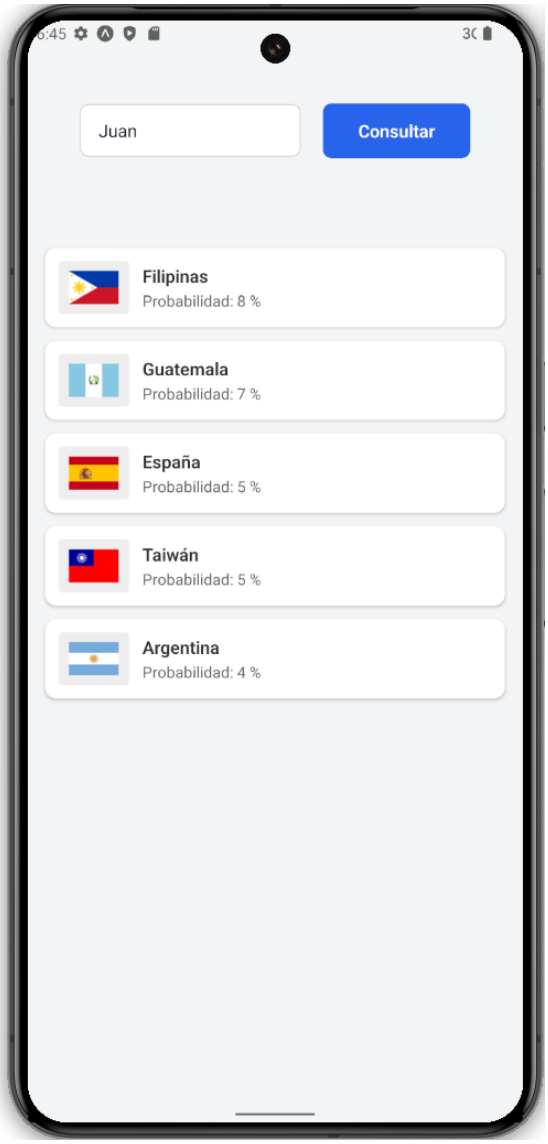
TUTORIAL 16

DISEÑO DE LA INTERFAZ POR CAPAS



En el tutorial anterior hicimos una app que consulta un api, pero los resultados tardan un tiempo apreciable en obtenerse y queremos mostrar al usuario un indicador de progreso. Para conseguirlo, haremos un diseño de la interfaz en capas, de forma que según la situación, se mostrará la capa apropiada (inicio, cargando y resultados)

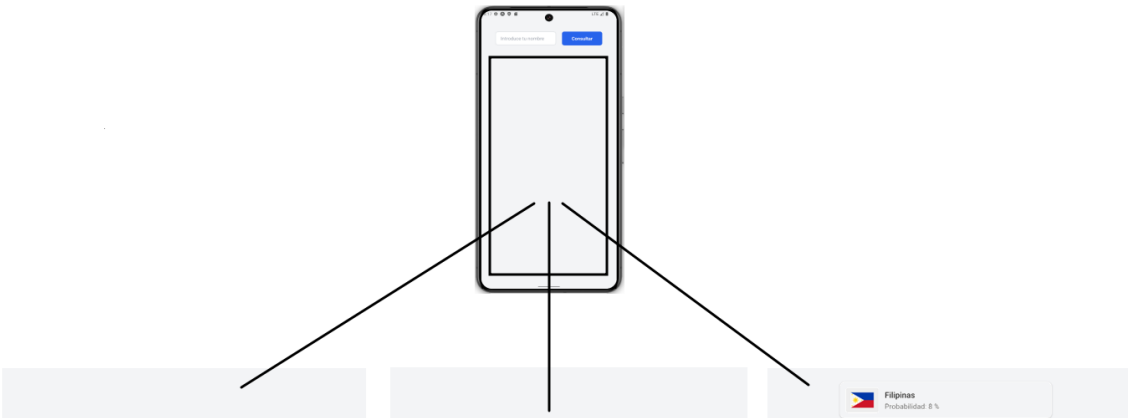
1.- Diseño de la interfaz por capas

*Si ejecutamos la app del tutorial anterior veremos que la lista de resultados tarda un poco en aparecer, y eso se debe a que la consulta al API tarda unos pocos segundos en completarse. Puesto que hemos usado **await** para consultar el API, el móvil está libre y puede hacer otra cosa mientras se consulta el API en segundo plano. Por ejemplo, puede mostrar un indicador de progreso.*

Vamos a hacer que la zona inferior de la pantalla conste de tres capas, de forma que en cada momento se muestre la capa correspondiente:

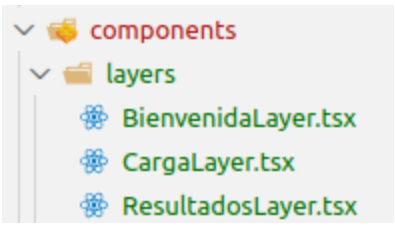
- ***Capa1*** → *Será una pantalla de bienvenida con instrucciones sobre la app*
- ***Capa 2*** → *Mostrará un indicador de progreso, y se hará visible cuando se pulse el botón de buscar*
- ***Capa 3*** → *Mostrará la tabla de resultados.*

Cada una de estas capas va a ser un componente que diseñaremos en su correspondiente archivo.



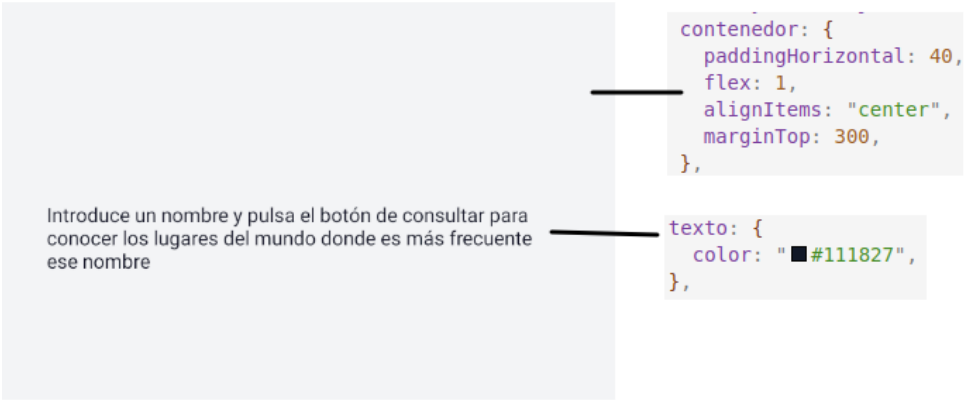
CAPA 1: BienvenidaLayer | CAPA 2: CargaLayer | CAPA 3: ResultadosLayer

1. Abre el proyecto del tutorial anterior y crea rama una llamada **tutorial16**
2. Dentro de la carpeta **components** crea una carpeta llamada **layers,** y dentro de ella haz tres archivos llamados **BienvenidaLayer.tsx, CargaLayer.tsx** y **ResultadosLayer.tsx**



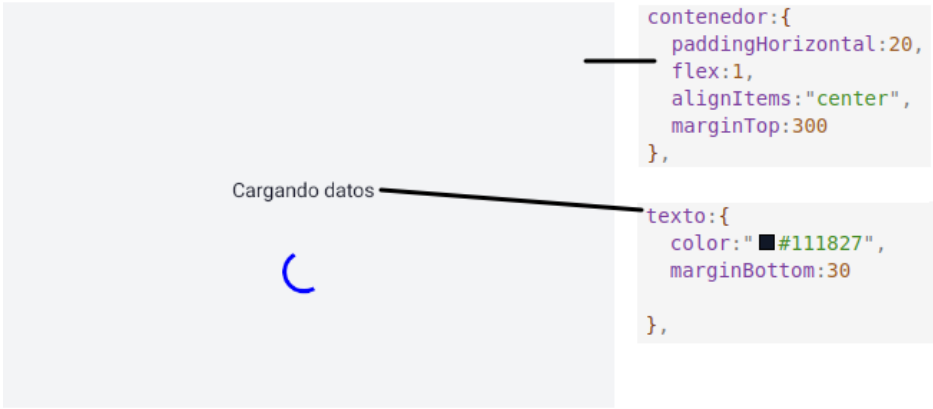
3. Abre **BienvenidaLayer.tsx**, borra todo lo que hay y teclea **rnfs+Intro**

4. Como ejercicio, programa **BienvenidaLayer** de la siguiente forma, sin mirar la solución (que viene después)




```
export default function BienvenidaLayer() {
  return (
    <View style={styles.contenedor}>
      <Text style={styles.texto}>
        Introduce un nombre y pulsa el botón de consultar para conocer los
        lugares del mundo donde es más frecuente encontrarlo
      </Text>
    </View>
  );
}
```

5. Abre **CargaLayer.tsx**, borra todo lo que hay y teclea **rnfs+Intro**
6. Como ejercicio, programa **CargaLayer** de esta forma, sin mirar la solución. El indicador de progreso es un **ProgressIndicator** que tiene estos dos props:
   - **size** → "large"
   - **color** → "#000ff"




```
export default function CargaLayer() {
  return (
    <View style={styles.contenedor}>
      <Text style={styles.texto}>Cargando datos</Text>
      <ActivityIndicator size={"large"} color={"#0000ff"}/>
    </View>
  )
}
```

7. Abre **ResultadosLayer.tsx**, borra todo lo que hay y teclea **rnfs+Intro**

8. Mueve el **FlatList** que hay en **App.tsx** al **return** que hay en **ResultadosLayer** y añade **listaProbabilidades** como prop a **ResultadosLayer**

```
export default function ResultadosLayer({listaProbabilidades}) {
  return (
    <FlatList
        data={listaProbabilidades}
        renderItem={ItemPaisProbabilidad}
        keyExtractor={ item => item.country_id}
        ListEmptyComponent={ () => <Text style="{{margin:"auto"}}">No se han encontrado
                                                                  resultados</Text>}/>
      />
  )
}
```

*Ahora que ya tenemos programadas las tres capas, vamos a colocarlas dentro del componente principal **App.tsx***

9. Abre **App.tsx** y crea una variable de estado llamada **capaActiva,** con valor inicial **1** (porque inicialmente se mostrará **BienvenidaLayer**) y su correspondiente setter **setCapaActiva**

```
export default function App(){
  const [nombre, setNombre] = useState("")
  const [listaProbabilidades, setListaProbabilidades] = useState([])
  const [capaActiva, setCapaActiva] = useState(1)
  // resto omitido
}
```

10. Dentro de la función **App,** crea una función llamada **getCapaActiva**, que devuelva la etiqueta correspondiente a la capa guardada en la variable **capaActiva** (usamos asignación condicional, pero se puede usar un if-else)

```
export default function App(){
  // inicio omitido
  async function botonPulsado(){
    // código omitido
  }
  function getCapaActiva(){
    return capaActiva === 1 ? <BienvenidaLayer/> :
           capaActiva === 2 ? <CargaLayer/> :
           capaActiva === 3 ? <ResultadosLayer listaProbabilidades={listaProbabilidades}/> :
                              <View/> // si capaActiva no es 1,2 o 3, se retorna un View vacío
  }
  return (
    // resto omitido
  )
}
```

11. Añade a **App.tsx** un estilo llamado **contenedorCapas** que simplemente lo convierta en un contenedor flex. Este estilo es el que se va a colocar a la zona donde estarán contenidas las capas

```
const styles = StyleSheet.create({
  // estilos omitidos
  contenedorCapas:{
    flex:1
  }
});
```

12. A continuación, dentro del return de la función **App**, coloca un **View** después de la fila de botones. Dicho **View** tendrá el estilo **contenedorCapas**

*Dentro de este **View** se va a colocar la capa que corresponda, según la variable **capaActiva**, y dicha etiqueta nos la da la función **getCapaActiva***

```
1.  export default function App(){
2.    // inicio omitido
3.    return (
4.      <View style={styles.contenedorPrincipal}>
5.        <View style={styles.fila}>
6.          <TextInput
7.            value={nombre}
8.            onChangeText={setNombre}
9.            placeholder={"Introduce tu nombre"}
10.           style={styles.cuadroTexto}
11.           placeholderTextColor={"#9CA3AF"}/>
12.         <Pressable
13.           onPress={botonPulsado}
14.           style={ ({pressed}) => pressed?styles.botonPresionado : styles.boton }>
15.           <Text style={styles.textoBoton}>Consultar</Text>
16.         </Pressable>
17.       </View>
18.       <View style={styles.contenedorCapas}>
19.       </View>
20.     </View>
21.   )
22. }
```

13. Dentro del **View** contenedor de capas, abre un bloque de llaves y llama en su interior a la función **getCapaActiva**, para que se inserte en ese punto la etiqueta que nos devuelve dicha función, y que se corresponde con la capa que debe ser mostrada según la variable **capaActiva**

```
1.  export default function App(){
2.    // resto omitido
3.    return (
4.      <View style={styles.contenedorPrincipal}>
5.        <View style={styles.fila}>
6.          <TextInput
7.            value={nombre}
8.            onChangeText={setNombre}
9.            placeholder={"Introduce tu nombre"}
10.           style={styles.cuadroTexto}
11.           placeholderTextColor={"#9CA3AF"}/>
12.         <Pressable
13.           onPress={botonPulsado}
14.           style={ ({pressed}) => pressed?styles.botonPresionado : styles.boton }>
15.           <Text style={styles.textoBoton}>Consultar</Text>
16.         </Pressable>
17.       </View>
18.       <View style={styles.contenedorCapas}>
19.         {
20.           getCapaActiva()
21.         }
22.       </View>
23.     </View>
24.   )
25. }
```

*Ya lo tenemos todo preparado, y lo único que falta es actualizar la variable **capaActiva** en los momentos apropiados del código, según lo que esté sucediendo*

14. Modifica la función **botonPulsado** de forma que:
    - Antes de comenzar la carga se ponga la capa activa al valor **2**.
    - Si hay un error, la capa activa se ponga al valor **1**
    - Si la carga termina sin error, la capa activa se ponga al valor **3**

```
function botonPulsado(){
  if(validarNombre()){
    setCapaActiva(2)
    consultarProbabilidades(nombre)
      .then( respuesta => {
        setListaProbabilidades(respuesta)
        setCapaActiva(3)
      })
      .catch( error => {
          Alert.alert("Error",error.toString())
          setCapaActiva(1)
        }
      )
  }else{
    Alert.alert("Error","El nombre no puede dejarse vacío")
  }
}
```

15. Ejecuta la app y comprueba que las capas se actualizan correctamente

6.- Impedir la entrada de datos durante la carga

Pese a que todo parece funcionar bien, mientras se realiza la consulta, podemos escribir (si nos da tiempo) otro nombre y realizar otra consulta pulsando el botón.

*Para que esto no suceda debemos deshabilitar el cuadro de texto y el botón de búsqueda mientras se cargan los datos. Lo haremos teniendo en cuenta que durante la carga la capa activa vale **2***

1. Añade los siguientes props a esos componentes:
   - **TextInput** → su prop **editable** valdrá **{capaActiva!==2}**
   - **Pressabe** → su prop **disabled** valdrá **{capaActiva===2}**

```
export default function App(){
  // inicio omitido
  return (
    <View style={styles.contenedorPrincipal}>
      <View style={styles.fila}>
        <TextInput
          value={nombre}
          onChangeText={setNombre}
          editable={capaActiva!==2}
          placeholder={"Introduce tu nombre"}
          style={styles.cuadroTexto}
          placeholderTextColor={"#9CA3AF"}/>
        <Pressable
          onPress={botonPulsado}
          disabled={capaActiva===2}
          style={ ({pressed}) => pressed?styles.botonPresionado : styles.boton }>
          <Text style={styles.textoBoton}>Consultar</Text>
        </Pressable>
      </View>
      <View style={styles.contenedorCapas}>
        {
          getCapaActiva()
        }
      </View>
    </View>
  )
}
```

*Observa que los diseñadores de React Native decidieron poner **editable** al **TextInput** y **disabled** al **Pressable***

7.- Versión TypeScript

*Vamos a programar la versión **TypeScript** de las capas*

1. Abre **ResultadosLayer.tsx** y crea el tipo **ResultadosLayerProps**, indicando que **listaProbabilidades** es de tipo **Array<Probabilidad>**

```
type ListaProbabilidadesProps = {
  listaProbabilidades: Array<Probabilidad>
}
export default function ResultadosLayer({listaProbabilidades}:ListaProbabilidadesProps) {
   // resto omitido
}
```

2. En **App.tsx** modifica **getCapaActiva** para que devuelva un **ReactNode**

```
function getCapaActiva():ReactNode{
  return capaActiva === 1 ? <BienvenidaLayer/> :
     capaActiva === 2 ? <CargaLayer/> :
     capaActiva === 3 ? <ResultadosLayer listaProbabilidades={listaProbabilidades}/>
                      : <View/>
```

3. Haz un commit titulado “Finalizado el tutorial 16”
4. Mezcla en la rama **main** la rama **tutorial16**
5. Haz un **push** de la rama **main**