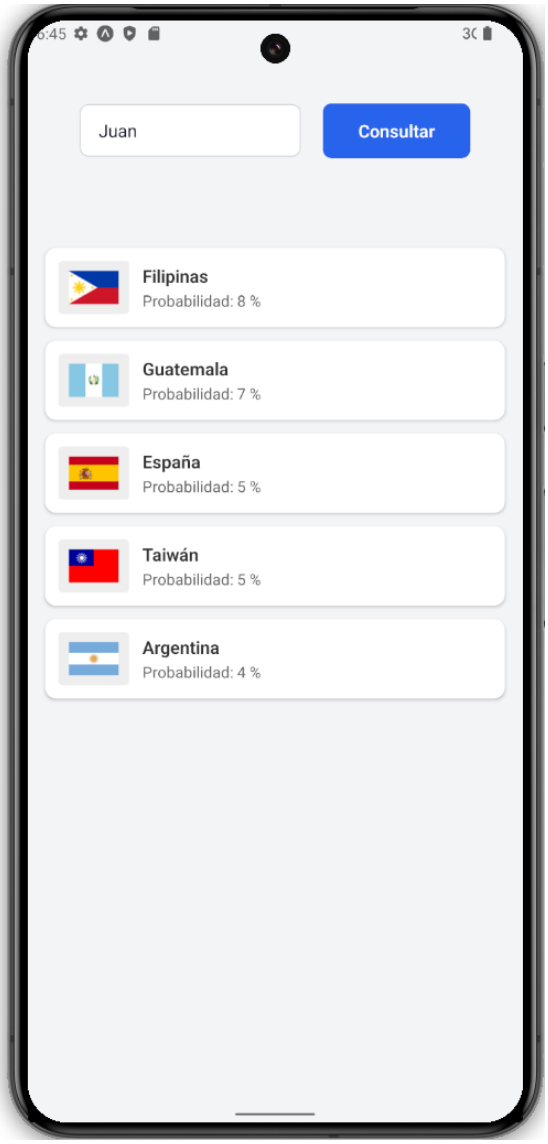
TUTORIAL 15

CONSULTAR APIS



En este tutorial vamos a aprender cómo acceder a APIs desde nuestra app. Para ello, usaremos el api gratuita **nationalize.io** que nos permite obtener la lista de países donde es más frecuente encontrar el nombre de una persona, y mostraremos los resultados en una lista.

1.- Las Apis

Hoy día las aplicaciones se desarrollan en varias versiones (web, móvil, escritorio) y todas ellas necesitan acceden al servidor donde se encuentran los datos que manejan. Dicho servidor, además de acceder a los datos, realiza operaciones con ellos, de acuerdo a los requerimientos realizados por los clientes. Por ejemplo, un cliente (el móvil) puede requerir al servidor que realice la compra de un determinado producto. Otro cliente (un navegador) puede requerir al servidor una lista con todos los productos cuyo precio está en un rango determinado.

Actualmente hay dos enfoques principales para que los clientes realicen solicitudes al servidor:

- ***APIs:** Un API es un programa instalado en el servidor que define una serie de peticiones, llamadas **endpoint,** que los clientes pueden realizarle. Las más frecuentes son las **API REST**, que utilizan el famoso protocolo HTTP y cada endpoint posee una **URL** que los clientes usan para acceder a ella. La respuesta del API es un documento JSON, que puede contener datos o información sobre lo que ha realizado el servidor.*
- ***GraphQL:** Es un tipo especial de API en la cual, el cliente usa un lenguaje de consulta (al estilo SQL) con el que define qué es lo que necesita, y la respuesta del servidor sigue siendo un documento JSON.*

*Vamos a realizar una app que acceda al API **nationalize.io.** Dicho API posee un endpoint, cuya URL es **https://api.nationalize.io/?name={tuNombre}** al que le pasamos un nombre (de cualquier lugar del mundo) y nos devolverá un JSON con los países donde es más probable encontrarlo.*

1. Accede con tu navegador de Internet a **https://api.nationalize.io/?name=Juan** y comprueba que la salida es un documento en formato JSON como este:

```
{ status: 200,
  data: {
    count: 47640,
    name: "juan",
    country: [
      {
        country_id: "PH",
        probability: 0.0775183248714279,
      },
      {
        country_id: "GT",
        probability: 0.066564204995023,
      },
      {
        country_id: "ES",
        probability: 0.049398767709787,
      },
      {
        country_id: "TW",
        probability: 0.0471336347699892,
      },
      {
        country_id: "AR",
        probability: 0.0419322961317013,
      },
    ],
  },
}
```

2.- Inicio del proyecto

Vamos a dar los primeros pasos del proyecto y a diseñar la interfaz básica:

1. Crea un proyecto llamado **rn_nombres** y súbelo a GitHub
2. Crea una rama llamada **tutorial1**
3. Instala la librería **Axios** usando el comando **npm install axios**

> *La librería **Axios** permite acceder a las APIs de forma muy sencilla*

4. En el proyecto, crea las siguientes carpetas:
   - **components** → Servirá para crear nuestros componentes
   - **helpers** → Tendrá funciones útiles para nuestro proyecto
5. Abre **App.tsx**, borra lo que contiene y teclea **rnfs** para iniciar el componente
6. Como ejercicio, crea el siguiente diseño en **App.tsx** sin mirar la solución:


```
1.  export default function App(){
2.  return (
3.      <View style={styles.contenedorPrincipal}>
4.        <View style={styles.fila}>
5.          <TextInput placeholder={"Introduce tu nombre"}
6.            style={styles.cuadroTexto} placeholderTextColor={"#9CA3AF"}/>
7.          <Pressable
8.            style={ ({pressed}) => pressed?styles.botonPresionado : styles.boton }>
9.            <Text style={styles.textoBoton}>Consultar</Text>
10.         </Pressable>
11.       </View>
12.     </View>
13.   )
14. }
```

7. Añade una variable de estado llamada **nombre** y su setter **setNombre**, inicializado a una cadena de texto vacía.

```
1.  export default function App(){
2.    const [nombre, setNombre] = useState("")
3.    return ( // resto omitido
4.    )
5.  }
```

8. Añade al **TextInput** los siguientes props:
    - **value={nombre}**
    - **onChangeText={setNombre}**

```
<TextInput
    value={nombre}
    onChangeText={setNombre}
    placeholder={"Introduce tu nombre"}
    style={styles.cuadroTexto}
    placeholderTextColor={"#9CA3AF"}/>
```

*El prop **value** indica que el valor que aparecerá en el cuadro de texto será igual a la variable de estado **nombre.** De esta forma, siempre estaremos viendo en el cuadro de texto el valor de **nombre**.*

*Por su parte, el prop **onChangeText** recibe una función que se ejecutará cada vez que cambie el texto escrito en el cuadro. Al pasar **setNombre** lo que sucede es que cada vez que el usuario escriba algo, el texto actualizado sea pasado a **setNombre**, actualizándose así la variable **nombre** cada vez que se escribe algo en el cuadro.*

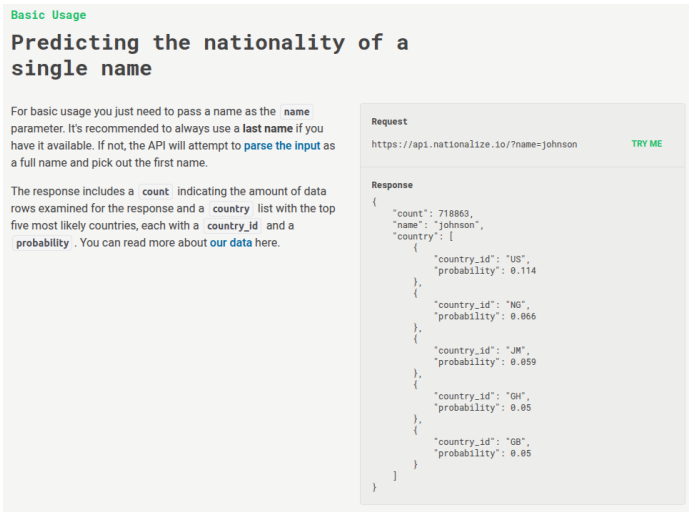
3.- Consultar el API

*En este apartado vamos a hacer una función que se encargue de acceder al API **nationalize.io** para obtener la lista de países-probabilidades donde encontrar un nombre, y posteriormente llamaremos desde **App** a dicha función.*

1. En **helpers** crea un archivo llamado **ConsultasApi.ts**
2. En **ConsultasApi.ts** crea y exporta una función llamada **consultarProbabilidades**. Recibirá como parámetro el nombre de una persona. Esta función nos devolverá una lista formada por objetos país-probabilidad. Como estamos en JavaScript, no es posible indicar los tipos.

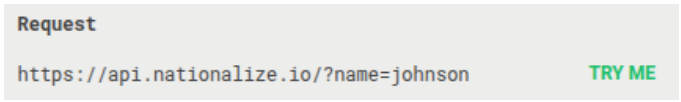
```
function consultarProbabilidades(nombre){
  // aquí consultaremos el api para obtener una lista de objetos pais-probabilidad
}
export {consultarProbabilidades}
```

3. Para consultar el API, primero tenemos que ver su documentación. Entra en https://nationalize.io/documentation y busca esta sección:



En ese cuadro puedes ver:

- *La URL del endpoint y un ejemplo de su uso*



- *Un ejemplo del documento JSON de respuesta*

*En la documentación del API falta una cosa muy importante, que es el **método** con el cual el cliente debe llamar al API. El método es uno de estos valores y debe venir indicado siempre en la documentación:*

- ***GET** → El cliente quiere consultar datos. Los datos se envían al servidor formando parte de la URL del endpoint, como vemos aquí (el nombre forma parte de la dirección)*
- ***POST** → El cliente quiere dar de alta una entidad en el servidor. Los datos para esa acción se envían ocultos, en la petición al servidor*
- ***PUT** → El cliente quiere actualizar una entidad en el servidor. Igualmente que en el anterior, los datos van ocultos.*
- ***DELETE** → El cliente quiere borrar una entidad del servidor. Los datos pueden ir en la URL o pueden enviarse ocultos.*

*Los métodos más usados son **GET** y **POST.** Si no se dice nada, es **GET***

4. Vamos a usar la librería **Axios** para consultar el API. En primer lugar, creamos un string con la URL del endpoint, colocando la variable **nombre** en el lugar del parámetro **name** de la URL

```
1.  function consultarProbabilidades(nombre){
2.    const endpoint = `https://api.nationalize.io/?name=${nombre}`
3.  }
```

5. A continuación, llamamos al método **axios.get** y le pasamos la variable con el endpoint. Como dicho método es asíncrono, usaremos **await** para llamarlo y esperar su respuesta, liberando al hilo principal de la espera.

```
1.  function consultarProbabilidades(nombre){
2.    const endpoint = `https://api.nationalize.io/?name=${nombre}`
3.    const respuestaServidor = await axios.get(endpoint)
4.  }
```

*Puesto que la consulta al API es una operación que tarda tiempo en completarse, todos los métodos de **axios** son asíncronos. Además, trabajan usando hilos separados, por lo que la interfaz de usuario no se bloquea durante la espera.*

6. Como **consultarProbabilidades** usa **await**, necesita ser una función asíncrona y estar marcada con la palabra **async**

```
1.  async function consultarProbabilidades(nombre){
2.    const endpoint = `https://api.nationalize.io/?name=${nombre}`
3.    const respuestaServidor = await axios.get(endpoint)
4.  }
```

7. Haz que la función **consultarProbabilidades** devuelva el campo **data** de la respuesta del servidor

```
async function consultarProbabilidades(nombre){
    const endpoint = `https://api.nationalize.io/?name=${nombre}`
    const respuestaServidor = await axios.get(endpoint)
    return respuestaServidor.data
}
```

*El documento JSON con la respuesta del api se encuentra en el campo **data** de la variable que guarda la respuesta del servidor.*

8. Vamos a hacer una prueba para comprobar si de verdad estamos accediendo al API y recuperando la lista de países-probabilidades correspondientes al nombre que escribimos. Abre **App.tsx** y crea, dentro de la función **App** una función llamada **botonPulsado**. Pasa dicha función al prop **onPress** del botón

```
export default function App(){
  const [nombre, setNombre] = useState("")
  function botonPulsado(){
    console.log("El botón ha sido pulsado")
  }
  return (
    <View style={styles.contenedorPrincipal}>
      <View style={styles.fila}>
        <TextInput
          value={nombre} onChangeText={setNombre}  placeholder={"Introduce tu nombre"}
          style={styles.cuadroTexto}
          placeholderTextColor={"#9CA3AF"}/>
        <Pressable
          onPress={botonPulsado}
          style={ ({pressed}) => pressed?styles.botonPresionado : styles.boton }>
          <Text style={styles.textoBoton}>Consultar</Text>
        </Pressable>
      </View>
    </View>
  )
}
```

9. Haz que la función **botonPulsado** llame a **consultarProbabilidades**, pasándole la variable **nombre**, y utiliza el encadenamiento de **then** y **catch** para mostrar en pantalla la respuesta del api si todo va bien, o mostrar una ventana emergente de error si falla.

```
function botonPulsado(){
  consultarProbabilidades(nombre)
    .then( respuesta => console.log(respuesta))
    .catch( error => Alert.alert("Error",error.toString()))
}
```

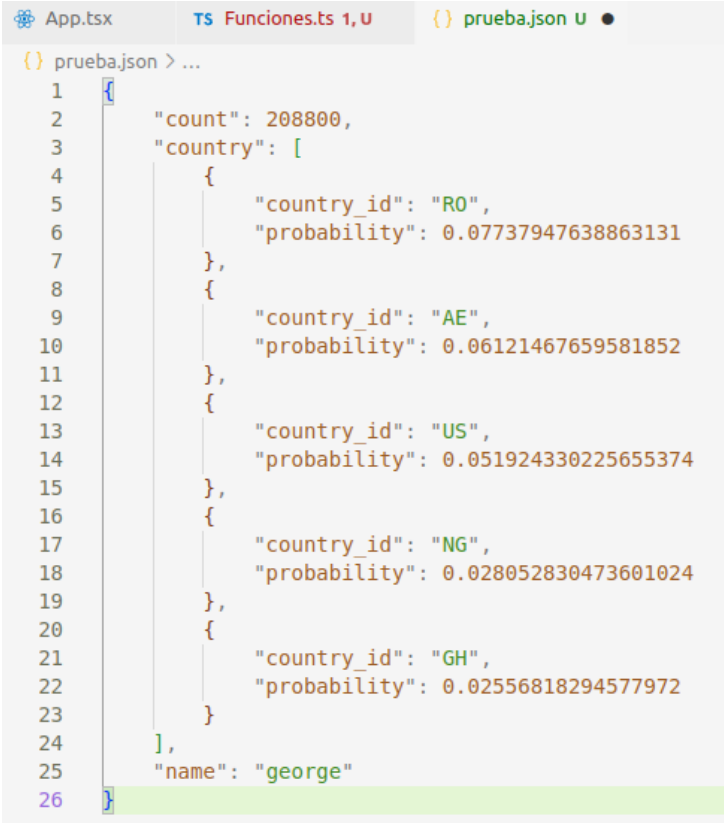
*Normalmente, en el código de la interfaz de usuario será más cómodo utilizar **then/catch** para llamar a una función asíncrona, aunque también se podría haber hecho usando **async/await** con un try-catch de esta forma:*

```
async function botonPulsado(){
  const respuesta = await consultarProbabilidades(nombre)
  try{
    console.log(respuesta)
  }catch(error){
    Alert.alert("Error",error.toString())
  }
}
```

10. Ejecuta la app y comprueba que en la terminal aparece un resultado como este

```
LOG {"count": 208800, "country": [{"country_id": "RO", "probability": 0.07737947638863131}, {"country_id": "AE", "proba
bility": 0.06121467659581852}, {"country_id": "US", "probability": 0.051924330225655374}, {"country_id": "NG", "probabili
ty": 0.028052830473601024}, {"country_id": "GH", "probability": 0.02556818294577972}], "name": "george"}
```

11. Copia el resultado anterior y pégalo en un archivo nuevo llamado **prueba.json.** Pulsa el botón derecho del ratón y la opción "**Format document**" para que se tabule el documento y poder ver mejor lo que nos ha enviado el API



*Observa que el API nos envía un objeto que tiene una propiedad **country** que guarda una lista con los países y sus probabilidades. El resto, no nos interesa*

12. Como queremos que la función **consultarProbabilidades** nos devuelva solamente la lista de países-probabilidades (actualmente nos está devolviendo el objeto anterior completo), modifica el **return** para que nos devuelva el valor del campo **country**

```
1.  async function consultarProbabilidades(nombre){
2.      const endpoint = `https://api.nationalize.io/?name=${nombre}`
3.      const respuestaServidor = await axios.get(endpoint)
4.      return resultado.data.country
5.  }
```

13. Ejecuta nuevamente la app y comprueba que al pulsar el botón, ahora ya si se muestra en la terminal solamente la lista de países-probabilidades

```
LOG [{"country_id": "RO", "probability": 0.07737947638863131}, {"country_id": "AE", "probability": 0.06121467659581852}
, {"country_id": "US", "probability": 0.051924330225655374}, {"country_id": "NG", "probability": 0.028052830473601024}, {
"country_id": "GH", "probability": 0.02556818294577972}]
```

14. Por último, vamos a crear una variable de estado llamada **listaProbabilidades** (inicialmente rellenada con una lista vacía **[ ]**) con su setter **setListaProbabilidades**, y haremos que, en lugar de mostrar por pantalla la lista consultada, la guardemos en dicha variable de estado

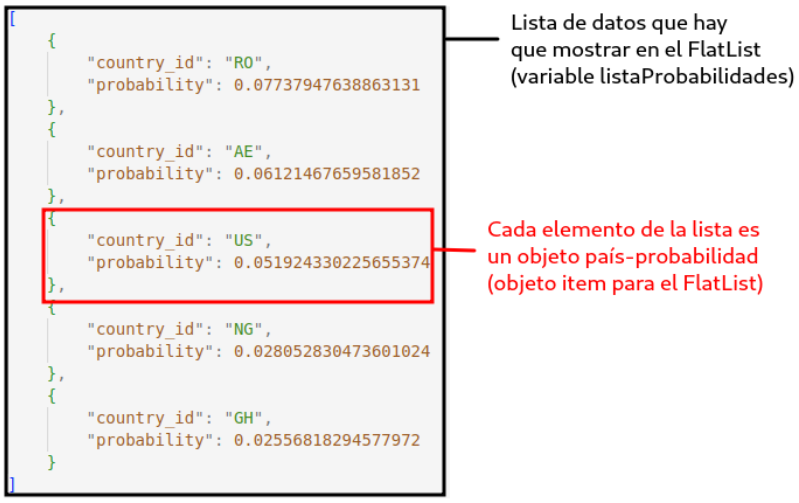
```
1.  export default function App() {
2.    const [nombre,setNombre] = useState("")
3.    const [listaProbabilidades,setListaProbabilidades] = useState([])
4.    function botonPulsado(){
5.      consultarProbabilidades(nombre)
6.        .then( respuesta => setListaProbabilidades(respuesta) )
7.        .catch( error => Alert.alert("Error",error.toString()))
8.    }
9.    // resto omitido
10. }
```

4.- Mostrar los resultados en un FlatList

*En este momento, ya tenemos una variable de estado que guarda la lista de países-probabilidades que nos da el API para el nombre escrito en el cuadro de texto. Ahora vamos a mostrar los resultados en una lista, y como ya sabemos, el **FlatList** es el componente ideal para ello.*

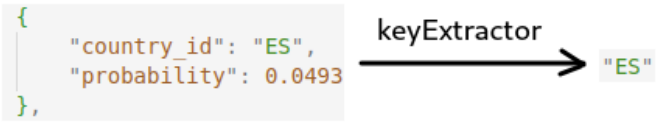
*Vamos a incluir en la interfaz un **FlatList** cuyos props rellenaremos de esta forma:*

- ***data:** Es la lista de datos que deberá mostrarse en la lista y en nuestro caso, será la lista de países-probabilidad que tenemos en la variable **listaProbabilidades**.*



- ***renderItem:** Es el componente que se usará para mostrar cada dato individual de la lista. Para que funcione, dicho componente debe tener un prop llamado **item** (que en este caso será un objeto con campos **country_id** y **probability**). Diseñaremos ese componente con forma de tarjeta:*

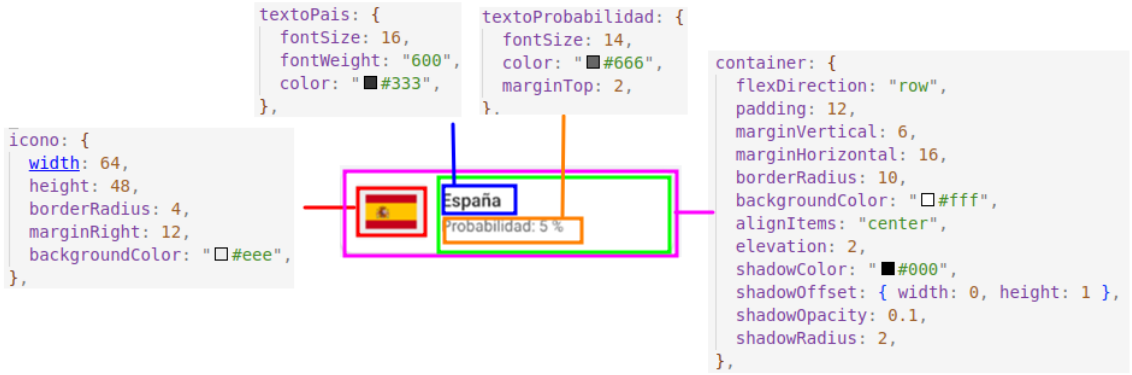
- ***keyExtractor:*** *Es una función o lambda que recibe un dato de la lista llamado* ***item*** *y nos devuelve el valor de su clave primaria. En nuestro caso, consideraremos que la clave primaria de cada objeto país-probabilidad será el código del país (****country_code****).*



- ***ListEmptyComponent:*** *Es una función (o lambda) que devuelve el componente que se muestra cuando la lista de datos está vacía. Es útil para mostrar un mensaje del tipo "No se han encontrado resultados"*

Comenzaremos diseñando un componente llamado ***ItemPaisProbabilidad,*** *que recibirá un prop* ***item*** *y diseñará una tarjeta donde se verá el* ***item.country_id*** *y el* ***item.probability*** *junto con la imagen de la bandera del país.*

1. Ejecuta **npx expo install expo-image** para instalar la libería **expo-image**
2. En **components** crea un archivo llamado **ItemPaisProbabilidad.tsx**
3. Abre **ItemPaisProbabilidad.tsx**, y escribe **rnfs+Intro**
4. Como ejercicio, diseña el siguiente componente sin mirar la solución (que está a continuación). De momento puedes poner una imagen de prueba (como el **icon.png** de la carpeta **assets**) y datos de prueba para el país y la probabilidad




```
export default function ItemPaisProbabilidad() {
  return (
    <View style={styles.container}>
      <Image
        source={require("../assets/icon.png")}
        style={styles.icono}
        contentFit="contain"/>
      <View>
        <Text style={styles.textoPais}>Nombre del país</Text>
        <Text style={styles.textoProbabilidad}>
          Probabilidad: Probabilidad del nombre %
        </Text>
      </View>
    </View>
  );
}
```

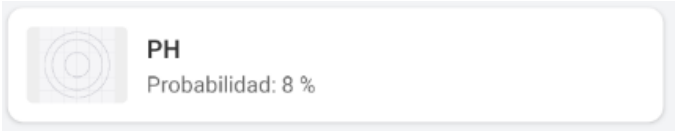
5. Añade a **ItemPaisProbabilidad** un prop llamado **item** (este item será un objeto con campos **country_id** y **probability**) y úsalo para mostrar **item.country_id** e **item.probability** (multiplicado por 100 y redondeado) en los dos **Text**

```
export default function ItemPaisProbabilidad({item}) {
  console.log(item);
  return (
    <View style={styles.container}>
      <Image
        source={require("../assets/icon.png")}
        style={styles.icono}
        contentFit="contain"
      />
      <View>
        <Text style={styles.textoPais}>{item.country_id}</Text>
        <Text style={styles.textoProbabilidad}>
          Probabilidad: {Math.round(100*item.probability)} %
        </Text>
      </View>
    </View>
  );
}
```

6. Aunque no esté terminado, vamos a comprobar que funciona. Para ello, abre **App.tsx** y después de la fila que contiene los botones, añade un **FlatList** con estos props:
   - **data** → La variable **listaProbabilidades**
   - **renderItem** → La función **ItemPaisProbabilidad**
   - **keyExtractor** → Una lambda que tome un **item** y devuelva **item.country_id**
   - **ListEmptyComponente** → Lambda que devuelve tu texto centrado con el mensaje “No se han encontrado resultados”

```
  return (
    <View style={styles.contenedorPrincipal}>
      <View style={styles.fila}>
        <TextInput
          value={nombre}
          onChangeText={setNombre}
          placeholder={"Introduce tu nombre"}
          style={styles.cuadroTexto}
          placeholderTextColor={"#9CA3AF"}/>
        <Pressable
          onPress={botonPulsado}
          style={ ({pressed}) => pressed?styles.botonPresionado : styles.boton }>
          <Text style={styles.textoBoton}>Consultar</Text>
        </Pressable>
      </View>
      <FlatList
        data={listaProbabilidades}
        renderItem={ItemPaisProbabilidad}
        keyExtractor={ item => item.country_id}
        ListEmptyComponent={ () => <text style="{{margin:"auto"}}">No se han encontrado
                                                                   resultados</Text>}/>
      />
    </View>
  )
}
```

Ejecuta la app y comprueba que ya se muestra el resultado, aunque las imágenes de los países aún son un icono de prueba. Otro detalle a mejorar es que solo mostramos el código del país, y no el nombre del país.



7. Vamos a poner el icono del país. Para ello abre Internet y entra en https://flagsapi.com/ y observa cómo esa página nos da un enlace con un icono (de 64 píxeles) a partir del código del país, y eso es justo lo que necesitamos

8. Abre **ItemPaisProbabilidad.tsx** y modifica el prop **source** del **Image** para que sea la URL que nos devuelve el icono del país:

```
<Image
  source={`https://flagsapi.com/${item.country_id}/flat/64.png`}
  style={styles.icono}

  contentFit="contain"
/>
```

> *Observa cómo el código del país (**item.country_id**) es colocado en el lugar adecuado que nos indica la documentación de la página web*

9. Ejecuta la app y comprueba que todo funciona correctamente

5.- Mostrar el nombre del país

*A continuación vamos a mostrar el nombre del país en lugar de su código. Para ello, consultaremos un endpoint del API **restcountries,** que nos devuelve un montón de datos de un país (entre ellos, su nombre) a partir de su código*

1. Entra en https://restcountries.com/ y en el menú **ENDPOINTS** que encontrarás en la izquierda, pulsa **Code**
2. Observa que la página se desplaza hasta un endpoint y allí aparece la descripción y la forma de utilizarlo.

> *Como vemos, el endpoint tiene un parámetro que es el código de país, que en este caso forma parte de la URL. El método sigue siendo **GET**, puesto que no se dice nada.*

3. Vamos a probar el endpoint. Abre el navegador de internet y accede a **https://restcountries.com/v3.1/alpha/ES**

4. Comprueba que se obtiene un documento JSON con un montón de información sobre el país encontrado:


```
[{"name":{"common":"Spain","official":"Kingdom of Spain","nativeName":{"spa":{"official":"Reino de España","common":"España"}}},"tld":
[".es"],"cca2":"ES","ccn3":"724","cioc":"ESP","independent":true,"status":"officially-assigned","unMember":true,"currencies":{"EUR":
{"symbol":"€","name":"Euro"}},"idd":{"root":"+3","suffixes":["4"]},"capital":["Madrid"],"altSpellings":["ES","Kingdom of Spain","Reino de
España"],"region":"Europe","subregion":"Southern Europe","languages":{"spa":"Spanish","cat":"Catalan","eus":"Basque","glc":"Galician"},"latlng":
[40.0,-4.0],"landlocked":false,"borders":["AND","FRA","GIB","PRT","MAR"],"area":505992.0,"demonyms":{"eng":{"f":"Spanish","m":"Spanish"},"fra":
{"f":"Espagnole","m":"Espagnol"}},"cca3":"ESP","translations":{"ara":{"official":"مملكة إسبانيا","common":"إسبانيا"},"bre":{"official":"Rouantelezh
Spagn","common":"Spagn"},"ces":{"official":"Španělské království","common":"Španělsko"},"cym":{"official":"Kingdom of
Spain","common":"Spain"},"deu":{"official":"Königreich Spanien","common":"Spanien"},"est":{"official":"Hispaania
Kuningriik","common":"Hispaania"},"fin":{"official":"Espanjan kuningaskunta","common":"Espanja"},"fra":{"official":"Royaume
d'Espagne","common":"Espagne"},"hrv":{"official":"Kraljevina Španjolska","common":"Španjolska"},"hun":{"official":"Spanyol
Királyság","common":"Spanyolország"},"ind":{"official":"Kerajaan Spanyol","common":"Spanyol"},"ita":{"official":"Regno di
Spagna","common":"Spagna"},"jpn":{"official":"スペイン王国","common":"スペイン"},"kor":{"official":"에스파냐 왕국","common":"스페인"},"nld":
{"official":"Koninkrijk Spanje","common":"Spanje"},"per":{"official":"پادشاهی اسپانیا","common":"اسپانیا"},"pol":{"official":"Królestwo Hiszpanii
","common":"Hiszpania"},"por":{"official":"Reino de Espanha","common":"Espanha"},"rus":{"official":"Королевство Испания","common":"Испания"},"slk":
{"official":"Španielske kráľovstvo","common":"Španielsko"},"spa":{"official":"Reino de España","common":"España"},"srp":{"official":"Краљевина
Шпанија","common":"Шпанија"},"swe":{"official":"Konungariket Spanien","common":"Spanien"},"tur":{"official":"İspanya
Krallığı","common":"İspanya"},"urd":{"official":"مملکتِ ہسپانیہ","common":"ہسپانیہ"},"zho":{"official":"西班牙王国","common":"西班
牙"}},"flag":"\uD83C\uDDEA\uD83C\uDDF8","maps":{"googleMaps":"https://goo.gl/maps/138JaXW8EZzRVitY9","openStreetMaps":"https://www.openstreetmap.org/
relation/1311341"},"population":47351567,"gini":{"2018":34.7},"fifa":"ESP","car":{"signs":["E"],"side":"right"},"timezones":
["UTC","UTC+01:00"],"continents":["Europe"],"flags":{"png":"https://flagcdn.com/w320/es.png","svg":"https://flagcdn.com/es.svg","alt":"The flag of
Spain is composed of three horizontal bands of red, yellow and red, with the yellow band twice the height of the red bands. In the yellow band is
the national coat of arms offset slightly towards the hoist side of center."},"coatOfArms":{"png":"https://mainfacts.com/media/images/coats_of_arms/
es.png","svg":"https://mainfacts.com/media/images/coats_of_arms/es.svg"},"startOfWeek":"monday","capitalInfo":{"latlng":[40.4,-3.68]},"postalCode":
{"format":"#####","regex":"^(\\d{5})$"}}]
```


5. En el documento **prueba.json** que creaste al principio para analizar las respuestas de **nationalize.io**, reemplaza lo que había por el documento que acabas de obtener y observa su estructura. Concretamente, comprueba que:
    - El documento define una **lista** que lleva dentro un **objeto**
    - Dicho objeto tiene campos como **name, tld, cca2, ccn3, cioc, independent, status, unMember, currencies**, etc y que muchos de ellos son a su vez, objetos con nuevos campos
    - Hay un campo llamado **translations**, donde aparece la traducción del nombre oficial y el nombre común a un montón de idiomas.
    - Las traducciones al español están en el campo **spa** del campo **translations**

Siempre que trabajamos con un API, es imprescindible analizar el documento JSON obtenido, porque así sabremos como acceder a los datos que nos interesan

6. Observa que podemos acceder al nombre del país en español siguiendo esta "ruta": **respuesta.data[0].translations.spa.common**

*Ponemos **respuesta.data[0]** porque **respuesta.data** es una lista y por tanto, **respuesta[0]** es su primer elemento, que es un objeto.*

7. Vamos a hacer una función que reciba un código de país y nos devuelva el nombre del país. Abre **ConsultasApi.ts** y crea una función llamada **consultarNombrePais** en la que programaremos eso

```
function consultarNombrePais(codigo){
  // aquí consultaremos el nombre del país que tiene ese código
}
```

*No es necesario exportar esa función, porque solo se usará en el archivo **Funciones.ts** y por tanto, puede quedarse privada en él.*

8. De forma similar a la consulta del API **nationalize**, lanza una consulta al endpoint del API **restcountries** que hemos probado. Una vez recuperado el documento JSON devuelto por el API, accede a la traducción del nombre del país en español, y ese es el resultado que debe devolver la función.

```
async function consultarNombrePais(codigo){
    const endpoint = `https://restcountries.com/v3.1/alpha/${codigo}`
    const respuesta = await axios.get(endpoint)
    return respuesta.data[0].translations.spa.common
}
```

*Normalmente en las funciones auxiliares será más cómodo llamar a las funciones asíncronas con **async/await** en lugar de con **then/catch***

9. Vamos a modificar **consultarProbabilidades** para que los objetos que devuelve (que ahora solo tienen **country_id** y **probability**) incluyan también un campo **país** que rellenaremos con la función **consultarNombrePais**. Por sencillez, usaremos bucles para recorrerlos e inyectarles el campo **país**.

```
async function consultarProbabilidades(nombre){
    const endpoint = `https://api.nationalize.io/?name=${nombre}`
    const respuestaServidor = await axios.get(endpoint)
    const resultado = respuestaServidor.data.country
    for(let objeto of resultado){
        objeto.pais = await consultarNombrePais(objeto.country_id)
    }
    return resultado
}
```

*Tras la llamada a esta función, la lista de objetos país-probabilidad, ya tiene tres campos: **country_id, probability** y **pais***

10. Abre el archivo **ItemPaisProbabilidad.tsx** y modifica el primer **Text** de la columna de información, para que ponga **item.pais**

```
export default function ItemPaisProbabilidad({ item }) {
  return (
    <View style={styles.container}>
      <Image source={`https://flagsapi.com/${item.country_id}/flat/64.png`}
        style={styles.icono}  contentFit="contain"/>
      <View>
        <Text style={styles.textoPais}>{item.pais}</Text>
        <Text style={styles.textoProbabilidad}>
          Probabilidad: {Math.round(100*item.probability)} %
        </Text>
      </View>
    </View>
  );
}
```

11. Ejecuta la app y comprueba que todo funciona correctamente

6.- Validación del nombre

En este punto vamos a validar que el nombre no sea una cadena de texto vacía, para que no se puedan hacer consultas con el nombre vacío.

1. Añade dentro de la función **App** una función llamada **validarNombre**. Esta función:

    - Eliminará los espacios a izquierda y derecha de la variable **nombre** (eso se hace con el método **trim** del **string**)
    - Comprobará si el texto resultante es igual a la cadena vacía, retornando **true** si no es así

```
function validarNombre(){
  return nombre.trim() !== ""
}
```

2. Usa la función **validarNombre** dentro de **botonPulsado** para realizar la búsqueda si dicha función nos devuelve **true**, y mostrar una ventana emergente en caso contrario

```
function botonPulsado(){
  if(validarNombre()){
    consultarProbabilidades(nombre)
      .then( respuesta => setListaProbabilidades(respuesta) )
      .catch( error => Alert.alert("Error",error.toString()))
  }else{
    Alert.alert("Error","El nombre no puede dejarse vacío")
  }
}
```

3. Ejecuta la app y comprueba que no es posible consultar un nombre en blanco

7.- Versión TypeScript

*Como siempre, vamos a proporcionar la versión **TypeScript** de los componentes, indicando los tipos de datos que se utilizan en las llamadas a funciones*

1. En el proyecto crea una carpeta llamada **model** y dentro de ella un archivo llamado **Tipos.tsx**

> *El **modelo** de la aplicación está formado por tipos que representan los conceptos del mundo real con los que trabaja la app. En este caso, el tipo **Probabilidad** cuyos campos son **country_id, probability** y **país** forma parte del modelo*

2. En **Tipos.tsx** crea y exporta un tipo llamado **Probabilidad** que tenga los campos **country_id, probability** y **país**

```
type Probabilidad = {
    country_id: string,
    probability: number,
    pais: string
}
export {Probabilidad}
```

3. Abre **ConsultasApi.tsx** y modifica la función **consultarNombrePais** para que reciba un **string** y devuelva un **Promise<string>**

```
async function consultarNombrePais(codigo:string):Promise<string>{
  // código omitido
}
```

> ☞ ***¿Si debe devolver el nombre del país, por qué devuelve Promise<string>?*** *El motivo es que una función* ***async*** *siempre devuelve un* ***Promise<T>*** *siendo* ***T*** *el tipo de dato del resultado obtenido asíncronamente. Aquí, ese dato es el nombre del país, que es un* ***string,*** *y por eso* ***consultarNombrePais*** *debe devolver* ***Promise<string>***

4. En **Funciones.tsx** realiza las siguientes modificaciones a la función **consultarProbabilidades:**
   - El parámetro **nombre** que recibe es un **string**
   - Como devuelve una lista de objetos **Probabilidad** y la función es **async**, entonces devolverá un **Promise<Array<Probabilidad>>**

```
async function consultarProbabilidades(nombre:string): Promise<Array<Probabilidad>>{
  // código omitido
}
```

5. Abre **ItemPaisProbabilidad.tsx** y crea allí el tipo **ItemPaisProbabilidadProps** que indique que **item** es de tipo **Probabilidad**

```
type ItemPaisProbabilidadProps = {
  item: Probabilidad
}
export default function ItemPaisProbabilidad({ item }:ItemPaisProbabilidadProps) {
  // resto omitido
}
```

6. En **App.tsx** indica que la variable de estado **listaProbabilidades** es de tipo **ArrayProbabilidad**

```
export default function App(){
  const [nombre, setNombre] = useState("")
  const [listaProbabilidades, setListaProbabilidades] = useState<Array<Probabilidad>>([])
  // resto omitido
}
```

> ☞ ***¿Por qué tenemos que indicarlo ahí y en las demás variables de estado no?*** *El motivo es que el valor inicial de* ***listaProbabilidades*** *es una lista vacía, y* ***TypeScript*** *no tiene forma de saber (si no se lo indicamos explícitamente) que es de tipo* ***Array<Probabilidad>.*** *Eso hace que puedan producirse mensajes de error como este:*
>
> ```
> function botonPulsado(){
>   if(validarNombre()){
>     consultarProbabilidades(nombre)
>       .then( respuesta => setListaProbabilidades(respuesta)
> ```
>
> *Dicho mensaje se quita cuando indicamos que* ***listaProbabilidades*** *es de tipo* ***Array<Probabilidad>*** *porque* ***TypeScript*** *ya si tiene claro que ese uso es correcto*

7. Abre **App.tsx** y modifica la función **validarNombre** para indicar que devuelve un **boolean**

```
function validarNombre():boolean{
    return nombre.trim() !== ""
}
```

8.- Llamadas a Apis que tienen muchos parámetros

*En este tutorial hemos realizado dos llamadas a Apis en su forma más simple. Sin embargo, en ocasiones las apis que usan la url para pasar parámetros (como cuando se usa el método **get**) definen en ella muchos parámetros y escribirlos todos en la url del endpoint es muy engorroso.*

*Cuando la url del endpoint tiene muchos parámetros, se hacer un objeto llamado **params** (**axios** impone ese nombre) que los contenga todos, y se pasa a **axios.get** un objeto que lo contenga.*

*Vamos a modificar la función **consultarProbabilidades** para pasar los parámetros del api en un objeto **params***

1. Vete a la función **consultarProbabilidades** y quita los parámetros del endpoint

```
async function consultarProbabilidades(nombre:string) : Promise<Array<Probabilidad>>{
    const endpoint = `https://api.nationalize.io/`
    const respuestaServidor = await axios.get(endpoint)
    const resultado = respuestaServidor.data.country
    for(let objeto of resultado){
        objeto.pais = await consultarNombrePais(objeto.country_id)
    }
    return resultado
}
```

*Los parámetros que se envían al servidor en la url comienzan a partir del signo **?** y se reconocen porque cada uno posee un nombre que tiene asignado un valor*

2. Crea un objeto **params** y ponle un campo **name** con valor **nombre**

```
async function consultarProbabilidades(nombre:string) : Promise<Array<Probabilidad>>{
    const endpoint = `https://api.nationalize.io/`
    const params = { name: nombre }
    const respuestaServidor = await axios.get(endpoint)
    const resultado = respuestaServidor.data.country
    for(let objeto of resultado){
        objeto.pais = await consultarNombrePais(objeto.country_id)
    }
    return resultado
}
```

3. Pasa un objeto que contenga **params** como segundo parámetro de **axios.get**

```
async function consultarProbabilidades(nombre:string) : Promise<Array<Probabilidad>>{
    const endpoint = `https://api.nationalize.io/`
    const params = { name: nombre }
    const respuestaServidor = await axios.get(endpoint,{params})
    const resultado = respuestaServidor.data.country
    for(let objeto of resultado){
        objeto.pais = await consultarNombrePais(objeto.country_id)
    }
    return resultado
}
```

*Cuando escribimos **{params}** lo que hacemos es crear un objeto que posee una clave llamada **params** y su valor es justamente la variable **params**. O sea, es como si escribiéramos **{params: params}***

9.- Llamadas a Apis que necesitan cabeceras

*En este tutorial hemos usado solamente apis públicas y gratuitas, que no necesitan identificación. Sin embargo, las apis comerciales poseen mecanismos de seguridad para controlar su uso y algunas solicitan el envío de **cabeceras http** con información sobre la entidad que hace la llamada al api o para pasar tokens de identificación.*

*Para enviar cabeceras al servidor, crearemos un objeto llamado **headers** que las contenga y se lo pasaremos a las funciones de **axios** dentro de un objeto, como hemos hecho antes.*

*Aunque no es necesario, vamos a enviar al servidor del api **nationalize.io** las siguientes cabeceras:*

- ***User-Agent** → le mandaremos información sobre nuestra app*
- ***Accept** → indicaremos que aceptamos respuestas en formato **json***

1. En la función **consultarProbabilidades**, crea un objeto llamado **headers** con el valor de las cabeceras indicado anteriormente:

```
1.  async function consultarProbabilidades(nombre) : Promise<Array<Probabilidad>>{
2.      const endpoint = `https://api.nationalize.io/`
3.      const params = { name: nombre }
4.      const headers = {
5.                          User-Agent: "Nombres/1.0 (correo@dominio.com)"
6.                          Accept: "application/json"
7.                      }
8.      const respuestaServidor = await axios.get(endpoint,{params})
9.      const resultado = respuestaServidor.data.country
10.     for(let objeto of resultado){
11.         objeto.pais = await consultarNombrePais(objeto.country_id)
12.     }
13.     return resultado
14. }
```

2. Añade al objeto pasado segundo parámetro de **axios.get** el objeto **headers**

```
1.  async function consultarProbabilidades(nombre) : Promise<Array<Probabilidad>>{
2.      const endpoint = `https://api.nationalize.io/`
3.      const params = { name: nombre }
4.      const headers = {
5.                          User-Agent: "Nombres/1.0 (correo@dominio.com)"
6.                          Accept: "application/json"
7.                      }
8.      const respuestaServidor = await axios.get(endpoint,{params, headers})
9.      const resultado = respuestaServidor.data.country
10.     for(let objeto of resultado){
11.         objeto.pais = await consultarNombrePais(objeto.country_id)
12.     }
13.     return resultado
14. }
```

*En este ejemplo vemos cómo tenemos separados los objetos **params** y **headers**. Podemos unificarlos en un solo objeto de configuración de la llamada y pasarlo directamente a **axios.get***

3. Borra los objetos **params** y **headers,** unifíca sus datos en un solo objeto llamado **configuración** y pásalo directamente a **axios.get**

```
async function consultarProbabilidades(nombre:String) : Promise<Array<Probabilidad>>{
    const endpoint = `https://api.nationalize.io/`
    const configuracion = {
                            params: {
                                name: nombre
                            },
                            headers: {
                                User-Agent: "Nombres/1.0 (correo@dominio.com)"
                                Accept: "application/json"
                            }
                        }
    const respuestaServidor = await axios.get(endpoint,configuracion)
    const resultado = respuestaServidor.data.country
    for(let objeto of resultado){
        objeto.pais = await consultarNombrePais(objeto.country_id)
    }
    return resultado
}
```

10.- (Voluntario) – Paralelización de llamadas

*La función **consultarProbabilidades** primero accede al api **nationalize** y obtiene una lista con objetos código de país-probabilidad. A continuación, por cada uno, llama al api **restcountries** para obtener el nombre de dicho país.*

*Ahora mismo las llamadas para obtener el nombre del país se están haciendo de forma secuencial, puesto que un bucle for va recorriendo de uno en uno la lista de códigos y por cada uno, accede a **restcountries** para consultar su nombre. Por tanto, el tiempo que tarda el método en terminar es la suma de los tiempos de consultar todos los nombres.*

Si nos damos cuenta, las llamadas para obtener los nombres de los países son independientes (una no influye en la otra) y la app ganaría mucha más eficiencia si los nombres de los países se pudieran consultar todos a la vez, en paralelo, en hilos diferentes. De esta forma, el tiempo que tarda el método en terminar sería el tiempo del nombre que más tarde en consultarse, y no la suma de todos.

Vamos a hace que las llamadas que consultan los nombres se hagan en paralelo, de esta forma:

- *Quitamos el bucle for*
- *Usamos el método **map** de la lista de códigos de países-probabilidades para transformarlo en una lista de **Promises** que nos devuelven el objeto completo (código de país, nombre de país y probabilidad)*
- *Pasamos la lista obtenida a la función **Promise.all**, que nos devuelve un **Promise** que finaliza cuando todos ellos hayan finalizado*
- *Esperamos a que termine dicho **Promise** y obtendremos la lista completa de países-probabilidades con las peticiones de nombres paralelizadas*

1. En **ConsultasApi.ts** crea una función asíncrona llamada **rellenarCampoPais** que reciba un objeto **Probabilidad** y nos devuelva el objeto con el campo país rellenado a partir de la función **consultarNombrePais**

```
async function rellenarCampoPais(objeto:Probabilidad):Promise<Probabilidad>{
  objeto.pais = await consultarNombrePais(objeto.country_id);
  return objeto
}
```

2. En la función **consultarProbabilidades,** cambia el bucle **for** por una llamada al método **map** de la lista obtenida tras consultar el api **nationalize.** El método **map** recibirá la función **rellenarCampoPais**

```
async function consultarProbabilidades(nombre:String) : Promise<array<Probabilidad>>{
    const endpoint = `https://api.nationalize.io/`
    const configuracion = {
                              params: {
                                  name: nombre
                              },
                              headers: {
                                  User-Agent: "Nombres/1.0 (correo@dominio.com)"
                                  Accept: "application/json"
                          }
                           }
    const respuestaServidor = await axios.get(endpoint,configuracion)
    const resultado = respuestaServidor.data.country
    const listaPromises = resultado.map(rellenarCampoPais)
}
```

*Este paso lo que ha hecho ha sido transformar cada objeto devuelto por el api (código de país-probabilidad) en un **Promise** que al completarse, dará un objeto **Probabilidad** completado con todos sus datos*

3. Usa la función **Promise.all** para pasarle la lista de todos los **Promise** obtenidos anteriormente y esperar a que se obtenga su resultado (que será la lista de objetos **Probabilidad** completados)

```
async function consultarProbabilidades(nombre:String) : Promise<array<probabilidad>>{
    const endpoint = `https://api.nationalize.io/`
    const configuracion = {
                              params: {
                                  name: nombre
                              },
                              headers: {
                                  User-Agent: "Nombres/1.0 (correo@dominio.com)"
                                  Accept: "application/json"
                          }
                           }
    const respuestaServidor = await axios.get(endpoint,configuracion)
    const resultado = respuestaServidor.data.country
    const listaPromises = resultado.map(rellenarCampoPais)
    return await Promise.all(listaPromises)
}
```

4. Comprueba que todo sigue funcionando correctamente (debe tardar menos tiempo en obtenerse el resultado, aunque es difícil notarlo)
5. Haz un commit titulado “finalizado el tutorial 15”
6. Cambia a la rama **main** y mezcla en ella la rama **tutorial15**
7. Haz un **push** del proyecto