

Les API React Native et les modules natifs

Table des matières

I. Contexte	3
II. La gestion des permissions	3
III. Exercice : Appliquez la notion	7
IV. Utiliser des API Natives	8
V. Exercice : Appliquez la notion	12
VI. Essentiel	15
VII. Auto-évaluation	15
A. Exercice final	15
B. Exercice : Défi.....	16
Solutions des exercices	17

I. Contexte

Durée : 1 h

Environnement de travail : Une application React Native initialisée ou utiliser <https://snack.expo.io/>

Pré-requis : Pas de pré-requis

Contexte

Lorsque l'on développe une application mobile, il est souvent nécessaire d'accéder à des fonctions natives du téléphone, comme la caméra ou le module GPS. Ces fonctions sont fournies par le système d'exploitation et sont utilisables via les API Java ou Objective-C sous-jacentes. Pour cela, il faut donc utiliser des ponts JavaScript pour communiquer avec ce code natif.

De plus, pour utiliser des API natives, il est souvent nécessaire (pour des raisons de confidentialité et de vie privée) de demander des autorisations d'accès à certains éléments du téléphone, comme la caméra.

Expo permet de traiter tous ces pré-requis très simplement, et c'est ce que ce module va nous apprendre.

II. La gestion des permissions

Objectif

- Comprendre le système de permissions sur les applications mobiles

Mise en situation

Lorsqu'il s'agit d'ajouter une fonctionnalité permettant d'accéder à des informations potentiellement sensibles sur l'appareil d'un utilisateur, telles que son emplacement ou éventuellement l'envoi de notifications push, il faut d'abord demander à l'utilisateur son autorisation. Ces permissions sont maintenues en mémoire une fois accordées.

Principes généraux sur les permissions

Si nous déployons notre application sur l'AppStore d'Apple, il faudra ajouter des métadonnées supplémentaires à l'application afin de personnaliser la boîte de dialogue des autorisations du système et, plus important encore, d'expliquer pourquoi notre application nécessite des autorisations. Sans cette explication, notre application peut être rejetée de l'AppStore.

Chaque plateforme gère normalement indépendamment son mécanisme de permissions système. Il est communément admis que iOS est plus restrictif que Android. Cependant, en utilisant Expo, nous profitons d'une abstraction assez sympathique pour gérer les permissions d'accès. En effet, Expo regroupe sous une même fonction les demandes de permissions et permet donc d'appeler une seule API quelle que soit la plateforme.

Plus une application demande de permissions, plus la revue de notre application peut prendre du temps. En effet, chacune d'entre elles devra trouver une justification aux yeux des équipes en charge de faire cette revue. De plus, avoir de nombreuses demandes d'autorisation peut inquiéter l'utilisateur.

Pour cette raison, il est donc recommandé de se modérer et de bien réfléchir en amont sur les permissions qui seront nécessaires à l'utilisation de l'application.

Les types de permissions

Commençons par installer le paquet applicatif suivant si l'on se trouve sur un projet Expo sur notre ordinateur (sinon, en version web, passez cette étape) :

```
expo install expo-permissions.
```

Ce paquet contient uniquement le code spécifique pour demander les permissions : si on demande par exemple la permission `CAMERA`, il faudra utiliser le paquet spécifique de caméra, également mis à disposition par Expo.

Le tableau ci-dessous fait correspondre les permissions au package Expo lié. Cette liste est susceptible d'évoluer, il faudra donc vérifier sur la documentation d'Expo si le cas d'utilisation que l'on souhaite traiter n'y figure pas.

Type de permission	Paquets applicatifs
NOTIFICATIONS	expo-notifications
USER_FACING_NOTIFICATIONS	expo-notifications
LOCATION	expo-location
CAMERA	expo-camera, expo-barcode-scanner
AUDIO_RECORDING	expo-av
CONTACTS	expo-contacts
CAMERA_ROLL	expo-image-picker, expo-media-library
CALENDAR	expo-calendar
REMINDERS	expo-calendar
SYSTEM_BRIGHTNESS	expo-brightness
MOTION	expo-sensors

Remarque Note

Sur Android, afin de pouvoir demander des permissions et de rester compatible avec l'ancien système de droits, celles-ci doivent être définies dans le fichier `AndroidManifest.xml`. En effet, dans de précédentes versions, Android ne supportait pas les permissions au *runtime*. Expo permet cela très simplement en ajoutant une clé `android.permissions` dans le fichier `app.json` à la racine du projet. Plus d'informations ici : <https://docs.expo.io/versions/latest/sdk/permissions/#android-permissions-equivalents-inside-appjson>.

Spécificités liées aux permissions

Lorsque l'on refuse une permission, les systèmes d'exploitation disposent de sécurités permettant d'éviter d'en demander l'accès en boucle, et donc d'essayer de forcer la main à l'utilisateur. Afin de pouvoir tester les différents cas, et de les traiter en affichant une interface utilisateur cohérente, il faudra donc sûrement désinstaller l'application Expo et la réinstaller à plusieurs reprises.

Certaines permissions peuvent cependant être redemandées plusieurs fois : pour cela, il faudra regarder sur le type de retour la valeur du booléen `canAskAgain`.

Le cas échéant, une solution peut consister à rediriger l'utilisateur vers les permissions système par le biais d'un `deep link` : on sort dans ce cas du contexte de l'application.

```
1 import { Linking } from "react-native";
2
3 Linking.openURL("app-settings:")
4 // ou sur des versions récentes de React Native
5 Linking.openSettings()
```

[cf. to-settings.mp4]

Demander des permissions

Il faut d'abord importer le paquet applicatif de cette manière : `import * as Permissions from 'expo-permissions';`.

Ensuite, ce paquet nous propose deux fonctions :

- `Permissions.getAsync(...permissionTypes)` qui permet de récupérer l'état courant des permissions passées en paramètres.
- `Permissions.askAsync(...types)` qui permet de demander les permissions en paramètres, sachant que, si elles ont déjà été accordées, la réponse sera positive.

Pour ce qui concerne les types de permissions, la liste exhaustive est disponible ici : <https://docs.expo.io/versions/latest/sdk/permissions/#permissions-types>.

Ces deux fonctions retournent un type `PermissionResponse`, qui est défini comme suit :

Nom du champ	Type	Description
status	string	Le statut, qui peut être : granted, denied, undetermined
granted	boolean	Un booléen indiquant si la permission a été accordée ou non
canAskAgain	boolean	Un booléen qui détermine si on peut demander la permission une nouvelle fois. C'est faux si l'utilisateur a choisi l'option <i>Ne plus demander</i> sur Android ou a refusé la permission sur iOS Le cas échéant, ce booléen vaut vrai
ios	Dépend du type de permission	Détails additionnels sur iOS (optionnel)
android	Dépend du type de permission	Détails additionnels sur Android (optionnel)

En réalité, il existe encore une clé un peu spécifique. Voyons cela avec l'exemple ci-dessous :

```
1 {
2   status, // Combinaison de tous les statuts des demandes de permission, si l'une d'entre
           // elles est différente de 'granted' alors son statut est reporté ici
3   expires, // Combinaison de toutes les valeurs de expires pour toutes les demandes de
           // permissions, comme pour status
4   canAskAgain,
5   granted,
6   permissions: { // Un objet avec une entrée pour chaque type de permission demandé
```

```

7   [Permissions.TYPE]: {
8     status,
9     expires,
10    canAskAgain,
11    granted,
12    ...
13  },
14  ...
15 },
16 }

```

La clé `permissions` contient toutes les demandes réalisées en fonction de leur type. C'est une `map` qui a pour clé le type de permission et en valeur un objet `PermissionResponse`. Les clés de l'objet global sont une combinaison de l'ensemble des permissions demandées : par exemple, si toutes les permissions demandées sont `granted`, alors la clé de premier niveau sera `status: granted`. Ou encore, quand la clé de premier niveau `expires` prend la valeur de la clé `expires` pour laquelle la date d'expiration de la permission est la plus basse, si aucune des permissions demandées n'expire, alors elle prend la valeur `never`. Plus de détails ici : <https://docs.expo.io/versions/latest/sdk/permissions/#returns>.

Exemple

```

1 import React from "react";
2 import { View, Button } from "react-native";
3 import * as Permissions from "expo-permissions";
4
5 export default function App() {
6   async function alertIfCameraDisabledAsync() {
7     const { status } = await Permissions.getAsync(Permissions.CAMERA);
8
9     if (status !== "granted") {
10      alert(
11        "Hey ! Si je n'ai pas accès à la caméra tu ne pourras pas prendre de photos depuis
12      l'app !"
13      );
14    }
15  }
16
17  async function checkMultiPermissions() {
18    const { status } = await Permissions.getAsync(
19      Permissions.CALENDAR,
20      Permissions.CONTACTS
21    );
22    if (status !== "granted") {
23      alert(
24        "Il faut avoir les permissions d'accès au calendrier et aux contacts pour
25      continuer..."
26      );
27    }
28  }
29
30  return (
31    <View style={{ padding: 30 }}>
32      <Button
33        title="Vérifier la permissions CAMERA"
34        onPress={alertIfCameraDisabledAsync}
35      />
36      <Button
37        title="Vérifier la permissions CALENDAR et CONTACTS en même temps"
38      />
39    </View>
40  );
41 }

```

```

36     onPress={checkMultiPermissions}
37   />
38 </View>
39 );
40 }

```

[cf. check-perms.mp4]

De la même manière, l'exemple ci-dessous permet de récupérer la position GPS du téléphone. Attention, le paquet `expo-location` est nécessaire.

```

1 async function getLocationAsync() {
2   const { status, permissions } = await Permissions.askAsync(Permissions.LOCATION);
3   if (status === 'granted') {
4     return Location.getCurrentPositionAsync({ enableHighAccuracy: true });
5   } else {
6     throw new Error('Location permission not granted');
7   }
8 }

```

Maintenant que nous avons les bases, commençons à utiliser quelques modules natifs.

Syntaxe À retenir

- Pour utiliser des composants système tels que le GPS ou la caméra, il faut demander des permissions au système d'exploitation qui protège les données personnelles des utilisateurs.
- Expo fournit une abstraction des API natives qui permet de demander les permissions au système à travers deux fonctions très simples, qui renvoient l'état d'acceptation desdites permissions.

Complément

- <https://docs.expo.io/versions/latest/sdk/permissions>

III. Exercice : Appliquez la notion

Question

[solution n°1 p.19]

Dans cet exercice, il va falloir demander la permission d'accéder à la liste des contacts du téléphone, et afficher le statut de récupération de cette permission à l'écran, avec `Permissions Status : true/false`.

Si la permission est refusée, on affiche une modale d'alerte qui renvoie via un *deep link* aux permissions du système pour l'application, ou qui propose de fermer la fenêtre et de quitter l'application.

Pour le résultat attendu, dans cet exercice, nous commençons très simplement. Affichons donc un écran de la sorte :

[cf. result_ex_1.mp4]

Indice :

La permission à demander est : `Permissions.CONTACTS`.

Indice :

Pour lancer une modale, on peut utiliser `Alert.alert` du module `react-native`. Le troisième paramètre est un tableau des choix possibles sur les boutons de la modale.

Indice :

Pour rediriger vers les paramètres système avec un *deep link*, on peut utiliser la fonction suivante :

```
1 import { Linking } from "react-native";
2
3 Linking.openURL("app-settings:")
4 // ou sur des versions récentes de React Native
5 Linking.openSettings()
```

IV. Utiliser des API Natives

Objectifs

- Voir quelques exemples d'utilisation d'API natives
- Apprendre à les utiliser

Mise en situation

Maintenant, on sait comment récupérer les permissions système. On va donc pouvoir utiliser quelques API natives. Pour cela, c'est très simple : chaque module est disponible via un paquet applicatif qu'il suffit d'importer et d'utiliser.

Nous allons voir un exemple pour récupérer les données spécifiques du téléphone et l'utilisation du GPS.

Récupérer les informations du terminal mobile

Pour récupérer les métadonnées comme le numéro de série, la version du système d'exploitation ou encore le modèle du téléphone, on utilise un pont natif fourni par Expo.

On commence par installer ce qu'il nous faut sur la version installée sur ordinateur d'Expo, ou bien on peut passer cette étape sur la version web Cloud : `expo install expo-device`.

Il suffit ensuite d'importer très simplement le paquet applicatif, sachant que, pour ce type de demande, il n'est pas nécessaire de demander de permission : `import * as Device from 'expo-device';`

On a ensuite à notre disposition un ensemble de constantes, remplies avec les données que l'on peut récupérer. La liste exhaustive est consultable ici : <https://docs.expo.io/versions/v38.0.0/sdk/device/#constants>.

Dans l'exemple ci-dessous, on va récupérer le nom du terminal, ainsi que son modèle, et l'afficher à l'écran.

Exemple

```
1 import React from "react";
2 import { Text, View } from "react-native";
3 import * as Device from "expo-device";
4
5 export default function App() {
6   return (
7     <View style={{ padding: 50 }}>
8       <Text>Nom du terminal : {Device.deviceName}</Text>
9       <Text>Modèle du terminal : {Device.modelName}</Text>
10    </View>
11  );
12 }
```


12:29



85 %

Nom du terminal : Oneplus 6

Modèle du terminal : ONEPLUS A6003

Se connecter au GPS du terminal

Ici, comme toujours sur la version installée en local, il faut lancer `expo install expo-location`. Sinon, sur la version web Cloud, on peut sauter cette étape.

L'exemple ci-dessous récupère la position actuelle au clic sur un bouton : notons qu'il existe un temps de latence pendant lequel le système recalcule la position, ce qui explique que le timestamp ne change pas tout de suite (et ce afin de préserver la batterie).

Exemple

```
1 import React, { useState } from "react";
2 import { Button, Text, View, StyleSheet } from "react-native";
3 import * as Location from "expo-location";
4
5 export default function App() {
6   const [location, setLocation] = useState(null);
7   const [errorMsg, setErrorMsg] = useState(null);
8
9   async function getUserLocation() {
10     const { status } = await Location.requestPermissionsAsync();
11
12     if (status !== "granted") {
13       setErrorMsg("La permission d'accès a été refusée...");
14     }
15
16     const location = await Location.getCurrentPositionAsync({});
17     setLocation(location);
18   }
19
20   let text = "En attente de réception..";
21   if (errorMsg) {
22     text = errorMsg;
23   } else if (location) {
24     text = JSON.stringify(location, null, 2);
25   }
26
27   return (
28     <View style={styles.container}>
29       <Button
30         title="Récupérer la position GPS"
31         onPress={() => getUserLocation()}
32       />
33       <Text style={styles.text}>{text}</Text>
34     </View>
35   );
36 }
37
38 const styles = StyleSheet.create({
39   container: {
40     flex: 1,
41     padding: 50,
42     alignItems: "center",
43   },
44 });
```

12:40



Récupérer la position GPS

```
{
  "coords": {
    "altitude": 0,
    "altitudeAccuracy": -1,
    "latitude": 37.785834,
    "accuracy": 5,
    "longitude": -122.406417,
    "heading": -1,
    "speed": -1
  },
  "timestamp": 1600252791963.403
}
```

Dans l'exemple ci-dessus, on utilise une méthode utilitaire `Location.requestPermissionsAsync()` qui permet de récupérer toutes les permissions nécessaires à l'utilisation du module. Sous le capot, cela utilise la méthode classique de récupération de permissions que nous avons vue précédemment. Mais Expo aimant simplifier la vie des développeurs, il met à disposition ce genre de petites fonctions qui nous permet encore de gagner de précieuses secondes.

Syntaxe À retenir

- Il est très simple d'utiliser des modules natifs avec Expo : on pourra par exemple utiliser le GPS pour récupérer les données du terminal et afficher une carte, ou encore des informations relatives au terminal telles que le modèle, et alimenter nos fichiers de logs pour tracer des bugs éventuels sur certains modèles.
- Chaque module natif est isolé dans un paquet applicatif qui lui est propre : il faut en consulter la liste exhaustive sur la documentation d'Expo <https://docs.expo.io/versions/latest/>.

Complément

- <https://docs.expo.io/versions/latest/>

V. Exercice : Appliquez la notion

Question

[solution n°2 p.19]

Nous allons améliorer l'exercice vu en première partie (code ci-dessous en indice) pour afficher la liste des contacts du répertoire téléphonique dans l'interface.

Pour ce faire, il faut toujours demander les permissions au lancement de l'application, puis, si elles sont refusées, afficher une modale qui renvoie vers le gestionnaire des paramètres de l'application via un *deep link*, ou proposer de quitter.

Une fois les permissions récupérées, il faut afficher les informations du contact : on veut une page de 3 contacts uniquement.

Pour chaque contact sera affiché son nom, son poste et sa société, si ces informations sont connues.

Voici un exemple du rendu escompté ci-dessous :

Carrier 

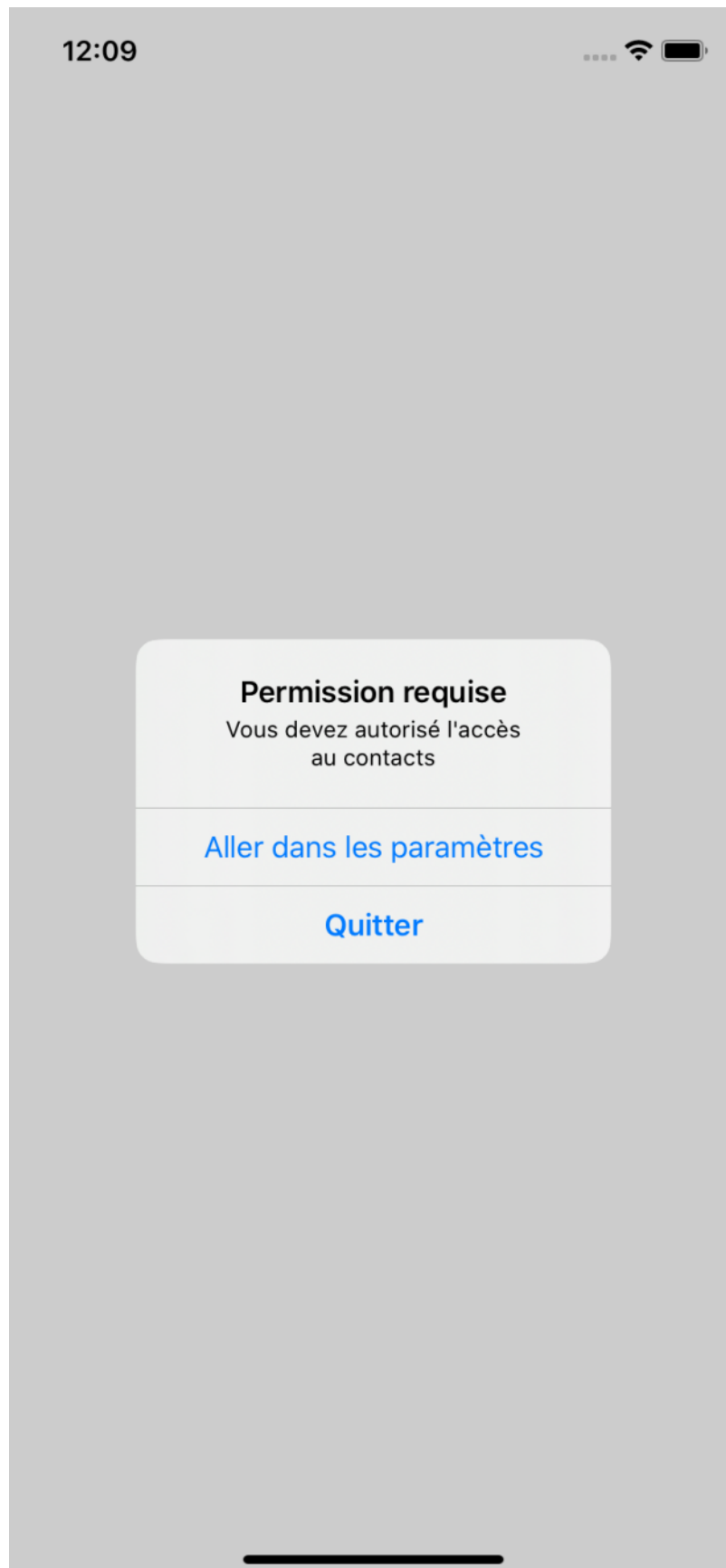
12:43 PM



Kate Bell - Producer@Creative Consulting

Daniel Higgins Jr.

John Appleseed



Indice :

La fonction permettant de récupérer seulement trois contacts et leurs informations est :

```
1 const { data } = await Contacts.getContactsAsync({ pageSize: 3 });
```

VI. Essentiel

Les modules natifs sont des ponts **JavaScript / code natif** permettant d'accéder à des éléments du système d'exploitation du terminal sous-jacent. On pourra par exemple accéder à la caméra du téléphone ou encore à la position GPS, permettant d'afficher l'utilisateur sur une carte en temps réel.

Pour accéder à ces modules, il faut demander l'accès à des permissions. En effet, les fabricants, afin de garantir l'accès à des données sensibles, ont rajouté un grand nombre de sécurités.

Pour demander des permissions, c'est également très simple, mais il faut bien évaluer quelles permissions on demande, car plus on en demande, plus on met des barrières à l'installation de notre application.

Enfin, chaque module est accessible par son propre paquet applicatif, ce qui permet de composer notre application de manière ultra-modulaire.

VII. Auto-évaluation

A. Exercice final

Exercice 1

[solution n°3 p.21]

Exercice

Toutes les API natives ont besoin de demander des permissions.

- ☐ Vrai
- ☐ Faux

Exercice

Il faut utiliser des API différentes pour récupérer les permissions selon qu'on est sur iOS ou Android.

- ☐ Vrai
- ☐ Faux

Exercice

Pour demander des permissions avec Expo, j'utilise...

- ☐ expo-get-perms
- ☐ expo-permissions
- ☐ permissons-manager

Exercice

Parmi ces permissions, laquelle n'existe pas ?

- ☐ USER_FACING_NOTIFICATIONS
- ☐ USER_WALKING_DAY_ACTIVITY
- ☐ CAMERA
- ☐ LOCATION

Exercice

`Permissions.askAsync(...types)` prend en paramètre...

- ☐ Un tableau de type de permissions
- ☐ Un nombre infini de paramètres étant des types de permissions

Exercice

Si l'utilisateur refuse une permission, il est possible de la redemander de la même manière depuis l'application.

- ☐ Vrai sur Android et iOS
- ☐ Vrai sur Android dans certains cas
- ☐ Vrai sur iOS dans certains cas
- ☐ Faux pour Android et iOS

Exercice

Si l'utilisateur refuse une permission et que je ne peux pas la redemander de la même manière, que puis-je faire ?

- ☐ Passer le paramètre `force` à `true`, pour lui forcer la fenêtre de permissions de nouveau
- ☐ Un *deep link* vers les paramètres du système

Exercice

Quelle valeur peut prendre la clé `status` d'une `PermissionResponse` ?

- ☐ `granted`
- ☐ `denied`
- ☐ `undetermined`

Exercice

Pour récupérer la position GPS du terminal, j'utilise le package...

- ☐ `expo-location`
- ☐ `expo-position`
- ☐ `expo-gps`

Exercice

C'est une bonne pratique de mettre toutes les permissions disponibles dans notre appli, comme ça, « on est tranquille ».

- ☐ Vrai
- ☐ Faux

B. Exercice : Défi

Pour cet exercice final, nous allons réaliser un enregistreur vidéo, qui enregistre une vidéo et affiche une prévisualisation de la vidéo enregistrée.

Question

[solution n°4 p.23]

Pour cela, il faudra avoir accès au module caméra `expo-camera` et au module `expo-av` pour pouvoir afficher la vidéo.

L'application affiche le module caméra en plein écran avec un bouton d'enregistrement qui change d'état quand la vidéo est en cours d'enregistrement. À la fin de l'enregistrement, une fenêtre de prévisualisation apparaît et nous montre notre enregistrement.

Attention, ce module n'est pas compatible avec les émulateurs : il faudra donc un vrai terminal ou utiliser la version web d'Expo et le mode rendu web (plutôt que sur Android/iOS).

(Attention, vous allez avoir besoin du `useRef()` lors de cet exercice. Si la notion est encore assez floue pour vous, n'hésitez pas à aller revoir le cours sur les animations : "L'animation avec React Native".)

Ci-dessous, le rendu final escompté :

[cf. `api5.mp4`]

Solutions des exercices

p.7 Solution n°1

```

1 import React, { useEffect, useState } from "react";
2 import { Text, Linking, Alert, View } from "react-native";
3 import * as Permissions from "expo-permissions";
4
5 export default function App() {
6   const [permissionsGranted, setPermissionGranted] = useState(false);
7
8   useEffect(() => {
9     async function playEffect() {
10       const { status } = await Permissions.askAsync(Permissions.CONTACTS);
11
12       if (status !== "granted") {
13         Alert.alert(
14           "Permission requise",
15           "Vous devez autoriser l'accès aux contacts",
16           [
17             {
18               text: "Quitter",
19               style: "cancel",
20             },
21             {
22               onPress: () => Linking.openURL("app-settings:"),
23               text: "Aller dans les paramètres",
24             },
25           ],
26         );
27         return;
28       }
29       setPermissionGranted(true);
30     }
31
32     playEffect();
33   }, []);
34
35   return (
36     <View style={{ padding: 50 }}>
37       <Text>Permissions Status : {permissionsGranted.toString()}</Text>
38     </View>
39   );
40 }

```

p.12 Solution n°2

```

1 import React, { useEffect, useState } from "react";
2 import { Text, Linking, Alert, View, ScrollView } from "react-native";
3 import * as Permissions from "expo-permissions";
4 import * as Contacts from "expo-contacts";
5
6 export default function App() {
7   const [permissionsGranted, setPermissionGranted] = useState(false);
8   const [contacts, setContacts] = useState([]);
9
10  useEffect(() => {

```

```

11  async function playEffect() {
12      const { status } = await Permissions.askAsync(Permissions.CONTACTS);
13
14      if (status !== "granted") {
15          Alert.alert(
16              "Permission requise",
17              "Vous devez autoriser l'accès aux contacts",
18              [
19                  {
20                      text: "Quitter",
21                      style: "cancel",
22                  },
23                  {
24                      onPress: () => Linking.openURL("app-settings:"),
25                      text: "Aller dans les paramètres",
26                  },
27              ]
28          );
29          return;
30      }
31      const { data } = await Contacts.getContactsAsync({ pageSize: 3 });
32      setPermissionGranted(true);
33      setContacts(data);
34  }
35
36  playEffect();
37  }, []);
38
39  if (!permissionsGranted) {
40      return (
41          <View style={{ padding: 50 }}>
42              <Text onPress={() => Linking.openURL("app-settings:")}>
43                  Il faut accepter les permissions pour accéder à l'application. Cliquez
44                  sur ce texte pour accéder aux paramètres.
45              </Text>
46          </View>
47      );
48  }
49
50  return (
51      <ScrollView style={{ padding: 50 }}>
52          {contacts.length === 0 ? (
53              <Text>Vous n'avez aucun contact dans votre répertoire.</Text>
54          ) : (
55              contacts.map((contact) => (
56                  <Text key={contact.id} style={{ marginVertical: 10 }}>
57                      {contact.name}
58                      {contact.company && contact.jobTitle
59                          ? " - " + contact.jobTitle + "@" + contact.company
60                          : null}
61                  </Text>
62              ))
63          )}
64      </ScrollView>
65  );
66 }
67

```

Pour répondre à la demande, il est nécessaire de stocker la liste des contacts dans le state de l'application après avoir obtenu l'autorisation d'y accéder :

```

1 [...]
2 const [contacts, setContacts] = useState([]);
3 [...]
4 useEffect(() => {
5   async function playEffect() {
6     const { data } = await Contacts.getContactsAsync({ pageSize: 3 });
7     setPermissionGranted(true);
8     setContacts(data);
9   }
10 [...]
```

Il suffit ensuite de les afficher sous forme de liste en concaténant les informations demandées :

```

1 <ScrollView style={{ padding: 50 }}>
2   {contacts.length === 0 ? (
3     <Text>Vous n'avez aucun contact dans votre répertoire.</Text>
4   ) : (
5     contacts.map((contact) => (
6       <Text key={contact.id} style={{ marginVertical: 10 }}>
7         {contact.name}
8         {contact.company && contact.jobTitle
9           ? " - " + contact.jobTitle + "@" + contact.company
10          : null}
11       </Text>
12     ))
13   )}
14 </ScrollView>
```


Exercice p. 15 Solution n°3

Exercice

Toutes les API natives ont besoin de demander des permissions.

☐ Vrai

☒ Faux


 Seulement certaines API accédant à des données sensibles, comme la localisation ou la caméra, doivent demander des permissions.

Exercice

Il faut utiliser des API différentes pour récupérer les permissions selon qu'on est sur iOS ou Android.


☐ Vrai

☒ Faux

 Non, car Expo fournit une abstraction à ce niveau et permet donc d'appeler une seule fonction, quel que soit le système cible.

Exercice


Pour demander des permissions avec Expo, j'utilise...

- ☐ expo-get-perms
- ☒ expo-permissions
- ☐ permisisions-manager
-  La seule réponse qui existe est la deuxième.

Exercice

Parmi ces permissions, laquelle n'existe pas ?


- ☐ USER_FACING_NOTIFICATIONS
- ☒ USER_WALKING_DAY_ACTIVITY
- ☐ CAMERA
- ☐ LOCATION

 On ne peut pas récupérer l'activité de marche de la journée de cette manière. Pour les autres, elles existent bien.

Exercice

`Permissions.askAsync(...types)` prend en paramètre...


- ☐ Un tableau de type de permissions
- ☒ Un nombre infini de paramètres étant des types de permissions

 `askAsync` prend un nombre illimité de paramètres. Pour cela, il suffit d'enchaîner les différentes permissions avec une virgule.

Exercice

Si l'utilisateur refuse une permission, il est possible de la redemander de la même manière depuis l'application.


- ☐ Vrai sur Android et iOS
- ☒ Vrai sur Android dans certains cas
- ☐ Vrai sur iOS dans certains cas
- ☐ Faux pour Android et iOS

 Sur Android et uniquement Android, il est possible de redemander une permission si l'utilisateur l'a refusée sans avoir coché « Ne plus demander ».

Exercice

Si l'utilisateur refuse une permission et que je ne peux pas la redemander de la même manière, que puis-je faire ?

- ☐ Passer le paramètre `force` à `true`, pour lui forcer la fenêtre de permissions de nouveau
- ☒ Un *deep link* vers les paramètres du système

 La seule solution éthique (et qui existe, d'ailleurs) est d'utiliser un *deep link* vers les paramètres système.

```
1 import { Linking } from "react-native";
2
3 Linking.openURL("app-settings:");
```

Exercice

Quelle valeur peut prendre la clé `status` d'une `PermissionResponse` ?

- ☒ `granted`
- ☒ `denied`
- ☒ `undetermined`

🔍 Les trois valeurs sont possibles. La troisième sera dans le cas où plusieurs permissions ont des valeurs différentes d'acceptation. Dans ce cas, la permission `root` ne sera pas capable de déterminer l'issue du statut.

Exercice

Pour récupérer la position GPS du terminal, j'utilise le package...

- ☒ `expo-location`
 - ☐ `expo-position`
 - ☐ `expo-gps`
- 🔍 Seul le package `expo-location` existe.

Exercice

C'est une bonne pratique de mettre toutes les permissions disponibles dans notre appli, comme ça, « on est tranquille ».

- ☐ Vrai
- ☒ Faux

🔍 Bien évidemment que non ! Plus l'on rajoute de permissions, plus les utilisateurs vont être méfiants vis-à-vis de notre application. De plus, il faut pouvoir justifier auprès des magasins applicatifs chaque permission.

p. 17 Solution n°4

```
1 import React, { useState, useEffect, useRef } from "react";
2 import {
3   Text,
4   View,
5   Button,
6   TouchableOpacity,
7   StyleSheet,
8   ActivityIndicator,
9 } from "react-native";
10 import { Camera } from "expo-camera";
11 import * as Permissions from "expo-permissions";
12 import { Video } from "expo-av";
13
14 export default function App() {
15   const [hasPermission, setHasPermission] = useState(null);
16   const [isRecording, setRecording] = useState(false);
17   const [videoPreviewUrl, setVideoPreviewUrl] = useState(null);
18   const camera = useRef();
19
20   useEffect(() => {
```

```

21 // C'est une IIFE : https://developer.mozilla.org/fr/docs/Glossaire/IIFE
22 (async () => {
23     const { status } = await Permissions.askAsync(
24         Permissions.CAMERA,
25         Permissions.AUDIO_RECORDING
26     );
27     setHasPermission(status === "granted");
28 })();
29 }, []);
30
31 // En attendant de recevoir les permissions
32 if (hasPermission === null) {
33     return <ActivityIndicator />;
34 }
35
36 // Si les permissions sont refusées
37 if (hasPermission === false) {
38     return (
39         <View style={{ padding: 50 }}>
40             <Text>Aucun accès à la camera</Text>
41         </View>
42     );
43 }
44
45 // Si on a une video de preview, qu'on vient d'enregistrer
46 if (videoPreviewUrl) {
47     return (
48         <View style={{ padding: 50 }}>
49             <Video
50                 source={videoPreviewUrl}
51                 resizeMode="contain"
52                 shouldPlay
53                 isLooping
54                 style={styles.videoPreview}
55             />
56             <Button
57                 title="Fermer la prévisualisation"
58                 onPress={() => setVideoPreviewUrl(null)}
59             />
60         </View>
61     );
62 }
63
64 return (
65     <Camera
66         style={styles.camera}
67         type={Camera.Constants.Type.back}
68         ref={camera}
69     >
70     <View style={styles.absolutePositionner}>
71         <RecordingButton
72             isRecording={isRecording}
73             onStartPress={async () => {
74                 setRecording(true);
75                 const targetFile = await camera.current.recordAsync();
76                 setVideoPreviewUrl(targetFile);
77             }}
78             onStopPress={() => {

```



```

79         camera.current.stopRecording();
80         setRecording(false);
81     }}
82     />
83 </View>
84 </Camera>
85 );
86 }
87
88 // Les styles de l'app
89 const styles = StyleSheet.create({
90   camera: {
91     flex: 1,
92   },
93   videoPreview: {
94     width: "100%",
95     aspectRatio: 1,
96     marginBottom: 50,
97   },
98   absolutePositionner: {
99     position: "absolute",
100    flexDirection: "row",
101    justifyContent: "center",
102    bottom: 50,
103    right: 0,
104    left: 0,
105  },
106 });
107
108 // Le bouton qui permet d'enregistrer
109 const RecordingButton = (props) =>
110   props.isRecording ? (
111     <TouchableOpacity
112       onPress={props.onStopPress}
113       style={[
114         recordBtnStyled.buttonContainer,
115         recordBtnStyled.buttonStopContainer,
116       ]}
117     >
118       <View style={recordBtnStyled.buttonStop}></View>
119     </TouchableOpacity>
120   ) : (
121     <TouchableOpacity
122       onPress={props.onStartPress}
123       style={[recordBtnStyled.buttonContainer]}
124     >
125       <View style={recordBtnStyled.circleInside}></View>
126     </TouchableOpacity>
127   );
128
129 // Ses styles
130 const recordBtnStyled = StyleSheet.create({
131   buttonContainer: {
132     width: 80,
133     height: 80,
134     borderRadius: 40,
135     backgroundColor: "#D91E18",
136     alignItems: "center",

```

```

137   justifyContent: "center",
138   borderWidth: 5,
139   borderColor: "white",
140 },
141 circleInside: {
142   width: 60,
143   height: 60,
144   borderRadius: 30,
145   backgroundColor: "#D91E18",
146 },
147 buttonStopContainer: {
148   backgroundColor: "transparent",
149 },
150 buttonStop: {
151   backgroundColor: "#D91E18",
152   width: 40,
153   height: 40,
154   borderRadius: 3,
155 },
156 });

```

Commençons par définir un composant représentant le bouton d'enregistrement d'une vidéo. Ce composant permettra de lancer l'enregistrement ou de l'arrêter si celui-ci est déjà en cours. Pour cela, un booléen permettant de connaître l'état de l'enregistrement, ainsi qu'une fonction de lancement et une fonction d'arrêt, seront passés en props :

```

1 const RecordingButton = (props) =>
2   props.isRecording ? (
3     <TouchableOpacity
4       onPress={props.onStopPress}
5       style={[
6         recordBtnStyled.buttonContainer,
7         recordBtnStyled.buttonStopContainer,
8       ]}
9     >
10      <View style={recordBtnStyled.buttonStop}></View>
11    </TouchableOpacity>
12  ) : (
13    <TouchableOpacity
14      onPress={props.onStartPress}
15      style={[recordBtnStyled.buttonContainer]}
16    >
17      <View style={recordBtnStyled.circleInside}></View>
18    </TouchableOpacity>
19  );
20 [...]

```

À présent, nous allons pouvoir utiliser ce composant dans notre composant principal App. Mais, tout d'abord, il est nécessaire de s'assurer que les permissions nécessaires sont accordées. Pour cela, on peut utiliser une structure déclarative de fonction permettant son exécution immédiatement :

```

1 useEffect(() => {
2   // C'est une IIFE : https://developer.mozilla.org/fr/docs/Glossaire/IIFE
3   (async () => {
4     const { status } = await Permissions.askAsync(
5       Permissions.CAMERA,
6       Permissions.AUDIO_RECORDING
7     );
8     setHasPermission(status === "granted");
9   })();

```

```
10 }, []);
```

De cette manière, à l'ouverture de l'application, on stocke dans le state le statut des permissions demandées et on va afficher un loader tant que celles-ci ne sont pas connues, ou un message si elles sont refusées :

```
1 // En attendant de recevoir les permissions
2 if (hasPermission === null) {
3   return <ActivityIndicator />;
4 }
5
6 // Si les permissions sont refusées
7 if (hasPermission === false) {
8   return (
9     <View style={{ padding: 50 }}>
10      <Text>Aucun accès à la camera</Text>
11    </View>
12  );
13 }
```

On définit ensuite dans l'interface le composant <Camera>, auquel on ajoute le bouton d'enregistrement <RecordingButton> avec ses méthodes de lancement et d'arrêt d'enregistrement :

```
1 return (
2   <Camera
3     style={styles.camera}
4     type={Camera.Constants.Type.back}
5     ref={camera}
6   >
7     <View style={styles.absolutePositionner}>
8       <RecordingButton
9         isRecording={isRecording}
10        onStartPress={async () => {
11          setRecording(true);
12          const targetFile = await camera.current.recordAsync();
13          setVideoPreviewUrl(targetFile);
14        }}
15        onStopPress={() => {
16          camera.current.stopRecording();
17          setRecording(false);
18        }}
19      />
20    </View>
21  </Camera>
22 );
```

Enfin, si un fichier de vidéo existe, on affiche sa prévisualisation avec :

```
1 // Si on a une video de preview, qu'on vient d'enregistrer
2 if (videoPreviewUrl) {
3   return (
4     <View style={{ padding: 50 }}>
5       <Video
6         source={videoPreviewUrl}
7         resizeMode="contain"
8         shouldPlay
9         isLooping
10        style={styles.videoPreview}
11      />
12       <Button
13         title="Fermer la prévisualisation"
14         onPress={() => setVideoPreviewUrl(null)}
15       />
16     </View>
17   );
18 }
```

```
15      />  
16    </View>  
17  );  
18 }
```