eBook Gratuit

APPRENEZ react-native

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#react-native

Table des matières

À propos ... 1

Chapitre 1: Démarrer avec rea-native ... 2

Remarques ... 2

Examples ... 2

Configuration pour Mac ... 2

Configuration pour Windows ... 14

Configuration pour Linux (Ubuntu) ... 15

Démarrez le terminal et exécutez les commandes suivantes pour installer nodeJS: ... 15

Si la commande de noeud est indisponible ... 16

Instalations NodeJS alternatives: ... 16

vérifier si vous avez la version actuelle ... 16

Exécutez le npm pour installer le réactif ... 16

Android SDK ou Android Studio ... 16

SDK Android e ENV ... 16

Exemple d'application init ... 17

Obs: Toujours vérifier si la version sur android/app/build.gradle est la même que les outi ... 18

Ouvrez Android AVD pour configurer un Android virtuel. Exécutez la ligne de commande: ... 18

Chapitre 2: Android - Bouton Retour de matériel ... 19

Examples ... 19

Détecter le bouton Retour du matériel dans Android ... 19

Exemple de BackAndroid avec Navigator ... 19

Exemple de détection de bouton retour matériel à l'aide de BackHandler ... 20

Gestion du bouton retour du matériel à l'aide de BackHandler et des propriétés de navigati ... 20

Chapitre 3: API d'animation ... 22

Examples ... 22

Animer une image ... 22

Chapitre 4: Bonjour le monde ... 23

Examples ... 23

Modification de index.ios.js ou index.android.js ... 23

Bonjour le monde! ... 23

Chapitre 5: Coiffant 24
Introduction 24
Syntaxe 24
Remarques 24
Examples 24
Style utilisant des styles en ligne 24
Styling à l'aide d'une feuille de style 24
Ajouter plusieurs styles 25
Styling conditionnel 26
Chapitre 6: Composants 27
Examples 27
Composant de base 27
Composant avec état 27
Composant sans état 27
Chapitre 7: Créer un partageable APK pour Android 29
Introduction 29
Remarques 29
Examples 29
Créer une clé pour signer l'APK 29
Une fois la clé générée, utilisez-la pour générer la version installable: 29
Générer la construction en utilisant gradle 29
Télécharger ou partager le fichier APK généré 29
Chapitre 8: Disposition 31
Examples 31
Flexbox 31
flexDirection 31
Axe d'alignement 32
Alignement 34
Taille de flex 34
Chapitre 9: ESLint in react-native 35
Introduction 35

Examples ... 35
Comment commencer ... 35
Chapitre 10: Etat ... 36
Syntaxe ... 36
Examples ... 36
setState ... 36
Exemple complet ... 36
Initialiser l'état ... 38
Chapitre 11: Exécuter une application sur l'appareil (version Android) ... 39
Remarques ... 39
Examples ... 39
Lancer une application sur un appareil Android ... 39
Chapitre 12: Images ... 40
Examples ... 40
Module d'image ... 40
Exemple d'image ... 40
Source d'image conditionnelle ... 40
Utilisation d'une variable pour le chemin de l'image ... 40
Pour adapter une image ... 41
Chapitre 13: Instructions en ligne de commande ... 42
Examples ... 42
Vérifier la version installée ... 42
Mettre à niveau le projet existant vers la dernière version RN ... 42
Enregistrement ... 42
Initialiser et démarrer avec le projet React Native ... 42
Démarrer React Native Packager ... 43
Ajouter un projet Android pour votre application ... 43
Chapitre 14: Intégration avec Firebase pour l'authentification ... 44
Introduction ... 44
Examples ... 44
React Native - ListView avec Firebase ... 44
Authentification dans React Native avec Firebase ... 45

Chapitre 15: Le débogage 47
Syntaxe 47
Examples 47
Démarrez le débogage JS à distance dans Android 47
Utiliser console.log () 47
Chapitre 16: Le routage 48
Introduction 48
Examples 48
Composant Navigateur 48
Chapitre 17: Les accessoires 49
Introduction 49
Examples 49
Quels sont les accessoires? 49
Utilisation des accessoires 49
PropTypes 50
Accessoires par défaut 51
Chapitre 18: Liaison de l'API native 52
Introduction 52
Examples 52
Liens sortants 52
Schémas d'URI 52
Liens entrants 53
Chapitre 19: ListView 54
Examples 54
Exemple simple 54
Chapitre 20: Meilleures pratiques de rendu 55
Introduction 55
Examples 55
Fonctions dans JSX 55
Chapitre 21: Meilleures pratiques du navigateur 57
Examples 57

Navigateur ... 57
Utiliser rea-navigation pour naviguer dans les applications natives de réaction ... 59
Navigation ré-native avec react-native-router-flux ... 60
Chapitre 22: Modal ... 62
Introduction ... 62
Paramètres ... 62
Examples ... 62
Exemple de base modal ... 62
Exemple modal transparent ... 63
Chapitre 23: Module de plate-forme ... 65
Examples ... 65
Trouver le type / version du système d'exploitation ... 65
Chapitre 24: Modules natifs ... 66
Examples ... 66
Créez votre module natif (IOS) ... 66
introduction ... 66
Exemple ... 66
Chapitre 25: Navigateur avec boutons injectés depuis les pages ... 68
Examples ... 68
introduction ... 68
Exemple complet commenté ... 68
Chapitre 26: Notification push ... 72
Introduction ... 72
Remarques ... 72
Examples ... 72
Push Setup Simple Setup ... 72
Navigation vers la scène à partir de la notification ... 74
Chapitre 27: Polices Personnalisées ... 77
Examples ... 77
Étapes pour utiliser des polices personnalisées dans React Native (Android) ... 77
Étapes pour utiliser des polices personnalisées dans React Native (iOS) ... 77

Polices personnalisées pour Android et IOS 78
Android 79
iOS 80
Chapitre 28: RefreshControl avec ListView 81
Remarques 81
Examples 81
Contrôle de rafraîchissement 81
Fonction onRefresh Exemple 81
Refresh Control with ListView Exemple complet 81
Chapitre 29: Rendu de plusieurs accessoires 84
Examples 84
rendre plusieurs variables 84
Chapitre 30: Requêtes HTTP 85
Syntaxe 85
Remarques 85
Examples 85
WebSockets 85
HTTP avec l'API de récupération 85
Mise en réseau avec XMLHttpRequest 86
Utiliser des promesses avec l'API d'extraction et Redux 86
Socket Web avec Socket.io 87
Http avec axios 88
Chapitre 31: Test d'unité 90
Introduction 90
Examples 90
Test unitaire avec blague 90
Test unitaire dans React Native utilisant Jest 91
Chapitre 32: WebView 92
Introduction 92
Examples 92
Composant simple utilisant webview 92
Crédits 93

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: react-native

It is an unofficial and free react-native ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official react-native.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec rea-native

Remarques

React Native vous permet de créer des applications mobiles en utilisant uniquement JavaScript. Il utilise la même conception que React, vous permettant de composer une interface utilisateur mobile riche à partir de composants déclaratifs.

Avec React Native, vous ne créez pas une «application Web mobile», une «application HTML5» ou une «application hybride». Vous construisez une véritable application mobile qui ne se distingue pas d'une application créée avec Objective-C ou Java. React Native utilise les mêmes éléments fondamentaux que les applications iOS et Android classiques. Vous venez de mettre ces blocs de construction ensemble en utilisant JavaScript et React.

Il est open-source et maintenu par Facebook.

- Site Internet
- Documentation
- GitHub Repository

Source: site web React Native

Examples

Configuration pour Mac

Installation gestionnaire de paquets Homebrew `brew`

Collez-le à l'invite du terminal.

```
/usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Installation de l'IDE Xcode

Téléchargez-le en utilisant le lien ci-dessous ou trouvez-le sur le Mac App Store

https://developer.apple.com/download/

> **REMARQUE:** Si **Xcode-beta.app est** installé avec la version de production de **Xcode.app** , assurez-vous d'utiliser la version de production de l'outil `xcodebuild` . Vous pouvez le configurer avec:
>
> ```
> sudo xcode-select -switch /Applications/Xcode.app/Contents/Developer/
> ```

Installation de l'environnement Android

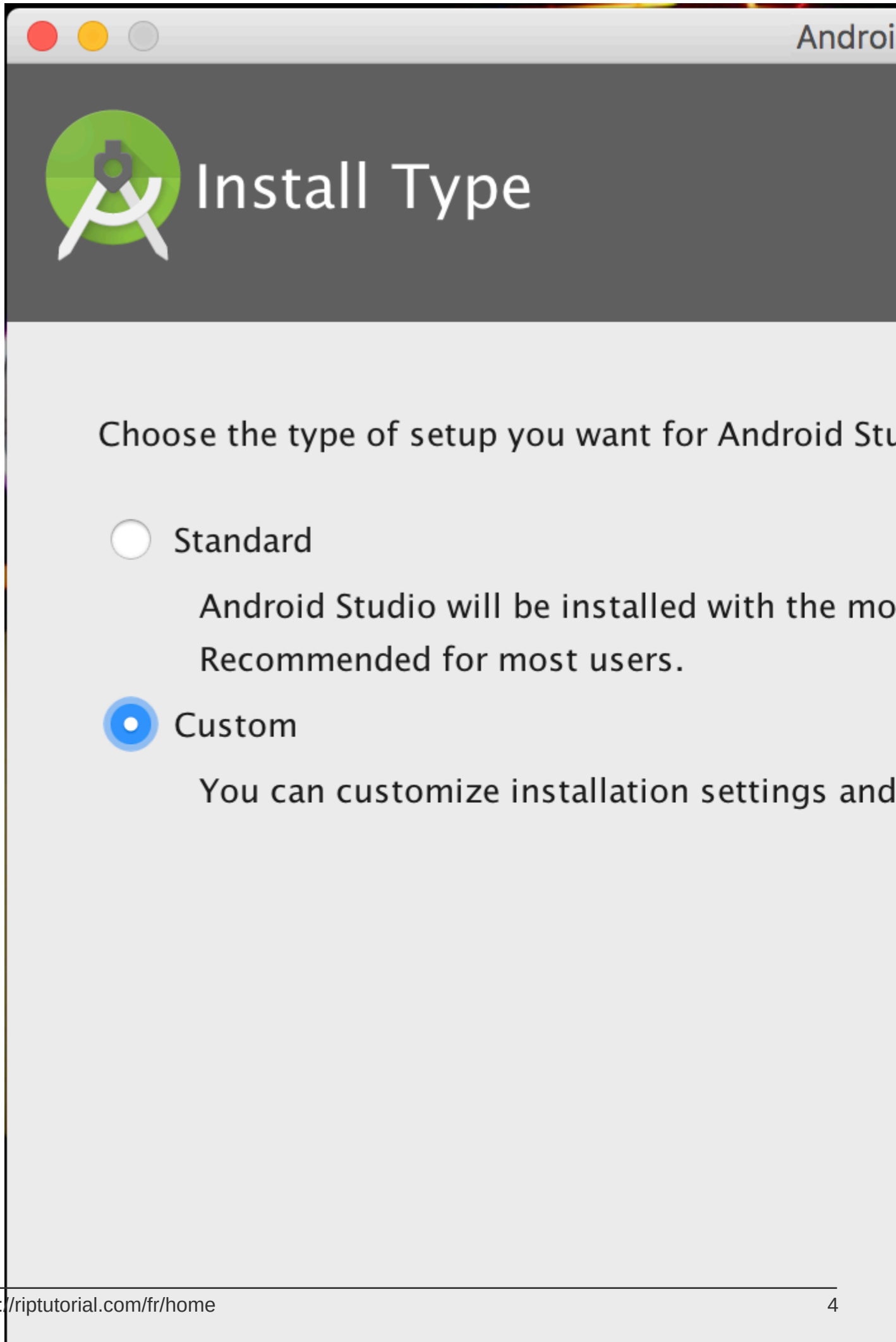
- git `git`

* Si vous avez installé XCode, Git est déjà installé, sinon exécutez la commande suivante

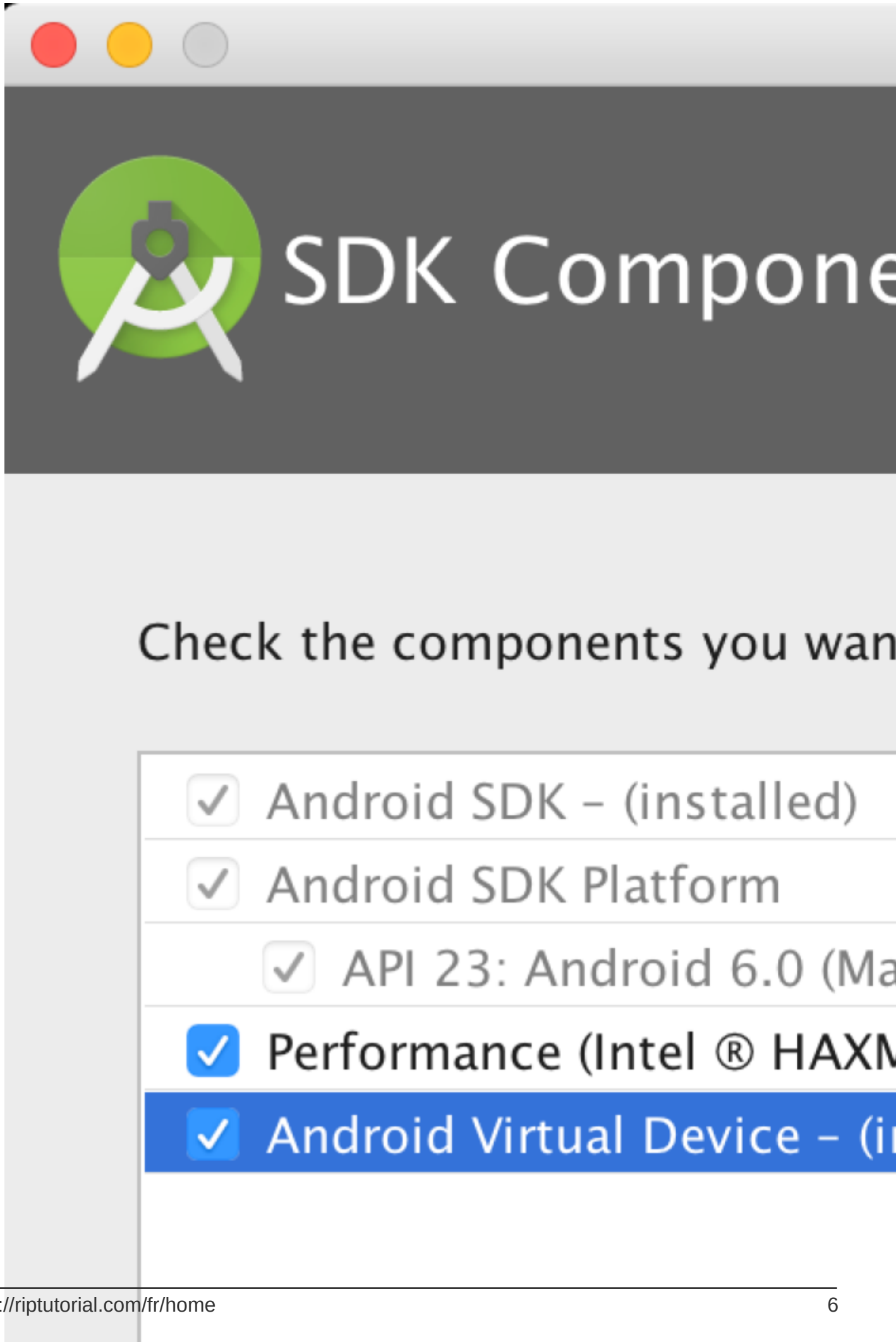
```
brew install git
```

- Dernier JDK
- Studio Android

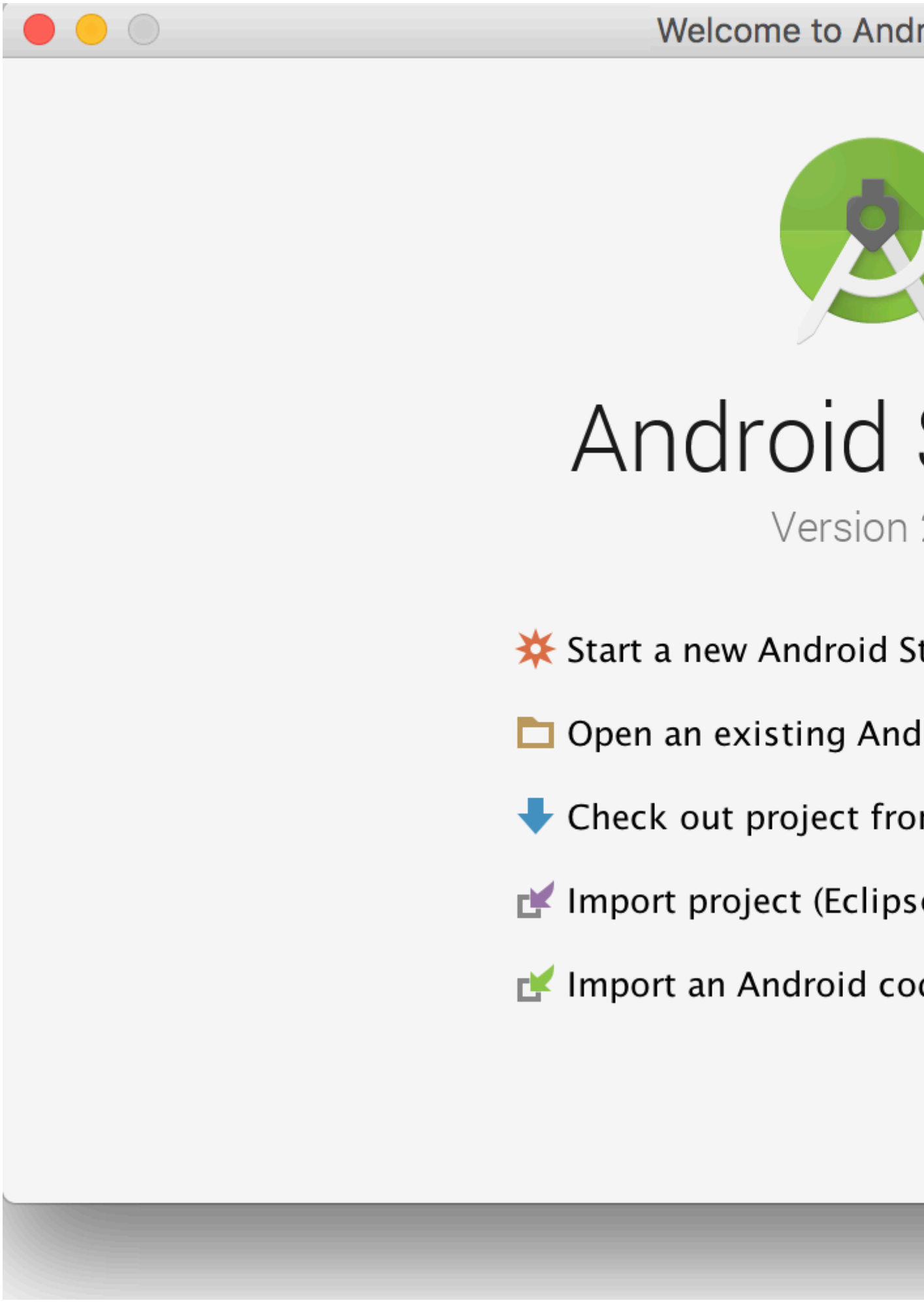
  Choisissez une installation personnalisée


Androi
Install Type
Choose the type of setup you want for Android Stu
Standard
Android Studio will be installed with the mo
Recommended for most users.
Custom
You can customize installation settings and

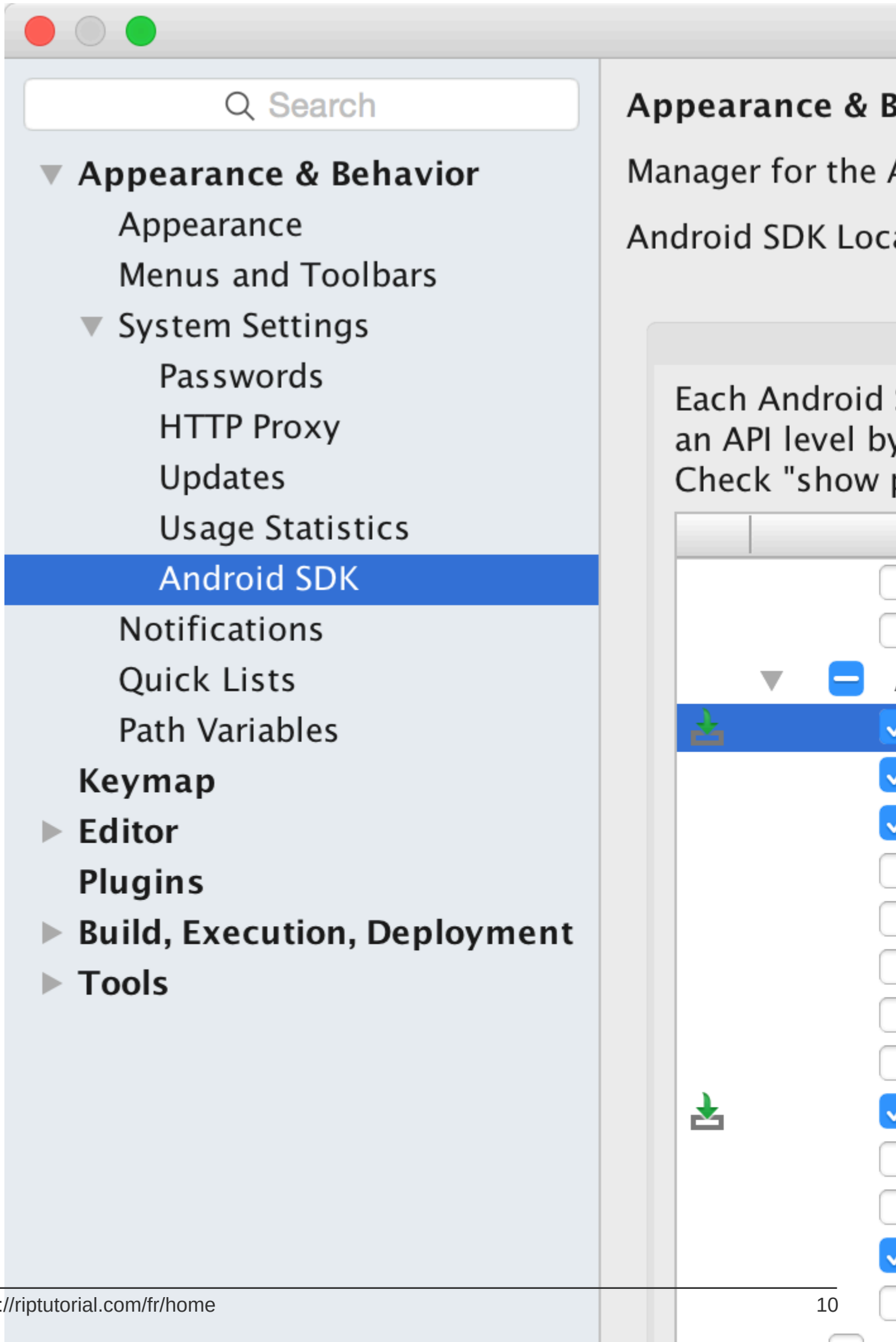
Choisissez Performance et Android Virtual Device


SDK Compone
Check the components you wan
Android SDK – (installed)
Android SDK Platform
API 23: Android 6.0 (Ma
Performance (Intel ® HAXM
Android Virtual Device – (i

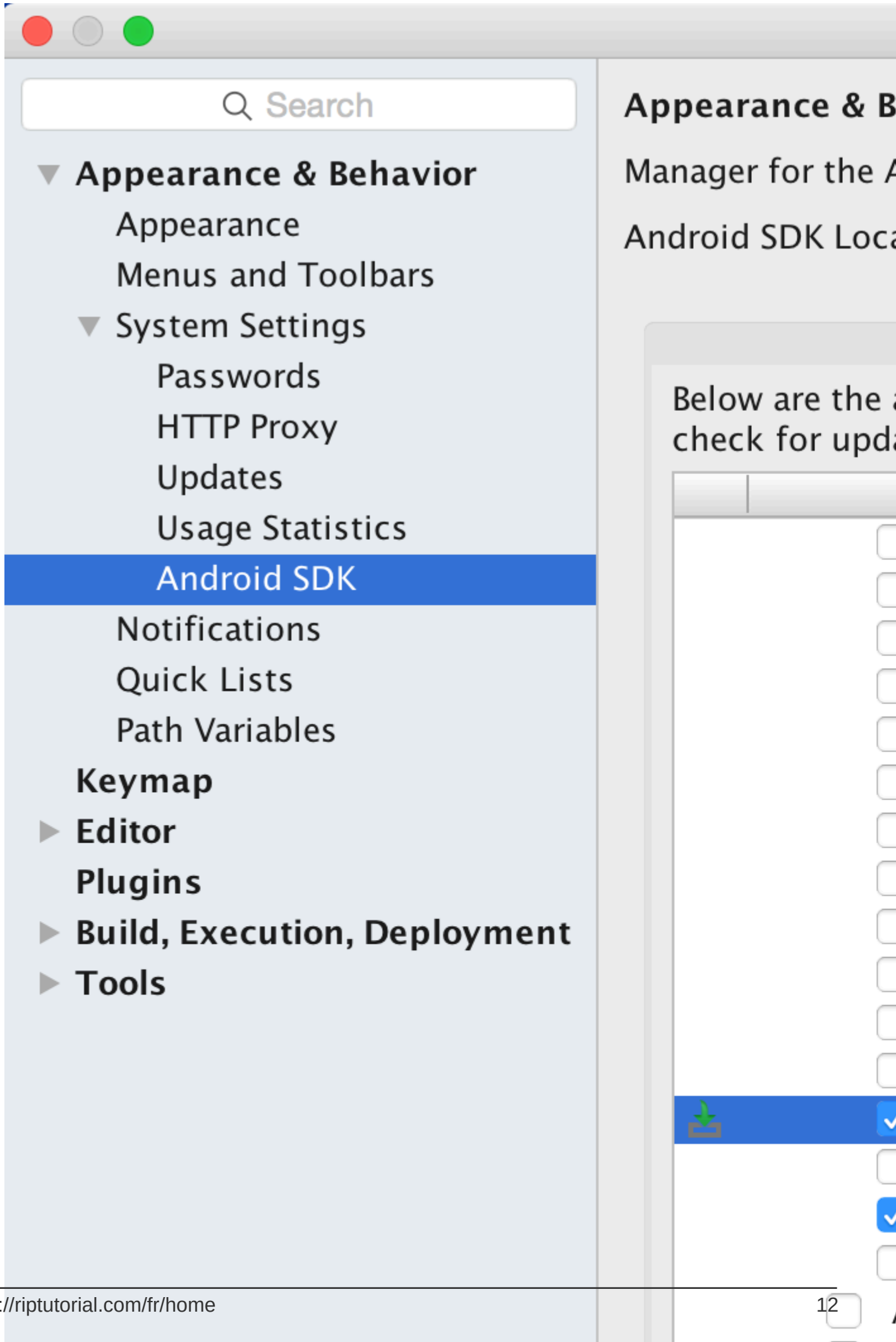
Après l'installation, choisissez Configurer -> SDK Manager dans la fenêtre de bienvenue d'Android Studio.


Welcome to Andr
Android S
Version
Start a new Android St
Open an existing And
Check out project fro
Import project (Eclips
Import an Android co

Dans la fenêtre Plateformes SDK, choisissez Afficher les détails du package et sous Android 6.0 (Marshmallow), vérifiez que les API Google, Image du système Intel x86 Atom, Image du système Intel x86 Atom_64 et API système Intel x86 Atom_64 sont cochées.


Search
Appearance & Behavior
Appearance
Menus and Toolbars
System Settings
Passwords
HTTP Proxy
Updates
Usage Statistics
Android SDK
Notifications
Quick Lists
Path Variables
Keymap
Editor
Plugins
Build, Execution, Deployment
Tools
Appearance & B
Manager for the A
Android SDK Loca
Each Android
an API level by
Check "show

Dans la fenêtre Outils du SDK, choisissez Afficher les détails du package et sous Outils de génération du SDK Android, assurez-vous que l'option Outils de génération du SDK Android 23.0.1 est sélectionnée.


Search
Appearance & Behavior
Appearance
Menus and Toolbars
System Settings
Passwords
HTTP Proxy
Updates
Usage Statistics
Android SDK
Notifications
Quick Lists
Path Variables
Keymap
Editor
Plugins
Build, Execution, Deployment
Tools
Appearance & B
Manager for the A
Android SDK Loca
Below are the
check for upd

- Variable d'environnement `ANDROID_HOME`

  Assurez-vous que la variable d'environnement ANDROID_HOME pointe vers votre SDK Android existant. Pour ce faire, ajoutez ceci à votre ~ / .bashrc, ~ / .bash_profile (ou à tout ce que votre shell utilise) et rouvrez votre terminal:

  Si vous avez installé le SDK sans Android Studio, cela peut être quelque chose comme: / usr / local / opt / android-sdk

  ```
  export ANDROID_HOME=~/Library/Android/sdk
  ```

Dépendances pour Mac

Vous aurez besoin de Xcode pour iOS et Android Studio pour Android, node.js, les outils de ligne de commande React Native et Watchman.

Nous vous recommandons d'installer node et watchman via Homebrew.

```
brew install node
brew install watchman
```

> Watchman est un outil de Facebook pour regarder les changements dans le système de fichiers. Il est fortement recommandé de l'installer pour de meilleures performances. C'est facultatif.

Node est livré avec npm, qui vous permet d'installer l'interface de ligne de commande native React.

```
npm install -g react-native-cli
```

Si vous obtenez une erreur de permission, essayez avec sudo:

```
sudo npm install -g react-native-cli.
```

Pour iOS, le moyen le plus simple d'installer Xcode est d'utiliser le Mac App Store. Et pour Android télécharger et installer Android Studio.

Si vous envisagez de modifier le code Java, nous vous recommandons d'utiliser Gradle Daemon, qui accélère la génération.

Test de votre installation native React

Utilisez les outils de ligne de commande React Native pour générer un nouveau projet React Native appelé "AwesomeProject", puis exécutez des run-ios natifs dans le dossier nouvellement créé.

```
react-native init AwesomeProject
cd AwesomeProject
react-native run-ios
```

Vous devriez voir votre nouvelle application s'exécuter sous iOS Simulator prochainement. run-ios natif de react-end n'est qu'un moyen d'exécuter votre application - vous pouvez également l'exécuter directement depuis Xcode ou Nuclide.

Modifier votre application

Maintenant que vous avez réussi à exécuter l'application, modifions-la.

- Ouvrez index.ios.js ou index.android.js dans votre éditeur de texte de choix et éditez certaines lignes.
- Hit Command + R dans votre simulateur iOS pour recharger l'application et voir votre changement! C'est tout!

Toutes nos félicitations! Vous avez exécuté et modifié avec succès votre première application React Native.

source: Pour commencer - React-Native

Configuration pour Windows

Remarque: vous ne pouvez pas développer d'applications réactives pour iOS sur Windows, mais uniquement des applications Android réactives.

Les documents de configuration officiels pour rea-native sur Windows peuvent être trouvés ici . Si vous avez besoin de plus de détails, voici un guide granulaire .

Outils / Environnement

- Windows 10
- outil de ligne de commande (par exemple, ligne de commande Powershell ou Windows)
- Chocolaté (étapes pour configurer via PowerShell)
- Le JDK (version 8)
- Studio Android
- Une machine Intel avec la technologie de virtualisation activée pour HAXM (facultatif, nécessaire uniquement si vous souhaitez utiliser un émulateur)

1) Configurez votre machine pour réagir au développement natif

Démarrez la ligne de commande en tant qu'administrateur exécutez les commandes suivantes:

```
choco install nodejs.install
choco install python2
```

Redémarrez la ligne de commande en tant qu'administrateur pour pouvoir exécuter npm

```
npm install -g react-native-cli
```

Après avoir exécuté la dernière commande, copiez le répertoire dans lequel react-native a été installé. Vous en aurez besoin pour l'étape 4. J'ai essayé ceci sur deux ordinateurs dans un cas:
`C:\Program Files (x86)\Nodist\v-x64\6.2.2`

. Dans l'autre c'était: `C:\Users\admin\AppData\Roaming\npm`

2) Définissez vos variables d'environnement

Un guide étape par étape avec des images peut être trouvé ici pour cette section.

Ouvrez la fenêtre Variables d'environnement en naviguant vers:

[Clic droit] Menu "Démarrer" -> Système -> Paramètres système avancés -> Variables d'environnement

Dans la section inférieure, recherchez la variable système "Path" et ajoutez l'emplacement d'installation de react-native à l'étape 1.

Si vous n'avez pas ajouté de variable d'environnement ANDROID_HOME, vous devrez également le faire ici. Dans la fenêtre "Variables d'environnement", ajoutez une nouvelle variable système nommée "ANDROID_HOME" et la valeur correspondant au chemin d'accès à votre SDK Android.

Redémarrez ensuite la ligne de commande en tant qu'administrateur pour pouvoir y exécuter des commandes réactives.

3) Créez votre projet En ligne de commande, accédez au dossier dans lequel vous souhaitez placer votre projet et exécutez la commande suivante:

```
react-native init ProjectName
```

4) Lancez votre projet Démarrez un émulateur depuis Android Studio Accédez au répertoire racine de votre projet en ligne de commande et exécutez-le:

```
cd ProjectName
react-native run-android
```

Vous pouvez rencontrer des problèmes de dépendance. Par exemple, il se peut que vous n'ayez pas la version correcte des outils de génération. Pour résoudre ce problème, vous devrez ouvrir le gestionnaire sdk dans Android Studio et télécharger les outils de construction à partir de là.

Félicitations!

Pour actualiser l'interface utilisateur, vous pouvez appuyer deux fois sur la touche `r` dans l'émulateur et exécuter l'application. Pour voir les options du développeur, vous pouvez appuyer sur `ctrl + m` .

Configuration pour Linux (Ubuntu)

1) Setup Node.JS

Démarrez le terminal et exécutez les

commandes suivantes pour installer nodeJS:

```
curl -sL https://deb.nodesource.com/setup_5.x | sudo -E bash -

sudo apt-get install nodejs
```

Si la commande de noeud est indisponible

```
sudo ln -s /usr/bin/nodejs /usr/bin/node
```

Instalations NodeJS alternatives:

```
curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
sudo apt-get install -y nodejs
```

ou

```
curl -sL https://deb.nodesource.com/setup_7.x | sudo -E bash -
sudo apt-get install -y nodejs
```

vérifier si vous avez la version actuelle

```
node -v
```

Exécutez le npm pour installer le réactif

```
sudo npm install -g react-native-cli
```

2) Configurer Java

```
sudo apt-get install lib32stdc++6 lib32z1 openjdk-7-jdk
```

3) Configuration d'Android Studio:

Android SDK ou Android Studio

```
http://developer.android.com/sdk/index.html
```

SDK Android e ENV

```
export ANDROID_HOME=/YOUR/LOCAL/ANDROID/SDK
```

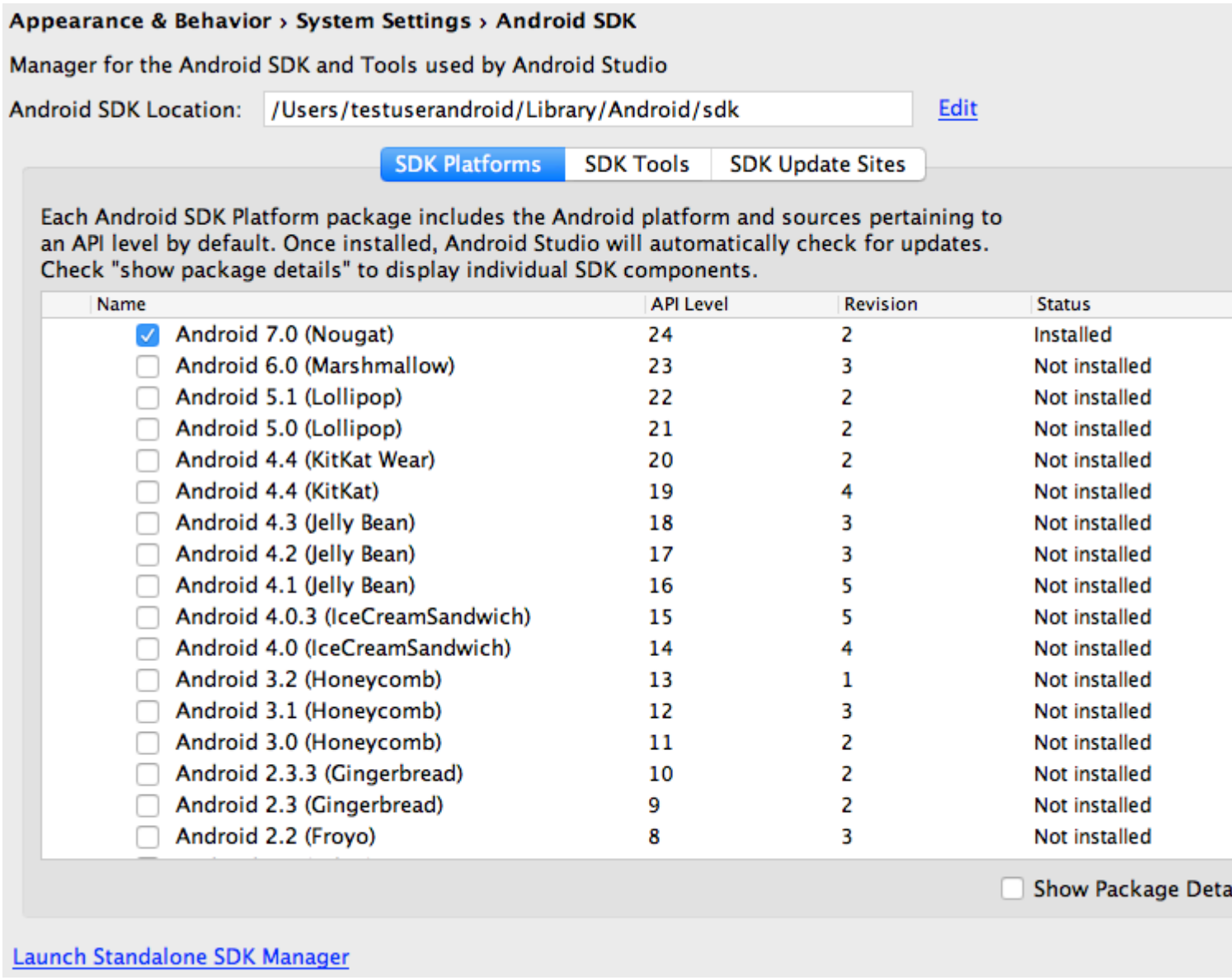
```
export PATH=$PATH:$ANDROID_HOME/tools:$ANDROID_HOME/platform-tools
```

4) émulateur d'installation:

Sur le terminal, exécutez la commande

```
android
```

Sélectionnez "SDK Platforms" dans le SDK Manager et vous devriez voir une coche bleue à côté de "Android 7.0 (Nougat)". Si ce n'est pas le cas, cliquez sur la case à cocher puis sur "Appliquer".



5) Lancer un projet

Exemple d'application init

```
react-native init ReactNativeDemo && cd ReactNativeDemo
```

Obs: Toujours vérifier si la version sur `android/app/build.gradle` est la même que les outils de construction téléchargés sur votre SDK Android

```
android {
    compileSdkVersion XX
    buildToolsVersion "XX.X.X"
...
```

6) Exécuter le projet

Ouvrez Android AVD pour configurer un Android virtuel. Exécutez la ligne de commande:

```
android avd
```

Suivez les instructions pour créer un périphérique virtuel et lancez-le

Ouvrez un autre terminal et exécutez les lignes de commande:

```
react-native run-android
react-native start
```

Lire Démarrer avec rea-native en ligne: https://riptutorial.com/fr/react-native/topic/857/demarrer-avec-rea-native

Chapitre 2: Android - Bouton Retour de matériel

Examples

Détecter le bouton Retour du matériel dans Android

```
BackAndroid.addEventListener('hardwareBackPress', function() {
    if (!this.onMainScreen()) {
        this.goBack();
        return true;
    }
    return false;
});
```

Remarque: `this.onMainScreen()` et `this.goBack()` ne sont pas des fonctions intégrées, vous devez également les implémenter. (https://github.com/immidi/react-native/commit/ed7e0fb31d842c63e8b8dc77ce795fac86e0f712)

Exemple de BackAndroid avec Navigator

Voici un exemple d'utilisation de `BackAndroid` React Native avec le `Navigator` .

`componentWillMount` enregistre un écouteur d'événement pour gérer les taps sur le bouton arrière. Il vérifie s'il y a une autre vue dans la pile d'historique, et s'il en existe une, elle reprend le comportement par défaut.

Plus d'informations sur les documents `BackAndroid` et les documents `Navigator` .

```
import React, { Component } from 'react'; // eslint-disable-line no-unused-vars

import {
  BackAndroid,
  Navigator,
} from 'react-native';

import SceneContainer from './Navigation/SceneContainer';
import RouteMapper from './Navigation/RouteMapper';

export default class AppContainer extends Component {

  constructor(props) {
    super(props);

    this.navigator;
  }

  componentWillMount() {
    BackAndroid.addEventListener('hardwareBackPress', () => {
      if (this.navigator && this.navigator.getCurrentRoutes().length > 1) {
```

```
        this.navigator.pop();
        return true;
      }
      return false;
    });
  }

  renderScene(route, navigator) {
    this.navigator = navigator;

    return (
      <SceneContainer
        title={route.title}
        route={route}
        navigator={navigator}
        onBack={() => {
          if (route.index > 0) {
            navigator.pop();
          }
        }}
        {...this.props} />
    );
  }

  render() {
    return (
      <Navigator
        initialRoute={<View />}
        renderScene={this.renderScene.bind(this)}
        navigationBar={
          <Navigator.NavigationBar
            style={{backgroundColor: 'gray'}}
            routeMapper={RouteMapper} />
        } />
    );
  }
};
```

Exemple de détection de bouton retour matériel à l'aide de BackHandler

Depuis BackAndroid est obsolète. Utilisez BackHandler au lieu de BackAndroid.

```
import { BackHandler } from 'react-native';

{...}
  ComponentWillMount(){
    BackHandler.addEventListener('hardwareBackPress',()=>{
      if (!this.onMainScreen()) {
        this.goBack();
        return true;
      }
      return false;
    });
  }
```

Gestion du bouton retour du matériel à l'aide de BackHandler et des propriétés de navigation (sans utiliser BackAndroid et Navigateur obsolète

obsolètes)

Cet exemple vous montrera la navigation qui est généralement attendue dans la plupart des flux. Vous devrez ajouter le code suivant à chaque écran en fonction du comportement attendu. Il y a 2 cas:

1. S'il y a plus d'un écran sur la pile, le bouton Retour de l'appareil affichera l'écran précédent.
2. S'il n'y a qu'un écran sur la pile, le bouton Retour du périphérique quittera l'application.

Cas 1: Afficher l'écran précédent

```
import { BackHandler } from 'react-native';

constructor(props) {
    super(props)
    this.handleBackButtonClick = this.handleBackButtonClick.bind(this);
}

componentWillMount() {
    BackHandler.addEventListener('hardwareBackPress', this.handleBackButtonClick);
}

componentWillUnmount() {
    BackHandler.removeEventListener('hardwareBackPress', this.handleBackButtonClick);
}

handleBackButtonClick() {
    this.props.navigation.goBack(null);
    return true;
}
```

Important: n'oubliez pas de lier method in constructor et de supprimer listener dans componentWillUnmount.

Cas 2: application de sortie

Dans ce cas, pas besoin de manipuler quoi que ce soit sur cet écran où vous souhaitez quitter l'application.

Important: Ceci ne devrait être qu'un écran sur la pile.

Lire Android - Bouton Retour de matériel en ligne: https://riptutorial.com/fr/react-native/topic/4668/android---bouton-retour-de-materiel

Chapitre 3: API d'animation

Examples

Animer une image

```
class AnimatedImage extends Component {
    constructor(props){
        super(props)
        this.state = {
            logoMarginTop: new Animated.Value(200)
        }
    }
    componentDidMount(){
        Animated.timing(
            this.state.logoMarginTop,
            { toValue: 100 }
        ).start()
    }
    render () {
      return (
        <View>
           <Animated.Image source={require('../images/Logo.png')} style={[baseStyles.logo, {
              marginTop: this.state.logoMarginTop
           }]} />
        </View>
      )
    }
}
```

Cet exemple anime la position de l'image en modifiant la marge.

Lire API d'animation en ligne: https://riptutorial.com/fr/react-native/topic/4415/api-d-animation

Chapitre 4: Bonjour le monde

Examples

Modification de index.ios.js ou index.android.js

Ouvrez `index.ios.js` ou `index.android.js` et supprimez tout ce qui se trouve entre `<View> </View>` . Après cela, écrivez `<Text> Hello World! </Text>` et lancez l'émulateur.

Vous devriez voir `Hello World!` écrit à l'écran!

Félicitations! Vous avez écrit avec succès votre premier Hello World!

Bonjour le monde!

```
import React, { Component } from 'react';
import { AppRegistry, Text } from 'react-native';

class HelloWorldApp extends Component {
  render() {
    return (
      <Text>Hello world!</Text>
    );
  }
}

AppRegistry.registerComponent('HelloWorldApp', () => HelloWorldApp);
```

Lire Bonjour le monde en ligne: https://riptutorial.com/fr/react-native/topic/3779/bonjour-le-monde

Chapitre 5: Coiffant

Introduction

Les styles sont définis dans un objet JSON avec des noms d'attributs de style similaires, comme dans CSS. Un tel objet peut soit être mis en ligne dans le style prop d'un composant, soit être transmis à la fonction `StyleSheet.create(StyleObject)` et être stocké dans une variable pour un accès en ligne plus court en utilisant un nom de sélecteur similaire à une classe. en CSS.

Syntaxe

- `<Component style={styleFromStyleSheet} />`
- `<Component style={styleObject} />`
- `<Component style={[style1,style2]} />`

Remarques

La plupart des styles React Native sont leurs formulaires CSS, mais dans un cas camel. Ainsi, la `text-decoration` devient `textDecoration` .

Contrairement aux CSS, les styles ne sont pas hérités. Si vous souhaitez que les composants enfants héritent d'un certain style, vous devez le fournir explicitement à l'enfant. Cela signifie que vous ne pouvez pas définir une famille de polices pour une `View` entière.
La seule exception à cette règle est le composant `Text` : les `Text` imbriqués héritent de leurs styles parents.

Examples

Style utilisant des styles en ligne

Chaque composant React Native peut prendre un accessoire de `style` . Vous pouvez lui transmettre un objet JavaScript avec des propriétés de style CSS:

```
<Text style={{color:'red'}}>Red text</Text>
```

Cela peut être inefficace car il doit recréer l'objet chaque fois que le composant est rendu. Utiliser une feuille de style est préférable.

Styling à l'aide d'une feuille de style

```
import React, { Component } from 'react';
import { View, Text, StyleSheet } from 'react-native';

const styles = StyleSheet.create({
    red: {
```

```
        color: 'red'
    },
    big: {
        fontSize: 30
    }
});

class Example extends Component {
    render() {
        return (
            <View>
                <Text style={styles.red}>Red</Text>
                <Text style={styles.big}>Big</Text>
            </View>
        );
    }
}
```

`StyleSheet.create()` renvoie un objet dont les valeurs sont des nombres. React Native sait convertir ces identifiants numériques en objet de style correct.

Ajouter plusieurs styles

Vous pouvez passer un tableau au `style` prop pour appliquer plusieurs styles. En cas de conflit, le dernier de la liste est prioritaire.

```
import React, { Component } from 'react';
import { View, Text, StyleSheet } from 'react-native';

const styles = StyleSheet.create({
    red: {
        color: 'red'
    },
    greenUnderline: {
        color: 'green',
        textDecoration: 'underline'
    },
    big: {
        fontSize: 30
    }
});

class Example extends Component {
    render() {
        return (
            <View>
                <Text style={[styles.red, styles.big]}>Big red</Text>
                <Text style={[styles.red, styles.greenUnderline]}>Green underline</Text>
                <Text style={[styles.greenUnderline, styles.red]}>Red underline</Text>
                <Text style={[styles.greenUnderline, styles.red, styles.big]}>Big red
underline</Text>
                <Text style={[styles.big, {color:'yellow'}]}>Big yellow</Text>
            </View>
        );
    }
}
```

Styling conditionnel

```
<View style={[(this.props.isTrue) ? styles.bgcolorBlack : styles.bgColorWhite]}>
```

Si la valeur de `isTrue` est `true` , la couleur de fond noire sera blanche.

Lire Coiffant en ligne: https://riptutorial.com/fr/react-native/topic/7757/coiffant

Chapitre 6: Composants

Examples

Composant de base

```
import React, { Component } from 'react'
import { View, Text, AppRegistry } from 'react-native'

class Example extends Component {
  render () {
    return (
      <View>
        <Text> I'm a basic Component </Text>
      </View>
    )
  }
}

AppRegistry.registerComponent('Example', () => Example)
```

Composant avec état

Ces composants auront des états changeants.

```
import React, { Component } from 'react'
import { View, Text, AppRegistry } from 'react-native'

class Example extends Component {
  constructor (props) {
    super(props)
    this.state = {
      name: "Sriraman"
    }
  }
  render () {
    return (
      <View>
        <Text> Hi, {this.state.name}</Text>
      </View>
    )
  }
}

AppRegistry.registerComponent('Example', () => Example)
```

Composant sans état

Comme leur nom l'indique, les composants sans état n'ont aucun état local. Ils sont également appelés **composants Dumb** . Sans aucun état local, ces composants n'ont pas besoin de méthodes de cycle de vie ou de la plupart des méthodes utilisées avec un composant avec état.

La syntaxe de classe n'est pas obligatoire, vous pouvez simplement faire `const name = ({props}) => ( ... )` . En règle générale, les composants sans état sont plus concis.

Ci-dessous se trouve un exemple de deux composants sans état `App` et `Title` , avec une démonstration de passerelles entre les composants:

```
import React from 'react'
import { View, Text, AppRegistry } from 'react-native'

const Title = ({Message}) => (
  <Text>{Message}</Text>
)

const App = () => (
  <View>
    <Title title='Example Stateless Component' />
  </View>
)

AppRegistry.registerComponent('App', () => App)
```

C'est le modèle recommandé pour les composants, si possible. Comme dans le futur, des optimisations peuvent être faites pour ces composants, réduisant les allocations de mémoire et les vérifications inutiles.

Lire Composants en ligne: https://riptutorial.com/fr/react-native/topic/5532/composants

Chapitre 7: Créer un partageable APK pour Android

Introduction

Étapes pour créer un APK (signé et non signé) que vous pouvez installer sur un périphérique à l'aide de l'interface de ligne de commande et partager également:

Énoncé du problème: J'ai créé mon application, je peux l'exécuter sur mon émulateur local (et aussi sur mon appareil Android en modifiant le serveur de débogage). Mais, je veux créer un fichier apk que je peux envoyer à quelqu'un sans accès au serveur de développement et je veux qu'il soit capable de tester l'application.

Remarques

Une description plus détaillée est également mentionnée ici: https://facebook.github.io/react-native/docs/signed-apk-android.html

Examples

Créer une clé pour signer l'APK

```
keytool -genkey -v -keystore my-app-key.keystore -alias my-app-alias -keyalg RSA -keysize 2048
-validity 10000
```

Utilisez un mot de passe lorsque vous y êtes invité

Une fois la clé générée, utilisez-la pour générer la version installable:

```
react-native bundle --platform android --dev false --entry-file index.android.js \
--bundle-output android/app/src/main/assets/index.android.bundle \
--assets-dest android/app/src/main/res/
```

Générer la construction en utilisant gradle

```
cd android && ./gradlew assembleRelease
```

Télécharger ou partager le fichier APK généré

Téléchargez le fichier APK sur votre téléphone. L'indicateur -r remplacera l'application existante (si elle existe)

```
adb install -r ./app/build/outputs/apk/app-release-unsigned.apk
```

L'API signable partageable se trouve à:

```
./app/build/outputs/apk/app-release.apk
```

Lire Créer un partageable APK pour Android en ligne: https://riptutorial.com/fr/react-native/topic/8964/creer-un-partageable-apk-pour-android

Chapitre 8: Disposition

Examples

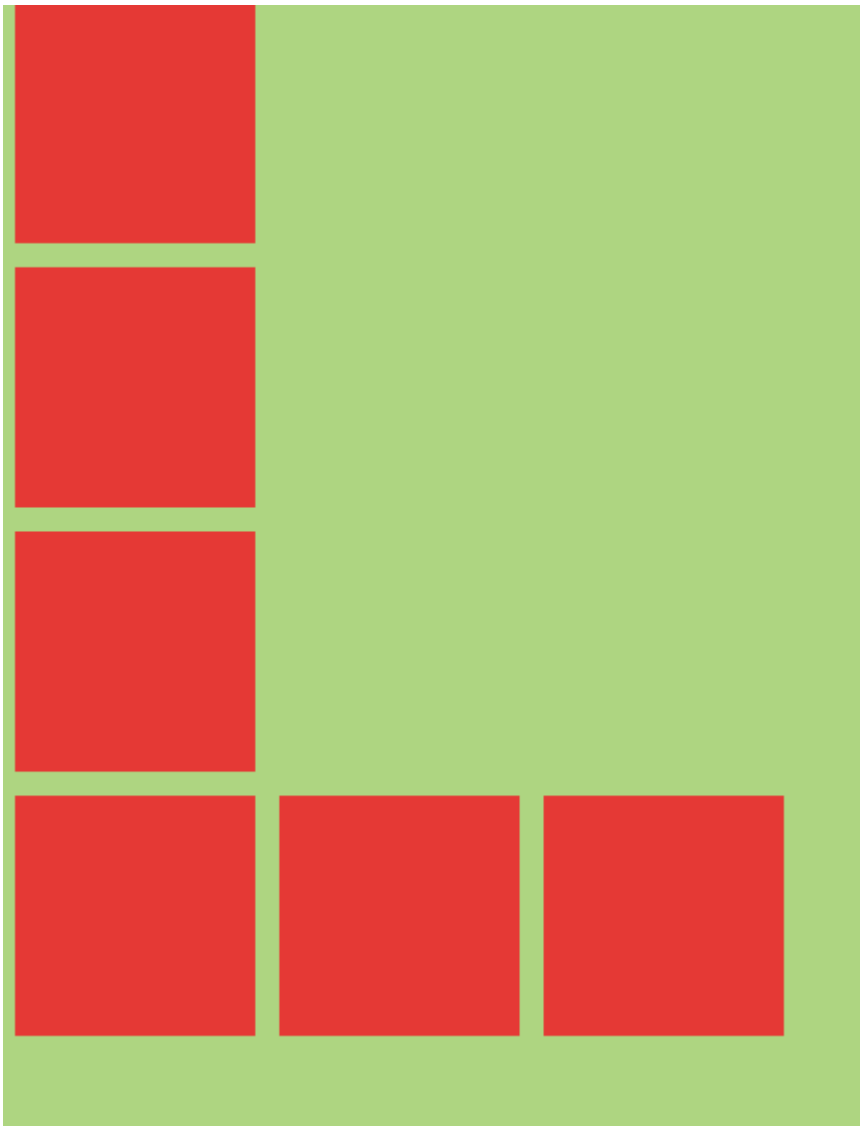
Flexbox

Flexbox est un mode de mise en page permettant la disposition des éléments sur une page de manière à ce que les éléments se comportent de manière prévisible lorsque la mise en page doit prendre en charge différentes tailles d'écran et différents périphériques d'affichage. Par défaut, flexbox organise les enfants dans une colonne. Mais vous pouvez le changer en ligne en utilisant `flexDirection: 'row'`.

flexDirection

```
const Direction = (props)=>{
  return (
    <View style={styles.container}>
      <Box/>
      <Box/>
      <Box/>
      <View style={{flexDirection:'row'}}>
        <Box/>
        <Box/>
        <Box/>
      </View>
    </View>
  )
}

const styles = StyleSheet.create({
  container: {
    flex:1,
    backgroundColor: '#AED581',
  }
});
```

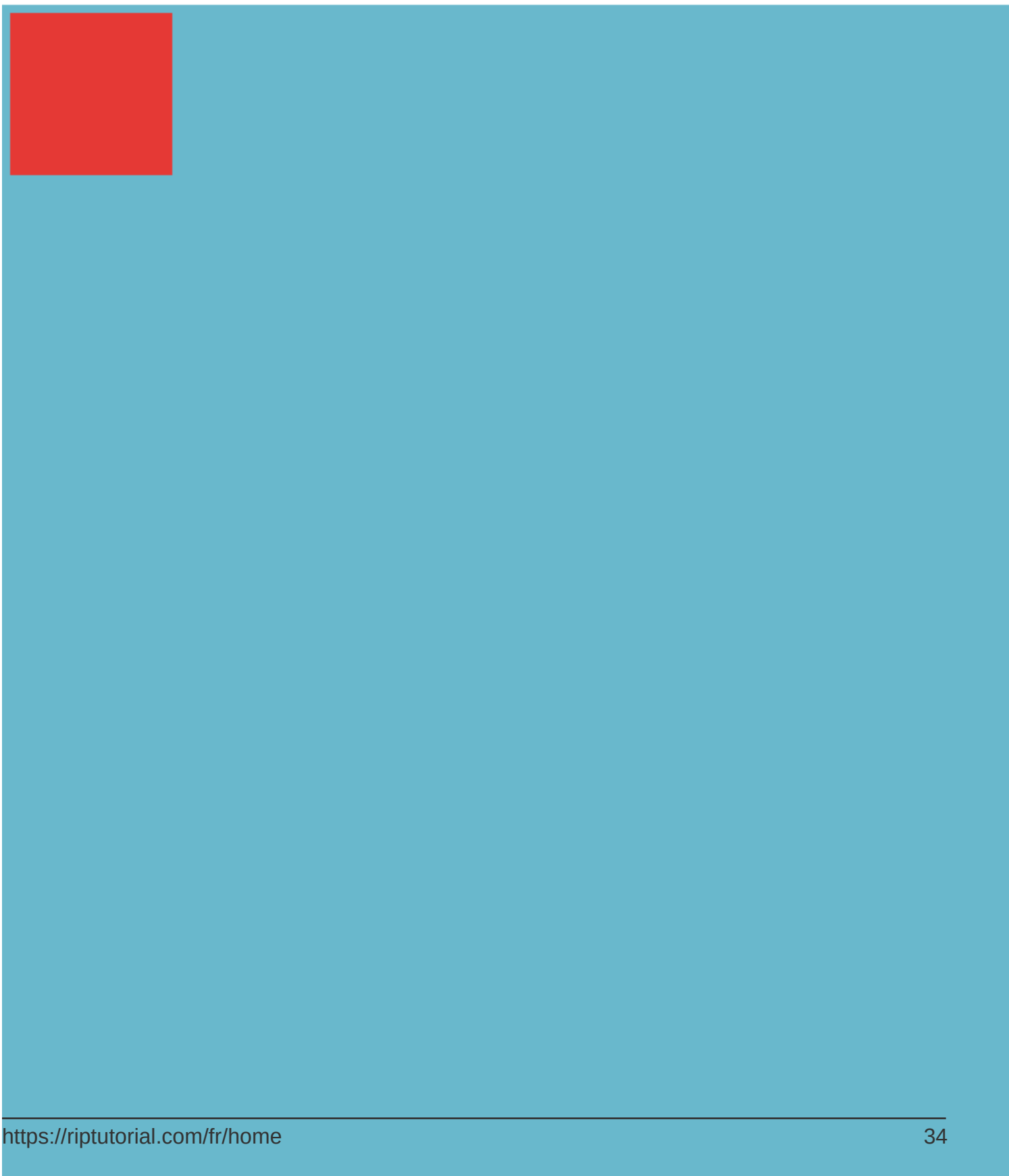


Axe d'alignement

```
const AlignmentAxis = (props)=>{
  return (
    <View style={styles.container}>
      <Box />
      <View style={{flex:1, alignItems:'flex-end', justifyContent:'flex-end'}}>
        <Box />
        <Box />
      </View>
      <Box />
    </View>
  )
}

const styles = StyleSheet.create({
  container: {
    flex:1,
    backgroundColor: `#69B8CC`,
  },
  text:{
    color: 'white',
    textAlign:'center'
  }
```

```
});
```


Carrier
8:17 PM
AlignmentAx

Chapitre 9: ESLint in react-native

Introduction

Ceci est le sujet de l'explication des règles ESLint pour rea-native.

Examples

Comment commencer

Il est fortement recommandé d'utiliser ESLint dans votre projet sur rea-native. ESLint est un outil de validation de code utilisant des règles spécifiques fournies par la communauté.

Pour rea-native, vous pouvez utiliser des ensembles de règles pour JavaScript, réagir et réagir de manière native.

Les règles communes ESLint avec motivation et explications pour javascript, vous pouvez trouver ici: https://github.com/eslint/eslint/tree/master/docs/rules . Vous pouvez simplement ajouter un jeu de règles prêt à l'emploi à partir des développeurs ESLint en ajoutant votre fichier .eslintr.json au noeud 'extend' 'eslint: recommended'. ("extend": ["eslint: recommended"]) Pour en savoir plus sur la configuration d'ESLint, vous pouvez lire ici: http://eslint.org/docs/developer-guide/development-environment . Il est recommandé de lire la documentation complète sur cet outil extrêmement utile.

Ensuite, des documents complets sur les règles pour ES Lint réagissent au plug-in que vous pouvez trouver ici: https://github.com/yannickcr/eslint-plugin-react/tree/master/docs/rules . Note importante: toutes les règles de réaction ne sont pas relatives à la réaction native. Par exemple: react / display-name et react / no-unknown-property par exemple. Une autre règle est 'must have' pour chaque projet sur react-native, tel que react / jsx-no-bind et react / jsx-key.

Soyez très prudent en choisissant votre propre jeu de règles.

Et finalement, il existe un plugin explicite pour rea-native: https://github.com/intellicode/eslint-plugin-react-native Remarque: Si vous divisez vos styles dans un fichier séparé, la règle rea-native / no-inline- les styles ne fonctionneront pas.

Pour que cet outil fonctionne correctement dans env-native env, vous devrez peut-être définir value ou "env" dans votre configuration: "env": {"browser": true, "es6": true, "amd": true} ,

ESLint est un outil clé pour le développement de produits de haute qualité.

Lire ESLint in react-native en ligne: https://riptutorial.com/fr/react-native/topic/10650/eslint-in-react-native

Chapitre 10: Etat

Syntaxe

- void setState (fonction | objet nextState, [rappel de fonction])

Examples

setState

Pour modifier la vue dans votre application, vous pouvez utiliser `setState` - cela restituera votre composant et tous ses composants enfants. setState effectue une fusion superficielle entre l'état nouveau et précédent et déclenche un re-render du composant.

`setState` prend soit un objet clé-valeur, soit une fonction qui retourne un objet clé-valeur

Objet clé-valeur

```
this.setState({myKey: 'myValue'});
```

Fonction

L'utilisation d'une fonction est utile pour mettre à jour une valeur basée sur l'état ou les accessoires existants.

```
this.setState((previousState, currentProps) => {
    return {
        myInteger: previousState.myInteger+1
    }
})
```

Vous pouvez également transmettre un rappel facultatif à `setState` qui sera déclenché lorsque le composant sera restitué avec le nouvel état.

```
this.setState({myKey: 'myValue'}, () => {
    // Component has re-rendered... do something amazing!
));
```

Exemple complet

```
import React, { Component } from 'react';
import { AppRegistry, StyleSheet, Text, View, TouchableOpacity } from 'react-native';

export default class MyParentComponent extends Component {
  constructor(props) {
    super(props);
```

```
    this.state = {
      myInteger: 0
    }

  }
  getRandomInteger() {
    const randomInt = Math.floor(Math.random()*100);

    this.setState({
      myInteger: randomInt
    });

  }
  incrementInteger() {

    this.setState((previousState, currentProps) => {
      return {
        myInteger: previousState.myInteger+1
      }
    });

  }
  render() {

    return <View style={styles.container}>

      <Text>Parent Component Integer: {this.state.myInteger}</Text>

      <MyChildComponent myInteger={this.state.myInteger} />

      <Button label="Get Random Integer" onPress={this.getRandomInteger.bind(this)} />
      <Button label="Increment Integer" onPress={this.incrementInteger.bind(this)} />

    </View>

  }
}

export default class MyChildComponent extends Component {
  constructor(props) {
    super(props);
  }
  render() {

    // this will get updated when "MyParentComponent" state changes
    return <View>
      <Text>Child Component Integer: {this.props.myInteger}</Text>
    </View>

  }
}

export default class Button extends Component {
  constructor(props) {
    super(props);
  }
  render() {

    return <TouchableOpacity onPress={this.props.onPress}>
        <View style={styles.button}>
```

```
          <Text style={styles.buttonText}>{this.props.label}</Text>
        </View>
      </TouchableOpacity>

  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
  },
  button: {
    backgroundColor: '#444',
    padding: 10,
    marginTop: 10
  },
  buttonText: {
    color: '#fff'
  }
});

AppRegistry.registerComponent('MyApp', () => MyParentComponent);
```

Initialiser l'état

Vous devez initialiser l'état dans la fonction constructeur de votre composant comme ceci:

```
export default class MyComponent extends Component {
  constructor(props) {
    super(props);

    this.state = {
      myInteger: 0
    }
  }
  render() {
    return  (
      <View>
        <Text>Integer: {this.state.myInteger}</Text>
      </View>
    )
  }
}
```

En utilisant setState, on peut mettre à jour la vue.

Lire Etat en ligne: https://riptutorial.com/fr/react-native/topic/3596/etat

Chapitre 11: Exécuter une application sur l'appareil (version Android)

Remarques

Dépannage:
`Could not connect to development server` => Procédez comme `adb reverse tcp:8081 tcp:8081` : `adb reverse tcp:8081 tcp:8081` , assurez-vous que votre téléphone est connecté (périphériques adb). Vérifiez également qu'un serveur local est lancé, sinon lancez `react-native start`

Examples

Lancer une application sur un appareil Android.

1. `adb devices`
    - Votre téléphone est-il affiché? Sinon, activez le mode développeur sur votre téléphone et connectez-le par USB.
2. `adb reverse tcp:8081 tcp:8081` :
    - Afin de relier correctement votre téléphone et que React-Native le reconnaisse pendant la construction. (**REMARQUE:** `Android Version 5` **ou supérieure.**)
3. `react-native run-android` :
    - Pour exécuter l'application sur votre téléphone.
4. `react-native start` :
    - Afin de démarrer un serveur local pour le développement (obligatoire). Ce serveur est automatiquement lancé si vous utilisez la dernière version de React-native.

Lire Exécuter une application sur l'appareil (version Android) en ligne:
https://riptutorial.com/fr/react-native/topic/5135/executer-une-application-sur-l-appareil--version-android-

Chapitre 12: Images

Examples

Module d'image

Vous allez devoir importer `Image` partir du package `react-native` , alors utilisez-le:

```
import { Image } from 'react';

<Image source={{uri: 'https://image-souce.com/awesomeImage'}} />
```

Vous pouvez également utiliser une image locale avec une syntaxe légèrement différente mais avec la même logique:

```
import { Image } from 'react';

<Image source={require('./img/myCoolImage.png')} />
```

Note: - Vous devriez donner de la hauteur, de la largeur à l'image sinon elle ne sera pas visible.

Exemple d'image

```
class ImageExample extends Component {
  render() {
    return (
      <View>
        <Image style={{width: 30, height: 30}}
          source={{uri: 'http://facebook.github.io/react/img/logo_og.png'}}
        />
      </View>
    );
  }
}
```

Source d'image conditionnelle

```
<Image style={[this.props.imageStyle]}
        source={this.props.imagePath
        ? this.props.imagePath
        : require('../theme/images/resource.png')}
    />
```

Si le chemin est disponible dans `imagePath` il sera assigné à la source sinon le chemin d'image par défaut sera assigné.

Utilisation d'une variable pour le chemin de l'image

```
let imagePath = require("../../assets/list.png");

<Image style={{height: 50, width: 50}} source={imagePath} />
```

De la ressource externe:

```
<Image style={{height: 50, width: 50}} source={{uri: userData.image}} />
```

Pour adapter une image

```
<Image
    resizeMode="contain"
    style={{height: 100, width: 100}}
    source={require('../assets/image.png')} />
```

Essayez également de **couvrir** , d' **étirer** , de **répéter** et de **centrer les** paramètres.

Lire Images en ligne: https://riptutorial.com/fr/react-native/topic/3956/images

Chapitre 13: Instructions en ligne de commande

Examples

Vérifier la version installée

```
$ react-native -v
```

Exemple de sortie

```
react-native-cli: 0.2.0
react-native: n/a - not inside a React Native project directory //Output from  different
folder
react-native: react-native: 0.30.0 // Output from the react native project directory
```

Mettre à niveau le projet existant vers la dernière version RN

Dans le dossier app, recherchez `package.json` et modifiez la ligne suivante pour inclure la dernière version, enregistrez le fichier et fermez-le.

```
"react-native": "0.32.0"
```

Dans le terminal:

```
$ npm install
```

Suivi par

```
$ react-native upgrade
```

Enregistrement

Android

```
$ react-native log-android
```

iOS

```
$ react-native log-ios
```

Initialiser et démarrer avec le projet React Native

Pour initialiser

```
react-native init MyAwesomeProject
```

Pour initialiser avec une version spécifique de React Native

```
react-native init --version="0.36.0" MyAwesomeProject
```

Pour courir pour Android

```
cd MyAwesomeProject
react-native run-android
```

Pour exécuter pour iOS

```
cd MyAwesomeProject
react-native run-ios
```

Démarrer React Native Packager

```
$ react-native start
```

Sur la dernière version de React Native, il n'est pas nécessaire d'exécuter le conditionneur. Il fonctionnera automatiquement.

Par défaut, le serveur démarre sur le port 8081. Pour spécifier le port sur lequel le serveur est installé

```
$ react-native start --port PORTNUMBER
```

Ajouter un projet Android pour votre application

Si vous avez soit des applications générées avec le support pré-Android, soit vous les avez faites exprès, vous pouvez toujours ajouter un projet Android à votre application.

```
$ react-native android
```

Cela générera un dossier `android` et `index.android.js` intérieur de votre application.

Lire Instructions en ligne de commande en ligne: https://riptutorial.com/fr/react-native/topic/2117/instructions-en-ligne-de-commande

Chapitre 14: Intégration avec Firebase pour l'authentification

Introduction

// Remplacez les valeurs de la base de feu par les valeurs de votre application api importez la base de données depuis 'firebase';

componentWillMount () {firebase.initializeApp ({apiKey: "yourAPIKey", authDomain: "authDomainNAme", databaseURL: "yourDomainBaseURL", ID du projet: "yourProjectID", storageBucket: "storageBUcketValue", messagingSenderId: "senderIdValue"}); firebase.auth (). signInWithEmailAndPassword (email, mot de passe) .then (this.onLoginSuccess)})}

Examples

React Native - ListView avec Firebase

C'est ce que je fais lorsque je travaille avec Firebase et que je veux utiliser ListView.

Utilisez un composant parent pour récupérer les données de Firebase (Posts.js):

Posts.js

```
import PostsList from './PostsList';

class Posts extends Component{
    constructor(props) {
        super(props);
        this.state = {
            posts: []
        }
    }

    componentWillMount() {
        firebase.database().ref('Posts/').on('value', function(data) {
            this.setState({ posts: data.val() });
        });
    }

    render() {
        return <PostsList posts={this.state.posts}/>
    }
}
```

PostsList.js

```
class PostsList extends Component {
    constructor(props) {
        super(props);
```

```
        this.state = {
            dataSource: new ListView.DataSource({
                rowHasChanged: (row1, row2) => row1 !== row2
            }),
        }
    }

    getDataSource(posts: Array<any>): ListView.DataSource {
        if(!posts) return;
        return this.state.dataSource.cloneWithRows(posts);
    }

    componentDidMount() {
        this.setState({dataSource: this.getDataSource(this.props.posts)});
    }

    componentWillReceiveProps(props) {
        this.setState({dataSource: this.getDataSource(props.posts)});
    }

    renderRow = (post) => {
        return (
            <View>
                <Text>{post.title}</Text>
                <Text>{post.content}</Text>
            </View>
        );
    }

    render() {
        return(
            <ListView
                dataSource={this.state.dataSource}
                renderRow={this.renderRow}
                enableEmptySections={true}
            />
        );
    }
}
```

Je tiens à souligner que dans `Posts.js` , je `Posts.js` pas de `firebase` car il vous suffit de l'importer une fois, dans le composant principal de votre projet (où se trouve le navigateur) et de l'utiliser n'importe où.

C'est la solution que quelqu'un a suggérée dans une question que j'ai posée quand je me débattais avec ListView. Je pensais que ce serait bien de le partager.

Source: [http://stackoverflow.com/questions/38414289/react-native-listview-not-rendering-data-from-firebase][1]

Authentification dans React Native avec Firebase

Remplacez les valeurs de la base de feu par les valeurs de votre application api:

```
import firebase from 'firebase';
componentWillMount() {
firebase.initializeApp({
```

```
  apiKey: "yourAPIKey",
  authDomain: "authDomainNAme",
  databaseURL: "yourDomainBaseURL",
  projectId: "yourProjectID",
  storageBucket: "storageBUcketValue",
  messagingSenderId: "senderIdValue"
});
    firebase.auth().signInWithEmailAndPassword(email, password)
  .then(this.onLoginSuccess)
  .catch(() => {
    firebase.auth().createUserWithEmailAndPassword(email, password)
      .then(this.onLoginSuccess)
      .catch(this.onLoginFail)
  })
}
```

Lire Intégration avec Firebase pour l'authentification en ligne: https://riptutorial.com/fr/react-native/topic/6391/integration-avec-firebase-pour-l-authentification

Chapitre 15: Le débogage

Syntaxe

- débogueur

Examples

Démarrez le débogage JS à distance dans Android

Vous pouvez démarrer le débogage à distance à partir du menu Developer. Après avoir sélectionné l'option Déboguer à distance, il ouvrira Google Chrome, afin que vous puissiez enregistrer la sortie dans votre console. Vous pouvez également écrire la syntaxe du débogueur dans votre code js.

Utiliser console.log ()

Vous pouvez imprimer le message de journal dans le terminal en utilisant `console.log()` . Pour ce faire, ouvrez un nouveau terminal et exécutez la commande suivante pour Android:

```
react-native log-android
```

ou suivant la commande si vous utilisez iOS:

```
react-native log-ios
```

Vous allez maintenant commencer à voir tout le message de journal dans ce terminal

Lire Le débogage en ligne: https://riptutorial.com/fr/react-native/topic/5105/le-debogage

Chapitre 16: Le routage

Introduction

Le routage ou la navigation permet des applications entre différents écrans. Son vital pour une application mobile car il fournit un contexte à l'utilisateur sur l'endroit où il se trouve, dissocie les actions de l'utilisateur entre les écrans et se déplace entre eux, fournit une machine d'état comme le modèle de l'application entière.

Examples

Composant Navigateur

Navigator fonctionne pour iOS et Android.

```
import React, { Component } from 'react';
import { Text, Navigator, TouchableHighlight } from 'react-native';

export default class NavAllDay extends Component {
  render() {
    return (
      <Navigator
        initialRoute={{ title: 'Awesome Scene', index: 0 }}
        renderScene={(route, navigator) =>
          <Text>Hello {route.title}!</Text>
        }
        style={{padding: 100}}
      />
    );
  }
}
```

Les itinéraires vers le `Navigator` sont fournis sous forme d'objets. Vous fournissez également une fonction `renderScene` qui rend la scène pour chaque objet route. `initialRoute` est utilisé pour spécifier le premier itinéraire.

Lire Le routage en ligne: https://riptutorial.com/fr/react-native/topic/8279/le-routage

Chapitre 17: Les accessoires

Introduction

Les accessoires ou propriétés sont des données transmises aux composants enfants dans une application React. Les composants React rendent les éléments d'interface utilisateur basés sur leurs accessoires et leur état interne. Les accessoires qu'un composant prend (et utilise) définit comment il peut être contrôlé de l'extérieur.

Examples

Quels sont les accessoires?

Les accessoires permettent de transférer des données d'un composant parent à un composant enfant. Les accessoires sont en lecture seule. Le composant enfant peut uniquement obtenir les propriétés transmises par le parent en utilisant **this.props.keyName** . En utilisant des accessoires, on peut rendre son composant réutilisable.

Utilisation des accessoires

Une fois l'installation terminée. Copiez le code ci-dessous dans `index.android.js` ou dans le fichier `index.ios.js` pour utiliser les accessoires.

```
import React, { Component } from 'react';
import { AppRegistry, Text, View } from 'react-native';

class Greeting extends Component {
  render() {
    return (
      <Text>Hello {this.props.name}!</Text>
    );
  }
}

class LotsOfGreetings extends Component {
  render() {
    return (
      <View style={{alignItems: 'center'}}>
        <Greeting name='Rexxar' />
        <Greeting name='Jaina' />
        <Greeting name='Valeera' />
      </View>
    );
  }
}

AppRegistry.registerComponent('LotsOfGreetings', () => LotsOfGreetings);
```

En utilisant des accessoires, on peut rendre son composant générique. Par exemple, vous avez un composant Button. Vous pouvez transmettre différents accessoires à ce composant afin de

pouvoir placer ce bouton n'importe où dans sa vue.

source: Props-React Native

PropTypes

Le package `prop-types` vous permet d'ajouter une vérification de type à l'exécution à votre composant pour vous assurer que les types des accessoires transmis au composant sont corrects. Par exemple, si vous ne transmettez pas un `name` ou `isYummy` prop au composant ci-dessous, une erreur se produira en mode développement. En mode production, les vérifications de type prop ne sont pas effectuées. La définition de `propTypes` peut rendre votre composant plus lisible et maintenable.

```
import React, { Component } from 'react';
import PropTypes from 'prop-types';
import { AppRegistry, Text, View } from 'react-native';

import styles from './styles.js';

class Recipe extends Component {
  static propTypes = {
    name: PropTypes.string.isRequired,
    isYummy: PropTypes.bool.isRequired
  }
  render() {
    return (
      <View style={styles.container}>
        <Text>{this.props.name}</Text>
        {this.props.isYummy ? <Text>THIS RECIPE IS YUMMY</Text> : null}
      </View>
    )
  }
}

AppRegistry.registerComponent('Recipe', () => Recipe);


// Using the component
<Recipe name="Pancakes" isYummy={true} />
```

Plusieurs PropTypes

Vous pouvez également avoir plusieurs `propTypes` pour un des accessoires. Par exemple, le nom des accessoires que je prends peut aussi être un objet, je peux l'écrire comme.

```
static propTypes = {
  name: PropTypes.oneOfType([
      PropTypes.string,
      PropTypes.object
  ])
}
```

Accessoires pour enfants

Il y a aussi des accessoires spéciaux appelés `children` , qui **ne** sont **pas** transmis comme

```
<Recipe children={something}/>
```

Au lieu de cela, vous devriez le faire

```
<Recipe>
  <Text>Hello React Native</Text>
</Recipe>
```

alors vous pouvez le faire dans le rendu de la recette:

```
return (
  <View style={styles.container}>
    {this.props.children}
    {this.props.isYummy ? <Text>THIS RECIPE IS YUMMY</Text> : null}
  </View>
)
```

Vous aurez un composant `<Text>` dans votre `Recipe` disant `Hello React Native` , plutôt cool hum?

Et le propType des enfants est

```
children: PropTypes.node
```

Accessoires par défaut

defaultProps vous permet de définir des valeurs par défaut pour votre composant. Dans l'exemple ci-dessous si vous ne passez pas le nom props, il affichera John sinon il affichera la valeur passée

```
class Example extends Component {
  render() {
    return (
      <View>
        <Text>{this.props.name}</Text>
      </View>
    )
  }
}


Example.defaultProps = {
  name: 'John'
}
```

Lire Les accessoires en ligne: https://riptutorial.com/fr/react-native/topic/1271/les-accessoires

Chapitre 18: Liaison de l'API native

Introduction

L'API de liaison vous permet d'envoyer et de recevoir des liens entre les applications. Par exemple, ouvrez l'application Téléphone avec le numéro composé ou ouvrez Google Maps et lancez une navigation vers la destination choisie. Vous pouvez également utiliser Linking pour que votre application puisse répondre aux liens qui l'ouvrent depuis d'autres applications.

Pour utiliser `Linking` vous devez d'abord l'importer depuis `react-native`

```
import {Linking} from 'react-native'
```

Examples

Liens sortants

Pour ouvrir un appel de lien openURL.

```
Linking.openURL(url)
.catch(err => console.error('An error occurred ', err))
```

La méthode préférée consiste à vérifier si une application installée peut gérer une URL donnée au préalable.

```
Linking.canOpenURL(url)
.then(supported => {
  if (!supported) {
    console.log('Unsupported URL: ' + url)
  } else {
    return Linking.openURL(url)
  }
}).catch(err => console.error('An error occurred ', err))
```

Schémas d'URI

Application cible	Exemple	Référence
Navigateur Web	`https://stackoverflow.com`	
Téléphone	`tel:1-408-555-5555`	Pomme
Courrier	`mailto:email@example.com`	Pomme
SMS	`sms:1-408-555-1212`	Pomme

Application cible	Exemple	Référence
Apple Maps	`http://maps.apple.com/?ll=37.484847,-122.148386`	Pomme
Google Maps	`geo:37.7749,-122.4194`	Google
iTunes	Voir iTunes Link Maker	Pomme
Facebook	`fb://profile`	Débordement de pile
Youtube	`http://www.youtube.com/v/oHg5SJYRHA0`	Pomme
Facetime	`facetime://user@example.com`	Pomme
Calendrier iOS	`calshow:514300000` [1]	iPhoneDevWiki

[1] Ouvre le calendrier au nombre de secondes indiqué depuis le 1. 1. 2001 (UTC?). Pour une raison quelconque, cette API n'est pas documentée par Apple.

Liens entrants

Vous pouvez détecter le lancement de votre application à partir d'une URL externe.

```
componentDidMount() {
  const url = Linking.getInitialURL()
  .then((url) => {
    if (url) {
      console.log('Initial url is: ' + url)
    }
  }).catch(err => console.error('An error occurred ', err))
}
```

Pour activer cela sur iOS Link `RCTLinking` à votre projet .

Pour activer cela sur Android, procédez comme suit .

Lire Liaison de l'API native en ligne: https://riptutorial.com/fr/react-native/topic/9687/liaison-de-l-api-native

Chapitre 19: ListView

Examples

Exemple simple

ListView - Composant central conçu pour un affichage efficace des listes de données changeantes défilant verticalement. L'API minimale consiste à créer un ListView.DataSource, à le remplir avec un simple tableau de blobs de données et à instancier un composant ListView avec cette source de données et un rappel renderRow qui prend un blob du tableau de données et renvoie un composant rendu.

Exemple minimal:

```
getInitialState: function() {
  var ds = new ListView.DataSource({rowHasChanged: (r1, r2) => r1 !== r2});
  return {
    dataSource: ds.cloneWithRows(['row 1', 'row 2']),
  };
},

render: function() {
  return (
    <ListView
      dataSource={this.state.dataSource}
      renderRow={(rowData) => <Text>{rowData}</Text>}
    />
  );
},
```

ListView prend également en charge des fonctionnalités plus avancées, notamment des sections avec des en-têtes de sections, des en-têtes et des pieds de page, des rappels à la fin des données disponibles (onEndReached) et l'ensemble des lignes visibles dans onport change plusieurs optimisations de performances.

Il y a quelques opérations de performance conçues pour que ListView défile en douceur tout en chargeant dynamiquement des ensembles de données potentiellement très volumineux (ou conceptuellement infinis):

- Ne restituez que les lignes modifiées - la fonction rowHasChanged fournie à la source de données indique à ListView si elle doit redéfinir une ligne car les données source ont été modifiées - voir ListViewDataSource pour plus de détails.
- Rendu de ligne limité par le débit - Par défaut, une seule ligne est rendue par boucle d'événement (personnalisable avec le prop de pageSize). Cela divise le travail en morceaux plus petits afin de réduire les risques de suppression d'images lors du rendu des lignes.

Lire ListView en ligne: https://riptutorial.com/fr/react-native/topic/3112/listview

Chapitre 20: Meilleures pratiques de rendu

Introduction

Sujet pour des notes importantes sur le comportement spécifique de la méthode Component.render.

Examples

Fonctions dans JSX

Pour de meilleures performances, il est important d'éviter d'utiliser la fonction array (lambda) dans JSX.

Comme expliqué à l' adresse https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-no-bind.md :

> Un appel de liaison ou une fonction de flèche dans un accessoire JSX créera une nouvelle fonction sur chaque rendu. Cela est mauvais pour les performances, car le récupérateur de mémoire sera plus sollicité que nécessaire. Cela peut également entraîner des rendus inutiles si une nouvelle fonction est transmise en tant que prop à un composant qui utilise la vérification d'égalité de référence sur le prop pour déterminer s'il doit être mis à jour.

Donc, si jsx bloque le code comme ceci:

```
<TextInput
  onChangeValue={  value => this.handleValueChanging(value) }
/>
```

ou

```
<button onClick={ this.handleClick.bind(this) }></button>
```

vous pouvez le rendre meilleur:

```
<TextInput
  onChangeValue={  this.handleValueChanging }
/>
```

et

```
<button onClick={ this.handleClick }></button>
```

Pour un contexte correct dans la fonction handleValueChanging, vous pouvez l'appliquer dans le constructeur du composant:

```
constructor(){
  this.handleValueChanging = this.handleValueChanging.bind(this)
}
```

plus en liant une fonction passée à un composant

Ou vous pouvez utiliser des solutions comme celle-ci: https://github.com/andreypopp/autobind-decorator et ajouter simplement @autobind decorator à chaque méthode à laquelle vous souhaitez vous connecter:

```
@autobind
handleValueChanging(newValue)
{
    //processing event
}
```

Lire Meilleures pratiques de rendu en ligne: https://riptutorial.com/fr/react-native/topic/10649/meilleures-pratiques-de-rendu

Chapitre 21: Meilleures pratiques du navigateur

Examples

Navigateur

`Navigator` est le navigateur par défaut de React Native. Un composant `Navigator` gère une *pile* d'objets route et fournit des méthodes pour gérer cette pile.

```
<Navigator
  ref={(navigator) => { this.navigator = navigator }}
  initialRoute={{ id: 'route1', title: 'Route 1' }}
  renderScene={this.renderScene.bind(this)}
  configureScene={(route) => Navigator.SceneConfigs.FloatFromRight}
  style={{ flex: 1 }}
  navigationBar={
    // see "Managing the Navigation Bar" below
    <Navigator.NavigationBar routeMapper={this.routeMapper} />
  }
/>
```

Gestion de la pile de route

Tout d'abord, notez l'accessoire `initialRoute` . Une route est simplement un objet javascript, et peut prendre la forme que vous voulez et avoir les valeurs que vous voulez. C'est la principale manière de transmettre des valeurs et des méthodes entre les composants de votre pile de navigation.

Le `Navigator` sait quoi restituer en fonction de la valeur renvoyée par son outil `renderScene` .

```
renderScene(route, navigator) {
  if (route.id === 'route1') {
    return <ExampleScene navigator={navigator} title={route.title} />; // see below
  } else if (route.id === 'route2') {
    return <ExampleScene navigator={navigator} title={route.title} />; // see below
  }
}
```

Imaginons une implémentation de `ExampleScene` dans cet exemple:

```
function ExampleScene(props) {

  function forward() {
    // this route object will passed along to our `renderScene` function we defined above.
    props.navigator.push({ id: 'route2', title: 'Route 2' });
  }

  function back() {
    // `pop` simply pops one route object off the `Navigator`'s stack
```

```
    props.navigator.pop();
  }

  return (
    <View>
      <Text>{props.title}</Text>
      <TouchableOpacity onPress={forward}>
        <Text>Go forward!</Text>
      </TouchableOpacity>
      <TouchableOpacity onPress={back}>
        <Text>Go Back!</Text>
      </TouchableOpacity>
    </View>
  );
}
```

Configuration du navigateur

Vous pouvez configurer les transitions du `Navigator` avec le composant `configureScene` . Ceci est une fonction qui a passé l'objet `route` et doit retourner un objet de configuration. Voici les objets de configuration disponibles:

- Navigator.SceneConfigs.PushFromRight (par défaut)
- Navigator.SceneConfigs.FloatFromRight
- Navigator.SceneConfigs.FloatFromLeft
- Navigator.SceneConfigs.FloatFromBottom
- Navigator.SceneConfigs.FloatFromBottomAndroid
- Navigator.SceneConfigs.FadeAndroid
- Navigator.SceneConfigs.HorizontalSwipeJump
- Navigator.SceneConfigs.HorizontalSwipeJumpFromRight
- Navigator.SceneConfigs.VerticalUpSwipeJump
- Navigator.SceneConfigs.VerticalDownSwipeJump

Vous pouvez renvoyer l'un de ces objets sans modification ou modifier l'objet de configuration pour personnaliser les transitions de navigation. Par exemple, pour modifier la largeur des `UINavigationController` aux bords pour émuler plus étroitement le `UINavigationController` `interactivePopGestureRecognizer` `UINavigationController` iOS:

```
configureScene={(route) => {
  return {
    ...Navigator.SceneConfigs.FloatFromRight,
    gestures: {
      pop: {
        ...Navigator.SceneConfigs.FloatFromRight.gestures.pop,
        edgeHitWidth: Dimensions.get('window').width / 2,
      },
    },
  };
}}
```

Gestion de la barre de navigation

Le composant `Navigator` est fourni avec un élément de `navigationBar` qui peut théoriquement

prendre tout composant React correctement configuré. Mais l'implémentation la plus courante utilise le `Navigator.NavigationBar` par défaut. Cela prend un accessoire de `routeMapper` que vous pouvez utiliser pour configurer l'apparence de la barre de navigation en fonction de l'itinéraire.

Un `routeMapper` est un objet JavaScript classique avec trois fonctions: `Title` , `RightButton` et `LeftButton` . Par exemple:

```
const routeMapper = {

  LeftButton(route, navigator, index, navState) {
    if (index === 0) {
      return null;
    }

    return (
      <TouchableOpacity
        onPress={() => navigator.pop()}
        style={styles.navBarLeftButton}
      >
        <Text>Back</Text>
      </TouchableOpacity>
    );
  },

  RightButton(route, navigator, index, navState) {
    return (
      <TouchableOpacity
        onPress={route.handleRightButtonClick}
        style={styles.navBarRightButton}
      >
        <Text>Next</Text>
      </TouchableOpacity>
   );
  },

  Title(route, navigator, index, navState) {
    return (
      <Text>
        {route.title}
      </Text>
    );
  },
};
```

Voir plus

Pour une documentation plus détaillée de chaque accessoire, consultez la documentation officielle React pour `Navigator` et le guide React Native sur l' utilisation des navigateurs .

Utiliser rea-navigation pour naviguer dans les applications natives de réaction

Avec l'aide de react-navigation , vous pouvez ajouter très facilement la navigation à votre application.

Installer react-navigation

```
npm install --save react-navigation
```

Exemple:

```
import { Button, View, Text, AppRegistry } from 'react-native';
import { StackNavigator } from 'react-navigation';

const App = StackNavigator({
  FirstPage: {screen: FirstPage},
  SecondPage: {screen: SecondPage},
});

class FirstPage extends React.Component {
  static navigationOptions = {
    title: 'Welcome',
  };
  render() {
    const { navigate } = this.props.navigation;

    return (
      <Button
        title='Go to Second Page'
        onPress={() =>
          navigate('SecondPage', { name: 'Awesomepankaj' })
        }
      />
    );
  }
}

class SecondPage extends React.Component {
  static navigationOptions = ({navigation}) => ({
    title: navigation.state.params.name,
  });

  render() {
    const { goBack } = this.props.navigation;
    return (
      <View>
        <Text>Welcome to Second Page</Text>
        <Button
          title="Go back to First Page"
          onPress={() => goBack()}
        />
      </View>
    );
  }
}
```

Navigation ré-native avec react-native-router-flux

Installez en utilisant `npm install --save react-native-router-flux`

Dans rea-native-routeur-flux, chaque route est appelée `<Scene>`

```
    <Scene key="home" component={LogIn} title="Home" initial />
```

`key` Chaîne unique pouvant être utilisée pour faire référence à la scène particulière.

`component` Quel composant montrer, ici c'est

`title` faire un NavBar et lui donner un titre 'Home'

`initial` Est-ce le premier écran de l'application

Exemple:

```
import React from 'react';
import { Scene, Router } from 'react-native-router-flux';
import LogIn from './components/LogIn';
import SecondPage from './components/SecondPage';

const RouterComponent = () => {
  return (
    <Router>
      <Scene key="login" component={LogIn} title="Login Form" initial />
      <Scene key="secondPage" component={SecondPage} title="Home" />
    </Router>
  );
};

export default RouterComponent;
```

Importez ce fichier dans le fichier principal App.js (fichier d'index) et rendez-le. Pour plus d'informations, visitez ce lien .

Lire Meilleures pratiques du navigateur en ligne: https://riptutorial.com/fr/react-native/topic/2559/meilleures-pratiques-du-navigateur

Chapitre 22: Modal

Introduction

Le composant modal est un moyen simple de présenter du contenu au-dessus d'une vue englobante.

Paramètres

Soutenir	détails
animationType	c'est un enum de (" **none** ", " **slide** ", " **fade** ") et il contrôle l'animation modale.
visible	c'est un bool qui contrôle la visibilité modale.
onShow	il permet de passer une fonction qui sera appelée une fois le modal affiché.
transparent	bool pour définir la transparence.
onRequestClose (**android**)	il définit toujours une méthode qui sera appelée lorsque l'utilisateur retourne le bouton
onOrientationChange (**IOS**)	il définit toujours une méthode qui sera appelée lorsque l'orientation change
SupportedOrientations (**IOS**)	enum («portrait», «portrait à l'envers», «paysage», «paysage à gauche», «paysage à droite»)

Examples

Exemple de base modal

```
import React, { Component } from 'react';
import {
  Modal,
  Text,
  View,
  Button,
  StyleSheet,
} from 'react-native';

const styles = StyleSheet.create({
  mainContainer: {
    marginTop: 22,
  },
```

```
  modalContainer: {
    marginTop: 22,
  },
});

class Example extends Component {
  constructor() {
    super();
    this.state = {
      visibility: false,
    };
  }


  setModalVisibility(visible) {
    this.setState({
      visibility: visible,
    });
  }

  render() {
    return (
      <View style={styles.mainContainer}>
        <Modal
          animationType={'slide'}
          transparent={false}
          visible={this.state.visibility}
        >
          <View style={styles.modalContainer}>
            <View>
              <Text>I'm a simple Modal</Text>
              <Button
                color="#000"
                onPress={() => this.setModalVisibility(!this.state.visibility)}
                title="Hide Modal"
              />
            </View>
          </View>
        </Modal>

        <Button
          color="#000"
          onPress={() => this.setModalVisibility(true)}
          title="Show Modal"
        />
      </View>
    );
  }
}

export default Example;
```

Exemple modal transparent

Voir cet exemple ici .

```
import React, { Component } from 'react';
import { Text, View, StyleSheet, Button, Modal } from 'react-native';
import { Constants } from 'expo';
```

```
export default class App extends Component {
  state = {
    modalVisible: false,
  };

  _handleButtonPress = () => {
    this.setModalVisible(true);
  };

  setModalVisible = (visible) => {
    this.setState({modalVisible: visible});
  }

  render() {
    var modalBackgroundStyle = {
      backgroundColor: 'rgba(0, 0, 0, 0.5)'
    };
    var innerContainerTransparentStyle = {backgroundColor: '#fff', padding: 20};
    return (
      <View style={styles.container}>
      <Modal
          animationType='fade'
          transparent={true}
          visible={this.state.modalVisible}
          onRequestClose={() => this.setModalVisible(false)}
          >
          <View style={[styles.container, modalBackgroundStyle]}>
            <View style={innerContainerTransparentStyle}>
              <Text>This is a modal</Text>
              <Button title='close'
                onPress={this.setModalVisible.bind(this, false)}/>
            </View>
          </View>
        </Modal>
        <Button
          title="Press me"
          onPress={this._handleButtonPress}
        />

      </View>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
    paddingTop: Constants.statusBarHeight,
    backgroundColor: '#ecf0f1',
  }
});
```

Lire Modal en ligne: https://riptutorial.com/fr/react-native/topic/8253/modal

Chapitre 23: Module de plate-forme

Examples

Trouver le type / version du système d'exploitation

La première étape consiste à importer la plate-forme à partir du package «réagit nativement» comme suit:

```
import { Platform } from 'react-native'
```

Après avoir fait cela, vous pouvez aller de l'avant et accéder au type de système d'exploitation via `Platform.OS` vous permettant de l'utiliser dans des instructions conditionnelles comme

```
const styles = StyleSheet.create({
  height: (Platform.OS === 'ios') ? 200 : 100,
})
```

Si vous souhaitez détecter la version Android, vous pouvez utiliser `Platform.Version` comme ceci:

```
if (Platform.Version === 21) {
  console.log('Running on Lollipop!');
}
```

Pour iOS, Platform.Version renvoie une chaîne, pour des conditions complexes, n'oubliez pas de l'analyser.

```
if (parseInt(Platform.Version, 10) >= 9) {
    console.log('Running version higher than 8');
}
```

Si la logique spécifique à la plate-forme est complexe, il est possible de rendre deux fichiers différents basés sur la plate-forme. Ex:

- `MyTask.android.js`
- `MyTask.ios.js`

et l'exiger en utilisant

```
const MyTask = require('./MyTask')
```

Lire Module de plate-forme en ligne: https://riptutorial.com/fr/react-native/topic/3593/module-de-plate-forme

Chapitre 24: Modules natifs

Examples

Créez votre module natif (IOS)

introduction

à partir de http://facebook.github.io/react-native/docs/native-modules-ios.html

> Parfois, une application a besoin d'accéder à l'API de la plate-forme et React Native n'a pas encore de module correspondant. Vous souhaitez peut-être réutiliser un code Objective-C, Swift ou C ++ existant sans avoir à le réimplémenter en JavaScript ou écrire du code multi-thread performant, tel que traitement d'image, base de données ou nombre d'extensions avancées.

Un module natif est simplement une classe Objective-C qui implémente le protocole `RCTBridgeModule` .

Exemple

Dans votre projet Xcode, créez un nouveau fichier et sélectionnez **Cocoa Touch Class** . Dans l'assistant de création, choisissez un nom pour votre classe (*par exemple, NativeModule*), faites-en une **sous - classe de** `NSObject` et choisissez `Objective-C` .

Cela créera deux fichiers `NativeModuleEx.h` et `NativeModuleEx.m`

Vous devrez importer `RCTBridgeModule.h` dans votre fichier `NativeModuleEx.h` comme suit:

```
#import <Foundation/Foundation.h>
#import "RCTBridgeModule.h"

@interface NativeModuleEx : NSObject <RCTBridgeModule>

@end
```

Dans votre `NativeModuleEx.m` ajoutez le code suivant:

```
#import "NativeModuleEx.h"

@implementation NativeModuleEx

RCT_EXPORT_MODULE();

RCT_EXPORT_METHOD(testModule:(NSString *)string )
```

```
{
  NSLog(@"The string '%@' comes from JavaScript! ", string);
}

@end
```

`RCT_EXPORT_MODULE()` rendra votre module accessible en JavaScript, vous pouvez lui transmettre un argument facultatif pour spécifier son nom. Si aucun nom n'est fourni, il correspondra au nom de la classe Objective-C.

`RCT_EXPORT_METHOD()` exposera votre méthode à JavaScript, seules les méthodes que vous exportez à l'aide de cette macro seront accessibles en JavaScript.

Enfin, dans votre JavaScript, vous pouvez appeler votre méthode comme suit:

```
import { NativeModules } from 'react-native';

var NativeModuleEx = NativeModules.NativeModuleEx;

NativeModuleEx.testModule('Some String !');
```

Lire Modules natifs en ligne: https://riptutorial.com/fr/react-native/topic/6155/modules-natifs

Chapitre 25: Navigateur avec boutons injectés depuis les pages

Examples

introduction

Au lieu de gonfler votre fichier js principal contenant votre navigateur avec des boutons. Il est plus simple d'injecter des boutons à la demande dans n'importe quelle page dont vous avez besoin.

```
//In the page "Home", I want to have the right nav button to show
//a settings modal that resides in "Home" component.

componentWillMount() {
  this.props.route.navbarTitle = "Home";

  this.props.route.rightNavButton = {
    text: "Settings",
    onPress: this._ShowSettingsModal.bind(this)
  };
}
```

Exemple complet commenté

```
'use strict';

import React, {Component} from 'react';
import ReactNative from 'react-native';

const {
  AppRegistry,
  StyleSheet,
  Text,
  View,
  Navigator,
  Alert,
  TouchableHighlight
} = ReactNative;


//This is the app container that contains the navigator stuff
class AppContainer extends Component {

    renderScene(route, navigator) {
        switch(route.name) {
            case "Home":
      //You must pass route as a prop for this trick to work properly
            return <Home route={route} navigator={navigator} {...route.passProps}  />
            default:
            return (
        <Text route={route}
        style={styles.container}>
```

```
            Your route name is probably incorrect {JSON.stringify(route)}
            </Text>
      );
        }
    }

  render() {
    return (
      <Navigator
        navigationBar={
          <Navigator.NavigationBar
            style={ styles.navbar }
            routeMapper={ NavigationBarRouteMapper } />
        }

        initialRoute={{ name: 'Home' }}
        renderScene={ this.renderScene }

      />
    );
  }
}


//Nothing fancy here, except for checking for injected buttons.
//Notice how we are checking if there are injected buttons inside the route object.
//Also, we are showing a "Back" button when the page is not at index-0 (e.g. not home)
var NavigationBarRouteMapper = {
  LeftButton(route, navigator, index, navState) {
    if(route.leftNavButton) {
      return (
        <TouchableHighlight
        style={styles.leftNavButton}
        underlayColor="transparent"
        onPress={route.leftNavButton.onPress}>
          <Text style={styles.navbarButtonText}>{route.leftNavButton.text}</Text>
        </TouchableHighlight>
      );
    }
    else if(route.enableBackButton) {
      return (
        <TouchableHighlight
        style={styles.leftNavButton}
        underlayColor="transparent"
        onPress={() => navigator.pop() }>
          <Text style={styles.navbarButtonText}>Back</Text>
        </TouchableHighlight>
      );
    }
  },
  RightButton(route, navigator, index, navState) {
    if(route.rightNavButton) {
      return (
        <TouchableHighlight
        style={styles.rightNavButton}
        underlayColor="transparent"
        onPress={route.rightNavButton.onPress}>
          <Text style={styles.navbarButtonText}>{route.rightNavButton.text}</Text>
        </TouchableHighlight>
      );
    }
```

```
  },
  Title(route, navigator, index, navState) {
    //You can inject the title aswell.  If you don't we'll use the route name.
    return (<Text style={styles.navbarTitle}>{route.navbarTitle || route.name}</Text>);
  }
};

//This is considered a sub-page that navigator is showing
class Home extends Component {

  //This trick depends on that componentWillMount fires before the navbar is created
  componentWillMount() {
        this.props.route.navbarTitle = "Home";

        this.props.route.rightNavButton = {
            text: "Button",
            onPress: this._doSomething.bind(this)
        };
    }

  //This method will be invoked by pressing the injected button.
  _doSomething() {
      Alert.alert(
      'Awesome, eh?',
      null,
      [
        {text: 'Indeed'},
      ]
    )
  }

  render() {
    return (
      <View style={styles.container}>
            <Text>You are home</Text>
        </View>
    );
  }
}

var styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
    marginTop: 66
  },
  navbar: {
    backgroundColor: '#ffffff',
  },
  navbarTitle: {
    marginVertical: 10,
    fontSize: 17
  },
  leftNavButton: {
    marginVertical: 10,
    paddingLeft: 8,
 },
  rightNavButton: {
    marginVertical: 10,
```

```
    paddingRight: 8,
  },
  navbarButtonText: {
    fontSize: 17,
    color: "#007AFF"
  }
});

AppRegistry.registerComponent('AppContainer', () => AppContainer);
```

Lire Navigateur avec boutons injectés depuis les pages en ligne: https://riptutorial.com/fr/react-native/topic/6416/navigateur-avec-boutons-injectes-depuis-les-pages

Chapitre 26: Notification push

Introduction

Nous pouvons ajouter Push Notification pour réagir avec une application native en utilisant le module npm **react -native-push-notification** par **zo0r** . Cela permet un développement multi-plateforme.

Installation

npm install --save rea-native-push-notification

lien réactif

Remarques

Référez-vous à GitHub Repo de ce module pour plus de détails.

Examples

Push Setup Simple Setup

Créer un nouveau projet PushNotification

```
react-native init PushNotification
```

Mettez suivant dans index.android.js

```
import React, { Component } from 'react';

import {
  AppRegistry,
  StyleSheet,
  Text,
  View,
  Button
} from 'react-native';

import PushNotification from 'react-native-push-notification';

export default class App extends Component {

    constructor(props){
        super(props);

        this.NewNotification = this.NewNotification.bind(this);
      }

    componentDidMount(){
```

```
        PushNotification.configure({

            // (required) Called when a remote or local notification is opened or received
            onNotification: function(notification) {
                console.log( 'NOTIFICATION:', notification );
            },

            // Should the initial notification be popped automatically
            // default: true
            popInitialNotification: true,

            /**
              * (optional) default: true
              * - Specified if permissions (ios) and token (android and ios) will requested or
not,
              * - if not, you must call PushNotificationsHandler.requestPermissions() later
              */
            requestPermissions: true,
        });

    }

      NewNotification(){

          let date = new Date(Date.now() + (this.state.seconds * 1000));

          //Fix for IOS
        if(Platform.OS == "ios"){
            date = date.toISOString();
        }

        PushNotification.localNotificationSchedule({
            message: "My Notification Message", // (required)
            date: date,// (optional) for setting delay
            largeIcon:""// set this blank for removing large icon
            //smallIcon: "ic_notification", // (optional) default: "ic_notification" with
fallback for "ic_launcher"
        });
    }

      render() {

        return (
            <View style={styles.container}>
                <Text style={styles.welcome}>
                  Push Notification
                </Text>
                <View style={styles.Button} >
                <Button
                  onPress={()=>{this.NewNotification()}}
                  title="Show Notification"
                  style={styles.Button}
                  color="#841584"
                  accessibilityLabel="Show Notification"
                />
                </View>
            </View>
        );
      }
}
```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
  },
  welcome: {
    fontSize: 20,
    textAlign: 'center',
    margin: 10,
  },
  Button:{
    margin: 10,
  }
});

AppRegistry.registerComponent('PushNotification', () => App);
```

Navigation vers la scène à partir de la notification

Voici un exemple simple pour démontrer comment sauter / ouvrir un écran spécifique en fonction de la notification. Par exemple, lorsqu'un utilisateur clique sur la notification, l'application doit s'ouvrir et accéder directement à la page des notifications au lieu de la page d'accueil.

```
'use strict';

import React, { Component } from 'react';
import {
    StyleSheet,
    Text,
    View,
    Navigator,
    TouchableOpacity,
    AsyncStorage,
    BackAndroid,
    Platform,
} from 'react-native';
import PushNotification from 'react-native-push-notification';

let initialRoute = { id: 'loginview' }

export default class MainClass extends Component
{
    constructor(props)
    {
        super(props);

        this.handleNotification = this.handleNotification.bind(this);
    }

    handleNotification(notification)
    {
        console.log('handleNotification');
        var notificationId = ''
        //your logic to get relevant information from the notification

    //here you navigate to a scene in your app based on the notification info
```

```
        this.navigator.push({ id: Constants.ITEM_VIEW_ID, item: item });
    }

    componentDidMount()
    {
        var that = this;

        PushNotification.configure({

            // (optional) Called when Token is generated (iOS and Android)
            onRegister: function(token) {
                console.log( 'TOKEN:', token );
            },

            // (required) Called when a remote or local notification is opened or received
            onNotification(notification) {
                console.log('onNotification')
                console.log( notification );

                that.handleNotification(notification);
            },

            // ANDROID ONLY: (optional) GCM Sender ID.
            senderID: "Vizido",

            // IOS ONLY (optional): default: all - Permissions to register.
            permissions: {
                alert: true,
                badge: true,
                sound: true
            },

            // Should the initial notification be popped automatically
            // default: true
            popInitialNotification: true,

            /**
              * (optional) default: true
              * - Specified if permissions (ios) and token (android and ios) will requested or
not,
              * - if not, you must call PushNotificationsHandler.requestPermissions() later
              */
            requestPermissions: true,
        });
    }

    render()
    {

        return (
            <Navigator
                ref={(nav) => this.navigator = nav }
                initialRoute={initialRoute}
                renderScene={this.renderScene.bind(this)}
                configureScene={(route) =>
                    {
                        if (route.sceneConfig)
                        {
                            return route.sceneConfig;
                        }
                        return Navigator.SceneConfigs.FadeAndroid;
```

```
                    }
                }
            />
        );
    }

    renderScene(route, navigator)
    {

        switch (route.id)
        {
            // do your routing here
            case 'mainview':
                return ( <MainView navigator={navigator} /> );

            default:
                return ( <MainView navigator={navigator} /> );
        }
    }
}
```

Lire Notification push en ligne: https://riptutorial.com/fr/react-native/topic/9674/notification-push

Chapitre 27: Polices Personnalisées

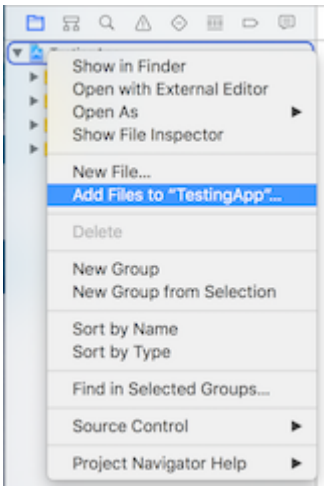
Examples

Étapes pour utiliser des polices personnalisées dans React Native (Android)

1. Collez votre fichier de polices à l'intérieur d' `android/app/src/main/assets/fonts/font_name.ttf`
2. Recompiler l'application Android en exécutant `react-native run-android`
3. Maintenant, vous pouvez utiliser `fontFamily: 'font_name'` dans vos styles natifs React

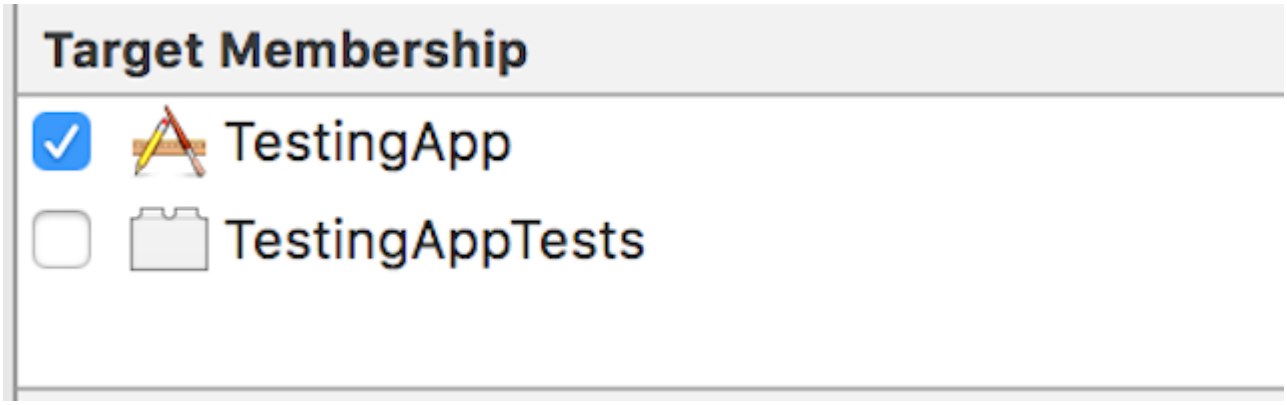
Étapes pour utiliser des polices personnalisées dans React Native (iOS)

1. Incluez la police dans votre projet Xcode.



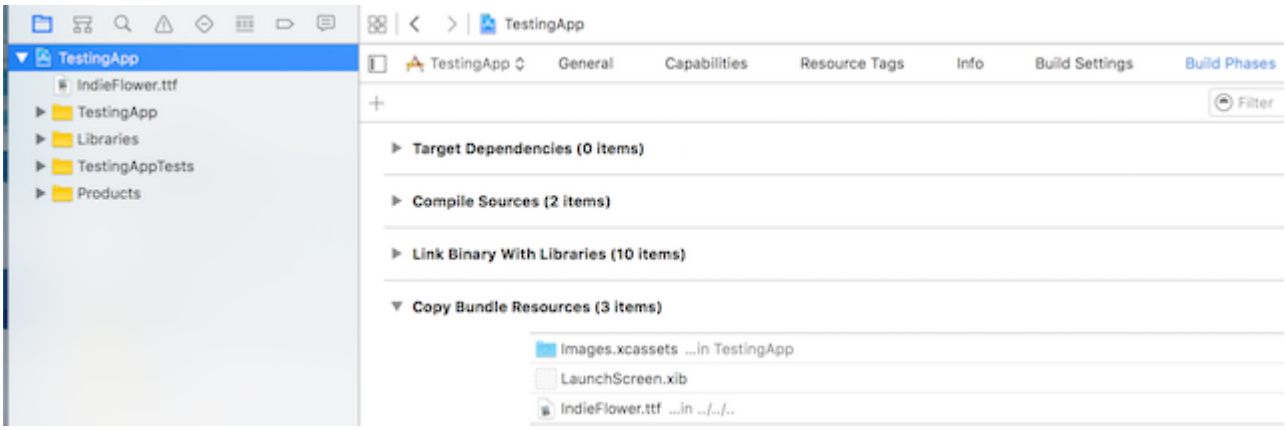
2. Assurez-vous qu'ils sont inclus dans la colonne Adhésion cible

Cliquez sur la police du navigateur et vérifiez si la police est incluse.



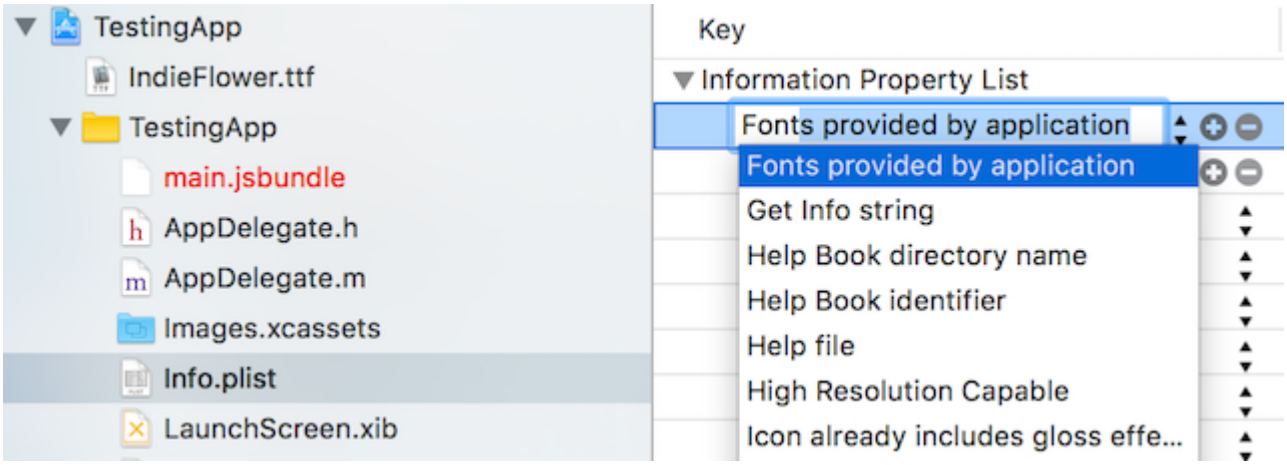
3. Vérifiez si la police incluse en tant que ressource dans votre bundle

Cliquez sur votre fichier de projet Xcode, sélectionnez "Build Phases, sélectionnez" Copy Bundle Resources ". Vérifiez si votre police est ajoutée.



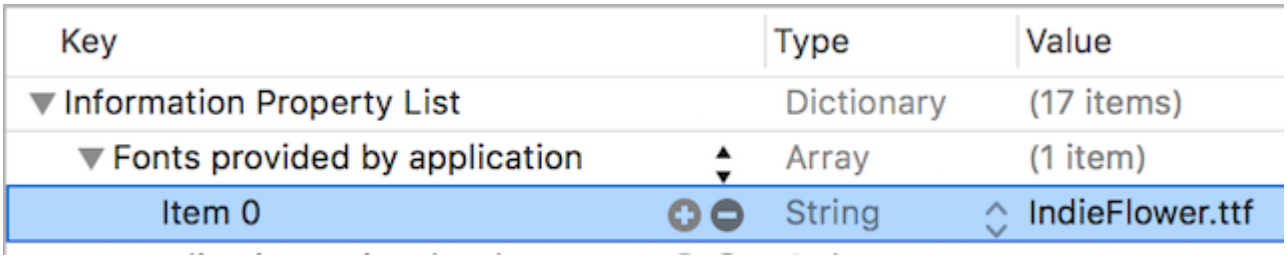
4. Inclure la police dans Application Plist (Info.plist)

Dans le dossier principal de l'application, ouvrez Info.plist, cliquez sur "Liste des propriétés d'informations", puis sur le signe plus (+). dans la liste déroulante, choisissez "Polices fournies par l'application".



5. Ajouter le nom de la police dans les polices fournies par l'application

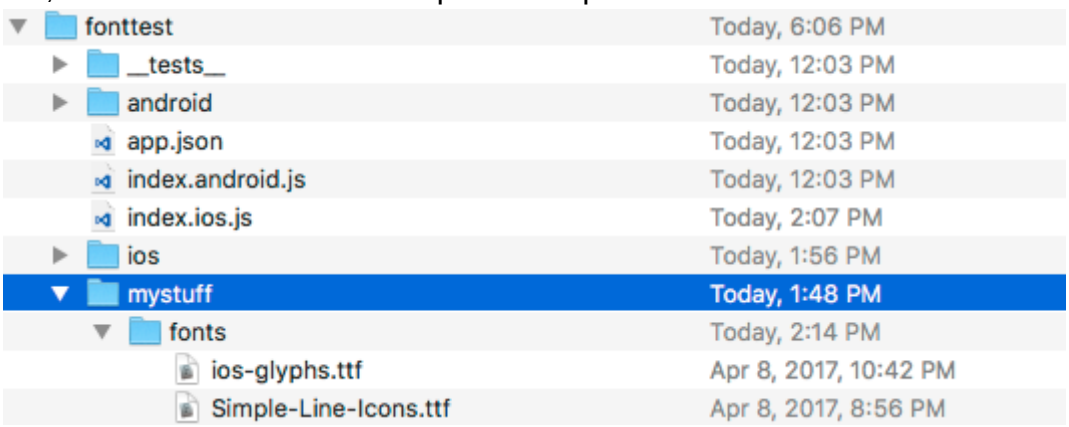
Développez Polices fournies par l'application et ajoutez le nom exact de la police à la colonne Valeur



6. Utilisez-le dans l'application

```
<Text style={{fontFamily:'IndieFlower'}}>
  Welcome to React Native!
</Text>
```

Polices personnalisées pour Android et IOS

- Créez un dossier dans votre dossier de projet et ajoutez-y vos polices. Exemple:
    - Exemple: Ici, nous avons ajouté un dossier dans la racine appelé "mystuff", puis "polices", et à l'intérieur nous avons placé nos polices:
        - 

- Ajoutez le code ci-dessous dans `package.json` .

```
{
    ...

    "rnpm": {
        "assets": [
          "path/to/fontfolder"
        ]
    },

    ...
}
```

    - Pour l'exemple ci-dessus, notre package.json aurait maintenant un chemin de "mystuff / fonts":

```
"rnpm": {
  "assets": [
    "mystuff/fonts"
  ]
}
```

- Exécutez la commande de `react-native link` .
- Utilisation de polices personnalisées sur le projet sous le code

```
<Text style={{ fontFamily: 'FONT-NAME' }}>
    My Text
</Text>
```

Où `FONT-NAME` est la plate-forme de préfixe spécifique.

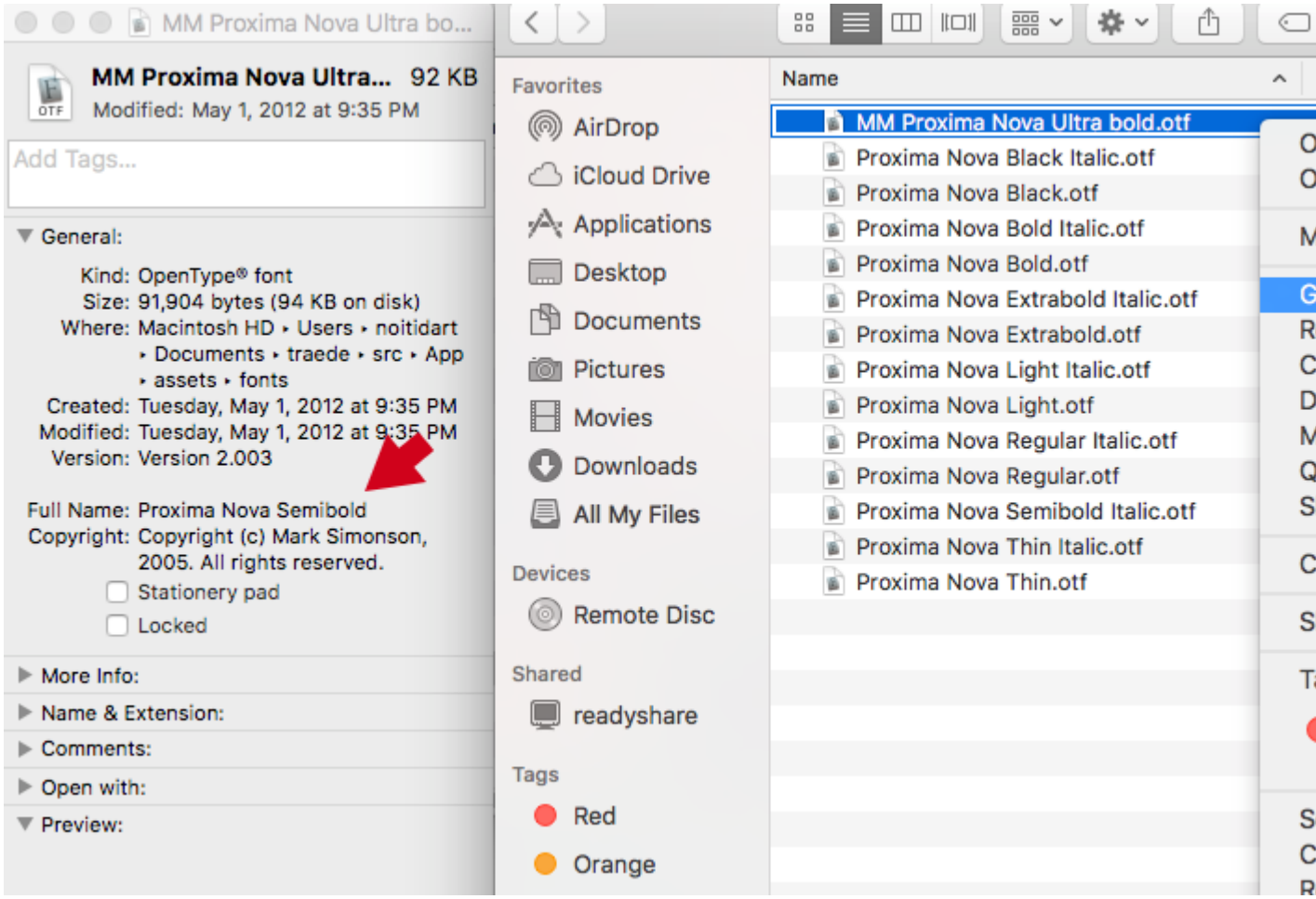
Android

FONT-NAME est le mot avant l'extension dans le fichier. Exemple: Le nom de fichier de

votre police est `Roboto-Regular.ttf` , vous devez donc définir `fontFamily: Roboto-Regular` .

iOS

FONT-NAME est "Full Name" trouvé après un clic droit sur le fichier de police, puis en cliquant sur "Get Info". (Source: https://stackoverflow.com/a/16788493/2529614), dans la capture d'écran ci-dessous, le nom du fichier est `MM Proxima Nova Ultra bold.otf` , cependant "Nom complet" est "Proxima Nova Semibold", vous devriez donc `fontFamily: Proxima Nova Semibold` . Capture d'écran -



- Exécutez `react-native run-ios` ou `react-native run-android` (ceci sera recompilé avec les ressources)

Lire Polices Personnalisées en ligne: https://riptutorial.com/fr/react-native/topic/4341/polices-personnalisees

Chapitre 28: RefreshControl avec ListView

Remarques

Les références:

RefreshControl: https://facebook.github.io/react-native/docs/refreshcontrol.html

ListView: https://facebook.github.io/react-native/docs/listview.html

Examples

Contrôle de rafraîchissement

```
_refreshControl(){
  return (
    <RefreshControl
      refreshing={this.state.refreshing}
      onRefresh={()=>this._refreshListView()} />
  )
}
```

rafraîchissant: c'est l'état du spinner (true, false).

onRefresh: cette fonction invoquera lors de l'actualisation de ListView / ScrollView.

Fonction onRefresh Exemple

```
_refreshListView(){
  //Start Rendering Spinner
  this.setState({refreshing:true})
  this.state.cars.push(
    {name:'Fusion',color:'Black'},
    {name:'Yaris',color:'Blue'}
  )
  //Updating the dataSource with new data
  this.setState({ dataSource:
      this.state.dataSource.cloneWithRows(this.state.cars) })
  this.setState({refreshing:false}) //Stop Rendering Spinner
}
```

Ici, nous mettons à jour le tableau et ensuite nous mettrons à jour le dataSource. nous pouvons utiliser fetch pour demander quelque chose du serveur et utiliser async / waiting.

Refresh Control with ListView Exemple complet

RefreshControl est utilisé dans un composant ScrollView ou ListView pour ajouter une fonctionnalité d'actualisation. dans cet exemple, nous allons l'utiliser avec ListView

```
'use strict'
import React, { Component } from 'react';
import { StyleSheet, View, ListView, RefreshControl, Text } from 'react-native'


class RefreshControlExample extends Component {
  constructor () {
    super()
    this.state = {
      refreshing: false,
      dataSource: new ListView.DataSource({
        rowHasChanged: (row1, row2) => row1 !== row2 }),
      cars : [
        {name:'Datsun',color:'White'},
        {name:'Camry',color:'Green'}
      ]
    }
  }

   componentWillMount(){
     this.setState({ dataSource:
       this.state.dataSource.cloneWithRows(this.state.cars) })
   }

  render() {
    return (
      <View style={{flex:1}}>
        <ListView
          refreshControl={this._refreshControl()}
          dataSource={this.state.dataSource}
          renderRow={(car) => this._renderListView(car)}>
        </ListView>
      </View>
    )
  }

  _renderListView(car){
    return(
      <View style={styles.listView}>
        <Text>{car.name}</Text>
        <Text>{car.color}</Text>
      </View>
    )
  }

  _refreshControl(){
    return (
      <RefreshControl
        refreshing={this.state.refreshing}
        onRefresh={()=>this._refreshListView()} />
    )
  }

  _refreshListView(){
    //Start Rendering Spinner
    this.setState({refreshing:true})
    this.state.cars.push(
      {name:'Fusion',color:'Black'},
      {name:'Yaris',color:'Blue'}
    )
    //Updating the dataSource with new data
```

```
    this.setState({ dataSource:
        this.state.dataSource.cloneWithRows(this.state.cars) })
    this.setState({refreshing:false}) //Stop Rendering Spinner
  }

}

const styles = StyleSheet.create({

  listView: {
    flex: 1,
    backgroundColor:'#fff',
    marginTop:10,
    marginRight:10,
    marginLeft:10,
    padding:10,
    borderWidth:.5,
    borderColor:'#dddddd',
    height:70
  }

})

module.exports = RefreshControlExample
```

Lire RefreshControl avec ListView en ligne: https://riptutorial.com/fr/react-native/topic/6672/refreshcontrol-avec-listview

Chapitre 29: Rendu de plusieurs accessoires

Examples

rendre plusieurs variables

Pour rendre plusieurs accessoires ou variables, nous pouvons utiliser `` .

```
render() {
  let firstName = 'test';
  let lastName = 'name';
  return (
    <View style={styles.container}>
      <Text>{`${firstName} ${lastName}` } </Text>
    </View>
  );
}
```

Sortie: nom du test

Lire Rendu de plusieurs accessoires en ligne: https://riptutorial.com/fr/react-native/topic/10781/rendu-de-plusieurs-accessoires

Chapitre 30: Requêtes HTTP

Syntaxe

- fetch (url, options) [. alors (...) [. catch (...)]]

Remarques

- L'API Fetch est l'API la plus utilisée pour les requêtes HTTP. Il est moderne, flexible et utilise des promesses.
- L'API XMLHttpRequest est également utilisée pour les requêtes HTTP et est principalement incluse afin que les développeurs puissent utiliser leurs bibliothèques existantes préférées, comme ApiSauce .
- L'API Websocket peut être utilisée pour des données "en direct" dans des scénarios en temps réel, par exemple dans des applications de discussion.

Examples

WebSockets

```
var ws = new WebSocket('ws://host.com/path');

ws.onopen = () => {
  // connection opened

  ws.send('something'); // send a message
};

ws.onmessage = (e) => {
  // a message was received
  console.log(e.data);
};

ws.onerror = (e) => {
  // an error occurred
  console.log(e.message);
};

ws.onclose = (e) => {
  // connection closed
  console.log(e.code, e.reason);
};
```

HTTP avec l'API de récupération

Il convient de noter que Fetch *ne prend pas en charge les rappels de progression* . Voir: https://github.com/github/fetch/issues/89 .

L'alternative est d'utiliser XMLHttpRequest https://developer.mozilla.org/en-

US/docs/Web/Events/progress .

```
fetch('https://mywebsite.com/mydata.json').then(json => console.log(json));

fetch('/login', {
  method: 'POST',
  body: form,
  mode: 'cors',
  cache: 'default',
}).then(session => onLogin(session), failure => console.error(failure));
```

Plus de détails sur la récupération peuvent être trouvés à MDN

Mise en réseau avec XMLHttpRequest

```
var request = new XMLHttpRequest();
request.onreadystatechange = (e) => {
  if (request.readyState !== 4) {
    return;
  }

  if (request.status === 200) {
    console.log('success', request.responseText);
  } else {
    console.warn('error');
  }
};

request.open('GET', 'https://mywebsite.com/endpoint/');
request.send();
```

Utiliser des promesses avec l'API d'extraction et Redux

Redux est la bibliothèque de gestion d'états la plus utilisée avec React-Native. L'exemple suivant montre comment utiliser l'API d'extraction et envoyer des modifications à votre réducteur d'état d'applications en utilisant redux-thunk.

```
export const fetchRecipes = (action) => {
  return (dispatch, getState) => {
    fetch('/recipes', {
        method: 'POST',
        headers: {
          'Accept': 'application/json',
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({
          recipeName,
          instructions,
          ingredients
        })
    })
    .then((res) => {
      // If response was successful parse the json and dispatch an update
      if (res.ok) {
        res.json().then((recipe) => {
          dispatch({
```

```
            type: 'UPDATE_RECIPE',
            recipe
          });
        });
      } else {
        // response wasn't successful so dispatch an error
        res.json().then((err) => {
          dispatch({
            type: 'ERROR_RECIPE',
            message: err.reason,
            status: err.status
          });
        });
      }
    })
    .catch((err) => {
      // Runs if there is a general JavaScript error.
      dispatch(error('There was a problem with the request.'));
    });
  };
};
```

Socket Web avec Socket.io

Installer *socket.io-client*

```
npm i socket.io-client --save
```

Module d'importation

```
import SocketIOClient from 'socket.io-client/dist/socket.io.js'
```

Initialiser dans votre constructeur

```
constructor(props){
    super(props);
    this.socket = SocketIOClient('http://server:3000');
  }
```

Maintenant, pour utiliser correctement votre connexion socket, vous devez également lier vos fonctions dans constructeur. Supposons que nous devions créer une application simple, qui enverra un ping à un serveur via un socket toutes les 5 secondes (considérez ceci comme un ping), puis l'application recevra une réponse du serveur. Pour ce faire, créons d'abord ces deux fonctions:

```
_sendPing(){
    //emit a dong message to socket server
    socket.emit('ding');
}

_getReply(data){
    //get reply from socket server, log it to console
    console.log('Reply from server:' + data);
}
```

Maintenant, nous devons lier ces deux fonctions dans notre constructeur:

```
constructor(props){
    super(props);
    this.socket = SocketIOClient('http://server:3000');

    //bind the functions
    this._sendPing = this._sendPing.bind(this);
    this._getReply = this._getReply.bind(this);
}
```

Après cela, nous devons également lier la fonction _getReply avec le socket afin de recevoir le message du serveur de socket. Pour ce faire, nous devons attacher notre fonction _getReply avec un objet socket. Ajoutez la ligne suivante à notre constructeur:

```
this.socket.on('dong', this._getReply);
```

Maintenant, chaque fois que le serveur de socket émet avec le «dong», votre application pourra le recevoir.

Http avec axios

Configurer

Pour les demandes Web, vous pouvez également utiliser la bibliothèque axios .

C'est facile à configurer. Pour cela, vous pouvez créer le fichier axios.js par exemple:

```
import * as axios from 'axios';

var instance = axios.create();
instance.defaults.baseURL = serverURL;
instance.defaults.timeout = 20000;]
//...
//and other options

export { instance as default };
```

et ensuite l'utiliser dans n'importe quel fichier que vous voulez.

Demandes

Pour éviter d'utiliser le pattern 'Swiss knife' pour chaque service de votre backend, vous pouvez créer un fichier séparé avec des méthodes pour cela dans le dossier pour la fonctionnalité d'intégration:

```
import axios from '../axios';
import {
    errorHandling
} from '../common';

const UserService = {
```

```
        getCallToAction() {
        return axios.get('api/user/dosomething').then(response => response.data)
            .catch(errorHandling);
    },
}
export default UserService;
```

Essai

Il existe une lib pour tester les axios: axios-mock-adapter .

Avec cette lib, vous pouvez définir sur axios toute réponse que vous souhaitez pour le tester. Vous pouvez également configurer des erreurs spéciales pour vos méthodes axois. Vous pouvez l'ajouter à votre fichier axios.js créé à l'étape précédente:

```
import MockAdapter from 'axios-mock-adapter';

var mock = new MockAdapter(instance);
mock.onAny().reply(500);
```

par exemple.

Redux Store

Parfois, vous devez ajouter des en-têtes à autoriser, que vous stockez probablement dans votre magasin Redux.

Dans ce cas, vous aurez besoin d'un autre fichier, interceptors.js avec cette fonction:

```
export function getAuthToken(storeContainer) {
    return config => {
        let store = storeContainer.getState();
        config.headers['Authorization'] = store.user.accessToken;
        return config;
    };
}
```

Suivant dans le constructeur de votre composant racine, vous pouvez ajouter ceci:

```
axios.interceptors.request.use(getAuthToken(this.state.store));
```

et ensuite toutes vos demandes seront suivies avec votre jeton d'autorisation.

Comme vous pouvez le voir, axios est une bibliothèque très simple, configurable et utile pour les applications basées sur react-native.

Lire Requêtes HTTP en ligne: https://riptutorial.com/fr/react-native/topic/2375/requetes-http

Chapitre 31: Test d'unité

Introduction

Le test unitaire est une pratique de test de bas niveau où les plus petites unités ou composants du code sont testés.

Examples

Test unitaire avec blague

`Jest` est un framework de test javascript largement utilisé pour tester les applications de réaction. C'est soutenu par facebook

Voici un test

```
import 'react-native';
import React from 'react';
import Index from '../index.android.js';

import renderer from 'react-test-renderer';

it('renders correctly', () => {
  const tree = renderer.create(
    <Index />
  );
});
```

Voici un code pour le faire passer

```
import React, { Component } from 'react';
import {
  AppRegistry,
  StyleSheet,
  Text,
  View
} from 'react-native';

export default class gol extends Component {
  render() {
    return (
      <View>
        <Text>
          Welcome to React Native!
        </Text>
        <Text>
          To get started, edit index.android.js
        </Text>
        <Text>
          Double tap R on your keyboard to reload,{'\n'}
          Shake or press menu button for dev menu
        </Text>
```

```
      </View>
    );
  }
}

AppRegistry.registerComponent('gol', () => gol);
```

Test unitaire dans React Native utilisant Jest

A partir de la version 0.38 de rea-native, une configuration Jest est incluse par défaut lors de l'exécution de init-native. La configuration suivante doit être automatiquement ajoutée à votre fichier package.json:

```
    "scripts": {
    "start": "node node_modules/react-native/local-cli/cli.js start",
    "test": "jest"
    },
    "jest": {
     "preset": "react-native"
    }
```

Vous pouvez exécuter `run npm test or jest` pour tester un fichier natif. Pour l'exemple de code: Link

Lire Test d'unité en ligne: https://riptutorial.com/fr/react-native/topic/8281/test-d-unite

Chapitre 32: WebView

Introduction

Webview peut être utilisé pour charger des pages Web externes ou du contenu HTML. Ce composant est là par défaut.

Examples

Composant simple utilisant webview

```
import React, { Component } from 'react';
import { WebView } from 'react-native';

class MyWeb extends Component {
  render() {
    return (
      <WebView
        source={{uri: 'https://github.com/facebook/react-native'}}
        style={{marginTop: 20}}
       />
    );
  }
}
```

Lire WebView en ligne: https://riptutorial.com/fr/react-native/topic/8763/webview

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec rea-native	Adam, Community, Damien Varron, Dmitry Petukhov, Dr. Nitpick, Idan, Kaleb Portillo, Lucas Oliveira, manosim, Scimonster, Sivart, Tushar Khatiwada, xhg, Yevhen Dubinin
2	Android - Bouton Retour de matériel	Cássio Santos, manosim, Michael S, Pascal Le Merrer, Sriraman, Virat18
3	API d'animation	Shashank Udupa, Sriraman, Tom Walters
4	Bonjour le monde	stereodenis, Zakaria Ridouh
5	Coiffant	Jigar Shah, Martin Cup, Scimonster
6	Composants	Michael Hancock, Sriraman, Tobias Lins
7	Créer un partageable APK pour Android	Aditya Singh
8	Disposition	Alex Belets, gwint, Jagadish Upadhyay, Scimonster, sudo bangbang
9	ESLint in react-native	Alex Belets
10	Etat	AndyI, David, Jagadish Upadhyay, Tim Rijavec, Tobias Lins
11	Exécuter une application sur l'appareil (version Android)	Jagadish Upadhyay, Lwin Kyaw Myat, Mayeul
12	Images	Jagadish Upadhyay, Jigar Shah, Serdar Değirmenci, Zakaria Ridouh
13	Instructions en ligne de commande	Dmitry Petukhov, epsilondelta, Idan, Jagadish Upadhyay, manosim, Mozak, Sriraman, Tim Rijavec
14	Intégration avec Firebase pour l'authentification	Ankit Sinha, corasan
15	Le débogage	Jagadish Upadhyay, mostafiz rahman

16	Le routage	sudo bangbang
17	Les accessoires	CallMeNorm, Chris Pena, corasan, fson, Gianfranco P., henkimon, Hugo Dozois, Idan, Jagadish Upadhyay, Tobias Lins, Yevhen Dubinin, zhenjie ruan
18	Liaison de l'API native	Viktor Seč
19	ListView	Kaleb Portillo
20	Meilleures pratiques de rendu	Alex Belets
21	Meilleures pratiques du navigateur	Ankit Sinha, Michael Helvey, Pankaj Thakur
22	Modal	Ahmed Ali, Liron Yahdav, Tobias Lins
23	Module de plate-forme	Florian Hämmerle, Gabriel Diez, Jagadish Upadhyay, Zakaria Ridouh
24	Modules natifs	Andres C. Viesca
25	Navigateur avec boutons injectés depuis les pages	Ahmed Al Haddad
26	Notification push	shaN, Tejashwi Kalp Taru
27	Polices Personnalisées	Abdulaziz Alkharashi, Lwin Kyaw Myat, Noitidart, Olivia, Sriraman
28	RefreshControl avec ListView	Abdulaziz Alkharashi
29	Rendu de plusieurs accessoires	Jigar Shah
30	Requêtes HTTP	Alex Belets, Alireza Valizade, AntonB, Chris Pena, Daniel Schmidt, Dmitry Petukhov, Everettss, Jagadish Upadhyay, manosim, MauroPorrasP, respectTheCode, shaN, Tejashwi Kalp Taru, Tobias Lins
31	Test d'unité	Ankit Sinha, sudo bangbang
32	WebView	sudo bangbang