

Boosty and Crypto payments are now supported! Checkout Payments tab.



[Home](#) [About](#) [All articles](#) [Payments](#) [GitHub](#)

Michael Lazeby



[Sign in](#)

[Subscribe](#)

FLUTTER

# Popups in Flutter



Michael Lazeby

10 Jun 2024 — 10 min read

## Popups in Flutter

Popups are essential UI elements that provide contextual information or options without navigating away from the current screen. They can be used for dropdowns, tooltips, dialogs, and more. In this article we'll find out how to craft them **the right way**.

### Outline

1. [Popup Widget](#)
  - a. [Example Usage](#)
  - b. [Making popups focusable and dismissible](#)
  - c. [How does it work](#)
2. [Creating reusable popups](#)
  - a. [Dropdown](#)
  - b. [Animated Tooltip](#)
3. [Built-in widgets: possibilities and limitations](#)
  - a. [MenuAnchor](#)
  - b. [Dropdowns](#)



## Popup Widget

The primary requirement for popups is to position them correctly. Dropdowns typically appear below the button, while tooltips can appear at the top, left, right, or cover the anchor.

Another important consideration is responsiveness. For instance, if the dropdown content doesn't fit below the anchor, it should be positioned above it, provided there is enough space. The popup should always remain within the screen's boundaries.

To address these needs, I've developed the `Popup` widget. You can access the code for it [here](#). This widget includes the following features:

- Positioning of the popup relative to an anchor
- Constraining the popup's size to stay within screen dimensions

- Managing overflow by flipping or offsetting the popup if it exceeds available space

## Example Usage

Here is a simple popup that displays a card with three items under a button:

```
class _BasicPopup extends StatelessWidget {
  const _BasicPopup();

  @override
  Widget build(BuildContext context) => Popup(
    // 3 !!!
    followerAnchor: Alignment.topCenter,
    targetAnchor: Alignment.bottomCenter,
    // 1 !!!
    child: (context, controller) => FilledButton(
      onPressed: () => controller.show(),
      child: const Text('Show Simple Popup'),
    ),
    // 2 !!!
    follower: (context, controller) => SizedBox(
      width: 200,
      child: Card(
        margin: EdgeInsets.zero,
        child: Column(
          mainAxisAlignment: MainAxisAlignment.min,
          children: [
            ListTile(
              title: const Text('Item 1'),
              onTap: controller.hide,
            ),
            ListTile(
              title: const Text('Item 2'),
              onTap: controller.hide,
            ),
            ListTile(
              title: const Text('Item 3'),
              onTap: controller.hide,
            ),
          ],
        ),
      ),
    ),
  );
```

```
}
```

- **Target:** The widget the popup is positioned relative to. When pressed, `controller.show` is called to display the popup.
- **Follower:** The actual popup widget positioned relative to the target.
- **Follower/Target Anchor:** Anchors that bind the target and follower to position the popup correctly.

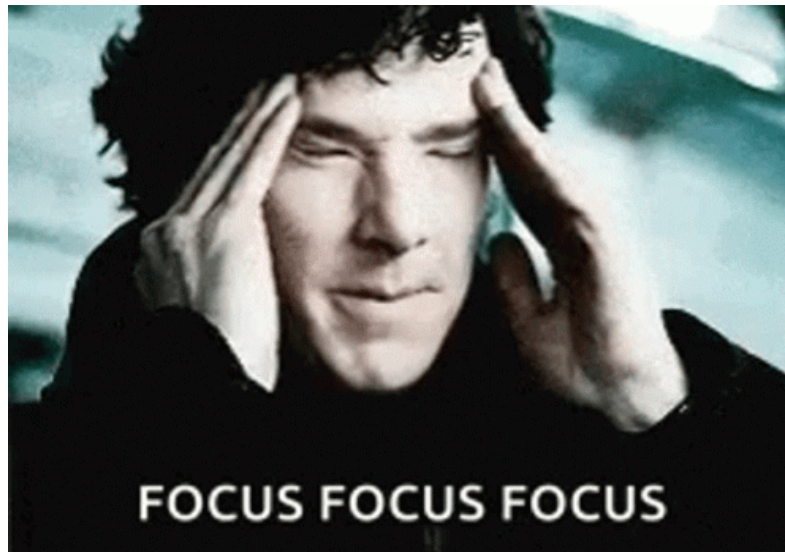
0:00 / 0:04

1×

The Popup Widget's minimalist style and high level of customization are its best features, as it doesn't have any hard-coded behavior. You can decide how it should work, it's limitations and possibilities.

## Making Popups Focusable and Dismissible





To enhance the popup's functionality, we can wrap it with necessary widgets. For example, to make the popup dismissible when tapping outside of it, use Flutter's `TapRegion` widget:

```
class _BasicPopup extends StatelessWidget {
  const _BasicPopup();

  @override
  Widget build(BuildContext context) => Popup(
    child: (context, controller) => FilledButton(
      onPressed: () => controller.show(),
      child: const Text('Show Simple Popup'),
    ),
    // Added TapRegion
    follower: (context, controller) => TapRegion(
      onTapOutside: (_) => controller.hide(),
      child: SizedBox(
        width: 200,
        child: Card(
          margin: EdgeInsets.zero,
          child: Column(
            mainAxisAlignment: MainAxisAlignment.min,
            children: [...],
          ),
        ),
      ),
    ),
  );
}
```

For accessibility, including tab traversals and screen readers, use `FocusScope` for managing focus and `Shortcuts` and `Actions` for handling keyboard shortcuts like dismissing a popup when pressing `Esc`:

## Sign up for Michael Lazebny

No spam. Unsubscribe anytime.

```
class _BasicPopup extends StatelessWidget {
  const _BasicPopup();

  @override
  Widget build(BuildContext context) => Popup(
    child: (context, controller) => FilledButton(
      onPressed: () => controller.show(),
      child: const Text('Show Simple Popup'),
    ),
    // Added Shortcuts
    follower: (context, controller) => Shortcuts(
      shortcuts: {
        LogicalKeySet(LogicalKeyboardKey.escape): const DismissInt
      },
      // Added Actions
      child: Actions(
        actions: {
          DismissIntent: CallbackAction<DismissIntent>(
            onInvoke: (_) => controller.hide(),
          ),
        },
      ),
      // Added FocusScope
      child: FocusScope(
        autofocus: true,
        child: TextButton(
```

```

child: TapRegion(
  onTapOutside: (_) => controller.hide(),
  child: SizedBox(
    width: 200,
    child: Card(
      margin: EdgeInsets.zero,
      child: Column(
        mainAxisAlignment: MainAxisAlignment.min,
        children: [],
      ),
    ),
  ),
),
),
),
),
),
),
),
);
}

```

This is how it behaves now:

0:00 / 0:09

1×

To avoid writing all this stuff every time you create a new popup, I've developed a widget

called `PopupFollower` that includes these features and a few more (optional dismissing on screen resize and on scrolling).

`popup_playground/lib/widgets/popup.dart` at main · hawkkiller/popup\_playground

A Flutter project that demonstrates the capabilities of Flutter SDK to build different variations of popups. - hawkkiller/popup\_playground

GitHub · hawkkiller

hawkkiller/  
popup\_playground

Flutter project that demonstrates the capabilities of Flutter SDK to build different variations of popups.

Contributor 0 Issues 11 Stars 4 Forks

## How does it work

Under the hood, `Popup` relies on three core widgets: `OverlayPortal` and patched versions of `CompositedTransformTarget` and `CompositedTransformFollower`.

- **OverlayPortal:** Renders the popup above other widgets.
- **CompositedTransformTarget & CompositedTransformFollower:** These widgets position the popup relative to a target. The patched versions also manage overflow, ensuring the popup is flipped or moved if it goes beyond the screen, allowing it to appear at full size.

## Creating reusable popups







While you can use the Popup widget directly, creating reusable widgets that use Popup under the hood is recommended. Here are a few general-purpose popups:

## Dropdown

The dropdown widget displays a list of options near the anchor, appearing below it if there's enough space, otherwise above it. Here's a simple implementation using the Popup widget:

```
final class CustomDropdownEntry<T> {
  final T value;
  final String label;

  const CustomDropdownEntry(this.value, this.label);
}

/// Dropdown implementation using [Popup]
class CustomDropdown<T> extends StatelessWidget {
  const CustomDropdown({
    required this.items,
    required this.activeItem,
    required this.onChanged,
    super.key,
  });

  final List<CustomDropdownEntry<T>> items;
  final CustomDropdownEntry<T>? activeItem;
  final ValueChanged<CustomDropdownEntry<T>>? onChanged;

  @override
  Widget build(BuildContext context) => Popup(
    child: (context, controller) => TapRegion(
      debugLabel: 'CustomDropdown',
      groupId: controller,
      child: Material(
        shape: RoundedRectangleBorder(
          borderRadius: const BorderRadius.all(Radius.circular(8))
          side: BorderSide(color: Theme.of(context).colorScheme.ou
        ),
      ),
    ),
  );
```

`CustomDropdown` is a straightforward, 70-line dropdown implementation. It takes a list of

items and displays them in a column. The `activeItem` parameter indicates the selected item.

## Sign up for Michael Lazebny

Subscribe

No spam. Unsubscribe anytime.

### Key features:

- Sets autofocus on `ListTile` so the current item (or the first item if none are active) is focused when the popup opens.
- Uses `SingleChildScrollView` to make the list scrollable if it exceeds screen boundaries.
- Automatically flips and moves by offset if popup overflows screen (this is provided by `Popup` widget itself).

The following code uses `CustomDropdown` to display 5 options to choose from:

```
class _CustomDropdown extends StatefulWidget {  
  const _CustomDropdown();  
  
  @override  
  State<_CustomDropdown> createState() => _CustomDropdownState();  
}  
  
class _CustomDropdownState extends State<_CustomDropdown> {  
  static final items = List.generate(  
    5,  
    (index) => CustomDropdownEntry(index, 'Item $index'),  
  );  
  
  CustomDropdownEntry? value;  
  
  @override
```

```

@override
Widget build(BuildContext context) => CustomDropdown(
  items: items,
  activeItem: value,
  onChanged: (entry) => setState(() => value = entry),
);
}

```

This is how it looks like:

0:00 / 0:07

1x

## Animated tooltip

Another widget I want to parse is the tooltip. Typically, it presents a small widget that provides more information about a specific aspect. For example, it may appear when you hover over a button to explain its function.

What's interesting about this tooltip is its animation. It shows and hides with an opacity transition. Take a look at its implementation:

```

/// A custom tooltip widget that can be used to display a tooltip with
class CustomTooltip extends StatefulWidget {
  const CustomTooltip({
    required this.content,
    required this.child,
    ...other
  }) : super(key: key);
}

```

```

        this.animationDuration,
        super.key,
    });

    final String content;
    final Widget child;
    final Duration? animationDuration;

    @override
    State<CustomTooltip> createState() => _CustomTooltipState();
}

class _CustomTooltipState extends State<CustomTooltip> with SingleTick
    final overlayController = OverlayPortalController(debugLabel: 'Custo
    late final AnimationController _animationController;

    Duration get _animationDuration => widget.animationDuration ?? Durat

    @override
    void initState() {
        _animationController = AnimationController(vsync: this, duration:
        super.initState();
    }

    @override
    void didUpdateWidget(covariant CustomTooltip oldWidget) {
        if (widget.animationDuration != oldWidget.animationDuration) {
            _animationController.duration = _animationDuration;
        }
        super.didUpdateWidget(oldWidget);
    }

    @override
    void dispose() {
        _animationController.dispose();
        super.dispose();
    }

    /// Shows popup and sets a timer to hide it after [duration] if it i
    void _showPopup([Duration? duration]) {
        overlayController.show();
        _animationController.forward();
        SemanticsService.tooltip(widget.content);

        if (duration != null) {
            Future.delayed(duration, _hidePopup).ignore();
        }
    }

```

```

    }
  }

  void _hidePopup() {
    _animationController.reverse().whenComplete(overlayController.hide
  }

  void _togglePopup([Duration? duration]) {
    overlayController.isShowing ? _hidePopup() : _showPopup(duration);
  }

  Widget _buildMobileTooltip(Widget child) => GestureDetector(
    onTap: () => _togglePopup(_animationDuration + const Duration(
      child: child,
    );

  Widget _buildDesktopTooltip(Widget child) => MouseRegion(
    onEnter: (_) => _showPopup(),
    onExit: (_) => _hidePopup(),
    child: child,
  );

  @override
  Widget build(BuildContext context) => Popup(
    controller: overlayController,
    follower: (context, controller) => PopupFollower(
      tapRegionGroupId: controller,
      onDismiss: _hidePopup,
      child: FadeTransition(
        opacity: _animationController,
        child: ConstrainedBox(
          constraints: const BoxConstraints(maxWidth: 200),
          child: Card(
            child: Padding(
              padding: const EdgeInsets.all(8),
              child: Text(widget.content),
            ),
          ),
        ),
      ),
    ),
  ),
  child: (context, controller) {
    final platform = Theme.of(context).platform;
    Widget result;

    switch (platform) {

```

```
    case TargetPlatform.android:
    case TargetPlatform.iOS:
      result = _buildMobileTooltip(widget.child);
      break;
    case TargetPlatform.fuchsia:
    case TargetPlatform.linux:
    case TargetPlatform.macOS:
    case TargetPlatform.windows:
      result = _buildDesktopTooltip(widget.child);
      break;
  }

  return Semantics(
    tooltip: widget.content,
    child: TapRegion(
      groupId: controller,
      child: result,
    ),
  );
},
);
}
```

CustomTooltip takes a string content that becomes visible when hovering over the anchor. It also handles semantics to ensure screen readers display the correct information, enhancing accessibility.

**Key points:**

- Uses an animation controller to show and hide the popup with an animation. The `controller.hide` is called when the animation completes.
- Builds different tooltips for mobile and desktop. On mobile, without a mouse, the popup opens with a click and closes automatically after a few seconds.

## Built-in widgets

Out of the box, Flutter provides a few widgets for creating anchored popups. For example, there is a `DropDownButton`, a `DropDownMenu`, a `MenuAnchor` and more. In this section, I will give a brief overview of most applicable ones and what you can do with them.

# Sign up for Michael Lazebny

Subscribe

No spam. Unsubscribe anytime.

## MenuAnchor

A widget used to mark the "anchor" for a set of submenus, defining the rectangle used to position the menu, which can be done either with an explicit location, or with an alignment.

[MenuAnchor](#) is a low-level widget used to show anchored popups. You can use any items, decorations, and styles for it, so that the menu is quite customizable. Here's the API:

```
MenuAnchor(  
  menuChildren: [...]
```



```

menuChildren: [
  ...widgets
],
builder: (context, controller, child) => FilledButton.icon(
  onPressed: controller.open,
  icon: const Icon(Icons.menu),
  label: const Text('Show Standard Menu'),
),
);

```

`MenuAnchor` uses `menuChildren` to show as options when the menu is visible. See examples of menus built with this widget:

0:00 / 0:06

1x

`MenuAnchor` is great for creating menus, submenus and dropdowns. It automatically positions the popup correctly against the target and adjusts the offset in case of overflow, so that the popup will never end up outside the borders of the screen.

**Dropdown**

## Dropdowns

There are two dropdowns in the material library: `DropDownButton` and `DropDownMenu`. One is for material 2 and the other is for material 3.

### DropDownButton

In Material 2, the `DropDownButton` is commonly used to create dropdown menus. It provides a straightforward way to present a list of options that the user can select from. To illustrate, consider the following examples:

0:00 / 0:05

1×

The dropdown button makes it easy to select from a list, but it doesn't offer much in the way of customization options. It might not be the best choice for designs that differ significantly from material 2.

### DropDownMenu

In Material 3, the `DropDownButton` has been replaced with a modern implementation - `DropDownMenu`, which uses `MenuAnchor` under the hood and renders a dropdown anchored to a text field that may optionally provide filtering and search functionalities. See

anchored to a text field that may optionally provide filtering and search functionalities. See how it looks like:

0:00 / 0:14

1x

This widget offers a superior UI/UX compared to the `DropDownButton` and includes enhanced functionalities. A text field for search or filtering greatly improves user experience. It can be configured to allow custom entries or restricted to predefined items, providing flexible options.

## Conclusion

Popups are a key part of modern UI design. This guide has shown you how to create popups in Flutter, with a focus on proper positioning, responsiveness, and accessibility. By following these tips, you can build popups that enhance your app's usability. Happy coding!

# Sign up for Michael Lazebny

Subscribe

No spam. Unsubscribe anytime.



READ MORE

## Sticky headers in Flutter

If you've landed here, you likely already know what sticky headers are. But if not,

By Michael Lazebny — 23 Nov 2024

## Navigation in Flutter

This article provides insights into how Navigator and router-based packages work

By Michael Lazebny — 19 Nov 2024

## Authentication in Flutter with Supabase (OAuth incl

I've been working on a Flutter while and recently decided to

# Michael Lazebny

jamie@example.com

Subscribe