

Función para Buscar la posición del array en la api

```
const posicion = (indice) => {
  return indice - 1;
}
```

```
const datos = async () =>{
  const data = await getData();
// No ponemos el spread porque si no se repite el objeto todo el tiempo y solo
// lo queremos una vez.
  setMyarray([{image1: data.items[posicion(number)].avatar_url, login:
  data.items[posicion(number)].login, id: data.items[posicion(number)].id}]);
}

}
```

Aquí se utiliza el items porque en la api utilizada aparece items como array de objetos.

```
<View style={{ marginTop: 10 }}>
  <Pressable style={styles.boton} onPress={() => {posicion()}}>
    <Text style={styles.title}>Busca posicion</Text>
  </Pressable>
</View>
```

### Siempre hay que poner lo de async y await

HAY 2 MANERAS DE GUARDAR LA INFORMACION QUE DEVUELVE UNA PETICION EN UNA VARIABLE DE ESTADO. LA PRIMERA ES CREARSE UN ARRAY DE OBJETOS CON ATRIBUTOS Y MAPEAR LA RESPUESTA QUE DEVUELVE LA PETICION Y ASIGNARSE A CADA ATRIBUTO DEL ARRAY DE OBJETOS LO QUE NOS INTERESE.

```
export default function Screen2() {
  const [searchTerm, setSearchTerm] = useState('');
  const [episode, setEpisode] = useState([
    {
      episodeName: '',
      airDate: '',
      episode: '',
    },
  ]);
}

const getEpisodes = async (name) => {
  const response = await getData(
```

```

`https://rickandmortyapi.com/api/episode/?episode=${name}`);
);
response.results.map(item => {
  setEpisode([{
    episodeName: item.name,
    airDate: item.air_date,
    episode: item.episode,
  }])
})
};

}

```

LA OTRA MANERA ES CREARSE UN ARRAY DE OBJETOS VACIO Y GUARDARSE TODO LO QUE DEVUELVE LA RESPUESTA EN EL ARRAY DE OBJETOS.

```

export default function Screen1() {
  const [searchTerm, setSearchTerm] = useState('');
  const [characters, setCharacters] = useState([
    {
      // name: '',
      // image: '',
      // episodes: [],
    },
  ]);
}

const getCharacters = async (name) => {
  const response = await
getData(`https://rickandmortyapi.com/api/character/?name=${name}`);
  setCharacters(response.results);
  console.log(characters);
};

```

LUEGO, DESDE EL RETURN, PODEMOS ESCOGER LA INFORMACION QUE QUERAMOS MOSTRAR REFIRIENDONOS A ELLA DENTRO DE UN MAP.

POR EJEMPLO, en element.image, image es el nombre que tiene esa variable dentro del resultado de la peticion.

```

<View style={styles.container}>
  {characters.map((element, index) => (
    <View style={{ width: '33%' }}>
      <Image
        style={{
          width: 120,
          height: 120,
        }}
        source={{

```

```

        uri: element.image,
    )}
/>
<Text style={styles.text}>{element.name}</Text>
</View>
))}
</View>
```

Si queremos que nos aparezcan todos los resultados del api hacemos un spread si no hacemos nada.

```

const datos = async () =>{
    const data = await getData();
// No ponemos el spread porque si no se repite el objeto todo el tiempo y solo
lo queremos una vez.
    setMyarray([{image1: data.items[0].avatar_url, login: data.items[0].login,
id: data.items[0].id}]);
```

}

Solo se crea el array y yau :)

De esta manera podemos colocar o usar dos api en un mismo screen. Esto sirve para que cuando usemos el api la imagen sea diferente, si usamos la misma nos aparecería la misma imagen en la petición.

```

const ScreenTwo = () => {
    const [image1, setImage1] = useState();
    const [image2, setImage2] = useState();

    const handleOnpress = async () => {
        const data = await GetData(
            'https://api.thecatapi.com/v1/images/search?size=full'
        );
        const data2 = await GetData(
            'https://api.thecatapi.com/v1/images/search?size=full'
        );

        const picture1 = data [0].url;
        const picture2 = data2[0].url;

        setImage1(picture1);
        setImage2(picture2);
    };
}
```

```

<View style={styles.layout}>
    <Image source={{ uri: image1 }} style={{ width: 200, height: 200 }} />
    <Image source={{ uri: image2 }} style={{ width: 200, height: 200,
marginTop: 5 }} />
    <View>
        <Pressable>
            title="PULSAME"
            style={styles.boton}
            onPress={handleOnpress}></Pressable>
        </View>
    </View>

```

En este caso como en la api de rick and morty, hay un array de objetos y por esto es que tenemos que usar el results.map si no hubiera nada sí que podríamos poner data.map.

```

const handleOnpress = async () => {
    const data = await GetData(
        `https://rickandmortyapi.com/api/character/?name=${name}`
    );

    const newArray = data.results.map((item) => ({
        image1: item.image,
        name1: item.name,
    }));
    setMyarray(newArray);
};

```

dManera para buscar nombre, episodio etc. Utilizamos las comillas estas raras, si no sabes cuales son has copy and paste.

```

const data = await GetData(
    `https://rickandmortyapi.com/api/character/?name=${name}`
);

```

Carpeta services/services.js.

```

const GetData = async (url) => {
    try {
        const response = await fetch(url);
        if (response.ok) return await response.json();
    } catch (error) {
        return console.log(error);
    }
}

```

```
};

export default GetData;
```

¡¡¡Ojo a este caso!!!. Para que las pantallas vayan a la pantalla que quieras Debes de poner el mismo nombre.

```
<View style={styles.container}>
    <Text style={styles.title}>Screen stack 1</Text>

    <Pressable onPress={() => props.navigation.navigate('Screen 2')}
style={styles.boton}>
        <Text style={styles.title}>ir a screen 2</Text>
    </Pressable>
</View>
```

Aquí podemos observar al caso de arriba. Si escribo screen2 no iría a la otra pantalla,

Por eso se debe de escribir exactamente el nombre.

```
<Stack.Navigator options="false">
    <Stack.Screen name="Screen 1" component={Screen1} />
    <Stack.Screen name="Screen 2" component={Screen2} />
    <Stack.Screen name="Screen 3" component={Screen3} />
</Stack.Navigator>
```

Esto sucede igual con el modal. Solo que es de una manera más compleja pero es simplemente seguir y copiar el modelo este y ya

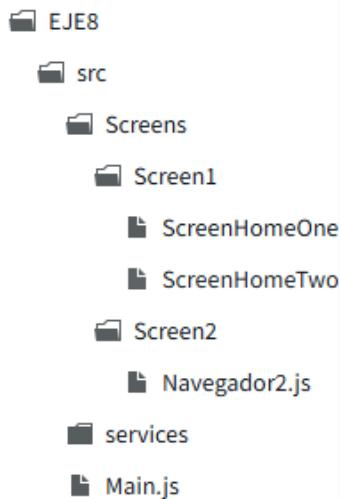
```
const App = () => (
    <NavigationContainer>
        <Stack.Navigator>
            <Stack.Group>
                <Stack.Screen name="Screen1" component={Screen1} />
            </Stack.Group>
            <Stack.Group screenOptions={{ presentation: 'modal' }}>
                <Stack.Screen
                    name="Screen2"
                    component={Screen2}
                    options={{ headerShown: true, headerMode: 'none' }}
                />
            </Stack.Group>
        </Stack.Navigator>
    </NavigationContainer>
```

```

const Screen1 = (props) => {
  return (
    <View style={styles.container}>
      <Text style={styles.title}>Pantalla screen1</Text>
      <Pressable onPress={() => props.navigation.navigate('Screen2')}>
        <Text style={styles.text}>Screen 2</Text>
      </Pressable>
    </View>
  );
};

```

La estructura para crear la navegacion de menus seria asi:



En main iria la navegacion principal, si queremos hacerlo Drawer o tab o stack serian de estas maneras:

esto por ejemplo es para hacerlo drawer:

MAPEADO CON SCROLL VIEW

```

import { createDrawerNavigator } from '@react-navigation/drawer';
import { NavigationContainer } from '@react-navigation/native';
import ScreenOne from './Screens/Screen1/ScreenHomeOne';
import ScreenTwo from './Screens/Screen1/ScreenHomeTwo';

```

```

const Drawer = createDrawerNavigator();

export default function App() {
  return (
    <NavigationContainer>
      <Drawer.Navigator useLegacyImplementation={false}>
        initialRouteName="ScreenOne"

          <Drawer.Screen name="ScreenOne" component={ScreenOne} />
          <Drawer.Screen name="ScreenTwo" component={ScreenTwo} />
        </Drawer.Navigator>
      </NavigationContainer>
    );
}

const handleOnPress = async () => {
  const data = await getData(
    `https://rickandmortyapi.com/api/episode/?episode=${name}`
  );

  if (data && data.results) {
    setInfo(data.results);
  }
};

```

y por ejemplo si queremos tener un buscador tab y dentro de screen uno un modal o un stack seria asi:

```

import { StyleSheet, Text, View, Button, Image, TextInput, ScrollView,
Pressable } from 'react-native';
import { useState } from 'react';
import GetData from '../../../../../services/services';

const Screen2 = (props) => {
  const [myArray, setMyarray] = useState([]);
  const [name, setName] = useState('');

  const handleOnpress = async () => {
    const data = await
GetData(`https://api.github.com/search/users?q=${name}`);
    const newArray = data.items.map((elem) => ({

```

```

        image: elem.avatar_url,
        login: elem.login,
        id: elem.id,
    })));
}

setMyarry(newArray);
};

return (
<View style={styles.container}>
<View>
<TextInput
    style={styles.input}
    placeholder="ingresa nombre... "
    onChangeText={(newtext) => setName(newtext)}
    value={name}
MAP CON SCROLL VIEW

/>
<Pressable style={styles.boton} onPress={() => {handleOnpress()}}>PULSAME</Pressable>
</View>
<ScrollView>
{myArray.map((elem, index) => (
<View key={index}>
<Image
    source={{ uri: elem.image }}
    style={{ width: 100, height: 100, marginTop: 3 }}
/>
<Text>{elem.id}</Text>
<Text> {elem.login} </Text>
</View>
))}
</ScrollView>
</View>
);
};

const styles = StyleSheet.create({
container: {
flex: 1,
backgroundColor: 'white',
alignItems: 'center',
justifyContent: 'center',

```

```
},
title: {
  margin: 24,
  fontSize: 18,
  fontWeight: 'bold',
  textAlign: 'center',
},
boton: {
  borderWidth: 1,
  borderColor: 'blue',
  height: 40,
  width: 100,
  backgroundColor: 'orange',
  borderRadius: 8,
  alignItems: 'center',
  justifyContent: 'center',
  marginTop: 10,
},
input: {
  width: '90%',
  borderWidth: 1,
  borderColor: '#ccc',
  borderRadius: 8,
  padding: 10,
  fontSize: 16,
  backgroundColor: '#fff',
  marginBottom: 10,
  marginTop: 10,
},
});
export default Screen2;
```

Los estilos para que se pongan las imagenes al lado y abajo

```
flex: 1,
  flexDirection: 'row',
  flexWrap: 'wrap',
  alignItems: 'flex-start',
},
```

