GUÍA PASO A PASO

Expo UI: una pantalla de ajustes con SwiftUI y Jetpack Compose

Cómo instalar @expo/ui, qué hace el Host y una pantalla de ajustes completa en React Native que se dibuja con SwiftUI en iOS y con Jetpack Compose en Android. Con la checklist para probarla en un dispositivo real.

PARA	Apps React Native con Expo que quieren controles nativos sin escribir Swift ni Kotlin
PROBADO CON	Sin ejecutar en dispositivo: contrastado con el anuncio del SDK 56 y la documentación del SDK 58
ACTUALIZADA	12 de noviembre de 2026
DOCUMENTACIÓN	docs.expo.dev/versions/latest/sdk/ui
ARTÍCULO	ecomlabs.dev/g/expoui
AUTOR	Luis Morales · @luismoralesdev

Índice

Antes de empezar	Qué vas a montar
Paso 01	Instalar
Paso 02	Los imports y el Host
Paso 03	La pantalla de ajustes
Paso 04	Cómo leerla
Paso 05	Comprobar en un dispositivo
Al terminar	Checklist

ANTES DE EMPEZAR

Qué vas a montar

Una pantalla de ajustes con `@expo/ui`: dos interruptores, un deslizador y un botón. El mismo código se dibuja con SwiftUI en iOS y con Jetpack Compose en Android. Expo UI es estable desde el SDK 56 (anuncio del 26 de mayo de 2026) y la documentación actual está en el SDK 58.

> **CUIDADO**
> No ejecuté este código en un iPhone ni en un Android, y no instalé `@expo/ui` para compilarlo. Es el ejemplo del anuncio de Expo, contrastado con la documentación. La comprobación en dispositivo es el paso 05.

PASO 01

Instalar

```
# proyecto nuevo del SDK 56: Expo UI viene en la plantilla por defecto
npx create-expo-app@latest --template default@sdk-56

# proyecto existente
npx expo install @expo/ui
```

En una app React Native sin Expo, instala primero el paquete `expo`. Expo UI está incluido en Expo Go, según el anuncio y la documentación.

PASO 02

Los imports y el Host

IMPORT	PARA QUÉ
`@expo/ui`	Componentes universales: iOS, Android y web (experimental)
`@expo/ui/swift-ui`	API de SwiftUI, solo Apple
`@expo/ui/jetpack-compose`	API de Jetpack Compose, Android
`@expo/ui/community/...`	Reemplazos casi compatibles de paquetes de la comunidad

- Cada componente va dentro de un `Host`: el puente entre React Native y lo nativo.
- El flexbox se aplica al `Host`. Dentro, el layout lo hacen `Row`, `Column` y `Spacer`.
- Para anidar componentes de React Native dentro de un árbol nativo y volver a Expo UI, usa otro `Host`.

PASO 03

La pantalla de ajustes

Guárdala como `SettingsScreen.tsx`. Es la del anuncio, con un solo cambio: `ReactNode` se importa como tipo.

```
import { useState, type ReactNode } from 'react';
import { Button, FieldGroup, Host, Row, Slider, Spacer, Switch, Text } from '@expo/ui';

export default function SettingsScreen() {
  const [notifications, setNotifications] = useState(true);
  const [sounds, setSounds] = useState(false);
  const [brightness, setBrightness] = useState(0.6);

  return (
    <Host style={{ flex: 1 }}>
      <FieldGroup>
        <FieldGroup.Section title="Notifications">
          <LabeledRow label="Push notifications">
            <Switch value={notifications} onValueChange={setNotifications} />
          </LabeledRow>
          <LabeledRow label="Sounds">
            <Switch value={sounds} onValueChange={setSounds} />
          </LabeledRow>
          <FieldGroup.SectionFooter>
            <Text textStyle={{ fontSize: 13, color: '#6c6c70' }}>
              Notification previews can expose sensitive content on the lock screen.
            </Text>
          </FieldGroup.SectionFooter>
        </FieldGroup.Section>

        <FieldGroup.Section title="Display">
          <LabeledRow label="Brightness">
            <Slider value={brightness} onValueChange={setBrightness} />
          </LabeledRow>
        </FieldGroup.Section>

        <FieldGroup.Section>
          <Row alignment="center" style={{ padding: 12 }}>
            <Spacer flexible />
            <Button variant="outlined" onPress={() => alert('Signed out')} label="Sign out" />
            <Spacer flexible />
          </Row>
        </FieldGroup.Section>
      </FieldGroup>
    </Host>
  );
}

function LabeledRow({ label, children }: { label: string; children: ReactNode }) {
  return (
```

```
    <Row alignment="center" spacing={16}>
      <Text>{label}</Text>
      <Spacer flexible />
      {children}
    </Row>
  );
}
```

PASO 04

Cómo leerla

- `FieldGroup` y `.Section` forman la lista de ajustes agrupados; `Row` y `Spacer` ponen el texto a la izquierda y el control a la derecha.
- `Switch` y `Slider` siguen el patrón `value` más `onValueChange`; `Button` usa `variant`, `label` y `onPress`.

En iOS, el anuncio describe un `Form` de SwiftUI con secciones agrupadas y controles del sistema; en Android, una lista agrupada de Material 3. No hace falta un archivo por plataforma.

PASO 05

Comprobar en un dispositivo

- Abre el proyecto en Expo Go en un iPhone y en un Android reales.
- Mueve cada control y confirma que el estado de React cambia.
- Revisa el modo oscuro: los ejemplos de la documentación actual fijan el color de `Text` con `useColorScheme()`.
- Anota qué se ve distinto entre iOS y Android; es esperable, cada sistema dibuja el suyo.

> **CUIDADO**
>
> Límites que reconoce Expo: la capa universal es una primera iteración y algunos componentes no existen en las tres plataformas; la web es experimental; `glassEffect` pide Xcode 26 e iOS 26.

AL TERMINAR

Checklist

- [ ] Instalé el paquete con `npx expo install @expo/ui`.
- [ ] Toda pantalla con Expo UI empieza en un `Host`.
- [ ] El layout dentro del `Host` usa `Row`, `Column` y `Spacer`.
- [ ] El código multiplataforma importa de `@expo/ui`, no de `swift-ui`.
- [ ] Probé en un iPhone y en un Android reales, en claro y en oscuro.
- [ ] Revisé las props de cada reemplazo de comunidad frente al original.