Les bases de React Native

Table des matières

I. Contexte

Durée : 1 h

Environnement de travail : Visual Studio Code et Expo

Pré-requis : Environnement de développement React Native fonctionnel

Contexte

React Native est un framework JavaScript pour le développement d'applications mobiles. Son fonctionnement est très proche de celui de React et permet d'utiliser les fonctionnalités natives des appareils sur lesquels une application est exécutée.

Nous allons voir ici les bases de la mise en place d'interfaces graphiques en React Native, en développant tout d'abord ses principes de templating, avant de décrire quelques composants d'interface proposés nativement par le framework.

II. Le templating

Objectif

- Découvrir la structuration d'une interface graphique

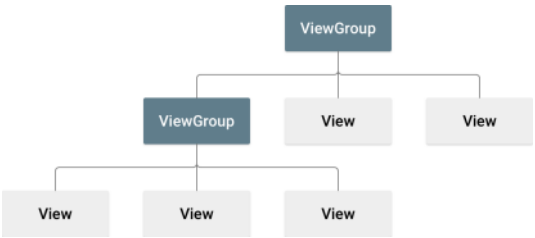
Mise en situation

Bien que React Native soit très proche de React, il existe quelques particularités qui les différencient. L'une d'entre elles se trouve dans la manière de gérer les interfaces graphiques. En effet, lors du développement d'une application mobile multiplateforme, il est courant de souhaiter que les éléments d'interface s'adaptent aux normes de la plateforme en cours. Nous allons ici voir comment React Native est capable de produire des interfaces graphiques s'adaptant à la plateforme d'exécution, sans nécessiter l'écriture de code spécifique.

Interface multiplateforme

Le point fort de React Native réside dans sa capacité à permettre l'écriture d'un seul élément de code agnostique de toute plateforme, mais qui aura un comportement identique toutes plateformes confondues, tout en proposant un rendu adapté aux codes ergonomiques de celle-ci.

Pour réaliser cela, React Native ne se base pas sur l'utilisation de HTML, mais sur des composants nativement présents dans le framework, qui seront ensuite traduits différemment en fonction de la plateforme ciblée.



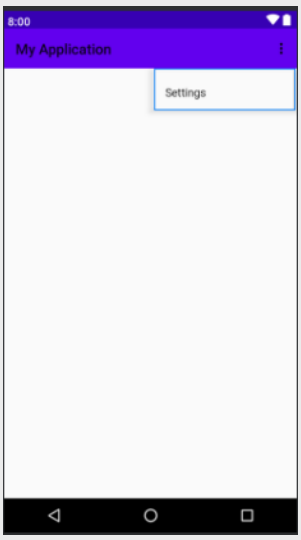
En effet, les applications mobiles natives ne se basent pas sur une plateforme web, mais sur une interprétation du code par le système de l'appareil.

Ainsi, sous Android par exemple, le template des interfaces d'une application se définit sous la forme d'un XML qui est ensuite traduit par le système.

Le schéma fourni illustre comment se hiérarchise l'imbrication des éléments graphiques au sein d'une application mobile Android.

Exemple

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="com.example.myapplication.MainActivity">
    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/action_settings"
        app:showAsAction="never" />
</menu>
```



Le fichier XML présenté décrit une structure de menu avec un élément intitulé `Settings`.

Syntaxe **À retenir**

- L'écriture d'interfaces graphiques en React Native ne se fait pas en utilisant du HTML comme avec React, mais par l'intégration de composants graphiques natifs au framework, qui sont ensuite traduits en éléments compatibles avec la plateforme exécutant l'application.
- Ainsi, même si React Native utilise du JavaScript, il n'est pas interprété dans un contexte web.

III. Appliquez la notion

Exercice 1

[solution n°1 p.17]

Exercice

Dans le cours précédent, le terme « templating » désigne la manière dont sont structurées les interfaces graphiques.

- ○ Vrai
- ○ Faux

Exercice

Comme React, React Native est capable d'interpréter du HTML pour l'affichage d'éléments graphiques.

- ○ Vrai
- ○ Faux

Exercice

Par quels procédés React Native peut-il réussir à proposer des interfaces adaptées à la plateforme exécutant l'application ?

- ☐ Grâce à des composants natifs du framework
- ☐ En traduisant un code agnostique en composants spécifiques en fonction de la plateforme
- ☐ En utilisant du HTML et en laissant le navigateur se charger de faire un rendu adapté
- ☐ Par l'écriture d'un code spécifique et adapté à la plateforme lorsque l'on souhaite afficher des éléments d'interface

Exercice

Puisque React Native utilise du JavaScript, alors il utilise une plateforme web lors de l'exécution d'une application.

- ○ Vrai
- ○ Faux

Exercice

De quelle manière sont décrits les éléments graphiques d'une application Android ?

- ○ En utilisant des balises HTML
- ○ Par une structure XML
- ○ Par des objets JavaScript

IV. Les composants de base

Objectif

- Se familiariser avec les composants de base de React Native

Mise en situation

L'écriture d'interfaces graphiques en React Native se fait par l'imbrication ordonnée de composants. Selon les spécificités de l'application, ceux-ci peuvent être des composants complexes et spécifiques, ou des composants présents nativement dans le framework. Nous allons présenter ici quelques-uns des composants de base du framework React Native.

View

Le composant `View` est le composant *container* de base de React Native : il peut être vu comme un équivalent de la balise `<div>` en HTML.

Ce composant peut contenir autant d'autres composants que voulus et est souvent utilisé comme *wrapper* des différents éléments graphiques de l'application.

Afin de l'utiliser, il est nécessaire de réaliser un import avec la syntaxe suivante : `import { View } from 'react-native';`.

Ensuite, dans le retour de la fonction de définition du composant en cours, il faut ajouter le composant `View` de la même manière que n'importe quel composant : `<View></View>`.

Exemple

```
1 import React from 'react';
2 import { View } from 'react-native';
3
4 export default function App() {
5   return (
6     <View style={{ borderColor: "red", borderWidth: 5 }}>
7     </View>
8   );
9 };
```

L'exécution du code présenté affiche un composant `View` vide. On peut remarquer l'utilisation de la `props style` du composant permettant de définir une bordure de couleur rouge d'une largeur de 5.

Text

Le composant `Text` est l'élément natif de React Native pour l'affichage de texte et peut être comparé à la balise `<p>` de HTML.

Son utilisation nécessite un import sous la forme `import { Text } from 'react-native';`.

Ensuite, dans le retour de la fonction de définition du composant en cours, il faut ajouter le composant `Text` de la même manière que n'importe quel composant : `<Text></Text>`.

Ce composant peut afficher des valeurs dynamiques associées à des variables via l'utilisation de la syntaxe `{myVariable}`. De plus, des comportements prédéfinis du composant sont disponibles via l'utilisation des différentes props qui lui sont associées.

Exemple

```
1 import React from 'react';
2 import {Text, View } from 'react-native';
3
4 const onPressText = () => {
5   console.log('Le texte a été pressé.');
6 };
```

```
7
8 export default function App() {
9   const text = 'texte dynamique';
10
11   return (
12     <View>
13       <Text onPress={() => onPressText()}>
14         Je suis un texte statique
15       </Text>
16       <Text>
17         Je suis un {text}
18       </Text >
19     </View>
20   );
21 }
```

L'exécution du code présenté affiche les textes « *Je suis un texte statique* » et « *Je suis un texte dynamique* » sur des lignes séparées.

Lorsque l'on clique sur la première ligne, le message « *Le texte a été pressé.* » s'affiche dans la console. En effet, sur le premier composant `Text`, la `props onPress` exécute la fonction `onPressText`, définie plus haut, lors de la détection d'une action de l'utilisateur.

Image

Le composant `Image` est le composant de bas niveau de React Native pour l'affichage d'une image, au même titre que la balise `<img>` peut l'être en HTML.

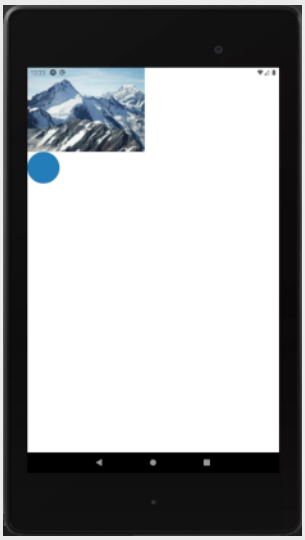
Son utilisation nécessite un import tel que `import { Image } from 'react-native';` avant d'être utilisé dans la fonction de retour du composant en cours, en suivant la syntaxe `<Image />`.

Ce composant embarque des props permettant notamment de spécifier le chemin de l'image à afficher via `source`, ou ses dimensions grâce à `style`. Il existe de nombreuses autres props et méthodes disponibles, que l'on peut retrouver dans la documentation du composant : https://reactnative.dev/docs/image. À noter également que si le composant `<Image />` n'a pas une largeur et une hauteur définies avec les propriétés *width* et *height*, l'image ne s'affichera pas.

Exemple

```
import React from 'react';
import { Image, View } from 'react-native';

export default function App() {
  return (
    <View>
      <Image
        style={{ width: 280, height: 200 }}
        source={{
          uri: 'https://i.pinimg.com/236x/5b/c2/c6/5bc2c65295d011c580ab5bf3563dabf9--stock-photo-free-free-stock-image.jpg'
        }}
      />
      <Image
        style={{ width: 77, height: 75 }}
        source={{
          uri:
'
        }}
      />
    </View>
  );
}
```



L'exécution de ce code affiche en premier une image depuis l'URL https://i.pinimg.com/236x/5b/c2/c6/5bc2c65295d011c580ab5bf3563dabf9--stock-photo-free-free-stock-image.jpg, puis une image au format de base 64.

Attention

Le composant `Image` de React Native ne prend en charge que les format généraux suivants : png, jpg, jpeg, bmp, gif. Les formats webp et psd ne sont quant à eux respectivement disponibles que pour les plateformes Android et iOS.

Attention

Il est nécessaire de toujours préciser une hauteur et une largeur au composant `Image`, sinon celui-ci ne s'affiche pas.

TextInput

Le composant `TextInput` est le composant de base permettant la saisie de données par un utilisateur, son équivalent HTML serait `<input>`.

Son utilisation nécessite un import préalable avec `import { TextInput } from 'react-native';`.

Puis il peut être utilisé dans la fonction de retour du composant en cours en suivant la syntaxe `<TextInput />`.

Afin de lire l'entrée de l'utilisateur, il est possible d'utiliser la propriété `onChangeText`, qui prend pour paramètre une fonction qui est appelée chaque fois que le texte est modifié.

De nombreuses props sont associées à ce composant, en particulier celles nécessaires à la gestion des événements de saisie, que l'on peut retrouver à l'adresse https://reactnative.dev/docs/textinput.

Exemple

```
import { useState } from 'react';
import React from 'react';
import { TextInput, View } from 'react-native';

export default function App() {
 const [name, setName] = useState('');

  return (
    <View>
      <TextInput
        style={{ height: 40, borderWidth: 1, marginTop: 30 }}
        onChangeText={text => setName(text)}
        value={name}
      />
    </View>
  );
}
```

Exemple

```
import React from 'react';
import { TextInput, View } from 'react-native';

const onEndEditingHandler = (event) => {
  console.log(event.nativeEvent.text);
}

export default function App() {
  return (
    <View>
      <TextInput
        style={{ height: 40, borderWidth: 1, marginTop: 30 }}
        onEndEditing={event => onEndEditingHandler(event)}
        placeholder='TextInput'
      />
    </View>
  );
}
```

Le code présenté permet d'afficher un champ de saisie avec un *placeholder* `TextInput`.

Lors de la validation de la saisie par l'utilisateur, l'événement `onEndEditing` est déclenché et la fonction `onEndEditingHandler` est appelée, ce qui affiche le texte saisi dans les logs.

ScrollView

Le composant `ScrollView` est un *container* permettant le défilement vertical et horizontal des éléments qu'il contient.

Son utilisation nécessite un import respectant la syntaxe `import { ScrollView } from 'react-native';`.

Il est ensuite ajouté à la fonction de retour du composant, en suivant la syntaxe `<ScrollView></ScrollView>`, et doit contenir les composants pour lesquels le défilement est nécessaire.

Exemple

```
1 import React from 'react';
2 import { ScrollView, Image } from 'react-native';
3
4 export default function App() {
5   return (
6     <ScrollView>
```

```
      <Image
        style={{ width: 590, height: 510 }}
        source={{
          uri: 'https://i.pinimg.com/236x/5b/c2/c6/5bc2c65295d011c580ab5bf3563dabf9--stock-photo-free-free-stock-image.jpg'
        }}
      />
      <Image
        style={{ width: 590, height: 510 }}
        source={{
          uri: 'https://i.pinimg.com/236x/5b/c2/c6/5bc2c65295d011c580ab5bf3563dabf9--stock-photo-free-free-stock-image.jpg'
        }}
      />
    </ScrollView>
  );
}
```

Le code ci-dessus affiche deux images superposées dont la taille ne permet pas un affichage complet sur l'écran de l'appareil.

Grâce au composant `ScrollView`, l'utilisateur peut faire défiler l'écran vers le bas et afficher entièrement la seconde image.

Attention

Le composant `ScrollView` charge la totalité des composants qu'il contient, même si ceux-ci ne sont pas affichés. Il est donc adapté pour de petites quantités d'éléments.

Si de nombreux éléments doivent être listés, le composant `FlatList` sera plus adapté.

Syntaxe **À retenir**

Le framework React Native propose des composants de bas niveaux permettant de gérer nativement les problématiques d'interface utilisateur. Parmi ceux-ci, on retrouve :

- `View` servant de *container* de composants,
- `Text` pour l'affichage d'éléments textuels,
- `Image` pour l'affichage d'une image,
- `TextInput` pour gérer les saisies des utilisateurs,
- `ScrollView` pour gérer le défilement horizontal ou vertical des éléments qu'il contient.

Complément **Liens utiles**

https://reactnative.dev/docs/view

https://reactnative.dev/docs/text

https://reactnative.dev/docs/image[1]

[2]https://reactnative.dev/docs/textinput

https://reactnative.dev/docs/using-a-scrollview

V. Exercice : Appliquez la notion

Question

[solution n°2 p.17]

En utilisant votre environnement de travail React Native, écrivez une interface graphique permettant de répondre au besoin suivant.

Afin de rester proche de ses utilisateurs, un éditeur d'application mobile souhaite ajouter une page permettant la saisie d'une remarque ou d'une question.

Pour cela, vous intégrerez les éléments listés ci-dessous :

- Une image de point d'interrogation, comme celle disponible ici : https://f0.pngfuel.com/png/343/183/question-mark-decal-png-clip-art.png
- Un texte indiquant « *Veuillez saisir une remarque ci-dessous* »
- Un champ de saisie multilignes avec un *placeholder* indiquant « *Saisissez votre remarque* »
- Le texte saisi devra s'afficher dans la console
- Afin de rester utilisable pour tous les écrans, cette page devra autoriser le défilement vertical si besoin

VI. Essentiel

1 https://reactnative.dev/docs/text
2 https://reactnative.dev/docs/text

VII. Auto-évaluation

A. Exercice final

Exercice 1

[solution n°3 p.20]

Exercice

Parmi les éléments proposés, lesquels sont des composants *containers* ?

- [ ] Text
- [ ] Image
- [ ] TextInput
- [ ] View
- [ ] ScrollView

Exercice

Si React Native n'utilise pas de HTML, c'est uniquement parce qu'il ne s'exécute pas dans un contexte web.

- ○ Vrai
- ○ Faux

Exercice

Une application React Native sera une application native de la plateforme l'exécutant.

- ○ Vrai
- ○ Faux

Exercice

Le composant de base `Image` prend en compte le format svg.

- ○ Vrai
- ○ Faux

Exercice

Le résultat du code présenté va afficher une image accessible via une URL.

```
import React from 'react';
import { Image, View } from 'react-native';

export default function App() {
  return (
    <View>
      <Image
        source={{
          uri: 'https://i.pinimg.com/236x/5b/c2/c6/5bc2c65295d011c580ab5bf3563dabf9--stock-photo-free-free-stock-image.jpg'
        }}
      />
    </View>
  );
}
```

- ○ Vrai
- ○ Faux

Exercice

Quel est le composant de base de React Native pour l'affichage d'un message ?

- ○ View
- ○ TextInput
- ○ Text

Exercice

Le composant `View` est...

- ☐ Le composant *container* de base de React Native
- ☐ Un composant permettant de hiérarchiser les composant d'une interface
- ☐ Un composant utilisé pour rendre visibles les éléments qu'il contient

Exercice

Le composant `ScrollView` est conseillé pour optimiser l'affichage d'une liste d'éléments si celle-ci est constituée...

- ○ De nombreux éléments
- ○ De peu d'éléments

Exercice

Cet extrait de code permet d'afficher dans la console un texte saisi lorsque celui-ci est validé.

```
import React from 'react';
import { TextInput, View } from 'react-native';

const onEndEditingHandler = (event) => {
  console.log(event.text);
}

export default function App() {
  return (
    <View>
      <TextInput
        onEndEditing={event => onEndEditingHandler(event.nativeEvent)}
      />
    </View>
  );
}
```

- ○ Vrai
- ○ Faux

Exercice

Quelles modifications faut-il apporter au code présenté pour autoriser le défilement vertical de la première image uniquement, tout en s'assurant que la deuxième image soit toujours entièrement visible ?

```
import React from 'react';
import { ScrollView, Image, View } from 'react-native';

export default function App() {
  return (
    <View>
      <ScrollView>
        <Image
          style={{ width: 590, height: 800 }}
          source={{
            uri: 'https://i.pinimg.com/236x/5b/c2/c6/5bc2c65295d011c580ab5bf3563dabf9--stock-photo-free-free-stock-image.jpg'
          }}
        />
      </ScrollView>

      <Image
        style={{ width: 590, height: '50%' }}
        source={{
          uri: 'https://i.pinimg.com/236x/5b/c2/c6/5bc2c65295d011c580ab5bf3563dabf9--stock-photo-free-free-stock-image.jpg'
        }}
      />
    </View>
  );
}
```

- Aucune, elle est déjà contenue dans un composant `ScrollView` permettant le défilement vertical
- Par l'ajout d'un composant `View`, d'une hauteur de 50 % de la hauteur de l'écran, englobant le composant `ScrollView` actuel

B. Exercice : Défi

Dans cet exercice, une société souhaite réaliser une application au sein de laquelle une des pages devra permettre de pouvoir voir deux images superposées, et dont la fonctionnalité principale répercutera une action exécutée dans le périmètre de la première image vers celui de la seconde image. Avant de lancer le développement du projet, la société souhaite s'assurer que ce fonctionnement est réalisable en React Native.

Question

[solution n°4 p.22]

Afin de démontrer que cela est possible, vous allez réaliser un POC (*Proof Of Concept*) de la page décrite par la société. Pour cela, il est nécessaire de présenter une page simplifiée permettant d'afficher deux images de tailles inconnues sur toute la hauteur de l'écran, puis de simuler une action via un champ de saisie texte, placé sous la première image, dont la saisie sera affichée sous forme de message sous la seconde image. Les images devront être scrollables de manière indépendante et occuper la totalité de l'écran disponible.

Indice :

Afin de stocker une valeur et d'impacter l'interface lors de son changement, nous allons utiliser le state. Au premier chargement l'état du composant à une valeur initiale. Le state va permettre de stocker les données modifiées dynamiquement par le visiteur. Cette notion sera utilisée pour le défi mais elle sera abordée plus tard dans le cours. Voici le code à utiliser pour cet exercice.

```
import React, { useState } from 'react';
export default function App() {
  const [textValue, setTextValue] = useState(''); // Défini une variable textValue pouvant être modifiée par la fonction setTextValue
  const setValueState = setTextValue('Valeur de test'); // Affecte une nouvelle valeur à textValue
  console.log(textValue); // Affiche la valeur de textValue dans la console
}
      const onEndEditingHandler = (event) => {
```

```
        setTextValue(event.text);
    } // Fonction fléchée à utiliser dans l'input qui permet de transmettre ses données.
```

Solutions des exercices

Exercice p. 4 Solution n°1

Exercice

Dans le cours précédent, le terme « templating » désigne la manière dont sont structurées les interfaces graphiques.

- (●) Vrai
- ( ) Faux

Exercice

Comme React, React Native est capable d'interpréter du HTML pour l'affichage d'éléments graphiques.

- ( ) Vrai
- (●) Faux

Exercice

Par quels procédés React Native peut-il réussir à proposer des interfaces adaptées à la plateforme exécutant l'application ?

- [x] Grâce à des composants natifs du framework
- [x] En traduisant un code agnostique en composants spécifiques en fonction de la plateforme
- [ ] En utilisant du HTML et en laissant le navigateur se charger de faire un rendu adapté
- [ ] Par l'écriture d'un code spécifique et adapté à la plateforme lorsque l'on souhaite afficher des éléments d'interface

Exercice

Puisque React Native utilise du JavaScript, alors il utilise une plateforme web lors de l'exécution d'une application.

- ( ) Vrai
- (●) Faux

Exercice

De quelle manière sont décrits les éléments graphiques d'une application Android ?

- ( ) En utilisant des balises HTML
- (●) Par une structure XML
- ( ) Par des objets JavaScript

p. 12 Solution n°2

```
1 import React from 'react';
2 import { ScrollView, Image, TextInput, Text } from 'react-native';
3
4 const onEndEditingHander = (event) => {
5   console.log(event.nativeEvent.text);
6 }
7
```

```
export default function App() {
  return (
    <ScrollView>
      <Image
        style={{ width: 50, height: 50 }}
        source={{
          uri: 'https://f0.pngfuel.com/png/343/183/question-mark-decal-png-clip-art.png'
        }}
      />
      <Text>Veuillez saisir une remarque ci-dessous</Text>
      <TextInput
        placeholder='Saisissez votre remarque'
        onEndEditing={event => onEndEditingHander(event)}
        multiline
      />
    </ScrollView>
  );
}
```


```
Unrecognized event: {"type":"client_log","level":"log","data":["Cette application gagnerait en lisibilité avec des couleurs et un peu de style.\nMerci d'en tenir compte."]}
Cette application gagnerait en lisibilité avec des couleurs et un peu de style.
Merci d'en tenir compte.
```

Exercice p. 13 Solution n°3

Exercice

Parmi les éléments proposés, lesquels sont des composants *containers* ?

- [x] Text
- [ ] Image
- [ ] TextInput
- [x] View
- [x] ScrollView

Exercice

Si React Native n'utilise pas de HTML, c'est uniquement parce qu'il ne s'exécute pas dans un contexte web.

- ○ Vrai
- ◉ Faux

  React Native n'utilise pas de HTML, car son fonctionnement nécessite une interprétation de ses composants en éléments natifs des différentes plateformes mobiles.

Exercice

Une application React Native sera une application native de la plateforme l'exécutant.

- ◉ Vrai

  Comme React Native adapte ses composants en éléments natifs de la plateforme lors de l'exécution de l'application, celle-ci est bien une application native de la plateforme en cours.

- ○ Faux

Exercice

Le composant de base `Image` prend en compte le format svg.

- ○ Vrai
- ◉ Faux

  Seuls les format png, jpg, jpeg, bmp, gif, webp (Adroid) et psd (iOS) sont pris en charge.

Exercice

Le résultat du code présenté va afficher une image accessible via une URL.

```
1 import React from 'react';
2 import { Image, View } from 'react-native';
3
4 export default function App() {
5   return (
6     <View>
7       <Image
8         source={{
9           uri: 'https://i.pinimg.com/236x/5b/c2/c6/5bc2c65295d011c580ab5bf3563dabf9--stock-photo-free-free-stock-image.jpg'
10        }}
11      />
12    </View>
```

```
  );
}
```

- ○ Vrai
- ◉ Faux
  L'écran sera vide, car aucune dimension n'a été donnée à l'image.

Exercice

Quel est le composant de base de React Native pour l'affichage d'un message ?

- ○ View
- ○ TextInput
- ◉ Text

Exercice

Le composant `View` est...

- ☑ Le composant *container* de base de React Native
- ☑ Un composant permettant de hiérarchiser les composant d'une interface
- ☐ Un composant utilisé pour rendre visibles les éléments qu'il contient

Exercice

Le composant `ScrollView` est conseillé pour optimiser l'affichage d'une liste d'éléments si celle-ci est constituée...

- ○ De nombreux éléments
- ◉ De peu d'éléments
  Le composant `ScrollView` charge tous les éléments d'une liste, il est donc adapté pour les listes courtes.

Exercice

Cet extrait de code permet d'afficher dans la console un texte saisi lorsque celui-ci est validé.

```
import React from 'react';
import { TextInput, View } from 'react-native';

const onEndEditingHandler = (event) => {
  console.log(event.text);
}

export default function App() {
  return (
    <View>
      <TextInput
        onEndEditing={event => onEndEditingHandler(event.nativeEvent)}
      />
    </View>
  );
}
```

- [x] Vrai

  À la validation de la saisie, l'événement `onEndEditing` est déclenché, permettant l'appel à la fonction `onEndEditingHandler` qui affiche le texte de l'événement racine dans la console.

- [ ] Faux

Exercice

Quelles modifications faut-il apporter au code présenté pour autoriser le défilement vertical de la première image uniquement, tout en s'assurant que la deuxième image soit toujours entièrement visible ?

```
import React from 'react';
import { ScrollView, Image, View } from 'react-native';

export default function App() {
  return (
    <View>
      <ScrollView>
        <Image
          style={{ width: 590, height: 800 }}
          source={{
            uri: 'https://i.pinimg.com/236x/5b/c2/c6/5bc2c65295d011c580ab5bf3563dabf9--stock-photo-free-free-stock-image.jpg'
          }}
        />
      </ScrollView>

      <Image
        style={{ width: 590, height: '50%' }}
        source={{
          uri: 'https://i.pinimg.com/236x/5b/c2/c6/5bc2c65295d011c580ab5bf3563dabf9--stock-photo-free-free-stock-image.jpg'
        }}
      />
    </View>
  );
}
```

- [ ] Aucune, elle est déjà contenue dans un composant `ScrollView` permettant le défilement vertical
- [x] Par l'ajout d'un composant `View`, d'une hauteur de 50 % de la hauteur de l'écran, englobant le composant `ScrollView` actuel

  Tel quel, le défilement se fait sur la taille de la première image, ce qui ne permet pas de faire défiler l'écran jusqu'à la seconde image. En effet, celui-ci s'interrompt lorsque le bas de la première image est atteint. En englobant l'image dans un composant `View` faisant 50 % de haut, celle-ci ne dépassera jamais cette hauteur et le composant `Scrollview` permettra de la faire défiler verticalement, sans cacher la seconde image.

p. 15 **Solution n°4**

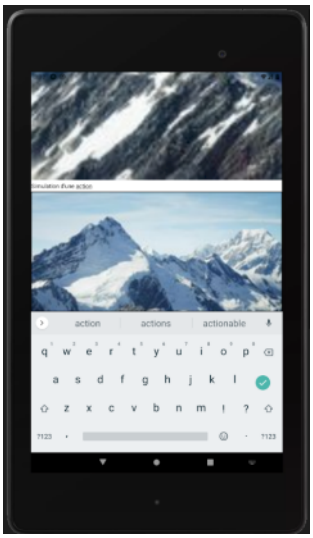
```
import React, { useState } from 'react';
import { ScrollView, Image, View, Text, TextInput } from 'react-native';


export default function App() {
  const [textValue, setTextValue] = useState('');

  const onEndEditingHandler = (event) => {
    setTextValue(event.text);
  }
```

```
11
12  return (
13    <View style={{ borderColor: 'black', borderWidth: 1 }}>
14      <View style={{ height: '50%' }}>
15        <ScrollView>
16          <Image
17            style={{ width: '100%', height: 800 }}
18            source={{
19              uri: 'https://i.pinimg.com/236x/5b/c2/c6/5bc2c65295d011c580ab5bf3563dabf9--
stock-photo-free-free-stock-image.jpg'
20            }}
21          />
22          <TextInput
23            placeholder="Ecrivez ici"
24            onEndEditing= {(event) => onEndEditingHandler(event.nativeEvent)}
25          />
26        </ScrollView>
27      </View>
28      <View style={{ height: '50%', borderColor: 'black', borderWidth: 1 }}>
29        <ScrollView>
30          <Image
31            style={{ width: '100%', height: 458 }}
32            source={{
33              uri: 'https://i.pinimg.com/236x/5b/c2/c6/5bc2c65295d011c580ab5bf3563dabf9--
stock-photo-free-free-stock-image.jpg'
34            }}
35          />
36          {
37            textValue !== '' &&
38            <View style={{ width: '100%', height: 100 }}>
39              <Text>Vous avez tapé {textValue}</Text>
40            </View>
41          }
42        </ScrollView>
43      </View>
44    </View>
45  );
46 }
47
```

La page contient deux images dans des *containers* permettant le défilement vertical, distincts et occupant la totalité de l'écran quelle que soit la taille de l'image utilisée.



En faisant défiler la première image, on accède à un champ de saisie sans que cela impacte l'affichage de la seconde image.

Suite à la saisie précédente, le bloc contenant la seconde image permet le défilement vertical et affiche un texte reprenant la saisie réalisée.