Drawer in Flutter

In apps that use Material Design, there are two primary options for navigation: tabs and drawers. When there is insufficient space to support tabs, drawers provide a handy alternative.

In Flutter, use the `Drawer` widget in combination with a `Scaffold` to create a layout with a Material Design drawer. This recipe uses the following steps:

1. Create a `Scaffold`.
2. Add a drawer.
3. Populate the drawer with items.
4. Close the drawer programmatically.

1. Create a `Scaffold`

To add a drawer to the app, wrap it in a `Scaffold` widget. The `Scaffold` widget provides a consistent visual structure to apps that follow the Material Design Guidelines. It also supports special Material Design components, such as Drawers, AppBars, and SnackBars.

In this example, create a `Scaffold` with a `drawer`:

```
Scaffold(
  appBar: AppBar(
    title: const Text('AppBar without hamburger button'),
  ),
  drawer: // Add a Drawer here in the next step.
);
```

2. Add a drawer

Now add a drawer to the `Scaffold`. A drawer can be any widget, but it's often best to use the `Drawer` widget from the material library, which adheres to the Material Design spec.

```
Scaffold(
  appBar: AppBar(
    title: const Text('AppBar with hamburger button'),
  ),
  drawer: Drawer(
    child: // Populate the Drawer in the next step.
  ),
);
```

3. Populate the drawer with items

Now that you have a `Drawer` in place, add content to it. For this example, use a `ListView`. While you could use a `Column` widget, `ListView` is handy because it allows users to scroll through the drawer if the content takes more space than the screen supports.

Populate the `ListView` with a `DrawerHeader` and two `ListTile` widgets. For more information on working with Lists, see the list recipes.

```
Drawer(
  // Add a ListView to the drawer. This ensures the user can scroll
  // through the options in the drawer if there isn't enough vertical
  // space to fit everything.
  child: ListView(
    padding: EdgeInsets.zero,
    children: [
      const DrawerHeader(
        decoration: BoxDecoration(
          color: Colors.blue,
        ),
        child: Text('Drawer Header'),
      ),
      ListTile(
        title: const Text('Item 1'),
        onTap: () {
          // Update the state of the app.
        },
      ),
      ListTile(
        title: const Text('Item 2'),
        onTap: () {
          // Update the state of the app.
        },
      ),
    ],
  ),
);
```

4. Close the drawer programmatically

After a user taps an item, you might want to close the drawer. You can do this by using the `Navigator`.

When a user opens the drawer, Flutter adds the drawer to the navigation stack. Therefore, to close the drawer, call `Navigator.pop(context)`.

```
ListTile(
  title: const Text('Item 1'),
  onTap: () {
    // Update the state of the app
    // ...
    // Then close the drawer
    Navigator.pop(context);
  },
),
```

Conclusion: we can treat drawer in Flutter Apps as a side menu that can have some options to help users navigate to the required page easily.

Reference:

https://docs.flutter.dev/cookbook/design/drawer