Les différents types de navigateurs

Table des matières

I. Contexte

Durée : 1 h

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

Pré-requis : Avoir vu le module sur les bases de la navigation dans une application React Native

Contexte

Pour apprendre le fonctionnement de base de la navigation dans une application React Native, nous avons étudié les navigateurs les plus proches de la gestion de la navigation dans une page web : les navigateurs en pile (`StackNavigator`).

Cependant, ce ne sont pas les seuls navigateurs disponibles dans une application mobile et, bien qu'on puisse inventer son propre type de navigateur, laissant libre cours à l'imagination des designers et des développeurs, `react-navigation` nous propose, en plus du navigateur en pile, deux autres types de navigateur : les navigateurs en tables (`TabNavigator`) et les navigateurs panneaux (`DrawerNavigator`).

Une application mobile est généralement la combinaison d'au moins deux de ces trois navigateurs et permet de définir des parcours utilisateurs complets et professionnels.

II. Le navigateur par onglets (TabNavigator)

Objectifs

- Voir les différents navigateurs par onglets
- Comprendre le contexte d'utilisation

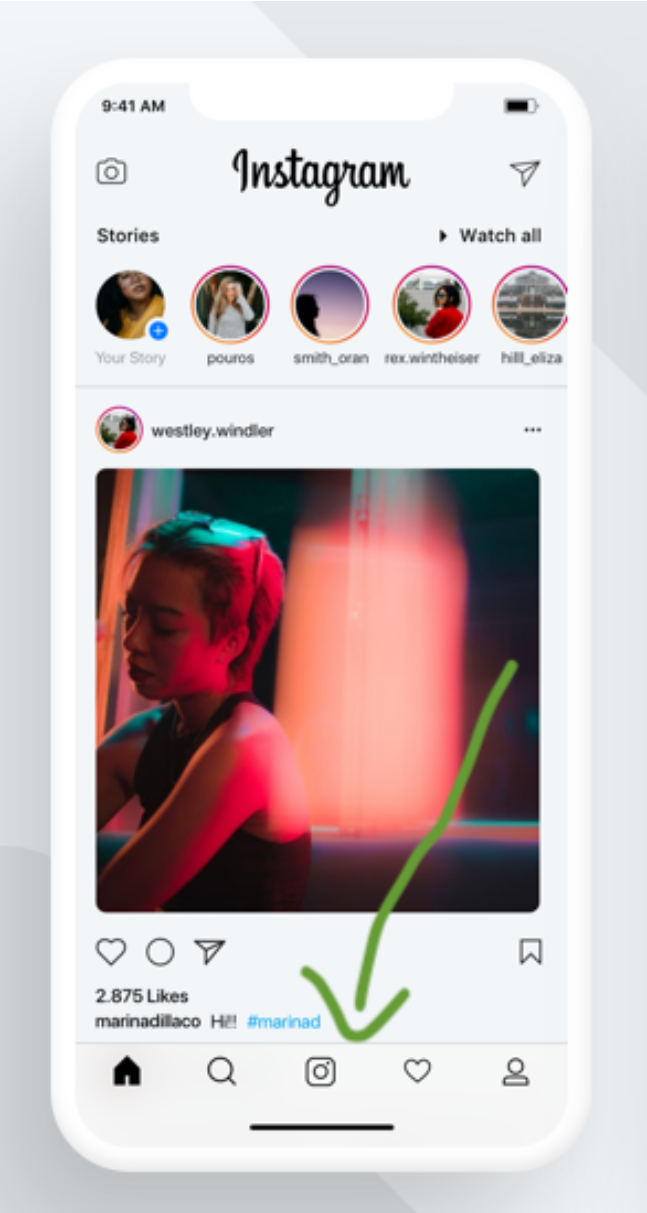
Mise en situation

Le navigateur par onglets (`TabNavigator`) est un navigateur très populaire pour la navigation mobile. En effet, les interfaces étant très petites à cause du format des écrans de téléphone et adaptées pour une utilisation tactile, cela en fait un pilier de choix dans la réalisation d'interfaces pratiques d'utilisation. Nous verrons également que le navigateur par onglets existe en plusieurs déclinaisons et que l'on peut personnaliser certains de ses éléments.

Qu'est-ce qu'un navigateur par onglets et quand l'utiliser ?

Le navigateur par onglets permet d'afficher un menu sur une ligne avec X éléments, on peut ensuite *swiper* ou cliquer sur un élément de ce menu pour afficher un écran correspondant.

Par exemple, l'application mobile d'Instagram utilise un navigateur par onglets en bas de son écran avec 5 éléments de navigation : ce sont des liens vers nos écrans.



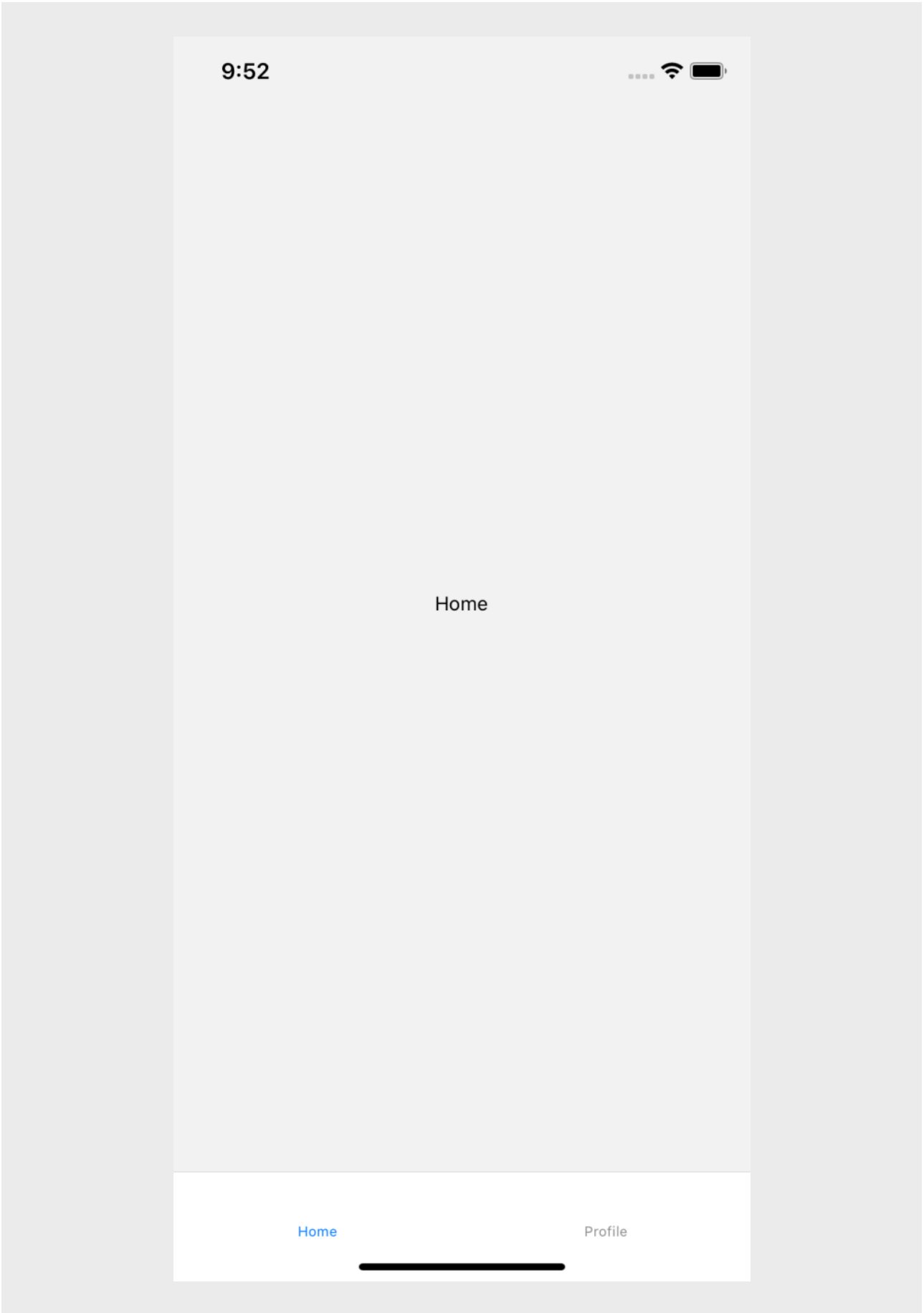
Pour utiliser un navigateur par onglets, il faut installer le module lié à ce dernier, `@react-navigation/bottom-tabs`. Mais, d'abord, il faut s'assurer d'avoir `@react-navigation/native` qui a normalement été installé dans le module précédent. Adaptons donc la commande d'installation suivante en fonction de nos besoins : `npm install @react-navigation/native @react-navigation/bottom-tabs`.

Voici un exemple très simple d'utilisation d'un navigateur par onglets qui s'attache en bas d'écran : il affiche deux écrans dans l'interface, un pour la page d'accueil et l'autre pour le profil de l'utilisateur.

Exemple

```
1 import * as React from 'react';
2 import { Text, View, StyleSheet } from 'react-native';
3 import { NavigationContainer } from '@react-navigation/native';
4 import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
5
6 // Notre premier écran
7 function HomeScreen() {
8   return (
9     <View style={styles.containers}>
```

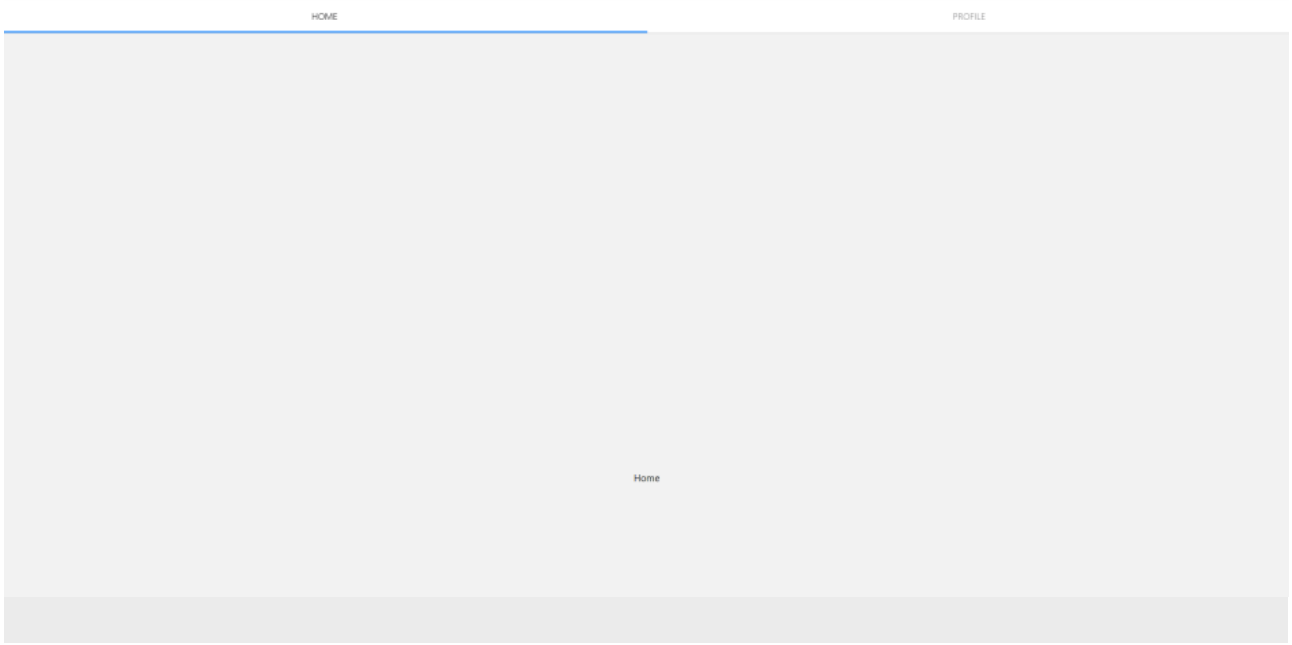
```
      <Text>Home</Text>
    </View>
  );
}

// Notre deuxième écran
function ProfileScreen() {
  return (
    <View style={styles.containers}>
      <Text>Profile</Text>
    </View>
  );
}

// Des styles communs pour l'affichage
const styles = StyleSheet.create({
  containers: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
});

// Notre navigateur en tables
const Tab = createBottomTabNavigator();

// Et notre application ainsi que la définition des écrans
export default function App() {
  return (
    <NavigationContainer>
      <Tab.Navigator>
        <Tab.Screen name="Home" component={HomeScreen} />
        <Tab.Screen name="Profile" component={ProfileScreen} />
      </Tab.Navigator>
    </NavigationContainer>
  );
}
```



Si l'on souhaite le même type de navigation, mais avec les onglets en haut, il faut importer `@react-navigation/material-top-tabs` avec la commande `npm install @react-navigation/material-top-tabs`.

À la suite de quoi, en changeant quelques lignes de l'exemple précédent, on obtient :

```
1 import { createMaterialTopTabNavigator } from '@react-navigation/material-top-tabs';
2 [...]
3 const Tab = createMaterialTopTabNavigator();
4 [...]
```



Pour ce qui concerne la déclaration du navigateur, il s'agit d'une structure très semblable à ce que l'on a vu avec `StackNavigator`, à savoir un composant parent qui prend des enfants `Screen` avec un nom et un composant.

Par défaut, les écrans sont paresseux (`lazy`), ce qui signifie que leurs composants ne seront montés qu'à la première navigation sur ces derniers. On peut passer une props `lazy={false}` à un `Tab.Navigator` si l'on souhaite pré-charger tous les écrans.

Syntaxe **À retenir**

- Le navigateur par onglets `TabNavigator` permet de mettre au point une autre expérience utilisateur très pratique pour une utilisation tactile. Il se décline par défaut en deux versions : une *bottom* qui permet d'afficher la barre de navigation sous les écrans comme vu dans nos exemples, ou une version *top* pour les applications qui préfèrent le menu au-dessus des écrans.
- C'est un outil que l'on peut combiner avec d'autres navigateurs, comme le navigateur en pile pour réaliser des interfaces complexes et pratiques.

Complément

- https://reactnavigation.org/docs/tab-based-navigation

III. Exercice : Appliquez la notion

Question

[solution n°1 p.21]

Étant donné qu'un navigateur n'est rien d'autre qu'un composant amélioré, imaginons une structure avec une navigation à 3 onglets, affichant pour chacun son propre navigateur en pile qui permet d'empiler un nombre infini d'écrans, et indiquant le niveau de l'écran dans la pile sur l'écran courant.

En indice, le corrigé d'un exercice vu précédemment proche de ce scénario. Et, ci-dessous, le rendu escompté, avec chaque état bien séparé entre les différentes tabs :

[cf. nav-adv-2.mp4]

Indice :

```
import * as React from 'react';
import {
  ActivityIndicator,
  View,
  Text,
  Button,
  StyleSheet,
} from 'react-native';
import {
  NavigationContainer,
  useNavigation,
  useRoute,
} from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';


// Notre composant personnalisé qui gère notre compteur entre les écrans
function HomeScreen(props) {
  const route = useRoute()
  const currentIndex = typeof route.params !== "undefined" ? route.params.index : 0

  return (
    <View style={homeStyles.container}>
      <Text style={homeStyles.title}>Index dans la pile {currentIndex}</Text>
      <Button title="Rajouter un écran dans la pile" onPress={() =>
props.navigation.push("Home", { index: currentIndex + 1})}/>
      <Button title="Démonter la pile" onPress={() => props.navigation.popToTop()}/>
    </View>
  );
}

// Les styles liés au composant HomeScreen
const homeStyles = StyleSheet.create({
  container: { flex: 1, alignItems: 'center', justifyContent: 'center' },
  title: { fontSize: 24, fontWeight: 'bold', marginBottom: 10 },
});

// On créé un navigateur de pile, ici avec la configuration par défaut
const Stack = createStackNavigator();

// Notre application
function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
```

```
          name="Home"
          component={HomeScreen}
          options={{ headerShown: false }}
        />
      </Stack.Navigator>
    </NavigationContainer>
  );
}

export default App;
```

IV. Le navigateur en panneau (DrawerNavigator)

Précision sur useRoute()

useRoute est un hook qui donne accès à l'objet "route". Il va nous retourner les propriétés de l'écran et nous pourrons alors accéder depuis cet objet aux données de l'écran et nous en servir pour naviguer.

Objectifs

- Comprendre le fonctionnement d'un navigateur panneau
- Découvrir un exemple d'un navigateur panneau

Mise en situation

Le navigateur panneau (`DrawerNavigator`) est un navigateur très populaire pour la navigation mobile. Du fait d'interfaces réduites à cause de la taille de l'écran d'un téléphone et généralement en mode portrait, les designers ont dû trouver des adaptations pour offrir une navigation efficace.

De là est né le navigateur panneau, qui consiste en un panneau sortant d'un côté de l'écran pour venir se superposer à l'interface. Il est généralement déclenché par un glissement de doigt du bord vers le centre de l'écran ou via son fameux bouton burger[1], très utilisé en UX design.

Qu'est-ce qu'un navigateur panneau et quand l'utiliser ?

Très utilisé dans le webdesign, on le retrouve forcément sur l'environnement des applications mobiles. Voici un exemple de panneau déclenché par un burger menu :

[cf. nav-adv-3.mp4]

Grâce à `react-navigation`, nous allons pouvoir développer un élément semblable. Mais, avant, il faut installer une librairie additionnelle grâce à la commande `npm install @react-navigation/drawer`.

Attention

De par sa nature, il est préférable de n'avoir qu'un seul `DrawerNavigator` dans son application. En effet, en avoir plusieurs pourrait déclencher des conflits qui casseraient le fonctionnement de l'application.

Voici un exemple de code pour l'utilisation d'un navigateur panneau.

1 https://fr.wikipedia.org/wiki/Menu_lat%C3%A9ral

Exemple

```
import * as React from 'react';
import { Button, View } from 'react-native';
import { createDrawerNavigator } from '@react-navigation/drawer';
import { NavigationContainer } from '@react-navigation/native';

function HomeScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Button
        onPress={() => navigation.navigate('Notifications')}
        title="Go to notifications"
      />
    </View>
  );
}

function NotificationsScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Button onPress={() => navigation.goBack()} title="Go back home" />
    </View>
  );
}

const Drawer = createDrawerNavigator();

export default function App() {
  return (
    <NavigationContainer>
      <Drawer.Navigator initialRouteName="Home">
        <Drawer.Screen name="Home" component={HomeScreen} />
        <Drawer.Screen name="Notifications" component={NotificationsScreen} />
      </Drawer.Navigator>
    </NavigationContainer>
  );
}
```

[cf. nav-adv-8.mp4]

Par défaut, le navigateur panneau se trouve sur la gauche. On peut cependant personnaliser cela en paramétrant la props `drawerPosition` à `right` ou `left`.

```
export default function App() {
  return (
    <NavigationContainer>
      <Drawer.Navigator
        initialRouteName="Home"
        drawerPosition="right"
      >
        <Drawer.Screen name="Home" component={HomeScreen} />
        <Drawer.Screen name="Notifications" component={NotificationsScreen} />
      </Drawer.Navigator>
    </NavigationContainer>
  );
}
```

Il permet également par défaut le geste de *swipe* du bord vers le centre, ce qui est très pratique, mais on peut également déclencher son ouverture manuellement via un bouton dans l'interface. Nous verrons cela un peu plus tard dans un exercice.

Dans l'exemple plus haut, nous déclarons un `DrawerNavigator` avec deux écrans, `Home` et `Notifications`. On peut également naviguer vers ces écrans sans utiliser le *drawer* grâce aux boutons et à l'API traditionnelle `navigate`.

Syntaxe **À retenir**

- Le navigateur en panneau est très pratique, car il n'occupe de l'espace à l'écran que lorsqu'il est affiché. C'est un incontournable de l'expérience utilisateur sur téléphone portable et il se couple parfaitement aux deux autres navigateurs fournis par `react-navigation`.
- Il n'en faut qu'un seul dans toute une application, sous peine de créer des conflits de navigation, mais il s'assemble parfaitement avec n'importe quel autre type de navigateur, car, comme les autres, il prend des composants en arguments.

Complément

- https://reactnavigation.org/docs/drawer-based-navigation

V. Exercice : Appliquez la notion

Question

[solution n°2 p.22]

Pour cet exercice, il vous faut créer une interface proche de celle présentée sur la vidéo ci-dessous.

Vous aurez besoin d'un navigateur en panneau et de créer un *burger button* présent dans tous les écrans. Pour cela, il suffit d'aligner trois composants `View` d'une hauteur de 5 px et d'une largeur de 30 px avec un fond noir, le tout ayant pour parent un composant `TouchableOpacity` d'une hauteur de 25 px en mode `justifyContent: "space-between"` afin de créer l'espacement entre les lignes du burger.

Lorsque l'on clique sur ce bouton, on utilise la fonction `navigation.toggleDrawer();` afin d'ouvrir ou fermer le panneau, selon la logique d'un interrupteur.

Dans le panneau, il y a trois éléments de menu : **Accueil**, **Résultats** et **Profil**.

[cf. nav-adv-4.mp4]

Indice :

Un *burger button* peut s'écrire de la manière suivante :

```
function BurgerButton(props) {
  return (
    <TouchableOpacity
      onPress={props.onPress}
      style={burgerButtonStyles.wrapper}>
      <View style={burgerButtonStyles.line} />
      <View style={burgerButtonStyles.line} />
      <View style={burgerButtonStyles.line} />
    </TouchableOpacity>
  );
}

const burgerButtonStyles = StyleSheet.create({
  wrapper: {
    width: 30,
    height: 25,
```

```
    justifyContent: 'space-between',
  },
  line: {
    height: 5,
    width: '100%',
    borderRadius: 2,
    backgroundColor: 'black',
  },
});
```

VI. Navigation avancée

Objectifs

- Découvrir l'utilisation avancée du `StackNavigator`
- Comprendre l'affichage conditionnel
- Réaliser une fenêtre modale

Mise en situation

Maintenant que nous avons vu le fonctionnement de trois navigateurs très complémentaires, nous allons étudier les manières de gérer les interactions entre eux, notamment à travers l'exemple d'un système modal global à l'application. Nous verrons également la construction conditionnelle de la navigation, ce qui permet de gérer des cas très fréquents de navigation et qui seraient complexes sans cela (par exemple, le flux d'authentification d'un utilisateur).

Une navigation complète à base de modale globale

Imaginons le cas d'utilisation où nous souhaitons afficher une barre de navigation en bas de l'écran avec deux éléments de menu, puis, à tout moment, on souhaite pouvoir afficher une modale qui se situera par-dessus ce menu. Pour cela, il est nécessaire d'imbriquer les navigateurs de manière à ce que le `TabNavigator` soit inclus dans un `StackNavigator`.

Petit point important sur <React.Fragment> : En React Native, il est courant pour un composant de renvoyer plusieurs éléments. Les fragments nous permettent de grouper une liste d'enfants sans ajouter de nœud supplémentaire à notre écran, ce qui est plus rapide et utilise moins de mémoire. Ils facilitent également les relations parent-enfant de certains mécanismes comme Flexbox et CSS Grid. L'ajout de <View> pourrait rendre difficile le maintien de la mise en page souhaitée lors de l'extraction des composants logiques.

Voyons un exemple de cela :

Exemple

```
import * as React from "react";
import { Button, Text, View } from "react-native";
import Constants from "expo-constants";
import { NavigationContainer } from "@react-navigation/native";
import { createStackNavigator } from "@react-navigation/stack";
import { createBottomTabNavigator } from "@react-navigation/bottom-tabs";

// Nos navigateurs
const MainBottomNavigation = createBottomTabNavigator();
const AppNavigation = createStackNavigator();

// Un composant simple qui affiche le nom de la route courante.
function NamedScreenComponent(props) {
```

```
  return (
    <React.Fragment>
      <Text>{props.route.name}</Text>
      <Button
        title="Ouvrir modale 1"
        onPress={() => props.navigation.navigate("Modale1")}
      />
      <Button
        title="Ouvrir modale 2"
        onPress={() => props.navigation.navigate("Modale2")}
      />
    </React.Fragment>
  );
}

// Notre navigation de bas d'écran
const BottomNav = () => (
  <MainBottomNavigation.Navigator>
    <MainBottomNavigation.Screen
      name="Premier"
      component={NamedScreenComponent}
    />
    <MainBottomNavigation.Screen
      name="Deuxieme"
      component={NamedScreenComponent}
    />
  </MainBottomNavigation.Navigator>
);

// Notre application avec notre navigateur en pile pour les modales
export default function App() {
  return (
    <View style={{ flex: 1, paddingTop: Constants.statusBarHeight }}>
      <NavigationContainer>
        <AppNavigation.Navigator mode="modal">
          <AppNavigation.Screen
            name="bottom"
            options={{ headerShown: false }}
            component={BottomNav}
          />
          <AppNavigation.Screen
            name="Modale1"
            component={NamedScreenComponent}
          />
          <AppNavigation.Screen
            name="Modale2"
            component={NamedScreenComponent}
          />
        </AppNavigation.Navigator>
      </NavigationContainer>
    </View>
  );
}

```

[cf. nav-adv-9.mp4]

L'exemple ci-dessus utilise comme navigateur principal un `StackNavigator`, nommé `AppNavigation`, avec comme premier écran un composant de navigation par onglets `TabNavigator`, appelé `MainBottomNavigation`, définissant deux éléments. De cette manière, la navigation du bas (`MainBottomNavigation`) et ses éléments feront partie de la boîte où s'affichent les écrans de notre navigateur en pile (`AppNavigation`).

Une fois que nous avons cela, si nous souhaitons ajouter une modale, il suffit de la déclarer à la suite de notre écran `MainBottomNavigation` dans la déclaration de notre navigateur en pile (`AppNavigation`).

On peut par la suite ajouter autant de modales qu'on le souhaite et adapter cette composition comme bon nous semble. Petit plus : nous avons personnalisé l'animation de ce navigateur pour que celle-ci change grâce à la props `mode="modal"`. Cela se verra essentiellement sur iOS, où les modales apparaîtront depuis le bas de l'écran et seront *swipeables* vers le bas, un pattern très commun sur iOS.

Utiliser un affichage conditionnel pour gérer un flux d'authentification

Un autre cas d'utilisation avancé et très pratique est la notion de « navigation dynamique ». En effet, imaginons que nous ayons une application où il y a deux modes : connecté/déconnecté. C'est un cas d'usage très classique et, grâce à `react-navigation`, on peut y répondre très simplement, ce qui n'a pas toujours été le cas.

Comment est-ce que ça fonctionne ?

Le principe est simple : en reprenant l'exemple vu précédemment, nous allons scinder notre navigation en deux parties distinctes, que nous afficherons conditionnellement selon l'état de connexion de l'utilisateur. Nous afficherons également un écran de chargement le temps que l'on récupère cet état. Pour cela, nous allons simuler une requête asynchrone à un état grâce à `setTimeout` afin de voir le *loader* s'afficher, puis nous afficherons aléatoirement le mode connecté ou déconnecté.

Exemple

```
import * as React from "react";
import { ActivityIndicator, Text, View } from "react-native";
import Constants from "expo-constants";
import { NavigationContainer } from "@react-navigation/native";
import { createStackNavigator } from "@react-navigation/stack";
import { createBottomTabNavigator } from "@react-navigation/bottom-tabs";

// Nos navigateurs
const MainBottomNavigation = createBottomTabNavigator();
const AppNavigation = createStackNavigator();

// Un composant simple qui affiche le nom de la route courante.
function NamedScreenComponent(props) {
  return (
    <React.Fragment>
      <Text>{props.route.name}</Text>
    </React.Fragment>
  );
}

// Notre navigation de bas d'écran
const BottomNav = () => (
  <MainBottomNavigation.Navigator>
    <MainBottomNavigation.Screen
      name="Premier"
      component={NamedScreenComponent}
    />
```

```
    <MainBottomNavigation.Screen
      name="Deuxieme"
      component={NamedScreenComponent}
    />
  </MainBottomNavigation.Navigator>
);

// Notre application avec notre navigateur en pile pour les modales
export default function App() {
  const [loginState, setLoginState] = React.useState("unknown");

  React.useEffect(() => {
    // On affiche aléatoirement l'état de connexion après 2 secondes
    setTimeout(() => {
      setLoginState(Math.random() > 0.5 ? "loggedIn" : "loggedOut");
    }, 2000);
  }, []);

  // On affiche le chargement au démarrage le temps de récupérer l'état de connexion
  if (loginState === "unknown") {
    return (
      <View style={{ flex: 1, justifyContent: "center", alignItems: "center" }}>
        <ActivityIndicator size="large" />
        <Text>Chargement en cours...</Text>
      </View>
    );
  }

  return (
    <View style={{ flex: 1, paddingTop: Constants.statusBarHeight }}>
      <NavigationContainer>
        <AppNavigation.Navigator mode="modal">
          {loginState === "loggedIn" ? (
            <AppNavigation.Screen
              name="bottom"
              options={{ headerShown: false }}
              component={BottomNav}
            />
          ) : (
            <AppNavigation.Screen
              name="Écran de connexion"
              options={{ headerShown: false }}
              component={NamedScreenComponent}
            />
          )}
          <AppNavigation.Screen
            name="Modale1"
            component={NamedScreenComponent}
          />
          <AppNavigation.Screen
            name="Modale2"
            component={NamedScreenComponent}
          />
        </AppNavigation.Navigator>
      </NavigationContainer>
    </View>
  );
```

```
85 }
86
```

[cf. nav-adv-10.mp4]

En rafraîchissant l'application, on pourra constater, qu'aléatoirement, une fois sur deux, il y aura le panneau de connexion ou le panneau connecté. Bien évidemment, dans une vraie application, il faut se baser sur un état de connexion vérifiée avec son back-end.

Syntaxe **À retenir**

- Il est possible de composer une navigation très avancée en combinant les trois navigateurs de `react-navigation`, cela nous permet notamment de gérer un système de modales et de contrôler la superposition des éléments dans l'interface.
- Pour aller plus loin, on peut également composer conditionnellement la navigation en utilisant de simples structures conditionnelles. Cela nous permettra de créer des scénarios complexes, par exemple pour l'*onboarding* d'un utilisateur ou bien pour masquer des éléments de l'interface à des membres qui n'ont pas payé d'accès premium à une fonctionnalité.

Complément

- https://reactnavigation.org/docs/modal
- https://reactnavigation.org/docs/auth-flow

VII. Exercice : Appliquez la notion

Question

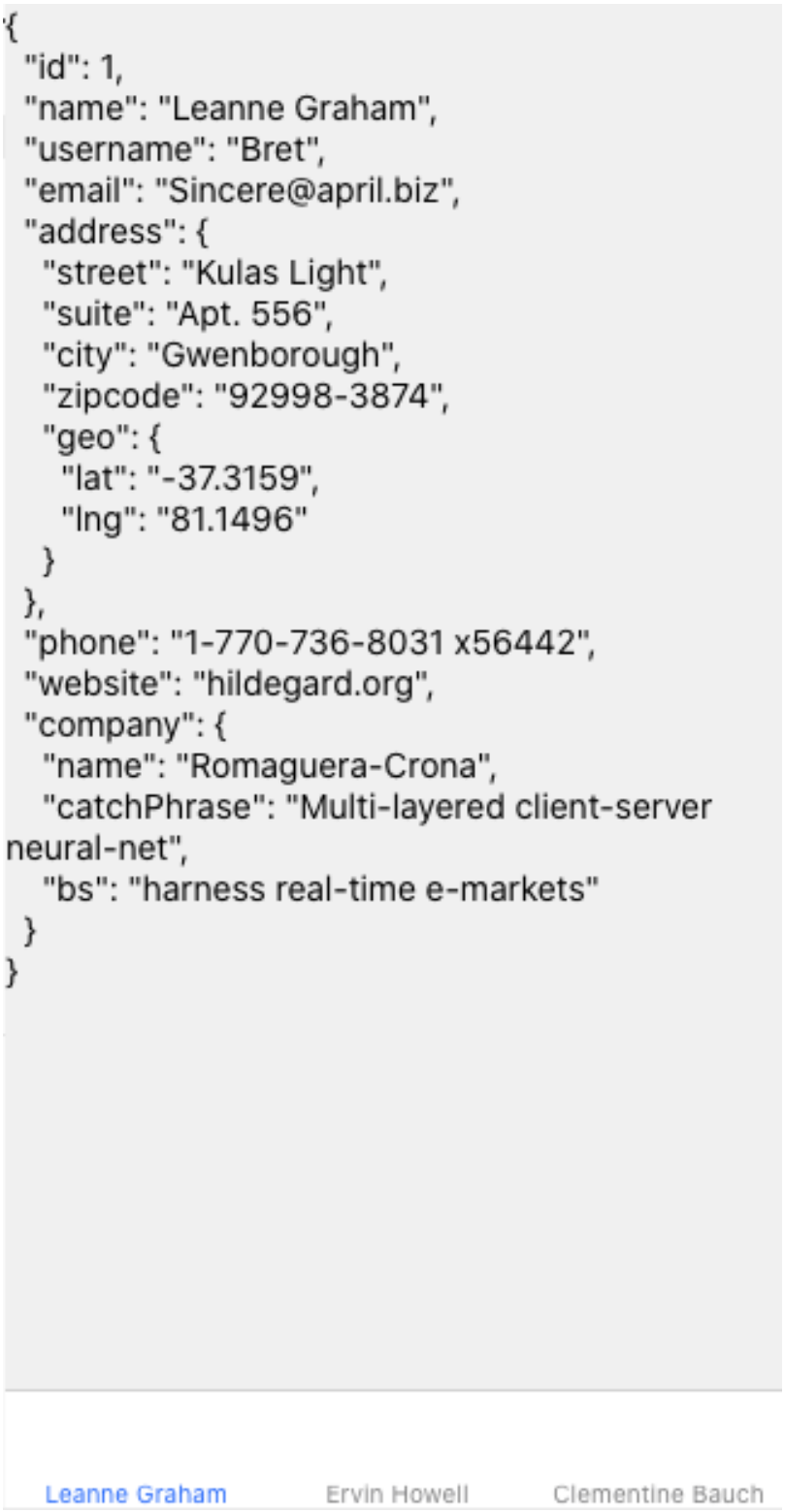
[solution n°3 p.24]

Créez une navigation dynamique avec un `TabNavigator` qui affichera trois utilisateurs basés sur les premiers éléments du Web service `https://jsonplaceholder.typicode.com/users`.

Chaque tab devra afficher le nom de l'utilisateur et, lorsqu'on clique dessus, afficher les données liées à cet utilisateur grâce à `JSON.stringify`. En attendant que les données génèrent la navigation, affichez un *loader* grâce à `ActivityIndicator`.

Pour aller plus loin, et sachant que vous possédez déjà les données, essayez de ne pas refaire l'appel HTTP pour sélectionner un seul utilisateur, afin d'économiser des données réseau. Pour ce faire, allez voir https://reactnavigation.org/docs/hello-react-navigation#passing-additional-props.

La capture d'écran ci-dessous montre le rendu escompté :



Indice :

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

```
fetch('https://jsonplaceholder.typicode.com/users')
  .then(response => response.json())
  .then(json => console.log(json))

```

L'API `https://jsonplaceholder.typicode.com/users` renvoie des données sous la forme d'un tableau d'objets respectant la structure suivante :

```
[
  {
    "id": 1,
    "name": "Leanne Graham",
    "username": "Bret",
    "email": "Sincere@april.biz",
    "address": {
      "street": "Kulas Light",
      "suite": "Apt. 556",
      "city": "Gwenborough",
      "zipcode": "92998-3874",
      "geo": {
        "lat": "-37.3159",
        "lng": "81.1496"
      }
    },
    "phone": "1-770-736-8031 x56442",
    "website": "hildegard.org",
    "company": {
      "name": "Romaguera-Crona",
      "catchPhrase": "Multi-layered client-server neural-net",
      "bs": "harness real-time e-markets"
    }
  },
  // ...
]
```

VIII. Essentiel

`react-navigation` nous offre trois navigateurs qui permettent de réaliser la plupart des interfaces imaginables. On peut configurer ces derniers et les composer entre eux très facilement, car ils ne sont que des composants React améliorés.

Le `TabNavigator` permet de masquer ou d'afficher des écrans grâce à un menu horizontal, qui peut se situer en dessous ou au-dessus des écrans qu'il affiche.

Le `DrawerNavigator`, lui, permet d'utiliser un panneau qui sort d'un côté de l'application au *swipe* ou au déclenchement d'un bouton.

Enfin, en utilisant le `StackNavigator`, on peut définir un routeur qui servira à gérer des modales qui passeront par-dessus un potentiel `TabNavigator`, c'est un pattern très classique de la réalisation d'applications mobiles.

Tous les `navigators` sont adaptables au *runtime* et peuvent être modifiés à la volée, ce qui permet de réaliser des navigations dynamiques en fonction de données asynchrones ou bien de gérer différents états utilisateurs, comme le mode connecté/déconnecté.

IX. Auto-évaluation

A. Exercice final

Exercice 1

[solution n°4 p.24]

Exercice

Un navigateur par onglets `TabNavigator` est forcément en dessous de l'écran qu'il affiche.

- Vrai
- Faux

Exercice

Un navigateur par onglets `TabNavigator` est par défaut en mode paresseux (`lazy`).

- Vrai
- Faux

Exercice

Un navigateur par onglets `TabNavigator` ne possède qu'un seul écran monté à la fois.

- Vrai, toujours
- Vrai, parfois
- Faux

Exercice

Il est conseillé d'avoir plusieurs `DrawerNavigator` dans une application.

- Vrai
- Faux

Exercice

Pour installer un `DrawerNavigator`, j'ai besoin du package...

- @react-navigation/drawer-navigator
- @react-navigation/drawer
- @navigation/drawer

Exercice

Pour permuter à la manière d'un interrupteur l'affichage d'un `DrawerNavigator`, j'utilise la méthode...

- navigation.toggleDrawer();
- navigation.toggle();
- route.toggleDrawer();

Exercice

Il est possible de combiner un `StackNavigator` et un `TabNavigator` ensemble dans n'importe quel ordre.

- Seulement dans l'ordre de l'énoncé
- Vrai
- Faux

Exercice

Pour créer un affichage de modale par-dessus le menu d'un `TabNavigator`, je dois...

- ○ Déclarer un `StackNavigator` possédant un `TabNavigator`
- ○ Déclarer un `TabNavigator` possédant un `StackNavigator`

Exercice

On peut composer les navigateurs à la volée avec une structure conditionnelle.

- ○ Vrai
- ○ Faux

Exercice

Si un navigateur ne possède pas au moins un écran, que se passe-t-il ?

- ○ Rien du tout
- ○ L'application plante

B. Exercice : Défi

Maintenant que nous avons vu toutes ces notions, faisons un petit exercice pour combiner les savoirs.

Question [solution n°5 p.26]

Il vous faut réaliser une application, avec pour élément d'interface principal un `TabNavigator` *bottom* à deux éléments :

- Todo, qui est un `StackNavigator`, avec par défaut une liste de TODO affichée, et dans lequel on peut empiler un écran affichant un TODO individuel. Cet écran ne doit pas passer par-dessus la navigation du bas d'écran. Vous pouvez vous baser sur ce qui a été vu précédemment dans le module de navigation de base, notamment l'exercice final.
- Profile, qui est un simple composant React affichant votre profil utilisateur avec prénom, âge et un bouton « **Voir plus des détails** », qui ouvre un écran avec une petite description de soi. Cet écran s'affiche dans une modale qui passe par-dessus la navigation du bas.

Le rendu escompté est présenté ci-dessous :

[cf. nav-adv-6.mp4]

Solutions des exercices

p. 8 Solution n°1

```
import * as React from 'react';
import { Text, Button, View, StyleSheet } from 'react-native';
import {
  NavigationContainer,
  useNavigation,
  useRoute,
} from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';

// Notre composant personnalisé qui gère notre compteur entre les écrans
// Pour rappel, memo va mémoriser le retour de la fonction selon la
dépendance spécifiée en second paramètre. Si lors d'un nouveau rendu la
dépendance n'a pas changée, alors on récupérera la valeur en mémoire et on
évitera de faire le traitement.
const StackingScreen = React.memo(function StackingScreen(props) {
  const route = useRoute();

  const currentIndex =
    typeof route.params !== 'undefined' ? route.params.index : 0;

  return (
    <View style={homeStyles.container}>
      <Text style={homeStyles.title}>Index dans la pile {currentIndex}</Text>
      <Button
        title="Rajouter un écran dans la pile"
        onPress={() =>
          props.navigation.push(props.mainScreenName, { index: currentIndex + 1 })
        }
      />
      <Button
        title="Démonter la pile"
        onPress={() => props.navigation.popToTop()}
      />
    </View>
  );
})

// Les styles liés au composant HomeScreen
const homeStyles = StyleSheet.create({
  container: { flex: 1, alignItems: 'center', justifyContent: 'center' },
  title: { fontSize: 24, fontWeight: 'bold', marginBottom: 10 },
});

// On utilise une fonction utilitaire qui factorise le code et nous permet de générer des
StackNavigator à la volée et de les nommer
const CustomStack = (mainScreenName) => {
  const Stack = createStackNavigator();

  // Noter la syntaxe avec la render function : https://reactnavigation.org/docs/hello-react-
navigation#passing-additional-props
  return () => (
    <Stack.Navigator>
      <Stack.Screen name={mainScreenName} options={{ headerShown: false }}>
        {(props) => <StackingScreen {...props} mainScreenName={mainScreenName} />}
```

```
      </Stack.Screen>
    </Stack.Navigator>
  );
};

// Notre navigateur par onglets
const Tab = createBottomTabNavigator();

// Et notre application ainsi que la définition des écrans
const HomeStack = CustomStack('HomeStack');
const ResultsStack = CustomStack('ResultsStack');
const ProfileStack = CustomStack('ProfileStack');

export default function App() {
  // On ajoute 3 StackNavigator à-la-volée
  return (
    <NavigationContainer>
      <Tab.Navigator>
        <Tab.Screen name="Home" component={HomeStack} />
        <Tab.Screen name="Results" component={ResultsStack} />
        <Tab.Screen name="Profile" component={ProfileStack} />
      </Tab.Navigator>
    </NavigationContainer>
  );
}
```

p. 11 Solution n°2

```
/* eslint-disable */
import * as React from 'react';
import { Button, View, TouchableOpacity, StyleSheet, Text } from 'react-native';
import { NavigationContainer, useNavigation } from '@react-navigation/native';
import { createDrawerNavigator } from '@react-navigation/drawer';

// Un composant bouton burger classique
function BurgerButton(props) {
  return (
    <TouchableOpacity
      onPress={props.onPress}
      style={burgerButtonStyles.wrapper}>
      <View style={burgerButtonStyles.line} />
      <View style={burgerButtonStyles.line} />
      <View style={burgerButtonStyles.line} />
    </TouchableOpacity>
  );
}

// Ses styles
const burgerButtonStyles = StyleSheet.create({
  wrapper: {
    width: 30,
    height: 25,
    justifyContent: 'space-between',
  },
  line: {
    height: 5,
    width: '100%',
```

```
    borderRadius: 2,
    backgroundColor: 'black',
  },
});

// Cette fonction prend en paramètre un nom et retourne un composant écran de ce nom
function screenComponentFactory(name) {
  return () => {
    const navigation = useNavigation();

    return (
      <View style={screenStyles.wrapper}>
        <View style={screenStyles.header}>
          <BurgerButton onPress={navigation.toggleDrawer} />
          <Text style={screenStyles.title}>{name}</Text>
        </View>
      </View>
    );
  };
}

// Ses styles
const screenStyles = StyleSheet.create({
  wrapper: {
    flex: 1,
  },
  header: {
    flexDirection: 'row',
    alignItems: 'center',
    backgroundColor: 'lightgray',
    padding: 20,
  },
  title: { flex: 1, textAlign: 'center', fontSize: 24, fontWeight: 'bold' },
});

// Creation de notre drawer
const Drawer = createDrawerNavigator();

// Creation de nos écrans
const HomeScreen = screenComponentFactory('Accueil');
const ResultsScreen = screenComponentFactory('Résultats');
const ProfileScreen = screenComponentFactory('Profil');

// Notre application
export default function App() {
  return (
    <NavigationContainer>
      <Drawer.Navigator initialRouteName="Home">
        <Drawer.Screen name="Home" options={{title: "Accueil"}} component={HomeScreen} />
        <Drawer.Screen name="Results" options={{title: "Résultats"}} component=
{ResultsScreen} />
        <Drawer.Screen name="Profile" options={{title: "Profil"}} component={ProfileScreen}
/>
      </Drawer.Navigator>
    </NavigationContainer>
  );
}
```

p. 16 Solution n°3

```
import * as React from 'react';
import { ActivityIndicator, Text, View } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';

// Notre composant d'affichage, qui utilise React.memo pour des optimisations de performances.
const UserDetails = React.memo((props) => (
  <Text>{JSON.stringify(props.user, null, 2)}</Text>
));

const Tab = createBottomTabNavigator();

export default function App() {
  const [users, setUsers] = React.useState([]);

  // On récupère les 3 premiers utilisateurs
  React.useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/users')
      .then((response) => response.json())
      .then((json) => setUsers(json.slice(0, 3)));
  }, []);

  // En attendant que l'on récupère les utilisateurs
  if (users.length <= 0) {
    return (
      <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
        <ActivityIndicator size="large" />
        <Text>Chargement en cours...</Text>
      </View>
    );
  }

  // On génère dynamiquement la navigation
  return (
    <NavigationContainer>
      <Tab.Navigator>
        {users.map((user) => (
          <Tab.Screen name={'User' + user.name} options={{ title: user.name }}>
            {(props) => <UserDetails key={user.id} {...props} user={user} />}
          </Tab.Screen>
        ))}
      </Tab.Navigator>
    </NavigationContainer>
  );
}
```

Exercice p. 18 Solution n°4

Exercice

Un navigateur par onglets `TabNavigator` est forcément en dessous de l'écran qu'il affiche.

- ○ Vrai
- ◉ Faux

Il existe plusieurs variantes de navigateur par onglets : `createMaterialTopTabNavigator` et `createBottomTabNavigator`, par exemple.

Exercice

Un navigateur par onglets `TabNavigator` est par défaut en mode paresseux (`lazy`).

- (•) Vrai
- ( ) Faux

Par défaut, pour des raisons de performances, le navigateur est paresseux. On peut changer ce comportement en passant la propriété `lazy` à `false` dans la configuration.

Exercice

Un navigateur par onglets `TabNavigator` ne possède qu'un seul écran monté à la fois.

- ( ) Vrai, toujours
- (•) Vrai, parfois
- ( ) Faux

Sachant qu'il est en mode `lazy` par défaut, au premier rendu, il n'y a qu'une seule tab affichée. Il n'y a donc qu'un seul écran monté. À partir du moment où on visite une autre section de ce dernier, il maintient l'état du premier écran et en ouvre un deuxième : cela devient donc faux.

Exercice

Il est conseillé d'avoir plusieurs `DrawerNavigator` dans une application.

- ( ) Vrai
- (•) Faux

Faux, car cela est vivement déconseillé. En effet, `react-navigation` n'est pas conçu pour et cela risquerait de générer des conflits d'état de navigation. Cependant, comme la navigation est dynamiquement personnalisable, on pourrait adapter le contenu d'un seul *drawer* sans problèmes, en fonction du contexte.

Exercice

Pour installer un `DrawerNavigator`, j'ai besoin du package...

- ( ) @react-navigation/drawer-navigator
- (•) @react-navigation/drawer
- ( ) @navigation/drawer

En effet, nous avons besoin du package `@react-navigation/drawer` exportée du `namespace` de `react-navigation`.

Exercice

Pour permuter à la manière d'un interrupteur l'affichage d'un `DrawerNavigator`, j'utilise la méthode...

- (•) navigation.toggleDrawer();
- ( ) navigation.toggle();
- ( ) route.toggleDrawer();

La bonne réponse est la première, c'est une méthode de l'objet `navigation`.

Exercice

Il est possible de combiner un `StackNavigator` et un `TabNavigator` ensemble dans n'importe quel ordre.

- ○ Seulement dans l'ordre de l'énoncé
- ◉ Vrai
- ○ Faux

On peut combiner les navigateurs dans l'ordre que l'on souhaite.

Exercice

Pour créer un affichage de modale par-dessus le menu d'un `TabNavigator`, je dois...

- ◉ Déclarer un `StackNavigator` possédant un `TabNavigator`
- ○ Déclarer un `TabNavigator` possédant un `StackNavigator`

En effet, le `StackNavigator` étant antérieur, il s'affichera par-dessus les éléments du navigateur par onglets.

Exercice

On peut composer les navigateurs à la volée avec une structure conditionnelle.

- ◉ Vrai
- ○ Faux

Grâce à `react-navigation` v5, on peut en effet composer les navigateurs et leurs écrans à la volée.

Exercice

Si un navigateur ne possède pas au moins un écran, que se passe-t-il ?

- ○ Rien du tout
- ◉ L'application plante

L'application plantera pour permettre de relever le problème dans un environnement de développement

p. 20 Solution n°5

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

```
import {
  NavigationContainer,
  useNavigation,
  useRoute,
} from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';

// TODO TAB
const TodoStack = createStackNavigator();

// Le navigateur en pile du menu TODO
function TodoStackScreen() {
  return (
    <TodoStack.Navigator>
      <TodoStack.Screen
        name="TodoHome"
        options={{ headerShown: false }}
        component={TodoScreen}
      />
      <TodoStack.Screen
        name="TodoDetail"
        options={(props) => ({ title: 'TODO n°' + props.route.params.todoId })}
        component={TodoDetailScreen}
      />
    </TodoStack.Navigator>
  );
}

// Pour voir la TODO list
function TodoScreen(props) {
  return (
    <View style={homeStyles.container}>
      <Text style={homeStyles.title}>Choisir un TODO</Text>
      <View style={homeStyles.todosContainer}>
        {[1, 2, 3].map((todoId) => (
          <Button
            key={todoId}
            title={'Afficher le TODO n°' + todoId}
            onPress={() =>
              props.navigation.push('TodoDetail', { todoId: todoId })
            }
          />
        ))}
      </View>
    </View>
  );
}

// Les styles liés au composant HomeScreen
const homeStyles = StyleSheet.create({
  container: { flex: 1, alignItems: 'center', justifyContent: 'center' },
  todosContainer: {
    height: 200,
    justifyContent: 'space-around',
    padding: 20,
    backgroundColor: 'lightblue',
    borderRadius: 10,
```

```
  },
  title: { fontSize: 24, fontWeight: 'bold', marginBottom: 10 },
});

function TodoDetailScreen() {
  const navigation = useNavigation();
  const route = useRoute();
  const [todo, setTodo] = React.useState(null);

  // Récupération de la donnée du TODO via son API.
  React.useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/todos/' + route.params.todoId)
      .then((response) => response.json())
      .then((json) => setTodo(json));
  }, [route.params.todoId]);

  // Tant que l'API asynchrone n'a pas répondue, on affiche un loader, sinon on affiche le TODO
  return (
    <View style={todoStyles.card}>
      {!todo ? (
        <ActivityIndicator />
      ) : (
        <React.Fragment>
          <Text style={todoStyles.title}>{todo.title}</Text>
          <Text style={todoStyles.status}>{todo.completed ? '✅' : '❌'}</Text>
        </React.Fragment>
      )}
    </View>
  );
}

// Les styles pour le TODO element
const todoStyles = StyleSheet.create({
  card: {
    margin: 20,
    padding: 20,
    backgroundColor: 'lightblue',
    borderRadius: 10,
    flexDirection: 'row',
    alignItems: 'center',
  },
  title: {
    fontSize: 24,
    fontWeight: 'bold',
    flex: 1,
  },
  status: {},
});

// PROFILE TAB
function ProfileScreen() {
  const navigation = useNavigation();
  return (
    <View>
      <Text>Andréas</Text>
      <Text>27 ans</Text>
      <Button
```

```
        title="Voir plus de détails"
        onPress={() => navigation.navigate('ProfileDetailsModal')}
      />
    </View>
  );
}

// La modale detail du profile
function ProfileDetailsModalScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text style={{ fontSize: 30 }}>Je suis le créateur de ce cours</Text>
      <Button onPress={() => navigation.goBack()} title="Fermer" />
    </View>
  );
}

// Navigators
const MainBottomNav = createBottomTabNavigator();
const RootStack = createStackNavigator();

// Notre navigateur de bas de page
function MainStackScreen() {
  return (
    <MainBottomNav.Navigator>
      <MainBottomNav.Screen name="Todo" component={TodoStackScreen} />
      <MainBottomNav.Screen name="Profile" component={ProfileScreen} />
    </MainBottomNav.Navigator>
  );
}

// Notre navigateur principal avec les modales
function RootStackScreen() {
  return (
    <RootStack.Navigator mode="modal">
      <RootStack.Screen
        name="Main"
        component={MainStackScreen}
        options={{ headerShown: false }}
      />
      <RootStack.Screen
        name="ProfileDetailsModal"
        options={{ title: "Profil détaillé" }}
        component={ProfileDetailsModalScreen}
      />
    </RootStack.Navigator>
  );
}

// Notre application
export default function App() {
  return (
    <NavigationContainer>
      <RootStackScreen />
    </NavigationContainer>
  );
}
```

Nous ne nous attarderons pas ici à décrire les éléments nécessaires à l'affichage de la liste des TODO et de leurs détails, car leur explication complète peut être retrouvée en correction de l'exercice défi du cours sur les bases de la navigation avec `react-navigation`.

Afin d'obtenir le schéma de navigation demandé, il va être nécessaire d'encapsuler différents composants navigateurs.

En effet, un premier navigateur de type `StackNavigator`, `<RootStack>`, servira de navigateur parent et devra contenir un navigateur de type `BottomTabNavigator`, `<MainStackScreen>`, ainsi qu'un écran pour l'affichage de la modale, `<ProfileDetailsModalScreen>`:

```
function ProfileDetailsModalScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text style={{ fontSize: 30 }}>Je suis le créateur de ce cours</Text>
      <Button onPress={() => navigation.goBack()} title="Fermer" />
    </View>
  );
}

const MainBottomNav = createBottomTabNavigator();
const RootStack = createStackNavigator();

function MainStackScreen() {
  return (
    <MainBottomNav.Navigator>
      <MainBottomNav.Screen name="Todo" component={TodoStackScreen} />
      <MainBottomNav.Screen name="Profile" component={ProfileScreen} />
    </MainBottomNav.Navigator>
  );
}

function RootStackScreen() {
  return (
    <RootStack.Navigator mode="modal">
      <RootStack.Screen
        name="Main"
        component={MainStackScreen}
        options={{ headerShown: false }}
      />
      <RootStack.Screen
        name="ProfileDetailsModal"
        options={{ title: "Profil détaillé" }}
        component={ProfileDetailsModalScreen}
      />
    </RootStack.Navigator>
  );
}

export default function App() {
  return (
    <NavigationContainer>
      <RootStackScreen />
    </NavigationContainer>
  );
}
```

Le composant `<MainStackScreen>`, quant à lui, contient deux écrans, Todo et Profile. Le premier est associé à un navigateur de type `StackNavigator`, `<TodoStackScreen>`, qui contient les écrans associés à l'affichage des différents TODO. Le second écran correspond au composant `<ProfileScreen>` qui affiche les informations du profil, ainsi qu'un bouton de navigation vers la modale des détails.

```
const TodoStack = createStackNavigator();

// Le navigateur en pile du menu TODO
function TodoStackScreen() {
  return (
    <TodoStack.Navigator>
      <TodoStack.Screen
        name="TodoHome"
        options={{ headerShown: false }}
        component={TodoScreen}
      />
      <TodoStack.Screen
        name="TodoDetail"
        options={(props) => ({ title: 'TODO n°' + props.route.params.todoId })}
        component={TodoDetailScreen}
      />
    </TodoStack.Navigator>
  );
}

function ProfileScreen() {
  const navigation = useNavigation();
  return (
    <View>
      <Text>Andréas</Text>
      <Text>27 ans</Text>
      <Button
        title="Voir plus de détails"
        onPress={() => navigation.navigate('ProfileDetailsModal')}
      />
    </View>
  );
}
```