EcomLabs

STEP-BY-STEP GUIDE

Expo UI: a settings screen with SwiftUI and Jetpack Compose

How to install @expo/ui, what Host does, and a complete settings screen in React Native that renders with SwiftUI on iOS and Jetpack Compose on Android. With a checklist for testing it on a real device.

FOR	React Native apps built with Expo that want native controls without writing Swift or Kotlin
TESTED WITH	Not run on a device: cross-checked against the SDK 56 announcement and the SDK 58 docs
UPDATED	November 12, 2026
DOCS	docs.expo.dev/versions/latest/sdk/ui
ARTICLE	ecomlabs.dev/g/expoui-en
AUTHOR	Luis Morales · @luismoralesdev

Contents

Before you start	What you will build
Step 01	Install
Step 02	The imports and Host
Step 03	The settings screen
Step 04	How to read it
Step 05	Check on a device
When you finish	Checklist

BEFORE YOU START

What you will build

A settings screen with `@expo/ui`: two switches, a slider and a button. The same code renders with SwiftUI on iOS and Jetpack Compose on Android. Expo UI has been stable since SDK 56 (announced May 26, 2026) and the current docs are on SDK 58.

> **CAUTION**
> I did not run this code on an iPhone or an Android device, and I did not install `@expo/ui` to compile it. It is the example from the Expo announcement, cross-checked against the docs. The device check is step 05.

STEP 01

Install

```
# new SDK 56 project: Expo UI ships in the default template
npx create-expo-app@latest --template default@sdk-56

# existing project
npx expo install @expo/ui
```

In a React Native app without Expo, install the `expo` package first. Expo UI is included in Expo Go, per the announcement and the docs.

STEP 02

The imports and Host

IMPORT	WHAT FOR
`@expo/ui`	Universal components: iOS, Android and web (experimental)
`@expo/ui/swift-ui`	The SwiftUI API, Apple only
`@expo/ui/jetpack-compose`	The Jetpack Compose API, Android
`@expo/ui/community/...`	Near-compatible replacements for community packages

- Every component goes inside a `Host`: the bridge between React Native and native views.
- Flexbox applies to the `Host`. Inside it, `Row`, `Column` and `Spacer` do the layout.
- To nest React Native components inside a native tree and return to Expo UI, use another `Host`.

STEP 03

The settings screen

Save it as `SettingsScreen.tsx`. It is the announcement's screen with one change: `ReactNode` is imported as a type.

```
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>
  );
}
```

STEP 04

How to read it

- `FieldGroup` and `.Section` form the grouped settings list; `Row` and `Spacer` put the text on the left and the control on the right.
- `Switch` and `Slider` follow the `value` plus `onValueChange` pattern; `Button` takes `variant`, `label` and `onPress`.

On iOS, the announcement describes a SwiftUI `Form` with grouped sections and system controls; on Android, a Material 3 grouped list. You do not need a file per platform.

STEP 05

Check on a device

- Open the project in Expo Go on a real iPhone and a real Android device.
- Move every control and confirm the React state changes.
- Check dark mode: the current docs examples set the color of `Text` from `useColorScheme()`.
- Note what looks different between iOS and Android; that is expected, each system draws its own.

> **CAUTION**
> Limits Expo acknowledges: the universal layer is a first iteration and some components do not exist on all three platforms; web is experimental; `glassEffect` needs Xcode 26 and iOS 26.

WHEN YOU FINISH

Checklist

- [ ] I installed the package with `npx expo install @expo/ui`.
- [ ] Every screen with Expo UI starts in a `Host`.
- [ ] Layout inside the `Host` uses `Row`, `Column` and `Spacer`.
- [ ] Cross-platform code imports from `@expo/ui`, not from `swift-ui`.
- [ ] I tested on a real iPhone and a real Android device, in light and dark mode.
- [ ] I compared the props of each community replacement with the original.