Google, Javascript et Ajax : limites techniques et réalité actuelle (1ère partie)

Par Philippe Yonnet

Domaine :	Recherche	**Référencement**
Niveau :	Pour tous	**Avancé**

De plus en plus de sites web sont réalisés en utilisant des technologies comme Javascript ou l'Ajax. Pour les explorer, les moteurs de recherche doivent s'adapter quotidiennement à cette nouvelle donne. Mais arrivent-ils réellement à crawler et analyser aujourd'hui tous les sites, parfois complexes, qu'ils explorent ? Cet article en deux parties a pour objectif de faire un point impartial sur la réalité actuelle et les limites, obligatoires, que l'on trouve actuellement dans les possibilités des robots d'aujourd'hui. Ce mois-ci, c'est l'Ajax qui est décrit, avant de parler d'autres technologies comme Angular JS, Backbone JS ou Ember JS le mois prochain.

Depuis des mois, Google communique régulièrement sur ses progrès dans la prise en compte du code Javascript dans les pages web qu'il explore et indexe. Officiellement, Googlebot peut exécuter du Javascript. Google peut aussi générer une rendition complète de la page et donc savoir quelle apparence a réellement votre page dans un navigateur "normal", une fois les css et le Javascript chargés et exécutés. Et Google a même expliqué qu'ils avaient fait des progrès importants dans la compréhension des contenus générés en Ajax.

Cette évolution est logique, car les développeurs de sites web utilisent de plus en plus le Javascript pour générer le contenu (et le code HTML) des pages web. Certains sites sont même faits entièrement en Ajax, ou presque entièrement avec du code HTML généré en Javascript. Mais est-ce bien raisonnable ? Les progrès de Google dans le traitement des contenus générés en Javascript sont-ils suffisants aujourd'hui pour que l'on puisse réellement fabriquer des sites entièrement générés ainsi, sans impact sur le référencement ?

Google est forcé de s'adapter à ces changements rapides. Le 14 octobre dernier, Kazuchi Nagayama, *Search Quality Analyst* chez Google annonçait solennellement sur le blog pour les webmasters de Google que la firme de Mountain View considérait comme obsolète la méthode des "hashbangs" pour rendre les contenus affichés en Ajax crawlables.

Pourquoi un tel revirement, après avoir recommandé cette solution pendant six longues années ? Qu'est-ce qui a changé ? Peut-on toujours rendre l'Ajax crawlable ? Les contenus en Ajax sont ils bien indexés ? C'est ce que nous allons nous efforcer d'apprendre dans la première partie de cet article.

Dans la seconde partie, qui sera publiée dans le prochain numéro, nous nous intéresserons aux problèmes posés par l'emploi sur certains sites de technologies comme Angular JS, Backbone JS pour fabriquer des sites web. Nous verrons que l'on peut rendre ce type de page parfaitement crawlable, mais nous verrons aussi à quel point il est risqué pour le référencement d'utiliser ces technologies… Google et Googlebot ont leurs limites, qu'il faut connaître, et flirter avec ces limites a un prix. Mais commençons par faire le point sur l'Ajax et sa crawlabilité...

L'Ajax, c'est quoi ?

L'Ajax (*Asynchronous Javascript and XML*), est une technique qui permet de créer des applications web et des pages de sites web interactives, et qui rend la mise à jour du contenu d'une partie d'une page possible sans la recharger. Un site web "normal" comporte des pages qui, une fois chargées dans le navigateur, restent statiques : leur contenu ne change pas. Pour charger un nouveau contenu, il faut appeler une nouvelle URL et charger une nouvelle page dans le navigateur.

En Ajax, un programme en Javascript (le JA de Ajax) va aller interroger le serveur web *via* la méthode Httprequest (le *asynchronous* de Ajax) et récupérer en XML (le X de Ajax) un nouveau contenu délivré par le serveur web.

Le HTML de la page est modifié ensuite dynamiquement en Javascript pour afficher ce nouveau contenu, sans qu'il soit nécessaire de recharger une nouvelle page dans le navigateur.

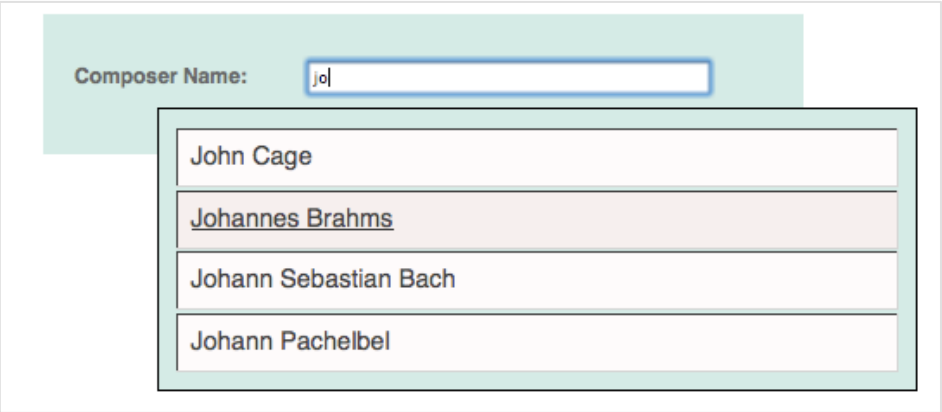

Fig.1. Un exemple de fonctionnalité programmée en AJAX : l'autocomplétion dans les formulaires.

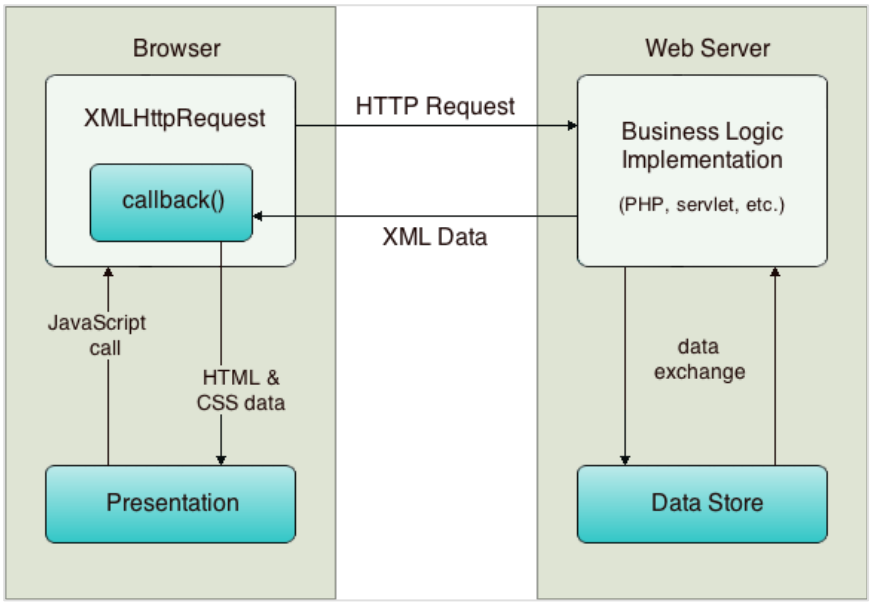

Fig.2. Et la technologie AJAX sous le capot…

L'Ajax, un challenge pour les bots des moteurs de recherche

L'emploi de l'Ajax crée une succession d'"états" pour une page web. L'état par défaut correspond au contenu chargé par le navigateur lors du premier affichage de la page, avant toute intéraction avec l'utilisateur, ou exécution d'une requête httprequest en Javascript. Chaque modification d'une partie, ou de la totalité du code HTML et/ou du contenu affiché d'une page web en Javascript constitue un nouvel état de la page.

Problème : à l'origine, un bot de moteur de recherche comme Googlebot ou Bingbot savait télécharger le code d'une page web, et l'indexait directement, sans exécuter le code Javascript de la page. Bref, un bot classique ne voit que le contenu de la page présent dans l'état par défaut. Tout contenu chargé dynamiquement ensuite est invisible pour le moteur, les contenus chargés en Ajax ont donc dans un premier temps été superbement ignorés dans le processus d'exploration de Google, Bing et consorts…

Si Google sait exécuter le Javascript, peut-il crawler et indexer de l'Ajax ?

Sur le papier, Googlebot peut effectivement aujourd'hui exécuter du Javascript, et donc charger des contenus appelés en Ajax comme n'importe quel utilisateur. Mais Googlebot n'est pas un utilisateur "normal", et dans les cas d'interface complexes, ne peut pas forcément deviner comment interagir avec la page pour charger les contenus intéressants : faut il cliquer à un endroit ? Faut-il cocher une case ? Attendre une mise à jour automatique du contenu après un certain délai ? Bref, Google peut le faire pour des cas simples, mais pas dans tous les cas.

De plus, avec une interface riche en Ajax, la combinatoire de toutes les interactions possibles rend l'exploration systématique de tous les contenus générés en Ajax extrêmement coûteuse en temps machine et en ressources. Qui plus est, il existe un risque de tomber dans un « spider trap » quand on crawle de l'Ajax .

Ce problème est encore amplifié par le caractère dynamique des contenus : une stratégie de mise en cache est impossible.

Algorithm III.1 Breadth-First AJAX Crawling Algorithm

```
1: Function init(url)
2: dom = readDocument(url)
3: dom.executeFunction(body.onLoad) {AJAX Specific}
4: appModel.add(dom) {Add first state to the App. Model}
5: crawl(dom)
6: end Function
7: Function crawl(State s)
8: for all Event e ∈ s do
9:    dom.executeFunction(e.function)
10:   if dom.hasChanged() then
11:      State newState = new State(dom)
12:      if appModel.contains(newState) then
13:         newState = appMode.get(newState)
14:      end if
15:      Transition t = new Transition(e.source, e.trigger, e.action*,
         e.modif*)
16:      appModel.add(t, s, newState)
17:      appModel.rollback(t)
18:   end if
19: end for
20: for all Transition t ∈ (s, s1) do
21:   Crawl s1 {Breadth-first traversal of reachable states}
22: end for
23: end Function
```

Fig.3. Code simplifié d'un crawler capable d'explorer de l'Ajax : à chaque changement apporté au DOM dans la page, un nouvel état de la page est détecté, et la transition correspondante est ajoutée au modèle de l'application Ajax.

Indexer les fragments en AJAX sans générer de quasi doublons

Comment indexer les contenus liés aux états successifs de la page ensuite ? Quelle importance donner à ces "fragments" de contenu ? En Ajax, on peut modifier le contenu entier d'une page, ou simplement un nombre dans un tableau ! Chaque état doit-il être indexé comme une page à part entière ? Sûrement pas ! Ou alors on indexe des paquets de quasi doublons ? Comme un contenu supplémentaire de la page ? Pourquoi pas, mais l'architecture du logiciel d'indexation d'un moteur grand public n'est pas faite pour fonctionner comme cela avec n'importe quel type et taille de contenu.

Ce problème d'incompatibilité avec le fonctionnement de l'algorithme d'un moteur de recherche est très profond. Ce dernier est entièrement bâti autour de la relation URL-contenu de la page. L'Ajax détruit cette relation, puisqu'il autorise la création de ce qu'on appelle les SPA (les *Single Page Applications*) : c'est-à-dire un site web dont toutes les "pages", tous les

contenus, sont chargés depuis une seule URL… Or l'algorithme associe de nombreux signaux aux URL…

C'est pourquoi les moteurs de recherche aujourd'hui ont tous adopté une approche commune. Ils n'explorent pas et n'indexent pas l'Ajax par défaut :
• Parce que c'est techniquement compliqué et hasardeux ;
• Parce que cela consomme des ressources considérables pour un gain faible le plus souvent ;
• Et de plus, parce que les contenus qui seraient explorés ainsi ne peuvent pas être traités correctement avec une architecture d'indexeur et de requêteur classique, ni classés de manière pertinente par l'algorithme de pertinence.

Bref, le casse-tête est gigantesque.

Par contre, les moteurs de recherche se sont dit que si le webmaster les aidait à identifier :
• La bonne méthode pour explorer les fragments de contenus accessibles en Ajax ;
• Les fragments intéressants à explorer et les fragments à délaisser ;
• Les notions de pages web, et d'urls associées,
Alors, l'exploration et l'indexation des contenus en Ajax redevenait possible.

Première tentative pour rendre l'Ajax crawlable : la méthode du "hashbang" (déclarée obsolète par Google)

La première méthode proposée par Google en octobre 2009 pour rendre l'Ajax crawlable était la méthode surnommée « hashbang ». Elle consistait à utiliser la séquence #! dans l'url (épelée « hash bang » en anglais américain) pour signaler au moteur l'existence d'une url alternative que le bot pouvait appeler pour télécharger une version statique du contenu.

En pratique : si l'URL Ajax de départ est :
http://example.com/dictionary.html#AJAX
et si elle est recodée avec cette syntaxe :
http://example.com/dictionary.html#!AJAX
alors le moteur va aller crawler l'url :
http://example.com/dictionary.html?_escaped_fragment_=AJAX
mais qui sera retransformée en : http://example.com/dictionary.html#!AJAX
Dans la page de résultats de Google.

L'une des difficultés posée par cette méthode était donc de générer à la fois la page « Ajax » et la version statique du fragment. Certaines solutions techniques existent pour faciliter le travail de « double » génération du code (la version pour l'utilisateur, et la version pour le crawler), mais ce n'était pas très élégant.

Google a donc finalement annoncé en octobre dernier que cette méthode des hashbangs était dorénavant considérée comme obsolète. Il existe donc un risque que Google cesse de la supporter dans les mois/années qui viennent.

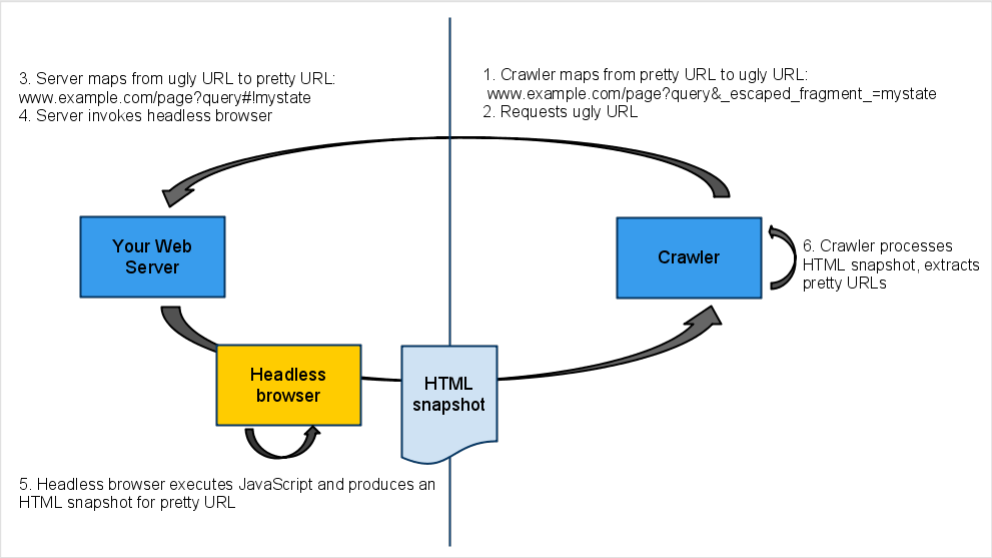

Fig.4. L'une des solutions proposées par Google est la suivante : l'appel de la syntaxe crawlable « ?_escaped_fragment_=[QUERY] » déclenche l'appel par le serveur web d'un « navigateur headless » (PhantomJS ou équivalent) qui va déclencher la génération du contenu HTML crawlable, retourné au crawler…
Une solution simple sur le papier, mais très technique !

La recommandation actuelle : la méthode du Pushstate()

Avec le temps, Google s'est rendu compte que les méthodes exploitant les possibilités des navigateurs récents, et sa nouvelle capacité à exécuter du Javascript étaient plus simples à déployer et beaucoup plus élégantes. Ces méthodes reposent sur l'exploitation de la méthode javascript Pushstate(), pour manipuler l'API qui gère l'historique de navigation.

L'avantage de cette solution est qu'il redevient possible de travailler avec des URL Ajax sans « ancres » derrière des caractères # pour appeler un contenu.

Notre URL Ajax de tout à l'heure :
http://example.com/dictionary.html#idcontenu-2
Peut devenir une URL classique :
http://example.com/dictionary.html?idcontenu=2
et continuer à appeler un comportement de mise à jour d'une partie du contenu gérée en AJAX !
Pour comprendre le fonctionnement, le mieux est d'aller sur ce site de démonstration :
http://html5.gingerhost.com

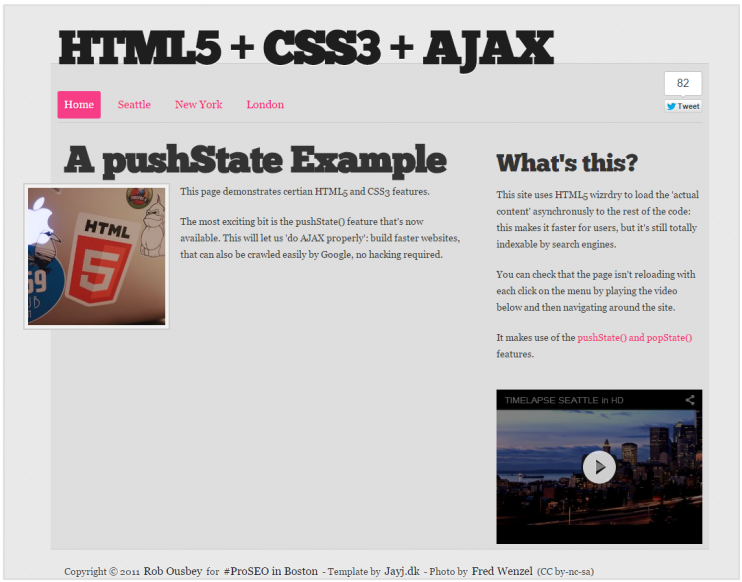

Fig.5. La page d'accueil du site exemple

Si l'on clique sur « Seattle », la page ne se recharge pas : une nouvelle image et un nouveau contenu se chargent en Ajax.

Par contre, et c'est plus surprenant : en cliquant sur Seattle, le code Javascript change aussi l'URL de la barre du navigateur (c'est l'effet de la commande Pushstate !)

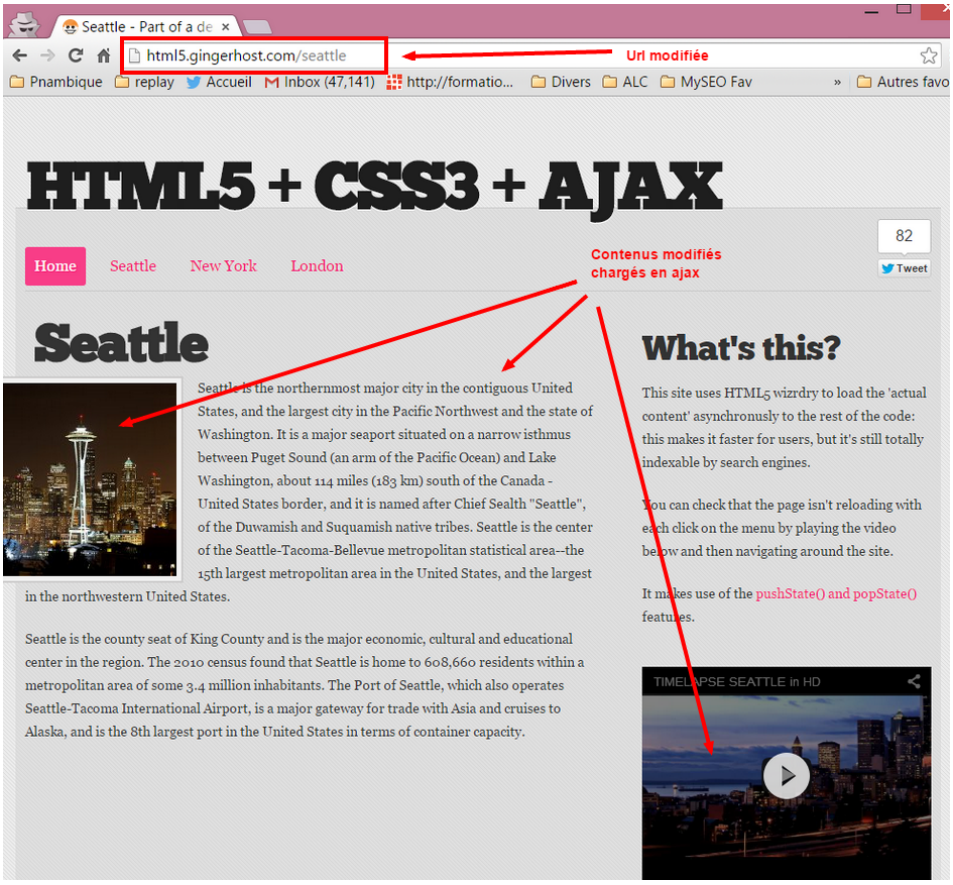

Fig.6. Le rendu après un clic.

Un coup d'œil sur le code source de la page vous permettra de voir comment pushstate() est implémenté dans ce cas ultrasimplifié.
La présence d'URL normales dans des pages dont le contenu est chargé en Ajax permet évidemment d'assurer la crawlabilité des contenus : les URL appelant ces comportements sont des URL classiques. Par exemple dans le code des pages ci-dessus :

```
<ul id="menu" class="clearfix">
        <li class="current"><a href="/">Home</a></li>
        <li><a href="/seattle">Seattle</a></li>
        <li><a href="/new-york">New York</a></li>
        <li><a href="/london">London</a></li>
</ul>
```

Il faut par contre parfaitement veiller à ce que les URL utilisées dans pushstate() ne génèrent pas de quasi doublons… Ces URL doivent appeler des contenus intéressants pour les crawlers des moteurs de recherche…

Cette méthode est parfaitement gérée par Google, Bing et Yandex. Mais pas encore par les autres moteurs.

On peut trouver un exemple d'implémentation pour rendre un contenu en Ajax « compatible SEO » dans un article de John Mueller de Google (qui date de février 2014), à propos des « infinite scroll pages » (pages à défilement infini) :
http://googlewebmastercentral.blogspot.fr/2014/02/infinite-scroll-search-friendly.html

Quel est l'impact de l'Ajax pour le référencement ?

Quels que soient les progrès faits par Google dans la gestion des contenus en Javascript, et même si la crawlabilité est assurée par un usage pertinent de la méthode pushstate(), charger des contenus intéressants pour le référencement en Ajax reste, quoi qu'il arrive, déconseillé pour avoir un référencement optimal.

Le premier risque est de voir ses contenus ignorés par Google, ou sous-pondérés, parce qu'il estime qu'un contenu qui n'est pas dans l'état par défaut de la page n'est probablement pas important. On peut néanmoins minimiser fortement ce risque en conservant autant que possible une associationURL-contenu pertinente.

Le second risque, et certainement le plus élevé, est de voir les moteurs de recherche buter sur des difficultés techniques et ne pas réussir à explorer correctement tout ou partie de vos contenus en Ajax.

Bien coder en respectant les règles pour obtenir un code SEO friendly est nécessaire, mais pas toujours suffisant.

Ce risque est partagé par toutes les implémentations de pages web qui génèrent une grande partie des contenus HTML en Javascript.

Car outre l'Ajax, d'autres méthodes de développement de sites web exploitant des frameworks Javascripts (Angular.JS, Backbone.js, Ember.js) sont de plus en plus employés. Et crawler ces sites web posent des problèmes similaires. Ces techniques sont elles compatibles avec un bon SEO ? Ce sera le sujet de notre prochain article.

Liens utiles

Posts du blog webmasters de Google
Présentation de la méthode des hash bangs (obsolète) :
http://googlewebmastercentral.blogspot.fr/2009/10/proposal-for-making-Ajax-crawlable.html

Annonce des progrès faits par Google dans la prise en compte du Javascript :
http://googlewebmastercentral.blogspot.fr/2014/05/understanding-web-pages-better.html

Google annonce qu'il ne promeut plus la méthode des hashbangs :
http://googlewebmastercentral.blogspot.fr/2015/10/deprecating-our-Ajax-crawling-scheme.html

Méthode des hashbangs (obsolète). Le lien vers l'ancienne spécification, toujours supportée par Google et d'autres moteurs, mais qui est considérée comme obsolète. L'utilisation de cette méthode sur de nouvelles pages/implémentations est déconseillée.
https://developers.google.com/webmasters/Ajax-crawling/docs/learn-more
https://developers.google.com/webmasters/Ajax-crawling/docs/getting-started
https://developers.google.com/webmasters/Ajax-crawling/docs/specification

Méthode employant pushstate() (recommandée)
La recommandation de Bing :
https://blogs.bing.com/webmaster/2013/03/21/search-engine-optimization-best-practices-for-Ajax-urls/
Et celle de Google : une vidéo de Matt Cutts datant de mars 2013 expliquant les avantages de la méthode pushstate
https://www.youtube.com/watch?v=yiAF9VdvRPw&feature=youtu.be

Philippe YONNET , *Directeur Général de l'agence Search-Foresight, groupe My Media (http://www.search-foresight.com)*