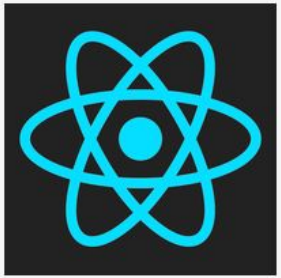
Présentation de React Native, quelles sont ses possibilités et peut-on l'utiliser en prod ?



On se demande souvent quelle solution mobile hybride choisir quand on démarre un projet et nous avons testé plusieurs frameworks pour explorer les possibilités de chacun. Nous vous présentons aujourd'hui React Native.

Qu'est-ce que React Native ?

React Native est un framework mobile hybride développé par Facebook depuis début 2015. Il continue d'évoluer avec le soutient de nombreux contributeurs sur Github.

Facebook a développé l'application de sa dernière Keynote en React Native, qui est un très bon exemple de ce qu'il est possible de faire. Toutes les explications et le code source sont disponibles ici.

"Learn once, write everywhere"

Le but de React Native est de pouvoir réutiliser le maximum de code entre les différentes plateformes (iOS et Android). Il offre un gain de temps considérable par rapport à du développement spécifique, tout en étant aussi performant.

L'écriture en javascript permet aux développeurs web de construire une application mobile native, contrairement à Cordova qui encapsule l'application dans une webview.

React Native utilise le moteur JavaScriptCore avec le transpileur Babel, il est compatible ES5, ES6 ou ES7 sans problème.

Dans la documentation, les exemples utilisent souvent la syntaxe Flow qui sert à vérifier les types en javascript, un peu à la manière de TypeScript

```
function foo(x: string, y: number): number {
  return x.length * y;
}
```

Nous avons annoté ici une function pour dire que nos paramètre seront de type `string` et `number`, et la réponse attendu de type `number`.

Spécificités par rapport au développement pour le web

- **Pas de DOM**

  Inutile de faire appel à `window` ou `document`, React Native n'utilise pas de DOM. **Attention donc à la compatibilité de certaines librairies JS**.

- **Pas de balises HTML**

  Les tags spécifiques au DOM ne sont donc pas admis non plus (`<div>`, `<section>`, `<article>`,...). L'interface doit être construite à partir des composants React-Native uniquement (ou de composants que nous aurions crée) : `<View>`, `<Text>`, `<Image>`,... voir la liste complète dans la documentation

- **Tout doit être packagé**

  Toutes les ressources doivent être appelée, soit par un chemin absolu, soit par un require : `{{uri: urlDeMonImage}}` ou `{require('./chemin/de/mon/image')}`.

- **Styles**

  On déclare le style d'un composant avec l'attribut style, puis on crée un objet `StyleSheet` pour les définir.

```
const Picture = React.createClass({
    render: function() {
        return (
            <View style={styles.container}>
            <Image source={{uri: this.props.url}} />
            <Text>Ceci est une légende</Text>
            </View>
        );
    }
});
var styles = StyleSheet.create({
    container: {
        flex: 1
    },
});
```

  Les styles sont limités à ceux disponibles en JS et peuvent avoir un comportement légèrement différent des styles dans un navigateur. Propriétés disponibles pour le positionnement en flexbox.

  Il n'est pas possible d'utiliser de fichier .css. On peut néanmoins externaliser le style d'un composant dans un fichier JS à part.

  À noter : pour certains composants, l'attribut style n'existe pas, il faut par exemple utiliser `contentContainerStyle` pour les composants `ListView` ou `ScrollView`. Attention donc a bien lire la doc lorsqu'on utilise un nouveau composant.

Une approche par composants

La syntaxe est celle React, si vous êtes déjà familier avec ce framework, vous pouvez passer directement aux spécificités par rapport au développement pour le web. Sinon, nous pouvez retrouver le diaporama "Comment React va révolutionner le web" et suivre ce récapitulatif sur les composants React.

Voici un exemple de composant React. En général un composant est appelé de la manière suivante, il peut aussi utiliser la méthode `extend`, cf. React.createClass versus extends React.Component[en].

```
const Picture = React.createClass({
    render: function() {
        return (
            <View>
                <Image source={{uri: 'http://url-de-mon-image.png'}} />
                <Text>Ceci est une légende</Text>
            </View>
        );
    }
});
```

La fonction `render()` est obligatoire, c'est ce qui va "rendre" les composants React dans le moteur du SDK.

Dans l'exemple ci-dessus, notre composant va rendre l'élément `View`, qui englobe deux autres composants : `Image` et `Text`. La fonction render ne peux rendre qu'un seul élément parent (qui peut contenir autant d'enfants que necéssaire). Nous n'aurions pas pu par exemple avoir directement `Image` et `Text` sans leur parent unique `View`.

La fonction `getInitialState()` permet de déclarer les données par défault à l'initialisation du composant. On peut ensuite les récupérer via `this.state`, comme ici dans la source du composant `Image`.

```
const Picture = React.createClass({
    getInitialState() {
        url: 'http://url-de-mon-image.png'
    },
    render: function() {
        return (
            <View>
                <Image source={{uri: this.state.url}} />
                <Text>Ceci est une légende</Text>
            </View>
        );
    }
});
```

Un composant peut récupérer les propriétés qui lui sont passées en attribut via `this.props`. Ici, le composant Picture récupère l'url de l'image via sa propriété url.

```
<Picture url='http://url-de-mon-image.png' />
```

```
const Picture = React.createClass({
    render: function() {
        return (
            <View>
                <Image source={{uri: this.props.url}} />
                <Text>Ceci est une légende</Text>
            </View>
        );
    }
});
```