**1)Préparation de l’environnent de développement**

*Avant de commencer, assurer vous d’avoir node version récente installé (V 14 +)*

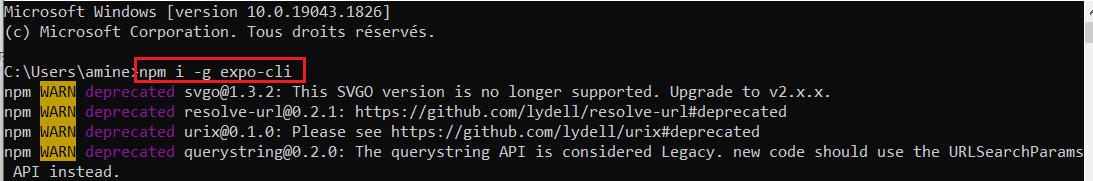


<https://reactnative.dev/docs/environment-setup>

Nous allons utiliser l’outil expo go permettant d’accélérer la création d’applications react native et présentant plusieurs services tel que le déploiement de notre application et la génération du code native.

<https://expo.dev/>

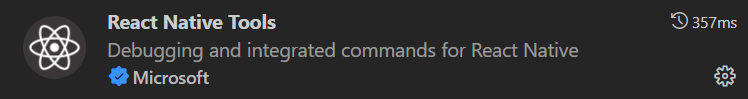
cmd1) : npm i -g expo-cli

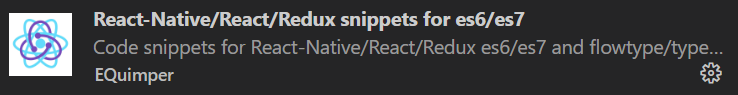


Ensuite on peut installer les applications expo-client sur notre smart phone

On peut également enrichir notre IDE VSCode avec quelques extensions

* react native tools,
* react/react native

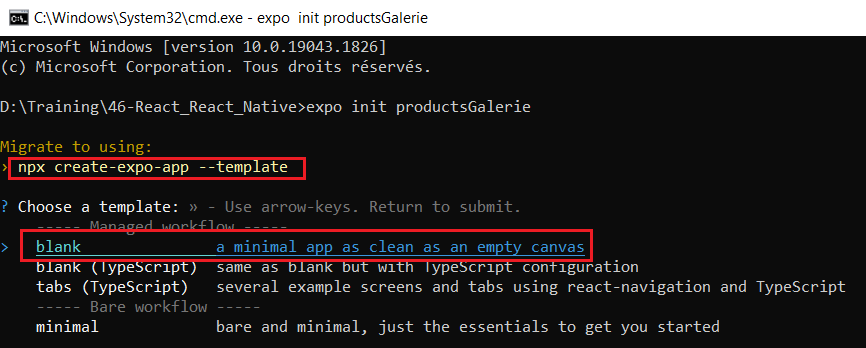


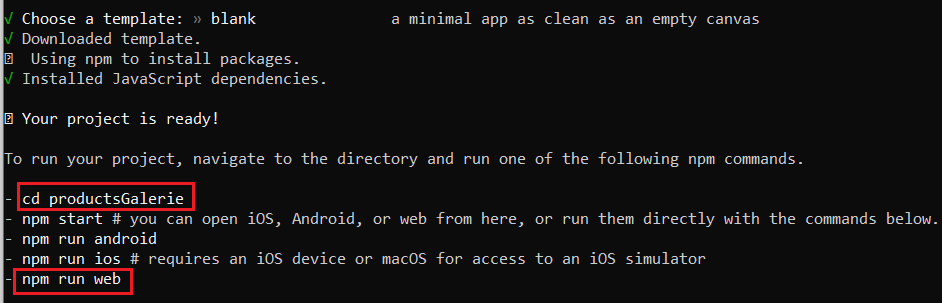


Créons un nouveau projet :

La commande est :

cmd2) expo init productsGalerie





cd to your project, then open it with vs code

Par défaut notre projet est centralisé sur un fichier App.js avec le code suivant :

import { StatusBar } from 'expo-status-bar';

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

export default function App() {

  return (

    <View style={styles.container}>

      <Text>Open up App.js to start working on your app!</Text>

      <StatusBar style="auto" />

    </View>

  );

}

const styles = StyleSheet.create({

  container: {

    flex: 1,

    backgroundColor: '#fff',

    alignItems: 'center',

    justifyContent: 'center',

  },

});

Commençons par installer quelques dépendances nécessaires pour le lancement de l’application sur un navigateur web:

Cmd 3) : npx expo install [react-native-web@~0.18.7](mailto:react-native-web@~0.18.7)

Cmd 4) : npx expo install [react-dom@18.0.0](mailto:react-dom@18.0.0)

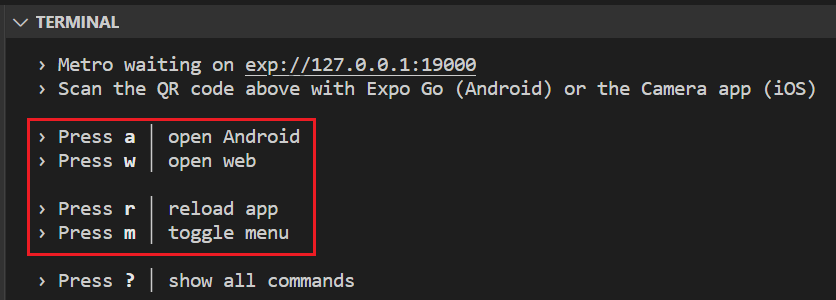
Cmd 5) : npx expo install @expo/webpack-config@^0.17.0

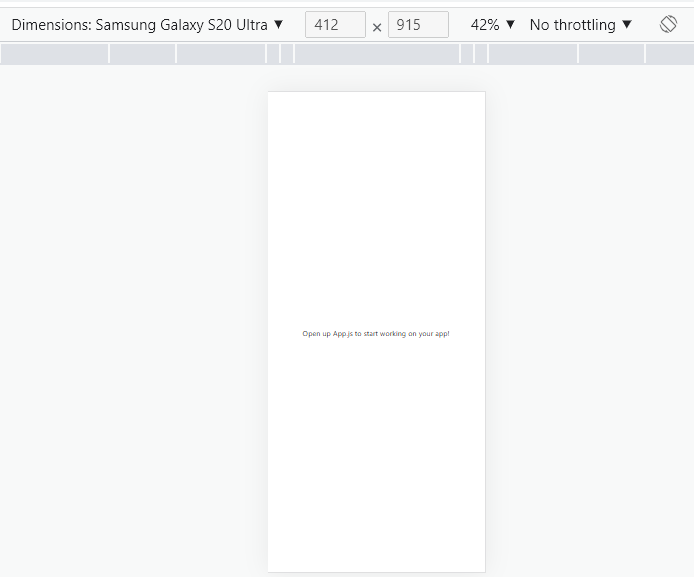
Ou en une seule commande :

Cmd 6) npx expo install react-native-web@~0.18.7 react-dom@18.0.0 @expo/webpack-config@^0.17.0

Ensuite lançons l’application via la commande suivante :

Cmd 7): npm start

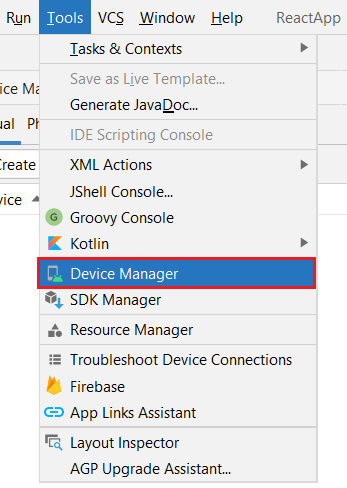




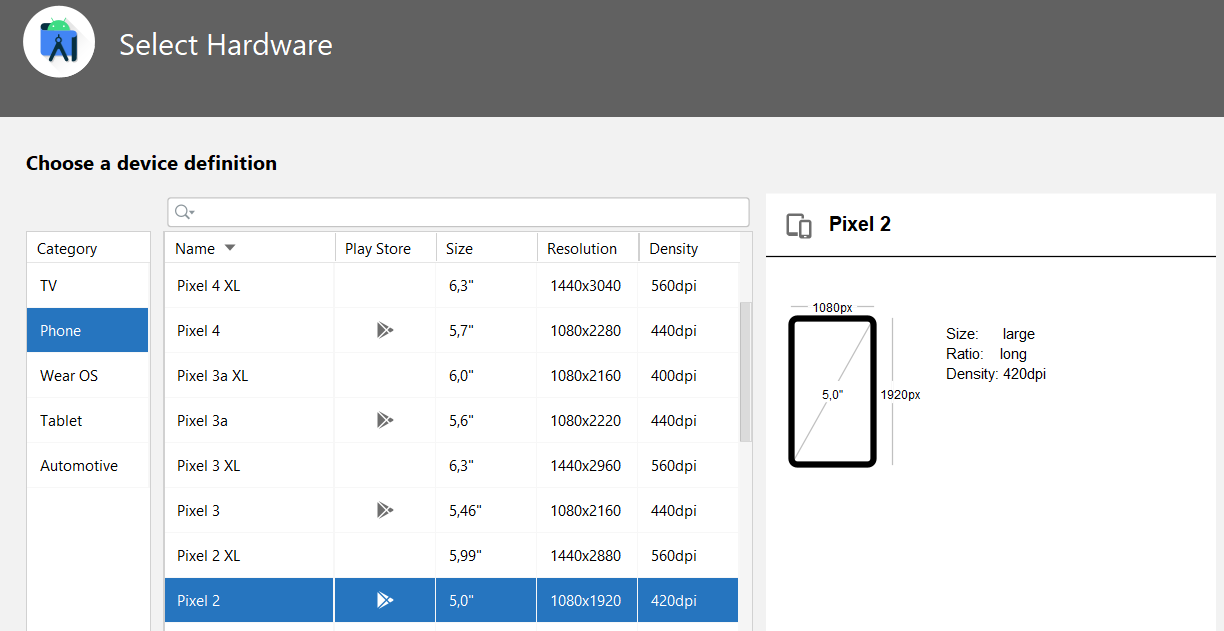
On peut également configurer sous Android Studio un émulateur Andoid

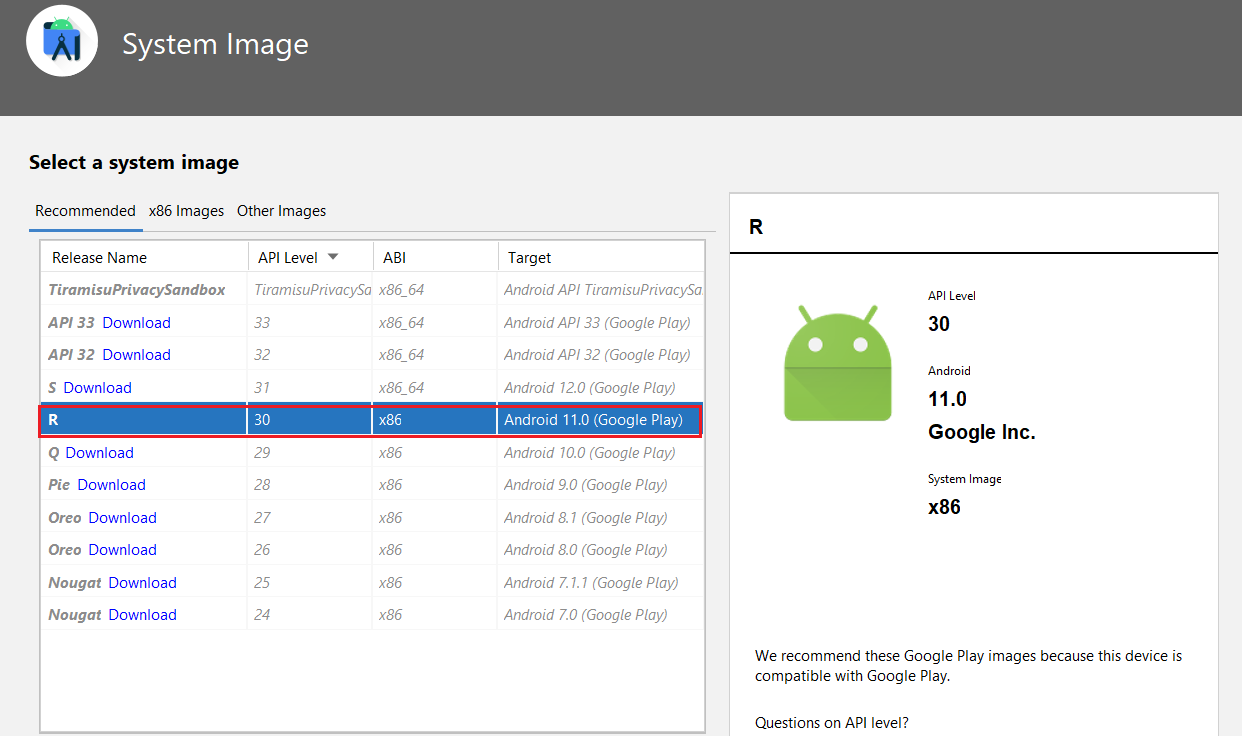
1)Lancer Andoid Studio

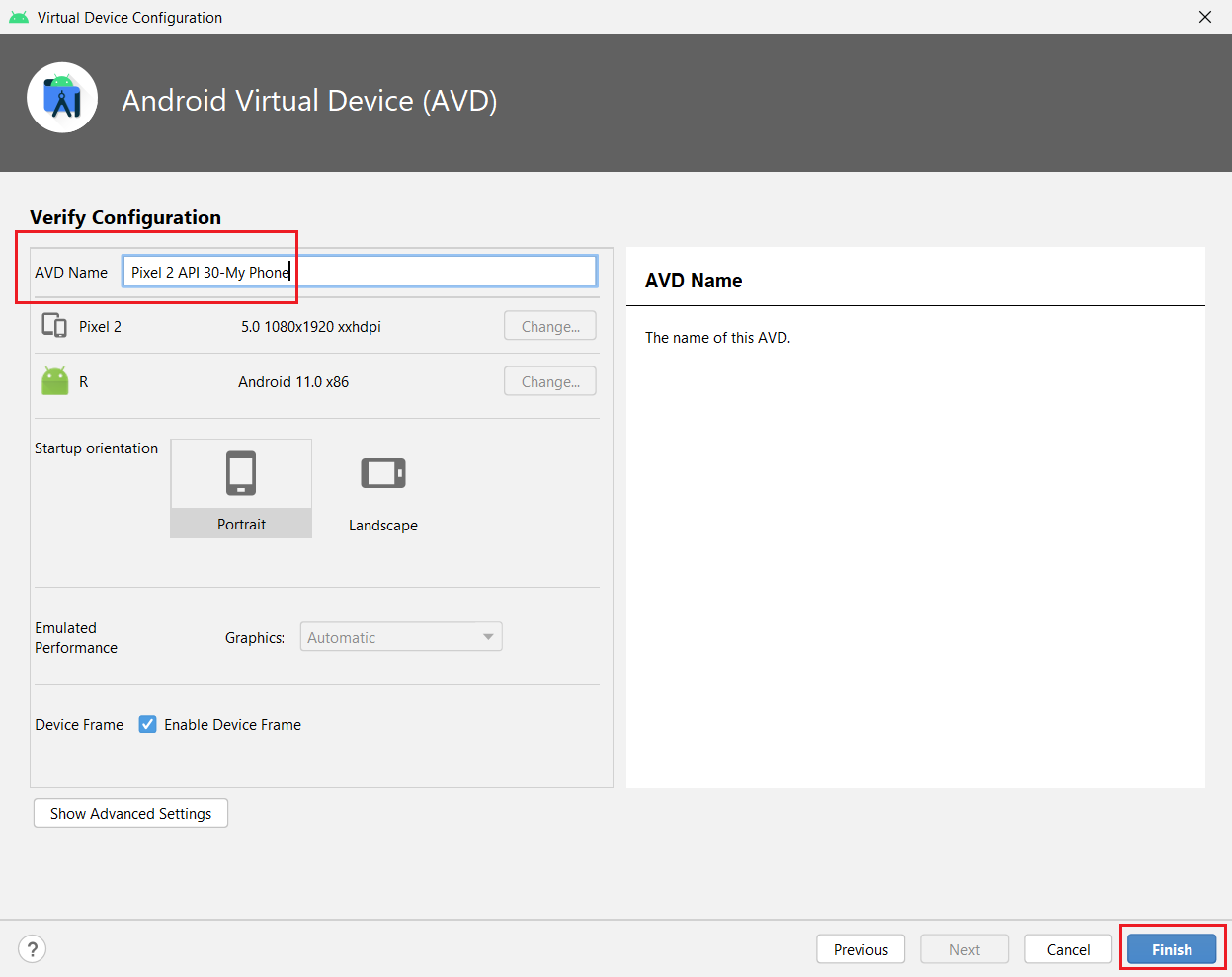
2)Tools

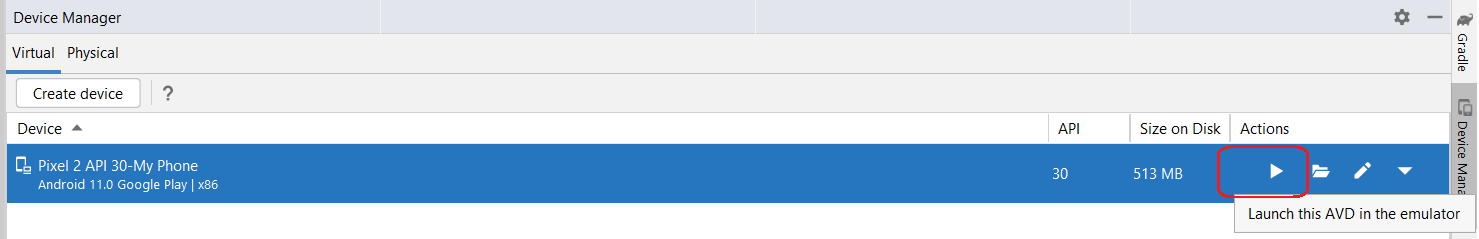


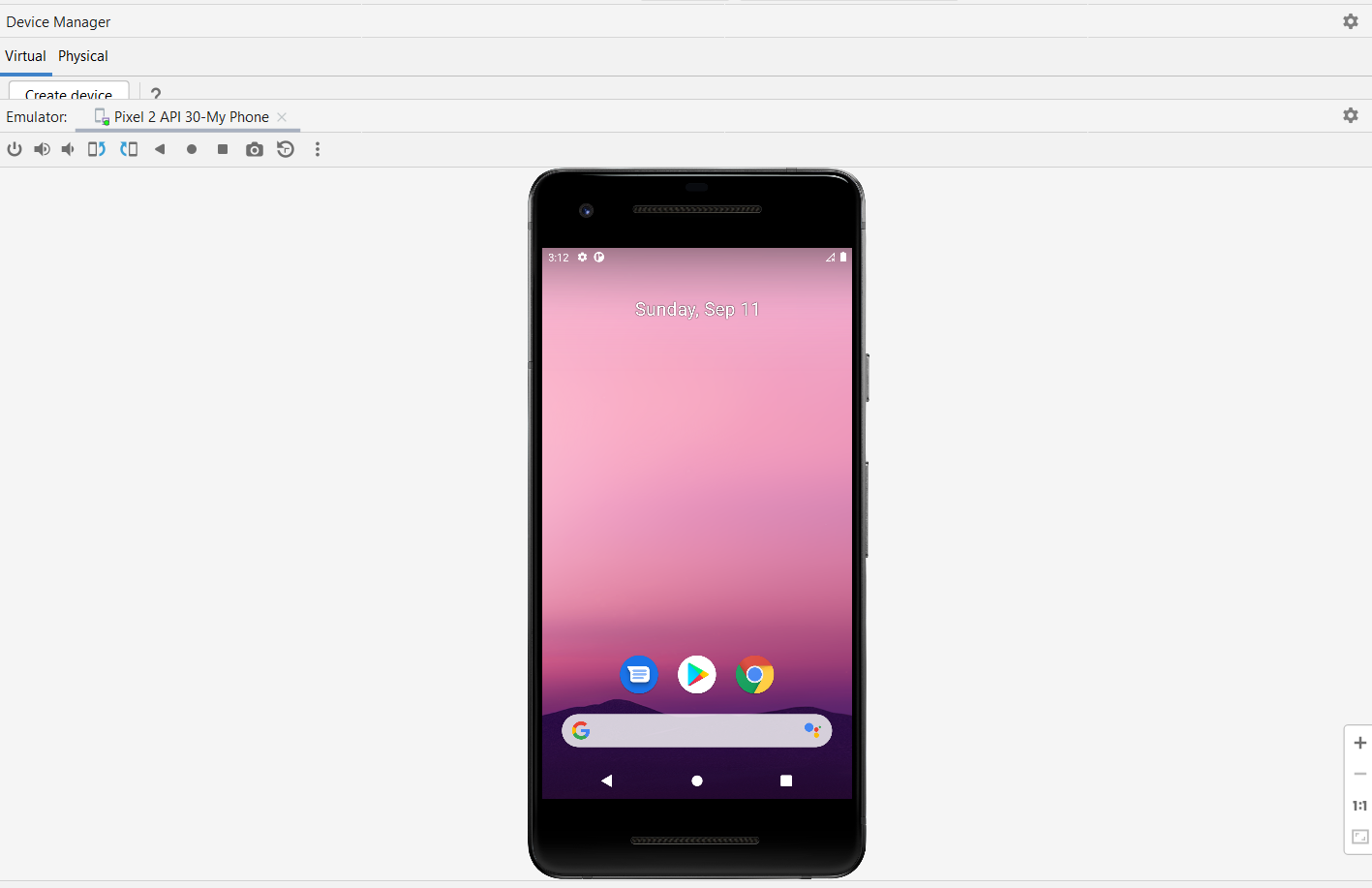
3)Choisir une machine



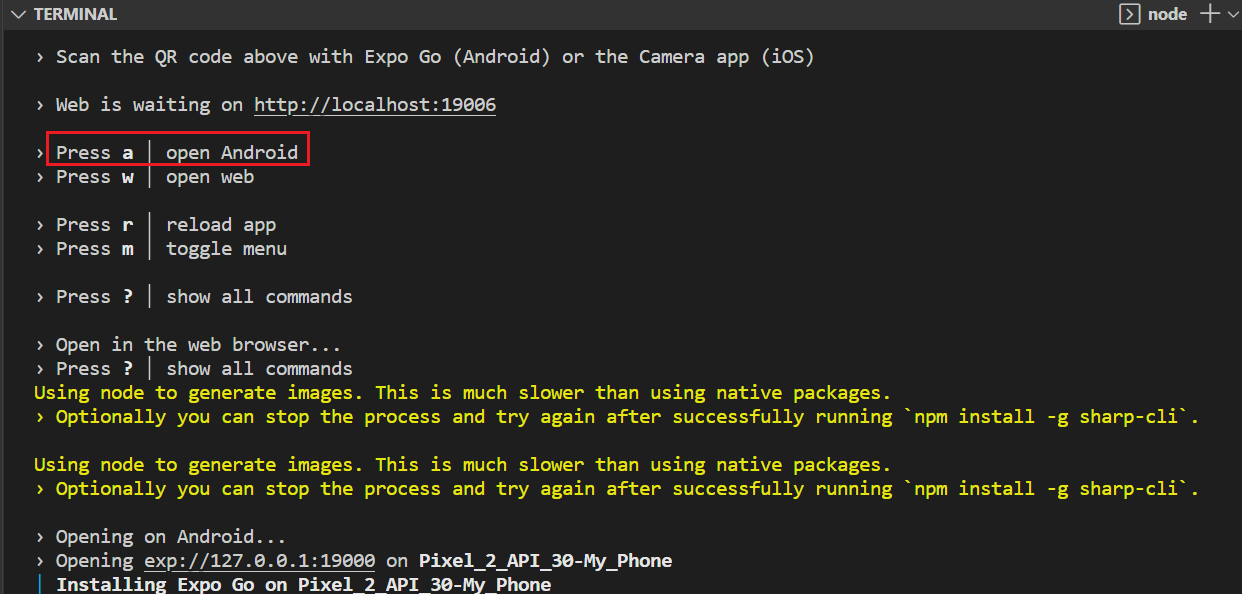


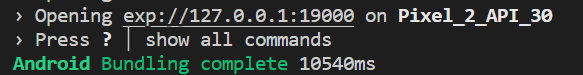






Maintenant au niveau de la console, si on clique sur « a » on va lancer l’application sur notre émulateur





    <View style={styles.container}>

      <Text>Hello World!</Text>

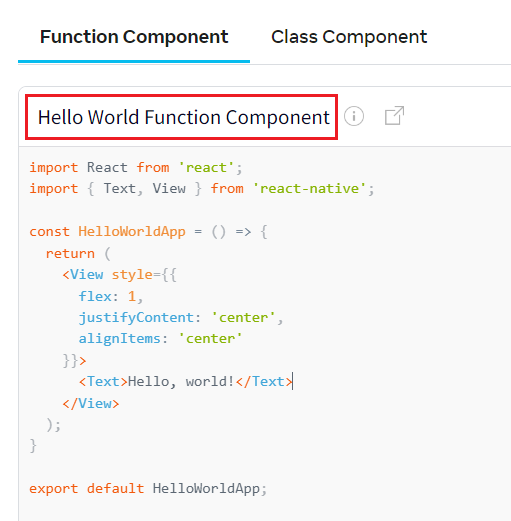
      <StatusBar style="auto" />

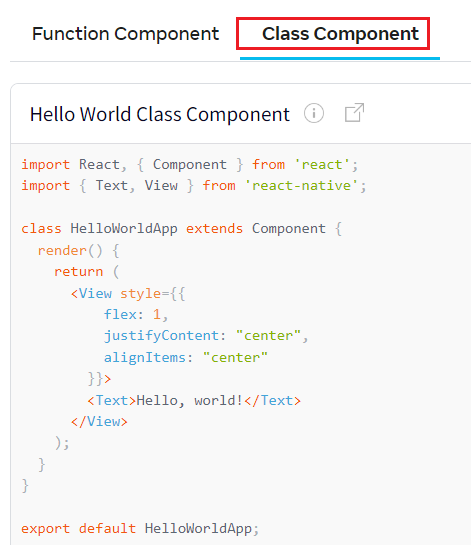
    </View>



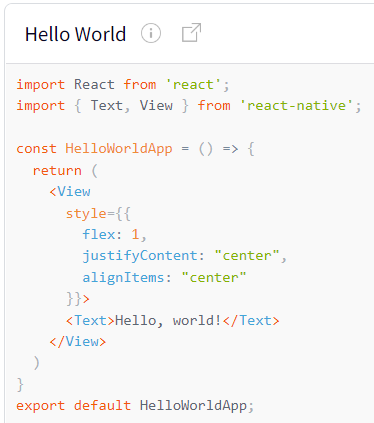
## Function Components and Class Components[​](https://reactnative.dev/docs/getting-started#function-components-and-class-components)

With React, you can make ***components*** using either classes or functions. Originally, class components were the only components that could have state. But since the introduction of React's ***Hooks*** API, you can add state and more to function components.





Let’s discuss about the following code



1. First of all, we need to import React to be able to use JSX, which will then be transformed to the native components of each platform.
2. On line 2, we import the Text and View components from react-native

Then we define the HelloWorldApp function, which is a [functional component](https://reactjs.org/docs/components-and-props.html#function-and-class-components) and behaves in the same way as in React for the web. This function returns a View component with some styles and a Text as its child.

The Text component allows us to render a text, while the View component renders a container. This container has several styles applied, let's analyze what each one is doing.

The first style that we find is flex: 1, the [flex](https://reactnative.dev/docs/layout-props#flex) prop will define how your items are going to "fill" over the available space along your main axis. Since we only have one container, it will take all the available space of the parent component. In this case, it is the only component, so it will take all the available screen space.

The following style is [justifyContent](https://reactnative.dev/docs/layout-props" \l "justifycontent): "center". This aligns children of a container in the center of the container's main axis. Finally, we have [alignItems](https://reactnative.dev/docs/layout-props" \l "alignitems): "center", which aligns children of a container in the center of the container's cross axis.

The other unusual thing in this code example is <View><Text>Hello world!</Text></View>. This is JSX - a syntax for embedding XML within JavaScript. Many frameworks use a specialized templating language which lets you embed code inside markup language(like angular \*ngFor inside div). In React, this is reversed. JSX lets you write your markup language inside code. It looks like HTML on the web, except instead of web things like <div> or <span>, you use React components. In this case, <Text> is a [Core Component](https://reactnative.dev/docs/intro-react-native-components) that displays some text and View is like the <div> or <span>.

## **Components**[**​**](https://reactnative.dev/docs/tutorial#components)

So this code is defining HelloWorldApp, a new Component. When you're building a React Native app, you'll be making new components a lot. Anything you see on the screen is some sort of component.

**Props**[**​**](https://reactnative.dev/docs/tutorial#props)

Most components can be customized when they are created, with different parameters. These creation parameters are called props.

Your own components can also use props. This lets you make a single component that is used in many different places in your app, with slightly different properties in each place. Refer to props.YOUR\_PROP\_NAME in your functional components or this.props.YOUR\_PROP\_NAME in your class components. Here's an example:

import React from 'react';

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

const styles = StyleSheet.create({

  center: {

    alignItems: 'center'

  }

})

const Greeting = (props) => {

  return (

    <View style={styles.center}>

      <Text>Hello {props.firstname}!  {props.lastname}!</Text>

    </View>

  );

}

const LotsOfGreetings = () => {

  return (

    <View style={[styles.center, {top: 50}]}>

      <Greeting firstname='Amine' lastname='Mezghich' />

      <Greeting firstname='Mohamed' lastname='Mezghich'/>

      <Greeting firstname='Med Amine' lastname='Mezghich' />

    </View>

  );

}

export default LotsOfGreetings;

Using name as a prop lets us customize the Greeting component, so we can reuse that component for each of our greetings. This example also uses the Greeting component in JSX. The power to do this is what makes React so cool.

The other new thing going on here is the [View](https://reactnative.dev/docs/view) component. A [View](https://reactnative.dev/docs/view) is useful as a container for other components, to help control style and layout.

With props and the basic [Text](https://reactnative.dev/docs/text), [Image](https://reactnative.dev/docs/image), and [View](https://reactnative.dev/docs/view) components, you can build a wide variety of static screens. To learn how to make your app change over time, you need to [learn about State](https://reactnative.dev/docs/tutorial#state).

## **State**[**​**](https://reactnative.dev/docs/tutorial#state)

Unlike props [that are read-only](https://reactjs.org/docs/components-and-props.html#props-are-read-only) and should not be modified, the state allows React components to change their output over time in response to user actions, network responses and anything else.

#### What's the difference between state and props in React?[​](https://reactnative.dev/docs/tutorial#whats-the-difference-between-state-and-props-in-react)

In a React component, the props are the variables that we pass from a parent component to a child component. Similarly, the state are also variables, with the difference that they are not passed as parameters, but rather that the component initializes and manages them internally.

## Un Hook, qu’est-ce que c’est ?

Pour notre nouvel exemple, commençons par importer le Hook useState de React :

import React, { useState } from 'react';

function Example() {

// ...

}

**Qu’est-ce qu’un Hook ?** Un Hook est une fonction qui permet de « se brancher » sur des fonctionnalités React. Par exemple, useState est un Hook qui permet d’ajouter l’état local React à des fonctions composants.

**Quand utiliser un Hook ?** Auparavant, si vous écriviez une fonction composant et que vous réalisiez que vous aviez besoin d’un état local à l’intérieur, vous deviez la convertir en classe. Désormais vous pouvez utiliser un Hook à l’intérieur de votre fonction composant.

## **Déclarer une variable d’état**

Dans une classe, on initialise l’état local count à 0 en définissant this.state à { count: 0 } dans le constructeur :

class Example extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

}

Dans une fonction composant, nous ne pouvons pas écrire ou lire this.state puisqu’il n’y a pas de this. Au lieu de ça, nous appelons directement le Hook useState dans notre composant :

import React, { useState } from 'react';

function Example() {

// Déclare une nouvelle variable d'état, que nous appellerons « count »

const [count, setCount] = useState(0);

**Appeler useState, qu’est-ce que ça fait ?**

Ça déclare une « variable d’état ». Notre variable est appelée count mais nous aurions pu l’appeler n’importe comment, par exemple banane. C’est un moyen de « préserver » des valeurs entre différents appels de fonctions. useState est une nouvelle façon d’utiliser exactement les mêmes possibilités qu’offre this.state dans une classe. Normalement, les variables « disparaissent » quand la fonction s’achève mais les variables d’état sont préservées par React.

**Qu’est-ce qu’on passe à useState comme argument ?**

Le seul argument à passer au Hook useState() est l’état initial. Contrairement à ce qui se passe dans les classes, l’état local n’est pas obligatoirement un objet. Il peut s’agir d’un nombre ou d’une chaîne de caractères si ça nous suffit. Dans notre exemple, nous voulons simplement le nombre de fois qu’un utilisateur a cliqué sur le bouton, nous passerons donc 0 comme état initial pour notre variable. (Si nous voulions stocker deux valeurs différentes dans l’état, nous appellerions useState() deux fois.)

**Que renvoie useState ?**

 Elle renvoie une paire de valeurs : l’état actuel et une fonction pour le modifier. C’est pourquoi nous écrivons const [count, setCount] = useState(). C’est semblable à this.state.count et this.setState dans une classe, mais ici nous les récupérons en même temps.

## **Lire l’état**

Quand nous voulons afficher la valeur actuelle de count dans une classe, nous récupérons la valeur de this.state.count :

<p>Vous avez cliqué {this.state.count} fois</p>

Dans une fonction, nous pouvons directement utiliser count :

<p>Vous avez cliqué {count} fois</p>

## **Mettre à jour l’état**

Dans une classe, nous devons appeler this.setState() pour mettre à jour l’état count :

<button onClick={() => this.setState({ count: this.state.count + 1 })}> Cliquez ici

</button>

Dans une fonction, nous récupérons directement setCount et count comme variables, nous n’avons donc pas besoin de this :

<button onClick={() => setCount(count + 1)}> Cliquez ici

</button>

## **En résumé**

Il est maintenant temps de **récapituler ce que nous avons appris ligne par ligne** et vérifier que nous avons bien compris.

1: import React, { useState } from 'react';

2:

3: function Example() {

4: const [count, setCount] = useState(0);

5:

6: return (

7: <div>

8: <p>Vous avez cliqué {count} fois</p>

9: <button onClick={() => setCount(count + 1)}>

10: Cliquez ici

11: </button>

12: </div>

13: );

14: }

* **Ligne 1 :** nous importons le Hook useState depuis React. Il nous permet d’utiliser un état local dans une fonction composant.
* **Ligne 4 :** dans le composant Example, nous déclarons une nouvelle variable d’état en appelant le Hook useState. Il renvoie une paire de valeurs que nous pouvons nommer à notre guise. Ici, nous appelons notre variable count puisqu’elle contient le nombre de clics sur le bouton. Nous l’initialisons à zéro en passant 0 comme seul argument à useState. Le second élément renvoyé est une fonction. Elle nous permet de modifier la variable count, nous l’appellerons donc setCount.
* **Ligne 9 :** quand l’utilisateur clique, nous appelons setCount avec une nouvelle valeur. React rafraîchira le composant Example et lui passera la nouvelle valeur de count.

Exercice 1 : Composants de base de React/React Native

TP 1 : Affichage aléatoire de chat sur son screen

import React, { useState } from 'react';

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

const App = () => {

  const[img, setImg]=useState(' ');

  const getCat = () => {

    fetch('https://aws.random.cat/meow').then(

      (res) => { return res.json() }).then(data => {

        console.log(data) ;

        setImg(data.file);

      })

  }

  //getCat();

  return (

    <View style={styles.container}>

    <Text>Cliquer ici pour avoir un chat</Text>

    <Image source={{uri: img}} style={styles.img}/>

    <Button

      onPress={getCat}

      title="Nouveau Chat"

      color="#f00"

      />

    </View>

  );

};

// React Native Styles

const styles = StyleSheet.create({

  container: {

    flex: 1,

    justifyContent: 'center',

    alignItems: 'center'

  },

  img: {

    width: '80%',

    height:'60%',

    marginBottom:'5%',

    marginTop:'5%',

  }

});

export default App;

TP 2: React navigation(TabBar)

Créon un nouveau projet et y ajouter les dépendances suivantes

Cmd1) expo init tabNavigation

Cmd 2) : npx expo install [react-native-web@~0.18.7](mailto:react-native-web@~0.18.7)

Cmd 3) : npx expo install [react-dom@18.0.0](mailto:react-dom@18.0.0)

Cmd 4) : npx expo install @expo/webpack-config@^0.17.0

**Pour les dépendances de navigation :**

npm install react-navigation

npm install @react-navigation/native

npm install @react-navigation/native-stack

npm install react-native-safe-area-context

npm install @react-navigation/bottom-tabs

Attention : Les noms des composants doivent toujours commencer par une majuscule !

import { StatusBar } from 'expo-status-bar';

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

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

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

const tab = createBottomTabNavigator();

function AccueilScreen() {

  return (<View style={styles.container}>

    <Text>Welcome to REACT native</Text>

    <StatusBar style="auto" />

  </View>);

}

function ContactScreen() {

  return (<View style={styles.container}>

    <Text>Leave a message here</Text>

    <StatusBar style="auto" />

  </View>);

}

export default function App() {

  return (

    <NavigationContainer>

      <tab.Navigator>

       <tab.Screen name="Home" component={AccueilScreen}/>

       <tab.Screen name="Contact" component={ContactScreen}/>

      </tab.Navigator>

    </NavigationContainer>

  );

}

const styles = StyleSheet.create({

  container: {

    flex: 1,

    backgroundColor: '#fff',

    alignItems: 'center',

    justifyContent: 'center',

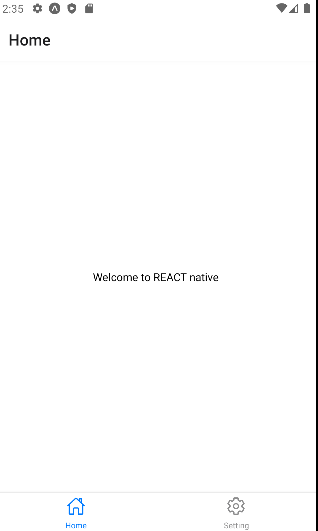
  },

});

Ajoutons maintenant des icones de navigation 😊

<https://ionic.io/ionicons>

<https://reactnavigation.org/docs/screen-options/>



import { StatusBar } from 'expo-status-bar';

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

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

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

import Ionicons from 'react-native-vector-icons/Ionicons';

const tab = createBottomTabNavigator();

function AccueilScreen() {

  return (<View style={styles.container}>

    <Text>Welcome to REACT native</Text>

    <StatusBar style="auto" />

  </View>);

}

function SettingScreen() {

  return (<View style={styles.container}>

    <Text>Vos paramètres</Text>

    <StatusBar style="auto" />

  </View>);

}

export default function App() {

  return (

    <NavigationContainer>

      <tab.Navigator

        screenOptions={({ route }) => ({

          tabBarIcon: ({ focused, color, size }) => {

            let iconName;

            if (route.name == "Home") { iconName = "home-outline"; }

            else if (route.name == "Setting") { iconName = "settings-outline"; }

            return (

              <Ionicons

                name={iconName}

                color={color}

                size={size}

              />

            );

          },

        })}

      >

        <tab.Screen name="Home" component={AccueilScreen} />

        <tab.Screen name="Setting" component={SettingScreen} />

      </tab.Navigator>

    </NavigationContainer>

  );

}

const styles = StyleSheet.create({

  container: {

    flex: 1,

    backgroundColor: '#fff',

    alignItems: 'center',

    justifyContent: 'center',

  },

});

On peut agir sur les autres options de taille et couleur :

<Ionicons

                name={iconName}

                color={'red'}

                size={20}

              />