

La navigation avec React Native

Table des matières

I. Contexte	3
II. Le concept de route dans une application mobile	3
III. Exercice : Appliquez la notion	7
IV. La navigation entre les écrans et l'API de base d'un navigateur en pile	7
V. Exercice : Appliquez la notion	14
VI. Configurer les routes et les options d'un navigateur	14
VII. Exercice : Appliquez la notion	17
VIII. Essentiel	20
IX. Auto-évaluation	20
A. Exercice final	20
B. Exercice : Défi.....	22
Solutions des exercices	25

I. Contexte

Durée : 1 h

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

Pré-requis : Le principe de composants avec React, les props et le state

Contexte

Contrairement à son homologue ReactJS qui fonctionne dans un navigateur web, React Native n'a pas accès au système de navigation d'un navigateur web qui permet des actions comme « précédent » ou « suivant » ni de gérer le contenu d'une URL *out of the box*.

De plus, les besoins de navigation dans une application mobile sont généralement plus complexes et nécessitent des animations que l'on n'observe pas toujours sur les sites web.

Bien que l'on puisse utiliser la plupart des bibliothèques de navigation de ReactJS pour concevoir son application mobile, certaines bibliothèques essaient de fournir des abstractions permettant de concevoir des navigations pas trop éloignées de ce que l'on connaît dans le web, tout en mimant le rendu graphique d'applications natives. On n'est donc pas réellement dépaycé lorsque l'on utilise une application React Native, en tant qu'utilisateur final, à la place d'une application entièrement native.

II. Le concept de route dans une application mobile

Objectifs

- Le concept de route dans une application mobile
- Comprendre le fonctionnement de la navigation dans une application mobile *cross-platform*

Mise en situation

Dans le chapitre qui suit, nous allons étudier le fonctionnement de la navigation dans une application React Native en nous basant sur une bibliothèque entièrement composée de JavaScript (et donc utilisable avec Expo), nommée `react-navigation`. Grâce à `react-navigation`, il est possible de créer une structure de navigation simple, mais qui saura s'adapter à une complexité métier, même avancée. La bibliothèque est compatible sur toutes les plateformes supportées par React Native.

`react-navigation` s'inspire des navigations natives pour les transitions et le comportement, mais ne nécessite aucun code natif : c'est une bibliothèque autonome développée et maintenue par les équipes d'Expo.

Le routing et la navigation dans une application mobile

À la différence des applications web, les applications mobiles ne se basent pas sur une URL pour présenter une page. Par conséquent, la notion de route y est différente.

Dans une application React Native utilisant `react-navigation`, le routing se décompose entre plusieurs composants que l'on peut répartir en plusieurs catégories : le conteneur principal, les navigateurs, les écrans et les composants métier.

Ainsi, généralement au plus haut niveau de l'application, on retrouve un composant parent servant de conteneur pour un ou plusieurs composants navigateurs, qui contiennent un ou plusieurs composants écran, pouvant inclure un ou plusieurs composants métier.

Les navigateurs dans une application React Native

En anglais, on les nomme *navigators*, mais que sont-ils ?

Un navigateur est un composant React qui possède une certaine logique permettant de décrire le fonctionnement de sa navigation. Ces composants reçoivent des props qui leur permettent de définir la structure des écrans à afficher, ainsi qu'un ensemble de paramètres de configuration. Nous y reviendrons un peu plus tard : voyons dans un premier temps ce que `react-navigation` qualifie d'« écran ».

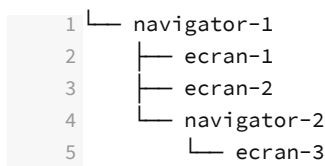
Définition Un écran dans React Navigation

Un écran n'est rien d'autre qu'un composant React Native qui reçoit des props de navigation spécifiques. De cette manière, il va pouvoir bénéficier d'éléments spécifiques à la page en cours et exécuter un certain nombre d'actions liées à la navigation, comme adapter le titre de l'écran courant ou changer l'état de navigation. Il affiche ensuite des composants graphiques et/ou métier pour que des données soient représentées.

Remarque

Notez qu'un navigateur est également un composant React, il peut donc être lui-même utilisé comme écran. Ce qui ouvre de nombreuses possibilités de navigation.

Une fois que nous avons la notion d'écran, nous allons pouvoir attacher nos écrans à notre/nos navigateurs. Hiérarchiquement, notre application va donc former une imbrication proche d'une structure en poupées russes.



Installation de react-navigation

Pour installer `react-navigation`, il faut ajouter les dépendances suivantes à notre projet :

- `@react-navigation/native` qui est le tronc commun
- `@react-navigation/stack` qui est le navigateur en pile

Pour cela, on utilise la commande `npm install @react-navigation/native @react-navigation/stack`.

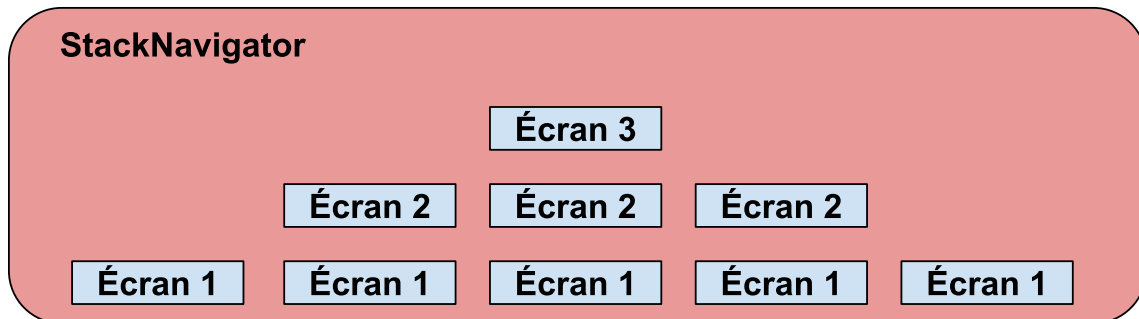
- Si on utilise Expo web, il faut rajouter les instructions d'import (que nous allons voir dans le chapitre suivant) et cliquer sur la notification en bas de l'écran, qui invite à installer les dépendances dans le `package.json`.
- Si on utilise une version d'Expo en local, il faudra également lancer cette commande : `expo install react-native-gesture-handler react-native-reanimated react-native-screens react-native-safe-area-context @react-native-community/masked-view`.

Notre premier exemple : le navigateur en pile (StackNavigator)

Dans ce premier exemple, nous allons représenter une structure simple d'enchaînement d'écrans dans une application React Native. Pour ce faire, nous allons utiliser l'un des trois navigateurs fournis par défaut dans `react-navigation` : le `StackNavigator` (ou « navigateur en pile »). Nous verrons d'autres navigateurs dans un autre module, une fois que nous aurons les concepts de base.

Le navigateur en pile s'approche du fonctionnement par défaut du navigateur web, à savoir que les écrans s'empilent les uns à la suite des autres. Dans l'exemple suivant, on peut voir une progression dans le temps (de gauche à droite) d'un empilement de composants « écran ».

Si l'on souhaite enlever l'écran 2, il faut d'abord enlever l'écran 3 qui est au-dessus de la pile. On parle de pile FILO (*First in, Last Out*, c'est-à-dire premier entré, dernier sorti). Les `StackNavigators` sont la colonne vertébrale d'une application React Native, nous les utiliserons très souvent afin d'afficher des modales ou de naviguer à travers les étapes d'un processus de paiement, par exemple.

**Exemple**

```
1 import * as React from 'react';
2 import { View, Text } from 'react-native';
3 import { NavigationContainer } from '@react-navigation/native';
4 import { createStackNavigator } from '@react-navigation/stack';
5
6 // Un composant écran
7 function HomeScreen() {
8   return (
9     <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
10       <Text>Écran d'accueil</Text>
11     </View>
12   );
13 }
14
15 // On crée un navigateur de pile, ici avec la configuration par défaut
16 const Stack = createStackNavigator();
17
18 function App() {
19   return (
20     <NavigationContainer>
21       <Stack.Navigator>
22         <Stack.Screen name="Home" component={HomeScreen} />
23       </Stack.Navigator>
24     </NavigationContainer>
25   );
26 }
27
28 export default App;
```

Home

Écran d'accueil

Dans cet exemple, on retrouve un composant conteneur de navigateurs, `<NavigationContainer>`, avec un seul composant navigateur, `<Stack.Navigator>` et un composant écran extrêmement simpliste, `<HomeScreen>`.

Remarque Petite particularité

Tant que des écrans se trouvent dans la pile, ils ne sont pas démontés au sens de React, ce qui signifie qu'ils consomment toujours de la mémoire, possèdent toujours leurs états et éventuels écouteurs d'événements qui ont été enregistrés. Leurs cycles de vie de `unmounting` n'ont pas été appelés !

Cela permet de revenir (lorsqu'on navigue en arrière) à notre écran précédent, qui contient tout ce que nous avons laissé, dans le même état (ce n'est pas une nouvelle instance de celui-ci).

Syntaxe **À retenir**

- React Native ne pouvant pas utiliser la navigation comme une application web en se basant sur l'URL, il existe des bibliothèques permettant de se rapprocher de ce comportement et de gérer automatiquement des animations de navigation.
- La principale bibliothèque pour cela est `react-navigation`, qui offre une grande liberté et propose notamment un navigateur en pile (`StackNavigator`) qui permet, comme son nom l'indique, d'empiler des écrans dans l'affichage.
- Les écrans ne sont que de simples composants React qui reçoivent des props leur permettant d'interagir avec la navigation et d'obtenir son état : de cette manière, on peut adapter notre expérience utilisateur grâce à des données dynamiques.

Complément

- <https://reactnavigation.org/docs/getting-started>

Exercice : Appliquez la notion

[solution n°1 p.27]

Exercice

`react-navigation` est en quelque sorte...

- ☐ Une API qui essaye de reproduire les comportements natifs de la navigation web en les transposant sur iOS et Android
- ☐ Une bibliothèque native qui offre d'excellentes performances, malgré le fait qu'elle ne s'utilise pas avec Expo facilement
- ☐ La seule bibliothèque disponible sur React Native pour gérer la navigation

Exercice

Le navigateur en pile (`StackNavigator`) se base sur une structure...

- ☐ FIFO (First In, First Out)
- ☐ FILO (First in, Last Out)
- ☐ LIFO (Last In, Last Out)
- ☐ FIRO (First In, Random Out)

Exercice

`react-navigation` est une bibliothèque...

- ☐ Développée par les équipes de Facebook uniquement
- ☐ Développée et maintenue par les équipes d'Expo
- ☐ Une bibliothèque forcément utilisée avec Expo
- ☐ Une bibliothèque autonome entièrement écrite en JavaScript

IV. La navigation entre les écrans et l'API de base d'un navigateur en pile

Objectifs

- Comprendre le paramétrage d'un navigateur dans une application React Native.
- Comprendre l'état de la navigation.

Mise en situation

Maintenant que nous avons vu le principe de base de la navigation dans une application mobile, intéressons-nous à la configuration des navigateurs. En effet, ces derniers sont très malléables et reçoivent un certain nombre de propriétés qui définissent leurs comportements et leur affichage. On peut par exemple choisir de désactiver certains éléments graphiques, comme l'élément du haut de la navigation ou personnaliser le label affiché en haut des éléments de navigation.

La configuration d'un navigateur en pile

Pour définir quels écrans composent notre navigateur, nous les intégrons en tant qu'enfants du composant navigateur : celui-ci utilise ensuite la props `children` pour récupérer les données de ses enfants.

Exemple

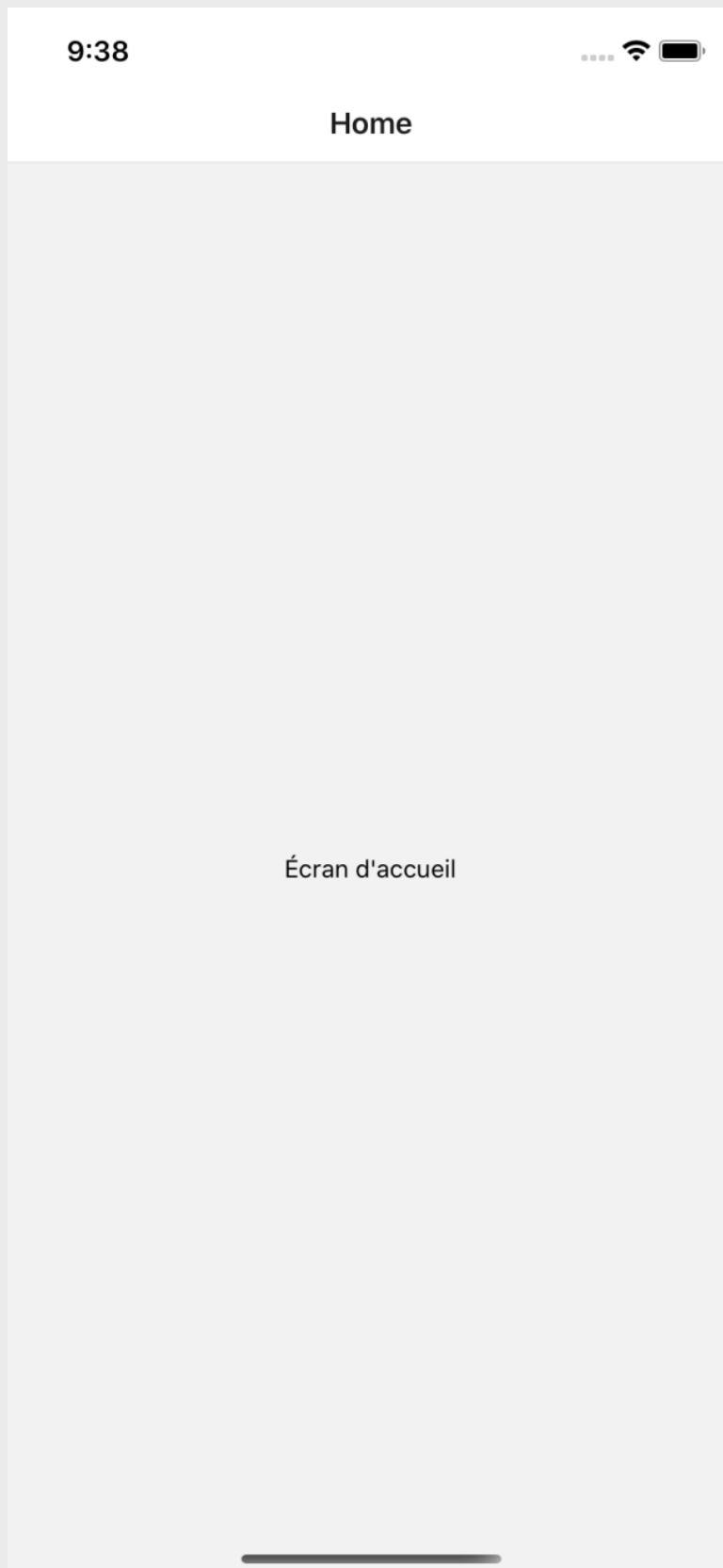
```

1 import * as React from 'react';
2 import { View, Text } from 'react-native';
3 import { NavigationContainer } from '@react-navigation/native';
4 import { createStackNavigator } from '@react-navigation/stack';
5
6 // Un composant qui agit en tant qu'« Écran » simple
7 function HomeScreen() {
8   return (
9     <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
10       <Text>Écran d'accueil</Text>
11     </View>
12   );
13 }
14
15 // Un composant qui agit en tant qu'« Écran » simple
16 function SecondaryScreen() {
17   return (
18     <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
19       <Text>Écran secondaire</Text>
20     </View>
21   );
22 }
23
24 // On crée un navigateur de pile, ici avec la configuration par défaut
25 const Stack = createStackNavigator();
26
27 // Notre application
28 function App() {
29   return (
30     <NavigationContainer>
31       <Stack.Navigator>
32         <Stack.Screen name="Home" component={HomeScreen} />
33         <Stack.Screen name="Secondary" component={SecondaryScreen} />
34       </Stack.Navigator>
35     </NavigationContainer>
36   );

```



```
37 }  
38  
39 export default App;
```



Comme on peut le constater, nous avons deux écrans qui sont de simples composants React affichant un texte les identifiant. Nous allons ensuite intégrer, assez haut dans l'arbre des composants de notre application, un composant `NavigationContainer`. Celui-ci gère notre arbre de navigation et contient l'état de navigation. Ce composant doit envelopper la structure de tous les navigateurs. Habituellement, ce composant se trouve à la racine de notre application, c'est-à-dire généralement dans le composant exporté depuis `App.js`.

Dans notre exemple, le navigateur contient deux écrans (identifiés par les clés `Home` et `Secondary`) qui affichent respectivement un composant `HomeScreen` et un composant `SecondaryScreen`. Par défaut, le premier écran affiché d'un composant `StackNavigator` sera le premier dans sa liste de navigation : ici, il s'agit donc de l'écran identifié par la clé `Home`.

Navigation entre les écrans

Chaque composant écran reçoit, par son parent `Screen` (dans l'exemple ci-dessus, `Stack.Screen`), des props lui permettant la navigation. On peut également utiliser la « Context API¹ » de React pour accéder à ces propriétés dans des composants imbriqués à l'intérieur des composants écrans. Pour cela, nous allons voir qu'il existe des hooks comme `useNavigation`, mis à disposition par `react-navigation`, et qui ont le même effet que les props de navigation passées au composant écran.

Parmi les props de navigation, voici la liste des props de navigation communes aux navigateurs².

Attardons-nous sur les plus importantes : `navigate` et `goBack`. Comme nous l'avons vu précédemment, pour naviguer dans une pile, on peut rajouter un élément sur la pile ou bien en enlever un. C'est exactement à cela que servent nos deux actions. Voici un exemple :

Exemple

```
1 import * as React from 'react';
2 import { View, Text, Button } from 'react-native';
3 import { NavigationContainer, useNavigation } from '@react-navigation/native';
4 import { createStackNavigator } from '@react-navigation/stack';
5
6 // Un composant qui agit en tant qu'« Écran » simple, il se base sur la props navigation
  injectée par son parent
7 function HomeScreen(props) {
8   return (
9     <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
10       <Text>Écran d'accueil</Text>
11       <Button
12         title="Aller à l'écran 2"
13         onPress={() => {
14           // Le paramètre passé est la valeur de la props name dans la définition du
15           // navigateur.
16           props.navigation.navigate('Secondary');
17         }}
18       />
19     </View>
20   );
21 }
22 // Un composant qui agit en tant qu'« Écran » simple, ici on utilise le hook useNavigation qui
  a le même comportement que la props navigation. Cependant on peut l'utiliser dans les enfants.
23 function SecondaryScreen() {
24   const navigation = useNavigation();
25
26   return (
```

¹ <https://fr.reactjs.org/docs/context.html>

² <https://reactnavigation.org/docs/navigation-actions/#navigate>

```

27   <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
28     <Text>Écran secondaire</Text>
29     <Button
30       title="Revenir en arrière"
31       onPress={() => {
32         // Permet de retirer un élément de la pile, pour le coup c'est l'écran lui-même.
33         navigation.goBack();
34       }}
35     />
36   </View>
37 );
38 }
39
40 // On crée un navigateur de pile, ici avec la configuration par défaut
41 const Stack = createStackNavigator();
42
43 // Notre application
44 function App() {
45   return (
46     <NavigationContainer>
47       <Stack.Navigator>
48         <Stack.Screen name="Home" component={HomeScreen} />
49         <Stack.Screen name="Secondary" component={SecondaryScreen} />
50       </Stack.Navigator>
51     </NavigationContainer>
52   );
53 }
54
55 export default App;

```

[cf. nav-8.mp4]

À noter que, dans l'exemple, le navigateur en pile (`StackNavigator`) affiche par défaut une barre en haut de l'écran permettant d'effectuer la même action que notre bouton, à savoir revenir en arrière dans la pile. Cette barre est personnalisable (nous le verrons dans le prochain chapitre).

Passer des paramètres à la navigation

Maintenant que nous avons une navigation de base fonctionnelle, voyons comment nous pouvons encore améliorer cela. Les changements dans l'état de navigation permettent de passer des paramètres qui seront exploitables par le composant cible : on pourrait par exemple passer les données liées à un document. C'est pratique, car on peut avoir des écrans génériques qu'on pourrait choisir d'empiler en changeant simplement les paramètres d'entrée. Pour cela, on peut utiliser `push` au lieu de `navigate` depuis l'API de navigation. En effet, `push` ajoutera un écran dans la pile quel que soit son contenu, c'est-à-dire qu'un même écran peut s'y retrouver plusieurs fois. À l'inverse, `navigate` vérifie si l'écran ciblé est présent ou non en pile et le fait passer en tête de pile le cas échéant.

Exemple

```

1 import * as React from 'react';
2 import { View, Text, Button } from 'react-native';
3 import { NavigationContainer, useNavigation, useRoute } from '@react-navigation/native';
4 import { createStackNavigator } from '@react-navigation/stack';
5
6 // Un composant qui agit en tant qu'« Écran » simple, il se base sur la props navigation
7 // injectée par son parent
8 function HomeScreen(props) {
9   return (

```

```

9      <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
10      <Text>Écran d'accueil</Text>
11      <Text>Données de navigation</Text>
12  // La fonction JSON.stringify va venir convertir notre objet en chaîne de caractère.
13  Transformer des objets en chaîne de caractère JSON
14      // permet notamment, sans entrer dans les détails, de stocker des données sur le
15      serveur, les transférer du serveur au client,
16      // du client au serveur, etc.
17      // Ici, l'objet props.route.params est donc converti en chaîne de caractère afin
18  d'être stocké et réutilisé par la suite.
19      <Text>{JSON.stringify(props.route.params)}</Text>
20      <Button
21      title="Empiler un nouveau HomeScreen avec Params A"
22      onPress={() => {
23          // Le paramètre passé est la valeur de la props name dans la définition du composant
24          navigateur.
25          props.navigation.push('Home', { someId: 25, anotherParam: "Hello"});
26      }}
27      />
28      <Button
29      title="Empiler un nouveau HomeScreen avec Params B"
30      onPress={() => {
31          // Le paramètre passé est la valeur de la props name dans la définition du composant
32          navigateur.
33          props.navigation.navigate('Home', { someId: 30, anotherParam: "World"});
34      }}
35      />
36      <Button
37      title="Empiler un nouveau SecondaryScreen avec Params C"
38      onPress={() => {
39          // Le paramètre passé est la valeur de la props name dans la définition du
40          navigateur.
41          props.navigation.navigate('Secondary', { someId: 12, anotherParam: "Coucou"});
42      }}
43      />
44  </View>
45  );
46  }
47  // Un composant qui agit en tant qu'« Écran » simple, ici on utilise le hook useNavigation qui
48  a le même comportement que la props navigation. Cependant on peut l'utiliser dans les enfants.
49  function SecondaryScreen() {
50      const navigation = useNavigation();
51      const route = useRoute();
52
53      return (
54          <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
55          <Text>Écran secondaire</Text>
56          <Text>Données de navigation</Text>
57          <Text>{JSON.stringify(route.params)}</Text>
58          <Button
59          title="Revenir en arrière"
60          onPress={() => {
61              // Permet de retirer un élément de la pile, pour le coup c'est l'écran lui-même.
62              navigation.goBack();
63          }}
64          />
65          </View>
66      );
67  }

```

```
62
63 // On créé un navigateur de pile, ici avec la configuration par défaut
64 const Stack = createStackNavigator();
65
66 // Notre application
67 function App() {
68   return (
69     <NavigationContainer>
70       <Stack.Navigator>
71         <Stack.Screen name="Home" component={HomeScreen} />
72         <Stack.Screen name="Secondary" component={SecondaryScreen} />
73       </Stack.Navigator>
74     </NavigationContainer>
75   );
76 }
77
78 export default App;
```

[cf. nav-9.mp4]

Dans cet exemple, si l'on navigue avec le premier bouton, on voit un empilement s'effectuer. Par contre, si on utilise le deuxième bouton, le navigateur va chercher à remplacer le paramètre de l'instance déjà montée de cet écran. On remarque que l'on peut récupérer les informations de navigation de deux manières différentes : consommer les paramètres passés en utilisant les props, ou utiliser des hooks comme `useNavigation` ou `useRoute`.

Syntaxe **À retenir**

- On utilise la navigation par programmation pour naviguer entre les écrans d'une application React Native. Le composant de navigation en pile permet de s'approcher d'une navigation d'un navigateur web, : il propose plusieurs méthodes qui permettent d'interagir avec la navigation.
- On peut par exemple utiliser `navigate` ou `push` selon si l'on souhaite ajouter un écran à la pile ou retourner vers l'écran précédent le plus proche dans la pile actuelle.
- Pour pouvoir factoriser son code, on peut également utiliser des paramètres de navigation afin d'utiliser des écrans génériques qui attendent certains paramètres pour remplir leur fonction. On pourrait imaginer passer l'identifiant d'un document à récupérer via une API web, par exemple.

Complément

- <https://reactnavigation.org/docs/navigating>
- <https://reactnavigation.org/docs/stack-actions/#push>
- <https://reactnavigation.org/docs/params>

V. Exercice : Appliquez la notion

Question

[solution n°2 p.27]

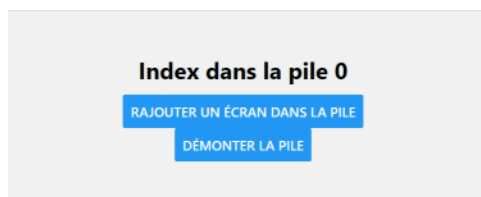
Créons une application qui va ajouter des éléments « écran » dans la pile, à l'infini.

L'écran est un composant `HomeScreen` qui affiche un compteur qui commence à zéro et qui indique la position de l'écran courant dans la pile du navigateur.

`HomeScreen` affiche également deux boutons :

- Le premier ajoute une autre instance de `HomeScreen` dans la pile courante,
- Le second démonte l'intégralité de la pile d'un coup. Pour cela, on utilisera l'instruction `navigation.popToTop()` qui permet ce comportement.

Le rendu escompté doit ressembler à la capture d'écran ci-dessous :



VI. Configurer les routes et les options d'un navigateur

Objectifs

- Personnaliser la configuration d'un routeur en pile
- Se servir des props et paramètres pour la configuration

Mise en situation

Les composants navigateurs de `react-navigation` permettent de naviguer dans une application mobile. Pour cela, ils présentent divers états et fonctions propres à la navigation. Nous allons voir qu'ils offrent aussi la possibilité de configurer le rendu des éléments graphiques qui les composent, en utilisant directement des propriétés qu'ils exposent.

Les propriétés du navigateur en pile et de ses écrans

On distingue deux types de propriétés : celles du navigateur et celles des écrans.

Pour le navigateur, on peut par exemple définir des propriétés par défaut pour les écrans ou son mode d'animation. Pour les écrans, on peut personnaliser le titre, la couleur ou encore les boutons de l'entête. Pour ce faire, c'est très simple : il suffit de passer quelques propriétés lors de la déclaration. Par exemple, ci-dessous, on modifie le titre de l'écran d'accueil de deux manières :

Exemple

```
1 import * as React from "react";
2 import { View, Text, Button } from "react-native";
3 import {
4   NavigationContainer,
5   useNavigation,
6   useRoute,
7 } from "@react-navigation/native";
8 import { createStackNavigator } from "@react-navigation/stack";
```

```

9
10 // Un composant qui agit en tant qu'« Écran » simple, il se base sur la props navigation
    injectée par son parent
11 function HomeScreen(props) {
12   return (
13     <View style={{ flex: 1, alignItems: "center", justifyContent: "center" }}>
14       <Text>Écran d'accueil</Text>
15       <Text>Données de navigation</Text>
16       <Text>{JSON.stringify(props.route.params)}</Text>
17       <Button
18         title="Empiler un nouveau HomeScreen avec Params A"
19         onPress={() => {
20           // Le paramètre passé est la valeur de la props name dans la définition du
navigateur.
21           props.navigation.push("Home", { someId: 25, anotherParam: "Hello" });
22         }}
23       />
24       <Button
25         title="Empiler un nouveau HomeScreen avec Params B"
26         onPress={() => {
27           // Le paramètre passé est la valeur de la props name dans la définition du
navigateur.
28           props.navigation.navigate("Home", {
29             someId: 30,
30             anotherParam: "World",
31           });
32         }}
33       />
34       <Button
35         title="Empiler un nouveau SecondaryScreen avec Params C"
36         onPress={() => {
37           // Le paramètre passé est la valeur de la props name dans la définition du
navigateur.
38           props.navigation.navigate("Secondary", {
39             someId: 12,
40             anotherParam: "Coucou",
41           });
42         }}
43       />
44     </View>
45   );
46 }
47
48 // Un composant qui agit en tant qu'« Écran » simple, ici on utilise le hook useNavigation qui
    a le même comportement que la props navigation. Cependant on peut l'utiliser dans les enfants.
49 function SecondaryScreen() {
50   const navigation = useNavigation();
51   const route = useRoute();
52
53   return (
54     <View style={{ flex: 1, alignItems: "center", justifyContent: "center" }}>
55       <Text>Écran secondaire</Text>
56       <Text>Données de navigation</Text>
57       <Text>{JSON.stringify(route.params)}</Text>
58       <Button
59         title="Revenir en arrière"
60         onPress={() => {
61           // Permet de retirer un élément de la pile, pour le coup c'est l'écran lui-même.
62           navigation.goBack();

```

```

63     }}
64     />
65   </View>
66 );
67 }
68
69 // On créé un navigateur de pile, ici avec la configuration par défaut
70 const Stack = createStackNavigator();
71
72 // Notre application
73 function App() {
74   return (
75     <NavigationContainer>
76       <Stack.Navigator
77         screenOptions={{
78           headerStyle: {
79             backgroundColor: "#f4511e",
80           },
81           headerTintColor: "#fff",
82           headerTitleStyle: {
83             fontWeight: "bold",
84           },
85         }}
86       >
87         <Stack.Screen
88           name="Home"
89           component={HomeScreen}
90           options={{ title: "Titre de l'écran d'accueil" }}
91         />
92         <Stack.Screen
93           name="Secondary"
94           component={SecondaryScreen}
95           options={(props) => ({
96             title: "Données du TODO n°" + props.route.params.someId,
97           })}
98         />
99       </Stack.Navigator>
100     </NavigationContainer>
101   );
102 }
103
104 export default App;

```

[cf. nav-10.mp4]

Dans le premier cas, on passe une valeur brute qui n'est pas sujette à changements. Dans le deuxième, on passe une fonction qui renvoie un objet avec un titre dynamique. Cette fonction reçoit en argument les props (dont les props de route et de navigation) que le composant de l'écran va également recevoir. De cette manière, on peut personnaliser l'affichage en fonction des données reçues.

On peut aussi définir des propriétés communes à tous les écrans grâce à `screenOptions`, sur le navigateur cette fois-ci. Chaque navigateur ayant ses propres options, il est pertinent de consulter la documentation (en annexe de ce chapitre) pour connaître la liste exhaustive des options disponibles.

Syntaxe **À retenir**

- Il est possible de personnaliser la configuration d'un navigateur ou de ses écrans. On peut également définir un certain nombre de propriétés par défaut pour ces écrans.
- Grâce à cela, il est possible de surcharger des comportements de base, tels que la couleur du titre dans la barre d'entête de navigation ou bien l'action des boutons dans cette dernière.
- La liste des options est propre au type de navigateur et est disponible sur la documentation de `react-navigation`.

Complément

- <https://reactnavigation.org/docs/stack-navigator#options>
- <https://reactnavigation.org/docs/headers>

VII. Exercice : Appliquez la notion

Question

[solution n°3 p.29]

L'écran d'accueil affiche un lien avec un bouton « **Consulter les résultats sportifs** », qui ouvre un écran affichant une liste de cartes avec des résultats sportifs en utilisant le composant `FlatList` de React Native¹.

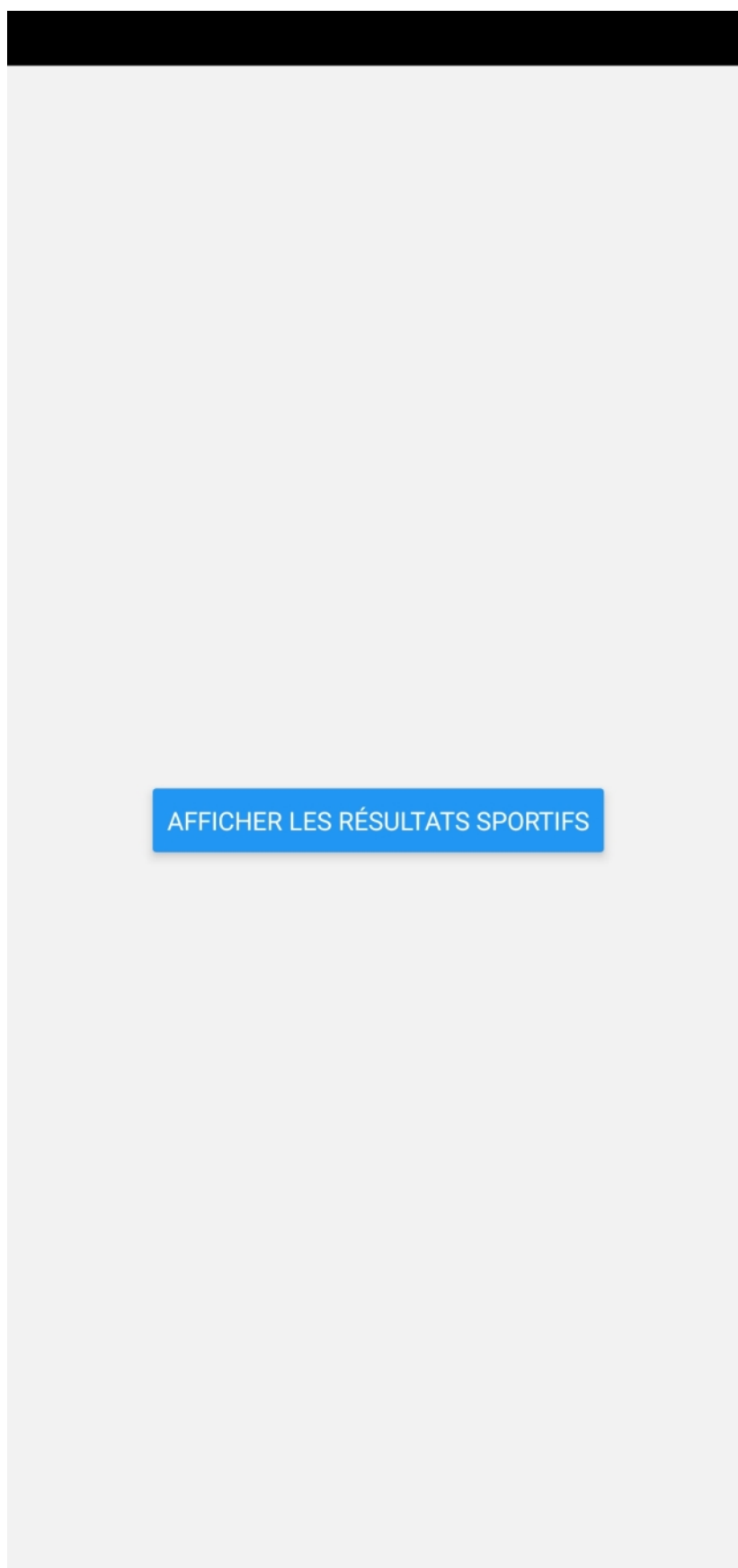
L'écran d'accueil ne doit pas afficher la barre d'entête, qui est désactivable depuis les options pour cet écran spécifiquement.

Le nom de l'écran qui affiche les résultats sera « *Les résultats sportifs* ».

La carte qui affiche les résultats d'un match est un composant réutilisable qui reçoit ce qu'il doit afficher en paramètres.

¹ <https://reactnative.dev/docs/flatlist.html>

Le rendu escompté doit s'approcher des deux captures d'écran ci-dessous :



← Les résultats sportifs

France Allemagne

2

3

Italie Canada

1

3

Japon Turquie

1

0

VIII. Essentiel

Étant donné que la navigation d'une application mobile ne peut pas se baser sur une URL comme c'est le cas avec un navigateur web, il faut adapter notre façon de penser. Pour cela, il existe des bibliothèques qui permettent d'afficher des composants React Native de manière conditionnelle en gérant un routeur personnalisé.

La bibliothèque la plus populaire est `react-navigation` qui permet une approche déclarative de la structure de notre navigation. Notamment via un routeur en pile (`StackNavigator`), qui permet d'empiler ou dépiler des écrans à afficher dans l'application. Cette bibliothèque fournit un grand nombre de fonctionnalités, comme des animations permettant de se rapprocher de la navigation d'une vraie application native. On peut personnaliser et étendre de nombreuses fonctionnalités en passant des paramètres de configuration au navigateur et aux composants écrans.

Enfin, on peut créer des composants génériques, qui reçoivent en paramètres des valeurs qui vont permettre d'alimenter leur logique métier. On pourrait par exemple imaginer recevoir l'identifiant d'un utilisateur à afficher, ce qui permettrait au composant écran de faire une requête HTTP pour récupérer le profil dudit utilisateur.

IX. Auto-évaluation

A. Exercice final

Exercice 1

[solution n°4 p.32]

Exercice

Comment se nomme la bibliothèque de navigation principalement utilisée dans ce cours ?

- ☐ `react-native-navigation`
- ☐ `react-navigation`
- ☐ `react-navigator`
- ☐ `navigation-stack`

Exercice

La navigation dans React Native requiert d'écrire du code natif.

- ☐ Vrai
- ☐ Faux

Exercice

Quelles phrases sont vraies à propos d'un navigateur dans `react-navigation` ?

- ☐ Un navigateur est un composant natif bas niveau mis à disposition dans React Native par la bibliothèque
- ☐ On ne peut avoir qu'un seul navigateur dans toute une application React Native
- ☐ C'est un composant React avec une logique métier particulière pour gérer un état de navigation et des animations

Exercice

Quelles phrases sont vraies à propos d'un écran dans `react-navigation` ?

- ☐ Un écran est un composant React qui reçoit des propriétés de navigation de son parent navigateur
- ☐ Un écran déclaré dans un navigateur en pile ne peut apparaître qu'avec une seule instance dans cette pile
- ☐ On peut personnaliser un écran avec des paramètres de configuration pour influencer sur son design et sa logique

Exercice

Pour passer des paramètres `param1` et `param2` via ma navigation, j'utilise la syntaxe...

- ☐ `navigation.navigate("ScreenName", { param1: "coucou", param2: 2 })`
- ☐ `navigation.navigate("ScreenName", { parameters: { param1: "coucou", param2: 2 } })`
- ☐ `navigation.push("ScreenName", { parameters: { param1: "coucou", param2: 2 } })`
- ☐ `navigation.push("ScreenName", { param1: "coucou", param2: 2 })`

Exercice

Lorsque j'ajoute un écran à la pile, qu'advient-il de ceux plus bas (dans la pile) ?

- ☐ Il restent montés, mais simplement non affichés à l'écran
- ☐ Ils sont démontés et retirés de l'affichage
- ☐ Ils se mettent en veille, ce qui fait que l'intégralité de leur code est inactif tant qu'ils n'ont pas le focus

Exercice

Quelles signatures sont correctes pour la props `options` du composant `Stack.Screen` (en partant du principe qu'on a une variable `Stack` qui est un `StackNavigator`) ?

- ☐ `options={{ title: "Titre de l'écran" }}`
- ☐ `options={(props) => ({ title: 'Article n°' + props.route.params.articleId })}`
- ☐ `options={{ screen: { title: "Titre de l'écran" } }}`

Exercice

`NavigationContainer` est un composant obligatoire.

- ☐ Vrai
- ☐ Faux

Exercice

Quelle est la différence entre les API `push` et `navigate` d'un `StackNavigator` ?

- ☐ Ce ne sont que des alias, elles font la même chose
- ☐ L'API `push` rajoute un élément dans la pile quoi qu'il arrive, alors que `navigate` cherchera à naviguer vers l'élément correspondant le plus proche dans la pile, s'il en existe un, et ne fera rien le cas échéant
- ☐ L'API `push` rajoute un élément dans la pile quoi qu'il arrive, alors que `navigate` cherchera à naviguer vers l'élément correspondant le plus proche dans la pile, s'il en existe un, et aura le même effet que `push` le cas échéant

Exercice

Quelles phrases sont vraies quant à l'utilisation de `props.navigation` ou du hook `useNavigation` de `react-navigation` ?

- ☐ Ils remplissent la même fonction
- ☐ On ne peut utiliser `useNavigation` que dans les enfants d'un écran, mais pas dans un écran lui-même
- ☐ `useNavigation` permet d'utiliser le contexte de navigation dans n'importe quel composant qui se situe N niveaux en dessous d'un `NavigationContainer`
- ☐ `props.navigation` est à préférer dans un écran

B. Exercice : Défi

Après avoir vu toutes les fonctionnalités d'un navigateur en pile, utilisons les connaissances acquises pour réaliser une mini application qui affiche une liste de TODO, récupérés via le réseau.



Question

[solution n°5 p.35]

L'application est composée d'un écran d'accueil qui invite à choisir quel TODO on souhaite afficher.

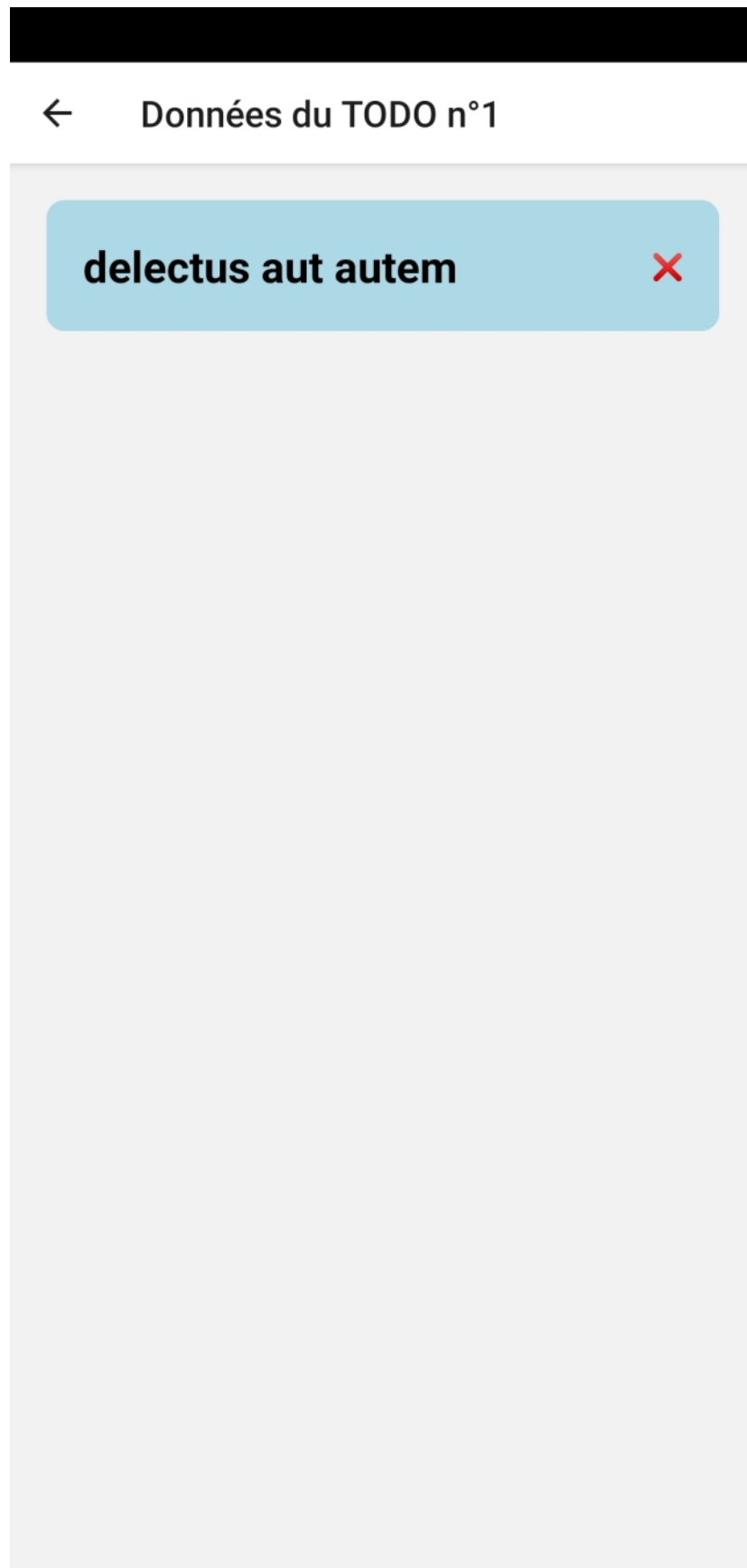
On propose d'afficher le TODO avec l'ID 1, 2 ou 3 : pour cela, l'écran d'accueil affiche 3 boutons permettant de se rendre sur ces écrans via la navigation.

L'écran qui affiche un TODO est générique et prend en paramètre l'ID du TODO à récupérer via l'API <https://jsonplaceholder.typicode.com/todos/<todoId>>.

Il faudra remplacer `<todoId>` par l'ID du TODO en question, récupérer la donnée et l'afficher dans la page sous la forme d'une carte qui affiche le titre et le statut de complétion, avec  s'il est complété, ou  sinon.

Le rendu final escompté doit s'approcher des deux captures d'écran ci-dessous :





Indice :

L'utilisation de l'API Fetch permettant d'interroger une API distante se fait de la manière suivante :

```
1 fetch('https://jsonplaceholder.typicode.com/todos/1')
2   .then(response => response.json())
3   .then(json => console.log(json))
4
```

L'API `https://jsonplaceholder.typicode.com/todos/1` renvoie des données sous la forme :

```
1 // Renvoi
2 const result = {
3   "userId": 1,
4   "id": 1,
5   "title": "delectus aut autem",
6   "completed": false
7 }
```

Solutions des exercices

Exercice p. 7 Solution n°1**Exercice**

react-navigation est en quelque sorte...

- ☒ Une API qui essaye de reproduire les comportements natifs de la navigation web en les transposant sur iOS et Android
- ☐ Une librairie native qui offre d'excellentes performances, malgré le fait qu'elle ne s'utilise pas avec Expo facilement
- ☐ La seule librairie disponible sur React Native pour gérer la navigation
- ☒ Bien qu'elle soit vivement recommandée pour développer une navigation dans React Native, elle n'est pas la seule. Ses performances sont très bonnes et elle n'a pas besoin de code natif.

Exercice

Le navigateur en pile (`StackNavigator`) se base sur une structure...

- ☐ FIFO (First In, First Out)
- ☒ FILO (First in, Last Out)
- ☐ LILO (Last In, Last Out)
- ☐ FIRO (First In, Random Out)
- ☒ Les derniers écrans empilés sont retirés en premiers, c'est donc une pile FILO.

Exercice

react-navigation est une librairie...

- ☐ Développée par les équipes de Facebook uniquement
- ☒ Développée et maintenue par les équipes d'Expo
- ☐ Une librairie forcément utilisée avec Expo
- ☒ Une librairie autonome entièrement écrite en JavaScript
- ☒ La librairie possède aussi quelques contributeurs des équipes Facebook, mais c'est un projet d'Expo, originellement.

p. 14 Solution n°2

```
1 import * as React from 'react';
2 import {
3   View,
4   Text,
5   Button,
6   StyleSheet,
7 } from 'react-native';
8 import {
9   NavigationContainer,
10  useRoute,
```

```

11 } from '@react-navigation/native';
12 import { createStackNavigator } from '@react-navigation/stack';
13
14
15 // Notre composant personnalisé qui gère un compteur entre les écrans
16 function HomeScreen(props) {
17   const route = useRoute()
18   // Cette ligne ci-dessous va définir le currentIndex..
19   // Si le type (typeof) de la propriété route.params est différent de "undefined",
20   (C'est-à-dire : Si le type de la propriété est défini) alors
21   // on définit currentIndex par la propriété route.params.index. Sinon, on définit le
22   currentIndex sur 0.
23   const currentIndex = typeof route.params !== "undefined" ? route.params.index : 0
24
25   return (
26     <View style={homeStyles.container}>
27       <Text style={homeStyles.title}>Index dans la pile {currentIndex}</Text>
28       <Button title="Rajouter un écran dans la pile" onPress={() =>
29         props.navigation.push("Home", { index: currentIndex + 1})}/>
30       <Button title="Démonter la pile" onPress={() => props.navigation.reset({ index: 0,
31         routes: [{ name: 'Home' }]}> />
32     </View>
33   );
34 }
35
36 // Les styles liés au composant HomeScreen
37 const homeStyles = StyleSheet.create({
38   container: { flex: 1, alignItems: 'center', justifyContent: 'center' },
39   title: { fontSize: 24, fontWeight: 'bold', marginBottom: 10 },
40 });
41
42 // On crée un navigateur de pile, ici avec la configuration par défaut
43 const Stack = createStackNavigator();
44
45 // Notre application
46 function App() {
47   return (
48     <NavigationContainer>
49       <Stack.Navigator>
50         <Stack.Screen
51           name="Home"
52           component={HomeScreen}
53           options={{ headerShown: false }}
54         />
55       </Stack.Navigator>
56     </NavigationContainer>
57   );
58 }
59
60 export default App;

```

Pour réaliser cet exercice, un composant conteneur, `<NavigationContainer>`, avec un composant de navigation de type `StackNavigator`, `<Stack.Navigator>`, est créé au niveau le plus haut de notre application. Le composant de navigation, quant à lui, est constitué du composant `<HomeScreen>` dont la référence est, ici, nommée **Home** :

```
1 const Stack = createStackNavigator();
2
3 <NavigationContainer>
4   <Stack.Navigator>
5     <Stack.Screen
6       name="Home"
7       component={HomeScreen}
8       options={{ headerShown: false }}
9     />
10  </Stack.Navigator>
11 </NavigationContainer>
```

Le composant `<HomeScreen>` permet deux actions sur la navigation en utilisant le paramètre `navigation` passé en props de manière automatique par le composant de navigation. La première de ces actions ajoute un écran à la pile de navigation avec `props.navigation.push()`, tandis que la seconde permet de vider la pile, puis d'y ajouter l'écran **Home** avec `props.navigation.reset()`.

De plus, ce composant utilise le hook `useRoute` pour récupérer les paramètres passés lors de la navigation sous la forme d'un objet `{ index : currentIndex }`. Ceux-ci se trouvent dans la propriété `params` fournie par `useRoute`.

```
1 function HomeScreen(props) {
2   const route = useRoute()
3   const currentIndex = typeof route.params !== "undefined" ? route.params.index : 0
4
5   return (
6     <View style={homeStyles.container}>
7       <Text style={homeStyles.title}>Index dans la pile {currentIndex}</Text>
8       <Button title="Rajouter un écran dans la pile" onPress={() =>
9         props.navigation.push("Home", { index: currentIndex + 1})}/>
10      <Button title="Démonter la pile" onPress={() => props.navigation.reset({ index: 0,
11        routes: [{ name: 'Home' }]} )}/>
12    </View>
13  );
14 }
```

p. 17 Solution n°3

```
1 import * as React from 'react';
2 import {
3   ActivityIndicator,
4   View,
5   Text,
6   Button,
7   StyleSheet,
8   FlatList,
9 } from 'react-native';
10 import {
11   NavigationContainer,
12   useNavigation,
13 } from '@react-navigation/native';
14 import { createStackNavigator } from '@react-navigation/stack';
15
16 // Un composant qui agit en tant qu'« Écran » simple, il se base sur le hook useNavigation
```

```

17 function HomeScreen() {
18   const navigation = useNavigation();
19
20   return (
21     <View style={homeStyles.container}>
22       <Button
23         title="Afficher les résultats sportifs"
24         onPress={() => navigation.push('SportResults')}
25       />
26     </View>
27   );
28 }
29
30 // Les styles liés au composant HomeScreen
31 const homeStyles = StyleSheet.create({
32   container: { flex: 1, alignItems: 'center', justifyContent: 'center' },
33 });
34
35 // Notre composant de résultat de match sous forme d'une carte
36 function MatchResultCard(props) {
37   return (
38     <View style={cardStyles.container}>
39       <View style={cardStyles.teamWrapper}>
40         <Text style={cardStyles.teamTitle}>{props.teamA}</Text>
41         <Text>{props.teamAScore}</Text>
42       </View>
43
44       <View style={cardStyles.teamWrapper}>
45         <Text style={cardStyles.teamTitle}>{props.teamB}</Text>
46         <Text>{props.teamBScore}</Text>
47       </View>
48     </View>
49   );
50 }
51
52 const cardStyles = StyleSheet.create({
53   container: {
54     flexDirection: 'row',
55     justifyContent: 'center',
56     backgroundColor: "lightblue",
57     margin: 10,
58     padding: 10,
59     borderRadius: 5
60   },
61   teamWrapper: {
62     alignItems: "center"
63   },
64   teamTitle: {
65     fontSize: 22,
66     fontWeight: 'bold',
67     marginHorizontal: 10,
68     marginBottom: 10
69   },
70 });
71
72 // Un composant qui fait office d'écran pour afficher une liste de cartes.
73 function SportResultsScreen() {
74   return (

```

```

75   <FlatList
76     data=[
77       { teamA: 'France', teamAScore: 2, teamB: 'Allemagne', teamBScore: 3 },
78       { teamA: 'Italie', teamAScore: 1, teamB: 'Canada', teamBScore: 3 },
79       { teamA: 'Japon', teamAScore: 1, teamB: 'Turquie', teamBScore: 0 },
80     ]
81     renderItem={(data) => <MatchResultCard {...data.item} />}
82     keyExtractor={(item) => item.teamA + item.teamB}
83   />
84 );
85 }
86
87 // On crée un navigateur de pile, ici avec la configuration par défaut
88 const Stack = createStackNavigator();
89
90 // Notre application
91 function App() {
92   return (
93     <NavigationContainer>
94       <Stack.Navigator>
95         <Stack.Screen
96           name="Home"
97           component={HomeScreen}
98           options={{ headerShown: false }}
99         />
100        <Stack.Screen
101          name="SportResults"
102          component={SportResultsScreen}
103          options={{ title: 'Les résultats sportifs' }}
104        />
105      </Stack.Navigator>
106    </NavigationContainer>
107  );
108 }
109
110 export default App

```

Pour la réalisation de cette application, on sait que deux écrans sont nécessaires : l'un d'entre eux constituera l'écran d'accueil, à partir duquel on pourra naviguer vers le second, qui affiche une liste de résultats sportifs, eux-mêmes modélisés dans un composant spécifique.

On peut d'ores et déjà préparer le composant de liste et le composant de résultat, qui se contenteront d'afficher des données :

```

1 function MatchResultCard(props) {
2   return (
3     <View style={cardStyles.container}>
4       <View style={cardStyles.teamWrapper}>
5         <Text style={cardStyles.teamTitle}>{props.teamA}</Text>
6         <Text>{props.teamAScore}</Text>
7       </View>
8
9       <View style={cardStyles.teamWrapper}>
10        <Text style={cardStyles.teamTitle}>{props.teamB}</Text>
11        <Text>{props.teamBScore}</Text>
12      </View>
13    </View>
14  );
15 }

```

```

16
17 function SportResultsScreen() {
18   return (
19     <FlatList
20       data=[
21         { teamA: 'France', teamAScore: 2, teamB: 'Allemagne', teamBScore: 3 },
22         { teamA: 'Italie', teamAScore: 1, teamB: 'Canada', teamBScore: 3 },
23         { teamA: 'Japon', teamAScore: 1, teamB: 'Turquie', teamBScore: 0 },
24       ]
25       renderItem={(data) => <MatchResultCard {...data.item} />}
26       keyExtractor={(item) => item.teamA + item.teamB}
27     />
28   );
29 }

```

On peut ensuite préparer la navigation en positionnant dans un conteneur un composant navigateur que l'on paramètre avec deux écrans : l'écran Home que l'on associe au composant <HomeScreen> à venir et SportResults associé au composant <SportResultsScreens> précédemment créé. Pour respecter la demande de ne pas afficher de barre d'entête, on passe l'option `headerShown: false` à l'écran Home.

```

1 const Stack = createStackNavigator();
2
3 // Notre application
4 function App() {
5   return (
6     <NavigationContainer>
7       <Stack.Navigator>
8         <Stack.Screen
9           name="Home"
10          component={HomeScreen}
11          options={{ headerShown: false }}
12        />
13        <Stack.Screen
14          name="SportResults"
15          component={SportResultsScreen}
16          options={{ title: 'Les résultats sportifs' }}
17        />
18      </Stack.Navigator>
19    </NavigationContainer>
20  );
21 }

```

Enfin, on peut créer le composant <HomeScreen> qui permettra de naviguer vers l'écran SportResults :

```


1 function HomeScreen() {
2   const navigation = useNavigation();
3
4   return (
5     <View style={homeStyles.container}>
6       <Button
7         title="Afficher les résultats sportifs"
8         onPress={() => navigation.push('SportResults')}
9       />
10    </View>
11  );
12 }

```

Exercice p. 20 Solution n°4


Exercice

Comment se nomme la librairie de navigation principalement utilisée dans ce cours ?

- ☐ react-native-navigation
 - ☒ react-navigation
 - ☐ react-navigator
 - ☐ navigation-stack
-  La bonne réponse est `react-navigation`.


Exercice

La navigation dans React Native requiert d'écrire du code natif.

- ☐ Vrai
 - ☒ Faux
-  Bien que s'inspirant des idées et animations disponibles sur les navigations natives, `react-navigation` est entièrement réalisée en JavaScript et ne nécessite pas de code natif spécifique pour s'utiliser. Certaines librairies, cependant, pourraient nécessiter du code natif, mais elles ne seront pas abordées dans ce cours.


Exercice

Quelles phrases sont vraies à propos d'un navigateur dans `react-navigation` ?

- ☐ Un navigateur est un composant natif bas niveau mis à disposition dans React Native par la librairie
 - ☐ On ne peut avoir qu'un seul navigateur dans toute une application React Native
 - ☒ C'est un composant React avec une logique métier particulière pour gérer un état de navigation et des animations
-  Un navigateur est un composant React proposant diverses fonctionnalités de navigation, ce qui en fait un composant de haut niveau, car il fait une abstraction des instructions de bas niveau. Ce n'est pas un composant natif, car `react-navigation` est entièrement codé en JavaScript. On peut également avoir plusieurs navigateurs dans une application React Native.


Exercice

Quelles phrases sont vraies à propos d'un écran dans `react-navigation` ?

- ☒ Un écran est un composant React qui reçoit des propriétés de navigation de son parent navigateur
 - ☐ Un écran déclaré dans un navigateur en pile ne peut apparaître qu'avec une seule instance dans cette pile
 - ☒ On peut personnaliser un écran avec des paramètres de configuration pour influencer sur son design et sa logique
-  Un écran est un composant recevant une props `navigation` permettant d'accéder à toutes les propriétés provenant du navigateur. Il peut être personnalisé directement depuis les options de configuration offertes par `react-navigation`. Dans le cas d'un `StackNavigator`, il est possible que plusieurs instances d'un même écran soient présentes dans la pile, grâce à l'utilisation de l'instruction `navigation.push`.


Exercice

Pour passer des paramètres `param1` et `param2` via ma navigation, j'utilise la syntaxe...

- ☒ `navigation.navigate("ScreenName", { param1: "coucou", param2: 2 })`
 - ☐ `navigation.navigate("ScreenName", { parameters: { param1: "coucou", param2: 2 } })`
 - ☐ `navigation.push("ScreenName", { parameters: { param1: "coucou", param2: 2 } })`
 - ☒ `navigation.push("ScreenName", { param1: "coucou", param2: 2 })`
-  L'objet en deuxième argument contient directement les paramètres et il n'y a pas besoin de spécifier un objet avec une clé `parameters`. On peut ensuite, selon le comportement désiré, utiliser `navigate` ou `push`.


Exercice

Lorsque j'ajoute un écran à la pile, qu'advient-il de ceux plus bas (dans la pile) ?

- ☒ Il restent montés, mais simplement non affichés à l'écran
 - ☐ Ils sont démontés et retirés de l'affichage
 - ☐ Ils se mettent en veille, ce qui fait que l'intégralité de leur code est inactif tant qu'ils n'ont pas le focus
-  Les éléments restent montés et ne se mettent pas en veille. Ils n'ont certes pas le focus, mais, si par exemple nous ouvrons une websocket, le code de traitement des messages restera actif et pourra par exemple déclencher des notifications visuelles.


Exercice

Quelles signatures sont correctes pour la `props options` du composant `Stack.Screen` (en partant du principe qu'on a une variable `Stack` qui est un `StackNavigator`) ?

- ☒ `options={{ title: "Titre de l'écran" }}`
 - ☒ `options={(props) => ({ title: 'Article n°' + props.route.params.articleId })}`
 - ☐ `options={{ screen: { title: "Titre de l'écran" } }}`
-  On peut passer une fonction pour permettre d'utiliser une valeur dynamique ou directement une valeur en dur. Le paramètre `screen` n'existe pas comme tel.

Exercice

`NavigationContainer` est un composant obligatoire.

- ☒ Vrai
 - ☐ Faux
-  C'est un composant obligatoire qui va contenir, comme son nom l'indique, l'état de navigation et qui va transmettre à ses enfants la valeur de cet état.

Exercice

Quelle est la différence entre les API `push` et `navigate` d'un `StackNavigator` ?

- ☐ Ce ne sont que des alias, elles font la même chose
- ☐ L'API `push` rajoute un élément dans la pile quoi qu'il arrive, alors que `navigate` cherchera à naviguer vers l'élément correspondant le plus proche dans la pile, s'il en existe un, et ne fera rien le cas échéant
- ☒ L'API `push` rajoute un élément dans la pile quoi qu'il arrive, alors que `navigate` cherchera à naviguer vers l'élément correspondant le plus proche dans la pile, s'il en existe un, et aura le même effet que `push` le cas échéant

Q En effet, `navigate` a le même effet que `push`, sauf s'il existe déjà un écran de ce type dans la pile. Dans ce cas, l'API naviguera à cet écran.

Exercice

Quelles phrases sont vraies quant à l'utilisation de `props.navigation` ou du hook `useNavigation` de `react-navigation` ?

- ☒ Ils remplissent la même fonction
- ☐ On ne peut utiliser `useNavigation` que dans les enfants d'un écran, mais pas dans un écran lui-même
- ☒ `useNavigation` permet d'utiliser le contexte de navigation dans n'importe quel composant qui se situe N niveaux en dessous d'un `NavigationContainer`
- ☐ `props.navigation` est à préférer dans un écran

Q `props.navigation` est surtout utile si nos composants sont basés sur des classes. Dans le cas où nous n'avons que des composants basés sur des fonctions, il est plus simple d'uniformiser et d'utiliser le hooks `useNavigation`, qui marche à tous les niveaux.

p. 22 Solution n°5

```

1 import * as React from 'react';
2 import {
3   ActivityIndicator,
4   View,
5   Text,
6   Button,
7   StyleSheet,
8 } from 'react-native';
9 import {
10   NavigationContainer,
11   useNavigation,
12   useRoute,
13 } from '@react-navigation/native';
14 import { createStackNavigator } from '@react-navigation/stack';
15
16 // Un tableau avec l'ID des TODOs à afficher, pour simplifier le code avec une boucle
17 const todosIds = [1, 2, 3];
18
19 // Un composant qui agit en tant qu'« Écran » simple, il se base sur la props navigation
    injectée par son parent
20 function HomeScreen(props) {
21   return (
22     <View style={homeStyles.container}>
23       <Text style={homeStyles.title}>Choisir un TODO</Text>
24       <View style={homeStyles.todosContainer}>
25         {todosIds.map((todoId) => (
26           <Button
27             key={todoId}
28             title={`Afficher le TODO n°` + todoId}
29             onPress={() => props.navigation.push('Todo', { todoId: todoId })}
30           </>
31         ))}
32       </View>
33     </View>
34   );
35 }

```

```

36
37 // Les styles liés au composant HomeScreen
38 const homeStyles = StyleSheet.create({
39   container: { flex: 1, alignItems: 'center', justifyContent: 'center' },
40   todosContainer: {
41     height: 200,
42     justifyContent: 'space-around',
43     padding: 20,
44     backgroundColor: 'lightblue',
45     borderRadius: 10,
46   },
47   title: { fontSize: 24, fontWeight: 'bold', marginBottom: 10 },
48 });
49
50 // Un composant qui agit en tant qu'« Écran » simple pour afficher le TODO passé en paramètre
51 function TodoScreen() {
52   const navigation = useNavigation();
53   const route = useRoute();
54   const [todo, setTodo] = React.useState(null);
55
56   // Récupération de la donnée du TODO via son API.
57   React.useEffect(() => {
58     fetch('https://jsonplaceholder.typicode.com/todos/' + route.params.todoId)
59       .then((response) => response.json())
60       .then((json) => setTodo(json));
61   }, [route.params.todoId]);
62
63   // Tant que l'API asynchrone n'a pas répondu, on affiche un loader, sinon on affiche le
64   TODO
65   return (
66     <View style={todoStyles.card}>
67       {!todo ? (
68         <ActivityIndicator />
69       ) : (
70         <React.Fragment>
71           <Text style={todoStyles.title}>{todo.title}</Text>
72           <Text style={todoStyles.status}>{todo.completed ? '✅' : '❌'}</Text>
73         </React.Fragment>
74       )}
75     </View>
76   );
77
78 const todoStyles = StyleSheet.create({
79   card: {
80     margin: 20,
81     padding: 20,
82     backgroundColor: 'lightblue',
83     borderRadius: 10,
84     flexDirection: 'row',
85     alignItems: 'center',
86   },
87   title: {
88     fontSize: 24,
89     fontWeight: 'bold',
90     flex: 1,
91   },
92   status: {},

```

```

93 });
94
95 // On crée un navigateur de pile, ici avec la configuration par défaut
96 const Stack = createStackNavigator();
97
98 // Notre application
99 function App() {
100   return (
101     <NavigationContainer>
102       <Stack.Navigator>
103         <Stack.Screen
104           name="Home"
105           component={HomeScreen}
106           options={{ title: 'Liste des TODOs' }}
107         />
108         <Stack.Screen
109           name="Todo"
110           component={TodoScreen}
111           // On personnalise le titre du header du TODO avec l'ID du todo
112           options={(props) => ({
113             title: 'Données du TODO n°' + props.route.params.todoId,
114           })}
115         />
116       </Stack.Navigator>
117     </NavigationContainer>
118   );
119 }
120
121 export default App;

```

L'application demandée contient deux écrans, avec un écran d'accueil permettant de naviguer vers un écran de visualisation de détails.

On peut commencer par la réalisation de l'écran d'accueil en nous basant sur une liste en dur d'éléments :

```

1 const todosIds = [1, 2, 3];
2
3 function HomeScreen(props) {
4   return (
5     <View style={homeStyles.container}>
6       <Text style={homeStyles.title}>Choisir un TODO</Text>
7       <View style={homeStyles.todosContainer}>
8         {todosIds.map((todoId) => (
9           <Button
10             key={todoId}
11             title={'Afficher le TODO n°' + todoId}
12             onPress={() => props.navigation.push('Todo', { todoId: todoId })}
13           />
14         ))}
15       </View>
16     </View>
17   );
18 }

```

L'affichage des boutons de navigation vers l'écran des détails correspondant à chaque élément se fait grâce à une boucle ajoutant un bouton pour chaque TODO existant dans le tableau `todosIds`. L'action de chacun des boutons consiste à ajouter un écran TODO dans la pile de navigation en passant en paramètre un objet sous la forme `{ todoId: todoId }` avec `props.navigation.push()`.

Ce composant devant être appelé dans la navigation, nous pouvons écrire les éléments nécessaires à son fonctionnement :

```
1 const Stack = createStackNavigator();
2
3 function App() {
4   return (
5     <NavigationContainer>
6       <Stack.Navigator>
7         <Stack.Screen
8           name="Home"
9           component={HomeScreen}
10          options={{ title: 'Liste des TODOs' }}
11        />
12        <Stack.Screen
13          name="Todo"
14          component={TodoScreen}
15          // On personnalise le titre du header du TODO avec l'ID du todo
16          options={(props) => ({
17            title: 'Données du TODO n°' + props.route.params.todoId,
18          })}
19        />
20      </Stack.Navigator>
21    </NavigationContainer>
22  );
23 }
```

On retrouve donc un conteneur, `<NavigationContainer>`, avec un navigateur `<Stack.Navigator>` définissant deux écrans, Home et Todo. L'utilisation des options permet de définir dynamiquement le titre de l'écran Todo grâce au contenu de la props route passée automatiquement lors de l'appel à l'écran.

Le composant `<HomeScreen>` ayant déjà été créé, il faut maintenant écrire le composant `<TodoScreen>`.

Celui-ci doit appeler une API externe afin d'obtenir les données nécessaires à son affichage. Cet appel sera effectué via un hook `useEffect`, appelé lorsque la valeur de `todoId` passée en paramètre change.

```
1 function TodoScreen() {
2   const navigation = useNavigation();
3   const route = useRoute();
4   const [todo, setTodo] = React.useState(null);
5
6   // Récupération de la donnée du TODO via son API.
7   React.useEffect(() => {
8     fetch('https://jsonplaceholder.typicode.com/todos/' + route.params.todoId)
9       .then((response) => response.json())
10      .then((json) => setTodo(json));
11   }, [route.params.todoId]);
12
13   // Tant que l'API asynchrone n'a pas répondu, on affiche un loader, sinon on affiche le
14   // TODO
15   return (
16     <View style={todoStyles.card}>
17       {!todo ? (
18         <ActivityIndicator />
19       ) : (
20         <React.Fragment>
21           <Text style={todoStyles.title}>{todo.title}</Text>
22           <Text style={todoStyles.status}>{todo.completed ? '✅' : '❌'}</Text>
23         </React.Fragment>
24       )}
25     </View>
26   );
27 }
```

25);

26 }