Les composants

React Native utilise ses propres composants de rendu, pour cibler les applications mobiles. Voyons un exemple :

```
<View>
    <TextInput placeholder='City'/>
    <Button title='Search' onPress={() => {}}/>
</View>
```

Le composant View est nécessaire pour afficher un contenu sur l'écran du téléphone. Le composant TextInput permet d'afficher un champ de recherche et le composant Button le bouton qui déclenchera l'action de recherche.

React Native gère les différents rendus en fonction de iOS ou Android et affichera le bouton de recherche spécifiquement dans le style de ces contextes. Le composant Button est donc générique, pour Android React Native utilisera un bouton **android.widget.Button** et pour iOS un bouton **UIButton**.

Présentation des principaux composants

React Native	Android	iOS	Web
<View>	<ViewGroup>	<UIView>	
<Text>	<TextView>	<UITextView>	<p>
<Image>	<ImageView>	<UIImageView>	<img>
<ScrollView>	<ScrollView>	<UIScrollView>	
<TextInput>	<EditText>	<UITextField>	<input type="text">

Vous trouvrez dans la documentation officielle la liste des composants principaux de React Native : compoments

Composants customisés

Comme vous travaillez avec la même librairie React, vous continuerez à utiliser tout ce que vous connaissez en React dans React Native.

Vous pouvez tout à fait créer des composants réutilisables, modulables...

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

const Tea = ({ genre }) => {
  return (
    <View>
      <Text>Tea time</Text>
    </View>
  );
```

```
}

export default Teapot = () => {
  return (
    <View>
      <Text>Tea time</Text>
      <Tea genre="mint" />
      <Tea genre="licorice" />
      <Tea genre="licorice" />
    </View>
  );
}
```

Composant View

Le composant View est un conteneur qui permet d'afficher les autres composants à l'écran, il permettra : de styliser les éléments de rendu avec flexbox et les styles CSS, d'inclure des composants tactiles et de gérer les actions d'accessibilitées liées à l'interface utilisateur.

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

const ViewExample = () => {
  return (
    <View
      style={{
        flexDirection: "row",
        height: 100,
        padding: 5
      }}
    >
      <View style={{ backgroundColor: "blue", flex: 0.3 }} />
      <View style={{ backgroundColor: "red", flex: 0.5 }} />
      <Text>Hello World!</Text>
    </View>
  );
};

export default ViewExample ;
```

Vous pouvez séparer les styles de vos composants en utilisant la classe **StyleSheet** de React Native :

```
import React from 'react';
import { StyleSheet, Text, View } from 'react-native';
const styles = StyleSheet.create({
  container: {
    flexDirection: "row",
    height: 100,
    padding: 5
  },
  blue: {
    backgroundColor: "blue", flex: 0.3
  },
  red: {
    backgroundColor: "red", flex: 0.5
  },

});

const ViewExample = () => {
    return (
      <View style={styles.container}>
        <Text style={[styles.blue]} />
        <Text style={[styles.red]} />
        <Text>Hello World!</Text>
```

```
      </View>
    );
}

export default ViewExample;
```

Vous pouvez également créer un fichier pour vos propres styles puis, vous les importez dans vos composants :

```
import React from 'react';
import { Text, View } from 'react-native';
import myStyle from './myStyle';

const ViewExample = () => {
    return (
      <View style={myStyle.container}>
        <Text style={[myStyle.blue]} />
        <Text style={[myStyle.red]} />
        <Text>Hello World!</Text>
      </View>
    );
}

export default ViewExample;
```

Flexbox rappels

Flex direction

Considérons la mise en page suivante :

```
<View style={{flex: 1, flexDirection: 'row'}}>
    <View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />
    <View style={{width: 50, height: 50, backgroundColor: 'skyblue'}} />
    <View style={{width: 50, height: 50, backgroundColor: 'steelblue'}} />
</View>
```

flexDirection : **'row'** :

```
------------
|XXX       |
|          |
|          |
|          |
|          |
|          |
|          |
------------
```

flexDirection : **'row-reverse'** :

```
------------
|       XXX|
|          |
|          |
|          |
|          |
|          |
|          |
------------
```

flexDirection : **'column'** :

```
------------
|X         |
|X         |
|X         |
|          |
|          |
|          |
|          |
------------
```

flexDirection : **'column-reverse'** :

```
------------
|          |
|          |
|          |
|          |
|X         |
|X         |
|X         |
------------
```

Justify Content

```
<View style={{
  flex: 1,
  flexDirection: 'column',
  justifyContent: 'center',
  alignItems: 'space-between',
}}>
  <View style={{height: 50, backgroundColor: 'powderblue'}} />
  <View style={{height: 50, backgroundColor: 'skyblue'}} />
  <View style={{height: 50, backgroundColor: 'steelblue'}} />
</View>
```

justifyContent : '**space-between**'

```
------------
|X          |
|           |
|           |
|X          |
|           |
|           |
|X          |
------------
```

justifyContent : '**space-around**'

```
------------
|           |
|X          |
|           |
|X          |
|           |
|X          |
|           |
------------
```

justifyContent : '**flex-start**'

```
------------
|X          |
|X          |
|X          |
|           |
|           |
|           |
|           |
------------
```

justifyContent : '**center**'

```
------------
|           |
|           |
|X          |
|X          |
|X          |
|           |
|           |
------------
```

justifyContent : '**flex-end**'

```
------------
|          |
|          |
|          |
|          |
|X         |
|X         |
|X         |
------------
```

Align Items

```
<View style={{
    flex: 1,
    flexDirection: 'column',
    justifyContent: 'center',
    alignItems: 'stretch',
  }}>
    <View style={{height: 50,  backgroundColor: 'powderblue'}} />
    <View style={{height: 50, backgroundColor: 'skyblue'}} />
    <View style={{height: 50, backgroundColor: 'steelblue'}} />
  </View>
```

alignItems: '**stretch**'

```
------------
|          |
|          |
|XXXXXXXXXX|
|XXXXXXXXXX|
|XXXXXXXXXX|
|          |
|          |
------------
```

alignItems: '**flex-start**'

Attention, dans ce cas il faut donner une dimension aux carrés :

```
<View style={{
  flex: 1,
  flexDirection: 'column',
  alignItems: 'flex-start',
}}>
  <View style={{height: 50, width: 50, backgroundColor: 'powderblue'}} />
  <View style={{height: 50, width: 50, backgroundColor: 'skyblue'}} />
  <View style={{height: 50, width: 50, backgroundColor: 'steelblue'}} />
</View>
```

```
------------
|X          |
|X          |
|X          |
|           |
|           |
|           |
|           |
------------
```

alignItems: '**center**'

```
------------
|     X     |
|     X     |
|     X     |
|           |
|           |
|           |
|           |
------------
```

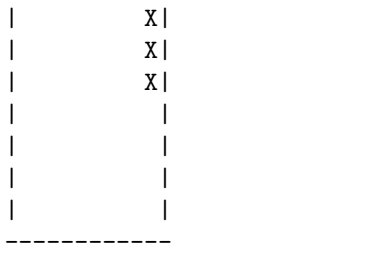
Si vous ajoutez la propriété justifyContent alors l'alignement se fera également sur l'axe verticale :

```
<View style={{
  flex: 1,
  flexDirection: 'column',
  justifyContent: 'center',
  alignItems: 'center'
}}>
  <View style={{height: 50, width: 50, backgroundColor: 'powderblue'}} />
  <View style={{height: 50, width: 50, backgroundColor: 'skyblue'}} />
  <View style={{height: 50, width: 50, backgroundColor: 'steelblue'}} />
</View>
```

```
------------
|           |
|           |
|     X     |
|     X     |
|     X     |
|           |
|           |
------------
```

alignItems: '**flex-end**'

```
------------
```



Pour plus de détails sur les flexbox vous pouvez vous reporter à la documentation sur le site de React Native : flexbox

Composant TextInput

Ce composant fait partie des composants principaux de React Native. Il vous permet de mettre en place un champ de saisi.

Exercice TextInput

Utilisez pour l'instant snack.expo pour faire l'exercice sandbox

Créez un champ centré permettant de saisir une chaine de caractères. Affichez le nombre de caractères saisis en dessus du champ, à chaque fois qu'il y a un espace compter le nombre de caractères par mot et affichez le :

[Hello World !]

5 5 1

Exercice votes

Développez l'application votes suivante, vous pouvez utiliser snack.expo. Il y a 4 couples de candidats, à chaque fois l'utilisateur doit voter pour un candidat. Une fois tous les votes effectués on affichera les résultats sur une "page" différente. Un bouton reset est prévu pour réinitialiser les votes. Utilisez l'approche fonctionnelle de React ainsi que le Hook **useReducer** pour réaliser cet exercice. Faites deux composants : App et Favorite. Vous pouvez gérer, pour l'instant, le développement de ces deux composants dans le même fichier App.js.

Pensez à utiliser la documentation officiel pour réaliser cet exercice.

Remarque sur le composant Button, ce dernier pose des problèmes de rendu (style CSS) sur iOS, on peut lui préférer une autre approche avec le composant suivant **TouchableOpacity**.

```
<View style={styles.countContainer}>
<TouchableOpacity
  style={styles.button}
  onPress={() => Alert('on Press')}
>
  <Text>Press Here</Text>
</TouchableOpacity>
</View>

const styles = StyleSheet.create({

  button: {
    alignItems: "center",
    backgroundColor: "#DDDDDD",
    padding: 10
  },
  countContainer: {
    alignItems: "center",
    padding: 10
  }
});
```

Vous utiliserez également le composant **FlatList** il permet de rendre une liste d'items, utilisez la documentation pour le mettre en place dans l'exercice :

Flatlist

- Liste des candidats :

```
 candidates: [
    { choice_1: "Alan", choice_2: "Juliette" },
    { choice_1: "Phi", choice_2: "Bernard" },
    { choice_1: "Lisa", choice_2: "Elise" },
    { choice_1: "Cecilia", choice_2: "Alice" },
  ],
```

- fig 1 composant App

```
--------------------
| Welcone          |
| React Native     |
|------------------|
|   It's Alan      |
|------------------|
|   It' Alice      |
--------------------
```

- fig 2 composant Favorite

```
--------------------
```

```
| Welcone          |
| React Native     |
|------------------|
|   1. Alan        |
|   2. Phi         |
|   3. It' Alice   |
|   4. It' Lisa    |
--------------------
|  Reset Favorites|
--------------------
```

Exercice Navigation School App 3h

Vous allez créer une application listant des étudiants avec leur nombre d'abscence(s) et les cours qu'ils suivent. Une page présentera tous les cours et une autre permettra d'administrer (ajouter) une abscence dans un premier temps. D'autres options sont à développer et détaillées dans ce qui suit.

Nous utiliserons React Native Navigation, pensez à utiliser la documentation officiel : react native navigation

Suivez les consignes ci-dessous :

1. Installez un nouveau projet et choisissez **blank** à partir de la commande expo cli :

```
expo init school
```

2. Installez ensuite les dépendances supplémentaires dans le projet school :

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

Navigation

Voici un exemple de navigation simple, lisez les commentaires avant de construire la navigation demandée dans le projet.

```
import React from 'react';
import { Button, View, Text } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';

// Construction des menus
const HomeScreen = ({ navigation })  => {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
```

```
      <Text>Home Screen</Text>
      <Button
        title="Go to Details"
        onPress={() => navigation.navigate('Details')}
      />
    </View>
  );
}

const DetailsScreen = ({ navigation }) =>{
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Details Screen</Text>
      {/*
        On navigue d'un composant à l'autre
        à l'aide de la méthode navigate de navigation
      */}
      <Button
        title="Go back..."
        onPress={() => navigation.navigate('Home')}
      />
    </View>
  );
}

// On utilise la classe createStackNavigator de React navigation
const Stack = createStackNavigator();

// Définition de la navigation pour votre application
// Notez que initialRouteName définit la page par défaut
// quand l'application se charge

// Vous devez définir un wraper NavigationContainer puis
// utilisez les composants Stack.Navigator et Stack.Screen
// pour définir les éléments de navigation
const Nav = () => {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Home">
        <Stack.Screen name="Home" component={HomeScreen} />
        <Stack.Screen name="Details" component={DetailsScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

```
const App = () => ( <Nav /> );

export default App;
```

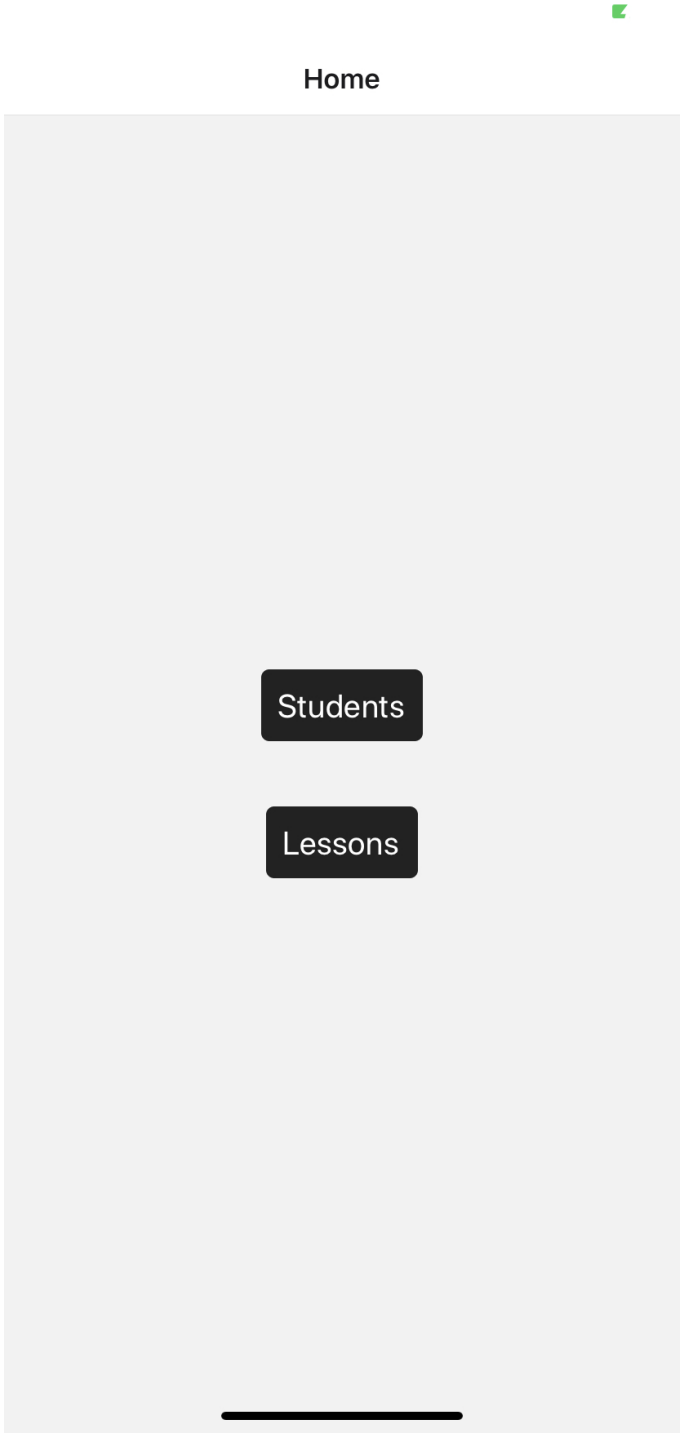
3. Création des menus de l'application et mise en place des données. Voyez ci-dessous les données à utiliser pour l'exercice.

Attention, on ne vous demande pas ici d'utiliser Redux mais les technologies suivantes :

- Création d'un contexte et d'un provider pour gérer le state.
- Utilisez useReducer pour factoriser l'algorithmique dans l'application.
- Pour garder les données initiales comme source de vérité vous allez créer une copie de l'objet initalState pour votre reducer. Attention, le spread operator ne peut pas faire une copie d'un objet trop complexe simplement. Pour copier l'objet ci-dessous nous vous proposons une solution voyez ce qui suit :

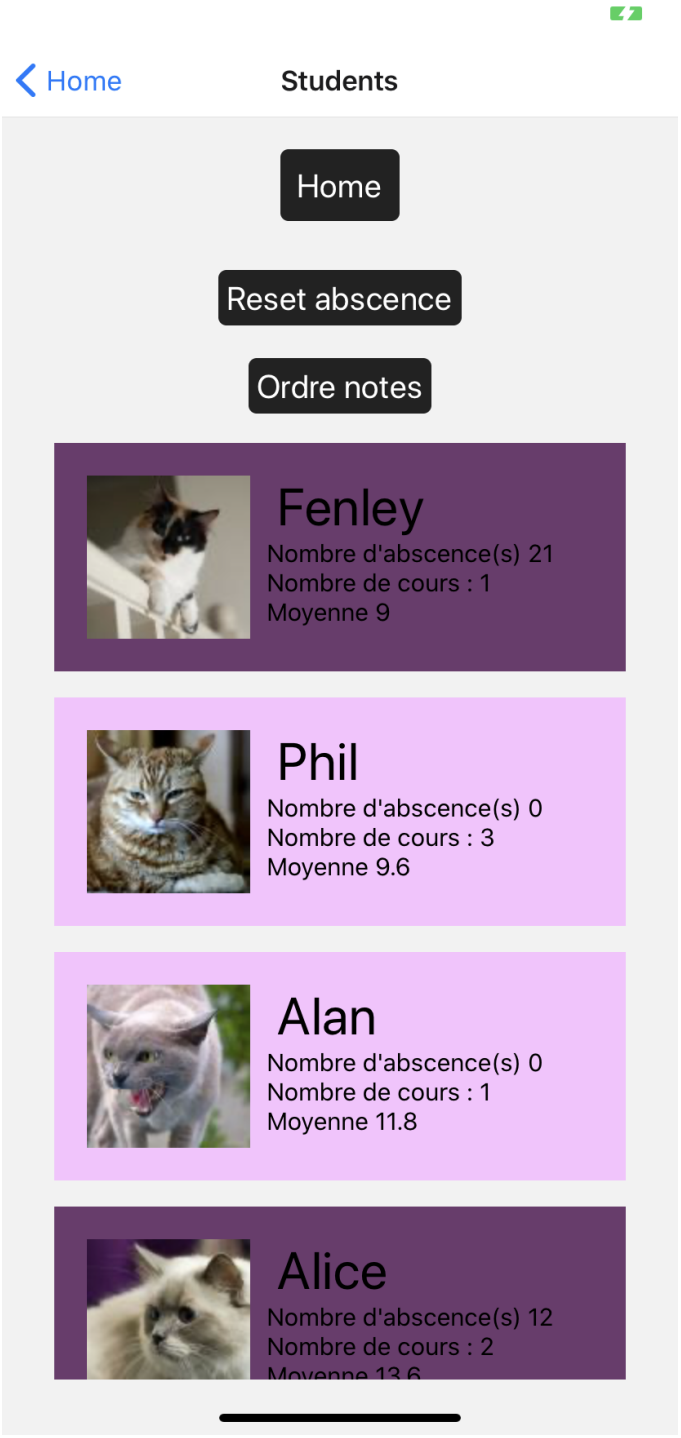
```
const initialState = {
    students: [
        { id: 1, name: "Alice", lessons: [1, 2], attendance: 0, notes: [11, 12, 18] },
        { id: 2, name: "Alan", lessons: [3], attendance: 0, notes: [10, 14.5, 11] },
        { id: 3, name: "Phil", lessons: [1, 2, 3], attendance: 0, notes: [11, 9, 9] },
        { id: 4, name: "Naoudi", lessons: [1], attendance: 0, notes: [14.5, 19, 18] },
        { id: 5, name: "Fenley", lessons: [3], attendance: 0, notes: [9, 7, 11] },
    ],
    lessons: [
        { id: 1, title: "React" },
        { id: 2, title: "React Native" },
        { id: 3, title: "MongoDB" },
    ],
    behaviours :  [],
    order: false
};

// Copie l'objet initialState dans copyInitialState
const copyInitialState = JSON.parse(JSON.stringify(initialState));
```

Voici les différentes pages à réaliser :


Home
Students
Lessons

La page Students affichera la liste des étudiants. Vous devez sur cette page afficher le nombre d'abscence, la moyenne et le nombre de cours suivis. Affichez pour chaque étudiant son avatar, voyez le code suivant pour l'implémenter dans React Native. Le composant Image est un composant de React Native. Les boutons Reset abscence et Ordre notes ne sont pas à faire pour l'instant.

```
<Image
  source={{ uri: 'http://lorempixel.com/100/100/cats/' + id }}
  style={{ width: 100, height: 100, marginRight : 10 }}
/>
```


Home
Students
Home
Reset abscence
Ordre notes
Fenley
Nombre d'abscence(s) 21
Nombre de cours : 1
Moyenne 9
Phil
Nombre d'abscence(s) 0
Nombre de cours : 3
Moyenne 9.6
Alan
Nombre d'abscence(s) 0
Nombre de cours : 1
Moyenne 11.8
Alice
Nombre d'abscence(s) 12
Nombre de cours : 2

4. Affichez maintenant l'ensemble des leçons :


Home
Lessons
Home
React
React Native
MongoDB

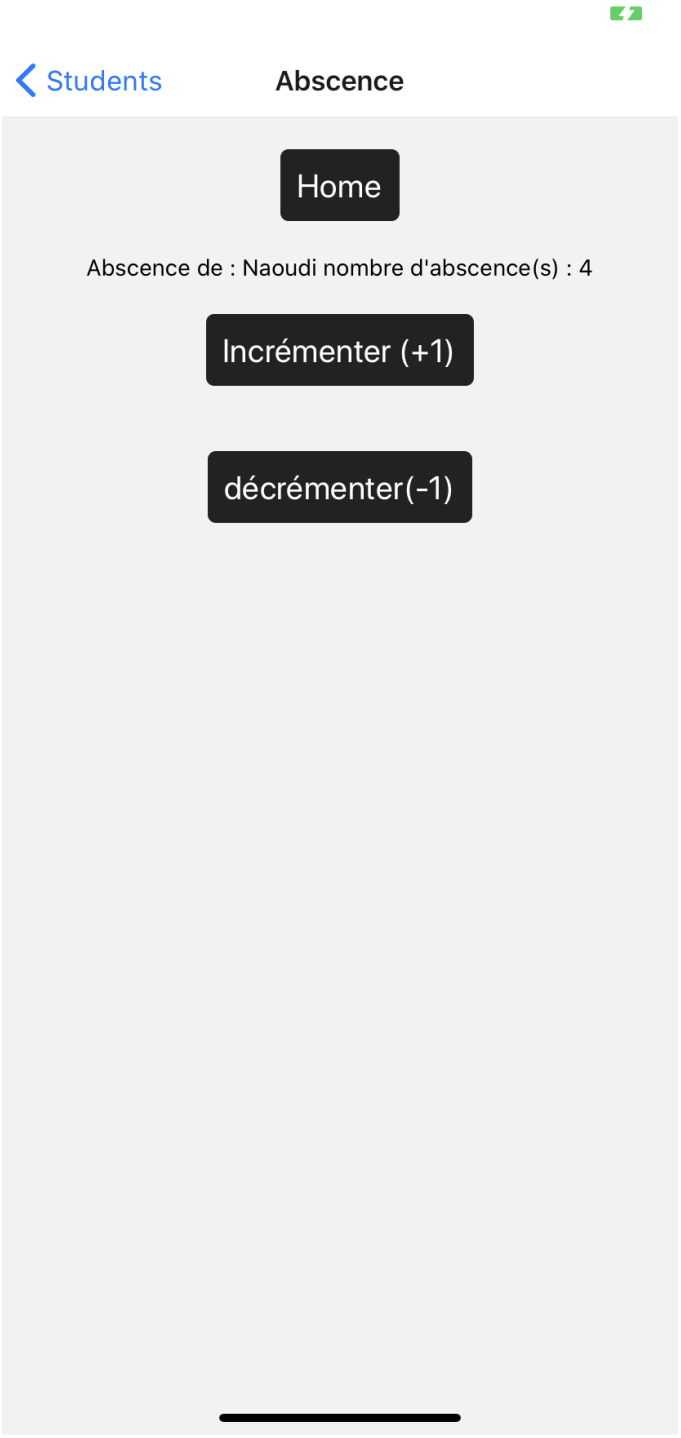
4. Administrer les abscences des étudiants.

Pour chaque étudiant, vous ferez un lien cliquable sur la page qui liste les étudiants. Une fois que l'on aura cliqué sur un étudiant on affichera la page suivante qui permettra d'incrémenter/décrémenter le nombre d'abscence(s). On bloquera la possibilité de décrémenter les abscences en dessous de 0, attendance = 0 correspondra à pas d'abscence. Dans ce dernier cas le bouton pour décrémenter les abscences ne sera pas afficher.

Si un étudiant à plus de 5 abscences vous changerez la couleur du background sur la page qui liste les étudiants (voir le wireframe précédent).

- Remarques sur la gestion du store pour travailler sur une copie de ce dernier voyez l'exemple qui suit, l'objectif n'est de ne pas changer l'initialState directement mais de travailler sur des copies.

```
// Source de vérité initialState ne doit pas muter
// const initialState = { ... }

// On crée un nouvel objet que l'on modifie
student = { ...state.students.find(s => s.id === action.id) };
student.attendance++;

// Crée un nouveau tableau
students = state.students.map( s => {
   if ( s.id != action.id ) return s;

   return student;
});

// puis on retourne le state avec un nouvel objet students
// les ...state sont nécessaires le pattern du flux est le suivant state => newState
return { ...state, students : students }
```


Students
Abscence
Home
Abscence de : Naoudi nombre d'abscence(s) : 4
Incrémenter (+1)
décrémenter(-1)

Refactorisez votre application comme suit. Créez tout d'abord une branche refactoring :

```
git checkout -b refactoring
```

Puis organiser l'application comme suit :

```
assets/
src/
  screens/
    AbscenceScreen.js
    HomeScreen.js
    StudentsScreen.js
  store/
    SchoolProvider.js
  styles.js

App.js
```

Une fois votre code refactoré créer un commit :

```
git add .
git commit -m "refactoring ok"
```

5. Implémentez un bouton sur la page Students permettant de remettre à jour le nombre d'abscence de tous les étudiants.
6. Ajoutez un bouton sur la page des étudiants permettant d'ordonner (toggle) la liste des étudiants par ordre croissant ou décroissant des moyennes.
7. Lorsqu'on clique sur un étudiant on donnera en plus de la gestion de ses abscences la possibilité du lui attribuer une appréciation au niveau de son comportement :

```
behaviours : [{ id : 1, mention : 'A'}, { id : 2, mention : 'B'}]
```

Exercice Algorithmique & Calculatrice

Installez react native navigation et organisez l'application comme suit :

Un item de menu Calculatrice et Memory respectivement affichera la calculatrice et l'ensemble des opérations de celles-ci, sous forme d'une liste ou en notation polonaise inversée. Notez que cette deuxième partie est facultative.

```
// Sous forme d'une liste
6 + 9 = 15
15 * 3 = 45
45 + 100 = 145

// Notation polonaise inversée
( ( (6 + 9) * 3) + 100 ) = 145
```

Installation et configuration du projet :

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

npm install

expo install react-native-gesture-handler react-native-reanimated
react-native-screens react-native-safe-area-context
@react-native-community/masked-view

expo install redux
expo install react-redux
```

Configuration des dossiers et fichiers :

```
components/
  TouchCalculator.js
screens/
  HomeScreen.js
  CalculatorScreen.js
  MemoryScreen.js
actions/
reducers/
actions-types/
assets/
App.js
```

Vous allez développer une calculatrice en React Native. Mettez en place l'aspect visuel de celle-ci, puis vous implémenterez ses fonctionnalités dans un second temps.

1. Rendu de la calculatrice suivante

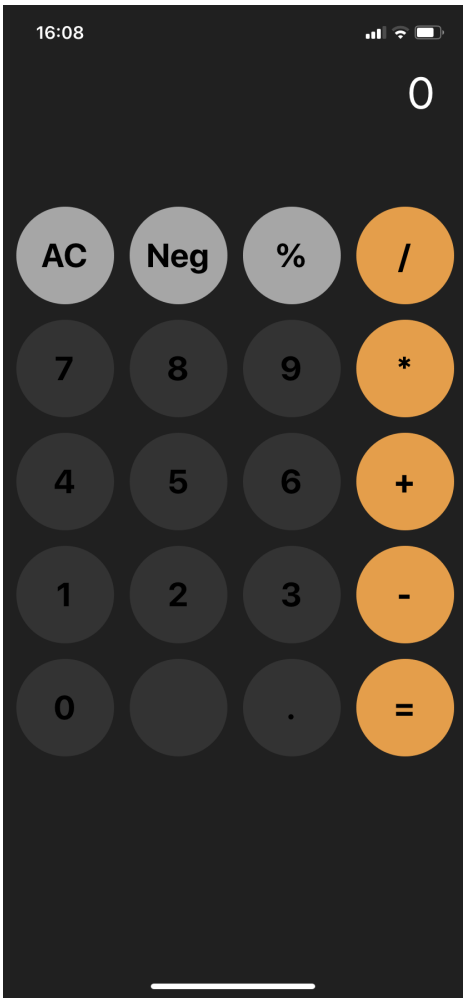


Figure 1: calculatrice

2. Organisation des dossiers et fichiers

Vous mettrez en place redux dans ce projet utilisez la structuration de dossiers et fichiers suivante pour l'application.

```
src/
  screens/
    HomeScreen.js
    CalculatorScreen.js
    MemoryScreen.js
  actions/
  reducers/
  actions-types/

assets/
App.js
```

3. Fonctionnalités de la calculatrice

- Saisi des valeurs numériques.
- Mettez en place la fonctionnalité d'effacement.
- Opérations de base lorsqu'on saisi une première valeur puis que l'on choisit un opérateur et enfin une deuxième valeur alors on affiche le résultat de l'opération sur l'écran principal.
- On peut continuer à faire des opérations, si on a rien effacer la dernier valeur est gardée en mémoire et utilisée dans les prochains calculs.
- Implémentez la fonctionnalité Neg/Pos qui ajoute un signe négatif/positif à la valeur affichée, si on ré-appuie sur ce bouton le signe change et devient positif ou négatif selon le signe de la valeur.

Partie facultative

- Développez la fonctionnalité memory, elle gardera l'ensemble des opérations en mémoire. La page Memory affichera l'ensemble des opérations effectuées par la calculatrice, soit en notation polonaise inversée, soit sous forme d'une liste :

```
// Sous forme d'une liste
6 + 9 = 15
15 * 3 = 45
45 + 100 = 145

// Notation polonaise inversée
( ( (6 + 9) * 3) + 100 ) = 145
```