# React Native
# Chapitre 1 - Module 2
# Créer une application Android avec React Native

## Déroulé et objectif

Cette formation est découpée en 2 chapitres et 3 modules, qui serviront de fil rouge à la formation.

À la fin de ce module, vous saurez :

- Créer un composant personnalisé.
- Structurer votre projet pour ajouter des composants personnalisés.
- Passer des props d'un composant parent à un composant enfant.
- Comprendre comment fonctionne le composant StyleSheet.
- Utiliser les vues pour découper votre application.
- Importer de nouveaux composants cores.
- Découper les différentes pages de votre application en "screens".
- Créer un APK pour exporter l'application sur un téléphone Android.

Objectif du module 2 : Créer la première version de l'application Calculateur d'IMC.

Objectif final du chapitre 1 : Créer une application calculateur d'IMC.

## Création d'un module personnalisé

Pour notre application, nous allons créer un premier composant personnalisé, qui nous permettra d'afficher le texte d'accueil de notre application et nous remplacerons le composant **<Text></Text>** par défaut par ce composant.

Pour commencer, créez le dossier **components** à la racine de votre application.

Ce dossier contiendra l'ensemble des **composants natifs** que nous allons créer pour cette application.

Dans ce dossier, créez le fichier **WelcomeText.js**.

Le composant **<WelcomeText/>** sera créé sous la forme d'une fonction et sera exportée par défaut.

Dans ce fichier, ajouter la ligne d'import des composants **View**, **Text** et **StyleSheet** de "**react-native**".

```
import {View, Text, Stylesheet } from 'react-native';
```

Créez la fonction fléchée **WelcomeText**, qui sera stockée dans une constante et qui prendra un paramètre, nommé (**props**).

```
const WelcomeText = (props) => {

}
```

Pensez à ajouter la ligne d'export par défaut de la fonction **WelcomeText**, à la fin de votre fichier.

```
export default WelcomeText;
```

Dans votre fonction **WelcomeText**, vous devez retourner votre **composant**.

Votre composant doit répondre à la syntaxe **JSX** pour être exportée correctement.

Notre composant **WelcomeText** nous permettra d'afficher le message de bienvenue sur l'écran d'accueil de notre application, nous allons donc utiliser le composant core **<Text></Text>** pour englober notre message et ce composant Text sera englobé par le composant core **<View></View>**, pour lui définir une vue spécifique.

Le code de votre composant qui doit être retourné par la fonction **WelcomeText** doit être à l'intérieur de la fonction **return()**.

Pour l'instant, vous devriez avoir ce résultat :

```
const WelcomeText = (props) => {
    return(
        <View>
            <Text>

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

Le paramètre **props** de notre fonction n'est pas un paramètre donné au hasard, il nous permet de récupérer les **paramètres** passés au composant lorsqu'il est appelé et de les exploiter.

Pour personnaliser le texte affiché par notre composant, passé dans le paramètre **text** lors de son utilisation, nous utiliserons donc **props.text**.

À l'intérieur du composant **<Text></Text>**, vous pouvez donc ajouter ce paramètre, en respectant la syntaxe **JSX**.

```
<Text>
   {props.text}
</Text>
```

Ajoutons un peu de style à nos composants.
Pour le composant View, passez lui la prop **style**, en lui donnant pour valeur **{styles.view}**.

```
 <View style={styles.view}>
```

Pour le composant Text, passez lui la prop **style**, en lui donnant pour valeur **{styles.text}**.

```
 <Text style={styles.text}>
```

Créez la constante **styles** et donnez lui une feuille de style en valeur avec le composant **StyleSheet** et sa fonction **.create()**.

```
 const styles = StyleSheet.create(

 )
```

La fonction **create** du composant StyleSheet vous permet de créer une feuille de style en lui passant un **objet**.
La syntaxe de cet objet ressemble à du **CSS**, mais c'est un élément trompeur.

À l'intérieur de votre feuille de style, créez un objet spécifiant les paramètres suivants pour **view** (styles.view) et pour **text** (styles.text).

```
const styles = StyleSheet.create({
    view:{
           width: "100%",
           backgroundColor: "#333",
           color: "#fff",
           padding: 20,
           marginVertical: 8,
           marginHorizontal: 16,
         },
    text:{
           fontSize: 24,
           color: "#fff",
         }
})
```

## Création d'un index des composants

Notre premier **composant natif** est prêt à être utilisé.
Avant de l'importer dans notre fichier **App.js** pour le tester, nous allons créer le fichier **index.js**, dans le dossier **components**.

Ce fichier contiendra l'import et l'export des modules contenus dans le dossier **components** et permettra de spécifier le chemin vers le dossier **components**, dans les imports, ce qui pourrait simplifier les imports à long terme (complexification de la structure de dossiers).
Ce fichier vous simplifiera également la vérification des composants créés pour l'application.

Dans le fichier **index.js**, importez votre module **WelcomeText**.

```
import WelcomeText from './WelcomeText';
```

Ajoutez une liste d'export, en spécifiant que vous souhaitez exporter le module **WelcomeText**.

```
export {
    WelcomeText,
}
```

À chaque composant créé, il vous suffira de l'importer dans ce fichier et d'inclure le **composant** dans la liste des exports.

Ce fichier vous permettra également d'avoir une visibilité rapide sur les modules créés pour votre application.

## Test de notre premier composant personnalisé

Ouvrez le fichier **App.js**, puis ajoutez la ligne d'import du composant **WelcomeText**, en ciblant le chemin du dossier **components**.

```
import { WelcomeText } from './components';
```

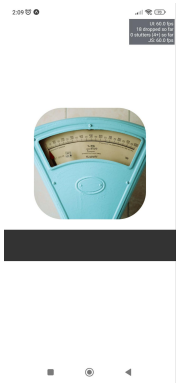
*Si nous n'avions pas créé le fichier **index.js**, le composant aurait dû être importé de cette manière :*

```
import WelcomeText from './components/WelcomeText.js';
```

*Pour les besoins de l'application Calculateur d'IMC, nous aurons besoin de créer plusieurs composants personnalisés.*
*Avec le fichier **index.js**, l'import des nouveaux composants pourra se faire sur une seule ligne. Exemple :*

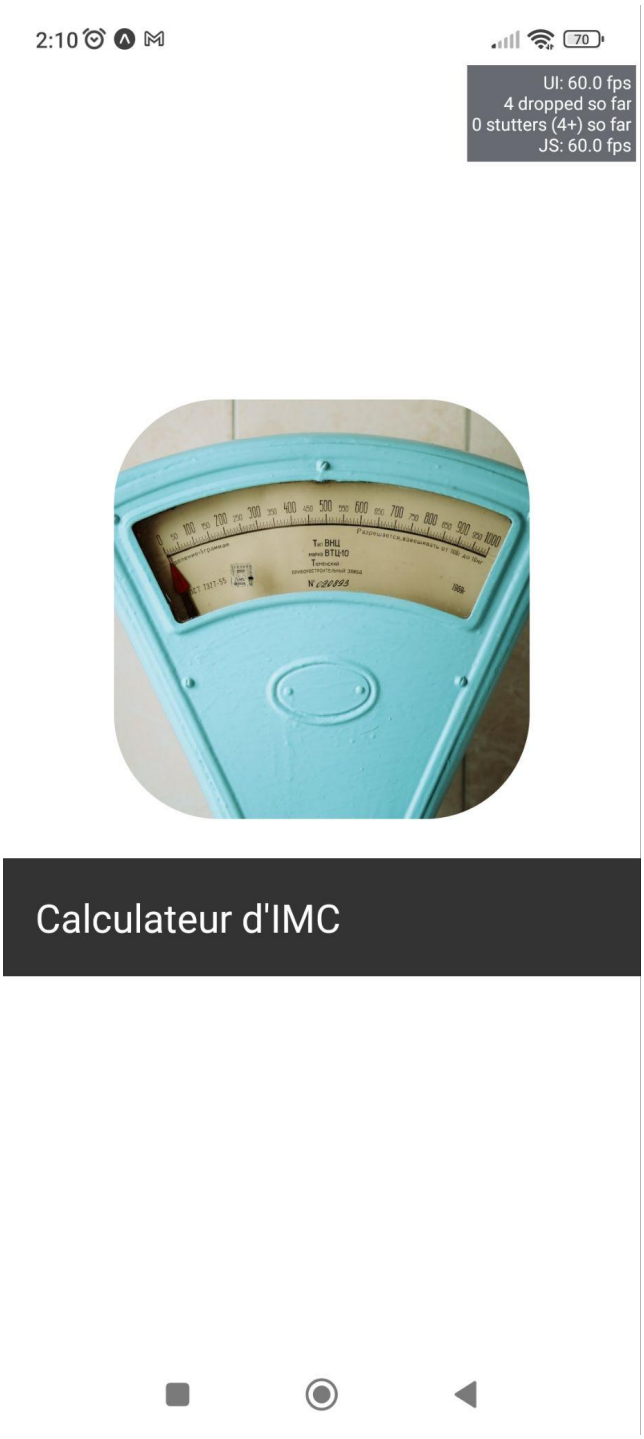
```
import { WelcomeText, Logo, Navigation, Calculator } from
'./components';
```

Vous pouvez désormais remplacer le composant **<Text></Text>** par le composant <**WelcomeText/>**.
Commencez par l'ajouter au fichier **App.js**, sans lui spécifier de **props** et ouvrez votre application dans **Expo Go**.



Votre **composant** est bien importé et affiché à l'écran, mais comme la prop **text** n'a pas été spécifié, vous avez juste l'aperçu de votre **View**.

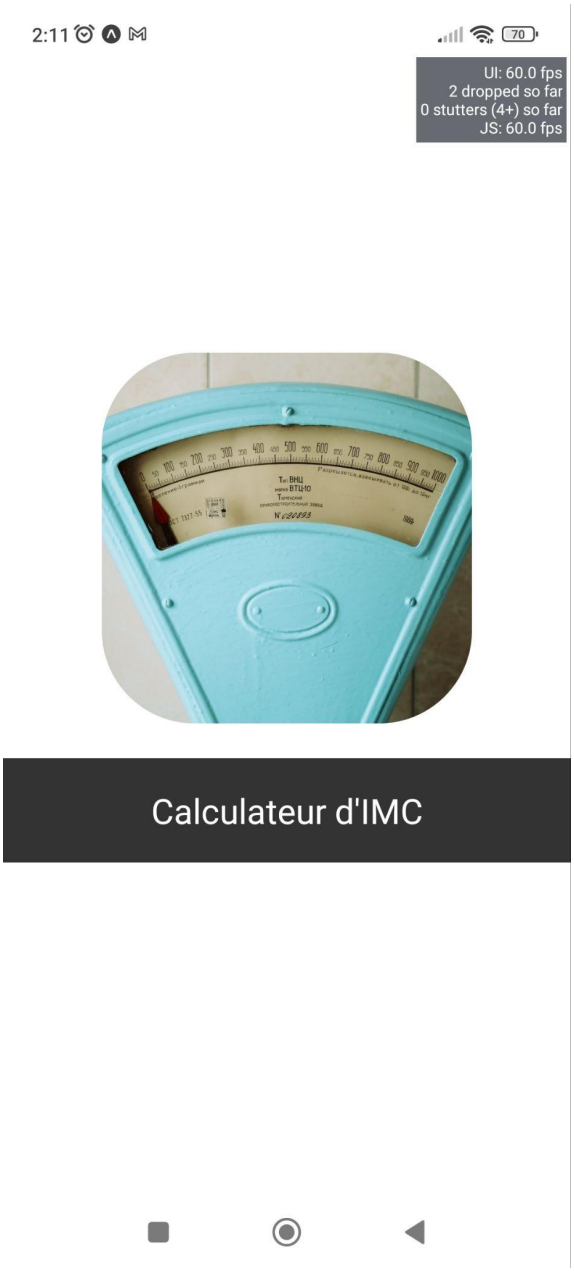
Spécifiez la prop **text** de votre composant et donnez lui la valeur suivante : **Calculateur d'IMC**.



Le texte passé en paramètre s'affiche désormais sur votre application.
Ayant pour but d'être le texte d'accueil de l'application, nous allons le centrer son texte, pour le rendre plus harmonieux.

Retournez dans le fichier **WelcomeText.js** et ajoutez à styles (pour le style de **text**) :

```
textAlign: "center",
```



## Création d'un composant de navigation personnalisé

Nous allons créer un composant de navigation, qui permettra de naviguer entre les différents écrans (**screens**) de l'application.

Pour créer cette navigation, nous allons utiliser 2 dépendances supplémentaires, qu'il faudra installer sur la projet.

Ouvrez votre terminal de commande et tapez la commande suivante :

```
npm install @react-navigation/native@latest
```

Cette dépendance nous permettra de récupérer le composant **{ NavigationContainer }**, depuis "**@react-navigation/native**".

Entrez également cette commande dans votre terminal :

```
npm install @react-navigation/stack@latest
@react-native-community/masked-view@latest
react-native-screens@latest
react-native-safe-area-context@4.5.0
react-native-gesture-handler@latest
```

Ces dépendances nous apporteront des éléments utilisables pour la création et la gestion de nos **screens**, ainsi que des éléments concernant la gestion de mouvements à l'écran (**gesture handler**).

Sur votre fichier **App.js**, ajoutez les imports suivants :

```
 import 'react-native-gesture-handler';
```

```
 import { NavigationContainer } from
'@react-navigation/native';
```

Attention à cette dernière ligne d'import à ajouter, elle doit importer une fonction, pas un composant :

```
 import { createStackNavigation } from
'@react-navigation/stack';
```

Initialisez directement une constante **Stack**, contenant l'appel à la fonction **createStackNavigator()**, entre les imports et la fonction **App**.

La constante **Stack** nous permettra d'implémenter un composant **<Stack/>** et d'utiliser ses paramètres.
Dans un premier temps, dans **App.js**, nous utiliserons **<Stack.Navigator>**, pour créer un navigateur nous permettant de naviguer entre nos pages.
À l'intérieur du **<Stack.Navigator></Stack.Navigator>**, nous pouvons implémenter des composants **<Stack.Screen/>**, ayant pour props **name** et **component**, ce qui nous permettra d'implémenter la liste des écrans disponibles dans notre application (écrans que nous allons créer et stocker dans un dossier **screens**).

Vous pouvez consulter la documentation de **React Navigation** pour aiguiller votre développement : https://reactnavigation.org/docs/getting-started.

Ces éléments vont nous permettre de mettre en place un système de navigation, ayant une logique similaire à l'historique de navigation d'un navigateur web.
Lorsqu'une page est appelée, l'ancienne page se place dans la pile et peut être retournée suite à l'appui sur le bouton de retour ou un lien de retour.

À l'implémentation, ces composants vont nous permettre d'avoir une barre de navigation, en haut de l'écran de notre application.

Commençons par créer le dossier **screens**, à la racine de notre projet, qui contiendra nos différents écrans.

Dans le dossier **screens**, créez le fichier **HomeScreen.js**.
Ce fichier contiendra l'écran d'accueil de notre application.

Dans **HomeScreen.js**, ajoutez les lignes d'import suivantes :

```
import React from 'react';
import { StyleSheet, View, Image } from 'react-native';
import { WelcomeText } from '../components';
```

Comme vous pouvez le voir, il faut modifier le chemin d'import ciblant le dossier **components**, car la recherche s'effectue depuis le dossier **screens**, il faut donc remonter d'un niveau pour accéder au dossier **components** qui se situe à la racine de l'application.

Créez une classe **HomeScreen**, étendant de **React.Component** et sa fonction de rendu, puis ajoutez sa ligne d'export en bas du fichier :

```
class HomeScreen extends React.Component {
    render() {
     return (

     )
   }
 }
 export default HomeScreen;
```

À l'intérieur du rendu de notre classe **HomeScreen**, nous souhaitons renvoyer une **View**, contenant notre **Image** d'accueil et notre composant natif **WelcomeText** (déplacez le code depuis **App.js**):

```
<View style={styles.container}>
   <Image
       style={styles.homeImg}
       source={{ uri: [uri de l'image], }}
   />
   <WelcomeText text="Calculateur d'IMC" />
</View>
```

Créez votre constante **styles**, en lui apportant les paramètres graphiques liés à **styles.container** et **styles.homeImg** :

```
const styles = StyleSheet.create({
   container: {
      flex: 1,
      backgroundColor: '#fff',
      alignItems: 'center',
      justifyContent: 'center',
   },
   homeImg: {
      width: 256,
      height: 256,
      borderRadius: 64,
      margin: 16,
   },
})
```

Pensez à implémenter le paramètre "**flex : 1**,", si vous souhaitez que votre écran prenne toute la largeur et la hauteur de votre écran.

Créez le fichier **index.js**, dans le dossier **screens**, pour faciliter l'import (comme pour le dossier **components**) :

```
import HomeScreen from './HomeScreen';

export {
   HomeScreen,
}
```

Nous allons créer à l'avance un deuxième écran, nommé **CalculatorScreen.js**, qui affichera notre calculateur d'IMC quand il sera prêt.

Pour l'instant, nous n'allons lui fournir qu'un composant **<Text></Text>**, qui nous permettra de vérifier si notre implémentation de la navigation fonctionne correctement.

Ajoutez les lignes d'import suivantes à votre fichier **CalculatorScreen.js** :

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

Pensez à importer également le composant **Button**, que nous utiliserons dans ce fichier pour créer un bouton de navigation supplémentaire, que l'on affichera sous notre texte de test.

Créez la classe **CalculatorScreen**, étendant de **React.Component**, et sa fonction de rendu, puis ajoutez sa ligne d'export :

```
class CalculatorScreen extends React.Component {
     render() {
      return (

          );
        }
}
export default CalculatorScreen;
```

Cet écran retournera une **View**, qui contiendra un **Text** de test et un **Button** ayant pour props **title** et **onPress** :

```
<View style={styles.container}>
     <Text>Screen : Calculator Screen.</Text>
      <Button
         title="Retour à l'accueil"
         onPress={()=>{ this.props.navigation.navigate("Home") }}
       />
</View>
```

Le composant **Button** est un composant que nous n'avons pas encore vu.
Il permet de créer des boutons.
Ici, la prop **title** nous permet d'ajouter un texte qui sera affiché à l'intérieur du bouton et la prop **onPress** nous permet de spécifier une action qui sera exécutée au clic sur ce bouton.

La fonction anonyme suivante, passée à la prop **onPress**, nous permet de spécifier une action de **navigation** à l'application :

```
()=>{
    this.props.navigation.navigate("Home")
}
```

- **this** : fait référence au contexte actuel, ici le bouton.
- **props** : fait référence aux props passées au composant **<Button>**.
- **navigation** : fait référence à la prop **navigation**, passée explicitement au composant **Button**.
- **navigate("Home")** : appelle la fonction **navigate()** et nous permet de spécifier la page qui doit être appelée (ici **Home**).

Nous spécifions ici une redirection sur la page "**Home**", mais si vous testez l'application sur **Expo Go**, vous obtiendrez une erreur, car nous n'avons pas encore configuré notre navigation dans le fichier **App.js**.

Ajoutons l'export de **CalculatorScreen** au fichier **index.js** contenu dans le dossier **screens**, avant d'implémenter notre navigation dans **App.js**.

```
import HomeScreen from './HomeScreen';
import CalculatorScreen from './CalculatorScreen';

export {
   HomeScreen,
   CalculatorScreen,
}
```

Ouvrez le fichier **App.js** pour apporter les modifications nécessaires.
Nous avons importé de nouveaux composants dans **App.js**, mais nous ne les avons pas encore implémentés.
Commencez par supprimer la ligne d'import au dossier **components**, qui importait le composant **WelcomeText**.
Remplacez cette ligne par l'import des **screens** :

```
import { HomeScreen, CalculatorScreen } from './screens';
```

Si vous avez terminé le transfert de vos composants d'accueil dans le screen **HomeScreen**, supprimez les composants **View**, **Image** et **WelcomeText** de votre fichier **App.js**.

Pour l'instant, supprimez votre constante **styles**, elle ne sera pas implémentée sur les éléments de **navigation**.
Supprimez la ligne d'import de '**react-native**', car nous n'utilisons plus ses composants pour le moment.
C'est une bonne pratique de nettoyer son code et de supprimer les imports inutiles, qui peuvent entraîner des baisses de performances.

Dans le **return()** de notre fonction **App**, nous allons désormais implémenter les composants de navigation
**<NavigationContainer></NavigationContainer>**,
**<Stack.Navigator></Stack.Navigator>** et nos **<Stack.Screen />**.

**NavigationContainer** va créer un container qui accueillera la navigation et sera en charge de la mise en lien entre vos liens et votre application.
Il notifie les changements d'état pour le suivi d'écran, la persistance d'états, etc… Permet de gérer le bouton de retour natif du système Android.

**Stack.Navigator** permet à l'application de passer d'écran en écran, en plaçant les **screens** dans une **pile**. Il contiendra nos **Stack.Screen**.

**Stack.Screen** nous permettra d'implémenter les écrans de notre application.

Après le nettoyage, vous devez avoir ce code dans le fichier **App.js** :

```
import 'react-native-gesture-handler';
import { StatusBar } from 'expo-status-bar';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
import { HomeScreen, CalculatorScreen } from './screens';

const Stack = createStackNavigator();

export default function App() {
  return (
  );
}
```

Nous allons commencer par implémenter le composant **NavigatorContainer** :

```
return (
    <NavigationContainer>

    </NavigationContainer>
  );
```

Dans le **NavigatorContainer**, nous allons implémenter le composant **Stack.Navigator** :

```
<Stack.Navigator>

</Stack.Navigator>
```

Dans le **StackNavigator**, nous allons implémenter nos deux **screens**, avec le composant **<Stack.Screen>**, qui prendra 2 props en paramètres (**title** et **component**) :

```
<Stack.Screen name="Home" component={HomeScreen} />
<Stack.Screen name="Calculator" component={CalculatorScreen} />
```

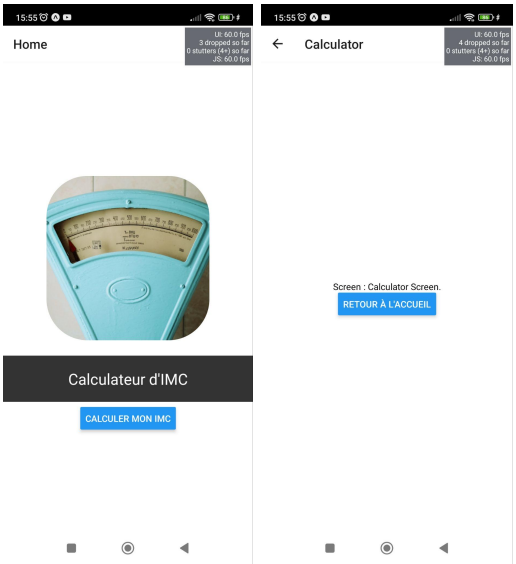
Le prop **title** doit contenir le nom de votre écran.
Attention, sa valeur est utilisée pour définir le lien vers cet écran et elle est sensible à la casse.

Le prop **component** doit contenir le **composant** qui fait office de **screen**.
Ici, nous avons créé au préalable 2 screens, qui sont **HomeScreen** et **CalculatorScreen**.

Notre **pile** (Stack.Navigator) contient désormais les **screens** suivants : **HomeScreen** (lié au lien **Home**) et **CalculatorScreen** (lié au lien **Calculator**).

Si vous essayez d'inverser l'ordre des **Stack.Screen** et que vous forcez le rechargement de l'application en appuyant sur la touche **r**, dans votre terminal de commandes, vous pourrez voir que le **screen** ouvert en priorité sera **CalculatorScreen**.
Cela démontre que l'ordre de la pile à une importance pour le composant **Stack.Navigator**.

Replacez **HomeScreen** en premier dans le **Stack.Navigator** et forcez le rechargement de l'application.



Vous aurez désormais accès aux screens **HomeScreen** et **CalculatorScreen**, grâce aux composants **Button** de la page.

Quand vous arrivez sur **CalculatorScreen**, vous pouvez voir que la prop **name** est affichée en haut de l'écran, dans la barre de navigation et une flèche de retour est disponible à cet endroit également.

Si vous cliquez sur la flèche de retour, vous serez redirigé vers le **screen** ayant la position précédente dans la pile de **Stack.Navigator**.

Pour voir plus en détail le comportement de cette barre de navigation, nous allons créer un troisième **screen**, qui se nommera **TestScreen**.

Pour créer cette page de test, nous allons juste copier/coller le contenu du fichier **CalculatorScreen.js**, en remplaçant :

- Le nom de la classe, par **TestScreen**.
- L'export de la classe, pour exporter **TestScreen**.
- Le contenu du composant **Text**, pour indiquer que c'est l'écran **TestScreen** qui s'affiche.

Nous allons également implémenter un composant **Button** supplémentaire qui redirigera sur **CalculatorScreen** :

```
<Button
    title="CalculatorScreen"
    onPress={() => {
        this.props.navigation.navigate("Calculator")
    }}
/>
```

Le prop **title** de votre bouton peut être un texte libre, mais la valeur passée dans **.navigate()** doit bien correspondre au lien vers le **screen** ciblé.

Actualisez votre fichier **index.js** (**screens > index.js**) pour exporter ce nouvel écran :

```
import HomeScreen from './HomeScreen';
import CalculatorScreen from './CalculatorScreen';
import TestScreen from './TestScreen';
export {
   HomeScreen,
   CalculatorScreen,
   TestScreen,
}
```

Ajoutez l'import de votre nouvel écran sur le fichier **App.js** :

```
import { HomeScreen, CalculatorScreen, TestScreen } from './screens';
```

Et implémentez votre nouveau **Stack.Screen**, en bas de la liste :

```
<Stack.Screen name="Home" component={HomeScreen} />
<Stack.Screen name="Calculator" component={CalculatorScreen} />
<Stack.Screen name="TestScreen" component={TestScreen} />
```

Ouvrez désormais **Expo Go**.
Sur l'écran d'accueil, cliquez sur le bouton "**CALCULER MON IMC**" pour être redirigé sur l'écran **CalculatorScreen**, puis cliquez sur le bouton "**TESTSCREEN**" pour être redirigé sur votre nouvel écran.

Vous verrez alors vos deux composants **Button**, qui vous permettent d'être redirigé vers **HomeScreen** ou **CalculatorScreen**.
Testez l'implémentation de vos **Button** en cliquant dessus et naviguez à nouveau vers **TestScreen**.

Si vous utilisez la flèche de retour sur **TestScreen**, vous serez redirigés sur **CalculatorScreen**.
Si vous cliquez sur la flèche de retour sur **CalculatorScreen**, vous serez redirigés sur **HomeScreen**.

Mais que se passe-t-il si vous naviguez sur **TestScreen**, puis accédez à **CalculatorScreen** via le bouton "**CALCULATORSCREEN**" et cliquez sur la flèche de retour ?

Vous serez bien redirigé sur **HomeScreen**, car l'ordre des **screens** implémentés dans le **Stack.Navigator** est le suivant :

1. **HomeScreen**
2. **CalculatorScreen**
3. **TestScreen**

Cet ordre nous impose donc d'utiliser des éléments complémentaires, sur les écrans de notre application, pour rediriger plus loin que l'élément **n-1** de la **pile**.

## Implémentation d'une TapBar

Cette navigation est pratique, mais elle reste basique.
Nous aimerions créer un système de navigation sous forme d'onglets qui s'affiche en bas de notre écran, pour pouvoir étendre les possibilités de navigation.

Le composant qui pourrait nous permettre d'implémenter ce type de navigation est géré par '**react-navigation/bottom-tabs**', mais il n'est pas encore installé sur notre projet.

Entrez la ligne suivante dans votre terminal de commande :

```
npm install @react-navigation/bottom-tabs
```

Pour pouvoir utiliser la navigation que nous avons implémenté précédemment et les **Tabs** conjointement, nous allons devoir créer un composant personnalisé qui gèrera uniquement le rendu du **Tab.Navigator** et des **Tabs.Screen**.

Ce composant devra être implémenté dans un **Stack.Screen**, avec le prop **options={{ headerShown:false, }}**.

Créez le fichier **NavigationTabs.js**, dans le dossier **components**.

Dans votre fichier **NavigationTabs.js**, importez la fonction **createBottomTabNavigator** depuis '**@react-navigation/bottom-tabs**' :

```
import { createBottomTabNavigator } from
'@react-navigation/bottom-tabs';
```

Importez vos **screens** :

```
import { HomeScreen, CalculatorScreen, TestScreen } from '../screens';
```

Créez la constante **Tab**, qui contiendra l'appel à la fonction **createBottomTabNavigator()** :

```
const Tab = createBottomTabNavigator();
```

Créez le composant **NavigationTabs** en respectant la syntaxe que nous avons étudié sur la section **"Création d'un module personnalisé"** :

```
const NavigationTabs = () => {
   return (
    )
}
export default NavigationTabs;
```

Ce composant devra retourner :

- Un composant **<Tab.Navigator></Tab.Navigator>** contenant :
    - 3 **<Tab.Screen/>** ayant pour props **name** et **component**.

```
const NavigationTabs = () => {
   return (
      <Tab.Navigator>
        <Tab.Screen name="Home" component={HomeScreen} />
        <Tab.Screen name="Calculator" component={CalculatorScreen} />
        <Tab.Screen name="TestScreen" component={TestScreen} />
      </Tab.Navigator>
    )
}
```

Pensez à éditer le fichier **index.js** de votre dossier **components** :

```
import WelcomeText from './WelcomeText';
import NavigationTabs from './NavigationTabs';

export {
   WelcomeText,
   NavigationTabs,
}
```

Retournez désormais dans le fichier **App.js**, puis importez le composant **NavigationTabs** :

```
import { NavigationTabs } from './components';
```

Vous pouvez désormais ajouter un nouveau **Stack.Screen**, qui fera appel à votre composant **NavigationTabs** :

```
<Stack.Screen name="NavigationTabs" component={NavigationTabs}
options={{ headerShown: false, }}/>
```

Le prop **options** permet de passer des options complémentaires à **Stack.Screen**.
Ici, le paramètre **headerShown** permet d'indiquer si la barre de navigation en haut de l'écran doit être affichée, il faut donc la désactiver pour ne pas avoir de doublons.

Si vous ouvrez votre application, vous devez avoir un message **Warning** qui s'affiche en bas de **Expo Go** et dans le terminal :

```
 WARN  Require cycle: screens\index.js ->
screens\NavigationTabs.js -> screens\index.js
```

Nous avons donc un bon exemple de message d'avertissement concernant notre application.

Pour corriger ce message d'avertissement, il vous suffit de retourner dans **NavigationTabs.js** et de modifier les imports des écrans :

```
import { HomeScreen, CalculatorScreen, TestScreen } from '../screens';
// Doit être transformé en :
import HomeScreen from './HomeScreen';
import CalculatorScreen from './CalculatorScreen';
import TestScreen from './TestScreen';
```

Le message d'avertissement a été généré, car lors de l'implémentation, nous avons importé le fichier "**index.js**", dans **NavigationTabs.js** mais **index.js** importait également **NavigationTabs.js** pour le servir aux autres fichiers.

Ce comportement a donc créé une boucle d'importation, qui pourrait causer des conflits et des baisses de performances.

Il faut donc penser à implémenter un à un les écrans ou composants contenus dans le même dossier.

Pour l'instant, notre **TapBar** n'affiche pas d'icônes.
Nous allons importer le composant **Foundation**, dépendant de '**react-native-vector-icons**' pour implémenter des icônes sur nos **Tabs.Screen**, dans le prop **options**.

```
 npm install react-native-vector-icons
```

```
import { Foundation as FoundationIcons } from
'react-native-vector-icons';
```

Pour l'import des icônes dans **NavigationTabs.js**, j'ai tenu à ajouter un **Alias**, qui nous permettra de visualiser plus facilement le composant dédié à l'ajout des icônes.

Vous pouvez consulter la liste des icônes disponibles sur la documentation de Foundation : https://zurb.com/playground/foundation-icon-fonts-3 et la liste des autres librairies utilisables sur https://github.com/oblador/react-native-vector-icons#bundled-icon-sets.

Nous allons désormais modifier notre **Tab.Screen** contenant le lien **Home**, pour lui ajouter une icône et un label personnalisé, grâce au prop **options** :

```
<Tab.Screen name="Home" component={HomeScreen} options={{
          tabBarLabel: 'Accueil',
          tabBarIcon: ({ color, size }) => (
            <FoundationIcons name="home" color={color} size={size} />
          ),
        }}/>
```

Ajoutez également des options au **Tab.Screen** nommé **Calculator** :

```
<Tab.Screen name="Calculator" component={CalculatorScreen} options={{
          tabBarLabel: 'Mon IMC',
          tabBarIcon: ({ color, size }) => (
            <FoundationIcons name="pencil" color={color} size={size} />
          ),
        }} />
```

Et à **ScreenTest** :

```
<Tab.Screen name="TestScreen" component={TestScreen} options={{
          tabBarLabel: 'TEST',
          tabBarIcon: ({ color, size }) => (
            <FoundationIcons name="widget" color={color} size={size} />
          ),
           tabBarBadge: "test",
        }} />
```

**tabBarBadge** permet d'ajouter un badge personnalisé sur l'élément.

Pour s'assurer que notre navigation avec le bouton de retour natif du téléphone utilise bien l'ordre de notre pile, nous allons lui passer le prop **backBehavior**, en lui donnant la valeur **order** :

```
<Tab.Navigator backBehavior='order'>
```

Avant de quitter le fichier **NavigationTabs.js**, ajoutez le paramètre **title**, dans le prop **options** de vos **Tab.Screen**, pour leur fournir un nom plus cohérent :

```
<Tab.Screen name="Home" component={HomeScreen} options={{
          tabBarLabel: 'Accueil',
          tabBarIcon: ({ color, size }) => (
            <FoundationIcons name="home" color={color} size={size} />
          ),
          title: "Accueil",
        }}/>
<Tab.Screen name="Calculator" component={CalculatorScreen} options={{
          tabBarLabel: 'Mon IMC',
          tabBarIcon: ({ color, size }) => (
            <FoundationIcons name="pencil" color={color} size={size}
/>

          ),
          title: "Calculer mon IMC",
        }} />
<Tab.Screen name="TestScreen" component={TestScreen} options={{
          tabBarLabel: 'TEST',
          tabBarIcon: ({ color, size }) => (
            <FoundationIcons name="widget" color={color} size={size}
/>

          ),
          title: "Page de test",
          tabBarBadge: "test",
        }} />
```

## Création du calculateur d'IMC

Nous avons à présent une navigation fonctionnelle, nous allons désormais nous pencher sur la fonctionnalité principale de notre application, le **calculateur d'IMC**.

Rendez-vous sur le fichier **CalculatorScreen.js**, puis supprimez les deux composants **Button**.

Avant d'aller plus loin, nous allons installer la librairie **React Native Elements**, qui nous facilitera la mise en place de nos éléments graphiques, car ils auront un style prédéfini.

Installez la dépendance suivante :

```
npm install @rneui/base @rneui/themed
```

Nous allons créer un composant, qui se nommera **ImcCalculator.js** et qui se trouvera dans notre dossier **components**.

Dans **ImcCalculator.js**, importez le composant **Input** depuis '**@rneui/base**' :

```
import { Input as ElementsInput } from '@rneui/base';
```

Pensez à lui donner un alias, pour le repérer plus facilement dans votre code et éviter les éventuels conflits avec les autres dépendances.

- Ajoutez les lignes d'import suivantes :

```
import React, { useState } from 'react';
import { RefreshControl, SafeAreaView, ScrollView, Text, StyleSheet,
TouchableOpacity } from 'react-native';
import { Input as ElementsInput } from '@rneui/base';
```

Sur ce composant, nous utiliserons les **useState** de React pour stocker et modifier dynamiquement les valeurs qui seront fournies par l'utilisateur.

Dans les imports **React Native**, on va ajouter **RefreshControl**, **SafeviewArea**, **ScrollView** et **TouchableOpacity**, qui seront de nouveaux composants utilisés dans notre application :

- **RefreshControl** : Ce composant permet d'activer une fonctionnalité de rafraîchissement de l'écran, en suivant vers le bas.
- **SafeAreaView**: Ce composant permet de condenser l'affichage de la vue dans la zone de sûreté du téléphone (IOS), elle permet de mieux calibrer l'affichage en fonction du placement de la caméra avant, l'arrondi des coins, etc…
- **ScrollView** : Ce composant permet d'intégrer une vue scrollable. Cela évitera de rencontrer des débordements de contenu sur la hauteur de l'écran.
- **TouchableOpacity** : Ce composant permet de mettre en place un **wrapper** pour les éléments pressables, qui verront leur opacité diminuer quand l'utilisateur cliquera dessus (Exemple : clic sur un bouton).

Créez la constante contenant votre composant **ImcCalculator** et ajoutez sa ligne d'export :

```
 const ImcCalculator = () => {
     return (

     )
}

export default ImcCalculator;
```

Commençons par instancier nos **useState**, dans le composant **ImcCalculator** :

```
const [size, setSize] = useState('')
const [weight, setWeight] = useState('')
const [imc, setImc] = useState(0)
const [result, setResult] = useState('')
```

- **size** contiendra la taille fournie par l'utilisateur.
- **weight** contiendra le poids fourni par l'utilisateur.
- **imc** contiendra le score IMC de l'utilisateur (après calcul de notre future méthode).
- **result** contiendra la phrase de résultat, associée au score IMC.

```
const [refreshing, setRefreshing] = React.useState(false);

   const onRefresh = React.useCallback(() => {
      setRefreshing(true);
      setTimeout(() => {
         setRefreshing(false);
      }, 2000);
   }, []);
```

Ce **useState** et cette méthode sont associés au composant **RefreshControl**.

Pour calculer le score IMC de notre utilisateur, nous allons devoir appliquer une formule et exécuter une logique pour retourner le résultat associé.

Savez-vous comment calculer le **score IMC** d'une personne ?
Il vous suffit de connaître sa **taille** et son **poids**, puis d'appliquer la formule suivante :

```
(Number(weight) * 10000) / (Number(size) * Number(size))
```

Par exemple :
(76kg * 10000) / (179cm * 179cm) = **23.7** de **Score IMC**.
**23.7** équivaut à un poids normal (entre **18.5** et **25**).

Nous allons créer la méthode **calculateImc**, qui sera stockée dans une constante et prendre 2 paramètres : **size** et **weight.**

Nous allons commencer par stocker le résultat du calcul du **Score IMC** dans la variable **imc** (en **let**) :

```
const calculateIMC = (size, weight) => {
     // IMC = ((poids * 10000) / (taille * taille))
      let imc = (Number(weight) * 10000) / (Number(size) *
Number(size))
}
```

Pour éviter d'avoir un nombre flottant difficilement compréhensible, nous allons fixer la taille des décimales de la variable **imc** à **1** :

```
imc = Number(imc).toFixed(1)
```

Quand notre méthode aura calculé le score IMC, nous souhaitons que sa valeur soit stockée pour être utilisée dans le reste du code, et qu'elle soit dynamique. Nous allons donc la fournir au **useState**, grâce à la méthode **setImc()** que nous avons défini plus haut :

```
setImc(imc)
```

Pour l'instant, seul le score IMC brut est stocké, nous allons configurer le stockage de la phrase de retour associé, que nous afficherons avec ce score. Contrairement aux idées reçues, la création d'un calculateur d'IMC n'a rien de compliqué si l'on possède la bonne formule de calcul.
Le plus gros du travail de l'application est de calculer le **score IMC** et de retourner un résultat approprié à l'utilisateur.
Pour associer facilement ce score au texte, nous allons utiliser une condition **switch** qui prendra comme paramètre **true** et qui utilisera des instructions **case** conditionnelles (sous cette forme : **case (imc <= 18.5): /*phrase à retourner*/; break;**) :

```
switch (true) {
         case (imc <= 18.5):
            setResult(`Maigreur, consultez un nutritionniste`);
            break;
         case (imc <= 25 && imc > 18.5):
            setResult(`Poids normal`);
            break;
         case (imc <= 30 && imc > 25):
            setResult(`Surpoids`);
            break;
         case (imc <= 35 && imc > 30):
            setResult(`Obésité modérée`);
            break;
         case (imc <= 40 && imc > 35):
            setResult(`Obésité sévère, consultez un nutritionniste`);
            break;
         case (imc > 40):
            setResult(`Obésité morbide, consultez un nutritionniste
rapidement`);
            break;
         default:
            setResult("Les valeurs données sont incorrectes");
            break;
      }
```

Comme vous pouvez le voir, la principale fonctionnalité de notre calculateur d'IMC tient en 6 lignes de retours et une ligne de calcul.

Le principal effort à fournir pour créer une application qui pourrait être publiée sur les plateformes de distribution sera de vous démarquer sur l'apparence et l'accessibilité de votre application.

Dans le **return()** de notre composant, nous allons commencer par implémenter le composant **<SafeAreaView></SafeAreaView>**, qui aura comme prop **style={styles.container}**.

```
<SafeAreaView style={styles.container}>
</SafeAreaView>
```

Dans ce composant, nous allons ajouter un composant **<ScrollView></ScrollView>** qui aura les props suivantes :

```
<ScrollView
     style={styles.scrollContainer}
     refreshControl={<RefreshControl refreshing={refreshing}
onRefresh={onRefresh} />}
     contentContainerStyle={{flexGrow : 1, justifyContent :
'center'}}>
</ScrollView>
```

Ce composant nous permettra d'éviter d'être confronté à un débordement des éléments, sur la hauteur de l'écran.

Nous allons désormais ajouter deux **<ElementsInput/>**, qui permettront à un utilisateur de rentrer sa taille et son poids !

```
<ElementsInput
              rightIcon={{ type: 'foundation', name: 'arrow-up' }}
              onChangeText={setSize}
              value={size}
              placeholder="Votre taille en cm"
              keyboardType="numeric"
           />

<ElementsInput
              rightIcon={{ type: 'foundation', name: 'burst' }}
              onChangeText={setWeight}
              value={weight}
              placeholder="Votre poids en kg"
              keyboardType="numeric"
           />
```

On passera la prop **placeholder** pour afficher un texte définissant l'attente du champ et la prop **rightIcon** nous permet d'utiliser les icônes de différentes librairies (nous resterons sur la librairie **Foundation**).

La prop **onChangeText** permet de déclencher un évènement au changement du texte de l'**ElementInput**.

La prop **value** sera équivalente à la variable fournie et déclarée lors de l'initialisation de nos **useState**.

La prop **keyboardType**, ayant pour valeur **numeric**, conditionnera le type de clavier à afficher. Ici, nous récupérons une taille et un poids, nous n'avons donc besoin que d'une valeur numérique.

Vous pouvez consulter la documentation du composant **Input** de **React Native Elements** ici :
https://reactnativeelements.com/docs/components/input.

À partir de maintenant, pensez à vérifier les autorisations de l'application **Expo Go** et veillez à désactiver la **Superposition à d'autres applications**. Cette autorisation peut vous empêcher d'utiliser le clavier pour entrer des valeurs.

Ajoutons le composant qui sera lié à notre méthode **calculateImc** :

```
<TouchableOpacity
    style={styles.calculateBtn}
    onPress={() => { calculateIMC(size, weight) }}
>
    <Text style={styles.calculateBtnText}>
        Calculer mon IMC
    </Text>
</TouchableOpacity>
```

Le composant **<TouchableOpacity></TouchableOpacity>** aura pour prop **onPress**, qui permet de déclencher un événement à la pression de la zone.

Lorsque cet élément sera pressé, nous voulons lancer le calcul du score IMC grâce à notre méthode **calculateIMC**, en lui fournissant les variables **size** et **weight**.

Si l'utilisateur déclenche l'événement avant l'insertion des données, la méthode exécutera donc le cas par défaut du **switch** et enregistrera ce résultat dans la variable **result** :

```
default:
    setResult("Les valeurs données sont incorrectes");
    break;
```

Le texte contenu dans le composant **<Text></Text>** peut être personnalisé, l'apparence finale de cette zone ressemblera à un bouton.

Il ne nous reste qu'à implémenter la **vue** qui affichera le résultat du calcul :

```
<SafeAreaView>
    <Text style={styles.resultImc}>
        {imc}
    </Text>
    <Text style={styles.resultText}>
        {result}
    </Text>
</SafeAreaView>
```

La **SafeAreaView** conditionnera l'affichage dans les limites de l'écran (IOS). Nous allons afficher le contenu de la variable **imc** et le contenu de la variable **result** dans deux composants textes différents.

Cette implémentation est uniquement liée à la modification de style qui sera appliquée par la suite.

Créez votre constante **styles**, créez une **StyleSheet** et ajoutez les instructions suivantes :

```
  container: {
     width: "100%",
     flex: 1,
     justifyContent:"center",
  },
  scrollContainer: {
     width: "100%",
     flex: 1,
  },
  calculateBtn: {
     margin: 30,
     borderWidth: 1 / 2,
     borderRadius: 4,
     backgroundColor: "#333",
     color: "#fff",
     textAlign: "center",
     paddingTop: 12,
     paddingBottom: 12,
     justifyContent: "center",
  },
  calculateBtnText: {
     fontSize: 24,
     textAlign: "center",
     color: "#fff",
  },
  resultImc: {
     fontSize: 48,
     fontWeight:"900",
     textAlign: "center",
     color:"#000",
  },
  resultText: {
     fontSize: 24,
     textAlign: "center",
     marginRight:12,
     marginLeft:12,
  }
```

N'oubliez pas d'actualiser votre fichier **index.js (components > index.js)** pour exporter votre nouveau **Composant natif** :

```
import WelcomeText from './WelcomeText';
import ImcCalculator from './ImcCalculator';

export {
   WelcomeText,
   ImcCalculator,
}
```

Ouvrez votre fichier **CalculatorScreen.js**.

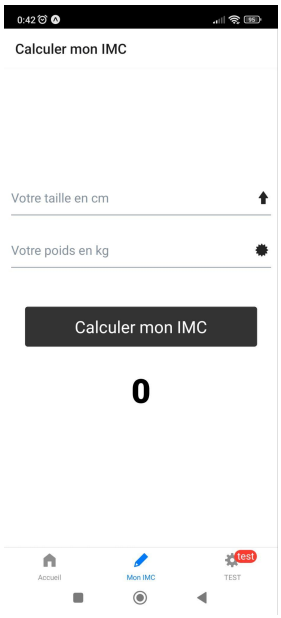
Importez votre nouveau composant :

```
import { ImcCalculator } from '../components';
```

Votre **CalculatorScreen** va pouvoir être allégé au maximum :

```
class CalculatorScreen extends React.Component {
   render() {
      return (
         <View style={styles.container}>
            <ImcCalculator/>
         </View>
      );
   }
}
const styles = StyleSheet.create({
   container: {
      flex: 1,
      backgroundColor: '#fff',
      alignItems: 'center',
      justifyContent: 'center',
   },
})
```

Vous pouvez désormais tester l'implémentation en lançant **Expo Go**.
Je vous conseille de redémarrer le serveur ou de le recharger, pour être sûr de visualiser tous vos changements.




Votre screen **Calculator (ou Mon IMC)** devrait ressembler à l'image ci-dessus.

## Création de l'écran À propos

Nous avons bien avancé dans la création de l'application.

Notre fonctionnalité principale, le calculateur d'IMC étant fonctionnel, il ne vous reste qu'à créer une page **À propos**, qui remplacera la page de test et qui contiendra vos informations "personnelles", en tant que développeur de l'application.

Nous reviendrons également sur l'écran d'accueil, pour améliorer son visuel.

Ouvrez votre fichier **TestScreen.js** et remplacez **TestScreen** par **AboutScreen**.

Renommez le fichier **AboutScreen.js**.

Ouvrez le fichier **screens>index.js** et renommez les occurrences de **TestScreen** en **AboutScreen.**

Ouvrez le fichier **NavigationTabs.js** et renommez les occurrences de **TestScreen** en **AboutScreen.**
Modifiez également le **tabBarLabel** en **À propos**.
Supprimez le **tabBarBadge** du **Tab.Screen** contenant l'écran **AboutScreen**.

Ouvrez le fichier **App.js** et renommez les occurrences de **TestScreen** en **AboutScreen.**

Revenez sur le fichier **AboutScreen.js** et vérifiez votre application en accédant à la page **À propos**, depuis la **TapBar**.

Nous allons commencer par supprimer les deux composants **Button** qui sont présents sur l'écran.

Nous allons utiliser des composants de la librairie **React Native Elements** sur cette page **À propos**.
Ajoutez la ligne d'import suivante :

```
import * as RNElements from '@rneui/base';
```

Pour ajouter un composant de **React Native Elements**, il faudra donc créer un composant **RNElements** et mentionner le composant à utiliser en le précédant d'un "**.**".
Par exemple :

```
<RNElements.Avatar/>
```

Nous allons ajouter un avatar, pour personnaliser la page **À propos**.

Je vous ai préparé un avatar animé, que vous pouvez récupérer ici : https://ressources.webdevoo.com/globals/detail-formation?formationId=1

Cliquez sur le bouton "**0,00€ / Téléchargement**" pour télécharger l'image et transférez cette image dans le dossier **assets** de votre projet d'application.

Dans le fichier **AboutScreen.js**, ajoutez un composant **<RNElements.Avatar />** au dessus du composant **Text** et donnez lui les props suivantes :

- **size="xlarge"**
- **rounded**
- **source={require("../assets/animated_cat.gif")}**

Vous devriez donc avoir ce composant :

```
<RNElements.Avatar
    size="xlarge"
    rounded
    source={require("../assets/animated_cat.gif")}
/>
```

Ouvrez **Expo Go** et vérifiez l'implémentation de ce nouveau composant sur l'écran **À propos.**

Espace l'avatar avec la prop **margin** :

```
margin={32}
```

Ne laissons pas les **composants cores** de côté.

Importez le composant **SectionList**, sur la ligne d'import de **'react-native'**.

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

Nous allons créer une liste de données, entre les imports et la classe **AboutScreen**, qui sera contenue dans la constante **aboutSectionsDatas**. Cette liste de données sera un objet **JSON** et contiendra des informations sur l'application :

```
const aboutSectionsDatas = [
   {
      title: "Développeur",
      data: ["Webdevoo"],
   },
   {
      title: "Type de projet",
      data: ["Open-Source"]
   },
   {
      title: "Date de création",
      data: ["Mai 2023"]
   },
   {
      title: "Objectif de l'application",
      data: ["Module de formation", "Apprendre à créer une application avec
React Native"]
   },
   {
      title: "Lieu de formation",
      data: ["G2R Formation", "102 Av. Philippe Auguste", "75011 Paris"]
   },
   {
      title: "Nom du formateur",
      data: ["Dufrène Valérian"]
   },
   {
      title: "Informations complémentaires",
      data: ["Projet réalisé lors d'une formation React Native", "Objectif
final [Chapitre 1, Module 2] de la formation React Native"]
   }
];
```

Nous allons implémenter le composant **SectionList**, en remplacement du composant **Text** présent sur **AboutScreen.js**.

**SectionList** utilisera notre liste de données et l'affichera en implémentant les props suivants :

- **sections** : Cette prop attends la valeur de la liste de données.
- **keyExtractor** : Nous n'avons pas passé de paramètre **key** à nos éléments, nous allons donc utiliser cette prop pour générer une **key** personnalisée, qui sera une concaténation de l'**item** et de son **index**.
- **renderItem** : Cette prop contiendra une fonction qui renverra l'**item** à l'intérieur d'un composant **Text**, et ce composant **Text** sera englobé d'un composant **View**.
- **renderSectionHeader** : Cette prop contiendra une fonction qui utilisera le paramètre (**{ section: { title } }**) et qui renverra un composant **Text** contenant **title**.
- **style** : Cette prop contiendra la valeur **{styles.sectionList}**, pour ajouter des instructions de style supplémentaires.

```
<SectionList
                  sections={aboutSectionsDatas}
                  keyExtractor={(item, index) => item + index}
                  renderItem={({ item }) => (
                     <View style={styles.item}>
                        <Text
style={styles.sectionListTitle}>{item}</Text>
                     </View>
                  )}
                  renderSectionHeader={({ section: { title } }) => (
                     <Text
style={styles.sectionListHeader}>{title}</Text>
                  )}
                  style={styles.sectionList}
               />
```

Ajoutez ces instructions de style pour mettre en forme la **SectionList** et ses **Composants** :

```
sectionList: {
      width:"100%",
      flex:1,
      textAlign: "center",
   },
   sectionListTitle: {
      fontSize: 18,
      textAlign: "center",
      color:"#666",
   },
   sectionListHeader: {
      fontSize: 22,
      fontWeight:"700",
      backgroundColor: '#fff',
      padding: 10,
      textAlign: "center",
   },
   item: {
      paddingLeft: 15,
      paddingRight: 15,
      marginVertical: 10,
   },
```

Vous pouvez désormais ouvrir l'application avec **Expo Go** pour vérifier votre intégration et modifier les informations présentes dans la liste de données.

## Réfaction de l'écran d'accueil

Nous avons laissé l'écran d'accueil de côté, pour nous concentrer sur le reste de l'application.

Il est temps de retourner sur le fichier **HomeScreen.js** pour harmoniser son contenu avec le reste de notre application.

Importez la librairie **React Native Elements** :

```
import * as RNElements from '@rneui/base';
```

Remplacer le composant **Button** par un composant **RNElements.Button** :

```
<RNElements.Button
                title="Calculer mon IMC"
                onPress={() => {
                   this.props.navigation.navigate("Calculator")
                }}
            />
```

Nous allons styliser ce bouton avec les **props** fournies par **React Native Elements.**

Commençons avec le style du **title,** ajoutez ces instructions :

```
titleStyle={{
    color: 'white',
    marginHorizontal: 20,
    fontWeight: "bold",
    letterSpacing: 1.5,
    fontSize:24,
}}
```

Modifions le style du bouton :

```
buttonStyle={{
    backgroundColor: 'black',
    borderWidth: 2,
    paddingTop:16,
    paddingBottom:16,
    borderColor: 'white',
    borderRadius: 8,
}}
```

À présent, nous allons modifier l'image utilisée sur la page d'accueil.

Je vous ai préparé un logo qui est téléchargeable à cette adresse : https://ressources.webdevoo.com/globals/detail-formation?formationId=3.

Téléchargez l'image et placez là dans le dossier **assets**, puis remplacez la prop **source** du composant **Image** pour utiliser cette image locale :

```
source={require("../assets/imc-calulator-home-logo.png")}
```

Modifiez à présent les instructions de style de **container** et **homeImg** :

```
container: {
      flex: 1,
      backgroundColor: '#fff',
      alignItems: 'center',
      justifyContent: "space-between",
      paddingBottom:64,
   },
   homeImg: {
      width: 170,
      height: 72,
      borderRadius: 4,
      margin: 96,
   },
```

Dernière modification pour harmoniser l'application, ouvrez à nouveau le fichier **NavigationTabs.js** et ajoutez la prop **tabBarActiveTintColor** à vos composants **Tab.Screen**, en leur donnant la valeur **"#333"** :

```
tabBarActiveTintColor: '#333',
```

## Apprendre à compiler et exporter son application

Nous allons pouvoir étudier la méthode de **build** d'une application avec **React Native** et **Expo**.

Comme le déploiement d'une application sur les stores nécessite un compte payant, nous allons étudier le build de l'application via **Expo EAS**, qui nous permettra d'extraire un fichier **.APK** de notre application.

Cet **APK** pourra alors être transféré sur un téléphone Android et installé, en acceptant les risques de sécurité auprès du vérificateur d'installation.

Pour commencer notre **build**, nous aurons besoin d'installer globalement la **CLI EXPO EAS** :

```
npm install -g eas-cli
```

Il vous faudra également créer un compte sur **expo.dev** :
https://expo.dev/signup.

Le compte expo vous permettra d'avoir un endroit de stockage de vos builds et vous pourrez télécharger l'apk depuis vos projets.

Dans votre terminal de commande, entrez :

```
eas login
```

Entrez vos informations de connexion **Expo**.

Une fois connecté à **eas**, entrez la commande :

```
eas whoami
```

Cette commande vous permet de vérifier si vous êtes connecté au bon compte.

Créez le fichier **eas.json** à la racine de votre projet et entrez cette configuration :

```
  {
    "build": {
         "local": {
             "android": {
             "buildType": "apk"
         }
    },
    "preview2": {
      "android": {
        "gradleCommand": ":app:assembleRelease"
      }
    },
    "preview3": {
      "developmentClient": true
    },
    "production": {}
  }
}
```

Pour finir, entrez la commande :

```
eas build -p android --profile local
```

Cette commande va lancer le build de votre application.

Si EAS vous demande une autorisation pour configurer le projet, entrez **Y**.

Cliquez sur **Entrée**, quand **EAS** va vous demander une confirmation du nom de domaine de l'application, nous n'avons pas besoin de domaine particulier pour cette application.

Laissez le build s'effectuer, en attendant sa compilation, vous pouvez aller sur votre compte **Expo**, cliquer sur votre projet et voir l'avancement du build pour vérifier si aucune erreur ne se produit.
De mon côté, le build complet à duré 8 minutes et  10 secondes.

Dès que le build est terminé, cliquez sur votre projet, cliquez ensuite sur **Android Play Store Build** pour ouvrir le build et cliquez sur **Download** pour télécharger l'apk de votre projet.

Vous pouvez utiliser votre téléphone pour vous connecter sur **Expo** et suivre ces étapes, cela vous permettra de télécharger et d'installer l'apk directement. Sinon, transférez l'apk sur votre téléphone.

Lors de l'installation de l'apk sur votre téléphone, vous aurez certainement une alerte de sécurité qui vous avertira que l'application n'est pas connue.

C'est tout à fait normal, car votre application n'a pas été publiée sur un compte payant pour la diffusion sur le Play Store et n'a donc pas été signée de la même façon que les applications que vous installez régulièrement.

À l'affichage de ce message, cliquez simplement sur **Plus de détails > Installer quand même**.
Si vous avez cliqué sur le bouton **OK**, relancez l'installation de l'apk.

Une demande vous sera exposée, pour envoyer l'application à la vérification du Play Protect, cliquez sur **Ne pas envoyer**.

Votre application est désormais installée sur votre téléphone, lancez là et explorez tous les **Screens**.