

Programación Multimedia y Dispositivos Móviles (UP6)

Autores: Antonio Calabuig Puigvert y Sebastián Villa Ponce

Centro: IES Salvador Gadea

Departamento: Informática

Ciclo: Desarrollo de Aplicaciones Multiplataforma

Fecha: 10/01/2026

1. Introducción y base teórica

1.1. ¿Qué es un Provider y cómo funciona?

¿Qué hace un Provider?

¿Qué NO debe hacer?

1.2. ¿Qué son un State y un Ref?

useState

useRef

Tabla comparativa clara

1.3. ¿Qué es y cómo funciona una promesa?

Estados de una promesa

Ejemplo simple

Por qué son esenciales en permisos

1.4. ¿Qué es una suscripción y cómo funciona?

Estructura general

En nuestra aplicación(AppState)

Por qué hay que limpiar la suscripción

1.5. Conclusiones teóricas

2. Primeros pasos: Creación APP

3. Acciones - Permisos de geolocalización

4. Gestor de estados con Zustand

4. Provider - Permission checker

1. Introducción y base teórica

- Gestión de permisos con Zustand
- Gestión de los estados (ciclos de vida) de la aplicación
- Gestión de un `Provider`
- Creación de código utilizable
- `Expo Location`
- Gestión de permisos
- Acciones
- Estado de los permisos
- Abrir ajustes en caso de tener acceso denegado

1.1. ¿Qué es un Provider y cómo funciona?

Un **Provider** es un componente cuyo objetivo **no es pintar UI**, sino **preparar, mantener o exponer información global** al resto de la aplicación.

Es un componente de infraestructura, no uno de renderización de pantalla.

¿Qué hace un Provider?

- Inicializa lógica global (permisos, listeners, sockets, auth...)
- Mantiene estado compartido o accede a él
- Ejecuta efectos que deben vivir "mientras la app está viva"
- Envuelve a otros componentes

¿Qué NO debe hacer?

- Navegar (`router.replace`)
- Decidir qué pantalla se muestra
- Renderizar lógica de negocio visual

1.2. ¿Qué son un State y un Ref?

useState

Sirve para **datos que afectan a la UI**.

useRef

Sirve para **recordar cosas sin afectar a la UI**.

Tabla comparativa clara

Característica	useState	useRef
Provoca render	✓ Sí	✗ No
Persiste entre renders	✓ Sí	✓ Sí
Se usa para UI	✓ Sí	✗ No
Se puede mutar directamente	✗ No	✓ Sí (<code>ref.current</code>)
Ideal para	estado visible	memoria interna
Ejemplo típico	contador, permisos	estado anterior, timers



Si cambiarlo debería redibujar la pantalla → State
Si solo lo necesito para lógica interna → Ref

1.3. ¿Qué es y cómo funciona una promesa?

Una **promesa** representa **un valor que todavía no existe**, pero existirá en el futuro.

Estados de una promesa

- `pending` → esperando
- `fulfilled` → éxito
- `rejected` → error



Una promesa es una respuesta que no tenemos todavía que sabemos que tipo tendrá o que error será capaz de generar

Ejemplo simple

```
const status = await requestLocationPermission();
```

Aquí:

- el sistema operativo decide
- React espera
- cuando llega la respuesta, continúa

Por qué son esenciales en permisos

Los permisos:

- dependen del usuario
 - del SO
 - del tiempo
- **no pueden ser síncronos en el hilo principal**

1.4. ¿Qué es una suscripción y cómo funciona?

Una **suscripción** es registrarse para recibir eventos cuando algo cambia.

Estructura general

1. Te suscribes
2. El sistema te notifica
3. Reaccionas
4. Te desuscribes

En nuestra aplicación(**AppState**)

```
AppState.addListener('change', callback)
```

- El SO emite eventos
- Tu app escucha
- Reaccionas al cambio

Por qué hay que limpiar la suscripción

Si no:

- se duplican listeners
- se ejecuta código varias veces
- hay fugas de memoria

1.5. Conclusiones teóricas

Elemento	Decide estado	Decide navegación	Renderiza UI
Sistema operativo	✓	✗	✗
Expo Location	✗	✗	✗
Store (Zustand)	✓	✗	✗
Provider	✗	✗	✗
Pantallas	✗	✓ (declarativo)	✓
Router	✗	Ejecuta	✗

2. Primeros pasos: Creación APP

1. Como siempre, creamos nuestra aplicación con el comando de terminal en nuestro directorio de proyectos y lanzamos la aplicación.

```
npx create-expo-app@latest maps-app  
npm start -c
```

2. Continuamos con la reestructuración de la aplicación base (limpiar estructuras sobrantes y crear las necesarias).

- a. Creamos la carpeta `presentation/components/shared` en la raíz.
- b. Movemos el fichero `themed-text.tsx` de la carpeta `components/ui` a `presentation/components/shared`, en caso de que se nos notifique `Actualizar las importaciones` confirmamos.
- c. Movemos la carpeta `hooks` de raíz a la carpeta `presentation`, eventualmente eliminaremos sus ficheros. Actualizamos de ser necesario la importación en `themed-text.tsx` a `import { useThemeColor } from '@presentation/hooks/use-theme-color';`

- d. Eliminamos de la carpeta `app` la carpeta de `(tabs)` y el `modal.tsx` . Dentro del fichero `layout.tsx` borramos las líneas del `Stack.Screen` .

```
<Stack>
  <Stack.Screen name="(tabs)" options={{ headerShown: false }} /
>
  <Stack.Screen name="modal" options={{ presentation: 'modal', title: 'Modal' }} />
</Stack>
```

- e. Creamos dentro de `app` el fichero `loading/index.tsx` , creamos un nuevo componente y le cambiamos el nombre a `LoadingScreen` .

```
import { Text } from 'react-native'
import { SafeAreaView } from 'react-native-safe-area-context';

const LoadingScreen = () => {
  return (
    <SafeAreaView>
      <Text>LoadingScreen</Text>
    </SafeAreaView>
  )
}
export default LoadingScreen
```

- f. Creamos dentro de `app` el fichero `map/index.tsx` , creamos un nuevo componente y le cambiamos el nombre a `LoadingScreen` .

```
import { Text } from 'react-native'
import { SafeAreaView } from 'react-native-safe-area-context';

const MapScreen = () => {
  return (
    <SafeAreaView>
      <Text>MapScreen</Text>
    </SafeAreaView>
  )
}
```

```
}
export default MapScreen
```

- g. Creamos dentro de `app` el fichero `permissions/index.tsx` , creamos un nuevo componente y le cambiamos el nombre a `PermissionsScreen` .

```
import { Text } from 'react-native'
import { SafeAreaView } from 'react-native-safe-area-context';

const PermissionsScreen = () => {
  return (
    <SafeAreaView>
      <Text>PermissionsScreen</Text>
    </SafeAreaView>
  )
}
export default PermissionsScreen
```

- h. Accedemos al `Stack` de `layout.tsx` en `app` y ahora añadimos nuestras pantallas al `Stack` .

```
<Stack
  screenOptions={{
    headerShown: false,
  }}>
  <Stack.Screen name="loading/index" options={{animation: 'none'}} />
  <Stack.Screen name="map/index" options={{animation: 'fade'}} />
  <Stack.Screen name="permissions/index" options={{animation: 'fade'}} />
</Stack>
```

- i. Creamos dentro de `app` el fichero `index.tsx` , creamos un nuevo componente y le cambiamos el nombre a `MapsApp` .

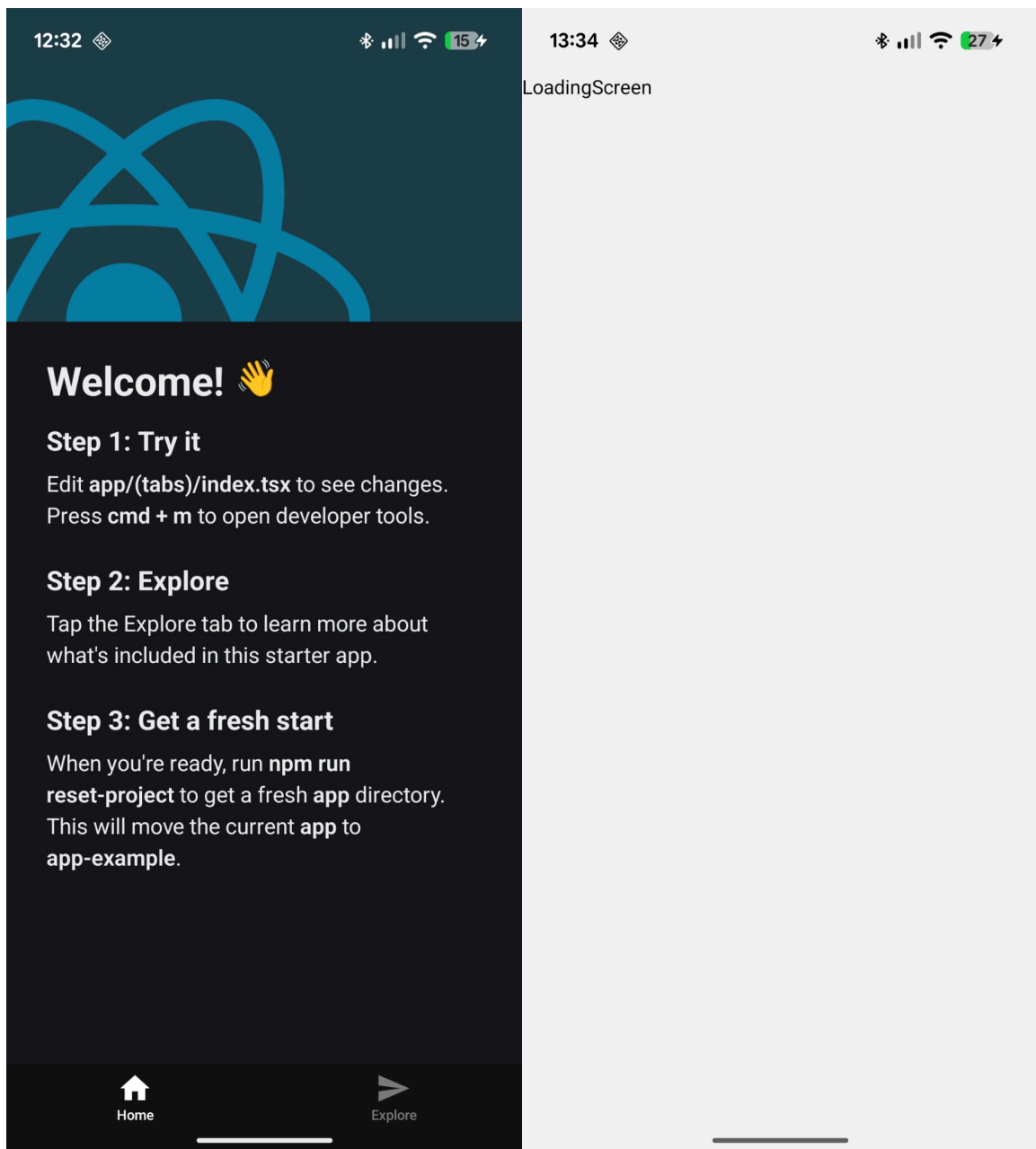
```
import { Text } from 'react-native'
import { SafeAreaView } from 'react-native-safe-area-context';

const MapsApp = () => {
  return (
    <SafeAreaView>
      <Text>MapsApp</Text>
    </SafeAreaView>
  )
}
export default MapsApp
```

- j. Lanzamos la aplicación y comprobamos que podemos empezar. A lo largo de la creación de la app iremos creando más carpetas para mantener una estructura del código adecuada.

Antes de eliminar estructuras

Después de eliminar estructuras



3. Acciones - Permisos de geolocalización

Vamos a proceder a la petición y activación de permisos por parte del usuario cuando inicia la aplicación, para ello seguimos los siguientes pasos:

1. Accedemos al enlace y seguimos el procedimiento oficial de instalación.

Location

A library that provides access to reading geolocation information, polling current location or subscribing location update events from the device.

 <https://docs.expo.dev/versions/latest/sdk/location/>



Location

A library that provides access to reading geolocation information, polling current location or subscribing location update events from the device.

Expo

2. Ejecutamos el comando en una terminal nueva o terminando el proceso del servidor Expo:

```
npx expo install expo-location
```

```
PS D:\RN\maps-app> npx expo install expo-location
> Installing 1 SDK 54.0.0 compatible native module using npm
> npm install

added 1 package, and audited 914 packages in 5s

165 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS D:\RN\maps-app>
```

3. Creamos en la raíz un fichero llamado `core/actions/permissions/location.ts`, este fichero tendrá las funciones que nos servirán de base para definir el comportamiento de la aplicación en base a los permisos de geolocalización.

```
import * as Location from 'expo-location'

export const requestLocationPermission = async() => {
  // Pedimos los permisos tras informar de por que los necesitamos
}

export const checkLocationPermission = async() => {
  // Comprobamos si tenemos permisos para poder acceder más tarde a los recursos del sistema
}
```

```
const manualPermissionRequest = async() => {
  // Lanzamos un acceso a los ajustes de la aplicación para cambiar m
  anualmente los permisos
}
```

4. Creamos en raíz el fichero `infrastructure/interfaces/location.ts` con el siguiente código:

```
export enum PermissionStatus{
  CHECKING = 'CHECKING',
  GRANTED = 'GRANTED',
  DENIED = 'DENIED',
  BLOCKED = 'BLOCKED',
  LIMITED = 'LIMITED',
  UNAVAILABLE = 'UNAVAILABLE',
  UNDETERMINATED = 'UNDETERMINATED',
}
```

5. Empezamos con la implementación del código del fichero

`core/actions/permissions/location.ts:`

```
import * as Location from 'expo-location'
import { PermissionStatus } from '@/infrastructure/interfaces/location'

export const requestLocationPermission = async(): Promise<PermissionStatus> => {
  // Pedimos los permisos tras informar de por que los necesitamos
  // Esta función solo puede ser llamada 1 VEZ
  // Si el usuario deniega los permisos, no volverá a mostrar el desplega
  ble
  const { status } = await Location.requestForegroundPermissionsAsync
  ();
  if (status !== 'granted'){
    //TODO: Crear la petición manual de cambio de permisos
    manualPermissionRequest();
    return PermissionStatus.DENIED;
  }
}
```

```

//Faltan los estados intermedios
return PermissionStatus.GRANTED;
}

export const checkLocationPermission = async() => {
  // Comprobamos si tenemos permisos para poder acceder más tarde a los recursos del sistema
  const { status } = await Location.getForegroundPermissionsAsync();

  switch(status){
    case 'granted':
      return PermissionStatus.GRANTED;
    case 'denied':
      return PermissionStatus.DENIED;
    default:
      return PermissionStatus.UNDETERMINATED;
  }
}

const manualPermissionRequest = async() => {
  // Lanzamos un acceso a los ajustes de la aplicación para cambiar manualmente los permisos
  //TODO: Hacer
}

```

4. Gestor de estados con Zustand

La gestión de estados se puede hacer de con varias librerías, en nuestro caso Zustand. Empezamos su uso preparando la estructura para poder usarlo. Para ello tenemos el siguiente enlace:

Zustand

 Bear necessities for state management in React

 <https://zustand-demo.pmnd.rs/>



1. Dentro de `presentation` creamos la carpeta `store` .

2. Instalamos la librería Zustand ejecutando el comando `npm install Zustand` en terminal.
3. Para comprobar la correcta instalación intentamos hacer la importación de `create` de la librería en el fichero `usePermissions.ts`.
4. Creamos la interfaz que usaremos para mantener la estructura del código:

```
interface PermissionState{
  locationStatus: PermissionStatus;
  requestLocationPermission: () ⇒ Promise<PermissionStatus>;
  checkLocationPermission: () ⇒ Promise<PermissionStatus>;
}
```

5. Creamos el objeto `usePermissionStore` que será el objeto creado mediante `Zustand` para gestionar el estado de nuestros permisos.

```
import { create } from 'zustand'
import { PermissionStatus } from '@/infrastructure/interfaces/location'
import { checkLocationPermission, requestLocationPermission } from '@/core/actions/permissions/location';

interface PermissionState{
  locationStatus: PermissionStatus;
  requestLocationPermission: () ⇒ Promise<PermissionStatus>;
  checkLocationPermission: () ⇒ Promise<PermissionStatus>;
}

export const usePermissionStore = create<PermissionState>()((set) ⇒ ({
  locationStatus: PermissionStatus.CHECKING,
  requestLocationPermission: async() ⇒ {
    const status = await requestLocationPermission();
    set({ locationStatus: status });
    return status;
  },
  checkLocationPermission: async() ⇒ {
    const status = await checkLocationPermission();
    set({ locationStatus: status });
  }
}));
```

```

    return status;
  }
}));

```

6. Recargamos la aplicación y de momento comprobamos que funciona.

4. Provider - Permission checker

Debido a los ciclos de vida de una aplicación, puede ocurrir que el usuario quite los permisos de geolocalización en cualquier momento por lo que debemos tener nuestra aplicación preparada para que en cualquier momento perdamos los permisos y la aplicación sepa responder instantáneamente sin errores.

1. Vamos a crear un fichero dentro de la carpeta `presentation` que se llame `providers/PermissionsCheckerProvider.tsx`. Creamos nuestro componente dentro del fichero.
2. Añadimos como entrada al componente un objeto del tipo `PropsWithChildren`:

```

const PermissionsCheckerProvider = ({ children }: PropsWithChildren)
⇒ {

```

3. Implementamos su comportamiento:

```

const PermissionsCheckerProvider = ({ children }: PropsWithChildren)
⇒ {
  const { locationStatus, checkLocationPermission } = usePermissionStore();

  useEffect(() ⇒ {
    checkLocationPermission();
  }, [])

  //TODO: Estar atento al cambio de estado de la aplicación

  return <>{children}</>
}

```

4. Volvemos al `Stack` principal en `_layout.tsx` y creamos el componente

`<PermissionCheckerProvider>` que envuelva el `<Stack>` :

```
return (  
  <ThemeProvider value={colorScheme === 'dark' ? DarkTheme : DefaultTheme}>  
    <PermissionsCheckerProvider>  
      <Stack  
        screenOptions={{  
          headerShown: false,  
        }}>  
        <Stack.Screen name="loading/index" options={{animation: 'none'}} />  
        <Stack.Screen name="map/index" options={{animation: 'fade'}} />  
        <Stack.Screen name="permissions/index" options={{animation: 'fade'}} />  
      </Stack>  
    </PermissionsCheckerProvider>  
  </ThemeProvider>  
)
```

5. En nuestro fichero `index.tsx` donde se mostrará la pantalla `LoadingScreen` ponemos el siguiente código:

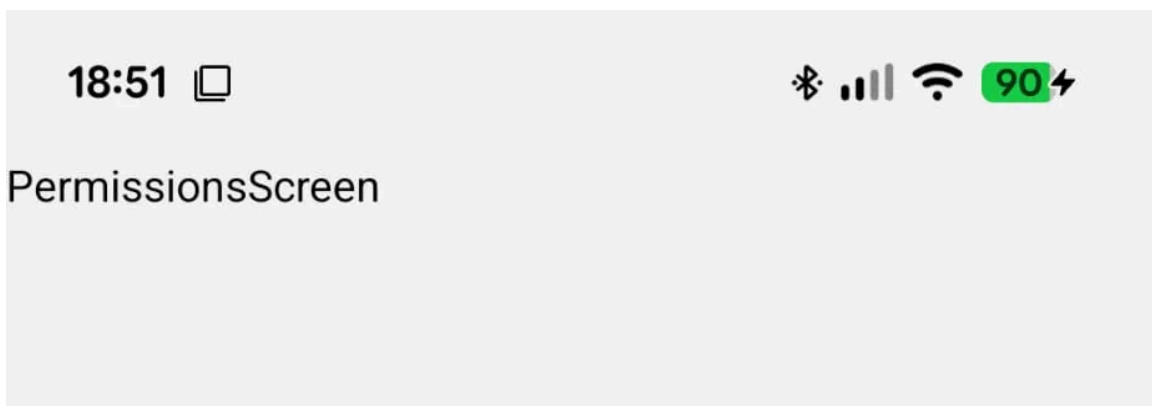
```
import { usePermissionStore } from '@presentation/store/usePermissions';  
import { PermissionStatus } from '@infrastructure/interfaces/location';  
import { Redirect } from 'expo-router';  
  
const LoadingScreen = () => {  
  const { locationStatus } = usePermissionStore();  
  
  if (locationStatus === PermissionStatus.CHECKING) {  
    return null;  
  }  
  
  if (locationStatus === PermissionStatus.GRANTED) {
```

```
return <Redirect href="/map" />;
}

return <Redirect href="/permissions" />;
}
export default LoadingScreen
```

En la última actualización del Framework de React Native no se acepta la navegación por la aplicación hasta que no se haya construido la estructura de navegación ya sea mediante un Stack o un Slot. Si intentamos que nuestro `PermissionsCheckerProvider` haga algún tipo de navegación durante la primera instancia de la aplicación antes de que se genera la estructura de navegación, recibiremos un error de React Native exigiendo ese cambio.

6. El resultado tras recargar la aplicación debería ser la pantalla de PermissionsScreen dado que no tenemos los permisos activos.



En estos momentos ya podemos ver que nuestra aplicación puede leer que no tenemos permisos al iniciar y reacciona de forma adecuada. Seguimos con el resto de casos.

5. Solicitar permisos

Vamos a concretar el código para pedir los permisos de forma explícita al usuario.

1. Vamos a reflejar el estado actual de los permisos en la pantalla de permisos a la que accedemos tras no tener los permisos. Para ello implementamos el siguiente código:

```

import { ThemedText } from '@presentation/components/shared/themed-text';
import { usePermissionStore } from '@presentation/store/usePermissions';
import { SafeAreaView } from 'react-native-safe-area-context';

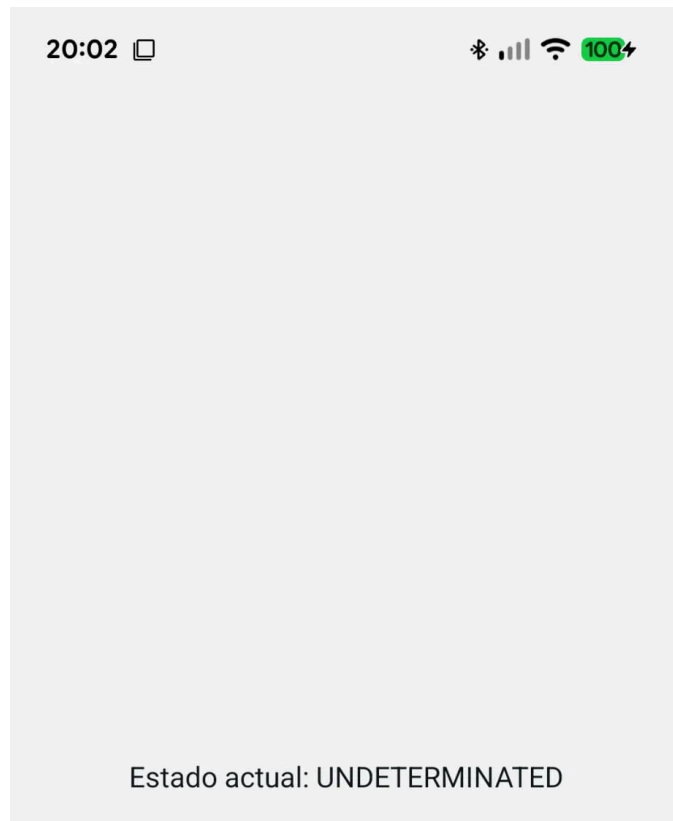
const PermissionsScreen = () => {
  const { locationStatus } = usePermissionStore();

  return (
    <SafeAreaView style={{
      flex: 1,
      justifyContent: 'center',
      alignItems: 'center',
    }}>
      <ThemedText>Estado actual: {locationStatus}</ThemedText>
    </SafeAreaView>
  );
};

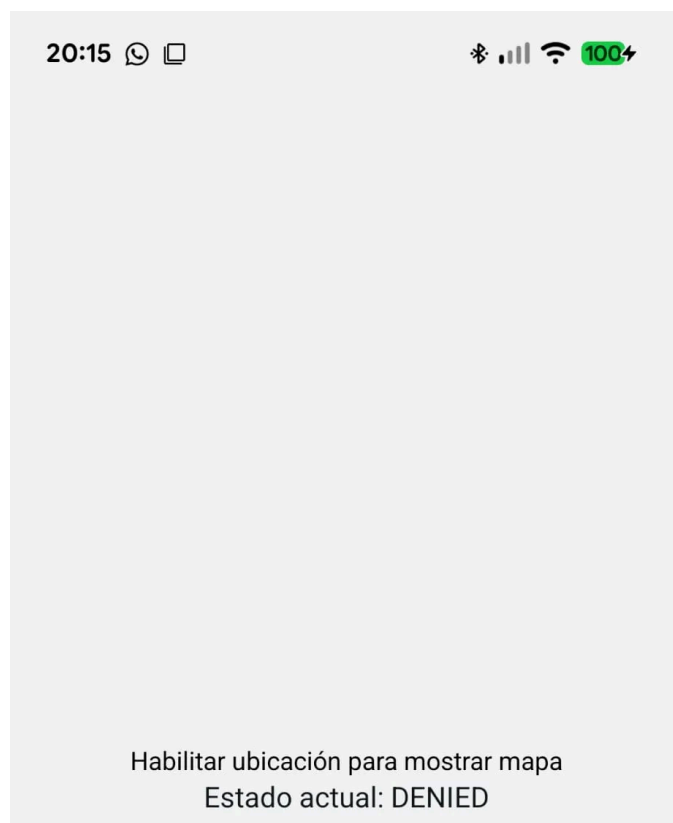
export default PermissionsScreen;

```

Tras implementarlo deberíamos tener el estado `UNDETERMINATED` que devolvemos por defecto en nuestro `checkLocationPermission` en la definición de la función en `core/actions/permissions/location.ts` . Todo lo que no sea `GRANTED` o `DENIED` será `UNDETERMINATED` por el momento. Si se recarga la aplicación dará `DENIED` .



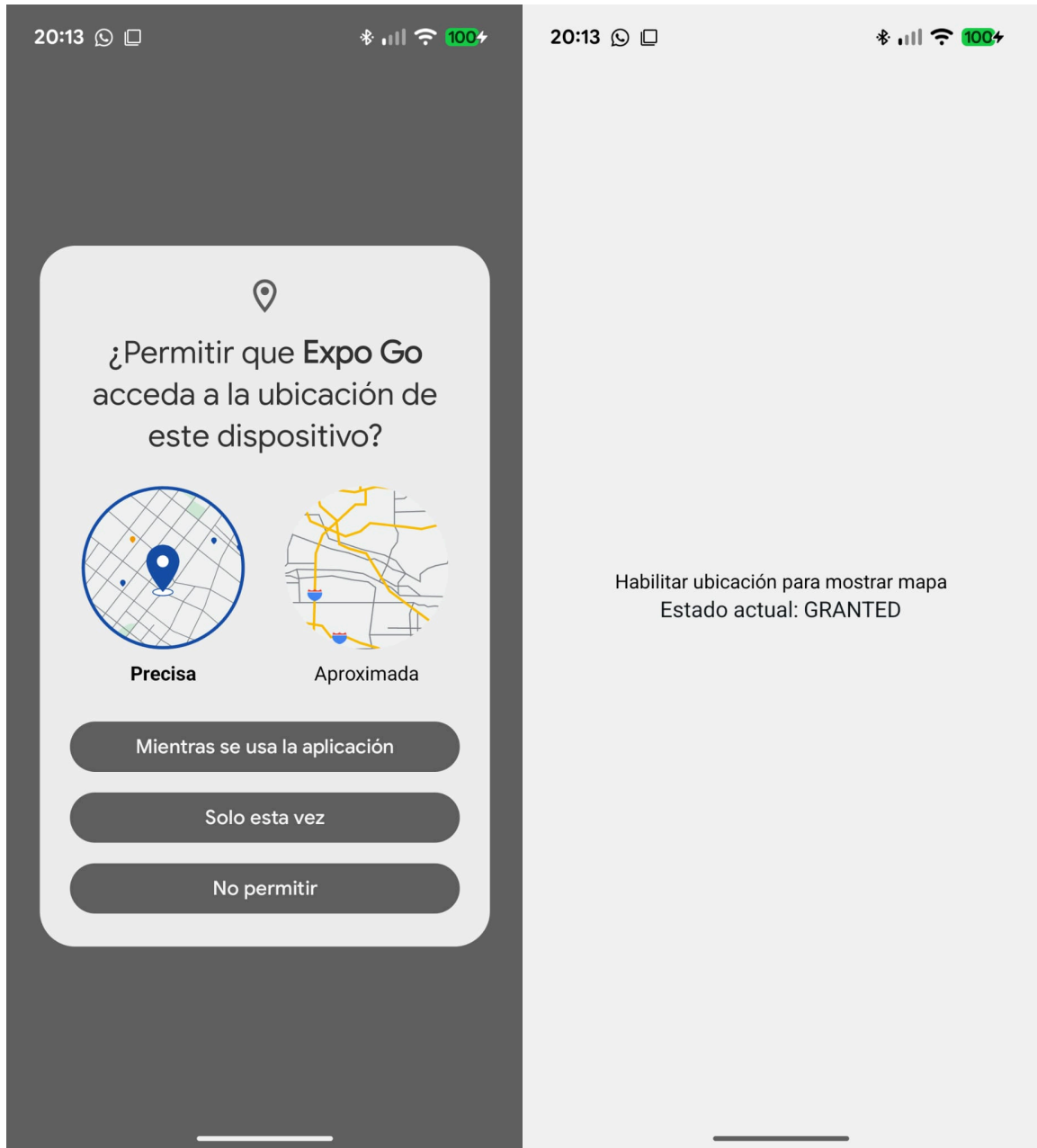
2. Añadimos un botón para realizar al acción de pedir permisos aunque sea muy básico.



3. Tras pulsar el botón se hará la petición en segundo plano de forma **SINCRONA** para pedir los permisos tal y como se ve a continuación:

Tras pulsar el botón

Tras aceptar permisos **UNA VEZ**



4. Si accedemos al apartado de ajustes y cambiamos los permisos asociados a la aplicación de **EXPO GO** veremos que al volver al contexto de nuestra aplicación los permisos cambian pero nuestra app no reacciona adecuadamente al cambio ya que no hay cambio de pantalla. **EXPO GO** recarga el contexto de la aplicación y se da cuenta del cambio de

permisos y recarga la aplicación pero dicha situación no es igual para iOS o no se va a dar siempre en Android.

5. Vamos a mejorar la visualización del botón. Añadimos en `shared` un nuevo fichero llamado `ThemedPressable.tsx` y le vamos a dar unos estilos:

```
import { Text, Pressable, StyleSheet, PressableProps } from 'react-native'

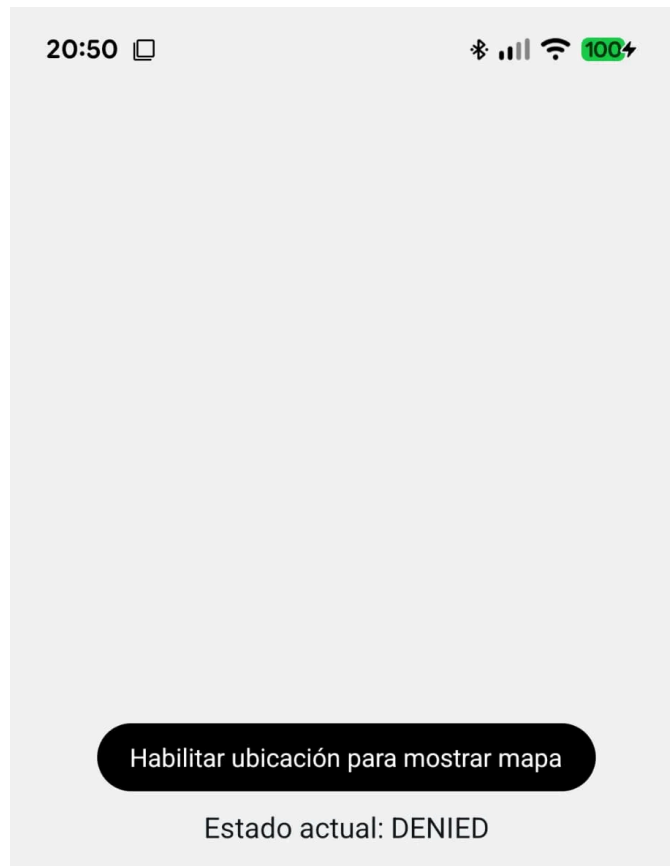
interface Props extends PressableProps {
  children: string;
}

const ThemedPressable = ({children, ...rest}: Props) => {
  return (
    <Pressable style={styles.btnPrimary}{...rest}>
      <Text style={{color: 'white'}}>{children}</Text>
    </Pressable>
  )
}

export default ThemedPressable

const styles = StyleSheet.create({
  btnPrimary: {
    backgroundColor: 'black',
    paddingVertical: 10,
    paddingHorizontal: 20,
    borderRadius: 100,
    margin: 10,
  }
})
```

6. El resultado final será:



6. Permisos fuera de la aplicación

Vamos a intentar controlar los permisos cuando la aplicación no está en el contexto principal, es decir, cuando el usuario sale de la aplicación y retira los permisos manualmente.

1. Vamos a implementar el **TODO** que teníamos pendiente en el

`PermissionsCheckerProvider.tsx` :

```
//Creación de una referencia del estado de la app
const appState = useRef(AppState.currentState);

useEffect(() => {
  checkLocationPermission();
}, [])

useEffect(() => {
  const subscription = AppState.addListener('change', (nextAppState) => {
    appState.current = nextAppState;
  });
}, [])
```

```

    console.log({nextAppState});
    if(nextAppState === 'active'){
      checkLocationPermission();
    }
  });

  return () => {
    subscription.remove();
  }
}, [])

```

2. Una vez implementado veremos como hay cambios de estado cuando la aplicación cambia de estado en su ciclo de vida.

```

> Reloading apps
No apps connected. Sending "reload" to all React Native apps failed. Make sure
your app is running in the simulator or on a phone connected via USB.
> Opening on Android...
> Opening exp://192.168.18.6:8081 on Pixel_7
> Press ? | show all commands
Android Bundled 42ms node_modules\expo-router\entry.js (1 module)
WARN [Reanimated] Reduced motion setting is enabled on this device. This warn
ing is visible only in the development mode. Some animations will be disabled b
y default. You can override the behavior for individual animations, see https:/
/docs.swmansion.com/react-native-reanimated/docs/guides/troubleshooting#reduced
-motion-setting-is-enabled-on-this-device.
LOG {"nextAppState": "background"}
LOG {"nextAppState": "active"}

```

Sin la referencia no podemos acceder al estado antiguo y al volver a crear el contexto de la aplicación tras volver del `background` React no es capaz de alcanzar el estado cambiado de forma adecuada.

3. Tras estos cambios nuestra app ya reacciona al cambio de contexto gracias a la recarga de `Android y Expo Go`. Continuamos con la petición manual de cambio en `Ajustes`

7. Abrir ajustes mediante nuestra aplicación

Ya tenemos una aplicación capaz de reaccionar a algunas situaciones, pero ante la negación del usuario a darnos permisos inicialmente nuestra aplicación no puede reaccionar más y hay muchos usuarios que no saben cambiar los permisos manualmente a través de ajustes lo que se traduce como una pérdida de clientes a nuestra aplicación. Vamos a solucionar este problema.

1. Accedemos al fichero `core/actions/permissions/location.ts` y completamos la función `manualPermissionRequest`:

```
const manualPermissionRequest = async() => {  
  // Lanzamos un acceso a los ajustes de la aplicación para cambiar m  
  anualmente los permisos  
  Alert.alert('Permiso de ubicación necesario',  
    'Para continuar debe habilitar el permiso de localización en los ajuste  
s de la app',  
    [  
      {  
        text: 'Abrir ajuste',  
        onPress: () => {  
          Linking.openSettings();  
        }  
      },  
      {  
        text: 'Cancelar',  
        style: 'destructive'  
      }  
    ]  
  )  
}
```

2. Ajustamos la función de `requestLocationPermission` :

```
export const requestLocationPermission = async(): Promise<Permissio  
nStatus> => {  
  // Pedimos los permisos tras informar de por que los necesitamos  
  // Esta función solo puede ser llamada 1 VEZ  
  // Si el usuario deniega los permisos, no volverá a mostrar el desplega  
ble  
  const { status } = await Location.requestForegroundPermissionsAsync
```

```
();  
if (status !== 'granted'){  
    //TODO: Crear la petición manual de cambio de permisos  
    if (status === 'denied'){  
        manualPermissionRequest();  
    }  
    return PermissionStatus.DENIED;  
}  
  
//Faltan los estados intermedios  
return PermissionStatus.GRANTED;  
}
```

3. En este momento nuestra app es capaz de responder a las necesidades del cambio de permisos incluso tras denegarlos.