Qu'est-ce que React.js et pourquoi l'utiliser ?

Objectifs

Comprendre ce qu'est réellement la librairie React.JS, et pourquoi l'utiliser.

Comprendre

Une librairie front-end

C'est une librairie front end, tout est exécuté dans le navigateur de l'utilisateur.

Pour créer un projet complet avec une base de données, on aura besoin d'y ajouter un backend qui pourra être par exemple du PHP, du Node.js, du .NET ou autre, qui va recevoir les requêtes de votre appli react et écrire ou lire dans la base de données.

Ici on ne va traiter que de react, et on va simuler le backend pour comprendre comment on utilise react en conditions réelles.

Si vous êtes familiers avec le pattern MVC (modèle - vue - controller), React est la partie vue.

Librairie ou framework ?

Beaucoup en parlent en tant que framework car habituellement une librairie est un utilitaire duquel on extrait quelques fonctions qui nous aident dans nos algorithmes, et en général une librairie est une sorte de fonctionnalité supplémentaire par dessus notre application, alors qu'avec react, on a plutôt l'impression que c'est notre application qui repose sur react, comme c'est le cas avec un framework, mais techniquement c'est une librairie que l'on peut inclure avec un simple tag <script> depuis le header d'un fichier html.

Frameworks concurrents

Il y a 2 autres gros frameworks, qui eux sont réellement des frameworks, qui permettent de concevoir des applications avec la même philosophie que react, ce sont Angular et Vue. Angular était extrêmement populaire il y a 5 ou 10 ans, mais React a vraiment pris le dessus et est aujourd'hui le plus populaire.

Avantages de React

Toute notre interface va être découpée en composants unitaires réutilisables qui portent leur propre état en eux et qui réagissent dynamiquement aux changements d'état.

Les composants:

Similaire à ce que l'on peut faire avec Figma par exemple.

Permet de structurer simplement et clairement notre front end. On peut coder des applications énormes tout en conservant une architecture claire car on raisonne en composants unitaires qui n'ont pas ou peu d'effets de bords entre eux.

Chaque composant mène sa vie, même s'il est possible d'échanger des infos entre les composants.

Une énorme application javascript sans react serait beaucoup plus difficile à maintenir et débugger car il n'y a pas de limite claire entre les différents morceaux de code.

JSX :

La syntaxe JSX permet d'écrire du HTML dynamique directement dans le code javascript du composant, ce qui est rend le développement extrêmement facile et agréable.

State et Virtual DOM :

React utilise un Virtual DOM, grossomodo une copie "à la sauce react" du DOM de notre page, qui permet à react de réagir à chaque fois que l'état d'un composant change et de mettre à jour sur la page uniquement la donnée qui a changé.

Avec des pages web classiques qui reposent sur un serveur PHP par exemple, si on supprime un élément à l'écran, on va probablement soumettre une requête au serveur (par exemple PHP) qui va nous répondre avec une nouvelle page web dans laquelle l'élément a disparu. On a donc un rechargement de la page, ce qui est très lent pour l'utilisateur.

Pour contourner cela, l'AJAX a vu le jour à l'époque du Web 2.0, qui permet de mettre à jour dynamiquement des éléments sur la page, mais lorsque la page devient complexe et que l'on veut que tout soit dynamique, il devient très compliqué (impossible ?) de gérer tout ce code dynamique.

Ici, React surveille en arrière plan quelle donnée a changé et met à jour la page web dynamiquement sans aucun rechargement, ce qui permet d'avoir des pages web très performantes et agréables à utiliser pour l'utilisateur. Même lorsqu'on change d'URL, tout se passe côté frontend de manière immédiate

Programmation déclarative :

L'approche déclarative (contrairement à l'approche impérative) permet d'éviter les effets de bord, nous reviendrons en détail sur ce point dans un module dédié.