

Tutorial React Native

Introduction

ReactNative est un framework JavaScript libre développée par Facebook. Il permet de faciliter le développement d'application mobiles natives multiplateformes en se basant sur le mécanisme de composants déjà mis en place par React.

Structure

Concernant la structure du projet, j'ai décidé de garder la structure générée automatiquement lors de la création d'un projet ReactNative. La structure se présente comme suit :

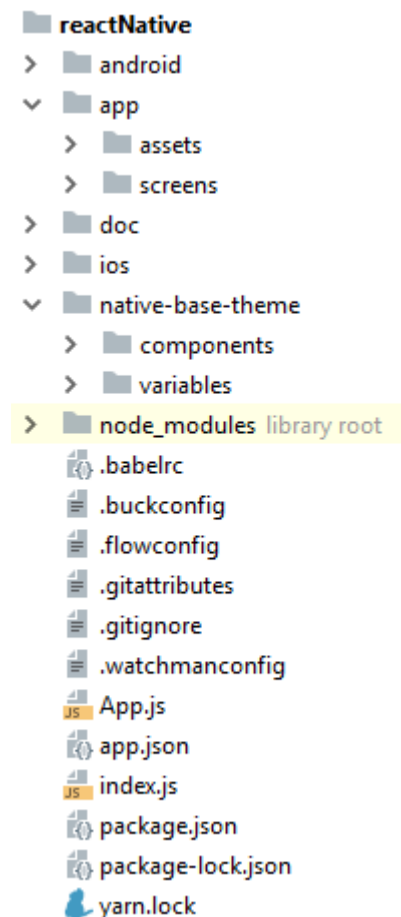


Figure 1 - Structure d'un projet ReactNative

Dans cette structure, on peut trouver deux dossiers « android » et « ios ». Ces dossiers contiennent les fichiers spécifiques aux deux plateformes et nécessaires au bon fonctionnement de l'application une fois compilée. On y trouve le plus souvent des fichiers de configuration, mais également les icônes de l'application ou du code natif servant à importer des modules.

On trouve ensuite un dossier « node_modules » contenant les modules NodeJS installés dans le projet. Ces modules sont des bibliothèques qui vont permettre d'accéder à

certaines fonctionnalités, permettant au développeur de gagner du temps en n'ayant pas à les développer.

Le dossier « app » est un dossier que j'ai créé afin d'y placer les éléments et le code qui serait commun aux différentes plateformes. On y trouve ainsi un dossier « assets » contenant les ressources du projet, ici des images, et un dossier « screens » contenant le code des écrans de l'application et leurs composants respectifs.

Enfin, le dossier « native-base-theme » est un dossier créé à la suite de l'installation du module « native-base ». Ce module permet d'accéder à des composants plus poussés que les composants de base de ReactNative et dont le style visuel est plus acceptable. Ce dossier permet donc de modifier l'apparences des composants du module « native-base » et m'a notamment servi à modifier les couleurs de ces derniers afin de coller avec la charte graphique que nous nous étions fixés pour ce projet.

Utilisation de la technologie

ReactNative est plutôt bien documenté et la mise en place d'un projet utilisant cette technologie n'est pas très complexe.

Le problème « Expo »

Un des premiers problèmes auquel j'ai eu à faire face a été « Expo ». « Expo » est un outil construit autour de ReactNative et qui est sensé rendre le projet plus portable en permettant de compiler une application ReactNative sans passer par AndroidStudio et ses outils ou XCode. ReactNative a ainsi choisi d'intégrer « Expo » par défaut à ses projets. Le problème est que j'ai personnellement eu de nombreux échecs lors de la compilation de l'application dus à « Expo ». Après recherche sur Internet, il s'avère que cette technologie n'est pas complètement stable et que je n'étais pas le seul à avoir des problèmes avec la compilation, ce qui pousse de nombreuses personnes à choisir de développer en se passant d'« Expo ». Suivant leurs conseils et ayant eu moi-même des problèmes avec cette technologie j'ai donc choisi de développer l'application sans « Expo ».

Création de l'application sans « Expo »

Afin de créer une application sans « Expo » il suffit d'utiliser la commande :

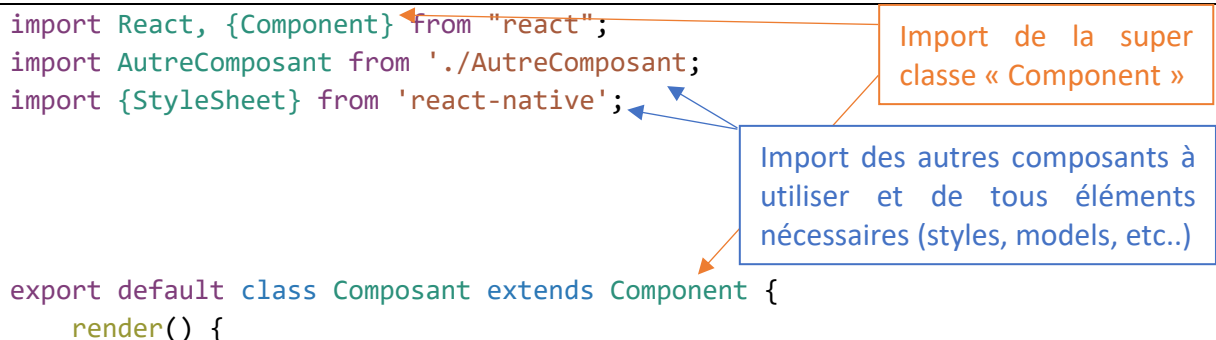
```
react-native init monApplication
```

Création d'un composant

Un composant ReactNative est construit sur le même principe que les composants React :

```
import React, {Component} from "react";
import AutreComposant from './AutreComposant';
import {StyleSheet} from 'react-native';

export default class Composant extends Component {
  render() {
```



```

    return (
      <div style={styles.maDiv}>
        <p> Un élément quelconques</p>
        <AutreComposant />
      </div>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
  },
  welcome: {
    fontSize: 20,
    textAlign: 'center',
    margin: 10,
  },
  instructions: {
    textAlign: 'center',
    color: '#333333',
    marginBottom: 5,
  },
  map: {
    ...StyleSheet.absoluteFillObject,
  },
});

```

ReactNative fonctionnant comme React, le DOM n'est modifié que lorsque cela est absolument nécessaire. La modification du DOM se fait dans la méthode « render ».

Interactions entre les composants

Comme en React, il est possible de passer des données à un composant fils à l'aide des « props ».

```

// Composant père :
const donnee = 'valeur quelconque';
render() {
  return (<div><ComposantFils proprieteDuFils={this.donnee} /></div>);
}

```

On passe les données nécessaires comme des attributs HTML, cela peut correspondre à des variables ou directement à des valeurs (syntaxe ES6)

```
// Composant fils :  
class ComposantFils extends Component {  
  constructor(props) {  
    super(props);  
    this.variable = props.proprieteDuFils;  
  }  
}
```

Le constructeur va par défaut récupérer ces données dans la variable « props ». Elles seront accessibles avec l'appel suivant :
« props.nomPasseLorsDeLaCreation »

Actualiser l'affichage d'un composant

Comme React, ReactNative utilise un système d'état. L'état va être une variable contenant les différentes variables propres à la vue. Pour modifier cet état, on doit faire appel à la méthode « setState » qui va non seulement changer la variable d'état mais également lancer l'actualisation de la vue.

Mise en place des adaptations

Afin de mettre en place les adaptations choisies au sein de l'application, j'ai dû avoir accès à la position de l'utilisateur. Pour cela ReactNative intègre une méthode permettant d'accéder aux coordonnées géographiques de l'utilisateur. Cette méthode se présente comme suit :

```
navigator.geolocation.getCurrentPosition(successCallback(), errorCallback());
```

Référencement des avantages et des inconvénients

Avantages

- Génération d'une application native à partir d'un code commun
- Application aussi fluide et performante qu'une application directement développée en natif
- Liberté de la structure de l'application (voir **Erreur ! Source du renvoi introuvable.**),
- Possibilité d'écrire du JavaScript librement (syntaxe ES6 supportée),
- « Interactions » (passage de données) entre les composants faciles (voir **Erreur ! Source du renvoi introuvable.**),
- Application en kit, ce qui permet de n'importer que le strict nécessaire (voir **Erreur ! Source du renvoi introuvable.**),
- Navigation entre les composants très rapide à prendre en main (voir **Erreur ! Source du renvoi introuvable.**).

Inconvénients

- Utilisation de modules « communautaires » qui risquent d'amener des problèmes de compatibilités lors de futures mises à jour