Applications Mobiles
React Native

TABLE DES MATIERES

Section 1 Généralités

1.1 But du document

Ce document est le support de formation pour le développement mobile avec React Native.

Il contient des exercices qui vont permettre de découvrir cet environnement.

Le premier nomé ‘introduction’ permet la prise en main et la définition de concepts généraux.

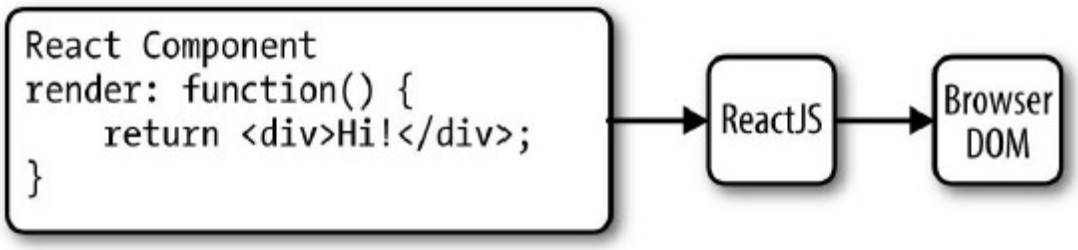
Le deuxième température converter reprend les concepts de base et présente de nouvelles fonctions.

Le 3ème ‘liste des tâches’ pourra être fait avec plus d’autonomie.

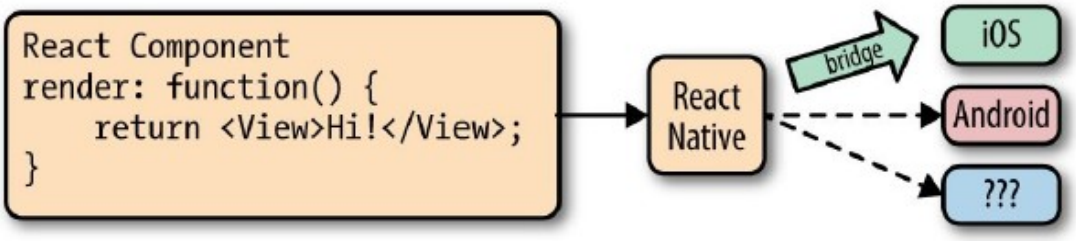
1.2 Description succincte du fonctionnement de react native

React native est basé sur React.

React gère un virtual DOM en javascript ce qui lui permet d’augmenter les performances par rapport à du code HTML CSS classique. Le ‘rendering’ du virtual DOM se fait vers le Browser DOM



React native utilise les mêmes concepts et le ‘rendering’ se fait vers l’API Objective C pour les mobiles IOS et se fait vers l’API Java pour les mobiles Android.

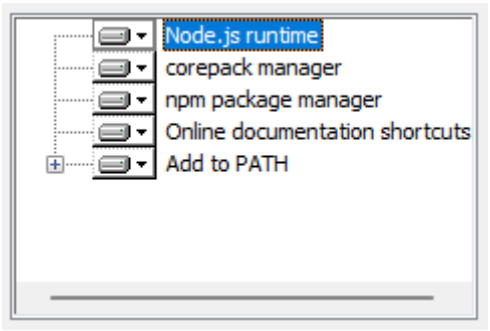


Section 2 Installation

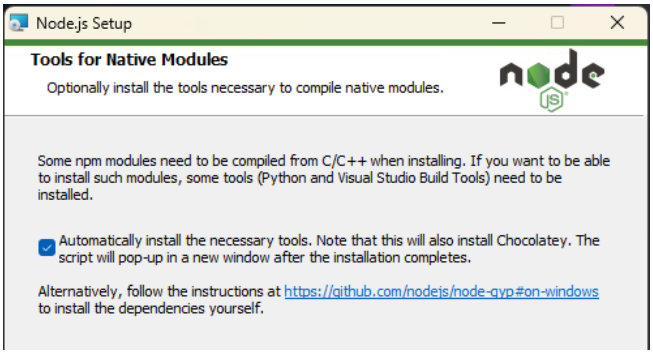
2.1 Installation de NodeJS

NodeJS est au cœur du système. Il faut l'installer.

Sur le site https://nodejs.org/en/download télécharger



Cocher comme suit :



Vérifier la bonne installation :

Ouvrir une fenêtre cmd

taper

`node -v`

`npm -v` gestionnaire de package

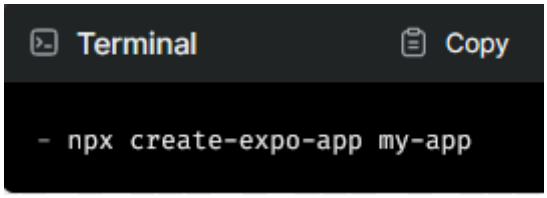
`npx -v` pour tester une librairie

Regarder le répertoire C:\Users\xxxxxx\AppData\Roaming

S'il ne contient pas le répertoire npm, le créer

2.2 Installation de EXPO

Sur le site https://docs.expo.dev/ regarder la commande Quick Start



Créer un répertoire travailCours.

Ouvrir une fenêtre cmd dans ce répertoire.

Taper la commande

```
npx create-expo-app introduction -template blank
```

il se peut que l'installation propose des mises à jour (warnings en jaune) . Executer dans ce cas

```
npm audit fix --force
```

La base du projet est créée. Se déplacer dans introduction par

```
cd introduction
npx expo start
```

(Si cette commande expo n’est pas reconnue, installer expo_cli par

```
npm install expo-cli
```

)

Ouvrir VS code par code .

Section 3 Choisir le support

Trois modes sont possibles pour exécuter l'application :

3.1 Tester sur smartphone - conseillé

Installation très facile et rapide, donc pas de potentiels problèmes de configuration.

- Donne un accès réaliste au système du téléphone comme : La géolocalisation, l'appareil photo, les photos, les fichiers etc..
- Une fois publiée sur EXPO l'application sera réellement utilisable sur ton smartphone. (même sans être publiée sur l'app store ou le play store)

3.2 Tester sur émulateur Android

Avantages :

- Pas besoin d'un vrai smartphone.
- Permet de voir le rendu sur Android.
- Fonctionne sur MAC, Linux et Windows

Inconvénients:

- Une installation longue avec de potentiels problèmes de configuration.
- Consomme pas mal de mémoire et d'espace disque.
- Donne un accès non réaliste au système du téléphone : Géolocalisation, appareil photo, etc...

3.3 Tester sur un simulateur IOS (Mac uniquement)

Avantages :

- Pas besoin d'un vrai smartphone
- Permet de voir le rendu sur Iphone.
- Installation moins complexe qu'un émulateur Android.

Inconvénients :

- Uniquement accessible pour les utilisateurs de Mac.
- Installation plus longue que sur un vrai smartphone
- Donne un accès non réaliste au système du téléphone : Géolocalisation, appareil photo, etc...
- Consomme pas mal de mémoire et d'espace disque.

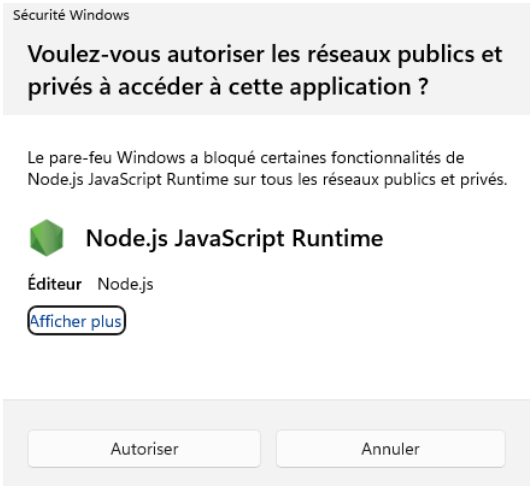
3.4 Tester sur Smartphone

Inspiré par le site https://docs.expo.dev/tutorial/create-your-first-app/

Aller sur apple store ou google play et chercher Expo Go

L'installer sur le mobile.

Accepter l'accès aux réseaux publics



Sur los utiliser l'appareil photo et scanner le Qrcode visible par la commande

`npx expo start`

précédemment tapée.

L'application doit s'afficher sur le mobile

Sur Android utiliser le lecteur de QRCode intégré à Expo Go

3.5 Tester sur émulateur Android Studio

Il faut télécharger Android Studio disponible à cette adresse

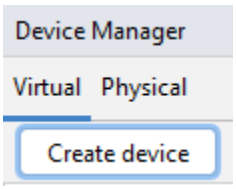
https://developer.android.com/studio

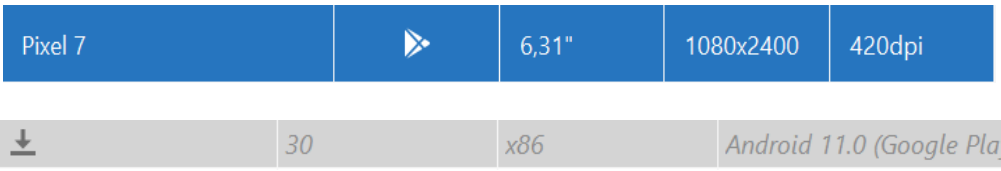
L'installer. Votre matériel doit avoir un espace disque suffisant.

Android Studio dispose d'un émulateur, gourmand en ressources, CPU et mémoire (Xamarin l'est encore plus).

Il faut configurer un mobile.

Lancer le Device manager :





Laisser le temps à l'appli de charger le composant associé (double clic sur la ligne ci-dessus)

Lancer ensuite le simulateur par le bouton

de

Attention ! Cette opération est en moyenne très longue (qques minutes), ne soyez pas impatient …

L'émulateur démarré ressemble à ceci :

Si un lancement est en cours dans une fenêtre cmd avec expo, le stopper.

Puis relancer le test par

```
npx expo start
```

Appui sur a pour open Android

L'installation de Expo go doit se faire automatiquement sur l'émulateur.

Puis l'application apparaît.

Noter qu'une modification depuis VS Code dans App.js est immédiatement prise en compte dans l'émulateur.

Section 4 Npm et npx

4.1 Npm

npm est un gestionnaire de package et de dépendance comme

- composer l'est pour Symfony
- pip pour Python
- maven et gradle en Java

https://www.npmjs.com/ est le site correspondant.

Ce site est un repository des bibliothèques disponibles.

Chercher par exemple lodash

lodash est une bibliothèque d'outils de base JS

Nous allons la charger dans le projet.

Depuis une cmd à la racine du projet (ou VS Code terminal) taper

`npm install lodash` ou `npm i lodash`

Le résultat est visible dans le fichier package.json dans la partie dependancies

La partie devDependencies existe pour les modules uniquement en développement. C'est le cas d'une biblio pour les tests : taper

```
npm i jest --save-dev
```

regarder le résultat dans package.json

pour installer une version précise :

npm i lodash@3.0.1

Installation globale d'une librairie accessible de partout option -g

exemple de l'importe quel répertoire taper :

npm i cowsay -g

cowsay "c'est fa meuhh"

npm remove cowsay -g pour enlever

Pour l'anecdote :

package.json contient la partie scripts.

Elle permet d'associer un nom à une commande

ex `"start": "expo start",`

npn run start lance la commande expo start

4.2 npx

Npx fait ce que fait npm mais de façon temporaire. Le chagement d'une librairie est temporaire le temps d'une exécution.

Section 5 Structure d'un projet React Native

5.1 Répertoires et fichiers

- .expo utile à l'outil expo. Ne pas toucher
- répertoire assets : contient les images du projet
- node_modules contient les lib du projet
- .gitignore, liste les fichiers que git doit ignorer
- app.json configure l'apparence des 3 plateformes Android Ios et Web https://docs.expo.dev/versions/latest/config/app/ fournit le contenu possible
- babel.config.js ne pas toucher
- package-lock.json, fichier généré. Contient des infos de toutes les biblios du projet

App.js est le point d'entrée de l'application.

Le réduire à ceci :

```
import { Text} from 'react-native';

export default function App() {
 return <Text>Hello !</Text>;

}
```

Text s'appelle un composant React Native

Le hello ! Doit s'afficher en haut à gauche …

Section 6 Les composants

6.1 Un 1er composant

Modifier App.js de cette façon :

```
export default function App() {
 return (
 <Text>Hello !</Text>
 <Text>Hello !</Text>
 <Text>Hello !</Text>
 <Text>Hello !</Text>
 );

}
```

VS Code signale des erreurs, le support mobile aussi.

Il faut un parent à ces composants Text

Un Ctrl ; dans VS Code ajoute un conteneur parent Wrap in JSX fragment ; ajout de <>

<View> est plus approprié. L'essayer

Créer le répertoire components

puis le sous répertoire Human

Y créer le fichier Human.jsx

Mettre ce contenu :

```
export function Human(){
  return <Text>Je suis humain</Text>
}
```

Ajouter l'import qui convient

dans App.js ajouter la balise <Human></Human>

aux Text existant.

Tester. Exliquer le fonctionnement.

6.2 Formater le code

Dans VS Code, dans les extensions chercher Prettier-Code formatter

Taper ensuit Ctrl Shift P et preferences

Section 7 Imports / export

Se référer à https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Statements/export

Quelques exemples :

Dans un fichier b.js

```
export const car = "Peugeot";
export function drive(){

}
```

Dans un fichier a.js

```
import {car,drive} from "./b"
console.log(car);
drive();
```

Variante pour b.js

```
const car = "Peugeot";
function drive(){
}
export {car,drive};
```

Autre variante avec export par défaut dans b.j

```
const car = "Peugeot";
function drive(){
}
export default car;
export {drive};
```

Dans ce cas a.j devient

```
import car, {drive} from "./b"
console.log(car);
drive();
```

Section 8 Appliquer du style

Installer un module :

npm i react-native-safe-area-context

Dans App.js remplacer <Text> par <SafeAreaProvider>

Placer en dessous un `<SafeAreaView>`

Le texte doit descendre dans l'appli

Il est possible de mettre un style directement sur un composant. Ex :

```
<Text style={{color:"red",backgroundColor:'#3300ee'}}>Hello !</Text>
```

ou

```
<SafeAreaView style={{backgroundColor:"lightblue", height:"50%"}}>
```

Il est plus productif de mettre les styles dans un fichier séparé :

créer le fichier de style contenant dy jss : app.style.js

Y mettre ceci :

```
import { StyleSheet } from "react-native"

export const sty = StyleSheet.create({
  square: {
    backgroundColor:"lightgreen",
    width:"50%",
    height:200,
  },
  rectangle:{
    // to do
  }
})
```

Une partie de App.js devient :

```
<SafeAreaView style={sty.square}>
```

importer ce qu'il faut …

Ceci permet d'éditer dans VSCode le fichier app et celui de style côte à côte …

Section 9 Communiquer avec un composant

Human est notre composant.

On souhaite lui envoyer dynamiquement un nom, prénom, âge, une voiture, un booléen, une fonction !

La balise <Human> de App.js devient :

```
<Human firstName={"John"} lastName={"Smith"} age={30}
    car={{brand:"Citroen",maxSpeed:180}}
    isHappy={true}
    doSomething={function () {
     console.log("Passage dans doSomeThing")
    }}>
```

Human.js devient

```
export function Human(props){
  console.log(props)
  return <Text style={{fontSize:20}}>Je suis {props.lastName}</Text>
}
```

Compléter pour afficher le nom, âge, marque de la voiture, vitesse max

regarder le résultat et le log de la fenêtre serveur

regarder ce qu'est « Destructuring » à cette adresse pour des objets

https://www.w3schools.com/react/react_es6_destructuring.asp

Le paramètre props peut être déstructuré et le composant Human devient

```
export function Human({firstName, lastName, age}){
```

Compléter le code.

Essayer

Puis ajouter la marque de la voiture et sa vitesse max.

9.1 La notion de children

Ajouter d'abord dans le composant Human.jsx ce code (importer Image):

```
<Image
  style={{height:200, width:300}}
  source={{uri:"https://picsum.photos/200/300"}}
  />
```

Vérifier que l'image s'affiche

Ce code peut en fait être envoyé par l'appelant App.js dans la balise <Human></Human>

Ce code est vu comme children dans Human.js

Voici ce que devient le code de App.js

```
export default function App() {
 return (
  <SafeAreaProvider>
   {/*<SafeAreaView style={{backgroundColor:"lightblue", height:"50%"}}> */}
   <SafeAreaView style={sty.square}>
    <Text style={{ color: "red", backgroundColor: "#3300ee" }}>
     Hello !
    </Text>
    <Text>Hello !</Text>
    <Human lastName={"John"} firstName={"Smith"} age={30}
    car={{brand:"Citroen",maxSpeed:180}}
    isHappy={true}
    doSomething={function () {
     console.log("Passage dans doSomeThing")
    }}>
     <Image
     style={{height:200, width:300}}
     source={{uri:"https://picsum.photos/200/300"}}
     />
    </Human>
    <Text>Hello !</Text>
    <Text>Hello !</Text>
   </SafeAreaView>
  </SafeAreaProvider>
 );
}
```

et le code de Human.jsx

```
export function Human({firstName, lastName, age, car,children}){

  return(
  <>
  <Text style={{fontSize:20}}>Je suis {lastName} {" "}
   {firstName} et j'ai {age} ans
  </Text>
  <Text style={{fontSize:20}}>
    Ma voiture est une {car.brand} elle roule au max à {car.maxSpeed}
  </Text>
  {children}
  </>);
}
```

9.2 Utiliser un booléen

L'utilisation de isHappy pour afficher la valeur dans le composant Human ne se réalise pas.

Il faut utiliser l'opérateur &&

exemple :

```
{isHappy && "bb"}              affiche bb si isHappy vaut true, rien sinon
```

On peut utiliser une opération ternaire :

```
{isHappy ? "texte si content" : "texte si pas content"}
```

Réaliser un commit git avec un libellé adapté

Section 10 Utiliser des flex box

Explorer le site : https://reactnative.dev/docs/flexbox#flex

1) Dans App.js revenir à un contenu minimal :

```
export default function App() {
 return (
  <SafeAreaProvider>
   <SafeAreaView >

   </SafeAreaView>
  </SafeAreaProvider>
 );
}
```

2) Pour voir la limite du safeAreaView, lui affecter un style background :

```
<SafeAreaView style={{backgroundColor:"red"}}>
```

Seul le bandeau est coloré.

3) En ajoutant la propriété flex:1 le SafeAreaView occupe toute la page.

4) En ajoutant une view avec flex:1 chaque partie prend 50 %

```
   <View style={{backgroundColor:"blue", flex:1}}>
   </View>
```

5) Essayer avec d'autres valeurs de flex

6) Créer ensuite un nouveau composant appelé FlexDemo (un répertoire sous components).

Y créer le fichier FlexDemo.style.js avec ce contenu :

```
import { StyleSheet } from 'react-native';

export const s = StyleSheet.create(
  {

  }
);
```

puis créer le fichier FlexDemo.jsx avec ce contenu

```
import {s} from "./FlexDemo.style";

export function FlexDemo(){
  return <></>;
```

```
}
```

Ajouter ce composant FlexDemo dans le SafeAreaView de App.js par

```
<FlexDemo />
```

7) Dans FlexDemo.style.js ajouter le style container

```
container:{
      flex:1,
      backgroundColor:"green"
    },
```

8) Dans FlexDemo.jsx ajouter à la View

```
style={s.container}
```

Regarder le résultat

9) Nous allons créer 3 carrés et organiser leurs placements

Dans FlexDemo.style.js ajouter le style Box1 de 100 x 100 et couleur de fond bleue

Idem avec Box2 et box3 avec les couleurs blanches et rouge

Dans FlexDemo.jsx ajouter dans la View existante

```
<View style={s.box1}/>
<View style={s.box2}/>
<View style={s.box3}/>
```

10) Dans FlexDemo.style.js container est le style parent. Nous allons gérer les box dans ce parent

Dans container essayer les propriétés :

flexDirection

justifyContent qui agit dans la direction de flexDirection

puis alignItems qui agit dans l'autre sens de celui de flexDirection

Faire un commit git

Section 11 Les states

Il y a 2 type de données associées à un Component : props et state.

Props sont écrit par un parent et envoyés au Component au moment de sa construction. Les informations sont statiques.

Pour des données amenées à changer il faut utiliser state.

En général on initialise un state dans un constructeur puis on change sa valeur par un setState.

Nous allons coder un exemple : sur l'appui d'un bouton une valeur va augmenter.

Créer un composant AgeCounter (le répertoire et le fichier AgeCounter.jsx)

Mettre un bouton TouchableOpacity contenant un Text « Augmenter »

Ajouter en dessous un Text `J'ai {age} ans`

Déclarer la variable age let age = 20 ;

Dans App.js remplacer l'emploi de FlexDemo par AgeCounter

Faire ensuite évoluer AgeCounter de la façon suivante :

```
export function AgeCounter(){
  let age = 30;
  function increaseAge(){
    age ++;
    console.log("dans increaseAge : ",age);
  }
  console.log(age);
  return(
  <>
  <TouchableOpacity onPress={increaseAge}>
    <Text style={{fontSize:40}}>Augmenter</Text>
  </TouchableOpacity>
  <Text  style={{fontSize:40}}>J'ai {age} ans</Text>
  </>)
}
```

Utiliser l'application.

On voit dans la console s'incréménter l'âge, mais pas dans l'application. Il manque un raffraîchissement des données

=> utiliser State

Tout composant peut définir un ou plusieurs states.

Un state a ceci de particulier que, quand on change sa valeur par un setState, le système un ré affichage par un « re render ».

Ce mécanisme est un mécanisme de React

https://www.w3schools.com/react/react_usestate.asp

Nous allons l'utiliser comme ceci

Modifier dans AgeCounter()

let age = 30 ; par useState(30)

Utiliser setAge pour incrémenter l'âge.

Tester

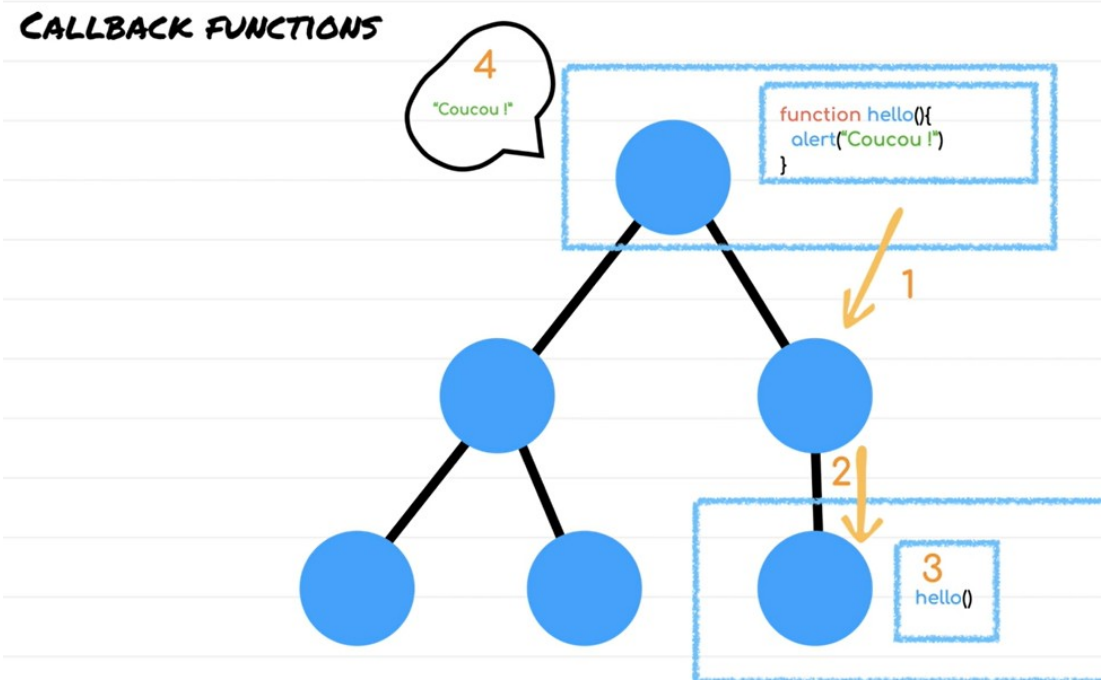
Ne jamais mettre de setXX directement dans le corps de component car boucle infinie …

faire un commit git

Section 12Les callbacks

12.1 Fonction sans paramètre

Une fonction callback est une fonction qu'un composant parent va fournir à son enfant et c'est l'enfant qui va appeler cette fonction.



Dans App.js créer la fonction suivante :

```
function hello(){
  Alert.alert("Bonjour !");
 }
```

Créer un composant Child et l'appeler dans App.js à la place du précédent AgeCounter

Dans le Child.jsx placer ce code :

```
export function Child({onPress}) {
  return(
    <>
    <TouchableOpacity onPress={onPress}>
      <Text style={{fontSize:40}}>Clic ici </Text>
    </TouchableOpacity>
    </>
  );
}
```

Vérifier le résultat.

C’est donc un code de composant qui appelle une fonction du parent.

Cette méthode peut être appliquée sur une lignée de plusieurs parents/enfants.

12.2 Fonction avec paramètre

Nous souhaitons maintenant passer de l’information dans la fonction parente.

Modifier la fonction hello() pour ajouter un paramètre name

Utiliser name dans l’alert.

Dans le composant Child il faut ajouter une fonction qui fait le relai :

```
function surClic(){
  onPress("machin");
}
```

Le TouchableOpacity doit maintenant utiliser surClic à la place de onPress.

Tester

Un commit git

Section 13 Détecter le type de mobile

Le code d'une application react Native est peu adhérent à un type de mobile.

Voici ici pour autant comment on récupère le type.

C'est simple !

La classe Platform fournit l'information.

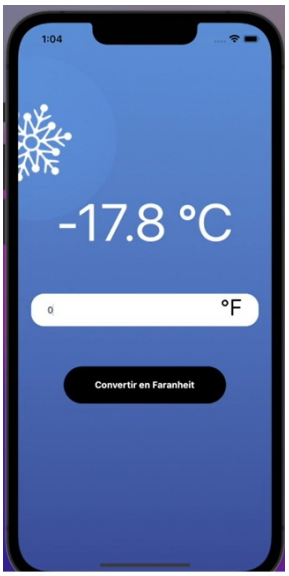
Dans App.js afficher un texte « je suis un IOS » ou « Je suis un Android » en utilisant

Platform.OS et une opération ternaire

Section 14 Application 1 : un convertisseur degré / farenheit

14.1 L'objectif

Pour mettre en application ce que l'on vient de voir une petite application qui convertit dans les deux sens une température de degré celcius en farenheit



La couleur de fond change sur un degré < 0

Une image soleil ou flocon s'affiche en fonction de degré 0

14.2 Créer le projet

Un nouveau projet est créé.

Créer un nouveau répertoire. Y lancer une fenêtre cmd.

npx create-expo-app react-native-temperature-converter

se déplacer dans le répertoire créé

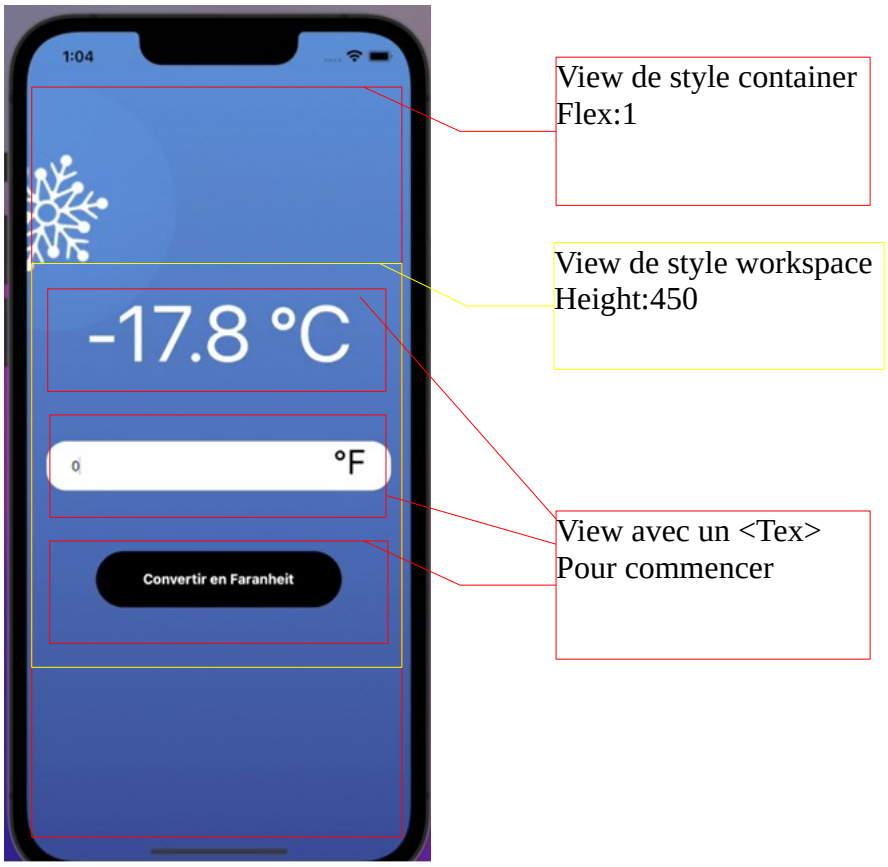
Ouvrir VSCode à cet endroit par **code .**

Lancer le serveur par

npx expo start

Créer un fichier App.style.js et y déplacer le code de styles présent dans App.js

14.3 Le layout : positionner les parties de l'appli



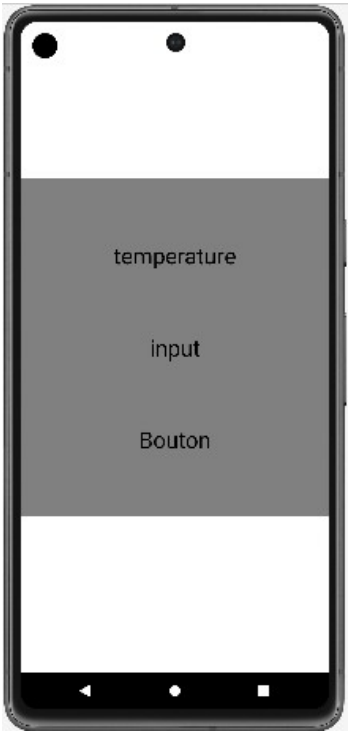
Le schéma ci-dessus représente les composants et éléments de style à créer

Dans App.style.js créer les styles :

container, workspace

Dans App.js créer l'arborescence de composants avec pour certains un style ci-dessus.

Obtenir qque chose qui ressemble à ceci :



Ajout d'une image de fond.

Plutôt que d'ajouter un composant image dans l'arborescence, nous remplaçons le 1er <View> par un <ImageBackground> l'attribut **source** décrit l'image à afficher.

Copier les images hot.png et cold.png dans les assets du projet (par un glisser/déplacer).

Dans App.js :

ajouter la ligne

```
import hotBackGround  from "./assets/hot.png"
```

Ceci donne le nom symbolique hotBackGround à l'image hot.png

Modifier le 1er <View> par un <ImageBackground> l'attribut

```
source={hotBackGround}
```

1er commit git

14.4 Le composant InputTemperature

Créer un dossier components

puis créer le composant InputTemperature avec un fichier de style séparé.

Le style contient :

```
input:{
    backgroundColor: "white",
    borderRadius:20,
    height:60,
    alignSelf: "stretch",
  }
```

Le composant InputTemperature.jsx utilise un <TextInput> avec le style décrit ci-avant.

Dans App.js remplacer le <Text> par le composant <InputTemperature>

tester

regarder le résultat, obtenir environ ceci

Faire en sorte que le clavier soit numérique et que la valeur soit limitée à 4 chiffres.

On souhaite ajouter un paramètre à InputTemperature pour fournir une valeur par défaut, attribut defaultValue

Ajout de l'unité °C en fin de l'input

Faire un commit git

14.5 Afficher la température

Créer un composant **TemperatureDisplay**

Il contient 2 paramètres : **value** et **unit**

L'appeler depuis App.js avec des valeurs en « dur » pour unit et value.

Ensuite il faut lier la valeur saisie dans l'input avec cet affichage.

Créer dans App.js **inputValue** et **currentUnit** avec un setState.

Utiliser ces valeurs pour le TemperatureDisplay.

Il faut maintenant que le InputTemperature rende l'information saisie. Il va le faire par une fonction callback.

Dans InputTemperature ajouter le paramètre **onChangeText** qui sera appelé par le onChangeText du composant <TextInput>

Dans App.js lier le onChanText à l'information inputValue.

Tester. La saisie d'une valeur doit être visible dans le Display au fur et à mesure de la saisie.

Nous allons maintenant voir l'utilisation de constantes.

Créer un fichier constant.js au même niveau que App.js

Y placer ce code :

```
export const DEFAULT_TEMPERATURE = "0";

export const UNITS = {
  celcius: "°C",
  farenheit: "°F",
};
export const DEFAULT_UNIT = UNITS.celcius;
```

Utiliser les constantes dans App.js.

Faire un commit git

14.6 Réaliser la conversion

Celsius en Farenheit : Celcius x 1,8 + 32

Farenheit en Celcius : (Farenheit -32) / 1.8

On va avoir besoin de récupérer l'unité opposée à celle courante affichée.

On va définir aussi une fonction de conversion.

Créer un dossier services et créer temperature-service.js

Créer la fonction `getOppositUnit(unit)`

qui retourne l'unité opposée. L'utiliser dans App.js

Créer la fonction `convertTemperature(unit, value)`

qui rend la valeur convertie. L'utiliser dans App.js

Tester avec différentes valeurs.

Problèmes possibles : affichage avec trop de décimales, affichage de NaN sur le caractère ','

Utiliser Number.parseFloat(), isNaN() et toFixed() pour corriger cela.

14.7 Bouton de conversion

Créer un nouveau composant ButtonConvert avec fichier de style séparé.

ButtonConvert a pour paramètre unit et myOnPress.

Il contient un <TouchableOpacity> contenant lui même un <Text>

Ce texte est Convertir en °C ou Convertir en °F selon la valeur de unit.

Créer un style 'button' pour le <TouchableOpacity> et un style 'text' pour le <Text>

Pour rendre le bouton <TouchableOpacity> sensible au clic, utiliser l'attribut onPress lié à myOnPress. MyOnPress est utilisé pour un callback.

Coder la fonction myOnPress côté App.js, elle change l'unité courante …

Un commit git

14.8 Le hook useEffect

Côté React : https://www.w3schools.com/react/react_useeffect.asp

Un essai : Créer un composant MyComponent sans fichier de style

y mettre le code suivant

```
export function MyComponent(){
  const [number1, setNumber1] = useState(0);
  const [number2, setNumber2] = useState(0);

  useEffect(()=>{
    console.log("passage dans useEffect() number1");
  },[]);
  useEffect(()=>{
    console.log("passage dans useEffect() number2");
  },[]);

  return(
    <View>
      <TouchableOpacity  onPress={()=> setNumber1(Math.random)}>
        <Text>Clique ici pour number1</Text>
      </TouchableOpacity>
      <TouchableOpacity  onPress={()=> setNumber2(Math.random)}>
        <Text>Clique ici pour number2</Text>
      </TouchableOpacity>
    </View>
  );
}
```

Regarder le résultat dans la console.

Faire en sorte que 1er useEffect() se réactive sur modification de number1.

Faire en sorte que 2ème useEffect() se réactive sur modification de number2.

Puis faire en sorte que 1er useEffect() se réactive sur modification de number1 et number2

Le useEffect peut contenir un return() qui sera appelé une seule fois au moment quand le composant disparaît. (Cf code w3schools)

14.9 Le useEffect dans l'application

Nous allons utiliser le use effect pour changer l'image de fond en fonction de la température < 0.

Dans App.js faire connaître le fichier cold.png avec le nom coldBackGround

Créer un useState pour `currentBackGround` sans valeur par défaut

Utiliser cette valeur dans App.js

Nous allons créer une fonction qui indique par un bool si la température est celle du gel.

Dans temperature-services.js créer `isIceTemperature(unit,value)`

Enfin dans App utiliser useEffect() pour réagir à la modification de inputValue. SetCurrentBackGround() est utilisé pour changer l'image de fond.

Essayer.

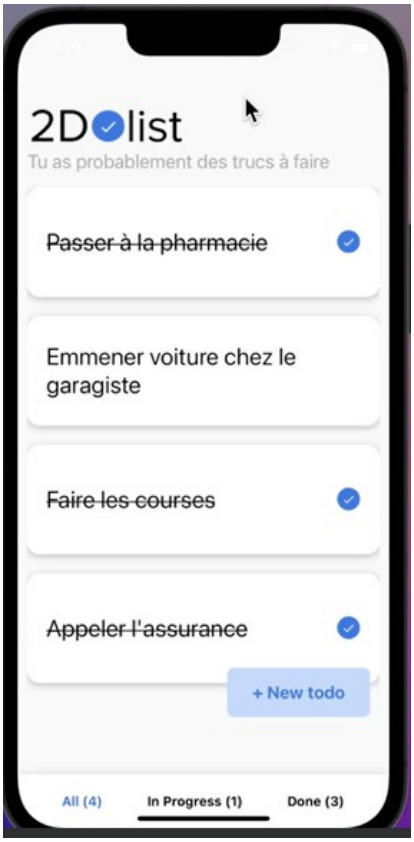
Essayer avec 2 degrés et appui sur le bouton Convertir en °C. Correct ?

Section 15Application liste de tâches

15.1 But

Nous allons mettre en pratique ce qui a été vu avant.

Cette application permet de gérer une liste de tâches de la vie courant : créer, modifier, supprimer.



15.2 Créer le projet

Créer le projet par

```
npx create-expo-app react-native-todolist
```

Nous installons une bibliothèque par

```
cd react-native-todolist
npm install react-native-safe-area-context
```

Créer une barre de titre en remplaçant le code de App par

```
<SafeAreaProvider>
   <SafeAreaView>
    <Text>Hello</Text>
   </SafeAreaView>
</SafeAreaProvider>
```

15.3 Le layout

On sépare la présentation en 3 partie : titre, corps et bas de page

Dans App, dans <SafeAreaView> , créer 3 <View>

Chaque <View> va contenir un <Text> et indiquer ‘header’, ‘body’ et ‘footer’

Chaque <View> a aussi un style nommé ‘header’, ‘body’ et ‘footer’

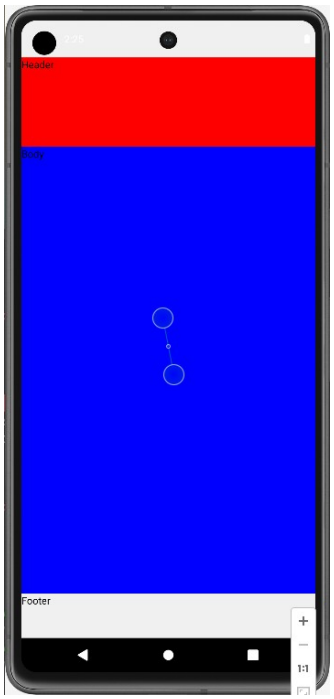
Créer un fichier de style App.style.js et créer les 3 styles.

Créer un stype ‘app’ pour le <SafeAreaView> , lui affecter la couleur ”#f0f0f0” et faire en sorte qu’il occupe toute la page (un flex …)

Faire en sorte que le header occupe 1 part, le body 5 parts et le footer 0,5 part (flex)

leur donner des couleurs différentes.

On doit obtenir ceci :



Faire un commit

15.4 Le header

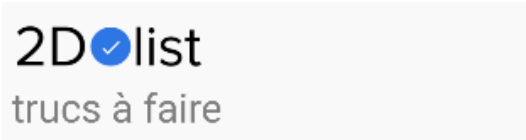
Nous allons créer plusieurs composants dans l'application.

Créer le dossier components.

Si vous le jugez utilise, créer un composant Empty avec son style. Y mettre le minimum. Nous l'utiliserons par coper coller pour d'autres vrais composants.

Créer le composant Header avec son style.

Il devra ressembler à ceci :



Récupérer les images logo.png, check.png et splash.png et les placer dans assets.

Le header va contenir une <Image> et un <Text>

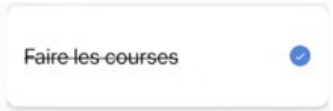
L'image aura une largeur de 150, le texte une taille de 30.

15.5 Le Card

https://reactnativeelements.com/docs/2.3.2/card

Une Card ressemble graphiquement à une carte de visite.

Dans l'appli chaque tâche est dans une Card. Elle contient un texte et une icône



Créer un composant CardToDo({toDo})

Il contient un `<TouchableOpacity>`

contenant un <Text> et <Image> . L'image est check.png.

Le paramètre toDo va contenir un objet

Placer au dessus de App() le code suivant qui simule pour l'instant une liste de tâches :

```
const TODO_LIST = [
 { id: 1, title: "Sortir le chien", isCompleted: true },
 { id: 2, title: "Aller chez le garagiste", isCompleted: false },
 { id: 3, title: "Faire les courses", isCompleted: true },
 { id: 4, title: "Appeler le vétérinaire", isCompleted: true },
 { id: 5, title: "Sortir le chien", isCompleted: true },
 { id: 6, title: "Aller chez le garagiste", isCompleted: false },
 { id: 7, title: "Faire les courses", isCompleted: true },
 { id: 8, title: "Appeler le vétérinaire", isCompleted: true },
];
```

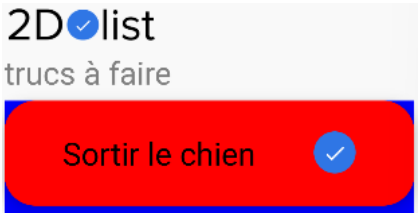
Utiliser le CardToDo dans App.de la façon suivante pour appeler la 1ère tâche de la liste :

```
<CardToDo toDo={TODO_LIST[0]} />
```

on obtient graphiquement ceci :

Créer des styles pour la Card nommé card, pour texte <Text> et pour l'image.

Obtenir ceci (les couleurs seront revues à la fin) :



Vous aurez besoin de régler :

```
backgroundColor:,
    display:"flex",
    flexDirection: ,
    justifyContent:,
    borderRadius:,
    alignItems:,
    height:,
```

A une card on associe de l'ombre pour détacher la carte du plan de l'image, une élévation.

Aller sur le site https://ethercreative.github.io/react-native-shadow-generator/

Avec la réglette choisir la quantité d'ombre et récupérer le code à place dans le style 'card'

Autre point : l'image check doit s'afficher uniquement si le toDo.isCompleted est à true.

La ligne de <Image> de CardToDo devient :

```
{toDo.isCompleted && <Image source={checkImg} style={st.img}/>}
```

Le && fait que la partie droite est évaluée si la partie gauche vaut true.

De même le texte doit être barré si isCompleted est à true.

La ligne de texte devient :

```
<Text style={[st.text, toDo.isCompleted && {textDecorationLine:"line-
through"}]}>{toDo.title}</Text>
```

15.6 L'utilisation de map sur une liste

Correspond au mécanisme de « comprehension' en Python.

Regarder :

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map

15.7 La liste des tâches

La liste actuelle TODO_LIST est statique. Il faut la mettre dans un useState comme ceci :

```
const [toDoList, setToDoList] = useState ([
  { id: 1, title: "Sortir le chien", isCompleted: false },
  { id: 2, title: "Aller chez le garagiste", isCompleted: false },
  { id: 3, title: "Faire les courses", isCompleted: true },
  { id: 4, title: "Appeler le vétérinaire", isCompleted: true },
  { id: 5, title: "Sortir le chien", isCompleted: true },
  { id: 6, title: "Aller chez le garagiste", isCompleted: false },
  { id: 7, title: "Faire les courses", isCompleted: true },
  { id: 8, title: "Appeler le vétérinaire", isCompleted: true },
 ]);
```

Il faut créer un Card par tâche. Nous utilisons une fonction qui réalise un map :

```
function renderToDoList(){
  return toDoList.map((todo)=>( <CardToDo toDo={todo} />));
 }
```

Utiliser cette fonction à la place du `<CardToDo toDo={TODO_LIST[0]} />`

Tester.

Mettre des padding, margin … pour obtenir environ ceci :

Il faut ajouter un scroll à la page : encadrer `{renderToDoList()}`

par un <ScrollView>

15.8 Utilisation de find() sur une liste

Regarder https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find

15.9 Mise à jour d'une tâche

Lorsque l'on clique sur une tâche, le texte est barré et l'icône disparaît.

Sur un clic d'une Card il faut retrouver son index et inverser le isCompleted.

Créer une fonction updateTodo() qui va être un callback pour le composant Card. Voici un contenu minimal :

```
function updateTodo(todo){
 console.log(todo);
}
```

L'appel du Card dans App se fait ainsi :

```
<CardToDo toDo={todo} onPress={updateTodo}/>
```

Et le code du Card devient :

```
export function CardToDo({toDo, onPress}) {
  //console.log(toDo);
  return(
    <TouchableOpacity style={st.card} onPress={()=>onPress(toDo)}>
      <Text style={[st.text, toDo.isCompleted && {textDecorationLine:"line-
through"}]}>{toDo.title}</Text>
      {!toDo.isCompleted && <Image source={checkImg} style={st.img}/>}
    </TouchableOpacity>
  );
}
```

Voici le code de updateTodo, noter l'utilisation de ' ...todo':

```
 function updateTodo(todo){
  console.log("avant " , todo);
  const updatedTodo = { ...todo, isCompleted: !todo.isCompleted};
  console.log("apres " , updatedTodo);
  const indexToUpdate = toDoList.findIndex((todo)=>todo.id ===
updatedTodo.id);
  console.log("index=",indexToUpdate);
  console.log("indexdirect=",updatedTodo.id -1); // fournit plus facilement
l'index ...
  const updatedTodoList = [...toDoList];
  updatedTodoList[indexToUpdate] = updatedTodo;
  setToDoList(updatedTodoList);
 }
```

15.10 Usage de Reduce

Regarder ici https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce

15.11 Ajout d'un menu

Dans le footer de l'appli on souhaite afficher ce menu:

Le clic sur un de ces éléments :

- le fait passer en bleu
- réalise le filtre des tâches

Créer le composant TabBottonMenu avec son fichier de style.

Voici le code jsx :

```
export function TabBottomMenu({ selectedTabName, onPress }) {
  function getTextStyle(tabName) {
   return {
    fontWeight: "bold",
    color: tabName === selectedTabName ? "#2F76E5" : "black",
   };
  }
  return (
   <View style={st.container}>
    <TouchableOpacity onPress={() => onPress("all")}>
     <Text style={getTextStyle("all")}>All</Text>
    </TouchableOpacity>
    <TouchableOpacity onPress={() => onPress("inProgress")}>
     <Text style={getTextStyle("inProgress")}>In progress</Text>
    </TouchableOpacity>
    <TouchableOpacity onPress={() => onPress("done")}>
     <Text style={getTextStyle("done")}>Done</Text>
    </TouchableOpacity>
   </View>
  );
 }
```

App appelle ce composant de cette façon :

```
<TabBottomMenu onPress={setSelectedTabName}
selectedTabName={selectedTabName}/>
```

15.12 Usage de filter

Regarder https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter

15.13 Filtrer les tâches

Les tâches sont filtrées en fonction du choix du menu.

Les items du menu sont suivis d'un nombre de tâches correspondantes.

Dans TabBottonMenu mettre la fonction suivante et la comprendre :

```
export function TabBottomMenu({ selectedTabName, onPress, todoList }) {
 //console.log(todoList);
 const countByStatus = todoList.reduce(
  (acc,unTodo) => {
   unTodo.isCompleted ? acc.done ++ : acc.inProgress ++;
   return acc;
  },
  {all: todoList.length, inProgress: 0, done:0}
 );
console.log(countByStatus);
```

L'utiliser dans la partie JSX :

```
return (
   <View style={st.container}>
    <TouchableOpacity onPress={() => onPress("all")}>
     <Text style={getTextStyle("all")}>All({countByStatus.all})</Text>
    </TouchableOpacity>
    <TouchableOpacity onPress={() => onPress("inProgress")}>
     <Text style={getTextStyle("inProgress")}>In
progress({countByStatus.inProgress})</Text>
    </TouchableOpacity>
    <TouchableOpacity onPress={() => onPress("done")}>
     <Text style={getTextStyle("done")}>Done({countByStatus.done})</Text>
    </TouchableOpacity>
   </View>
  );
```

Réaliser maintenant le filtrage

Dans App ajouter une fonction de filtrage

```
function getFilteredList() {
 switch (selectedTabName) {
  case "all":
   return toDoList;
  case "inProgress":
   return toDoList.filter((todo) => !todo.isCompleted);
  case "done":
   return toDoList.filter((todo) => todo.isCompleted);
 }
}
```

Utiliser ensuite `getFilteredList` dans `renderToDoList`

Tester les items du menu

15.14 Supprimer une tâche

Un clic long (onLongPress) sur un Card doit supprimer la tâche.

Dans App créer une fonction :

```
function deleteTodo(todoTodelete){
 Alert.alert("Suppression","Supprimer cette tâche ?",
 [{
  text: "Supprimer",
  style: "destructive",
  onPress:()=>{},
 },
 {
  text: "Annuler",
  style:"cancel",
 }])
}
```

Ajouter `onLongPress={deleteTodo}`

à l'appel de CardToDo

Modifier CardToDo

Essayer et constater l'affichage de l'alert.

Pour supprimer l'élément de la liste compléter le code onPress() de deleteTodo.

Utiliser un filter et setToDoList …

15.15 Ajouter une tâche

Créer un composant BoutonAdd

contient un <TouchableOpacity> avec un <Text>

BoutonAdd a un style position :" absolute"

Il existe un onPress qui permet d'appeler un callback d'ouverture d'une fenêtre de dialogue pour saisir la nouvelle tâche …

Pour la fenêtre de dialogue chercher des articles du genre

react native dialog popup

Installer la bibliothèque par

npm install react-native-dialog

installer aussi

```
npm install react-native-uuid
```

Ajouter le code suivant dans APP, après la balise </SafeAreaProvider>

```
   <Dialog.Container
    visible={isAddDialogVisible}
    onBackdropPress={() => setIsAddDialogVisible(false)}
   >
    <Dialog.Title>Créer une tâche</Dialog.Title>
    <Dialog.Description>
     Choisi un nom pour la nouvelle tâche
    </Dialog.Description>
    <Dialog.Input onChangeText={setInputValue} />
    <Dialog.Button
     disabled={inputValue.trim().length === 0}
     label="Créer"
     onPress={addTodo}
    />
   </Dialog.Container>
```

ceci nécessite une variable dynamique

```
const [isAddDialogVisible, setIsAddDialogVisible] = useState(false);
```

et les fonctions de App

```
function showAddDialog() {
  setIsAddDialogVisible(true);
 }
 function addTodo() {
  const newTodo = {
   id: uuid.v4(),
   title: inputValue,
   isCompleted: false,
  };

  setToDoList([...toDoList, newTodo]);
  setIsAddDialogVisible(false);
 }
```

15.16 Donner de la persistance aux données

Comment sauvegarder les tâches sur le mobile ?

Nous allons utiliser `react-native-async-storage`

https://react-native-async-storage.github.io/async-storage/docs/install/

commande :

```
npm install @react-native-async-storage/async-storage
```

On ne peut stocker que du texte.

Les objets doivent alors être transformés en texte. Nous choisissons un format JSON.

Créer une fonction dans App pour sauvegarder les données.

```
async function saveTodoList() {
 console.log("SAVE");
 try {
  await AsyncStorage.setItem("@todolist", JSON.stringify(toDoList));
 } catch (err) {
  alert("Erreur " + err);
 }
}
```

Créer une fonction pour charger les données

```
async function loadTodoList() {
 console.log("LOAD");
 try {
  const stringifiedTodoList = await AsyncStorage.getItem("@todolist");
  if (stringifiedTodoList !== null) {
   const parsedTodoList = JSON.parse(stringifiedTodoList);
   isLoadUpdate = true;
   setToDoList(parsedTodoList);
  }
 } catch (err) {
  alert("Erreur " + err);
 }
}
```

Créer au dessus de App() les variables

```
let isFirstRender = true;
let isLoadUpdate = false;
```

tester ...

Section 16 Publication d'une application sur Expo

Créer un compte sur expo

Créer ensuite un projet. La création sur Expo nous propose des lignes de commande pour obtenir un client EAS

```
└➤  npm install --global eas-cli && \
eas init --id 82f69134-8936-4589-9292-691035468213
```

EAS : https://docs.expo.dev/eas-update/migrate-from-classic-updates/

faire

eas login

eas update configure

créer un fichier eas.json avec un contenu visible sur le site.

Eas update development –message "premier push"

Suite à cela sur le site expo un clic sur preview ouvre un QRCode qui permet de charger l'application sur un support mobile.