

Introduction à React Native : l'outil Expo

Table des matières

I. Contexte	3
II. Qu'est-ce que Expo.io ?	3
III. Exercice : Appliquez la notion	6
IV. Installation et prise en main d'Expo	6
V. Exercice : Appliquez la notion	11
VI. Les fonctionnalités de base d'Expo	11
VII. Exercice : Appliquez la notion	14
VIII. Essentiel	14
IX. Auto-évaluation	14
A. Exercice final	14
B. Exercice : Défi.....	16
Solutions des exercices	18

I. Contexte

Durée : 1 h

Environnement de travail : Un carnet pour prendre des notes, un ordinateur avec un navigateur web et un téléphone portable iOS ou Android

Pré-requis : Aucun

Contexte

Contrairement aux applications natives où il faut maîtriser fondamentalement plusieurs langages pour développer une application, grâce à React Native, on va pouvoir s'approcher d'une expérience utilisateur finale semblable, en n'utilisant pratiquement qu'un seul langage : JavaScript.

Ce code est interprété dans un moteur JavaScript embarqué dans une application native et s'exécute dans un processus qui lui est dédié. On obtient grâce à cela une application *cross-platform* pour iOS, Android, voire pour du web.

De plus, React Native est une extension de nombreux concepts de son homologue : React Web. Un développeur web peut donc très facilement prendre en main l'outil et devenir productif sur ce dernier, ce qui est un vrai plus !

L'avantage est également que JavaScript est un langage de scripting, non compilé. Donc, de cette manière, on peut tout à fait imaginer pouvoir envoyer et recharger à chaud du code dans le moteur embarqué. Pratique, pour l'expérience de développement.

II. Qu'est-ce que Expo.io ?

Objectifs

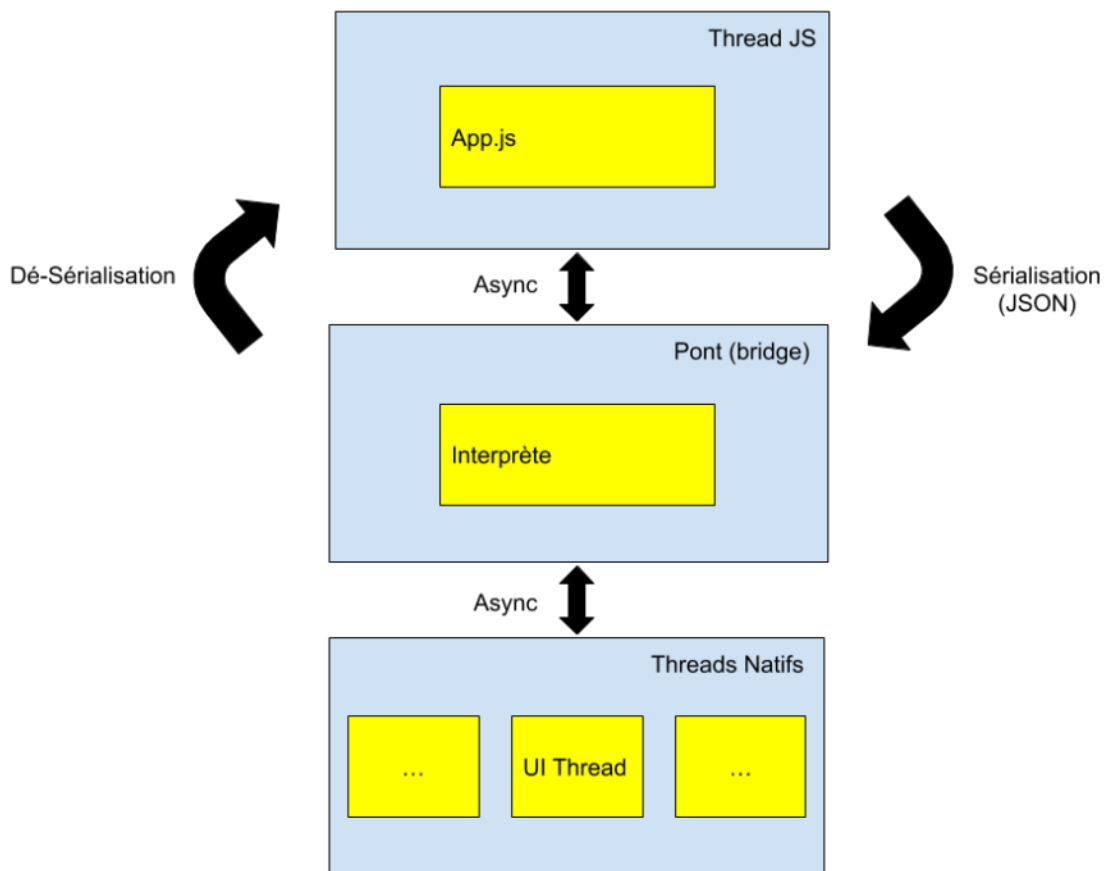
- Comprendre le fonctionnement d'une application hybride en React Native
- Comprendre l'intérêt de Expo, le framework pour React Native
- Comprendre les différences entre Expo et React Native

Mise en situation

Nous allons utiliser le framework Expo pour ce cours.

Pour faire le parallèle avec PHP, par exemple, nous avons Symfony qui est un framework pour PHP : eh bien, Expo est un framework pour React Native. C'est donc principalement une surcouche qui facilite le développement, et il faut donc comprendre le fonctionnement de React Native pour comprendre ce que Expo apporte.

Pour résumer rapidement, le fonctionnement d'une application React Native est le suivant :



Le thread JavaScript

C'est ici que le code que nous écrivons en JavaScript s'exécute, il est donc empaqueté en un seul fichier `App.js` par le packager de React Native, puis ce fichier est lancé dans un moteur JavaScript à l'intérieur d'une application Native sur iOS ou sur Android.

Ce thread communique avec un pont natif en sérialisant des instructions en JSON. Cette opération arrive au minimum une fois toutes les 16.67 ms en fonction de l'*event loop* du moteur JavaScript, ce qui permet un affichage jusqu'à 60 images par seconde. S'il n'arrive pas à générer une instruction toutes les 16.67 ms, on perd alors une image, ce qui donne l'effet de ralenti qui dégrade l'expérience utilisateur.

Le pont natif (Bridge)

Ici se trouve la plus grosse partie du code de React Native (la librairie). Ce pont natif est composé d'un ensemble de choses, mais, très brièvement, il reçoit des instructions de la part du thread JavaScript, lui indiquant des changements à opérer sur l'application. De là, il va retranscrire ces informations en instructions qui seront interprétables par le thread natif, afin de pouvoir impacter le cycle de vie et l'affichage de l'application.

Les threads natifs

Ici sont lancés plusieurs threads (processus) propres à une application native. Le plus important pour l'instant, à ce niveau du cours, est de comprendre que le thread UI (*User Interface* → Interface Utilisateur) reçoit des instructions pour afficher des composants natifs Android ou iOS, qui ont été pré-mâchés par le pont React Native.

Que vient faire Expo là-dedans ?

Expo part du principe que, comme le thread JS ne contient que du JavaScript et que JavaScript n'est pas un langage compilé, on peut donc compiler une application avec un ensemble de dépendances natives incluses par défaut (telles que le support de la caméra ou du GPS, qui ne seront pas amenées à changer) et uniquement recharger à la volée le code JavaScript en l'envoyant par le réseau à chaque modification de développement.

De cette manière, on simplifie grandement l'expérience de développement, car nous n'avons plus besoin de compiler l'application, du moins pendant la phase de programmation. Tout cela facilite l'accès à la création d'applications mobiles, qui sont généralement très lourdes à mettre en place pour un débutant.

Mais ce n'est pas tout, Expo fournit et maintient un ensemble d'APIs système, comme l'accès à la caméra ou au système de fichiers du téléphone, ce qui est très pratique. Nous aborderons cela plus tard dans ce cours.

Trois différents cadres de travail avec Expo

Expo propose de travailler avec trois workflows, qui ont chacun leurs avantages et inconvénients.

Le « Managed Workflow »

Ce mode permet de coder très rapidement une application ou un POC (*Proof of Concept*, preuve de concept). C'est d'ailleurs avec ce mode que nous allons faire la plupart des exercices de ce cours.

Expo fournit une application qui contient l'ensemble des APIs disponibles sur la plateforme. On flashe un QR code que nous donne le framework, et c'est parti, nous sommes dans la vue de notre application. Nous détaillerons plus tard son installation.

L'avantage est que l'on peut commencer très rapidement à coder ou prototyper quelque chose sans avoir à monter un environnement de développement mobile complet avec tout ce que cela implique. On peut même le faire sur Internet en utilisant <https://snack.expo.io>.

Par contre, il y a quelques contraintes à utiliser ce mode, notamment la taille de notre application, qui sera plus lourde du fait qu'elle est composée de bibliothèques qui ne seront probablement pas utilisées. De plus, on ne peut pas étendre les APIs qu'Expo ne fournit pas, telle que le Bluetooth. Dans ce cas-là, il faudra passer par le « Bare Workflow ».

Le « Bare Workflow »

Ce mode permet donc de pallier les problèmes du Managed Workflow. Il offre plus de liberté et de personnalisation, au détriment de la simplicité. Dans ce mode, il nous faut un environnement de développement mobile complet et nous devons compiler notre propre package applicatif, ce qui complique l'initialisation d'un projet et sa maintenance, car c'est à nous de gérer cette complexité, pas au tiers (Expo).

Le mode « Eject to ExpoKit »

C'est l'ancienne façon de faire le Bare Workflow. Dépréciée, cette méthode n'est plus recommandée et permet seulement d'accéder à certaines fonctionnalités qui n'auraient pas encore été portées sur le Bare Workflow. Son utilisation n'est plus supportée à partir du SDK 38 : <https://docs.expo.io/expokit/eject/>.

Complément

Pour plus de détails sur les workflows, consultez <https://docs.expo.io/introduction/managed-vs-bare/#workflow-comparison>.

Syntaxe À retenir

- Expo est donc une grosse boîte à outils pour React Native. Pour faire un parallèle : avec PHP, nous avons Symfony, en React Native, nous avons Expo. C'est un framework qui accélère les développements en React Native en fournissant plusieurs APIs système (comme la caméra) fonctionnant sur iOS et Android, et qui sont maintenues par la société Expo. C'est donc un gage de qualité qui nous facilite la vie.
- Expo propose 3 façons de travailler :
 - Un mode **Managed**, qui est le plus facile d'accès, mais n'est pas trop personnalisable en termes de librairies embarquées.
 - Un mode **Bare** qui offre plus de souplesse sur les librairies embarquées. Il permet d'installer du code natif qui ne serait pas disponible dans Expo, tel qu'un module de *machine learning* par exemple, mais s'accompagne de toute la complexité liée à une application React Native classique.
 - Enfin, un mode **Ejected**, qui est l'ancienne version du Bare Workflow. Le mode Ejected possède toutes les fonctionnalités progressivement ajoutées au Bare Workflow, destiné à le remplacer.
- On peut commencer en mode Managed, puis migrer vers le Bare ou Ejected à tout moment. L'autre sens n'est par contre pas simple et peut représenter une vraie complexité : à éviter, donc.

Complément

Site d'Expo¹

Documentation d'Expo²

Exercice : Appliquez la notion

[solution n°1 p.19]

Exercice

Une application React Native est...

- ☐ Pas compilée du tout
- ☐ Partiellement compilée
- ☐ Entièrement compilée

Exercice

Quel est le principal avantage à l'utilisation du Managed Workflow de Expo ?

- ☐ Des performances maximales
- ☐ La possibilité d'utiliser son propre code natif
- ☐ Une expérience de développement facilitée

IV. Installation et prise en main d'Expo

¹ <https://expo.io/>

² <https://docs.expo.io/>

Objectifs

- Créer une première application simple avec Expo
- Comprendre l'utilisation du mode « Managed » d'Expo
- Voir les différentes manières d'utiliser Expo en Cloud ou installé sur son PC

Mise en situation

Expo est un framework agissant comme une boîte à outils pour React Native, permettant de n'écrire que du JavaScript.

Pour créer une application avec Expo, vous devez au préalable avoir installé son interface en ligne de commande.

```
1 <terminal>
2 npm install --global expo-cli
3 </terminal>
```

La partie native d'Expo contient par défaut un certain nombre d'APIs natives déjà embarquées et cela permet donc quelques interactions sympathiques, comme le fait de pouvoir écrire du code directement dans une page web hébergée sur le site d'Expo. C'est très pratique, notamment pour faire de petits essais et/ou partager du code.

Il existe deux manières de commencer un projet avec Expo.

Pour créer une application React Native, il faut taper les commandes suivantes dans votre terminal :

```
1 <terminal>
2 expo init premier-projet-rn
3 cd premier-projet-rn
4 npm start
5 </terminal>
```

Méthode Utiliser la version Cloud web d'Expo

La manière la plus simple de commencer avec Expo – et celle recommandée pour la réalisation des exercices de ce cours – est d'utiliser l'interface web <https://snack.expo.io>. De cette manière, nous n'avons rien besoin d'installer sur notre ordinateur.

Il est vivement recommandé d'utiliser cette méthode pour suivre ce cours, sachant qu'il est possible de télécharger le projet sur son ordinateur par la suite, malgré le fait d'avoir commencé sur la version web.

Attention Installer un projet Expo sur son propre ordinateur

Si nous souhaitons quand même faire une installation sur notre propre ordinateur afin d'utiliser notre IDE préféré ou réaliser un vrai projet tout de suite, le plus simple est d'utiliser l'outil en ligne de commande (CLI) d'Expo.

La procédure décrite ci-dessous est susceptible de varier dans le temps, c'est pourquoi il est recommandé de suivre les instructions définies ici¹. Cependant, voici une version traduite de cette page, pour résumer dans les grandes lignes les pré-requis.

Pré-requis pour la partie ordinateur

1. Avoir NodeJS installé sur l'ordinateur, de préférence la dernière version stable disponible (pour ce faire, se référer à la documentation de NodeJS²)
2. Avoir Git installé sur l'ordinateur (plus d'informations ici <https://git-scm.com/downloads>)
3. Installer l'utilitaire en ligne de commande globalement sur l'ordinateur via npm, le gestionnaire de paquets de NodeJS : `npm install -g expo-cli`

1 <https://docs.expo.io/get-started/installation/>

2 <https://nodejs.org/fr/>

4. Initialiser un projet en créant un nouveau dossier et en tapant `expo init <nom_du_projet>` en spécifiant le nom du projet, et choisir le *Managed Workflow Blank*

Une fois la partie serveur/ordinateur faite, Expo va se charger de traiter le code JavaScript et de fournir un portail pour l'envoyer dans une application mobile qui saura l'exécuter.

Il nous faut désormais un client mobile qui interprète ce code. Ici, la manière recommandée est d'utiliser l'application officielle d'Expo disponible sur Android et iOS :

- Pour Android : <https://play.google.com/store/apps/details?id=host.exp.exponent>
- Pour iOS : <https://apps.apple.com/fr/app/expo-client/id982107779>

Ou bien taper « Expo » dans la recherche de notre store favori.

Attention Système d'exploitation minimum requis

Pour Android, la version minimum est Lollipop (5), et pour iOS, la version minimum est iOS 10.0.

Remarque

À partir d'ici et pour la suite du cours, selon votre choix entre la version Cloud ou celle installée sur votre PC, il sera possible de suivre les instructions correspondantes :

- **[cloud]** : les instructions pour la version Cloud seront préfixées par ce tag
- **[desktop]** : tandis que les instructions pour la version bureau seront préfixées par ce tag

Se connecter à son projet

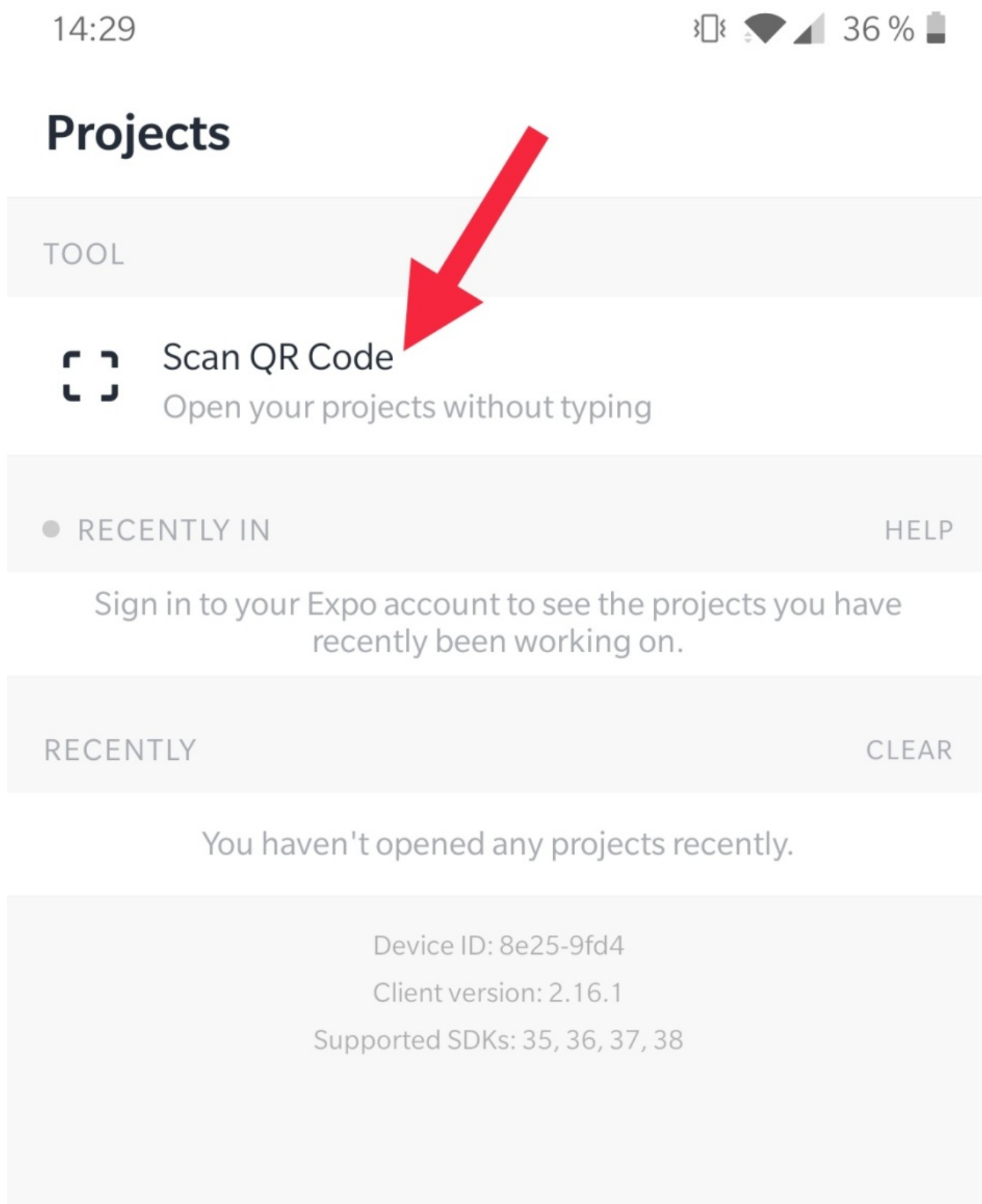
Une fois que nous avons initialisé notre projet dans le Cloud ou sur notre propre ordinateur, il faut s'y connecter.

Attention Attention à votre téléphone !

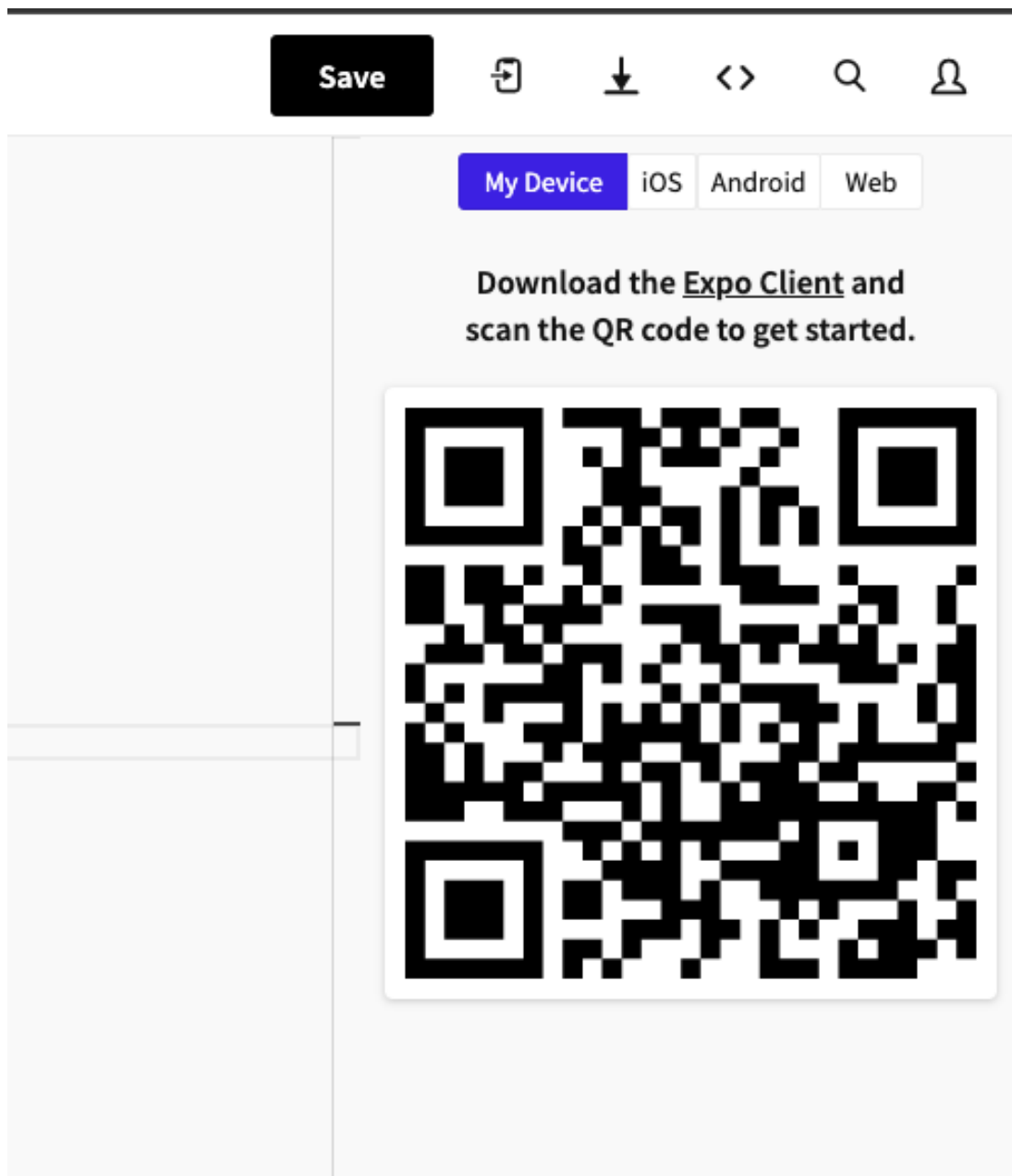
Pour Android, c'est assez simple et décrit ci-dessous avec le mode QR code.

Pour iOS, cela peut être un peu plus compliqué. Il faut télécharger l'application Expo, puis utiliser l'application Caméra de l'iPhone pour scanner le QR code si la version d'iOS le supporte. Si ce n'est pas le cas, alors il faudra envoyer un lien par e-mail ou par SMS et cliquer dessus depuis l'iPhone.

Ouvrons l'application Expo et cliquons sur **Scan QR Code**, comme sur la capture d'écran, pour flasher le QR code :



[cloud] Pour la version Cloud web



Sur la version Cloud, il est présent sur la partie droite de l'application web sur <https://snack.expo.io>. Une fois cela fait, patientons quelques secondes et nous devrions voir l'application s'ouvrir.

[desktop] Pour la version ordinateur

Positionnons-nous dans le dossier du projet créé précédemment et lançons la commande `expo start`. De là, un QR code apparaît dans la console, mais une page web s'ouvre dans le navigateur avec une URL de la sorte : `http://localhost:19002/`. Sur cette page, nous pouvons flasher le QR code en bas à gauche.

Remarque

Il faut par défaut que le téléphone soit sur le même réseau Wifi que l'ordinateur avant de flasher, sinon il faudra cliquer sur `tunnel` et re-flasher le nouveau QR code.

Syntaxe **À retenir**

- Expo Managed peut s'utiliser rapidement via une version Cloud web, mais dans le cadre d'un vrai projet, nous utiliserons plutôt la CLI de développement.
- Il faut utiliser l'application Expo disponible sur iOS et Android pour développer sa vraie future application en flashant un QR code.

ComplémentDocumentation d'Expo¹Installation d'Expo²

V. Exercice : Appliquez la notion

Question

[solution n°2 p.19]

Maintenant que vous avez votre première application affichée, changez le texte initial affiché à l'écran par « *Coucou ça marche* » et constatez le changement instantané sur votre application mobile sans rien toucher.

Pour faire cette modification, il faut modifier le fichier `App.js`, affiché sur la gauche dans <https://snack.expo.io>.

VI. Les fonctionnalités de base d'Expo

Objectifs

- Découvrir les principales commandes d'Expo CLI
- Découvrir les fonctionnalités fournies par Expo

Mise en situation

Afin de configurer le lancement de votre application, il est possible de transmettre des paramètres à Expo. En effet, nous aurons peut-être besoin de bloquer l'application en mode portrait, par exemple.

La plus grosse partie de la configuration Expo se trouve dans un fichier `app.json` à la racine du projet. Nous pouvons modifier ici bon nombre de paramètres. Nous reviendrons sur certains de ces paramètres dans un autre chapitre, mais n'hésitez pas à consulter la liste exhaustive sur la documentation³. Dans un premier temps, nous allons voir les commandes de base d'Expo, ainsi que ses fonctionnalités.



Via la CLI, les commandes les plus importantes sont :

- `expo start` qui permet de lancer l'interface web en mode bureau
- `expo android` et `expo ios` qui nous permettront de lancer l'application sur un émulateur, ce qui fait l'objet d'un autre chapitre de ce cours
- `expo doctor` qui permet d'effectuer un diagnostic de notre environnement, au cas où rien ne fonctionnerait

Pour le reste des commandes, nous pouvons nous référer à la documentation⁴ si nous le souhaitons. Pour le moment, nous n'en avons pas besoin ; de plus, nous découvrirons tout ce qu'il faut au fur et à mesure des chapitres.

1 <https://docs.expo.io/>

2 <https://docs.expo.io/get-started/installation/>

3 <https://docs.expo.io/versions/latest/config/app/>

4 <https://docs.expo.io/workflow/expo-cli/>

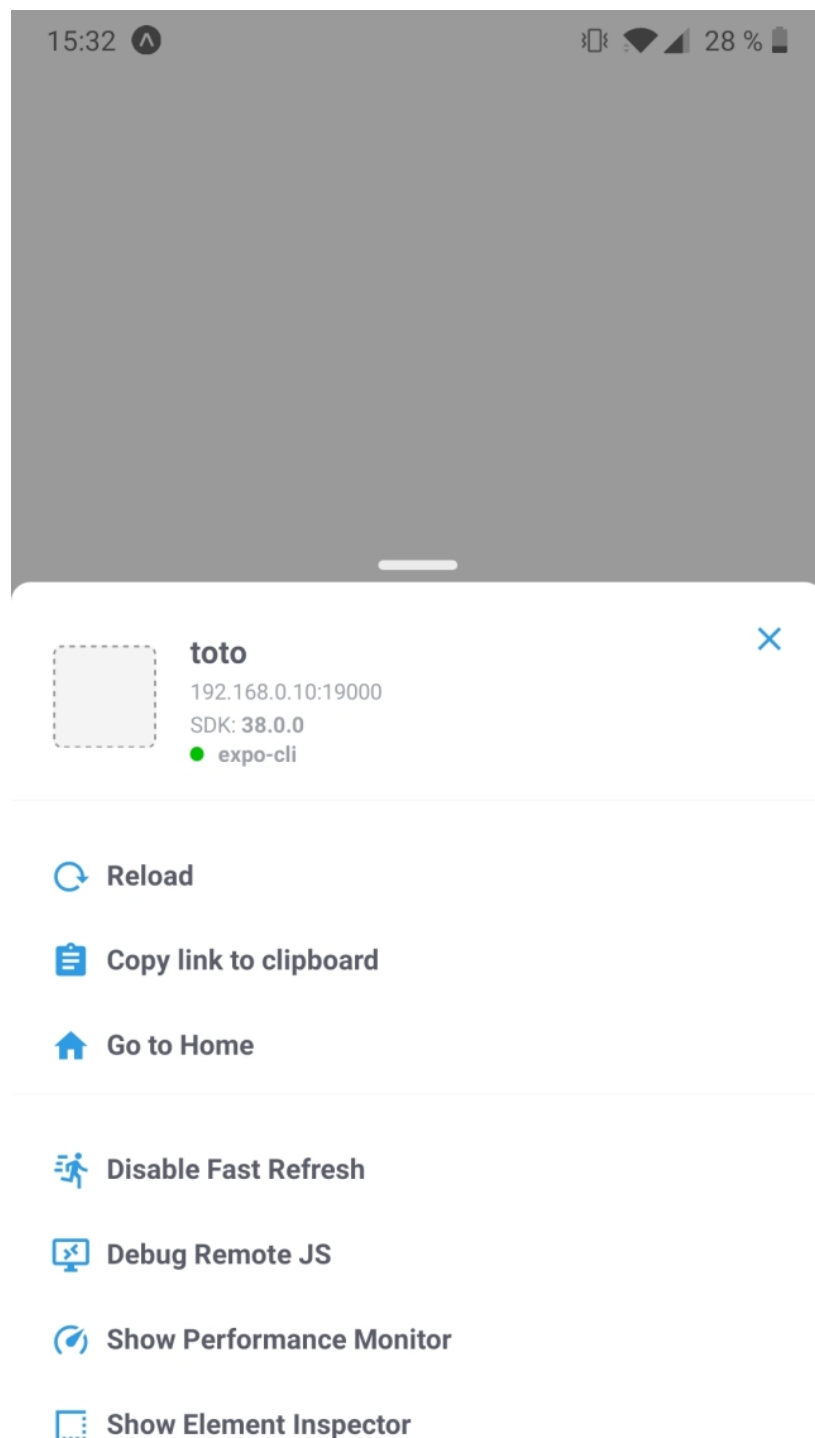
Installer des composants additionnels (API)

[desktop] Lorsque nous souhaiterons plus tard utiliser des modules natifs disponibles via Expo (comme le module de caméra), il nous faudra utiliser la commande `expo install <package>` (par exemple, `expo install expo-camera`). Cette commande permet d'initialiser l'API qui est utilisable en JavaScript.

[cloud] Pour la version Cloud web, c'est plus simple : il nous suffit d'importer la librairie dans le code JavaScript avec l'instruction `import Camera from 'expo-camera';` et le système détectera tout seul que l'installation est nécessaire, et la fera.

Les fonctionnalités d'Expo

Lorsque l'application est ouverte, nous pouvons secouer notre téléphone pour faire apparaître le menu de développement.





[desktop] Quand nous utilisons la version installée sur notre ordinateur, nous pouvons activer/désactiver un certain nombre de comportements, comme affichés sur la capture d'écran.

- Fast Refresh permet une mise à jour graphique sans relancer l'application : c'est ce que nous avons observé en modifiant le texte, précédemment.
- Remote JS permet d'ouvrir le *debugger* dans un onglet du navigateur et de faire tourner le code de l'application à l'intérieur : nous en parlerons dans un prochain chapitre.
- Show Performance Monitor permet d'afficher un certain nombre de métriques sur les performances de l'application.
- Show Element Inspector permet de *debugger* l'interface utilisateur : nous l'utiliserons également plus tard.

[cloud] Quand nous utilisons la version Cloud, la plupart des configurations sont déjà activées. Nous n'avons donc que très peu de marge de manœuvre.

Syntaxe À retenir

- Expo fournit une CLI très performante pour simplifier les tâches de développement. C'est un outil complet que nous allons utiliser dans le reste du cours.
- Nous pourrions utiliser la version  Cloud (marquée par un petit nuage) sur les autres chapitres, sauf indication contraire. Dans ces cas, nous l'indiquerons avec un icône  (desktop).

Complément

Expo CLI¹

VII. Exercice : Appliquez la notion

Question

[solution n°3 p.20]

Maintenant que nous avons vu et compris tout cela, créons une application en mode installé sur notre ordinateur.

Pour cela, il faut avoir la CLI d'Expo installée et initialiser un projet comme vu dans ce cours.

Ensuite, désactivons `Fast Refresh` en secouant le téléphone et changeons encore le texte à l'écran. On constate que cela ne change plus. Secouons le téléphone de nouveau et cliquons sur `Reload` pour observer la modification.

VIII. Essentiel

IX. Auto-évaluation

A. Exercice final

Exercice 1

[solution n°4 p.20]

Exercice

Si l'on devait résumer Expo en une seule phrase...

- ☐ C'est la seule façon de faire des applications React Native
- ☐ C'est un framework pour React Native qui simplifie grandement l'expérience de développement
- ☐ C'est une librairie de composants pour utiliser la caméra du téléphone

¹ <https://docs.expo.io/workflow/expo-cli/>

Exercice

Combien de workflow différents sont possibles avec Expo ?

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4

Exercice

De quelle manière le thread JavaScript pilote-t-il le rendu de l'application ?

- ☐ En affichant du HTML et du CSS via JavaScript, le tout dans une *webview*
- ☐ En rendant directement des composants natifs spécifiés dans le code JavaScript
- ☐ En communiquant avec le bridge via des instructions sérialisées qui sont interprétées et retranscrites au thread natif

Exercice

Laquelle de ces fonctionnalités ne fait pas partie des outils intégrés à Expo ?

- ☐ Fast Refresh
- ☐ Remote JS
- ☐ Show Element Inspector
- ☐ Power Consumption

Exercice

La commande `expo start` permet de compiler le projet pour la production.

- ☐ Vrai
- ☐ Faux

Exercice

Dans quel but la commande `expo doctor` existe-t-elle ?

- ☐ Elle affiche une page de démonstration avec une icône médicale
- ☐ Elle permet de diagnostiquer une panne sur l'environnement de développement
- ☐ Elle permet de réparer une application qui démarre, mais affiche un écran d'erreur

Exercice

Pour installer un package Expo, j'utilise la commande...

- ☐ `expo install`
- ☐ `expo <nom_du_package>`
- ☐ `expo install <nom_du_package>`

Exercice

Sur un projet en équipe, j'utilise de préférence...

- ☐ [cloud] Le mode Cloud web d'Expo
- ☐ [desktop] Le mode installé sur son PC d'Expo

Exercice

Je n'ai besoin que de la caméra du téléphone et des fonctionnalités de base, quel workflow est le plus adapté à la situation ?

- ☐ Le workflow Managed
- ☐ Le workflow Bare
- ☐ Le mode ExpoKit

Exercice

Il est possible de migrer d'un projet Expo Bare vers Managed.

- ☐ Vrai
- ☐ Faux

B. Exercice : Défi

Super ! Maintenant que nous avons vu toutes ces notions, vous allez créer votre première application mobile.

Question

[solution n°5 p.22]

Rendez-vous sur <https://snack.expo.io/>, puis flashez le QR code comme nous l'avons vu dans ce cours avec l'appli Expo. L'application devrait désormais apparaître sur le téléphone.

L'objectif de ce défi va être de vous présenter très succinctement en affichant dans l'application votre prénom, votre ville et votre genre (masculin, féminin...).

Pour cela, il faudra utiliser d'autres composants `<Text>` en leur passant une propriété `style` qui utilisera cette fois la clé `bioElement` de l'objet `styles`, plutôt que celle `title`, contrairement au titre.

Voici une capture d'écran pour montrer un exemple du rendu final à obtenir.

```
1 import * as React from 'react';
2 import { Text, View, StyleSheet } from 'react-native';
3 import Constants from 'expo-constants';
4
5 export default function App() {
6   return (
7     <View style={styles.container}>
8       <Text style={styles.title}>
9         Ma fiche biographique
10      </Text>
11      <View>
12        <Text style={styles.bioElement}>• Remplaçons ce composant texte par notre fiche
13        biographique</Text>
14      </View>
15    </View>
16  );
17 }
18
19 const styles = StyleSheet.create({
20   container: {
21     flex: 1,
22     justifyContent: 'center',
```



```
22     paddingTop: Constants.statusBarHeight,  
23     backgroundColor: '#ecf0f1',  
24     padding: 8,  
25   },  
26   title: {  
27     margin: 24,  
28     fontSize: 18,  
29     fontWeight: 'bold',  
30     textAlign: 'center',  
31   },  
32   bioElement: {  
33     fontSize: 14,  
34     textAlign: "left"  
35   }  
36 });
```


Ma fiche biographique

- Prénom : Andréas
- Ville : Montpellier
- Genre : Masculin

Solutions des exercices


Exercice p. 6 Solution n°1**Exercice**

Une application React Native est...

- ☐ Pas compilée du tout
- ☒ Partiellement compilée
- ☐ Entièrement compilée
-  Une application React Native est partiellement compilée, car son code natif servant à lancer le moteur JavaScript qui exécute du code non-compilé est lui-même compilé. Le code JavaScript embarqué, par contre, ne l'est pas et sera interprété au *runtime*.

Exercice

Quel est le principal avantage à l'utilisation du Managed Workflow de Expo ?

- ☐ Des performances maximales
- ☐ La possibilité d'utiliser son propre code natif
- ☒ Une expérience de développement facilitée
-  Une application dans le mode Managed ne peut pas installer son propre code natif et doit utiliser le Bare Workflow si elle le souhaite. Expo ne change pas spécialement quelque chose par rapport à React Native sur les performances, la bonne réponse est donc l'expérience de développement qui est grandement améliorée.

p. 11 Solution n°2

```
1 import * as React from 'react';
2 import { Text, View, StyleSheet } from 'react-native';
3 import Constants from 'expo-constants';
4
5 // You can import from local files
6 import AssetExample from './components/AssetExample';
7
8 // or any pure javascript modules available in npm
9 import { Card } from 'react-native-paper';
10
11 export default function App() {
12   return (
13     <View style={styles.container}>
14       <Text style={styles.paragraph}>
15         Coucou ça marche
16       </Text>
17       <Card>
18         <AssetExample />
19       </Card>
20     </View>
21   );
22 }
23
24 const styles = StyleSheet.create({
25   container: {
26     flex: 1,
```

```

27   justifyContent: 'center',
28   paddingTop: Constants.statusBarHeight,
29   backgroundColor: '#ecf0f1',
30   padding: 8,
31 },
32 paragraph: {
33   margin: 24,
34   fontSize: 18,
35   fontWeight: 'bold',
36   textAlign: 'center',
37 },
38 });

```

p. 14 Solution n°3

```

1 import * as React from 'react';
2 import { Text, View } from 'react-native';
3
4 export default function App() {
5   return (
6     <View>
7       <Text>
8         Un autre texte
9       </Text>
10    </View>
11  );
12 }
13

```

Exercice p. 14 Solution n°4

Exercice

Si l'on devait résumer Expo en une seule phrase...

- ☐ C'est la seule façon de faire des applications React Native
- ☒ C'est un framework pour React Native qui simplifie grandement l'expérience de développement
- ☐ C'est une librairie de composants pour utiliser la caméra du téléphone
- ☐ Expo permet d'utiliser la caméra du téléphone et est une des manières de faire du React Native, mais pas la seule. C'est un framework pour React Native.

Exercice

Combien de workflow différents sont possibles avec Expo ?

- ☐ 1
- ☒ 2
- ☐ 3
- ☐ 4

Q Les workflow Managed et Bare.

Exercice

De quelle manière le thread JavaScript pilote-t-il le rendu de l'application ?

- ☐ En affichant du HTML et du CSS via JavaScript, le tout dans une *webview*
- ☐ En rendant directement des composants natifs spécifiés dans le code JavaScript
- ☒ En communiquant avec le bridge via des instructions sérialisées qui sont interprétées et retranscrites au thread natif

Q Le HTML et CSS est pour les applications web embarquées. Ici, on utilise des applications hybrides. Le thread JavaScript fournit, via son code, une abstraction de la plateforme (Android / iOS) : les instructions doivent donc être interprétées par le bridge pour être utilisées par le thread natif.

Exercice

Laquelle de ces fonctionnalités ne fait pas partie des outils intégrés à Expo ?

- ☐ Fast Refresh
 - ☐ Remote JS
 - ☐ Show Element Inspector
 - ☒ Power Consumption
- Q Power Consumption est la seule instruction qui n'existe pas.

Exercice

La commande `expo start` permet de compiler le projet pour la production.

- ☐ Vrai
 - ☒ Faux
- Q `expo start` permet de lancer le mode développement avec l'interface web.


Exercice

Dans quel but la commande `expo doctor` existe-t-elle ?

- ☐ Elle affiche une page de démonstration avec une icône médicale
 - ☒ Elle permet de diagnostiquer une panne sur l'environnement de développement
 - ☐ Elle permet de réparer une application qui démarre, mais affiche un écran d'erreur
- Q `expo doctor` permet uniquement de corriger les soucis liés à l'environnement de développement Expo, pas les soucis applicatifs.


Exercice

Pour installer un package Expo, j'utilise la commande...

- ☐ expo install
- ☐ expo <nom_du_package>
- ☒ expo install <nom_du_package>
-  expo install <nom_du_package> est la seule commande qui existe.


Exercice

Sur un projet en équipe, j'utilise de préférence...

- ☐ [cloud] Le mode Cloud web d'Expo
- ☒ [desktop] Le mode installé sur son PC d'Expo
-  [desktop] Il est préférable d'utiliser le mode installé sur son PC, car l'on pourra développer à plusieurs en utilisant Git sur le projet. De plus, les options de développement et de debug sont plus précises.


Exercice

Je n'ai besoin que de la caméra du téléphone et des fonctionnalités de base, quel workflow est le plus adapté à la situation ?

- ☒ Le workflow Managed
- ☐ Le workflow Bare
- ☐ Le mode ExpoKit
-  Étant donné que le workflow Managed peut utiliser toutes les APIs présentes dans Expo et que la caméra en fait partie, on peut se contenter de l'utiliser. Sinon, nous aurions dû installer une librairie native et donc utiliser le workflow Bare.

Exercice

Il est possible de migrer d'un projet Expo Bare vers Managed.

- ☒ Vrai
- ☐ Faux
-  C'est possible, bien que très improbable, à condition que l'on n'utilise aucune API avec un code natif qui ne serait pas disponible dans Expo, et que nous n'avons pas besoin de personnaliser des paramètres de build.

p. 16 Solution n°5

```
1 import * as React from 'react';
2 import { Text, View, StyleSheet } from 'react-native';
3 import Constants from 'expo-constants';
4
5 export default function App() {
6   return (
7     <View style={styles.container}>
8       <Text style={styles.title}>Ma fiche biographique</Text>
9       <View>
10        <Text style={styles.bioElement}>• Prénom : Andréas</Text>
11        <Text style={styles.bioElement}>• Ville : Montpellier</Text>
12        <Text style={styles.bioElement}>• Genre : Masculin</Text>
13      </View>

```

```
14     </View>
15   );
16 }
17
18 const styles = StyleSheet.create({
19   container: {
20     flex: 1,
21     justifyContent: 'center',
22     paddingTop: Constants.statusBarHeight,
23     backgroundColor: '#ecf0f1',
24     padding: 8,
25   },
26   title: {
27     margin: 24,
28     fontSize: 18,
29     fontWeight: 'bold',
30     textAlign: 'center',
31   },
32   bioElement: {
33     fontSize: 14,
34     textAlign: 'left',
35   },
36 });
37
```