Approche Littérale ReactJS

Sommaire

I. React repose sur la composition

Avec les derniers nés de la communauté open source, la composition est à la mode et nous allons expliquer quels sont ses avantages.

II. Qu'est ce qu'un composant et la composition ?

Un composant est une partie unitaire de l'application. Chacune de ces parties peuvent être elles-mêmes composées d'autres composants. On parle de composition. Une page web n'est alors rien d'autre qu'un arbre de composants dont les feuilles sont des balises html classiques.

III. Les composants sont isolés

Cette composition permet d'isoler des éléments de l'interface, leur comportement, leur état, leur forme et même, dans une certaine mesure, leur style.

L'isolation des composants permet d'intégrer des éléments React dans une application existante : chez Facebook le chat est écrit en React alors que le thread d'actualités ne l'est pas. On peut grâce à la composition migrer une application progressivement.

IV. Les composants sont réutilisables, testables

Un composant isolé du reste d'un layout est plus facilement **réutilisable**, il suffit de le brancher aux bons endroits, avec la bonne source de données.

Il est plus facilement **testable** et vous permet d'avoir une bonne couverture de tests unitaires. Il suffit de tester le comportement du composant en faisant abstraction de son contexte.

Avec des composants isolés, et bien réutilisés, lorsque vous détectez un bug dans un composant vous le corrigez partout où vous avez utilisé ce même composant. Lorsque vous modifiez un composant, les effets de bord sont limités à ce composant uniquement. Votre application est plus maintenable.

V. "Learn once, write everywhere"

La possibilité d'avoir un arbre de composants permet de créer une abstraction où seules les feuilles sont liées à l'implémentation : au DOM dans le cadre d'une page web.

L'équipe de React pousse le concept plus loin : la version 0.14 se détache du DOM. React n'est plus une façon de représenter des éléments du DOM mais c'est une librairie qui permet de créer des composants. **Le rendu en élément DOM n'est qu'une implémentation et peut être étendu à d'autres environnements.**

C'est comme ça qu'est né ReactNative qui propose d'utiliser React mais cette fois pour créer des interfaces natives pour IOS et Android. Apprendre à utiliser React vous facilite le portage de votre application sur différentes plateformes sans avoir à apprendre un nouveau langage pour chacune d'elles.

VI. React ouvre les portes des applications universelles

Vous avez sûrement déjà entendu parlé de SPA ? Il s'agit d'une **Single Page Application** qui **s'exécute entièrement dans le navigateur.**

VII.Les SPA ont l'inconvénient de leur avantage

Le principal **avantage** des SPA est que la **navigation est plus fluide** : finis les allers-retours avec le serveur pour générer la prochaine page visitée. **Le serveur est alors moins sollicité**, les données sont récupérées par des appels API. Les interactions utilisateur sont plus riches.

Mais la page n'est jamais générée côté serveur, uniquement dans le navigateur, ainsi **le premier chargement peut être long**. En effet, il faut attendre que le script soit téléchargé, parsé et exécuté puis que les données soient récupérées et finalement que la page soit rendue. Cette lenteur peut vite devenir critique en situation de mobilité.

Le chargement initial peut poser problème pour le **référencement** du site. En effet, lorsque la page n'est pas générée côté serveur, un appel renvoie une page blanche (cf: figure1). Pour les moteurs de recherche qui ne supportent pas l'exécution du javascript, c'est problématique. Cependant, depuis Mai 2014, Google exécute le javascript.

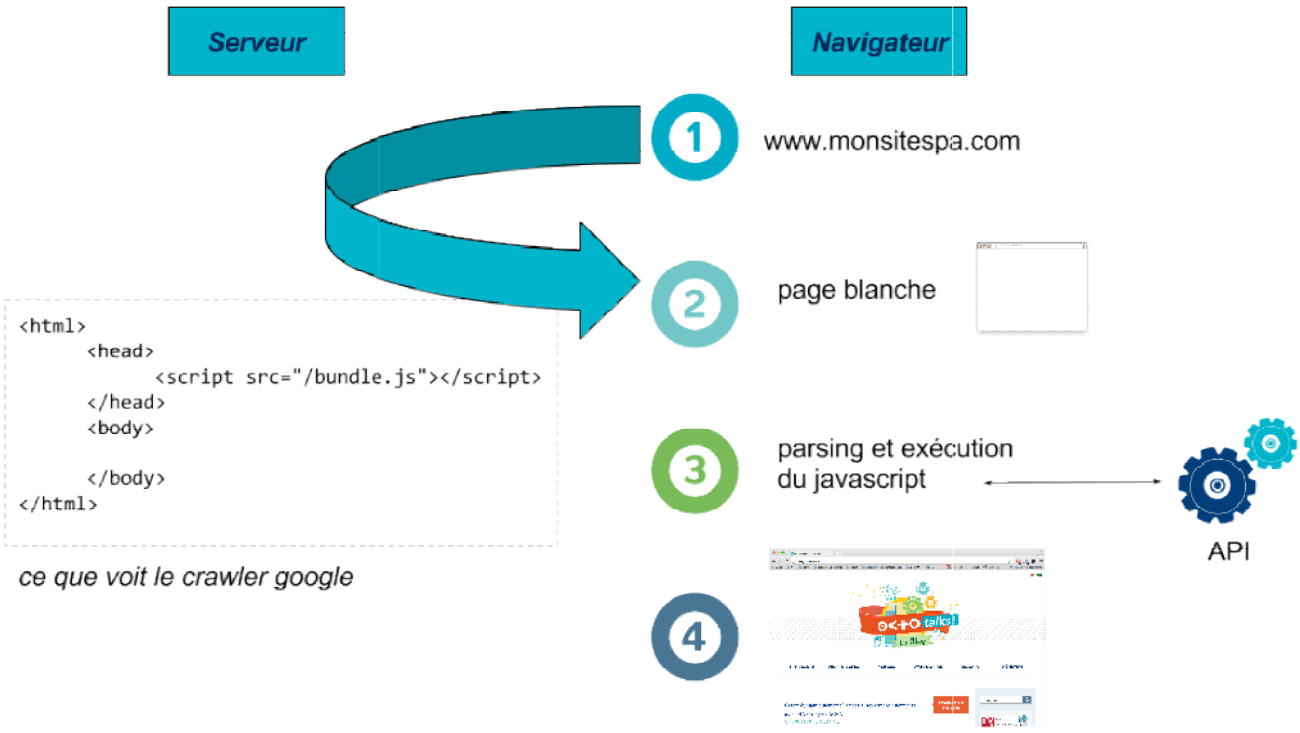


Figure 1 : Séquence de chargement une single page application dans un navigateur

L'application universelle résout les problèmes des SPA.

VIII. Qu'est ce qu'une application universelle ?

L'**application universelle** (aussi appelée "isomorphique") est née du besoin d'avoir un chargement initial qui renvoie l'application chargée avec les données. Le premier rendu de l'application est fait côté serveur, puis côté client, on retrouve une SPA traditionnelle.

Le premier rendu sera donc rapide, l'application fluide et le référencement sera plus efficace.

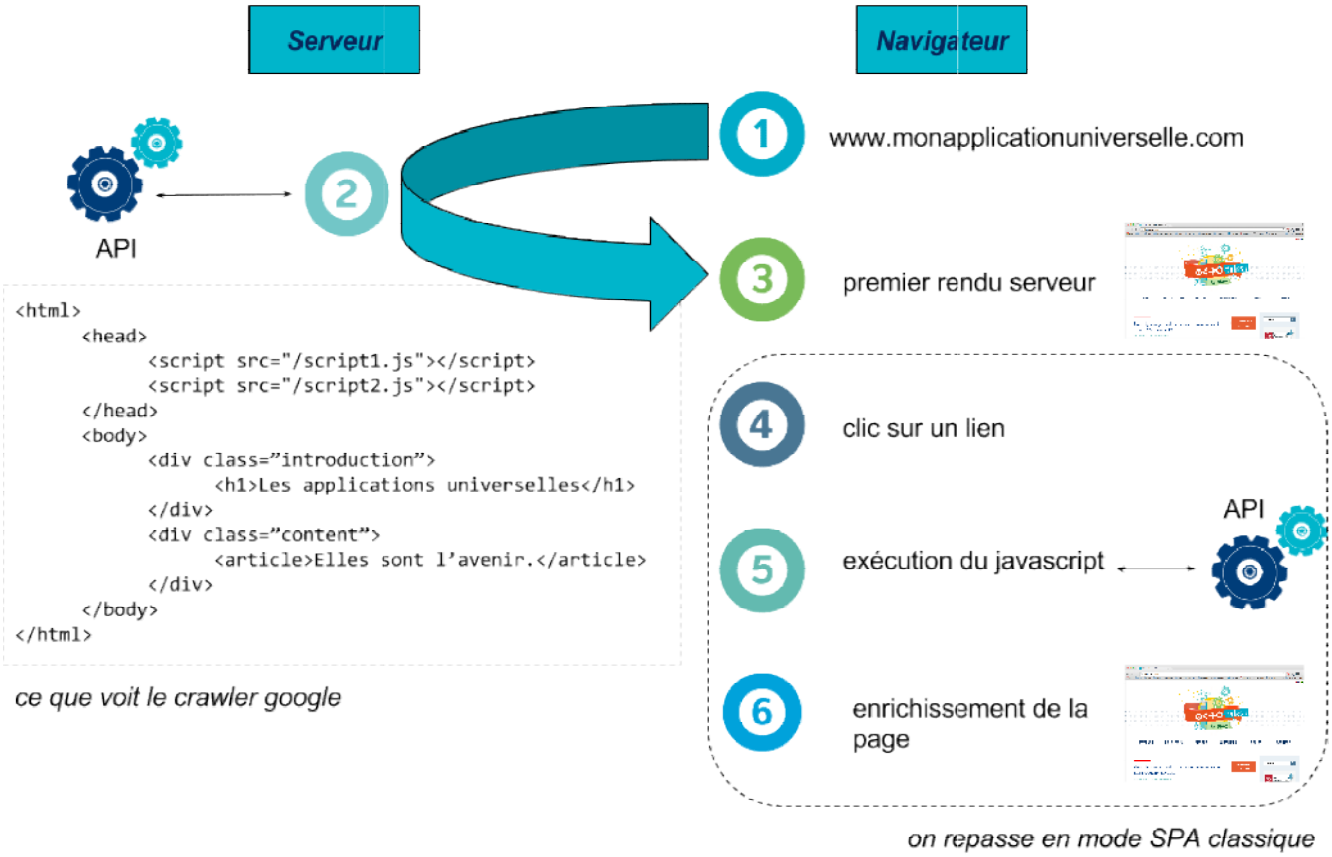


figure 2 : Séquence de chargement une application universelle dans un navigateur

Pour qu'une application universelle fonctionne, il faut que le code source exécuté sur le navigateur soit également exécutable sur le serveur. Le langage exécuté par le navigateur est le javascript, il faut donc que le serveur exécute également du javascript, comme c'est le cas avec Node.js.

IX. Le DOM virtuel à la rescousse des applications universelles

Le DOM virtuel est un concept implémenté dans React. Il s'agit d'une **représentation interne en javascript de l'arbre du DOM** qui n'est pas le "vrai" DOM du navigateur. React va pouvoir agir sur cette abstraction et non pas directement sur le DOM du navigateur.

Le DOM virtuel rend possible la génération d'une page HTML sans qu'elle soit couplée à un "vrai" DOM donc à un navigateur. Il est ainsi possible de **générer des pages coté serveur**.

De plus, le DOM virtuel permet d'**optimiser les accès au "vrai" DOM**. Lors d'un changement, React établit à partir de sa représentation virtuelle le différentiel optimal à appliquer au "vrai" DOM. Les modifications sont appliquées en une fois.

X. Quelques conseils avant de partir avec React

Il faut, lorsque l'on met en place une stack technique, avoir tous les éléments en main. Nous utilisons pour le site du ClubMed React depuis janvier, le site est en production depuis avril et voici quelques recommandations.

Au premier abord React est simple. Il met en oeuvre des concepts faciles à appréhender mais la construction d'une**application React universelle** apporte son lot de challenges.

XI. Les principes classiques d'architecture sont chamboulés

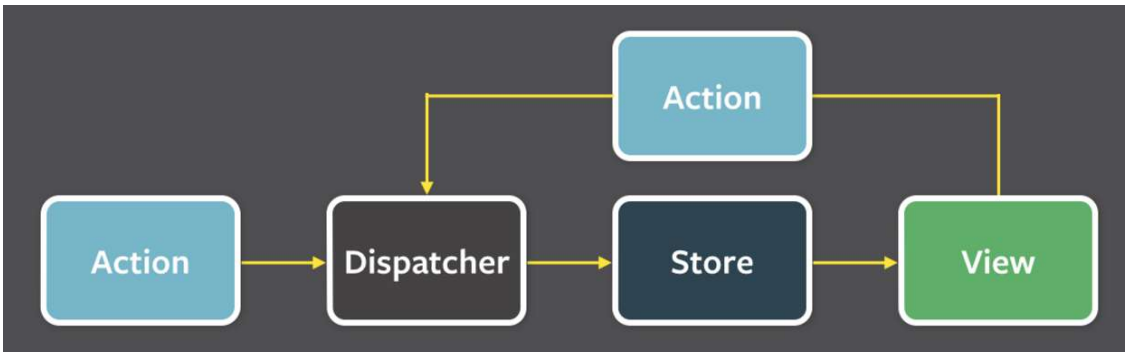
A. Une application React n'est pas un MVC, Il faut repenser sa vision de l'architecture

Contrairement à ses concurrents (AngularJS, Ember ou Backbone), React n'est pas un framework. **Il s'agit uniquement d'une librairie permettant de créer des composants**, il n'y a pas de cadre qui vous permette de structurer votre application (pas de service, de contrôleur ou de modèle). Chaque application doit définir sa propre architecture, sa gestion des données, ses règles d'implémentation et d'organisation du code.

Ceci est un avantage parce que chaque projet structure son application en fonction de ses besoins. Mais cela peut être à double tranchant parce qu'on a aussi la **liberté de faire des architectures non viables**.

Afin de répondre à ce besoin d'architecture, Facebook a publié le modèle qu'ils utilisent en interne : **Flux**, sans toutefois imposer une implémentation particulière. Il existe donc une multitude d'implémentations du modèle Flux (plus ou moins proche de la description originelle de Facebook) dans la communauté open source.

Flux est dédié à l'écriture d'application côté client. Flux repose sur le one way data flow : la vue (arbre de composants React) se branche sur les données qui sont stockées dans un store, et elle s'enregistre auprès d'un dispatcher. Les actions permettent de manipuler le store et les vues sont notifiées de la mise à jour de ce store.



source : https://facebook.github.io/flux/docs/actions-and-the-dispatcher.html

La difficulté est donc de choisir une implémentation qui :

- répond aux besoins du projet,
- est stable,
- est activement maintenue.

En parallèle, on a assisté à la naissance de **redux** : une librairie qui permet de mettre en place un Flux "allégé" dans une application – qui fonctionne du côté client et serveur. Elle repose sur le paradigme suivant : un store unique et immutable, une façon unique de gérer les états de tous les composants.

B. Les développeurs ne sont pas encore formés

Les développeurs React ne courent pas encore les rues et il y a très probablement plus de compétences AngularJS disponibles sur le marché. Il sera donc pour le moment délicat de trouver la perle rare.

Cependant, React est relativement facile à appréhender et un de ses avantages est que le développeur est libre d'utiliser le Javascript natif (sans surcouche). Ainsi un développeur expérimenté en Javascript aura une courbe d'apprentissage de React très rapide.

XII.React et les autres ?

A. Intégrer du React aux principaux frameworks SPA est possible

Vous l'aurez compris, React n'est pas un framework, il permet "simplement" de gérer les vues de votre application. Cette particularité rend possible l'intégration avec d'autres frameworks front-end dès lors qu'on isole la vue. On peut alors profiter des avantages de React combinés à la structuration du code source apportée par un framework SPA tels que Backbone, AngularJS ou Ember.

L'intérêt d'intégrer des composants React à un framework SPA classique est plus ou moins limité et complexe selon le framework auquel on veut l'intégrer.

Il serait assez aisé de l'utiliser en combinaison avec les vues Backbone.

Coupler du React et de l'AngularJS est également possible, mais la tâche est beaucoup plus complexe. Il faut mettre en commun le fonctionnement des directives et des modèles AngularJS (basé sur le two way data-binding) avec le fonctionnement très différent de React, l'intérêt est ici limité.

L'écosystème React permet aisément de se passer des frameworks traditionnels. On trouvera facilement les différentes briques qui s'intègrent avec React. On peut par exemple citer "react-router" pour le routage.

B. Intégrer des librairies manipulant le DOM avec parcimonie

Un des principes de React est de rajouter un niveau d'abstraction au dessus du DOM, l'interaction avec des librairies externes type JQuery devient plus complexe parce qu'elles deviennent des utilisatrices concurrentes du DOM.

Il faut donc veiller à **utiliser avec précaution** ce genre de librairie externe afin qu'elles interviennent au bon moment (quand le composant React est monté dans le "vrai" DOM) et de façon isolée. En règle générale on évitera l'utilisation de JQuery et on cherchera une alternative en React.

Il faut noter que le fonctionnement React permet de s'abstenir de JQuery. La vue est maintenant une vue-contrôleur avec React. En effet, le rendu (templaté avec du jsx) est intégré au javascript. **Vous n'avez plus besoin d'utiliser des sélecteurs pour manipuler le DOM**, vous le manipulez directement grâce aux propriétés et à l'état d'un composant.

XIII. React est jeune

A. La quantité de composants open sources est encore assez faible.

La communauté React est très active mais est encore jeune et en pleine expansion, il est encore **difficile de trouver des composants** variés et matures qui répondent à vos besoins : exécutables côté serveur, dépendants de la même version de React que votre projet, stables et bien maintenus.

Autant il est facile avec jQuery de trouver du code sur internet, autant c'est plus délicat avec React et vous aurez souvent besoin d'écrire vous même ces composants. Cependant cela évolue vite et on trouve quand même une bonne majorité des composants traditionnels.

B. Et l'écosystème évolue vite !

React évolue rapidement. De notre expérience il y a des montées de version faciles et d'autres plus douloureuses avec leur lot de breaking changes.

Il faut être capable de suivre le rythme, de prendre le temps de mettre à jour sa codebase sous peine de travailler dans un contexte qui peut vite devenir obsolète.

Ce besoin est d'autant plus prégnant si on utilise des librairies tierces liées à React qui elles mêmes évoluent vite.

XIV. Les applications universelles et la composition apportent leur lot de complexité

A. Trouver le bon découpage et le bon taux de réutilisation de ses composants n'est pas évident

La composition, même si elle a plein d'avantages, nécessite d'ajouter un niveau d'abstraction et d'écrire du code supplémentaire par rapport à un template html classique. Il n'est probablement pas la peine de se lancer dans React pour créer des POC ou des petites applications qui évolueront peu.

Le concept de composition est simple à comprendre. Mais on peut vite **tomber dans le sur-découpage** et se retrouver avec un nombre trop important de

composants ralentissant la réalisation de user stories. L'article Thinking in react nous a beaucoup aidé, néanmoins il n'est pas **évident de trouver la bonne granularité** pour le découpage et d'obtenir le meilleur taux de réutilisation.

Phil Karlton disait : "Il n'y a que deux choses difficiles en informatique : l'invalidation du cache et **nommer les choses**". La deuxième partie de la phrase est particulièrement vrai avec React : il faut constamment réfléchir au nom à donner à ses composants et ne pas tomber dans le travers de tout nommer : "Block", "Infos" ou encore "Details".

B. Application universelle = deux environnements

Écriture du code

Lorsque vous développez une application universelle, vous développez pour deux environnements d'exécution: votre serveur et votre navigateur.

Il faut alors :

- veiller à ce que votre code ou celui de librairies tierces fonctionne dans ces deux environnements,
- vous mettre à disposition des outils de debugging du code côté serveur,
- implémenter un système de routage qui soit exécutable des deux côtés,
- gérer le fetching des données (qui doit être synchrone coté serveur mais peut être asynchrone lors de la navigation).

Si vous travaillez sur une application interne qui n'a pas de forts besoins de référencement, si votre page peut supporter un temps de premier chargement un peu plus long, il n'est pas nécessaire de partir sur du React, une SPA type AngularJS ou Ember remplira parfaitement la tâche.

Build du code

On l'a vu : le code d'une application universelle est exécuté côté serveur et côté navigateur, il faut donc que vous ayez la même spécification de javascript des deux côtés.

Or le javascript Node.js et le javascript navigateur ont chacun leurs spécificités. Par exemple le "document" du navigateur n'existe pas en Node.js. Ou le pattern de module avec require est absent dans le navigateur (mais bientôt intégré nativement dans ES2015).

Pour harmoniser ces deux environnements il faut ajouter une étape de **build** pour rendre le code compatible navigateur. On peut par exemple utiliser à cet effet **Browserify** ou **Webpack**.

On l'a vu, React utilise du javascript pur. Pouvoir utiliser la dernière version de ce langage (**ES2015**) est un atout considérable. Évidemment aujourd'hui tous les navigateurs ne supportent pas encore la totalité des nouveautés de javascript. Nous vous conseillons d'utiliser **Babel**, il s'agit d'un transpileur javascript qui va compiler votre code Javascript ES2015 en code compatible tout navigateur (ES5).

XV. Pour finir

React a apporté son lot de nouveautés dans le monde en évolution rapide du développement front-end. Il définit un pattern élaboré de **composition** permettant de construire proprement des interfaces utilisateurs de plus en plus complexes. Il donne la possibilité de faire un **premier rendu coté serveur** (pour un bon référencement de votre application). Ce premier rendu serveur est un atout considérable lorsque l'on

est en **situation de mobilité** pour avoir un **premier affichage rapide tout en conservant les avantages des Single Page Applications.**

Cette bibliothèque répondait à nos besoins mais nous nous sommes confronté à plusieurs problèmes : comment trouver le bon découpage de nos composants ? Comment architecturer notre application pour la rendre maintenable ? Comment écrire une application double environnement (application universelle) ? Comment rester à jour dans un environnement en constante évolution?

La communauté open source évoluant rapidement, elle s'inspire des innovations et des succès des uns pour faire évoluer et améliorer les autres. Les deux "concurrents" de React se mettent à la page. La version 2 d'Ember permet de faire du rendu coté serveur grâce à son système de fastboot. Angular suit la même ligne et proposera dans sa deuxième version un système de rendu coté serveur ainsi qu'une construction de template orienté composants.