'avoir une représentation en mémoire du DOM. Chaque modification, au lieu d'être effectuée sur le DOM physique, sera effectuée sur le DOM Virtuel, donc de manière beaucoup plus performante. Le framework Angular s'occupera ensuite de synchroniser les modifications effectuées sur le DOM Virtuel vers le DOM physique.

Les modifications peuvent ainsi être appliquées de manière différentielle, en une seule fois. Le gain de performance est majeur.

5. Rendu côté serveur possible

Depuis l'arrivée de Node.js, il est possible d'exécuter du code JavaScript côté serveur, c'est-à-dire en dehors du navigateur. Grâce à NodeJS, combiné à la fonctionnalité de DOM Virtuel, Angular peut être exécuté côté serveur afin de renvoyer une vue déjà pré-calculée.

Dans une architecture sans rendu côté serveur, le navigateur va télécharger tous les *assets* de l'application (fichiers HTML, CSS, JavaScript, images, etc.). Ensuite, il va exécuter le JavaScript et l'application va pouvoir commencer son rendu. Ce séquençage pose deux problèmes. Le premier est qu'entre le moment où le navigateur commence à télécharger les *assets* et le moment où l'application a terminé son rendu, l'utilisateur n'aura rien d'affiché (ou alors uniquement un message statique). Le second problème vient du **SEO** (ou référencement). Les *crawler* (robots d'indexation) des moteurs de recherche vont analyser une page vide puisqu'au moment de l'analyse de la page, l'application n'aura pas eu le temps d'effectuer son rendu (il existe cependant des solutions alternatives).

Dans une architecture avec rendu côté serveur, le navigateur va télécharger de manière classique tous les *assets* de l'application. Il va pouvoir afficher la page pré-calculée côté serveur et exécuter le JavaScript. L'application Angular va pouvoir se rendre dans le DOM Virtuel et un différentiel pourra être fait pour mettre à jour le DOM physique si nécessaire. Le temps d'affichage est très nettement raccourci et les *crawler* peuvent facilement indexer le contenu.

AngularJS vs Angular

AngularJS est un framework web créé en 2009.

À l'époque, le développement web était totalement différent de ce qu'on connaît de nos jours. Les API HTML 5 étaient en cours de spécifications (terminées en 2014), Node.js venait à peine de sortir une première *release* et n'était pas aussi populaire et implanté qu'aujourd'hui, etc. Niveau architecture, les sites web étaient très majoritairement rendus uniquement côté serveur. Côté client, pour dynamiser les pages, des librairies comme **jQuery** étaient utilisées.

AngularJS a révolutionné le développement de site web. Ce framework a apporté toutes les briques techniques nécessaires au développement d'applications s'exécutant totalement côté client : *Templating*, *binding*, navigation sans rafraîchissement du navigateur, etc. Mais ce framework n'est plus optimisé pour le Web d'aujourd'hui et ne tire pas parti des différentes possibilités apportées ces dernières années.

S'est alors posée la question des nouvelles versions d'AngularJS. L'équipe de **Google** a décidé de repartir d'une feuille blanche afin de développer un nouveau framework adapté aux nouvelles spécificités du Web, sans avoir à payer le prix de la maintenance et de la rétrocompatibilité.

Migrer une application AngularJS vers Angular

Initialement, l'équipe de Google n'avait pas prévu de fournir un chemin de migration d'une application AngularJS vers une application Angular.

Face aux réactions négatives de la communauté utilisant AngularJS, cette décision a été revue et une migration est désormais possible, bien qu'elle reste assez fastidieuse.

Pour cela, Google a continué à faire évoluer AngularJS pour utiliser les principes de composants. De cette manière, les dernières versions d'AngularJS proposent une syntaxe très proche de ce qu'on peut avoir sur Angular, ce qui facilite la migration.

Conscient qu'il n'est pas possible de migrer la totalité d'une application AngularJS, à moins qu'elle ne soit assez petite, un module `UpgradeModule` a été créé permettant de faire cohabiter au sein d'une même application du code Angular avec du code AngularJS.

La documentation d'Angular propose un guide très détaillé sur le processus de migration décrit précédemment. Afin de ne pas paraphraser inutilement, vous pouvez vous référer à ce guide très complet pour plus de détails : https://angular.io/docs/ts/latest/guide/upgrade.html