Creacion de componentes

Para poner un alert seria asi :

import { Alert } from 'react-native';

Alert.alert("Alerta", "Este es un mensaje de alerta.");

Para poner un spread para un Array:

import { useState } from 'react';

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

export default function App() {

const [myArray, setMyArray] = useState([]);

const [number, setNumber] = useState(1);

const handleOnPress = () => {

setNumber(number + 1);

let newArray = [...myArray];

newArray.push(number);

setMyArray(newArray);

};

return (

<View style={styles.container}>

<Pressable onPress={handleOnPress}>

<Text style={styles.text}>Púlsame!</Text>

</Pressable>

{myArray.map((value, index) => (

<Text key={index.toString()}>{value}</Text>

))}

</View>

);

}

Ejemplo del componente Article hecho en las APs

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

export default function Article({paragraphOne, imagen, paragraphTwo, funcion }) {

return (

<View style={styles.container}>

<Pressable onPress={funcion}>

<Text style= {styles.title}>{paragraphOne}</Text>

</Pressable>

<Image source={imagen} style={{height: 110}}></Image>

<Text style= {styles.title}>{paragraphTwo}</Text>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: 'white',

alignItems: 'center',

justifyContent: 'center',

},

title: {

fontSize: 12,

fontStyle: 'italic',

fontWeight: 'bold',

},

});

Llamar a article e importarlo

import Article from '../components/Article';

<Article

paragraphOne={text}

imagen={imagen1}

paragraphTwo={text2}

funcion={handleOnPress1}></Article>

Poner estilos en un text input o text

<Text style = {{fontSize: 20, fontWeight:'bold'}}> Nombre </Text>

<TextInput

style={{ height: 40 }}

placeholder="Inserta tu nombre"

onChangeText={(newText2) => setText2(newText2)}

value={text2}

/>

Almacenar strings o numeros en un array

const handleOnPress = () => {

if (text.length == ""){

alert("Mensaje vacio")

}

if (isNaN(text)){

alert("No es numero")

}

if (!isNaN(text)){

let newArray = [...myArray];

newArray.push(text);

setMyArray(newArray);

}

<TextInput

style={{ height: 40 }}

placeholder="Inserta tu numero"

onChangeText={(newText) => setText(newText)}

defaultValue={text}

/>

Buscar una posicion especifica en un array

const handleOnPress2 = () => {

const valor = parseInt(text2);

if (isNaN(valor) || valor < 0 || valor >= myArray.length) {

alert('Índice inválido');

return;

}

setResultado(myArray[valor]);

};

Crear un array de objetos ( tambien se añaden al array )

const [myArray, setMyArray] = useState([]);

const handleOnPress = () => {

let newArray = [...myArray, { dni: text, nombre: text2 }];

setMyArray(newArray);

setText('');

setText2('');

};

Mostrar objetos de un array de objetos

{myArray.map((value, index) => (

<Text key={index}>

dni: {value.dni}, nombre: {value.nombre}

</Text>

))}

Añadir dos valores a una Array de objetos con push:

const [myArray, setMyArray] = useState([]);

const [dni, setDni] = useState('');

const [nombre, setNombre] = useState('');

const handleOnPress = () => {

let newArray = [...myArray, { dni: dni, nombre : nombre}];

newArray.push(`${dni} / ${nombre}`);

};

Validador de DNI

const numeros = text.slice(0,8);

const letra = text[8];

if (isNaN(numeros)) {

alert("DNI no valido, los primeros 8 carácteres deben ser números")

setText('');

setText2('');

return;

}

if (!isNaN(letra)) {

alert("DNI no valido, el último carácter debe ser una letra");

setText('');

setText2('');

return;

}

Mostrar fila de objetos

<View style={styles.container}>

<View style={{ flexDirection: 'row' }}>

{circulos1.map((elem, item) => (

<View key={item} style={elem}/>

))}

</View>

(El objeto)

const [circulos1, setCirculos1] = useState([

{

width: 100,

height: 100,

borderRadius: 100 / 2,

backgroundColor: 'red',

},

{

width: 100,

height: 100,

borderRadius: 100 / 2,

backgroundColor: 'blue',

},

{

width: 100,

height: 100,

borderRadius: 100 / 2,

backgroundColor: 'green',

},

]);

Uso de TextInput y Pressable

<Text style={styles.text}>Password</Text>{*/\*Texto de titulo\*/*}

<TextInput style={styles.textinput}

style={styles.textinput}

value={password}

onChangeText={setPassword}

placeholder="Introduce tu contraseña..."

/>{*/\*Cuadrado donde añadir el texto\*/*}

<Pressable

onPress={dniValidation}

style={{

backgroundColor: 'orange',

width: 130,

height: 30,

borderColor: 'black',

borderRadius: 10,

borderWidth: 2,

padding: 4,

alignItems: 'center',

}}>{*/\*BOTON para poder aceptar los cambios, llamar a la funcion y ejecutar\*/*}

<Text>Validar</Text>

</Pressable>

1. Trabajamos por componentes, es decir, parecido a las clases de Java, donde tenemos un App.js (programa principal) y luego dentro de la carpeta components vamos creando los diferentes componentes.

EJEMPLO DE FUNCIÓN PARA HACER UN FOR QUE PRINTEE 4 VECES POR PANTALLA LO MISMO. LO QUE HACEMOS AQUÍ ES LLAMAR A LA FUNCION ARTICLE, QUE ENCUENTRA DENTRO DEL MISMO COMPONENTE.

function Articles() {

let arts = [];

for (let i=0; i < 4; i++) {

arts.push(<Article key={i.toString()}/>);

}

return (arts)

}

PARA PODER USAR COMPONENTES DESDE OTRO, DEBEMOS IMPORTARLOS, POR EJEMPLO, EN App.js IMPORTAMOS LAS FUNCIONES ARTICLE Y ARTICLES DE LA SIGUIENTE MANERA

import {Article, Articles} from './components/Article'

COMO HACER FUNCIONES

function Article() {

return (

<View style={styles.container} >

<Text style = {styles.title}>{content.title}</Text>

<Text>{content.firstParagraph}</Text>

<Text>{content.secondParagraph}</Text>

<Text>{nombre}</Text>

<Image style={styles.tinyLogo} source={require('../assets/spongebob-smile.png')}

/>

<StatusBar style="auto" />

</View>

);

}

let nombre="Samuel Rodriguez Garcia"

let content= {

firstParagraph: "Tilin",

secondParagraph: "vs Zaza",

title:"La vida de waza"

}

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: 'yellow',

alignItems: 'center',

justifyContent: 'center',

},

title: {

fontSize: 12,

fontWeight: 'bold',

fontStyle: 'italic',

textDecorationLine: 'underline'

},

tinyLogo: {

width: 100,

height: 100,

},

});

ESTA FUNCION, APARTE DE INCLUIR COSAS COMO <View>,<Text>…

LO QUE HACE ES LLAMAR A ATRIBUTOS DE VARIABLES. POR EJEMPLO, LLAMAMOS A content.firstParagraph, que lo que contiene es “Tilin”.

<Text style = {styles.title}>{content.title}</Text>

Aquí por ejemplo, estamos añadiendo estilo mediante la palabra clave “style” después de text desde la constante styles, que contiene el atributo title.

EVENTOS EN REACT NATIVE

SE USAN, POR EJEMPLO, PARA HACER X COSAS AL PULSAR UN BOTÓN O AL INTERACTUAR CON OTROS ELEMENTOS DE LA INTERFAZ

function handleOnPress() {

alert('Hola');

}

return (

<View style={styles.container}>

<Pressable onPress={handleOnPress}>

<Text style={styles.text}>Pulsame!</Text>

</Pressable>

</View>

);

EN ESTE EJEMPLO, MEDIANTE EL ELEMENTO <Pressable> INTRODUCIMOS UN BOTÓN QUE MEDIANTE LA PALABRA CLAVE onPress HACEMOS QUE EJECUTE LO QUE SE ENCUENTRA DENTRO DE LA FUNCIÓN.

HOOKS EN REACT NATIVE

PERMITEN REALIZAR DETERMINADAS ACCIONES, COMO POR EJEMPLO CAMBIAR EL FONDO DE COLOR DE LA APLICACIÓN.

MEDIANTE LA PALABRA CLAVE useState podemos manejar el estado interno de los componentes.

(hay que importar useState desde React Native para usarlo)

const [color, setColor] = useState('yellow');

function handleOnPress() {

setColor('green');

alert('Cambiado el color a verde');

}

PRIMERO, CREAMOS UNA CONSTANTE QUE HAY QUE DECLARAR ENTRE []. LA SINTAXIS ES:

Const [nombreVaribale, setColor] = useState ();

Y LUEGO MEDIANTE LA FUNCIÓN CAMBIAMOS EL COLOR DEL FONDO PARA QUE DESDE UN PRESSABLE PASE DE COLOR AMARILLO A VERDE.

EJEMPLOS PARA DECLARAR DIFERENTES VARIABLES USANDO ESTA SINTAXIS

const [bColor, setbColor] = useState('green');

const [bColorSquare, setbColorSquare] = useState('yellow');

const [wRectangle, setwRectangle] =useState(200);

const [hRectangle, sethRectangle] = useState(200);

const windowWidth = Dimensions.get('window').width;

const windowHeight = Dimensions.get('window').height;

A su vez, también podemos declarar variables de esta manera que tengan distintas propiedades.

POR EJEMPLO, PODEMOS DEFINIR UN CUADRADO DE LA SIGUIENTE MANERA

const [cuadradoVerde, setCuadradoVerde] = useState({

height: 200,

width: 200,

backgroundColor: 'green',

});

UN EJEMPLO DE COMO DEFINIR UN OBJETO COMO VARIABLE Y ACCEDER A SUS PROPIEDADES EN UN MÉTODO PARA CAMBIAR SUS VALORES

const [square, setSquare] = useState({

color: 'green',

side: 150,

});

function handleOnPress() {

if (square.color === 'yellow' && square.side <= limit) {

setLimit(390);

setSquare({

color: 'green',

side: square.side + 10

});

PROPIEDADES ÚTILES

side à permite modificar el ancho de un objeto por ambos lados

SI QUEREMOS VARIOS ITEMS APILADOS EN COLUMNA, UNO DEBAJO DEL OTRO:

columna: {

flexDirection: 'column',

alignItems: 'center',

marginBottom: 30, *// espacio entre la columna y el grid*

},

SI QUEREMOS VARIOS ITEMS DESPUESTOS EN VARIAS FILAS Y COLUMNAS:

grid: {

flexDirection: 'row',

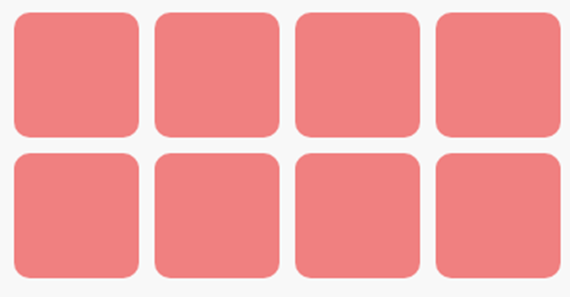
flexWrap: 'wrap',

justifyContent: 'center',

alignContent: 'center',

},

Esta disposicion de grid muestra lo siguiente a partir del siguiente fragmento de codigo



<View style={styles.grid}>

{[...Array(8)].map((\_, i) => (

<View key={i} style={styles.itemGrid} />

))}

</View>

Lo que hacemos es, encapsular todo dentro de un View que tendrá los estilos del grid para, posteriormente dentro de otro view visualizar la información de styles.itemGrid.  
  
Para si es un numero ponerlo en un array y si no ponerlo en otro array distinto :

const [numeroArray, setNumeroArray] = useState([]);

const [stringArray, setStringArray] = useState([]);

const [text, setText] = useState('');

const comprobador = () => {

if (isNaN(text)) {

const nuevoArray = [...stringArray, text];

setStringArray(nuevoArray);

} else {

const nuevoArray = [...numeroArray, parseFloat(text)];

setNumeroArray(nuevoArray);

}

};

Y ahora ponerlo en el return:

<View style={styles.container}>

<View style={{ flexDirection: 'row', gap: 10 }}>

<TextInput

style={styles.input}

onChangeText={(newText) => setText(newText)}

value={text}

placeholder="Escribe algo"

/>

<Pressable

onPress={comprobador}

style={({ pressed }) => [styles.boton, pressed && styles.botonPresionado]}>

<Text style={styles.texto}>Enviar</Text>

</Pressable>

</View>

{*/\* Mostrar arrays \*/*}

<View style={{ marginTop: 20 }}>

<Text style={styles.titulo}>Números:</Text>

{numeroArray.map((num, index) => (

<Text key={index.toString()}>{num}</Text>

))}

</View>

<View style={{ marginTop: 20 }}>

<Text style={styles.titulo}>Palabras:</Text>

{stringArray.map((palabra, index) => (

<Text key={index.toString()}>{palabra}</Text>

))}

</View>

</View>

);

Y con los estilos:

container: {

flex: 1,

backgroundColor: '#fff',

alignItems: 'center',

justifyContent: 'center',

padding: 20,

},

input: {

backgroundColor: '#909090',

width: 140,

height: 30,

borderColor: 'black',

borderWidth: 2,

borderRadius: 10,

padding: 5,

},

boton: {

backgroundColor: '#4CAF50',

paddingVertical: 10,

paddingHorizontal: 20,

borderRadius: 8,

},

botonPresionado: {

opacity: 0.7,

transform: [{ scale: 0.97 }],

},

texto: {

color: '#fff',

fontSize: 16,

fontWeight: '600',

},

titulo: {

fontWeight: 'bold',

fontSize: 18,

marginBottom: 5,

},

BUSCAR OBJETO DENTRO DE UN ARRAY

const handleOnPress2 = () => {

const posicion = parseInt(text3); *// Convertimos el texto a número entero*

if (!isNaN(posicion) && posicion >= 0 && posicion < arrayPalabras.length) {

*// Si la posición existe en el array*

const elemento = arrayPalabras[posicion];

setResultado(`Texto: ${elemento.texto}, Número: ${elemento.numeroArray}`);

} else {

*// Si la posición no es válida*

setResultado('Posición no válida');

}

MOSTRAR SOLO LA ULTIMA ENTRADA DE UN ARRAY  
 {arrayEntries.length > 0 ? arrayEntries[arrayEntries.length - 1] : ''}

const handleOnPress = (index) => {

let newCirculos = [...circulos];

let nuevoColor = newCirculos[index].color == "blue" ? "red" : "blue";

newCirculos[index].color = nuevoColor;

if (index > 0) {

newCirculos[index - 1].color = nuevoColor;

}

*// Si el círculo presionado no es el último, cambiar el color del círculo a la derecha*

if (index < newCirculos.length - 1) {

newCirculos[index + 1].color = nuevoColor;

}

setCirculo(newCirculos);

}