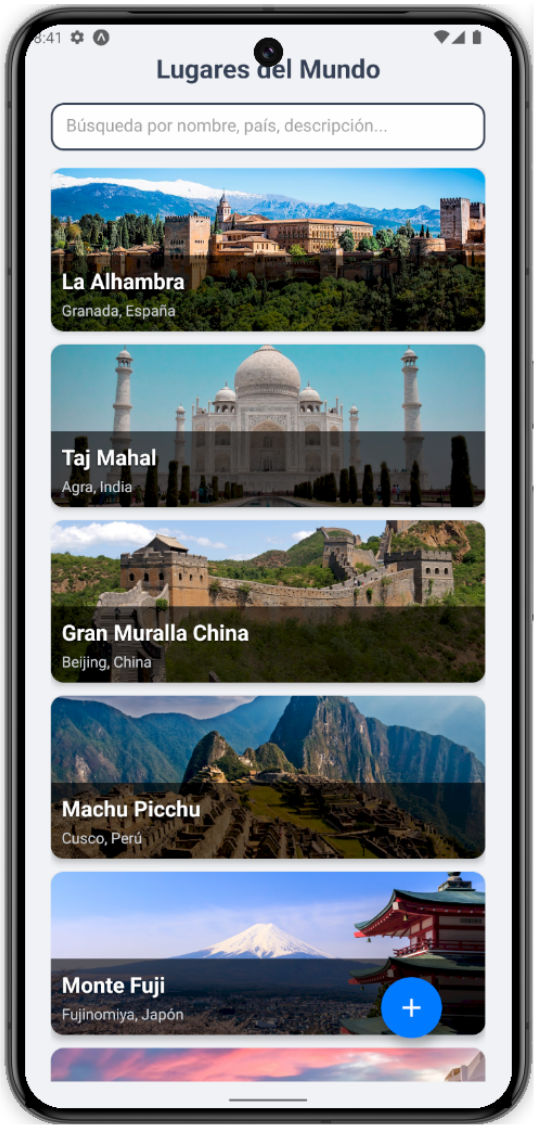
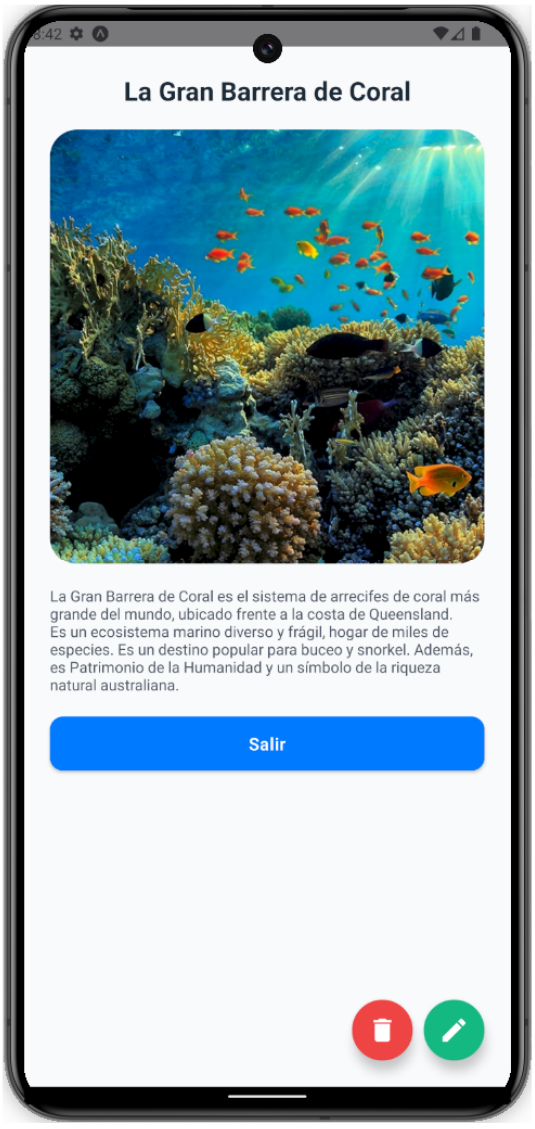
TUTORIAL 19

CUADROS DE BÚSQUEDA





En este tutorial vamos a añadir al proyecto realizado en el tutorial anterior un cuadro de búsqueda para encontrar los lugares cuyo nombre introducimos. Realizaremos varias versiones, desde el más sencillo, que solo filtre los lugares hasta más complejos que muestren sugerencias mientras escribimos.

1.- Inicio del proyecto

El punto de partida es el final del tutorial anterior

1. Abre el proyecto **lugares**
2. Instala las siguientes librerías:
   - **AutocompleteInput → npx expo install react-native-autocomplete-input**

> ☞ ***¿Para qué sirve esta librería?*** *Esta librería nos proporciona un componente llamado* ***AutocompleteInput*** *que permite implementar un cuadro de texto que muestra sugerencias mientras escribimos*

3. Crea una rama llamada **tutorial19**
4. Inicia el servidor del api → **npx json-server datos.json**

2.- Creación de un buscador sencillo

El primer buscador que vamos a implementar es un simple cuadro de texto que consulta el api cada vez que se escribe algo sobre él. Para ello nos ayudaremos de un endpoint que recibe una cadena de texto y nos devuelve todos los lugares que incluyen en algún campo (nombre, país, etc) dicho texto.

Los datos del endpoint son:

- ***url:*** *http:localhost:3000/lugares?q=**{texto buscado}***
- ***método:*** *GET*

1. Abre **CrudLugares.ts** y añade una función asíncrona llamada **buscarLugares** que reciba un texto y llame al endpoint descrito anteriormente

```
export async function buscarLugares(texto:string):Promise<Lugares>{
    const url=`http://${IP}:3000/lugares?q=${texto}`
    const respuesta = await axios.get(url)
    return respuesta.data
}
```

2. Abre **GlobalStyles.tsx** y añade el siguiente estilo **buscador**

```
buscador:{
  height:44,
  paddingHorizontal:12,
  backgroundColor:"#fefefe",
  borderWidth:2,
  borderColor:"#344055",
  borderRadius:12,
  fontSize:16,
  color: '#344055',
  marginBottom:16
},
```

3. En **components** crea un archivo llamado **BuscadorSencilo.tsx**
4. Abre **BuscadorSencillo.tsx** y teclea **rnfs+intro**
5. Coloca en **BuscadorSencillo** una variable de estado llamada **textoBusqueda** con su setter. Esa variable estará inicializada a un texto vacío y contendrá el texto que el usuario escribe en el buscador

```
export default function BuscadorSencillo() {
  const [textoBusqueda, setTextoBusqueda] = useState("")
  // resto omitido
}
```

6. Añade el prop **setListaLugares** a la función **BuscadorSencillo**:

```
ype BuscadorSencilloProps = {
    setListaLugares: React.Dispatch<React.SetStateAction<Lugares>>
}
export default function BuscadorSencillo({setListaLugares}) {
  const [textoBusqueda, setTextoBusqueda] = useState("")
  // resto omitido
}
```

7. Haz que la función **BuscadorSencillo** devuelva un **TextInput** con el estilo global **buscador** y vincúlalo a la variable de estado **textoBusqueda**

```
<TextInput
  placeholder={"Búsqueda por nombre, país, descripción..."}
  style={globalStyles.buscador}
  value={textoBusqueda}
  onChangeText={setTextoBusqueda}
/>
```

8. Añade a **BuscadorSencillo** una función **accionBuscarLugares** que:
   - Use la función **buscarLugares** para que el api proporcione los lugares que en algún campo incluyen la variable **textoBusqueda**
   - En caso de encontrarlos, la lista obtenida se guardará en **listaLugares**
   - Si no se encuentran, se mostrará un mensaje de error en la consola

```
function accionBuscarLugares(){
    buscarLugares(textoBusqueda)
      .then(lugares => setListaLugares(lugares))
      .catch(error => console.log(error.toString()))
}
```

☞ ***¿Por qué no se muestra el error en una ventana emergente?*** *Si lo hiciéramos así se mostraría un error cada vez que se escriba en el buscador, y eso sería muy pesado. Es mejor para no saturar al usuario mostrar el error en la terminal*

9. Haz que la función **accionConsultarLugares** se llame automáticamente cada vez que cambia el valor de la variable **textoBusqueda**

```
export default function BuscadorSencillo({setListaLugares}:BuscadorSencilloProps) {
  const [textoBusqueda, setTextoBusqueda] = useState("")
  useEffect(accionBuscrLugares, [textoBusqueda])
  // resto omitido
}
```

10. Añade a **App** un componente **BuscadorSencillo**

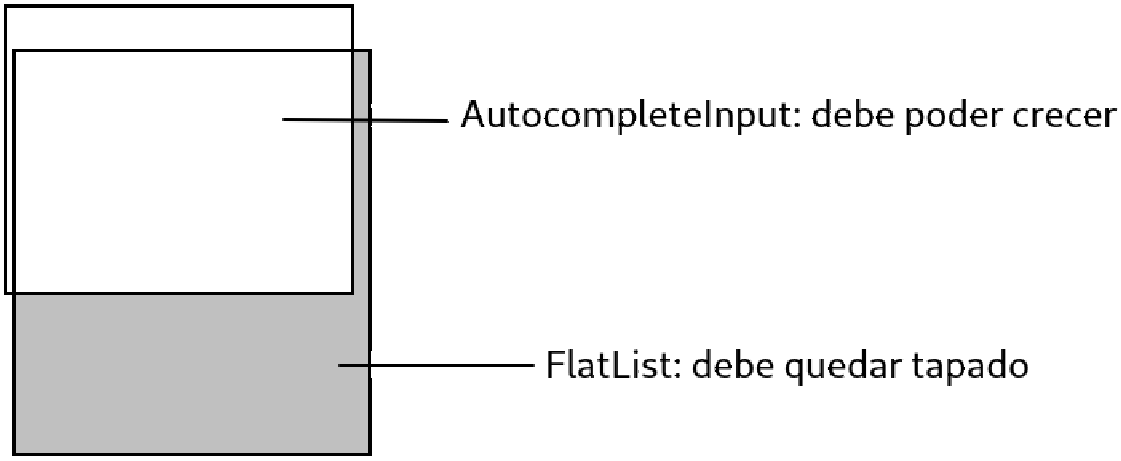
```
export default function App() {
  // inicio omitido
  return (
    <View style={styles.contenedor}>
      <Text style={globalStyles.titulo}>Lugares del Mundo</Text>
      <BuscadorSencillo setListaLugares={setListaLugares}/>
      // resto omitido
}
```

3.- Creación de un buscador con sugerencias de texto

*A continuación vamos a programar un buscador que nos sugiere una lista con los lugares que incluyen algún campo con lo que escribimos en él. El componente que usaremos será **AutocompleteInput**, que muestra un **TextInput** y un **FlatList** conforme escribimos en él. Sus principales props son:*

- ***data:** Es la lista con las sugerencias que se muestran en el **FlatList***
- ***defaultValue:** Es el valor que se muestra inicialmente en el **TextInput***
- ***onChangeText:** Es la función que se ejecuta cuando cambia lo que se escribe en el **TextInput***
- ***flatListProps:** Es un objeto que contiene los props **renderItem** y **keyExtractor** que usará el **FlatList** de las sugerencias*
- ***inputContainerStyle:** Es el estilo del **TextInput***
- ***placeHolder:** Es el placeholder del **TextInput***

*Es importante indicar que para colocar el **AutocompleteInput** necesitaremos utilizar el posicionamiento absoluto, ya que es necesario un componente que "crezca" (cuando se abren las sugerencias) y "tape" parte del contenido que viene tras él.*



1. En **components** crea un archivo llamado **BuscadorSugerencias.tsx**
2. Abre **BuscadorSugerencias.tsx** y teclea **rnfs + intro**
3. Añade a **BuscadorSugerencias.txt** estos elementos:
    - La variable de estado **textoBusqueda** y su setter **setTextoBusqueda**.
    - Una variable de estado **sugerencias** con su setter **setSugerencias**. Inicialmente será una lista vacía y es la lista de sugerencias que se muestran en el buscador
    - Un prop con la función **setLugarSeleccionado**

```
type BuscadorSugerenciasProps = {
    setLugarSeleccionado: React.Dispatch<React.SetStateAction<Lugar|undefined>>
}
export default function BuscadorSugerencias({setLugarSeleccionado}:BuscadorSugerenciasProps) {
  const [textoBusqueda, setTextoBusqueda]=useState("")
  const [sugerencias,setSugerencias] = useState([])
  return
    // resto omitido
}
```

4. Añade en **BuscadorSugerencias** la función **actualizarSugerencias**, que funcionará de esta forma:
    - Si **textoBusqueda** está vacío, la lista de sugerencias se pondrá a una lista vacía
    - Si no está vacío, se llamará a **buscarLugares** para encontrar los lugares que en algún campo contienen **textoBusqueda**, y esos lugares se guardarán en la variable **sugerencias**

```
function actualizarSugerencias(){
  if(textoBusqueda===""){
      setSugerencias([])
  }else {
      buscarLugares(textoBusqueda)
        .then( lugares => setSugerencias(lugares))
  }
}
```

5. Haz que la función **actualizarSugerencias** se llame automáticamente cada vez que cambie la variable **textoBusqueda**

```
export default function BuscadorSugerencias({setLugarSeleccionado}:BuscadorSugerenciasProps) {
  const [textoBusqueda, setTextoBusqueda]=useState("")
  const [sugerencias, setSugerencias] = useState<Lugares>([])
  useEffect(actualizarSugerencias, [textoBusqueda])
  // resto omitido
}
```

6. Crea en **BuscadorSugerencias** los siguientes estilos, para colocar el contenedor del buscador en una posición fija de la pantalla por encima del resto de componentes, y para dar estilo al cuadro de texto del buscador

```
contenedor:{
    position:"absolute",
    top:60,
    left:24,
    width:"100%",
    zIndex:10
},
cuadroTexto:{
    backgroundColor: "#fefefe",
    fontSize: 16,
    color: "#344055",
    borderWidth: 2,
    borderColor: "#344055",
    borderRadius: 12,
    paddingHorizontal:10,
    paddingVertical:4
}
```

7. A continuación añade una función (vacía de momento) llamada **getEtiquetaSugerencia** que reciba un objeto **Lugar** y nos devuelva el componente donde se mostrará la sugerencia.

```
function getEtiquetaSugerencia(lugar:Lugar):React.ReactElement{
  // devuelve el componente donde se muestra la sugerencia de un lugar
}
```

*Las sugerencias son componentes que muestran un objeto **Lugar** sugerido. Estos componentes aparecen en un **FlatList** que lleva el **AutocompleteInput** cada vez que el usuario escribe algo en el buscador.*

8. Haz que la función **BuscadorSugerencias** devuelva un **View** con el estilo **contenedor** y un **AutocompleteInput** con estos props:

   - **data:** La lista de sugerencias que hay en **sugerencias**
   - **defaultValue:** La variable **textoBusqueda**
   - **onChangeText:** El setter **setTextoBusqueda**
   - **inputContainerStyle:** El estilo **cuadroTexto**
   - **placeHolder:** Un mensaje indicando el criterio de búsqueda
   - **flatListProps:** Un objeto que tenga estos dos campos:
     - **keyExtractor:** Una lambda que toma un lugar y devuelve su id
     - **renderItem:** Una lamda que recibe **{item}** y pase **item** a la función **getEtiquetaSugerencia**

```
1.  export default function BuscadorSugerencias({setLugarSeleccionado}:BuscadorSugerencias) {
2.    // inicio omitido
3.    function getEtiquetaSugerencia(item:Lugar){
4.    }
5.    return (
6.      <View style={styles.contenedor}>
7.          <AutocompleteInput
8.              data = {sugerencias}
9.              defaultValue={textoBusqueda}
10.             onChangeText={setTextoBusqueda}
11.             inputContainerStyle={styles.cuadroTexto}
12.             placeholder={"Búsqueda por nombre, país, descripción..."}
13.             flatListProps={{
14.                 keyExtractor: lugar => lugar.id,
15.                 renderItem: ({item}) => getEtiquetaSugerencia(item)
16.             }}
17.         />
18.     </View>
19.   )
20. }
```

*Mucho cuidado con el prop **flatListProps**, porque recibe un objeto. De ahí la doble llave **{ { } }** y también esa coma tras el campo **keyExtractor**, que suele pasar desapercibida*

9. En **components** crea un archivo llamado **VisorSugerencia.tsx**

*El componente **VisorSugerencia** va a ser el componente donde se mostrará una sugerencia (recuerda que el **AutocompleteInput** tiene un **FlatList** donde se muestran las sugerencias, y cada sugerencia es un **Lugar**). Este componente va a recibir un prop de tipo **Lugar** y mostrará la correspondiente sugerencia, que de momento, será el nombre del lugar.*

10. Abre **VisorSugerencia** y en él teclea **rnfs+intro**
11. Añade el siguiente estilo **contenedor** a **VisorSugerencia.tsx**

```
1.  contenedor:{
2.      paddingHorizontal:10,
3.      paddingVertical:5
4.  }
```

12. Añade a **VisorSugerencia** un prop llamado **lugar,** de tipo **Lugar**

```
type VisorSugerenciaProps = {
    lugar:Lugar
}
export default function VisorSugerencia({lugar}:VisorSugerenciaProps) {
  // resto omitido
}
```

13. Programa **VisorSugerencia** para que devuelva un contenedor con el estilo **contenedor** y un **Text** cuyo texto sea el nombre del lugar recibido en el prop

```
export default function VisorSugerencia({lugar}:VisorSugerenciaProps) {
  return (
    <View style={styles.contenedor}>
      <Text>{lugar.nombre}</Text>
    </View>
  )
}
```

14. Abre **BuscadorSugerencias.tsx** y programa la función **getEtiquetaSugerencia** de forma que devuelva un **Pressable** que contenga un **VisorSugerencia.** Al pulsar en el **Pressable:**

    - Se pondrá el **item** como lugar seleccionado → *para así confirmar la sugerencia y que se muestre el detalle del lugar sugerido*
    - El texto de búsqueda se dejará vacío → *para así "reiniciar" el estado del buscador y que no quede rastro de la búsqueda que hemos hecho*

```
function getEtiquetaSugerencia(lugar:Lugar):React.ReactElement{
  return (
      <Pressable onPress={ ()=>{
              setLugarSeleccionado(lugar)
              setTextoBusqueda("")
          }
      }>
          <VisorSugerencia lugar={lugar}/>
      </Pressable>
  )
}
```

15. Abre **App.tsx** y cambia **BuscadorSencillo** por **BuscadorSugerencias**

```
export default function App() {
  // inicio omitido
  return (
    <View style={styles.contenedor}>
      <Text style={globalStyles.titulo}>Lugares del Mundo</Text>
      <BuscadorSugerencias setLugarSeleccionado={setLugarSeleccionado}/>
      // resto omitido
}
```

16. Ejecuta la app y comprueba que al pulsar en una sugerencia, se muestra el detalle del lugar correspondiente, pero hay que pulsar "dos veces" para que se active.

Este efecto se debe al teclado, que necesita una pulsación para desactivarse, y así la segunda pulsación es la que de verdad activa la sugerencia.

*Vamos a solucionarlo añadiendo añadiendo el prop **keyboardShouldPersistTaps** del **FlatList** interno del **AutocompleteInput***

15. Añade **keyboardShouldPersistTaps:"handled"** al objeto de **flatListProps**

```
<AutocompleteInput
  data={sugerencias}
  defaultValue={textoBusqueda}
  onChangeText={setTextoBusqueda}
  inputContainerStyle={styles.cuadroTexto}
  placeholder={"Búsqueda por nombre, país, descripción..."}
  flatListProps={{
    renderItem: ({ item }) => getEtiquetaSugerencia(item),
    keyExtractor: (lugar) => lugar.id,
    keyboardShouldPersistTaps:"handled"
  }}
/>
```

☞ ***¿Para qué sirve keyboardShouldPersistTaps?*** *Este prop controla lo que sucede cuando se pulsa en un lugar de la pantalla con el teclado abierto.*

- *¿La pulsación debe cerrar el teclado e ignorar el componente de la pantalla donde se pulsa? → Si es así, se pondrá el valor **"never"***
- *¿La pulsación debe ignorar que está el teclado abierto?*
  - *Valor **handled** → Si se ha pulsado sobre un elemento, la pulsación se aplica a ese elemento y se cierra el teclado (en caso de no haber elemento, el teclado no se cierra)*
  - *Valor **always** → Es como el anterior, pero el teclado no se cierra en ningún caso*

16. Ejecuta la app y comprueba que ya puedes pulsar en una sugerencia, y que dicha pulsación cierra el teclado.
17. En **App** encierra el **FlatList** en un **View** con margen superior **60**

```
export default function App() {
  // inicio omitido
  return (
    <View style={styles.contenedor}>
      <Text style={styles.titulo}>Lugares del Mundo</Text>
      <BuscadorSugerencias setLugarSeleccionado={setLugarSeleccionado}/>
      <View style={{marginTop:60}}>
        <FlatList
          data={listaLugares}
          keyExtractor={(lugar) => lugar.id.toString()}
          renderItem={({ item }) => getItemLugar(item)}
        />
      </View>
      // resto omitido
}
```

*Tenemos que hacer esto porque nuestro buscador ahora usa posicionamiento absoluto y está anclado en la pantalla en una posición fija. Esto hace que a la hora de dibujarse, no se tenga en cuenta con respecto a los demás elementos, y por tanto, es necesario separar manualmente el **FlatList** del título*

18. Ejecuta la app y comprueba que el buscador ya si está claramente separado del resto de la interfaz

4.- Mejora del estilo de las sugerencias

*Puesto que cada sugerencia recibe un objeto **Lugar** que se muestra en un componente, podemos mejorar el estilo de ese componente para así poder mostrar una lista de sugerencias más chula que simples etiquetas de texto.*

1. Abre **VisorSugerencia.tsx** y sin mirar la solución, crea el siguiente diseño


```
export default function VisorSugerencia({lugar}:VisorSugerenciaProps) {
  return (
    <View style={styles.contenedor}>
      <Image source={lugar.foto} style={styles.foto}/>
      <View>
        <Text style={styles.nombre}>{lugar.nombre}</Text>
        <Text style={styles.detalle}>{lugar.ciudad}, {lugar.pais}</Text>
      </View>
    </View>
  )
}
```

2. Ejecuta la app y comprueba que el diseño de las sugerencias ahora es mucho más vistoso

*Aunque el aspecto ha mejorado mucho, vemos que el **FlatList** donde se muestran las sugerencias tiene un aspecto rectangular que no queda muy bien. Vamos a mejorarlo usando el prop **listContainerStyle** del **AutocompleteInput,** que sirve para definir el estilo de un **View** interno que contiene al **FlatList** de sugerencias*

3. Abre **BuscadorSugerencias.tsx** y añade el estilo **estiloFlatList**, que se va a asignar al estilo del **FlatList** que lleva dentro el **AutocompleteInput**

```
estiloFlatList: {
  backgroundColor: '#fff',
  borderRadius:12,
  marginTop: 4,
  elevation: 3,
},
```

4. Coloca dicho estilo al prop **listContainerStyle** del **AutocompleteInput**

```
<AutocompleteInput
  data={sugerencias}
  defaultValue={textoBusqueda}
  onChangeText={setTextoBusqueda}
  inputContainerStyle={styles.cuadroTexto}
  placeholder={"Búsqueda por nombre, país, descripción..."}
  listContainerStyle = {styles.estiloFlatList}
  flatListProps={{
    renderItem: ({ item }) => getEtiquetaSugerencia(item),
    keyExtractor: (lugar) => lugar.id,
    keyboardShouldPersistTaps:"handled"
  }}
/>
```

5. Ejecuta la app y verás que el “envoltorio” de las sugerencias ha cambiado, pero se ve un molesto rectángulo gris que las encierra

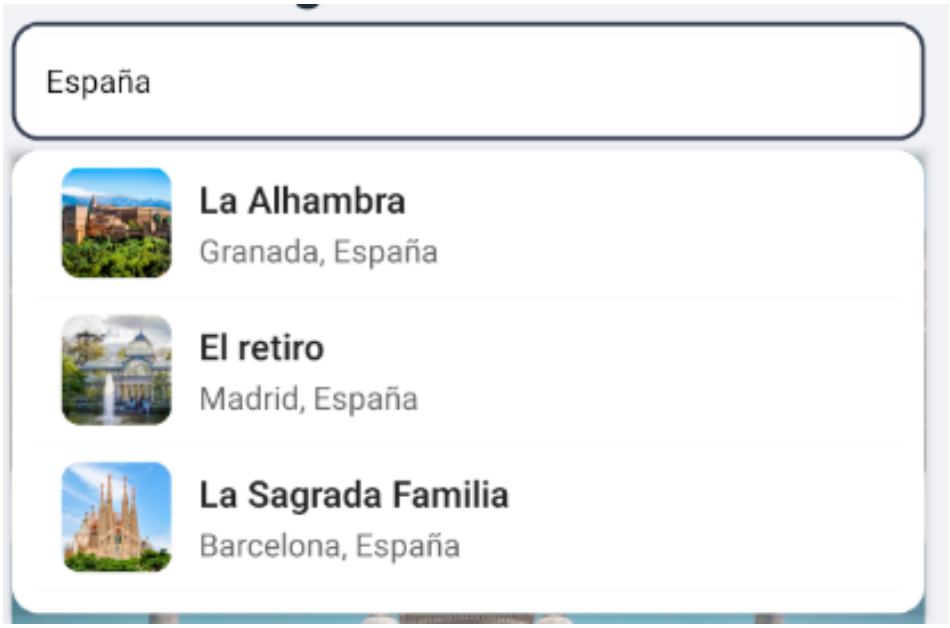
*El motivo es que al dar un estilo al **View** que encierra al **FlatList** interno del **AutocompleteInput,** se activa un estilo por defecto para dicho **FlatList** que incluye el borde rectangular gris.*

*Se lo quitaremos pasando en el **flatListProps** el estilo **{borderWith:0}***

6. En el objeto definido en el **flatListProps** pasa un campo **style** con valor la regla **{borderWith:0}**

```
<AutocompleteInput
  data={sugerencias}
  defaultValue={textoBusqueda}
  onChangeText={setTextoBusqueda}
  inputContainerStyle={styles.cuadroTexto}
  placeholder={"Búsqueda por nombre, país, descripción..."}
  listContainerStyle = {styles.estiloFlatList}
  flatListProps={{
    renderItem: ({ item }) => getEtiquetaSugerencia(item),
    keyExtractor: (lugar) => lugar.id,
    keyboardShouldPersistTaps:"handled"
    style: {borderWidth:0},
  }}
/>
```

7. Abre **VisorSugerencia.tsx** y quita la regla **borderBottomWidth:1** del estilo **contenedor**
8. Ejecuta la app y comprueba que ya se muestra todo correctamente



9. Haz un commit titulado “finalizado el tutorial 19”
10. Sitúate en la rama **main** y mezcla sobre ella la rama **tutorial 19**
11. Haz **push**