Flutter Tips and Tricks

Knowledge Download

Simon Lightfoot

Flutter Community Lead
CTO of DevAngels London

@devangelslondon

Flutter Community

DEVANGELS .LONDON

Topics

- App Startup
- Build, Building, Builders
- Widget Layout
- Debugging
- Accessibility

App Startup

App Startup

Typical startup flow (Android)

1

Native App Launch

OS starts the MainActivity which initializes the Dart Runtime Environment.

2

main() execution

Some asynchronous work is done to prepare to app before Widgets are drawn.

3

runApp()

Flutter bindings are initialized and MaterialApp is installed as base widget.

4

FlutterView renders

First frame of graphics is scheduled and rendered and displayed in Activity.

5

User Interaction

The user can now interact and start using your application.

Native splash screen displayed

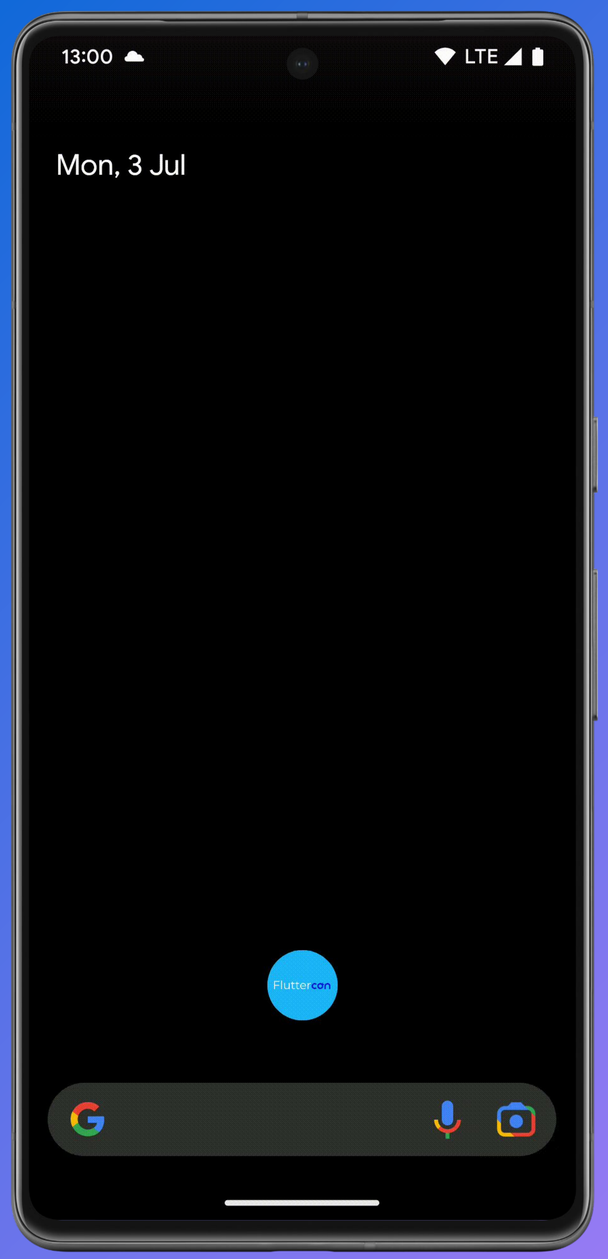
Flutter based splash screen displayed

No blank splash screens

1

Pre Android 12

```
android/app/src/main/res/values/styles.xml
<resources>
    <style name="LaunchTheme" parent="@android:style/Theme.Light.NoTitleBar">
        <item name="android:windowBackground">@drawable/launch_background</item>
    </style>
    <style name="NormalTheme" parent="@android:style/Theme.Light.NoTitleBar">
        <item name="android:windowBackground">@android:color/white</item>
    </style>
</resources>
```



https://docs.flutter.dev/platform-integration/android/splash-screen

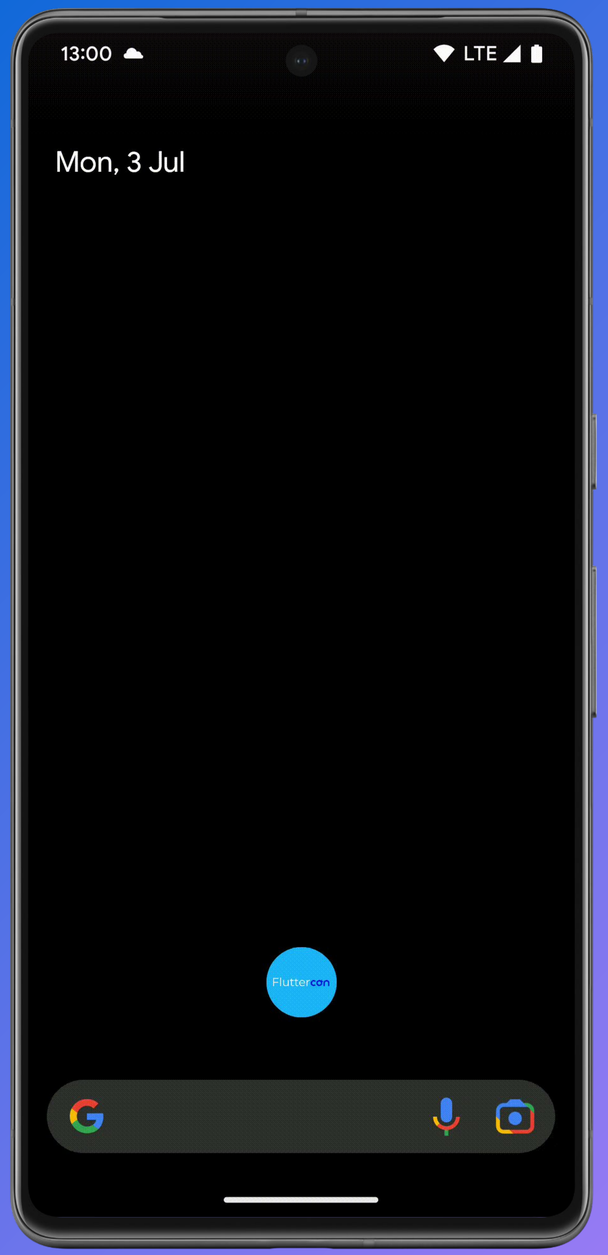
No blank splash screens

1

Pre Android 12

```
android/app/src/main/res/values/styles.xml
<resources>
    <style name="LaunchTheme" parent="@android:style/Theme.Light.NoTitleBar">
        <item name="android:windowBackground">@drawable/launch_background</item>
    </style>
    <style name="NormalTheme" parent="@android:style/Theme.Light.NoTitleBar">
        <item name="android:windowBackground">@android:color/white</item>
    </style>
</resources>

android/app/src/main/res/drawable/launch_background.xml
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@color/ic_launcher_background" />
    <item>
        <bitmap android:src="@drawable/logo" android:gravity="center" />
    </item>
</layer-list>
```

https://docs.flutter.dev/platform-integration/android/splash-screen



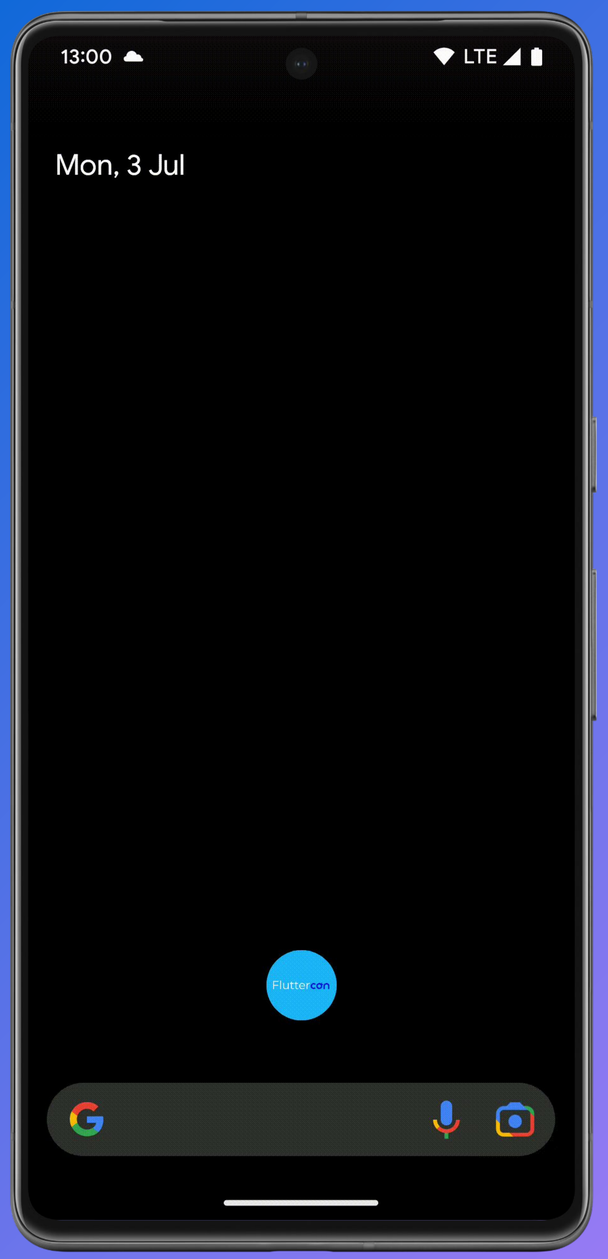
No blank splash screens

1

Android 12+ (API 31+)

```
android/app/src/main/res/values/styles.xml
<resources>
  <style name="LaunchTheme" parent="NormalTheme">
    <item name="android:windowBackground">@drawable/launch_background</item>
    <item name="android:windowSplashScreenBackground">@color/ic_launcher_background</item>
    <item name="android:windowSplashScreenAnimatedIcon">@drawable/logo</item>
    <item name="android:windowSplashScreenAnimationDuration">0</item>
  </style>
  <style name="NormalTheme" parent="@android:style/Theme.Light.NoTitleBar">
    <item name="android:windowBackground">@android:color/white</item>
    <item name="android:statusBarColor">@android:color/transparent</item>
    <item name="android:navigationBarColor">@android:color/transparent</item>
    <item name="android:windowDrawsSystemBarBackgrounds">true</item>
    <item name="android:enforceNavigationBarContrast">false</item>
  </style>
</resources>
```

https://docs.flutter.dev/platform-integration/android/splash-screen



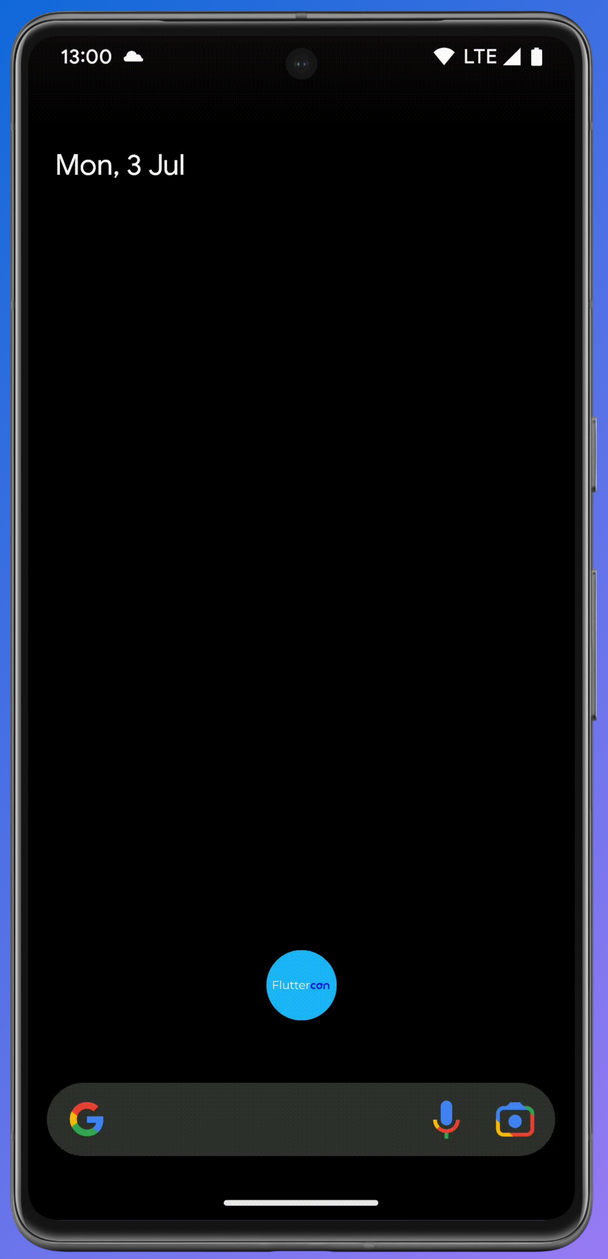
No blank splash screens

Android 12+ (API 31+)

1

```
android/app/src/main/res/values/styles.xml
<resources>
  <style name="LaunchTheme" parent="NormalTheme">
    <item name="android:windowBackground">@drawable/launch_background</item>
    <item name="android:windowSplashScreenBackground">@color/ic_launcher_background</item>
    <item name="android:windowSplashScreenAnimatedIcon">@drawable/logo</item>
    <item name="android:windowSplashScreenAnimationDuration">0</item>
  </style>
  <style name="NormalTheme" parent="@android:style/Theme.Light.NoTitleBar">
    <item name="android:windowBackground">@android:color/white</item>
    <item name="android:statusBarColor">@android:color/transparent</item>
    <item name="android:navigationBarColor">@android:color/transparent</item>
    <item name="android:windowDrawsSystemBarBackgrounds">true</item>
    <item name="android:enforceNavigationBarContrast">false</item>
  </style>
</resources>
```

https://docs.flutter.dev/platform-integration/android/splash-screen



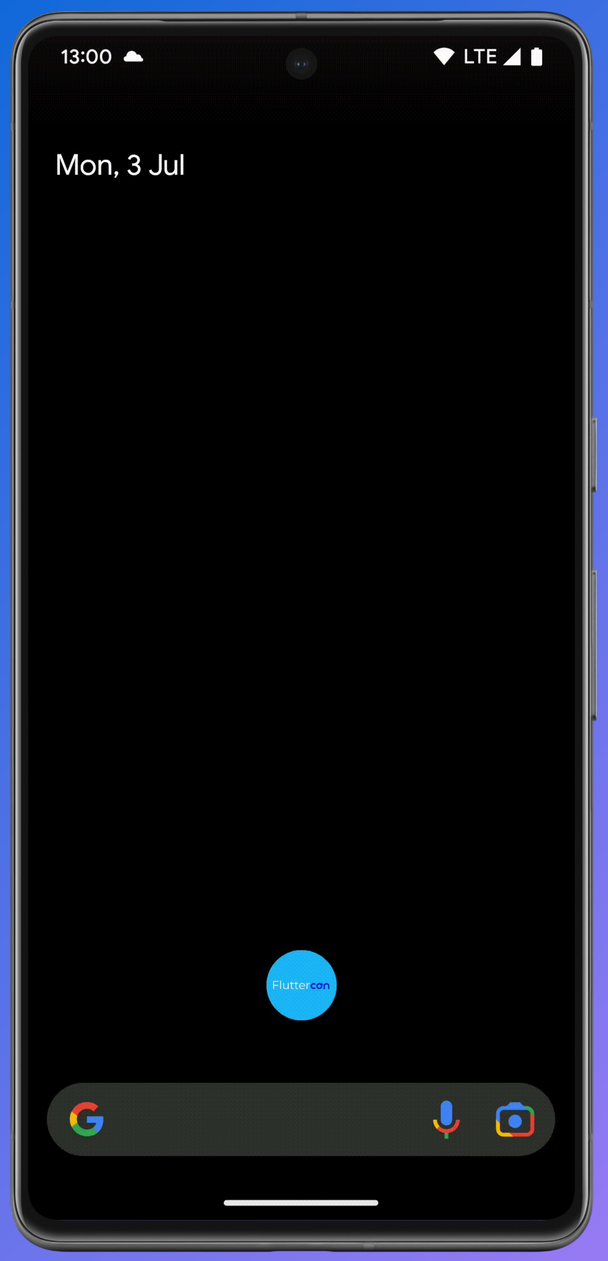
No blank splash screens

Android 12+ (API 31+)

1

```
android/app/src/main/res/values/styles.xml
<resources>
  <style name="LaunchTheme" parent="NormalTheme">
    <item name="android:windowBackground">@drawable/launch_background</item>
    <item name="android:windowSplashScreenBackground">@color/ic_launcher_background</item>
    <item name="android:windowSplashScreenAnimatedIcon">@drawable/logo</item>
    <item name="android:windowSplashScreenAnimationDuration">0</item>
  </style>
  <style name="NormalTheme" parent="@android:style/Theme.Light.NoTitleBar">
    <item name="android:windowBackground">@android:color/white</item>
    <item name="android:statusBarColor">@android:color/transparent</item>
    <item name="android:navigationBarColor">@android:color/transparent</item>
    <item name="android:windowDrawsSystemBarBackgrounds">true</item>
    <item name="android:enforceNavigationBarContrast">false</item>
  </style>
</resources>
```

https://docs.flutter.dev/platform-integration/android/splash-screen



App Initialization

Entry point execution main()

2

```
Future<void> main() async {
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  // Pass all uncaught "fatal" errors from the framework to Crashlytics
  FlutterError.onError = FirebaseCrashlytics.instance.recordFlutterFatalError;
  // Pass all uncaught asynchronous errors that aren't handled to Crashlytics
  PlatformDispatcher.instance.onError = (Object error, StackTrace stackTrace) {
    FirebaseCrashlytics.instance.recordError(error, stackTrace, fatal: true);
    return true;
  };
  final backend = await Backend.init();
  runApp(FlutterTipsApp(backend: backend));
}
```

App Initialization

Entry point execution main()

2

```
main_dev.dart
void main() {
  runApp(
    FlutterTipsApp(
      config: AppConfig(
        env: AppEnv.dev,
        firebaseOptions: FbDev.currentPlatform,
      ),
    ),
  );
}
```

```
main_prod.dart
void main() {
  runApp(
    FlutterTipsApp(
      config: AppConfig(
        env: AppEnv.prod,
        firebaseOptions: FbProd.currentPlatform,
      ),
    ),
  );
}
```

App Widget

FlutterTipsApp


```
class _FlutterTipsAppState extends State<FlutterTipsApp> {
  Future<void>? _appLoader;
  Backend? _backend;

  @override
  void didChangeDependencies() {
    super.didChangeDependencies();
    _appLoader ??= _loadApp(context);
  }

  Future<void> _loadApp(BuildContext context) async {
    await Firebase.initializeApp(
      options: widget.config.firebaseOptions,
    );
    if (widget.config.env == AppEnv.prod) {
      FlutterError.onError = ...;
      PlatformDispatcher.instance.onError = ...;
    }
    _backend = await Backend.init();
  }

  @override
  void dispose() {
    _backend?.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return ...
  }
}
```

App Widget

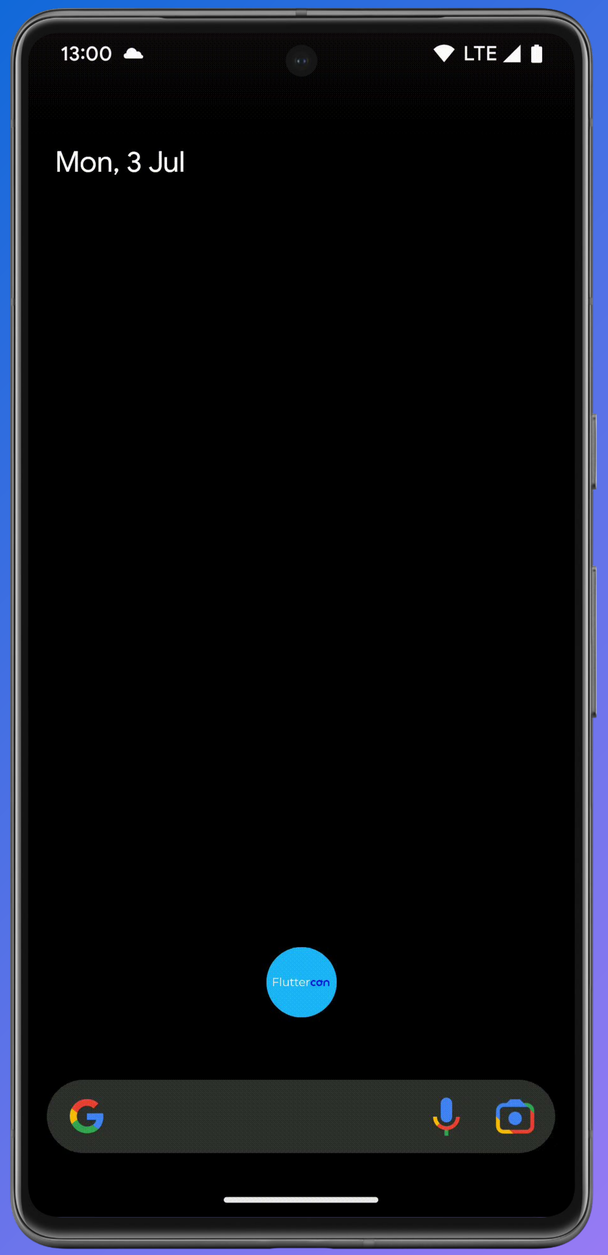
FlutterTipsApp

Flicker Free Splash

deferFirstFrame / allowFirstFrame

4

```
Future<void>? _splashLoader;

@override
void initState() {
  super.initState();
  RendererBinding.instance.deferFirstFrame();
}

@override
void didChangeDependencies() {
  super.didChangeDependencies();
  _splashLoader ??= _loadSplash(context).whenComplete(
    () ⇒ RendererBinding.instance.allowFirstFrame(),
  );
}

Future<void> _loadSplash(BuildContext context) async {
  await SplashScreen.precacheAssets(context);
}
```

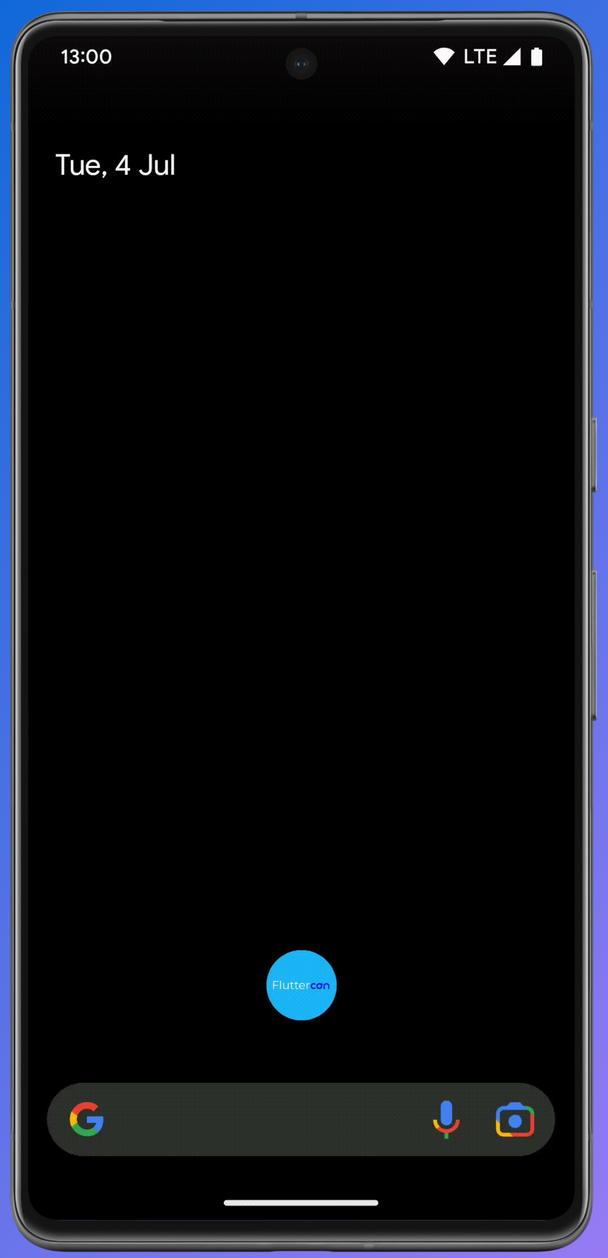


Flicker Free Splash

deferFirstFrame / allowFirstFrame

4

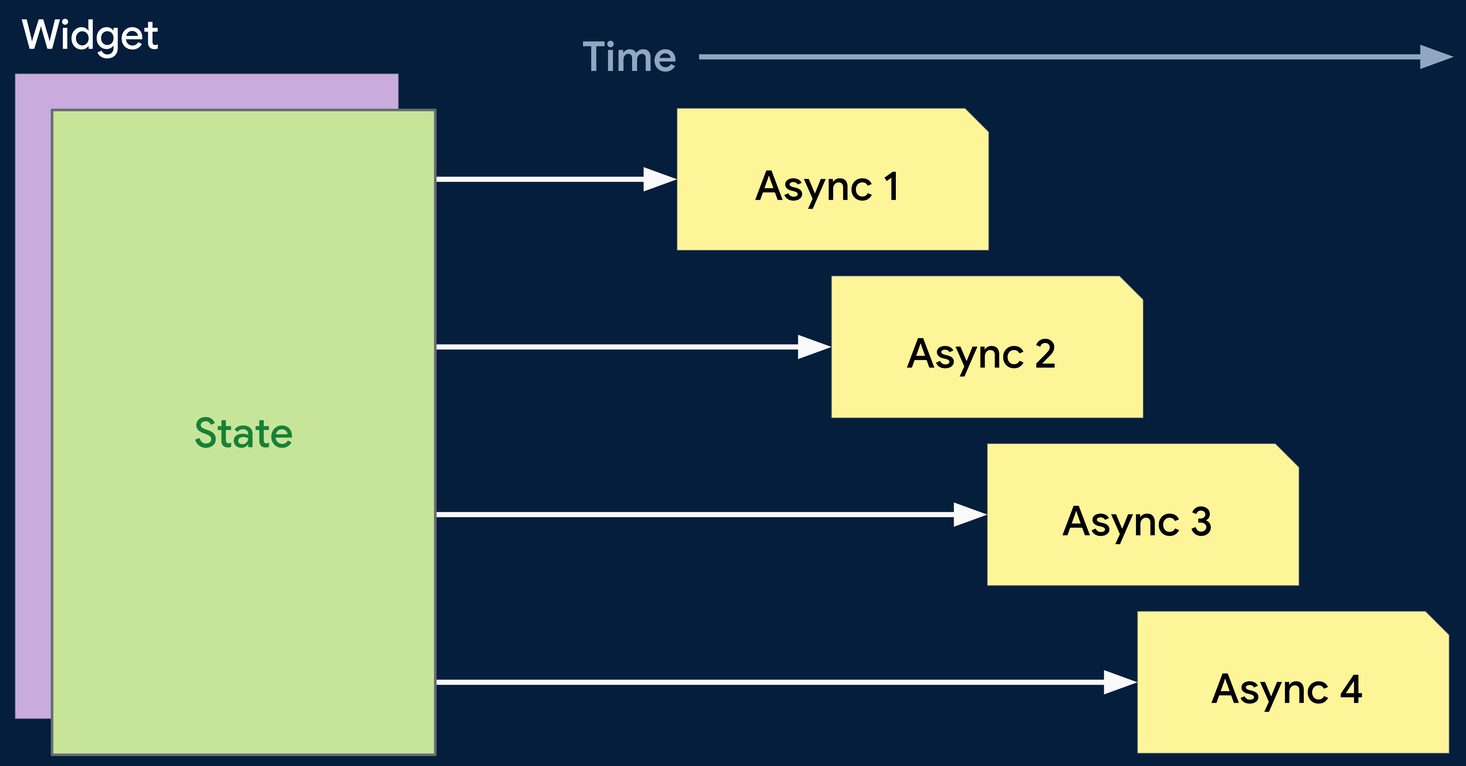
```
Future<void>? _splashLoader;

@override
void initState() {
  super.initState();
  RendererBinding.instance.deferFirstFrame();
}

@override
void didChangeDependencies() {
  super.didChangeDependencies();
  _splashLoader ??= _loadSplash(context).whenComplete(
    () ⇒ RendererBinding.instance.allowFirstFrame(),
  );
}

Future<void> _loadSplash(BuildContext context) async {
  await SplashScreen.precacheAssets(context);
}
```



Build, Building, Builders

Build Side-Effects

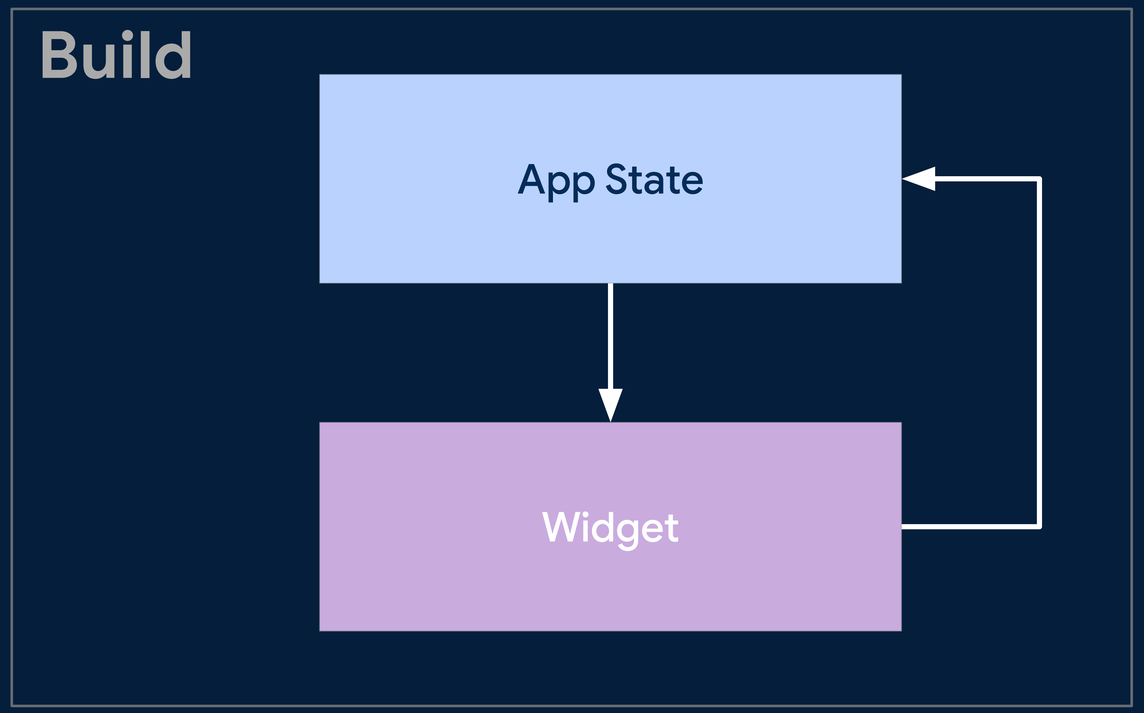
Don’t call async methods during `build()` execution



You don’t control the timing of when and how many times **`build`** is called!

Build Side-Effects

Don’t change your app state during `build()` execution

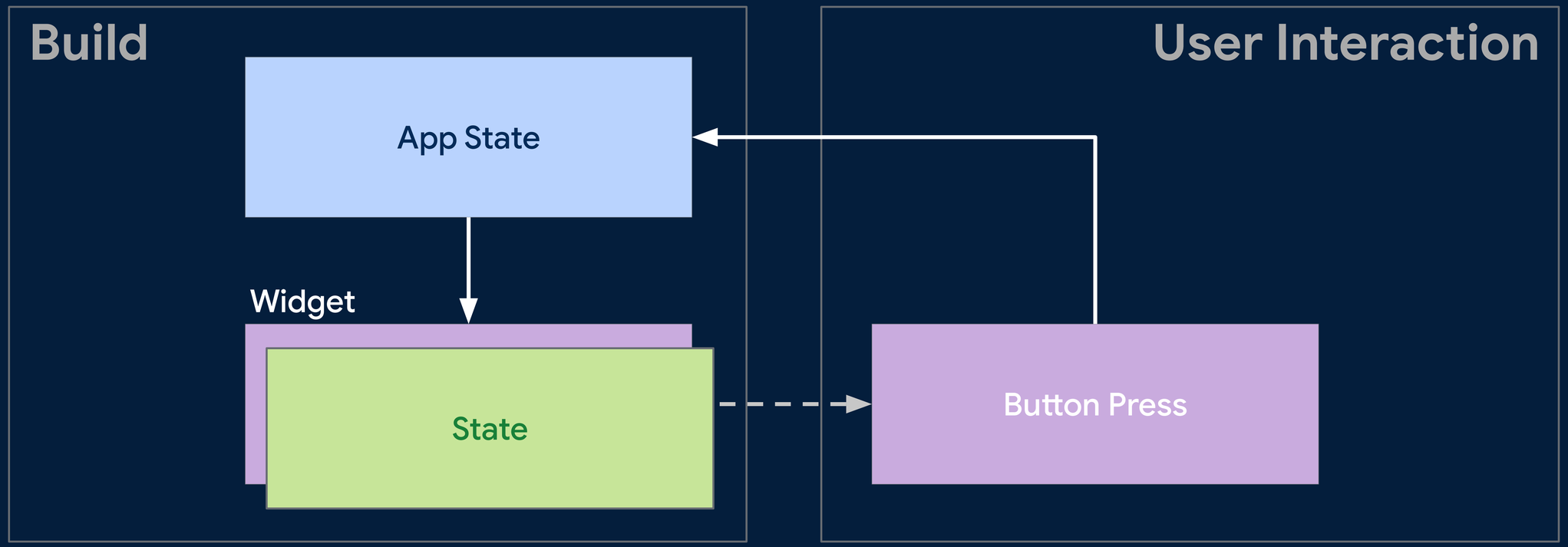


setState() or markNeedsBuild() called during build.

~~`scheduleMicrotask()`~~

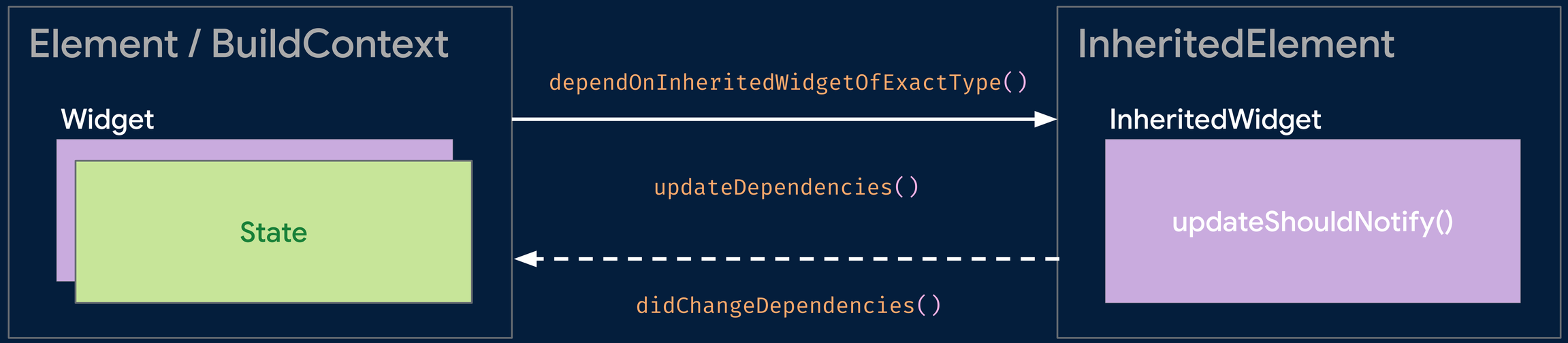
Build Side-Effects

Don’t change your app state during `build()` execution



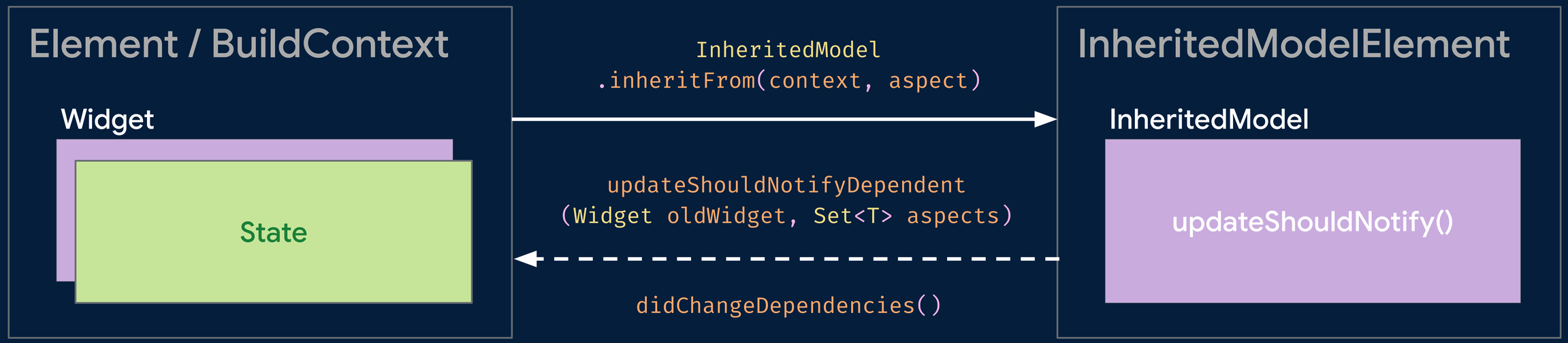
Unnecessary Builds

`.of(context)` can trip you up



Unnecessary Builds

`.of(context)` can trip you up



Unnecessary Builds

MediaQuery

```
const MediaQueryData({
  this.size = Size.zero,
  this.devicePixelRatio = 1.0,
  this.textScaleFactor = 1.0,
  this.platformBrightness = Brightness.light,
  this.padding = EdgeInsets.zero,
  this.viewInsets = EdgeInsets.zero,
  this.systemGestureInsets = EdgeInsets.zero,
  this.viewPadding = EdgeInsets.zero,
  this.alwaysUse24HourFormat = false,
  this.accessibleNavigation = false,
  this.invertColors = false,
  this.highContrast = false,
  this.disableAnimations = false,
  this.boldText = false,
  this.navigationMode = NavigationMode.traditional,
  this.gestureSettings = const DeviceGestureSettings(touchSlop: kTouchSlop),
  this.displayFeatures = const <ui.DisplayFeature>[],
});
```

dependOnInheritedWidgetOfExactT
ype()

Unnecessary Builds

MediaQuery

```
MediaQuery.of(context).size
```

```
MediaQuery.sizeOf(context)
```

```
MediaQuery.orientationOf(context)
MediaQuery.devicePixelRatioOf(context)
MediaQuery.textScaleFactorOf(context)
MediaQuery.platformBrightnessOf(context)
MediaQuery.paddingOf(context)
MediaQuery.viewInsetsOf (context)
MediaQuery.viewPaddingOf(context)
```

ChildBuilder

Solving unnecessary builds without InheritedModel

```
typedef WidgetChildBuilder = Widget Function(BuildContext context, Widget child);

@immutable
class ChildBuilder extends StatelessWidget {
  const ChildBuilder({
    super.key,
    required this.builder,
    required this.child,
  });

  final WidgetChildBuilder builder;
  final Widget child;

  @override
  Widget build(BuildContext context) ⇒ builder(context, child);
}
```

ChildBuilder

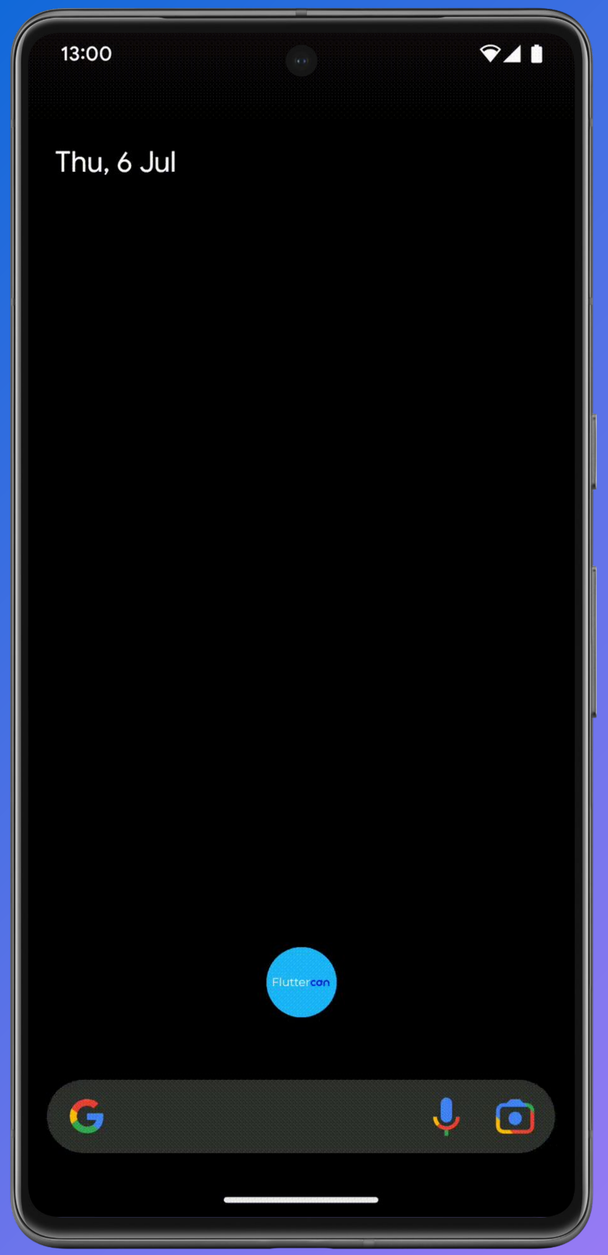
Solving unnecessary builds without InheritedModel

```
ChildBuilder(
  builder: (BuildContext context, Widget child) {
    final theme = Theme.of(context);
    final brightness = MediaQuery.of(context).platformBrightness;
    return Material(
      color: brightness == Brightness.light
          ? theme.colorScheme.primary
          : theme.colorScheme.inversePrimary,
      child: child,
    );
  },
  child: const DeepWidgetHierarchy(),
),
```

StreamBuilder / FutureBuilder

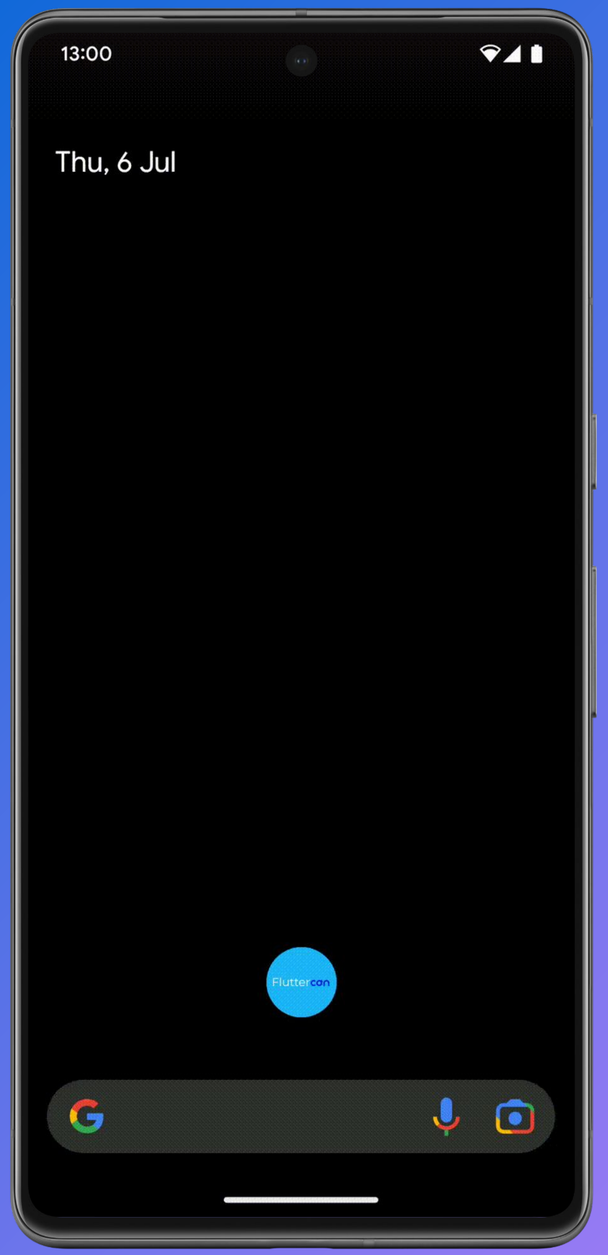
```
Stream<int> autoCounter() ⇒ Stream.periodic(
  Duration(seconds: 1), (el) ⇒ el + 1).take(10);


int _counter = 0;

// build()
StreamBuilder(
  stream: autoCounter(),
  builder: (BuildContext context, AsyncSnapshot<int> snapshot) {
    if (snapshot.connectionState == ConnectionState.waiting) {
      return const Center(child: CircularProgressIndicator());
    } else {
      return Column(
        mainAxisSize: MainAxisSize.min,
        children: [
          Text('Auto counter: ${snapshot.data}'),
          Text('Normal counter: $_counter'),
        ],
      );
    }
  },
),
```

```
FloatingActionButton(
  onPressed: () ⇒ setState(() ⇒ _counter++),
  child: const Icon(Icons.add),
)
```



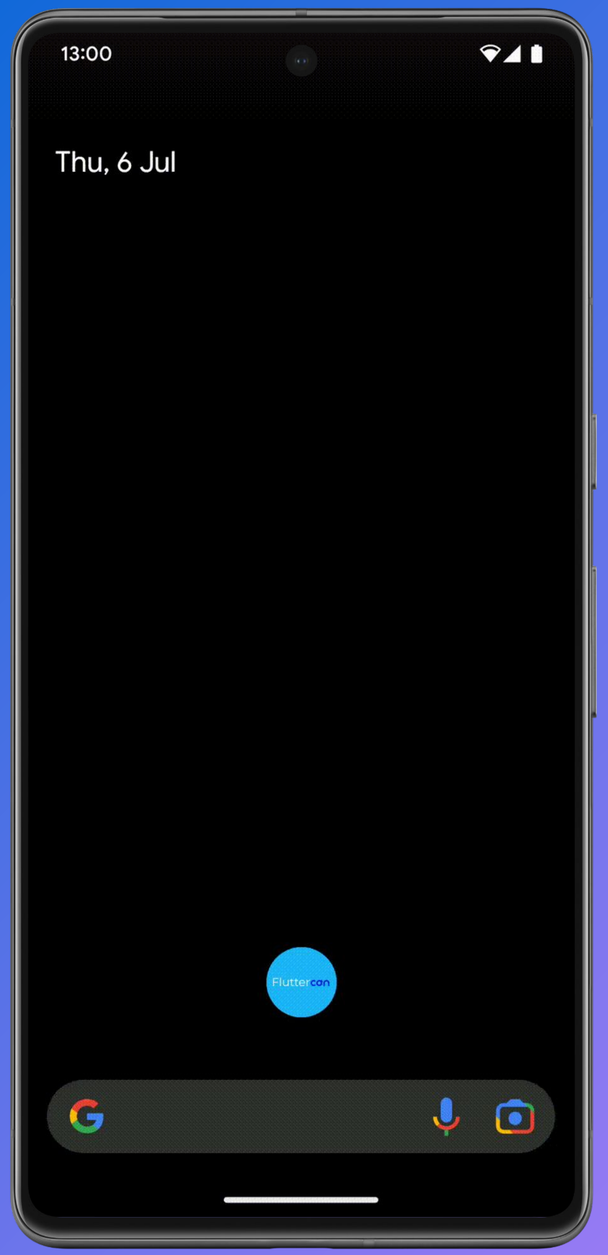
StreamBuilder / FutureBuilder

```
Stream<int> autoCounter() => Stream.periodic(
  Duration(seconds: 1), (el) => el + 1).take(10);

final _autoCounterStream = autoCounter();
int _counter = 0;

// build()
StreamBuilder(
  stream: _autoCounterStream,
  builder: (BuildContext context, AsyncSnapshot<int> snapshot) {
    if (snapshot.connectionState == ConnectionState.waiting) {
      return const Center(child: CircularProgressIndicator());
    } else {
      return Column(
        mainAxisSize: MainAxisSize.min,
        children: [
          Text('Auto counter: ${snapshot.data}'),
          Text('Normal counter: $_counter'),
        ],
      );
    }
  },
),
```



StreamBuilder / FutureBuilder

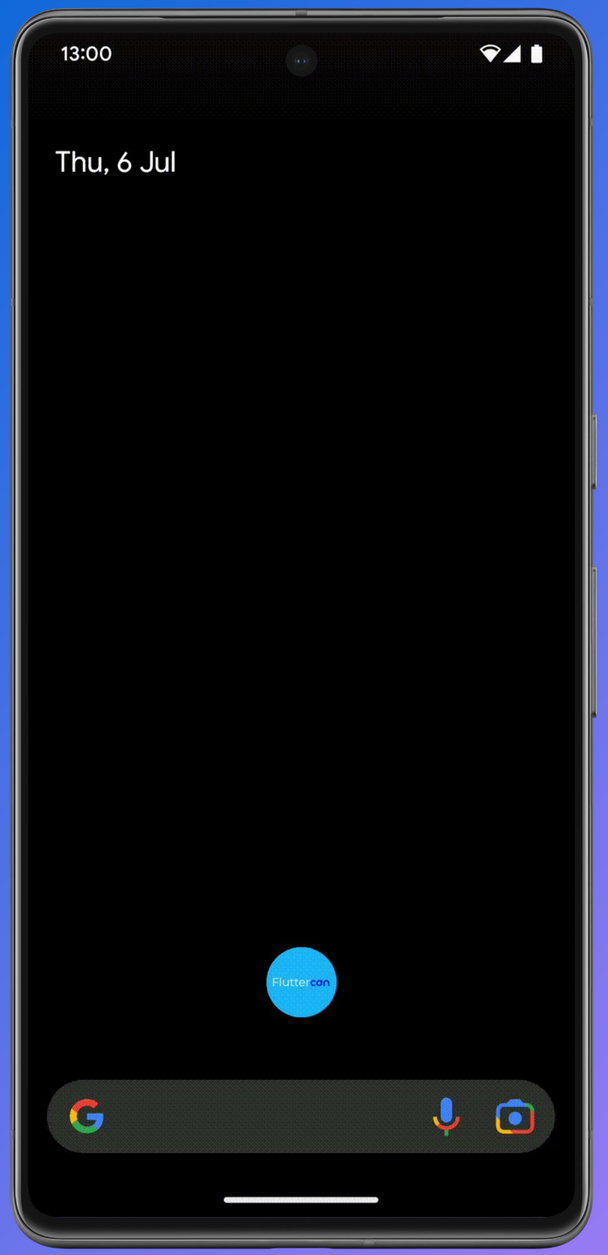
```
Stream<int> autoCounter() => Stream.periodic(
  Duration(seconds: 1), (el) => el + 1).take(10);

final _autoCounterStream = autoCounter();
int _counter = 0;

// build()
StreamBuilder(
  stream: _autoCounterStream,
  builder: (BuildContext context, AsyncSnapshot<int> snapshot) {
    if (snapshot.connectionState == ConnectionState.waiting) {
      return const Center(child: CircularProgressIndicator());
    } else {
      return Column(
        mainAxisSize: MainAxisSize.min,
        children: [
          Text('Auto counter: ${snapshot.data}'),
          Text('Normal counter: $_counter'),
        ],
      );
    }
  },
),
```



Don’t be afraid to make more widgets!

Group, Extract, Abstract and Generify!

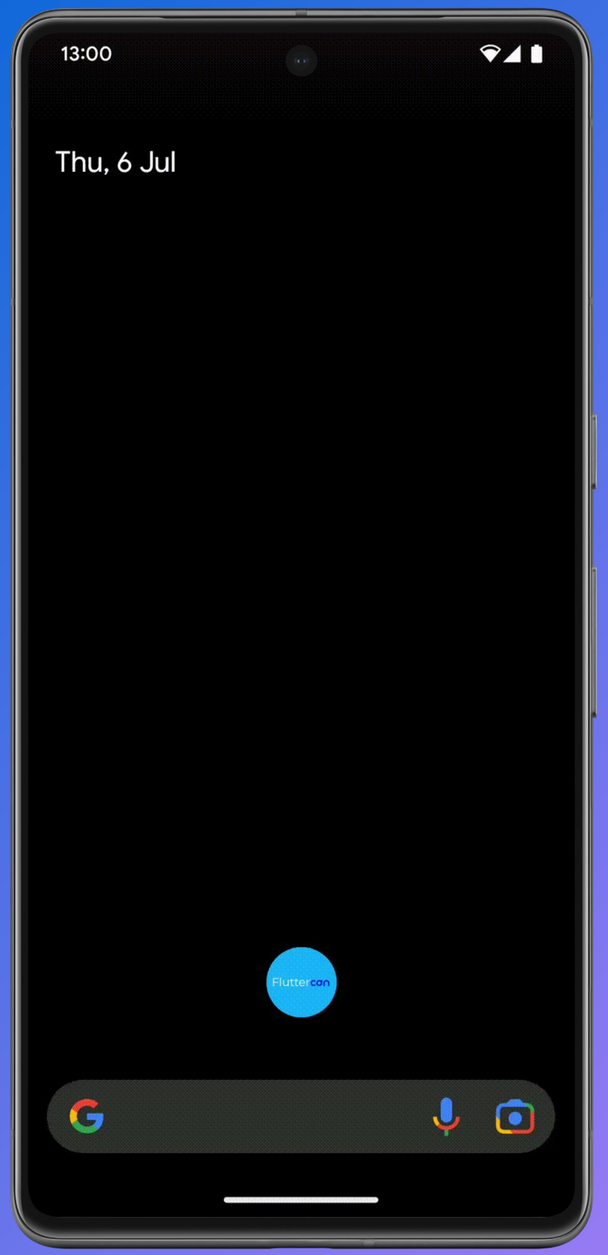
```
StreamCreateBuilder(
  create: (context) ⇒ autoCounter(),
  builder: (BuildContext context, AsyncSnapshot<int> snapshot) {
    if (snapshot.connectionState == ConnectionState.waiting) {
      return const Center(child: CircularProgressIndicator());
    } else {
      return Column(
        mainAxisSize: MainAxisSize.min,
        children: [
          Text('Auto counter: ${snapshot.data}'),
          Text('Normal counter: $_counter'),
        ],
      );
    }
  },
),
```



Don’t be afraid to make more widgets!

Group, Extract, Abstract and Generify!

```
StreamCreateBuilder(
  create: (context) ⇒ autoCounter(),
  builder: (BuildContext context, AsyncSnapshot<int> snapshot) {
    if (snapshot.connectionState == ConnectionState.waiting) {
      return const Center(child: CircularProgressIndicator());
    } else {
      return Column(
        mainAxisSize: MainAxisSize.min,
        children: [
          Text('Auto counter: ${snapshot.data}'),
          Text('Normal counter: $_counter'),
        ],
      );
    }
  },
),
```



AsyncSnapshot

when() extension

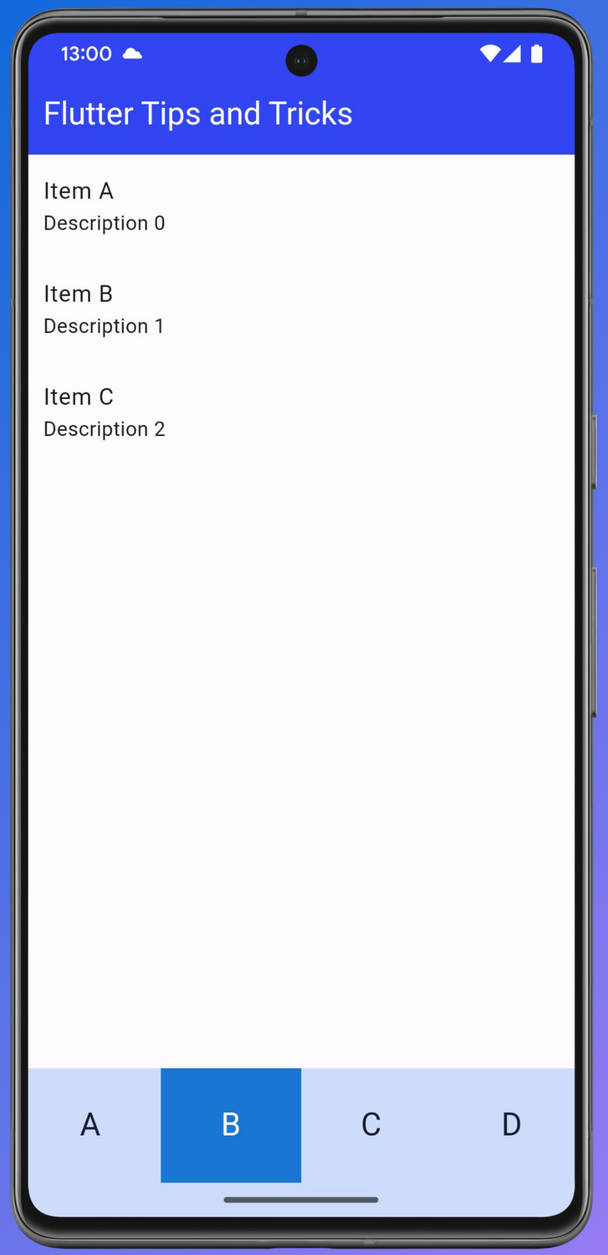
```
extension WhenAsyncSnapshot<T> on AsyncSnapshot<T> {
  R when<R>({
    R Function()? empty,
    R Function(dynamic error, StackTrace? stackTrace)? error,
    R Function()? loading,
    R Function(T value)? data,
  }) {
    if (hasData && data ≠ null) // If we have data t
      return data(requireData);
    if (connectionState ≠ ConnectionState.done && lo
      return loading();
    else if (hasError && error ≠ null) // Did we get
      return error(this.error, stackTrace);
    else if (empty ≠ null) // No data, not loading,
      return empty();
    else // We only get here if the developer does no
      throw UnsupportedError('Missing parameters to w
  }
}
```

```
StreamBuilder(
  stream: _stream,
  builder: (BuildContext context, AsyncSnapshot<int> snapshot) {
    return snapshot.when(
      empty: () ⇒ Text('no data'),
      error: (error, _) ⇒ ErrorWidget(error),
      loading: () ⇒ Center(child: CircularProgressIndicator()),
      data: (data) ⇒ Text('$data'),
    );
  },
)
```

Widget Layout

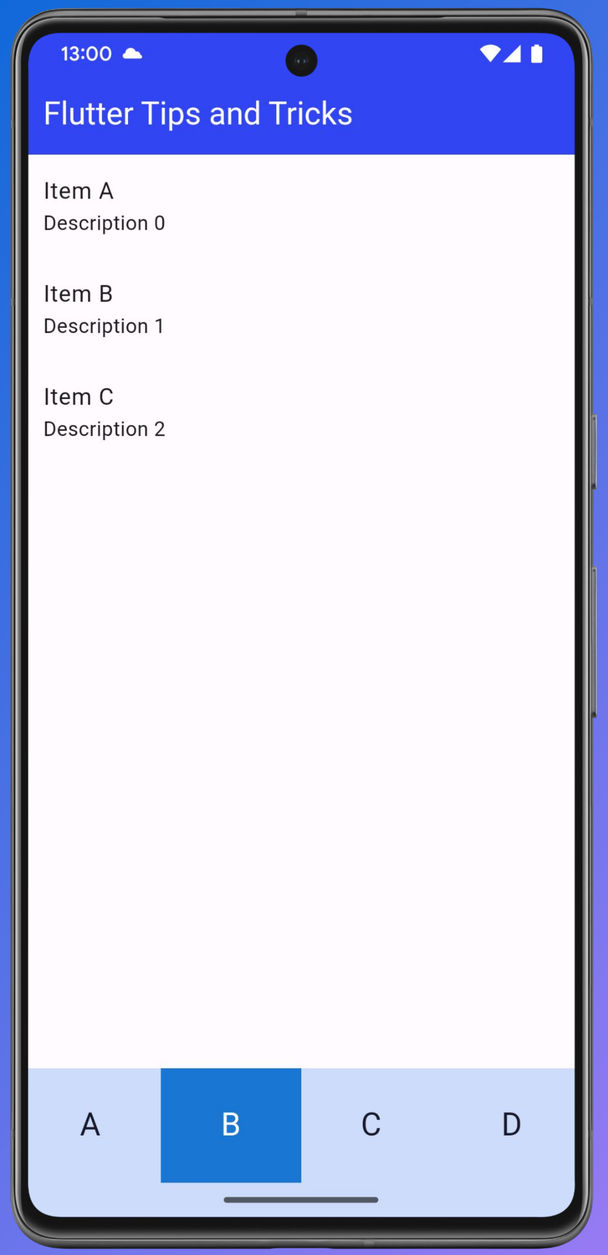
IterableExtensions

```
final _items = <String, Widget>{
  'item1': const Text('Item A'),
  'item2': const Text('Item B'),
  'item3': const Text('Item C'),
};

Widget build(BuildContext context) {
  return Material(
    child: ListView(
      children: [
        for (final (i, MapEntry(key: k, value: v)) in _items.entries.indexed) //
          ListTile(
            key: Key(k),
            onTap: () ⇒ debugPrint('Tapped Item $i'),
            title: v,
            subtitle: Text('Description $i'),
          ),
      ],
    ),
  );
}
```



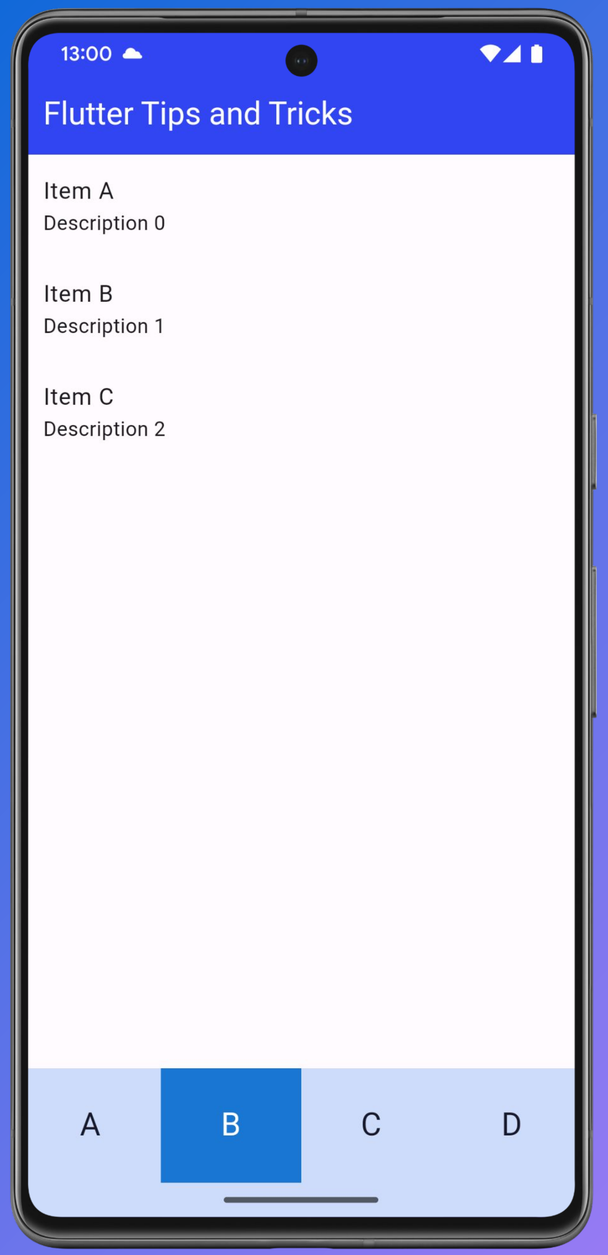
IterableExtensions

```
final _items = <String, Widget>{
  'item1': const Text('Item A'),
  'item2': const Text('Item B'),
  'item3': const Text('Item C'),
};

Widget build(BuildContext context) {
  return Material(
    child: ListView(
      children: [
        for (final (i, MapEntry(key: k, value: v)) in _items.entries.indexed) //
          ListTile(
            key: Key(k),
            onTap: () ⇒ debugPrint('Tapped Item $i'),
            title: v,
            subtitle: Text('Description $i'),
          ),
      ],
    ),
  );
}
```



IterableExtensions

```
final _items = <String, Widget>{
  'item1': const Text('Item A'),
  'item2': const Text('Item B'),
  'item3': const Text('Item C'),
};

Widget build(BuildContext context) {
  return Material(
    child: ListView(
      children: [
        for (final (i, MapEntry(key: k, value: v)) in _items.entries.indexed) //
          ListTile(
            key: Key(k),
            onTap: () ⇒ debugPrint('Tapped Item $i'),
            title: v,
            subtitle: Text('Description $i'),
          ),
      ],
    ),
  );
}
```



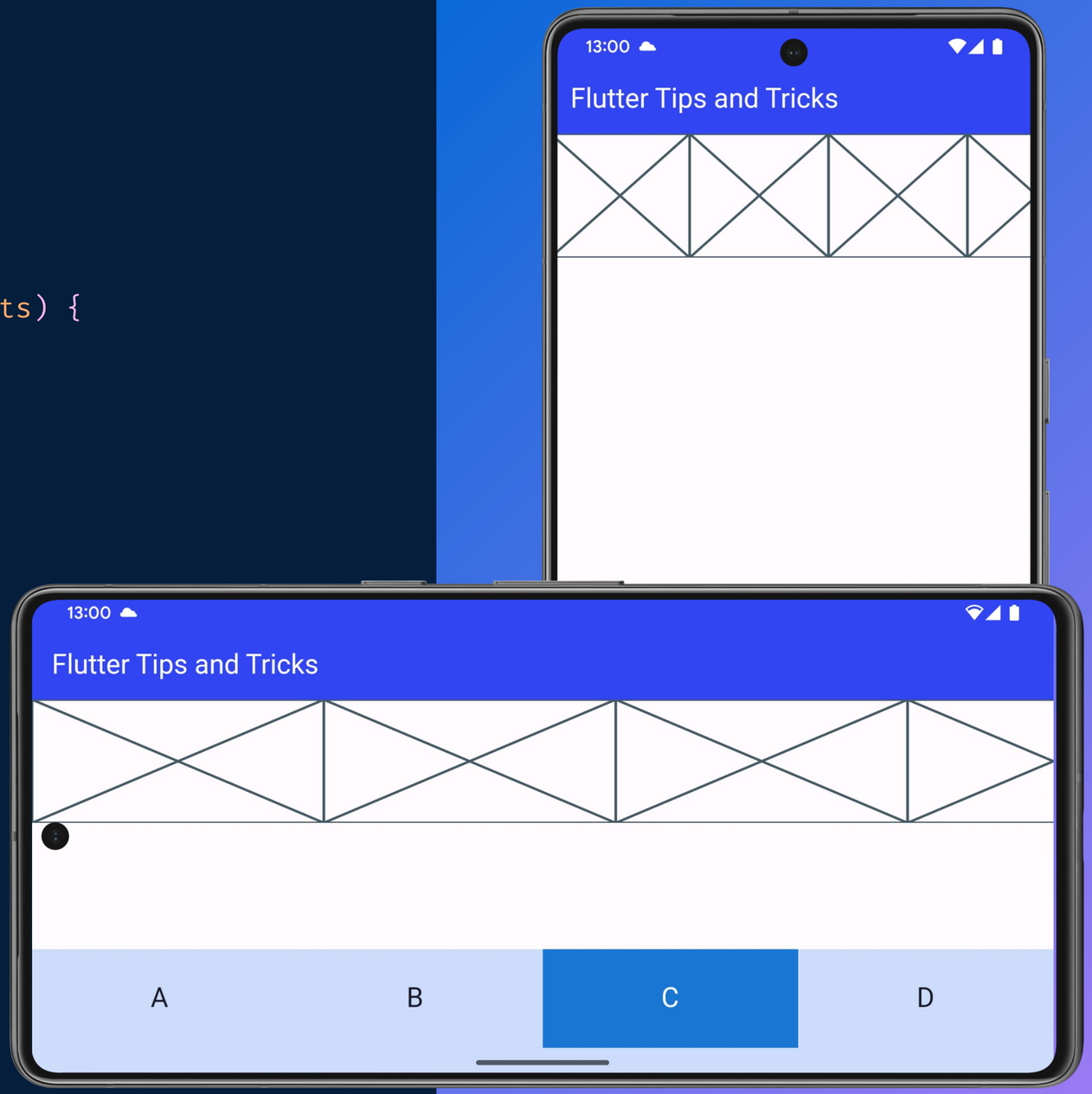
Layout Flow

The basics

LayoutBuilder

Misused

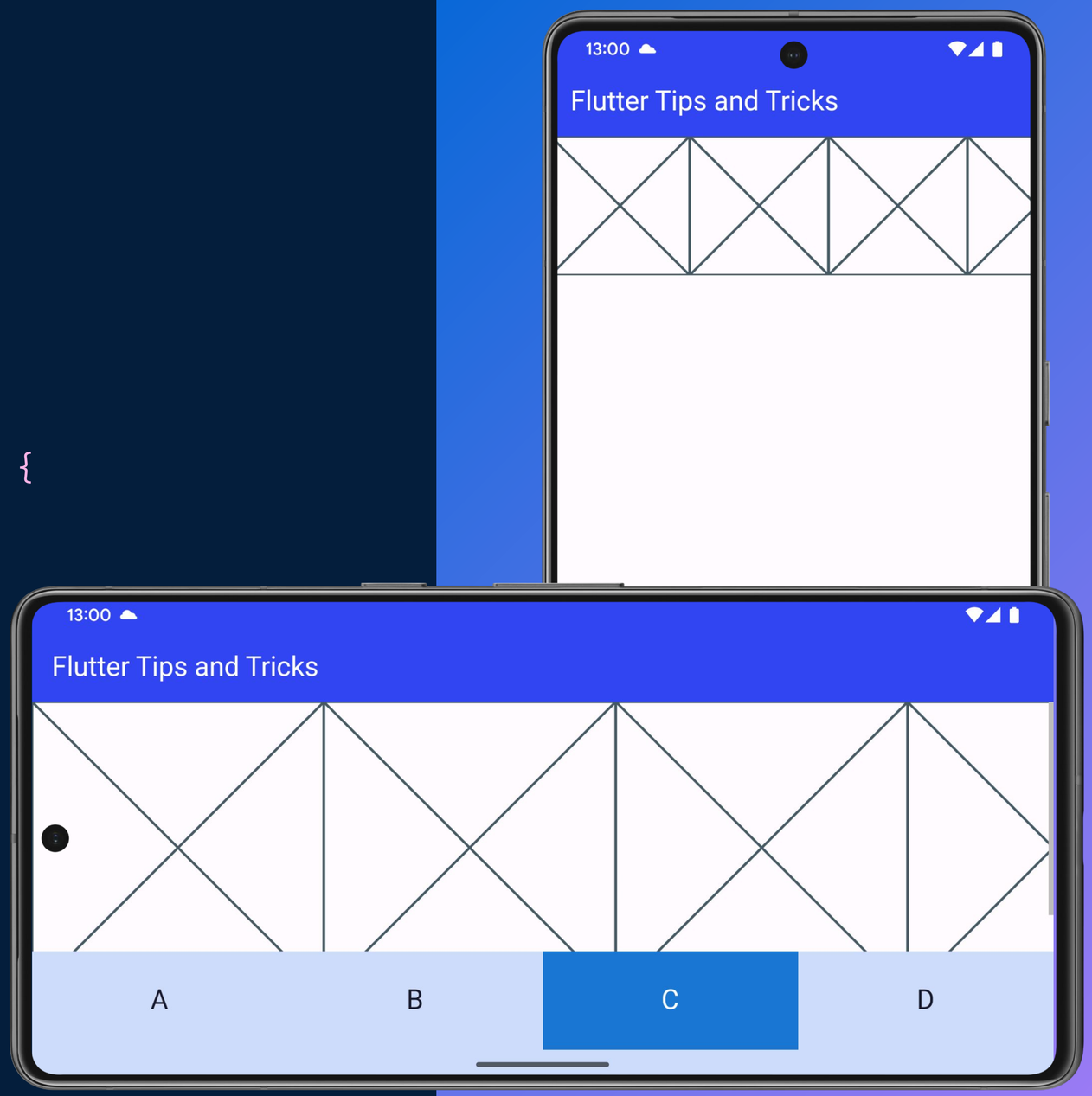
```
LayoutBuilder(
  builder: (BuildContext context, BoxConstraints constraints) {
    final itemWidth = constraints.maxWidth / 3.5;
    return SizedBox(
      height: 100.0,
      child: ListView.builder(
        scrollDirection: Axis.horizontal,
        itemBuilder: (BuildContext context, int index) {
          return SizedBox(
            width: itemWidth,
            child: const Placeholder(),
          );
        },
      ),
    );
  },
),
```



LayoutBuilder

Corrected

```
AspectRatio(
  aspectRatio: 3.5,
  child: ListView.builder(
    scrollDirection: Axis.horizontal,
    itemBuilder: (BuildContext context, int index) {
      return const AspectRatio(
        aspectRatio: 1.0,
        child: Placeholder(),
      );
    },
  ),
),
```



Late final

Simple cache

```
late final user = Provider.of<User>(context);

@override
Widget build(BuildContext context) {
  return ...;
}
```

Late final

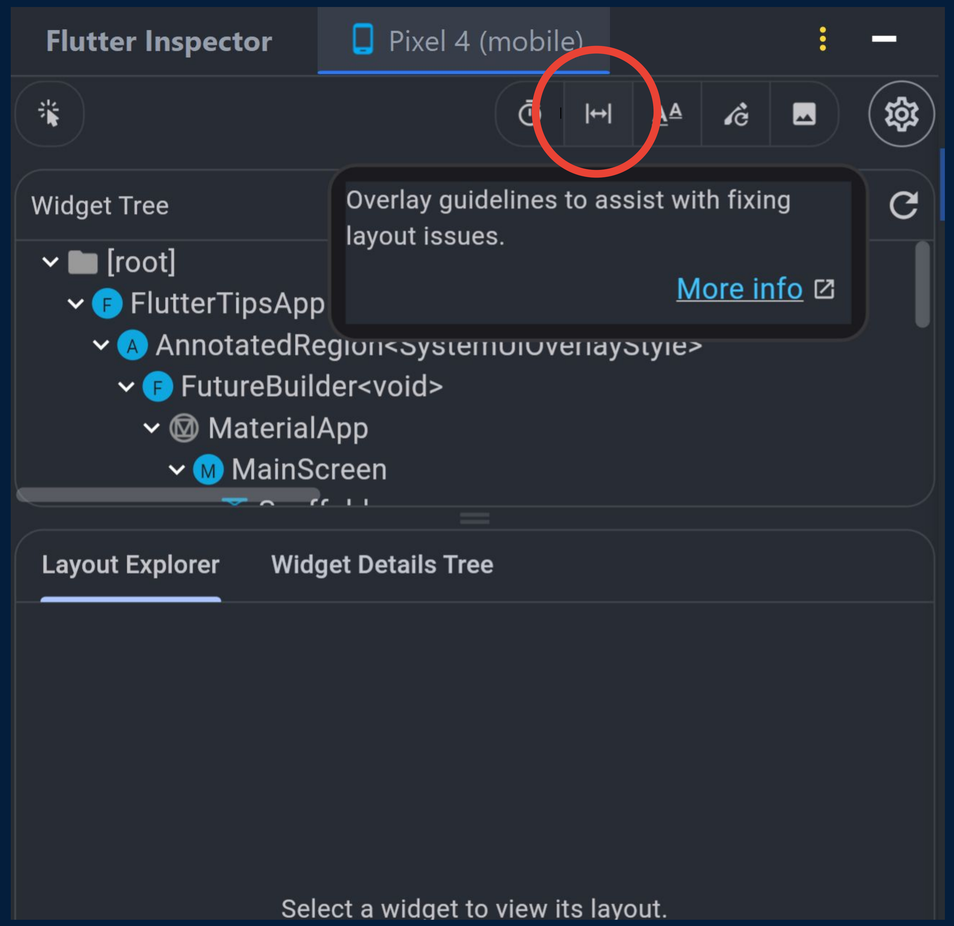
Simple cache

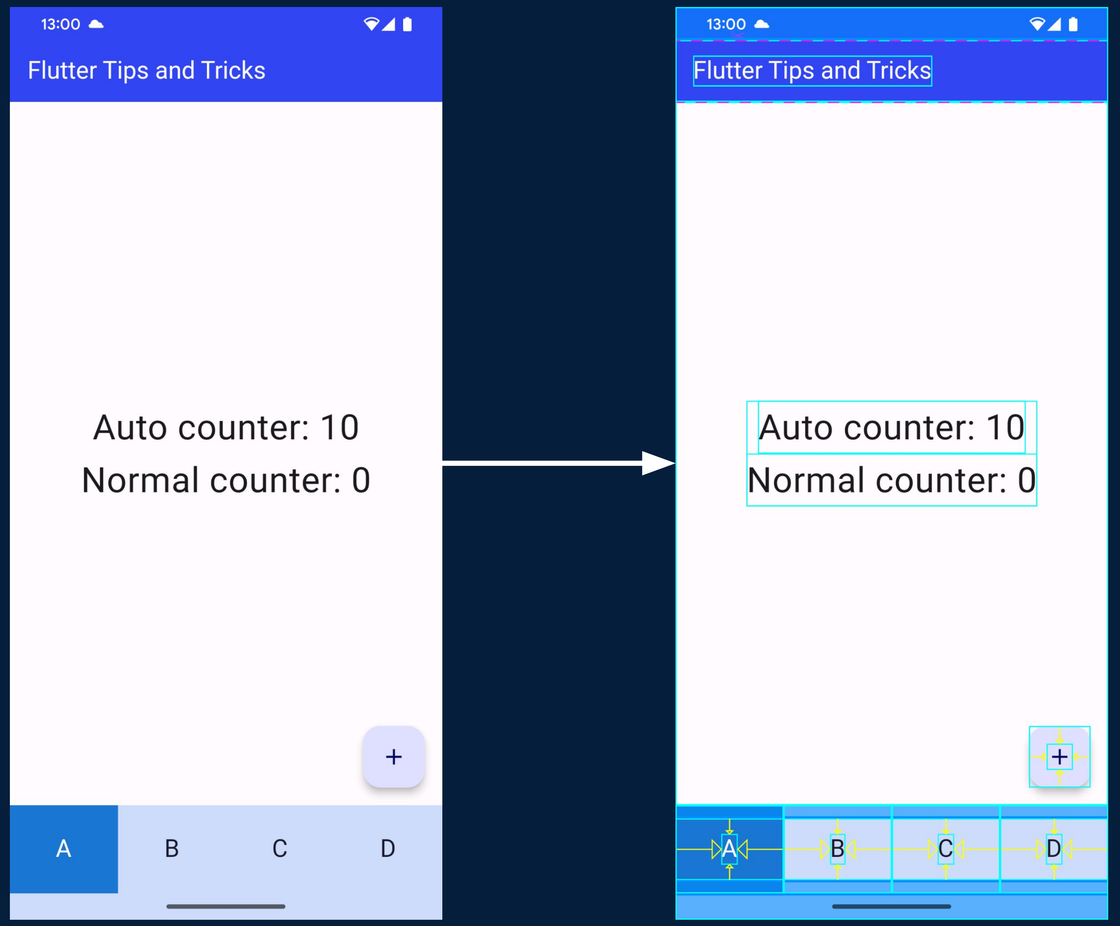
```
late final user = Provider.of<User>(context);

@override
Widget build(BuildContext context) {
  return ...;
}
```

Debugging

Debug Paint

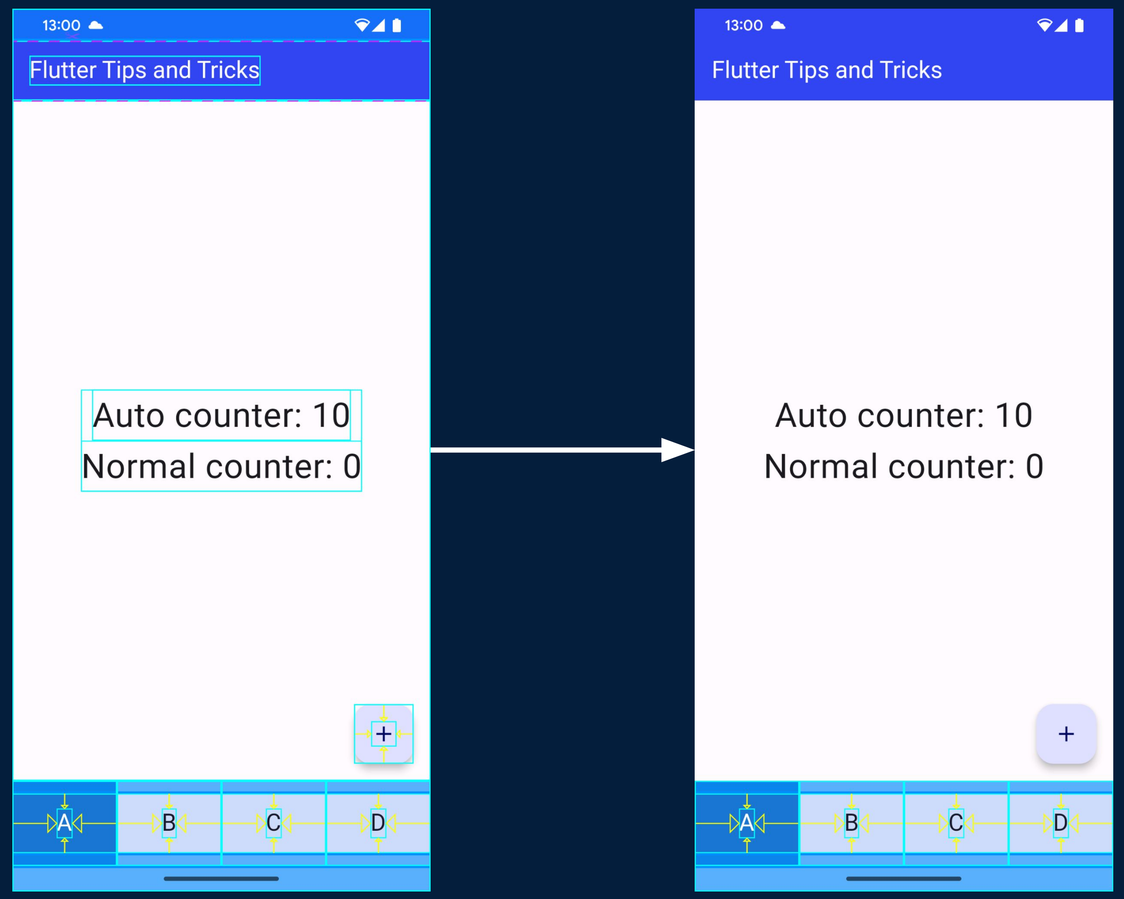
debugPaintSizeEnabled





Debug Paint

debugPaintSizeEnabled



Debug Paint

debugPaintSizeEnabled

```
class ShowDebugPaint extends SingleChildRenderObjectWidget {
  const ShowDebugPaint({
    Key? key,
    this.enabled = true,
    required Widget child,
  }) : super(key: key, child: child);

  final bool enabled;

  @override
  RenderObject createRenderObject(BuildContext context) {
    return RenderShowDebugPaint(enabled: enabled);
  }

  @override
  void updateRenderObject(BuildContext context, RenderShowDebugPaint renderObject) {
    renderObject.enabled = enabled;
  }
}
```

Debug Paint

debugPaintSizeEnabled

```
class RenderShowDebugPaint extends RenderProxyBox {
  RenderShowDebugPaint({required bool enabled, RenderBox? child})
      : _enabled = enabled, super(child);

  bool _enabled;
  bool get enabled => _enabled;

  set enabled(bool value) {
    if (_enabled != value) {
      _enabled = value; markNeedsPaint();
    }
  }

  @override
  void paint(PaintingContext context, Offset offset) {
    final previousState = debugPaintSizeEnabled;
    debugPaintSizeEnabled = enabled;
    super.paint(context, offset);
    debugPaintSizeEnabled = previousState;
  }
}
```

Debug Paint

devangels_show_debug_paint 0.0.1

Published 13 hours ago Dart 3 compatible

SDK FLUTTER PLATFORM ANDROID IOS LINUX MACOS WEB WINDOWS 0

Readme Changelog Example Installing Versions Scores

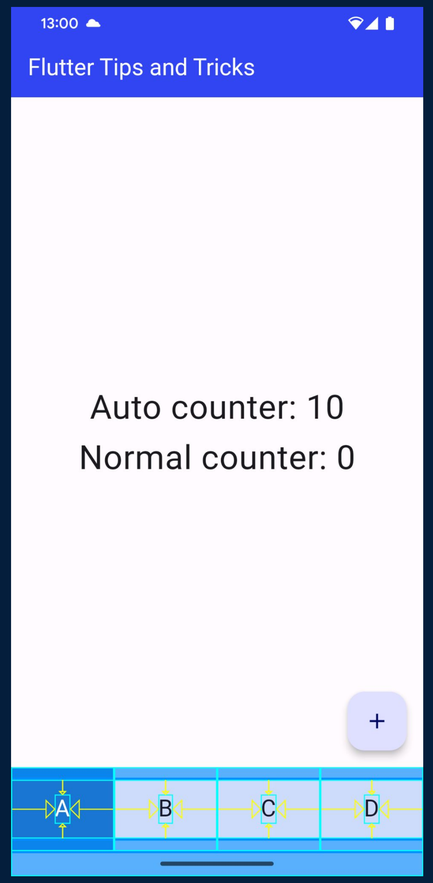
devangels_show_debug_paint

DEVANGELS .LONDON

An open-source widget filled with ✨ brought to you by the DevAngels team 😇

license MIT

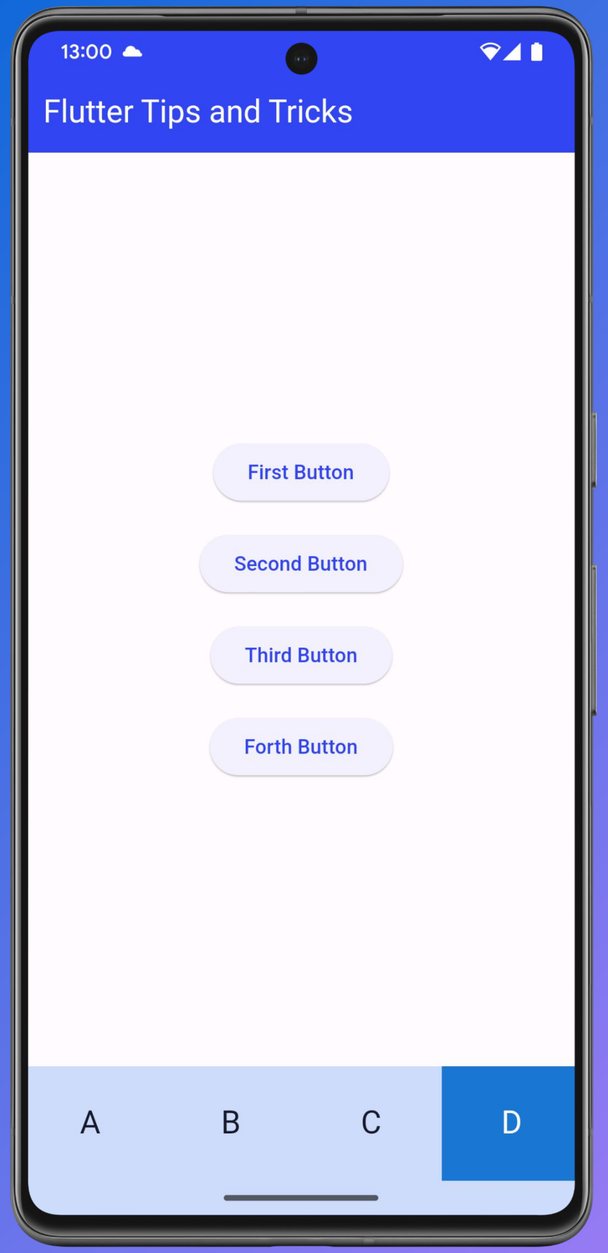
A debug widget which paints outlines for a single widget and it's child subtree.



Accessibility

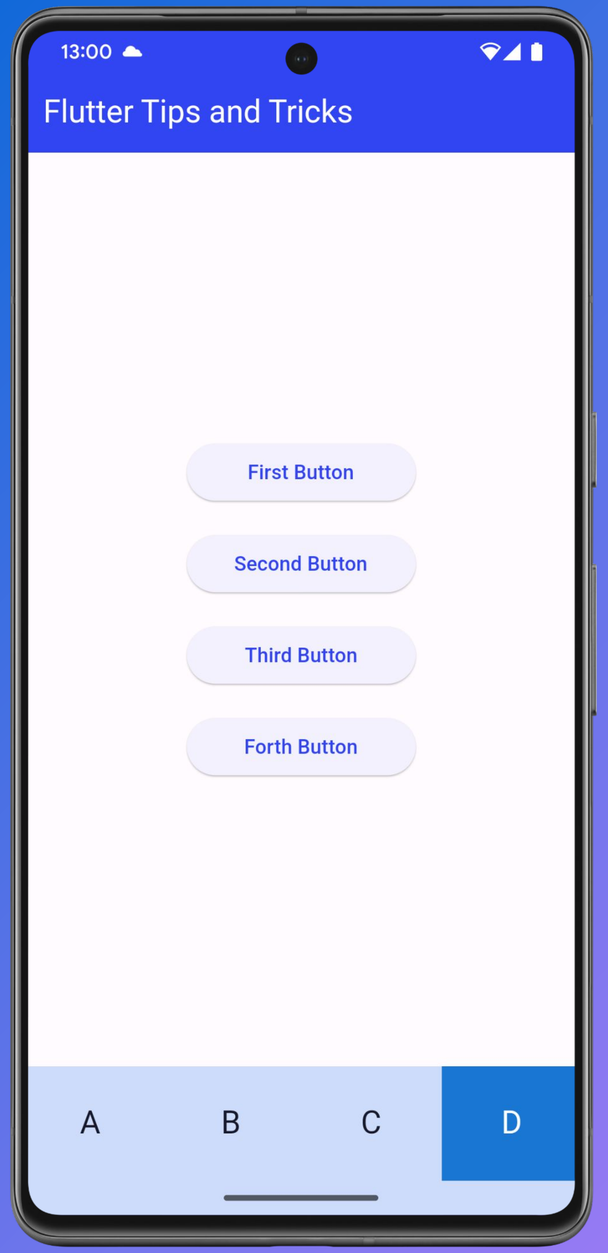
No fixed widths/heights

```
Center(
  child: Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      ElevatedButton(onPressed: () {}, child: const Text('First Button')),
      verticalMargin16,
      ElevatedButton(onPressed: () {}, child: const Text('Second Button')),
      verticalMargin16,
      ElevatedButton(onPressed: () {}, child: const Text('Third Button')),
      verticalMargin16,
      ElevatedButton(onPressed: () {}, child: const Text('Forth Button')),
    ],
  ),
)
```



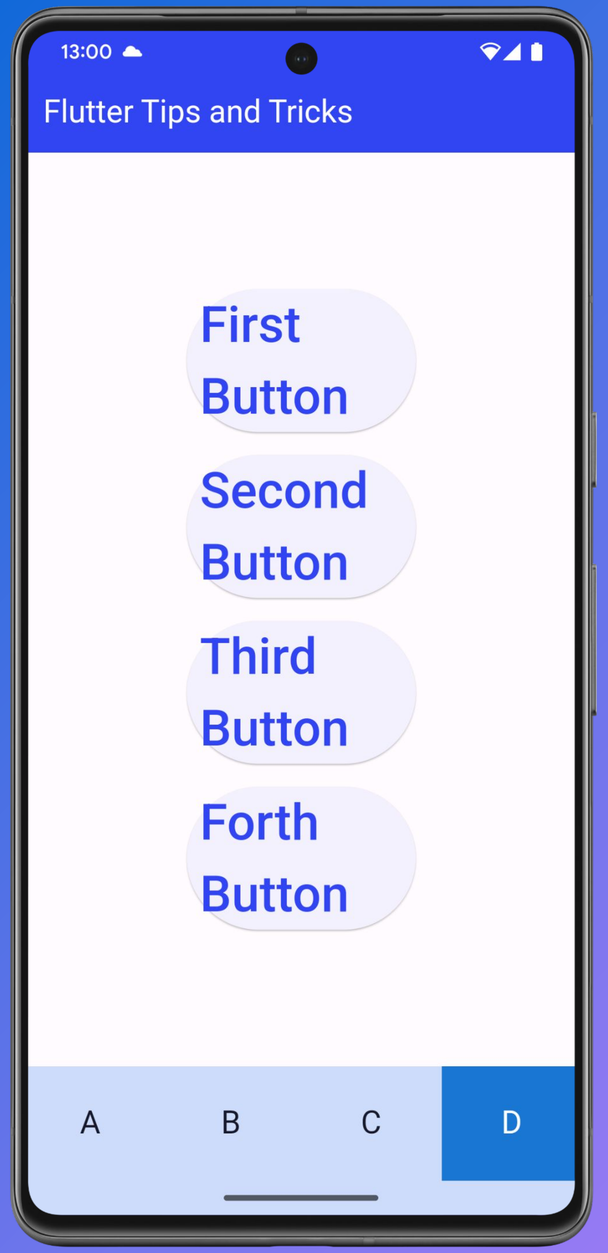
No fixed widths/heights

```
Center(
  child: SizedBox(
    width: 160.0,
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      crossAxisAlignment: CrossAxisAlignment.stretch,
      children: [
        ElevatedButton(onPressed: () {}, child: const Text('First Button')),
        verticalMargin16,
        ElevatedButton(onPressed: () {}, child: const Text('Second Button')),
        verticalMargin16,
        ElevatedButton(onPressed: () {}, child: const Text('Third Button')),
        verticalMargin16,
        ElevatedButton(onPressed: () {}, child: const Text('Forth Button')),
      ],
    ),
  ),
)
```



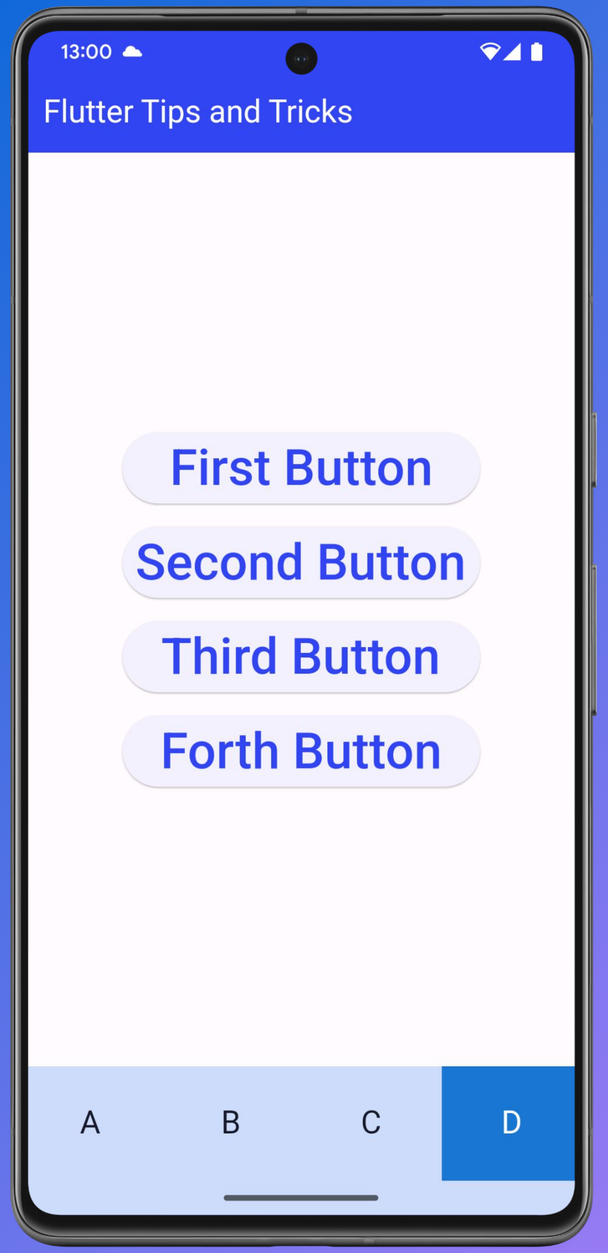
No fixed widths/heights

```
Center(
  child: SizedBox(
    width: 160.0,
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      crossAxisAlignment: CrossAxisAlignment.stretch,
      children: [
        ElevatedButton(onPressed: () {}, child: const Text('First Button')),
        verticalMargin16,
        ElevatedButton(onPressed: () {}, child: const Text('Second Button')),
        verticalMargin16,
        ElevatedButton(onPressed: () {}, child: const Text('Third Button')),
        verticalMargin16,
        ElevatedButton(onPressed: () {}, child: const Text('Forth Button')),
      ],
    ),
  ),
)
```



No fixed widths/heights

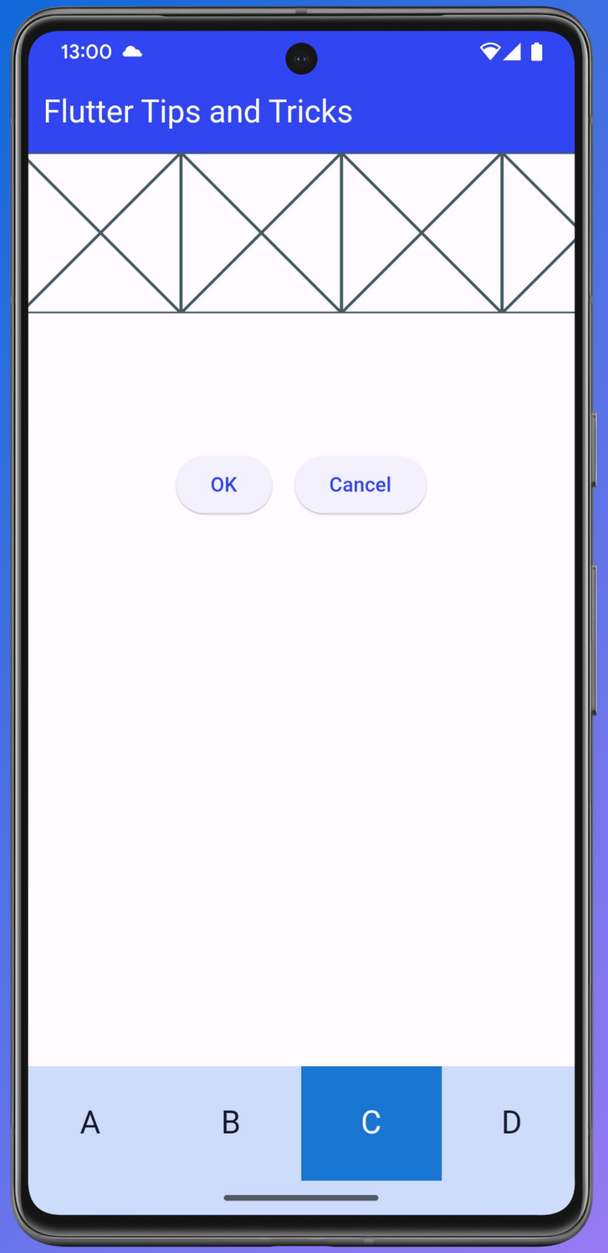
```
Center(
  child: IntrinsicWidth(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      crossAxisAlignment: CrossAxisAlignment.stretch,
      children: [
        ElevatedButton(onPressed: () {}, child: const Text('First Button')),
        verticalMargin16,
        ElevatedButton(onPressed: () {}, child: const Text('Second Button')),
        verticalMargin16,
        ElevatedButton(onPressed: () {}, child: const Text('Third Button')),
        verticalMargin16,
        ElevatedButton(onPressed: () {}, child: const Text('Forth Button')),
      ],
    ),
  ),
)
```



Localization

No fixed widths and heights

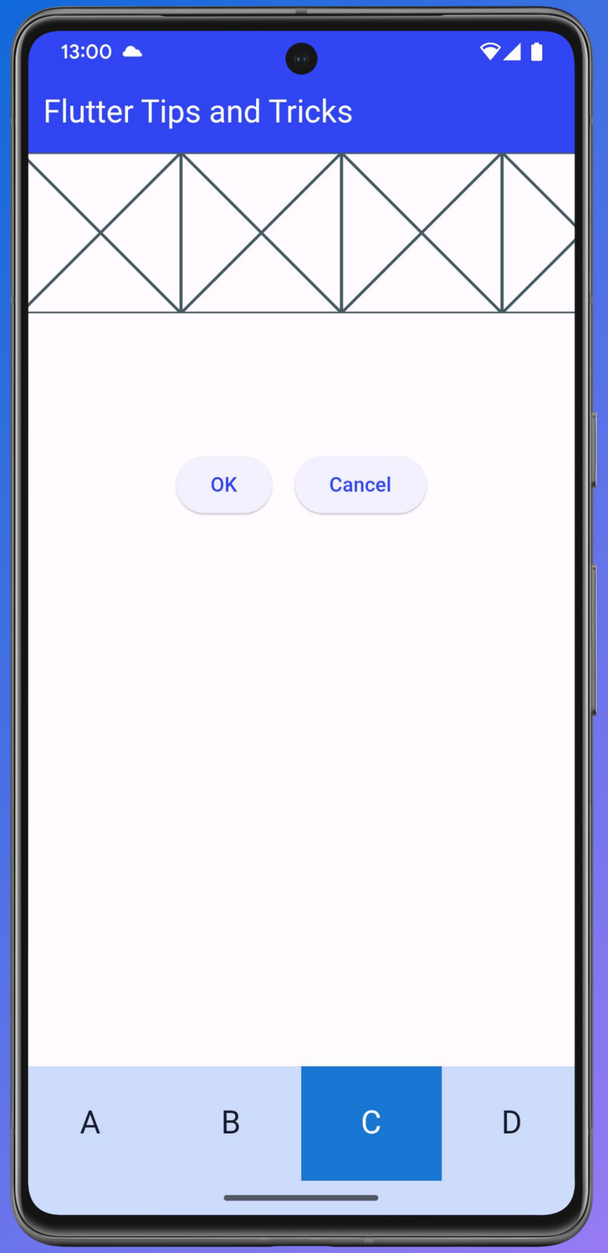
```
Column(
  crossAxisAlignment: CrossAxisAlignment.stretch,
  children: [
    // ...
    verticalMargin48 + verticalMargin48,
    Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        ElevatedButton(onPressed: (){}, child: const Text('OK')),
        horizontalMargin16,
        ElevatedButton(onPressed: (){}, child: const Text('Cancel')),
      ],
    ),
  ],
)
```



Localization

IntrinsicWidth

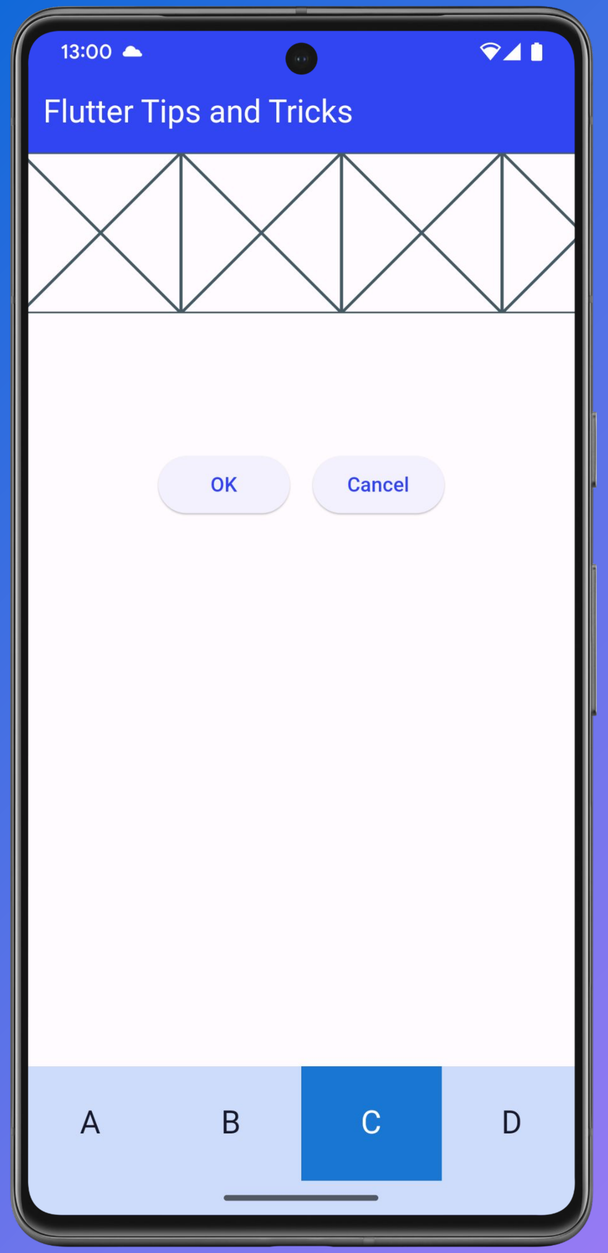
```
Column(
  crossAxisAlignment: CrossAxisAlignment.stretch,
  children: [
    // ...
    verticalMargin48 + verticalMargin48,
    IntrinsicWidth(
      child: Row(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Expanded(
            child: ElevatedButton(onPressed: () {}, child: const Text('OK')),
          ),
          horizontalMargin16,
          Expanded(
            child: ElevatedButton(onPressed: () {}, child: const Text('Cancel')),
          ),
        ],
      ),
    ),
  ],
)
```



Localization

IntrinsicWidth

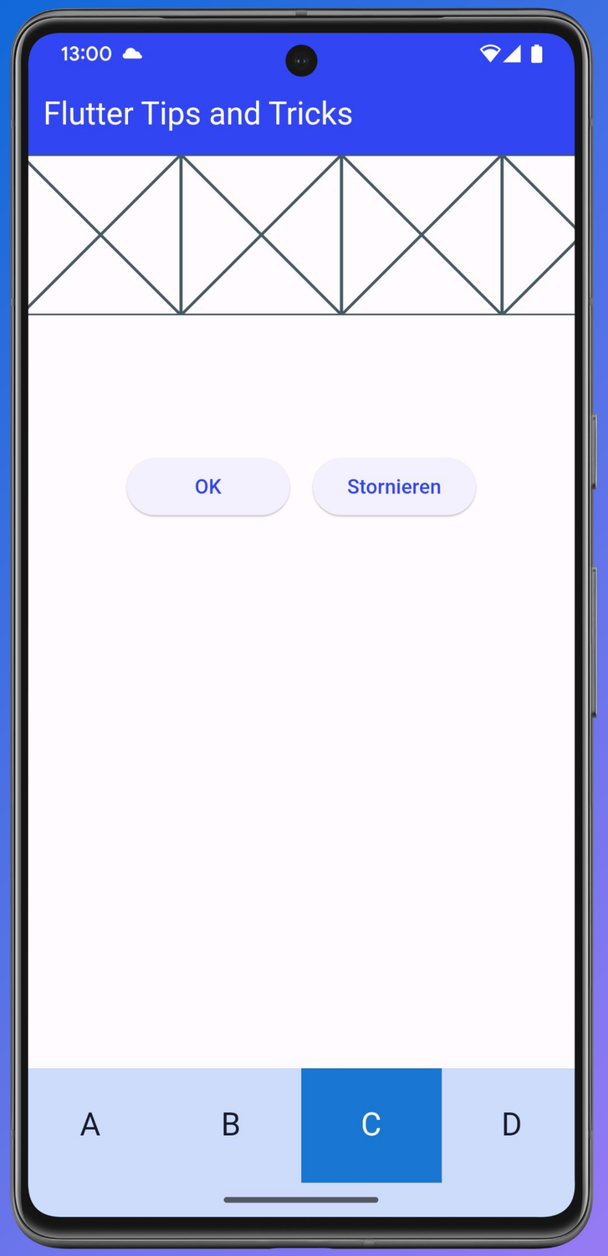
```
Column(
  crossAxisAlignment: CrossAxisAlignment.stretch,
  children: [
    // ...
    verticalMargin48 + verticalMargin48,
    IntrinsicWidth(
      child: Row(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Expanded(
            child: ElevatedButton(onPressed: () {}, child: const Text('OK')),
          ),
          horizontalMargin16,
          Expanded(
            child: ElevatedButton(onPressed: () {}, child: const Text('Cancel')),
          ),
        ],
      ),
    ),
  ],
)
```



Localization

IntrinsicWidth

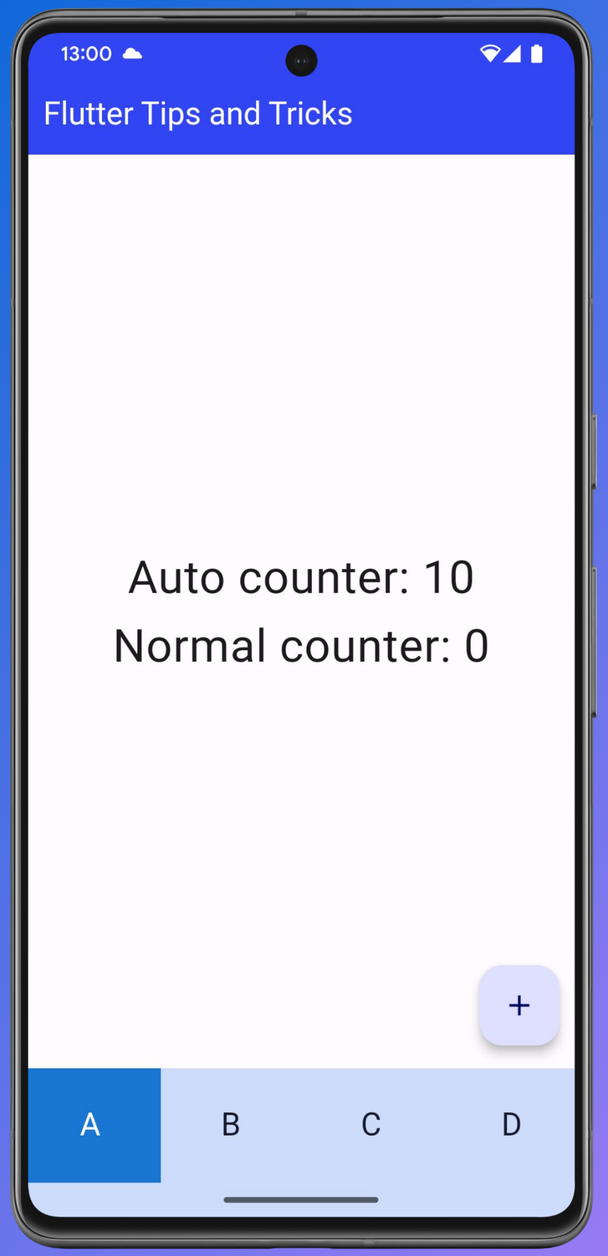
```
// ...
IntrinsicWidth(
  child: Row(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      Expanded(
        child: ElevatedButton(onPressed: () {}, child: const Text('OK')),
      ),
      horizontalMargin16,
      Expanded(
        child: ElevatedButton(onPressed: () {}, child: const Text('Stornieren')),
      ),
    ],
  ),
),
// ...
```



SemanticsDebugger

```
MaterialApp(
  debugShowCheckedModeBanner: false,
  title: 'Flutter Tips and Tricks',
  theme: appTheme,
  navigatorKey: _navigatorKey,
  onGenerateRoute: (RouteSettings settings) {
    return switch (settings.name) {
      SplashScreen.routeName ⇒ SplashScreen.route(),
      MainScreen.routeName ⇒ MainScreen.route(),
      _ ⇒ null,
    };
  },
)
```

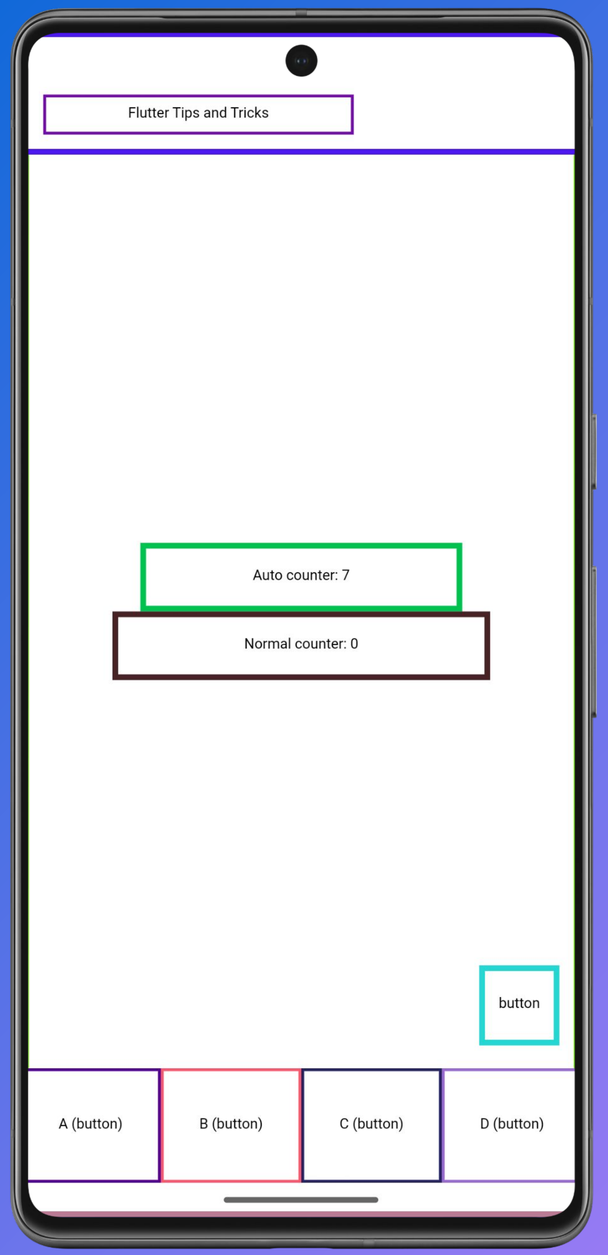
https://docs.flutter.dev/accessibility-and-localization/accessibility



SemanticsDebugger

```
MaterialApp(
  debugShowCheckedModeBanner: false,
  title: 'Flutter Tips and Tricks',
  theme: appTheme,
  navigatorKey: _navigatorKey,
  onGenerateRoute: (RouteSettings settings) {
    return switch (settings.name) {
      SplashScreen.routeName ⇒ SplashScreen.route(),
      MainScreen.routeName ⇒ MainScreen.route(),
      _ ⇒ null,
    };
  },
  showSemanticsDebugger: true,
)
```

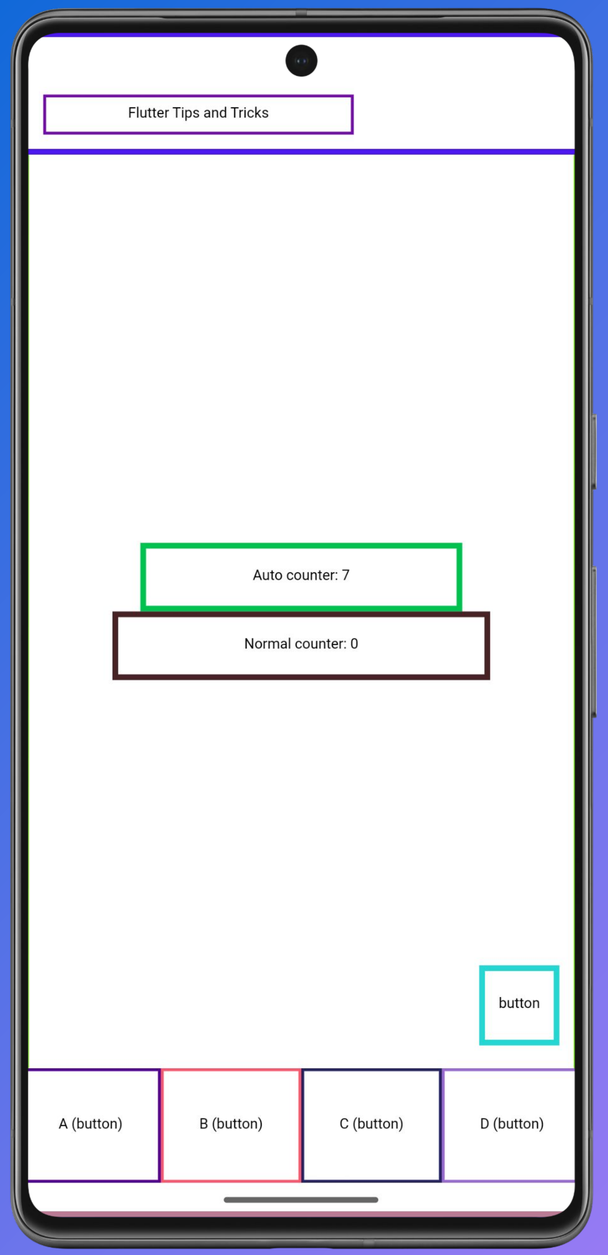
https://docs.flutter.dev/accessibility-and-localization/accessibility



SemanticsDebugger

```
MaterialApp(
  debugShowCheckedModeBanner: false,
  title: 'Flutter Tips and Tricks',
  theme: appTheme,
  navigatorKey: _navigatorKey,
  onGenerateRoute: (RouteSettings settings) {
    return switch (settings.name) {
      SplashScreen.routeName ⇒ SplashScreen.route(),
      MainScreen.routeName ⇒ MainScreen.route(),
      _ ⇒ null,
    };
  },
  showSemanticsDebugger: true,
)
```

https://docs.flutter.dev/accessibility-and-localization/accessibility



Common Widgets

```
const emptyWidget = SizedBox();
const emptyWidgetWide = SizedBox(width: double.infinity);

// Margins

const horizontalMargin4 = SizedBox(width: 4.0);
const horizontalMargin8 = SizedBox(width: 8.0);
const horizontalMargin12 = SizedBox(width: 12.0);
const horizontalMargin16 = SizedBox(width: 16.0);
const horizontalMargin24 = SizedBox(width: 24.0);
const horizontalMargin32 = SizedBox(width: 32.0);
const horizontalMargin48 = SizedBox(width: 48.0);

const verticalMargin4 = SizedBox(height: 4.0);
const verticalMargin8 = SizedBox(height: 8.0);
const verticalMargin12 = SizedBox(height: 12.0);
const verticalMargin16 = SizedBox(height: 16.0);
const verticalMargin24 = SizedBox(height: 24.0);
const verticalMargin32 = SizedBox(height: 32.0);
const verticalMargin48 = SizedBox(height: 48.0);
```

```
// Paddings

const emptyPadding = EdgeInsets.zero;

const horizontalPadding4 = EdgeInsets.symmetric(horizontal: 4.0);
const horizontalPadding8 = EdgeInsets.symmetric(horizontal: 8.0);
const horizontalPadding12 = EdgeInsets.symmetric(horizontal: 12.0);
const horizontalPadding16 = EdgeInsets.symmetric(horizontal: 16.0);
const horizontalPadding24 = EdgeInsets.symmetric(horizontal: 24.0);
const horizontalPadding32 = EdgeInsets.symmetric(horizontal: 32.0);
const horizontalPadding48 = EdgeInsets.symmetric(horizontal: 48.0);

const verticalPadding2 = EdgeInsets.symmetric(vertical: 2.0);
const verticalPadding4 = EdgeInsets.symmetric(vertical: 4.0);
const verticalPadding8 = EdgeInsets.symmetric(vertical: 8.0);
//…

const allPadding4 = EdgeInsets.all(4.0);
const allPadding8 = EdgeInsets.all(8.0);
```

Thank You

Questions?

Simon Lightfoot

Flutter Community Lead
CTO of DevAngels London

@devangelslondon

github.com/slightfoot/flutter_tips_and_tricks

Flutter Community

DEVANGELS .LONDON