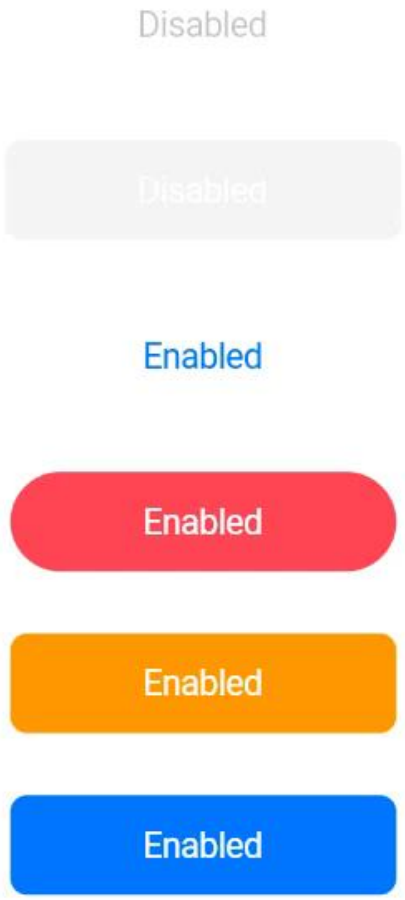
Cupertino (iOS-style) widgets

Cupertino library

- flutter.dev/widgets/cupertino for a catalog of all Cupertino widgets.
- flutter.dev/widgets for a catalog of commonly-used Flutter widgets.

```
import 'package:flutter/cupertino.dart';
```

Button




```
const CupertinoButton(
  onPressed: null,
  child: Text('Disabled'),
), // CupertinoButton
const SizedBox(height: 30),
const CupertinoButton.filled(
  onPressed: null,
  child: Text('Disabled'),
), // CupertinoButton.filled
const SizedBox(height: 30),
CupertinoButton(
  onPressed: () {},
  child: const Text('Enabled'),
), // CupertinoButton
const SizedBox(height: 30),
CupertinoButton(
  color: CupertinoColors.systemPink,
  pressedOpacity: 0,
  onPressed: () {},
  borderRadius: BorderRadius.all(Radius.circular(55)),
  child: const Text('Enabled'),
), // CupertinoButton
const SizedBox(height: 30),
CupertinoButton(
  pressedOpacity: 0.5,
  color: CupertinoColors.activeOrange,
  onPressed: () {},
  child: const Text('Enabled'),
), // CupertinoButton
const SizedBox(height: 30),
CupertinoButton.filled(
  onPressed: () {},
  pressedOpacity: 1,
  child: const Text('Enabled'),
```

Button properties

borderRadius → BorderRadius?
The radius of the button's corners when it has a background color.
final

color → Color?
The color of the button's background.
final

disabledColor → Color
The color of the button's background when the button is disabled.
final

onPressed → VoidCallback?
The callback that is called when the button is tapped or otherwise activated.
final

pressedOpacity → double?
The opacity that the button will fade to when it is pressed.
final

Switch



```dart
class _CupertinoSwitchExampleState extends State<CupertinoSwitchExample> {
  bool switchValue = true;

  @override
  Widget build(BuildContext context) {
    return CupertinoPageScaffold(
      navigationBar: const CupertinoNavigationBar(
        middle: Text('CupertinoSwitch Sample'),
      ),  // CupertinoNavigationBar
      child: Center(
        child: CupertinoSwitch(
          // This bool value toggles the switch.
          value: switchValue,
          activeColor: CupertinoColors.activeBlue,
          onChanged: (bool? value) {
            // This is called when the user toggles the switch.
            setState(() {
              switchValue = value ?? false;
            });
          },
        ),  // CupertinoSwitch
      ),  // Center
    );  // CupertinoPageScaffold
  }
}
```

Switch properties

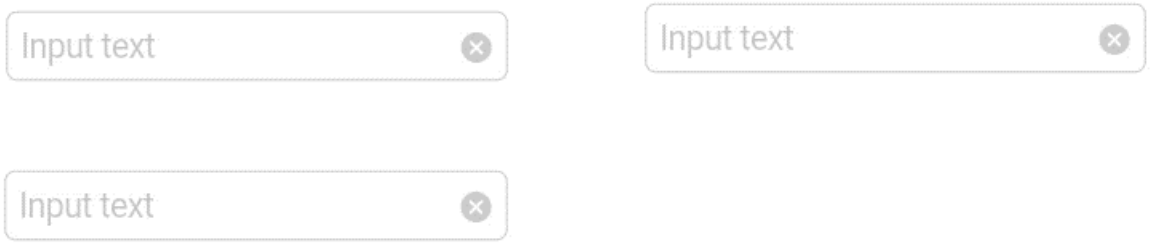
activeColor → Color?
The color to use when this switch is on.
final

dragStartBehavior → DragStartBehavior
Determines the way that drag start behavior is handled.
final

thumbColor → Color?
The color to use for the thumb of the switch.
final

trackColor → Color?
The color to use for the background when the switch is off.
final

TextField




```dart
@override
Widget build(BuildContext context) {
  return CupertinoPageScaffold(
    navigationBar: const CupertinoNavigationBar(
      middle: Text('CupertinoTextField Sample'),
    ),  // CupertinoNavigationBar
    child: Center(
      child: SizedBox(
        width: 250,
        child: CupertinoTextField(
          obscureText: true,
          obscuringCharacter: "*",
          cursorColor: CupertinoColors.destructiveRed,
          placeholder: "Input text",
          controller: _textController,
          clearButtonMode: OverlayVisibilityMode.always,
        )  // CupertinoTextField
      )  // SizedBox
      ),  // Center
  );  // CupertinoPageScaffold
}
```

Textfield properties

autocorrect → bool
Whether to enable autocorrection.
final

autofillHints → Iterable<String>?
A list of strings that helps the autofill service identify the type of this text input.
final

autofocus → bool
Whether this text field should focus itself if nothing else is already focused.
final

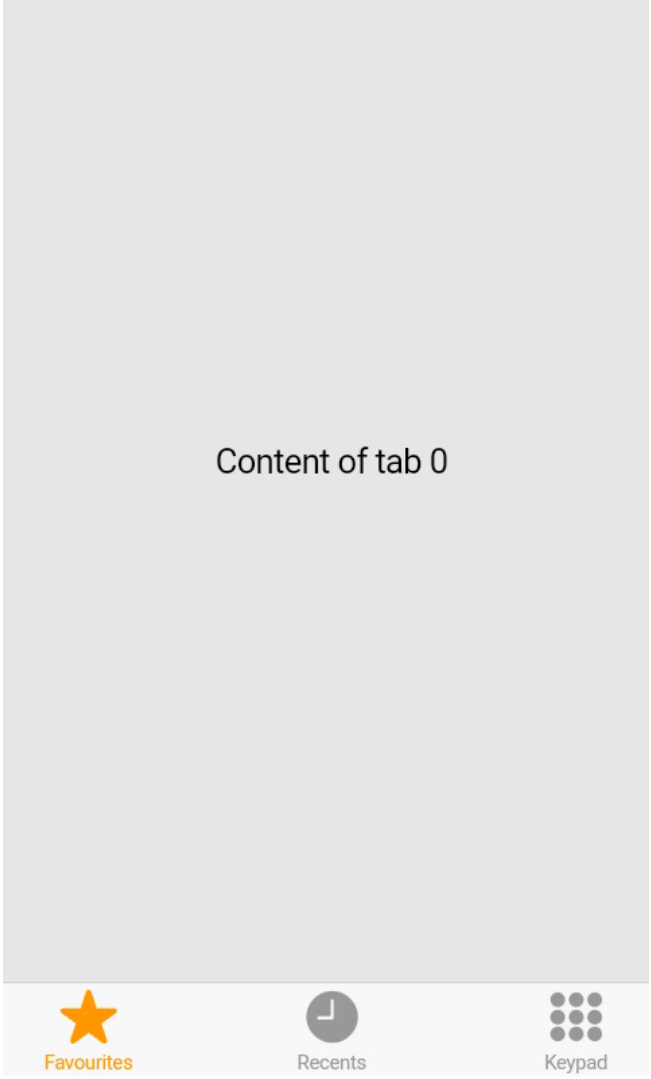
clearButtonMode → OverlayVisibilityMode
Show an iOS-style clear button to clear the current text entry.
final

obscureText → bool
Whether to hide the text being edited (e.g., for passwords).
final

obscuringCharacter → String
Character used for obscuring text if obscureText is true.
final

placeholder → String?
A lighter colored placeholder hint that appears on the first line of the text field when the text entry is empty.
final

TabBar




```dart
class CupertinoTabBarExample extends StatelessWidget {
  const CupertinoTabBarExample({super.key});

  @override
  Widget build(BuildContext context) {
    return CupertinoTabScaffold(
      tabBar: CupertinoTabBar(
        activeColor: CupertinoColors.activeOrange,
        items: const <BottomNavigationBarItem>[
          BottomNavigationBarItem(
            icon: Icon(CupertinoIcons.star_fill),
            label: 'Favourites',
          ),  // BottomNavigationBarItem
          BottomNavigationBarItem(
            icon: Icon(CupertinoIcons.clock_solid),
            label: 'Recents',
          ),  // BottomNavigationBarItem
          BottomNavigationBarItem(
            icon: Icon(CupertinoIcons.circle_grid_3x3_fill),
            label: 'Keypad',
          ),  // BottomNavigationBarItem
        ],  // <BottomNavigationBarItem>[]
      ),  // CupertinoTabBar
      tabBuilder: (BuildContext context, int index) {
        return CupertinoTabView(
          builder: (BuildContext context) {
            return Center(
              child: Text('Content of tab $index'),
            );  // Center
          },
        );  // CupertinoTabView
      },
    );  // CupertinoTabScaffold
  }
}
```

TabBar properties

activeColor → Color?
The foreground color of the icon and title for the BottomNavigationBarItem of the selected tab.
final

backgroundColor → Color?
The background color of the tab bar. If it contains transparency, the tab bar will automatically produce a blurring effect to the content behind it.
final

iconSize → double
The size of all of the BottomNavigationBarItem icons.
final

inactiveColor → Color
The foreground color of the icon and title for the BottomNavigationBarItems in the unselected state.
final

AlertDialog

CupertinoAlertDialog

```dart
class AlertDialogExample extends StatelessWidget {
  const AlertDialogExample({super.key});

  // This shows a CupertinoModalPopup which hosts a CupertinoAlertDialog.
  void _showAlertDialog(BuildContext context) {
    showCupertinoModalPopup<void>(
      context: context,
      builder: (BuildContext context) => CupertinoAlertDialog(
        title: const Text('Alert'),

        content: const Text('Proceed with destructive action?'),
        actions: <CupertinoDialogAction>[
          CupertinoDialogAction(
            /// This parameter indicates this action is the default,
            /// and turns the action's text to bold text.
            isDefaultAction: true,
            onPressed: () {
              Navigator.pop(context);
            },
            child: const Text('No'),
          ),  // CupertinoDialogAction
          CupertinoDialogAction(
            /// This parameter indicates the action would perform
            /// a destructive action such as deletion, and turns
            /// the action's text color to red.
            isDestructiveAction: true,
            onPressed: () {
              Navigator.pop(context);
            },
            child: const Text('Yes'),
          ),  // CupertinoDialogAction
        ],  // <CupertinoDialogAction>[]
      ),  // CupertinoAlertDialog
    );
  }
```

ActivityIndicator

Default indicator

Colored indicator

Partially revealed indicator

Non animated indicator

```dart
class CupertinoIndicatorExample extends StatelessWidget {
  const CupertinoIndicatorExample({super.key});

  @override
  Widget build(BuildContext context) {
    return CupertinoPageScaffold(
      navigationBar: const CupertinoNavigationBar(
        middle: Text('CupertinoActivityIndicator Sample'),
      ),  // CupertinoNavigationBar
      child: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          children: <Widget>[
            Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: const <Widget>[
                // Cupertino activity indicator with default properties.
                CupertinoActivityIndicator(),
                SizedBox(height: 10),
                Text('Default indicator'),
              ],  // <Widget>[]
            ),  // Column
            Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: const <Widget>[
                // Cupertino activity indicator with custom radius and color.
                CupertinoActivityIndicator(
                    radius: 20.0,
                    color: CupertinoColors.activeBlue,),  // CupertinoActivityIndicator
                SizedBox(height: 10),
                Text('Colored indicator')
              ],  // <Widget>[]
            ),  // Column
            Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: const <Widget>[
                // Cupertino activity indicator with custom radius and color.
                CupertinoActivityIndicator.partiallyRevealed(
                  radius: 20.0,
                  progress: 0.7,
                  color: CupertinoColors.destructiveRed,),  // CupertinoActivityIndicator.partiallyRevealed
                SizedBox(height: 10),
                Text('Partially revealed indicator'),
              ],  // <Widget>[]
            ),  // Column
            Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: const <Widget>[
                // Cupertino activity indicator with custom radius and disabled
                // animation.
                CupertinoActivityIndicator(radius: 20.0, animating: false),
                SizedBox(height: 10),
                Text('Non animated indicator'),
              ],  // <Widget>[]
            ),  // Column
          ],  // <Widget>[]
        ),  // Column
      );  // Center, CupertinoPageScaffold
  }
```

Activityindicator properties

animating → bool

Whether the activity indicator is running its animation.

final

progress → double

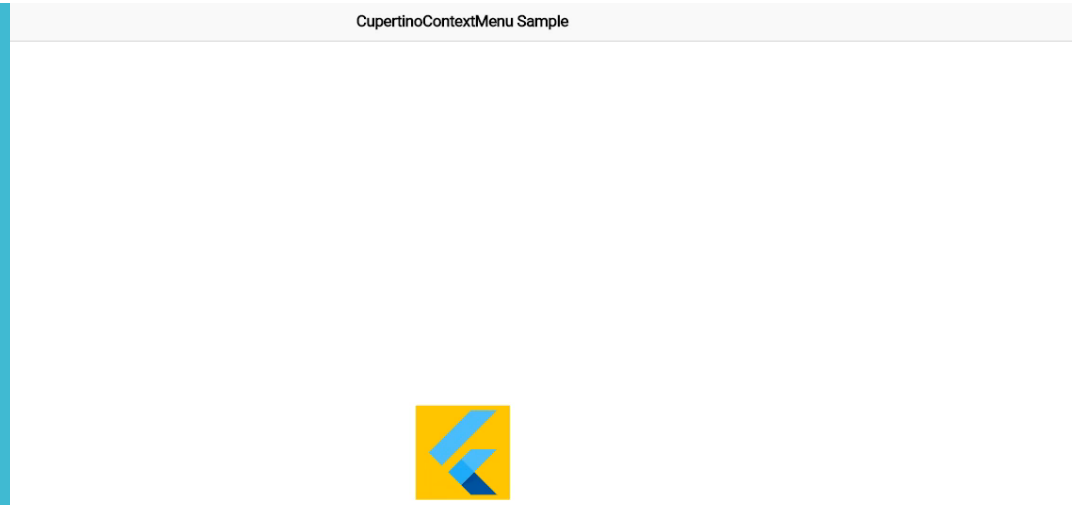
Determines the percentage of spinner ticks that will be shown.

radius → double

Radius of the spinner widget.

final

ContextMenu

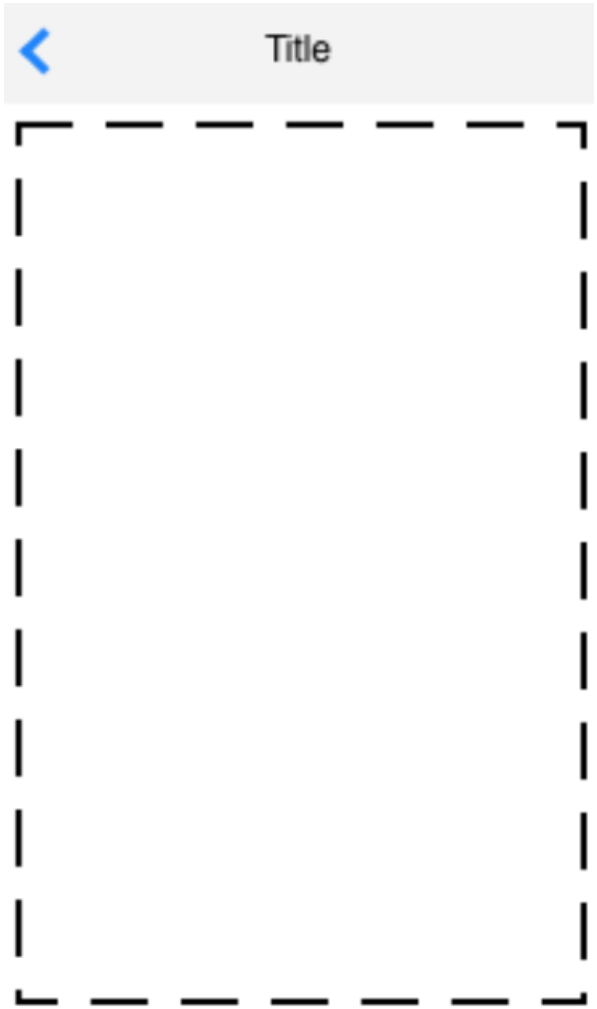



```dart
@override
Widget build(BuildContext context) {
  return CupertinoPageScaffold(
    navigationBar: const CupertinoNavigationBar(
      middle: Text('CupertinoContextMenu Sample'),
    ),  // CupertinoNavigationBar
    child: Center(
      child: SizedBox(
        width: 100,
        height: 100,
        child: CupertinoContextMenu(
          actions: <Widget>[
            CupertinoContextMenuAction(
              onPressed: () {
                Navigator.pop(context);
              },
              isDefaultAction: true,
              trailingIcon: CupertinoIcons.doc_on_clipboard_fill,
              child: const Text('Copy'),
            ),  // CupertinoContextMenuAction
            CupertinoContextMenuAction(
              onPressed: () {
                Navigator.pop(context);
              },
              trailingIcon: CupertinoIcons.share,
              child: const Text('Share'),
            ),  // CupertinoContextMenuAction
            CupertinoContextMenuAction(
              onPressed: () {
                Navigator.pop(context);
              },
              trailingIcon: CupertinoIcons.heart,
              child: const Text('Favorite'),
            ),  // CupertinoContextMenuAction
            CupertinoContextMenuAction(
              onPressed: () {
                Navigator.pop(context);
              },
              isDestructiveAction: true,
              trailingIcon: CupertinoIcons.delete,
              child: const Text('Delete'),
            ),  // CupertinoContextMenuAction
          ],  // <Widget>[]
          child: Container(
            color: CupertinoColors.systemYellow,
            child: const FlutterLogo(size: 500.0),
          ),  // Container
```

Cupertino PageScaffold

- CupertinoPageScaffold - это виджет в фреймворке Flutter, который представляет собой стандартный макет страницы в стиле Cupertino (iOS). Он обеспечивает базовый макет страницы, который состоит из:
  - верхней навигационной панели, которая может содержать кнопки возврата, заголовок и другие элементы управления;
  - основной области контента, где отображается основное содержимое страницы;
  - нижней навигационной панели, которая может содержать кнопки навигации или другие элементы управления.
- CupertinoPageScaffold также обеспечивает стандартную анимацию перехода между страницами, которая соответствует стилю iOS.

```dart
class MyPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return CupertinoPageScaffold(
      navigationBar: CupertinoNavigationBar(
        middle: Text('My Page'),
      ),
      backgroundColor: CupertinoColors.systemGrey6,
      child: Center(
        child: Text('Hello, world!'),
      ),
    );
  }
}
```



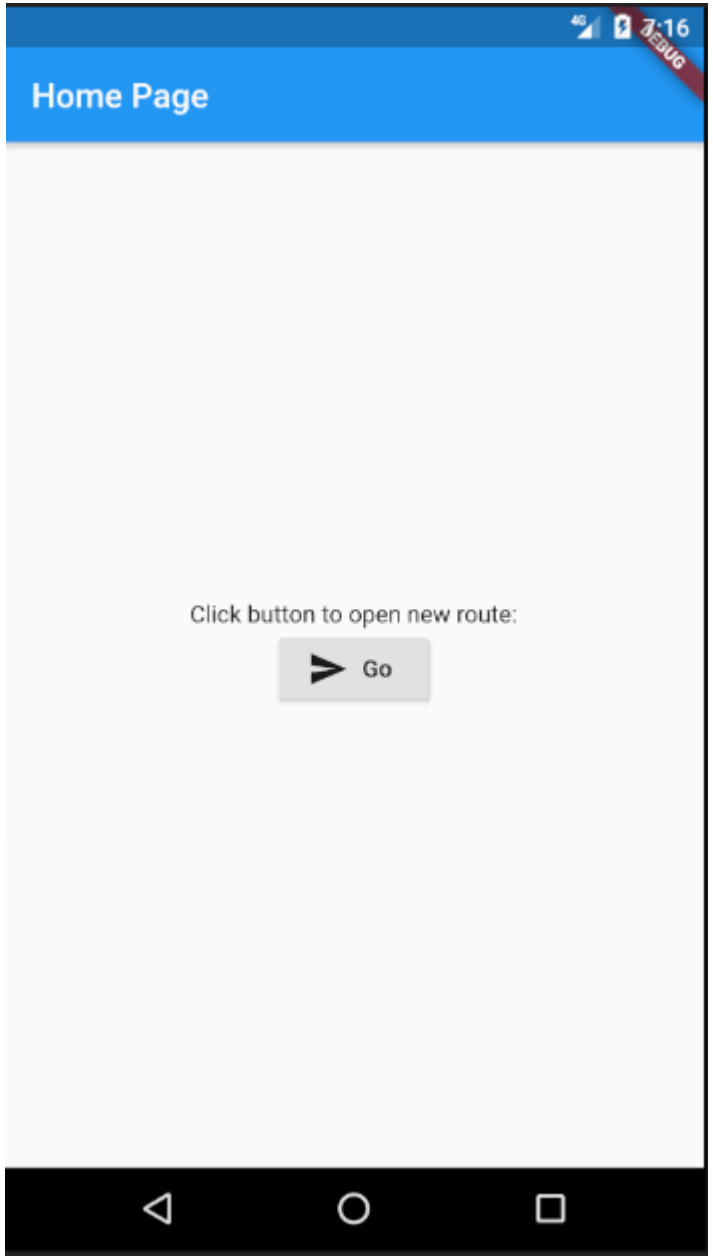
Cupertino Fullscreen Dialog Transition

- CupertinoFullscreenDialogTransition - это вид анимации, который используется в iOS-приложениях для создания перехода между экранами в полноэкранном режиме. Он используется вместе с CupertinoPageRoute, который отображает страницу в полноэкранном режиме и создает переход с помощью анимации, когда пользователь нажимает кнопку или свайпает экран.
- CupertinoFullscreenDialogTransition может использоваться для создания более плавного и естественного перехода между страницами в приложении, что может сделать пользовательский интерфейс более привлекательным и интуитивно понятным для пользователей iOS.

- Основные свойства этой анимации включают:
  - **primaryRouteAnimation**: Анимация, используемая для анимации исходной страницы. Эта анимация происходит одновременно с анимацией появления новой страницы.
  - **secondaryRouteAnimation**: Анимация, используемая для анимации появления новой страницы. Эта анимация происходит одновременно с анимацией исходной страницы.
  - **linearTransition**: Определяет, должна ли анимация перемещения страницы по экрану происходить линейно или с использованием кривой Безье.
  - **animationDuration**: Время, необходимое для завершения анимации перехода.
  - **barrierDismissible**: Определяет, может ли пользователь закрыть модальное окно, щелкнув вне его.
  - **barrierLabel**: Текст, который будет использоваться для пометки экрана, который блокирует пользовательский ввод.
  - **secondaryAnimation**: Анимация, используемая для анимации элементов на новой странице. Эта анимация начинается после завершения анимации появления новой страницы.

```
class FirstScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('First Screen'),
      ),
      body: Center(
        child: ElevatedButton(
          child: Text('Go to Second Screen'),
          onPressed: () {
            Navigator.push(
              context,
              CupertinoPageRoute(
                fullscreenDialog: true,
                builder: (context) =>
CupertinoFullscreenDialogTransition(
                  linearTransition: true,
                  primaryRouteAnimation:
AlwaysStoppedAnimation(0),
                  secondaryRouteAnimation:
AlwaysStoppedAnimation(1),
                  child: SecondScreen(),
                ),),);},),),);
  }
}
```

```
class SecondScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Second Screen'),
      ),
      body: Center(
        child: ElevatedButton(
          child: Text('Go back to First Screen'),
          onPressed: () {
            Navigator.pop(context);
          },),),);}
}
```

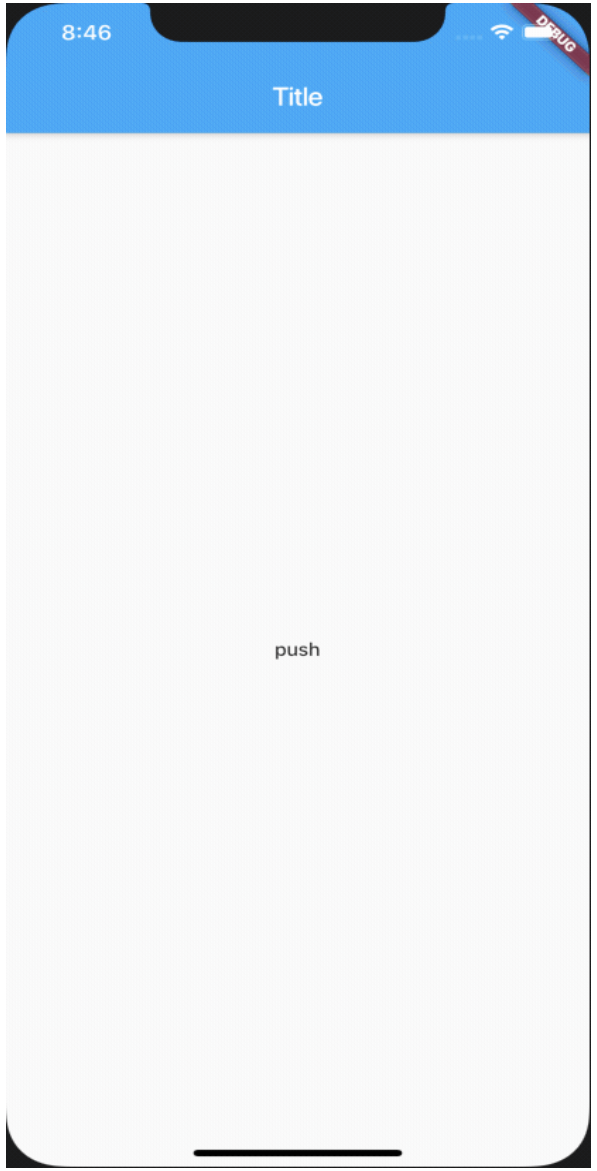


Cupertino Page Transition

- CupertinoPageTransition - это виджет в библиотеке Flutter, который позволяет создавать анимированный переход между двумя экранами (страницами) в iOS-стиле. Он используется вместе с CupertinoPageRoute, который является специальным маршрутом в Flutter для iOS-стиля страниц.
- CupertinoPageTransition предоставляет несколько свойств для настройки анимации перехода:
  - **primaryRouteAnimation**: анимация для нового экрана
  - **secondaryRouteAnimation**: анимация для старого экрана
  - **linearTransition**: переход анимации линейный или кривой
  - **child**: Виджет, который будет использоваться в качестве новой страницы
- Некоторые свойства могут быть настроены, чтобы получить различные эффекты анимации, например, вы можете изменить скорость или замедлить анимацию, чтобы добавить больше эффектов.

```dart
class MyHomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return CupertinoPageTransition(
      primaryRouteAnimation:
AlwaysStoppedAnimation(1),
      secondaryRouteAnimation:
AlwaysStoppedAnimation(0.5),
      linearTransition: true,
      child: Scaffold(
        appBar: AppBar(
          title: Text('Home Page'),
        ),
        body: Center(
          child: CupertinoButton(
            child: Text('Go to Next Page'),
            onPressed: () {
              Navigator.of(context).push(
                CupertinoPageRoute(builder: (context) =>
NextPage()),
              );},),),),);
  }
}
```

```dart
class NextPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return CupertinoPageTransition(
      primaryRouteAnimation:
AlwaysStoppedAnimation(0.5),
      secondaryRouteAnimation:
AlwaysStoppedAnimation(1),
      linearTransition: false,
      child: Scaffold(
        appBar: AppBar(
          title: Text('Next Page'),
        ),
        body: Center(
          child: CupertinoButton(
            child: Text('Go Back'),
            onPressed: () {
              Navigator.of(context).pop();
            },),),),);
  }
}
```



Cupertino NavigationBar

- CupertinoNavigationBar - это виджет, который используется в фреймворке Flutter для создания навигационного заголовка в стиле Cupertino (iOS).
- Он обеспечивает основные функции навигации, такие как добавление кнопки "назад", заголовка и других элементов управления, которые могут использоваться для управления навигацией в вашем приложении.
- Кроме того, CupertinoNavigationBar также позволяет настроить цвет и стиль фона, а также добавить дополнительные элементы управления, такие как кнопки действий, если это необходимо.

- Основные свойства CupertinoNavigationBar, которые могут быть настроены:
  - **leading**: виджет, отображаемый слева от заголовка, обычно используется для кнопки "назад".
  - **middle**: виджет, отображаемый в центре навигационного заголовка, обычно используется для отображения названия страницы.
  - **trailing**: виджет, отображаемый справа от заголовка, обычно используется для кнопок действий.
  - **backgroundColor**: цвет фона навигационного заголовка.
  - **border**: бордер, который окружает навигационный заголовок.

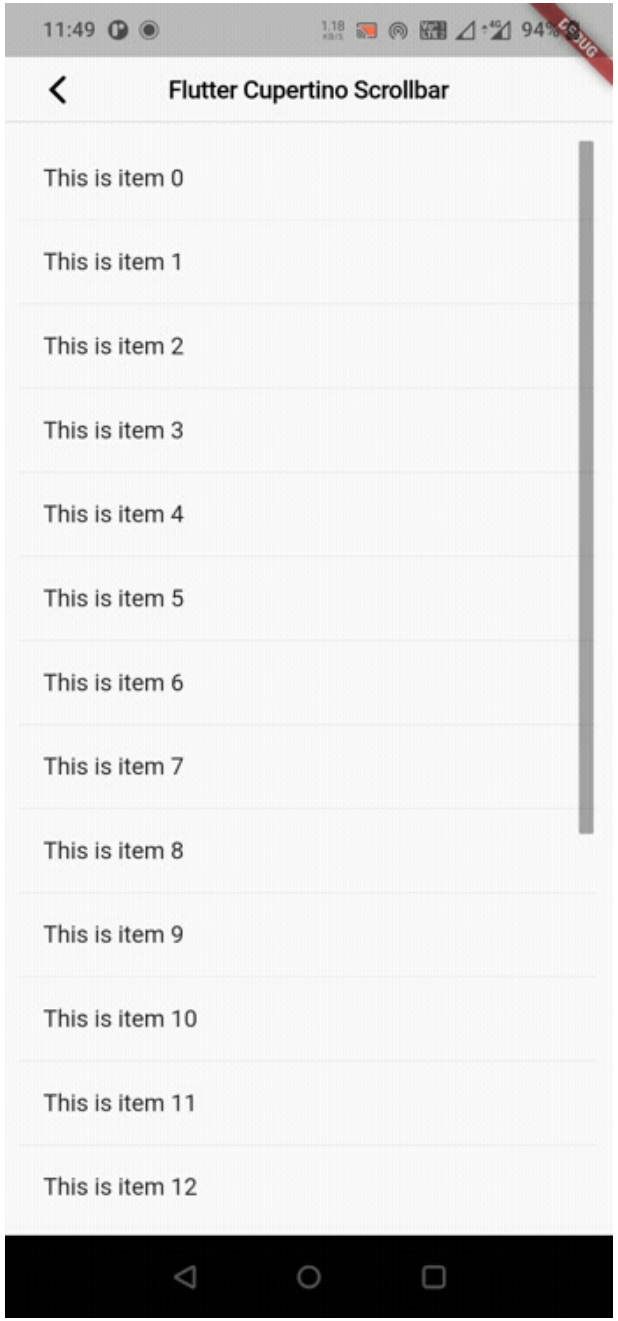
```dart
class MyHomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return CupertinoPageScaffold(
      navigationBar: CupertinoNavigationBar(
        middle: Text('My App'),
        leading: CupertinoButton(
          child: Icon(Icons.arrow_back),
          onPressed: () {
            Navigator.of(context).pop();
          },
        ),
        trailing: CupertinoButton(
          child: Icon(Icons.search),
          onPressed: () {
            // Implement search functionality here
          },
        ),
      ),
      child: Center(
        child: Text('Hello World!'),
      ),
    );
  }
}
```

Cupertino Scrollbar

- CupertinoScrollbar - это виджет, который представляет собой скроллбар для пользовательского интерфейса iOS в стиле Cupertino. Он обычно используется вместе с CupertinoScrollView, чтобы показать пользователю текущее положение прокрутки на экране.
- CupertinoScrollbar отображает вертикальный или горизонтальный скроллбар в зависимости от направления прокрутки содержимого. Скроллбар появляется только тогда, когда содержимое превышает доступную область просмотра, и исчезает, когда прокрутка закончена или содержимое умещается в область просмотра.
- Кроме того, CupertinoScrollbar также обеспечивает возможность управления скроллбаром с помощью жестов, таких как перетаскивание и касание. Когда пользователь тянет скроллбар, он обновляет положение содержимого в соответствии с текущим положением скроллбара.

- Основные свойства, которые можно использовать для настройки виджета CupertinoScrollbar:
  - **controller**: контроллер прокрутки, который управляет положением ползунка. Если вы хотите использовать собственный контроллер прокрутки, вы можете передать его в это свойство.
  - **thumbVisibility** : булевое значение, которое указывает, должен ли виджет полосы прокрутки всегда отображаться или скрываться, когда контент не прокручивается.
  - **thickness**: размер толщины полосы прокрутки.
  - **radius**: радиус закругления углов полосы прокрутки.
  - **semanticsLabel**: текст, который используется для оповещения пользователей с ограниченными возможностями о том, что этот виджет является полосой прокрутки.
  - **child**: виджет, который будет содержать полосу прокрутки.

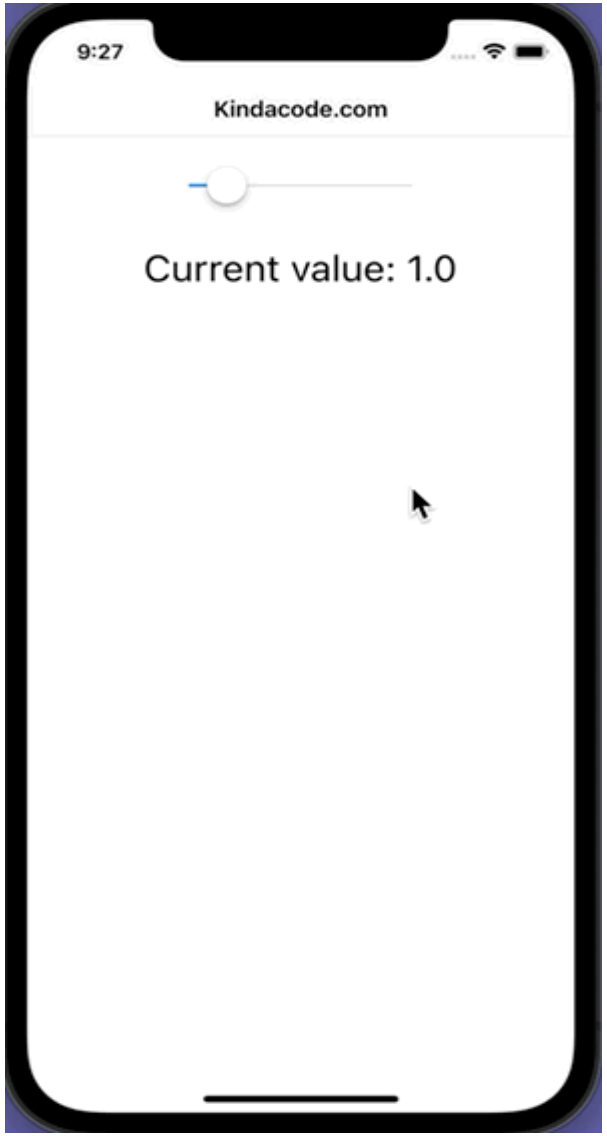
```dart
class _MyStatefulWidgetState extends State<MyStatefulWidget> {
 final ScrollController _firstController = ScrollController();
 @override
 Widget build(BuildContext context) {
  return LayoutBuilder(
     builder: (BuildContext context, BoxConstraints constraints) {
      return Row(
       children: <Widget>[
        SizedBox(
           width: constraints.maxWidth,
           child: Scrollbar(
            thumbVisibility: true,
            controller: _firstController,
            child: ListView.builder(
               controller: _firstController,
               itemCount: 20,
               itemBuilder: (BuildContext context, int index) {
                return Padding(
                  padding: const EdgeInsets.all(8.0),
                  child: Text('This is item $index'),
                );
               }),
           )),
       ],);});
 }
}
```



Cupertino Slider

- CupertinoSlider - это виджет ползунка в библиотеке Flutter. Он позволяет пользователю выбирать значение из диапазона, указанного при создании ползунка.
- CupertinoSlider имеет следующие параметры:
  - **value**: значение текущего положения ползунка, заданное в виде числа, которое может быть любым значением в диапазоне между минимальным и максимальным значениями;
  - **min**: минимальное значение, которое может принимать ползунок;
  - **max**: максимальное значение, которое может принимать ползунок;
  - **onChanged**: функция обратного вызова, которая вызывается при изменении положения ползунка пользователем;
  - **onChangeStart**: функция обратного вызова, которая вызывается при начале перемещения ползунка;
  - **onChangeEnd**: функция обратного вызова, которая вызывается при окончании перемещения ползунка.
- При перемещении ползунка пользователем, значение его положения передается в функцию обратного вызова onChanged, где вы можете обновить значение в соответствующем месте в вашем приложении.

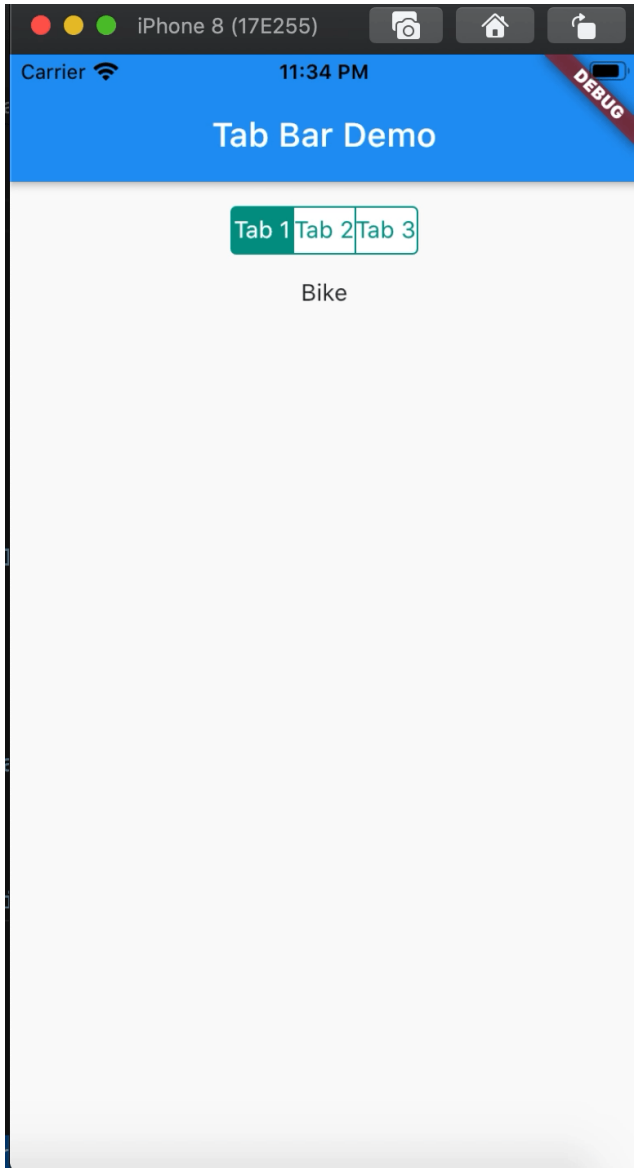
```dart
class MySlider extends StatefulWidget {
  @override
  _MySliderState createState() => _MySliderState();
}
class _MySliderState extends State<MySlider> {
  double _value = 5.0;
  @override
  Widget build(BuildContext context) {
    return CupertinoSlider(
      value: _value,
      min: 0.0,
      max: 10.0,
      onChanged: (newValue) {
        setState(() {
          _value = newValue;
        });
      },
    );
  }
}
```



Cupertino Segmented Control

- CupertinoSegmentedControl - это виджет визуального интерфейса (UI) в фреймворке Flutter, который отображает набор взаимозаменяемых опций, из которых пользователь может выбрать одну. Он использует стилизованный виджет в стиле iOS, который отображает набор кнопок, расположенных горизонтально или вертикально, где каждая кнопка представляет одну опцию выбора.
- При выборе опции пользователь может нажать на соответствующую кнопку, что вызовет событие выбора, и приложение может использовать это событие для принятия соответствующих действий. CupertinoSegmentedControl может быть настроен для отображения текста, изображений или даже пользовательских виджетов на каждой кнопке, а также для настройки стиля кнопок, включая цвет, рамку и радиус границы.

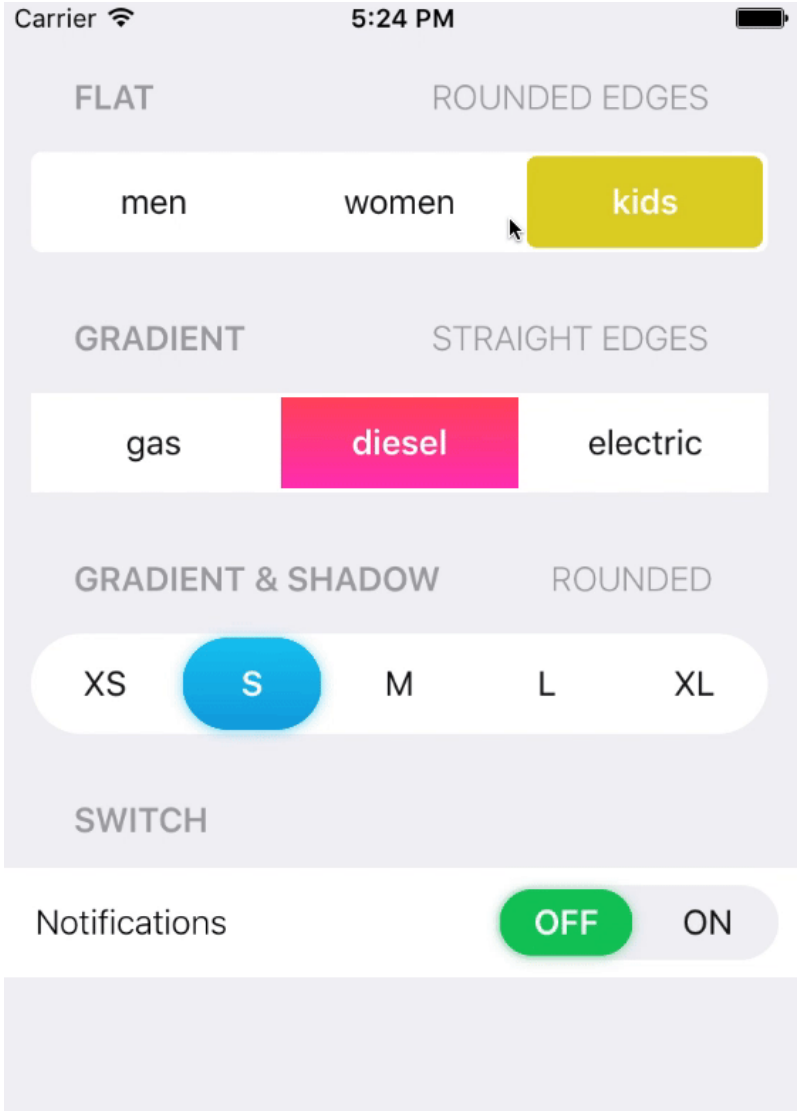
```dart
class ExampleSegmentedControl extends StatefulWidget {
  @override
  _ExampleSegmentedControlState createState() => _ExampleSegmentedControlState();
}
class _ExampleSegmentedControlState extends State<ExampleSegmentedControl> {
  int _selectedOption = 0;
  final Map<int, Widget> _options = {
    0: Text('Option 1'),
    1: Text('Option 2'),
    2: Text('Option 3'),
  };
  @override
  Widget build(BuildContext context) {
    return CupertinoSegmentedControl(
      children: _options,
      onValueChanged: (int value) {
        setState(() {
          _selectedOption = value;
        });
      },
      groupValue: _selectedOption,
    );
  }
}
```



Cupertino Sliding Segmented Control

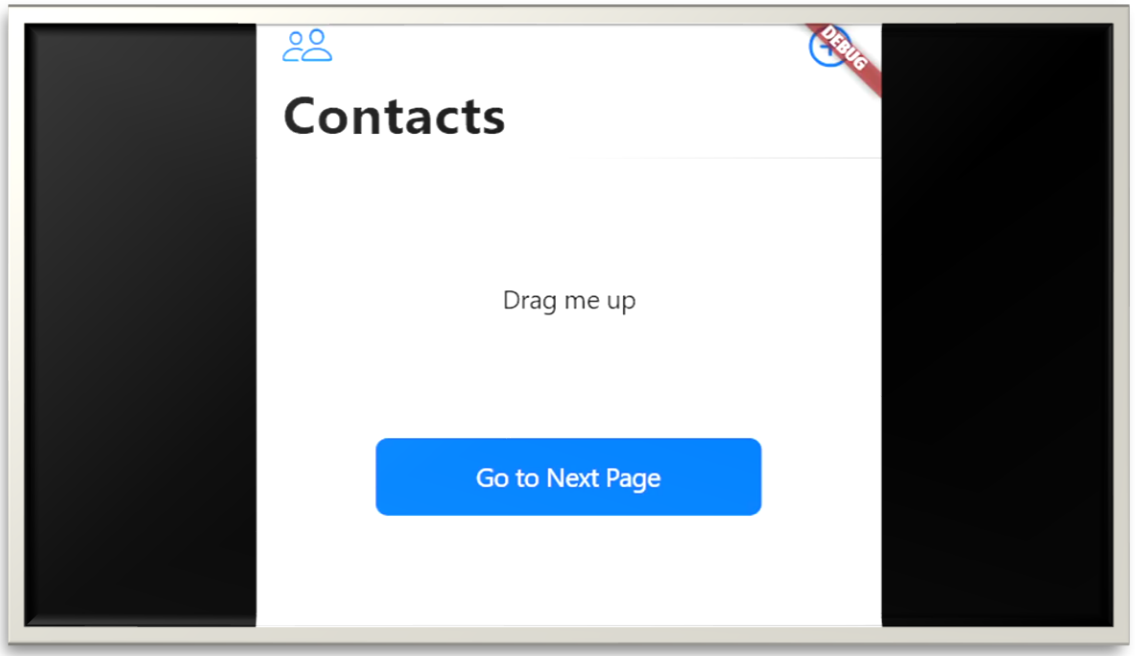
- CupertinoSlidingSegmentControl - это виджет визуального интерфейса пользовательского интерфейса (UI). Этот виджет позволяет пользователю выбирать один из нескольких вариантов, переключаясь между ними путем горизонтального смахивания (swiping). Каждый вариант представлен текстом, отображаемым в разных сегментах. Выбранный вариант выделяется выделением, что облегчает понимание текущего выбора.
- CupertinoSlidingSegmentControl имеет несколько настраиваемых свойств, которые позволяют изменять его внешний вид и поведение, например, цветовую схему, размер и т. д. Он может быть использован в качестве элемента управления в приложениях Flutter для iOS и других платформ, где желаемый стиль пользовательского интерфейса похож на Cupertino.

```dart
class ExampleScreen extends StatefulWidget {
  @override
  _ExampleScreenState createState() => _ExampleScreenState();
}
class _ExampleScreenState extends State<ExampleScreen> {
  int _selectedIndex = 0;
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Example'),
      ),
      body: Center(
        child: CupertinoSlidingSegmentedControl(
          children: {
            0: Text('Option 1'),
            1: Text('Option 2'),
            2: Text('Option 3'),
          },
          onValueChanged: (int? index) {
            setState(() {
              _selectedIndex = index!;
            });
          },
          groupValue: _selectedIndex,
        ),),); }}
```



CupertinoSliverNavigationBar

```
@override
Widget build(BuildContext context) {
 return CupertinoPageScaffold(
  // A ScrollView that creates custom scroll effects using slivers.
  child: CustomScrollView(
   // A list of sliver widgets.
   slivers: <Widget>[
    const CupertinoSliverNavigationBar(
     leading: Icon(CupertinoIcons.person_2),
     // This title is visible in both collapsed and expanded states.
     // When the "middle" parameter is omitted, the widget provided
     // in the "largeTitle" parameter is used instead in the collapsed state.
     largeTitle: Text('Contacts'),
     trailing: Icon(CupertinoIcons.add_circled),
    ),
    // This widget fills the remaining space in the viewport.
    // Drag the scrollable area to collapse the CupertinoSliverNavigationBar.
    SliverFillRemaining(
     child: Column(
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,
      children: <Widget>[
       const Text('Drag me up', textAlign: TextAlign.center),
       CupertinoButton.filled(
        onPressed: () {
         Navigator.push(context, CupertinoPageRoute<Widget>(
          builder: (BuildContext context) {
           return const NextPage();
          }));
        },
        child: const Text('Go to Next Page'),
       ),
      ], ), ), ], ), ); }}
```

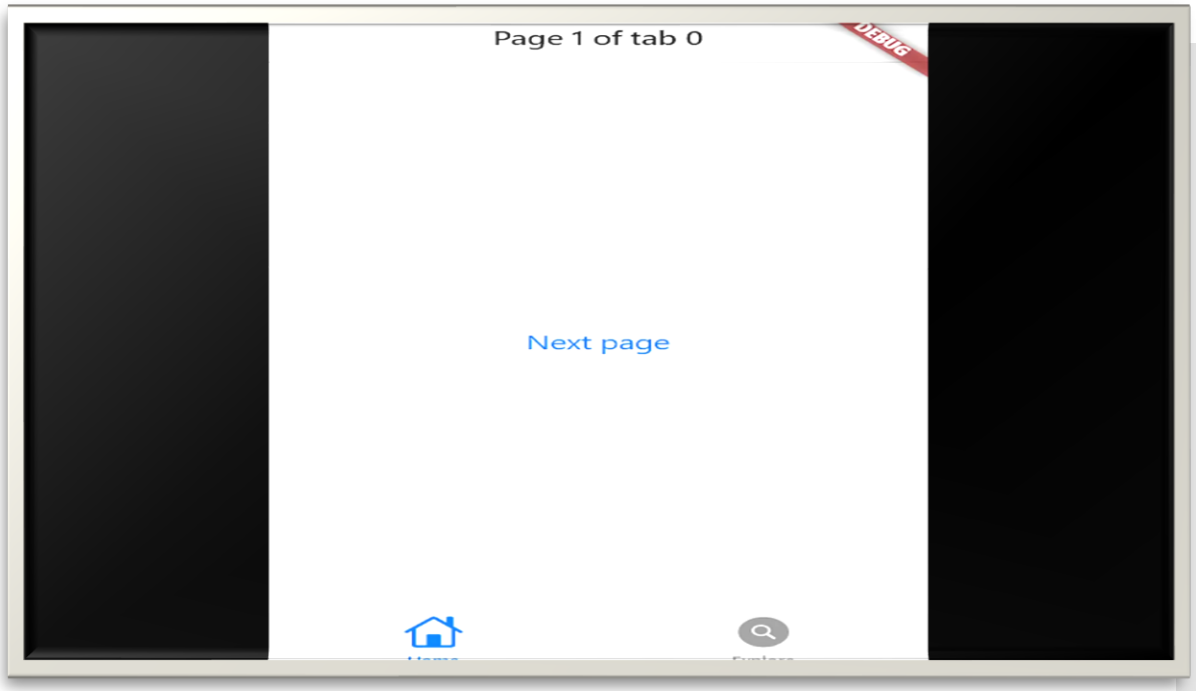


alwaysShowMiddle → bool
automaticallyImplyLeading
leading → Widget?

Альтерантива - CupertinoNavigationBar

CupertinoTabScaffold

```
class _TabScaffoldExampleState extends State<TabScaffoldExample> {
 @override
 Widget build(BuildContext context) {
  return CupertinoTabScaffold(
   tabBar: CupertinoTabBar(
    items: const <BottomNavigationBarItem>[
     BottomNavigationBarItem(
      icon: Icon(CupertinoIcons.home),
      label: 'Home',),
     BottomNavigationBarItem(
      icon: Icon(CupertinoIcons.search_circle_fill),
      label: 'Explore',), ], ),
   tabBuilder: (BuildContext context, int index) {
    return CupertinoTabView(
     builder: (BuildContext context) {
      return CupertinoPageScaffold(
       navigationBar: CupertinoNavigationBar(
        middle: Text('Page 1 of tab $index'),),
       child: Center(
        child: CupertinoButton(
         child: const Text('Next page'),
         onPressed: () {
          Navigator.of(context).push(
           CupertinoPageRoute<void>(
            builder: (BuildContext context) {
             return CupertinoPageScaffold(
              navigationBar: CupertinoNavigationBar(
               middle: Text('Page 2 of tab $index'),),
              child: Center(
               child: CupertinoButton(
                child: const Text('Back'),
                onPressed: () {
                 Navigator.of(context).pop();
```



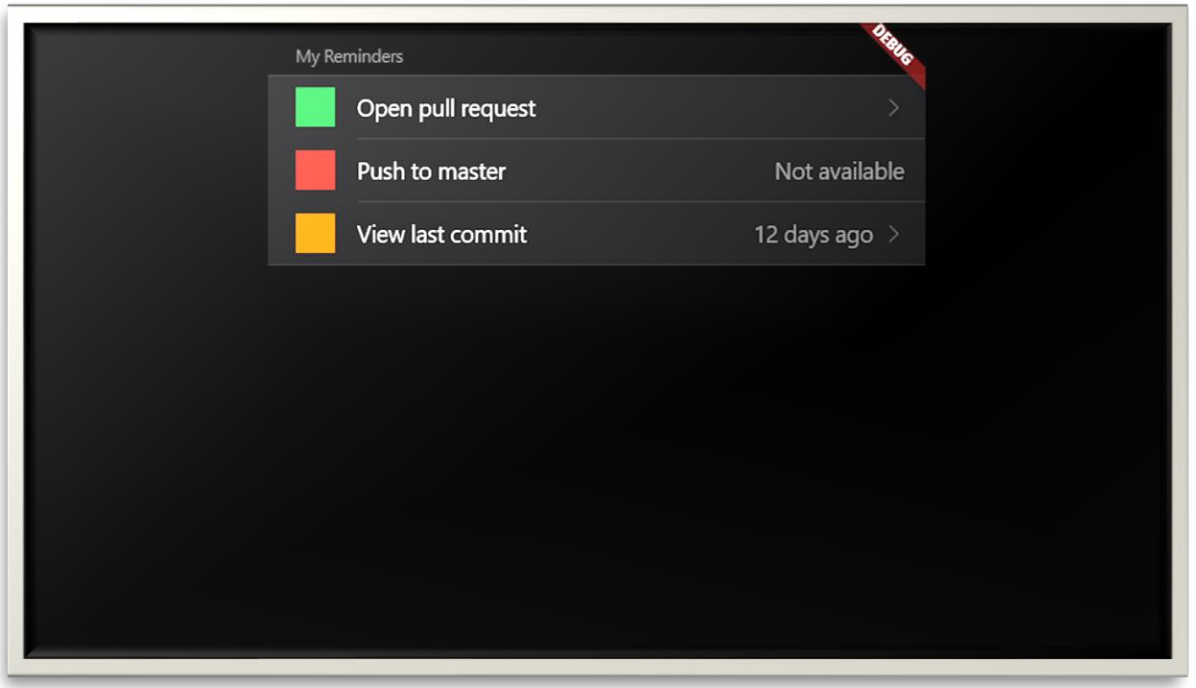
backgroundColor → Color?
controller → CupertinoTabController?
tabBar → CupertinoTabBar

CupertinoTabView используется в качестве содержимого каждой вкладки в CupertinoTabScaffold, где могут сосуществовать несколько stateful.

CupertinoListSection

```
class MyStatelessWidget extends StatelessWidget {
 const MyStatelessWidget({super.key});

 @override
 Widget build(BuildContext context) {
  return CupertinoPageScaffold(
   child: CupertinoListSection(
    header: const Text('My Reminders'),
    children: <CupertinoListTile>[
     CupertinoListTile(
      title: const Text('Open pull request'),
      leading: Container(
       width: double.infinity,
       height: double.infinity,
       color: CupertinoColors.activeGreen,
      ),
      trailing: const CupertinoListTileChevron(),
      onTap: () => Navigator.of(context).push(
       CupertinoPageRoute<void>(
        builder: (BuildContext context) {
         return const _SecondPage(text: 'Open pull request');
        },
       ),
      ),
     ),
```



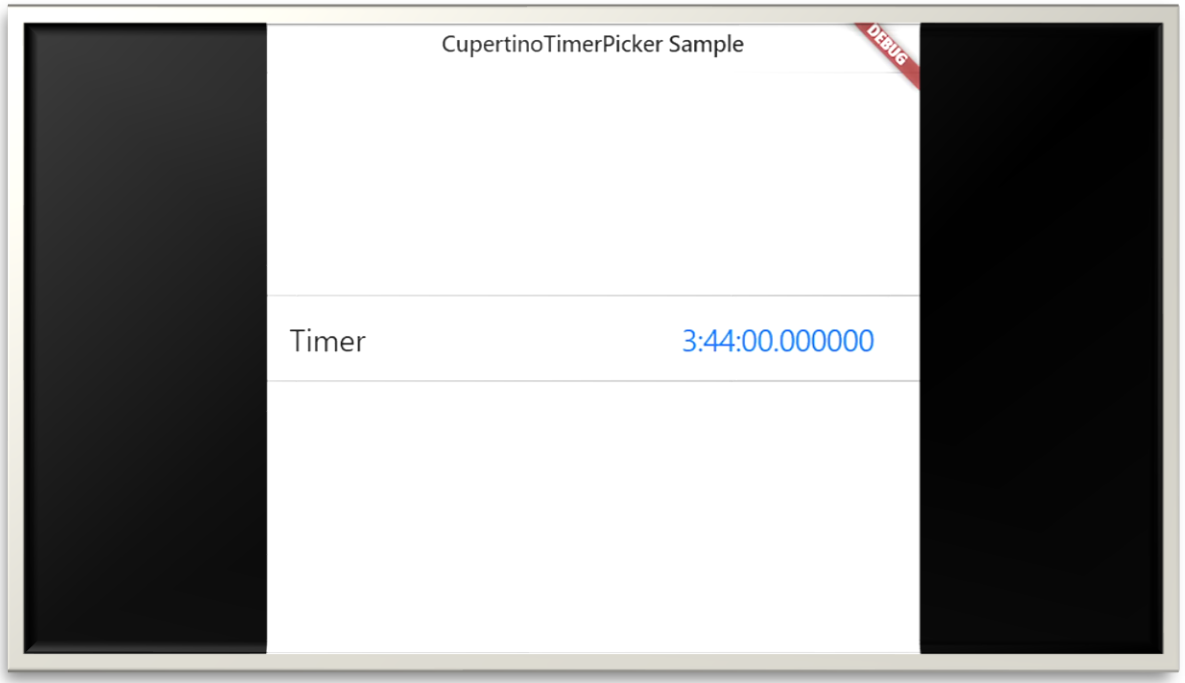
additionalDividerMargin → double
clipBehavior → Clip
header → Widget?

CupertinoListTile

Аналог в Android – ListView / RecyclerView

CupertinoTimerPicker

```
@override
Widget build(BuildContext context) {
 return CupertinoPageScaffold(
  navigationBar: const CupertinoNavigationBar(
   middle: Text('CupertinoTimerPicker Sample'),
  ),
  child: DefaultTextStyle(
   style: TextStyle(
    color: CupertinoColors.label.resolveFrom(context),
    fontSize: 22.0,
   ),
   child: Center(
    child: Column(
     mainAxisAlignment: MainAxisAlignment.center,
     children: <Widget>[
      _TimerPickerItem(
       children: <Widget>[
        const Text('Timer'),
        CupertinoButton(
         // Display a CupertinoTimerPicker with hour/minute mode.
         onPressed: () => _showDialog(
          CupertinoTimerPicker(
           mode: CupertinoTimerPickerMode.hm,
           initialTimerDuration: duration,
           // This is called when the user changes the timer's
           // duration.
           onTimerDurationChanged: (Duration newDuration) {
            setState(() => duration = newDuration);
           },
          ),
         ),
         // In this example, the timer's value is formatted manually.
         // You can use the intl package to format the value based on
         // the user's locale settings.
         child: Text(
          '$duration',
          style: const TextStyle(
           fontSize: 22.0,
          ),
```



CupertinoTimerPickerMode enum : hm, ms, hms
initialTimerDuration → Duration
onTimerDurationChanged → ValueChanged<Duration>

The picker has a fixed size of 320 x 216, in logical pixels, with the exception of CupertinoTimerPickerMode.hms, which is 330 x 216. If the parent widget provides more space than it needs

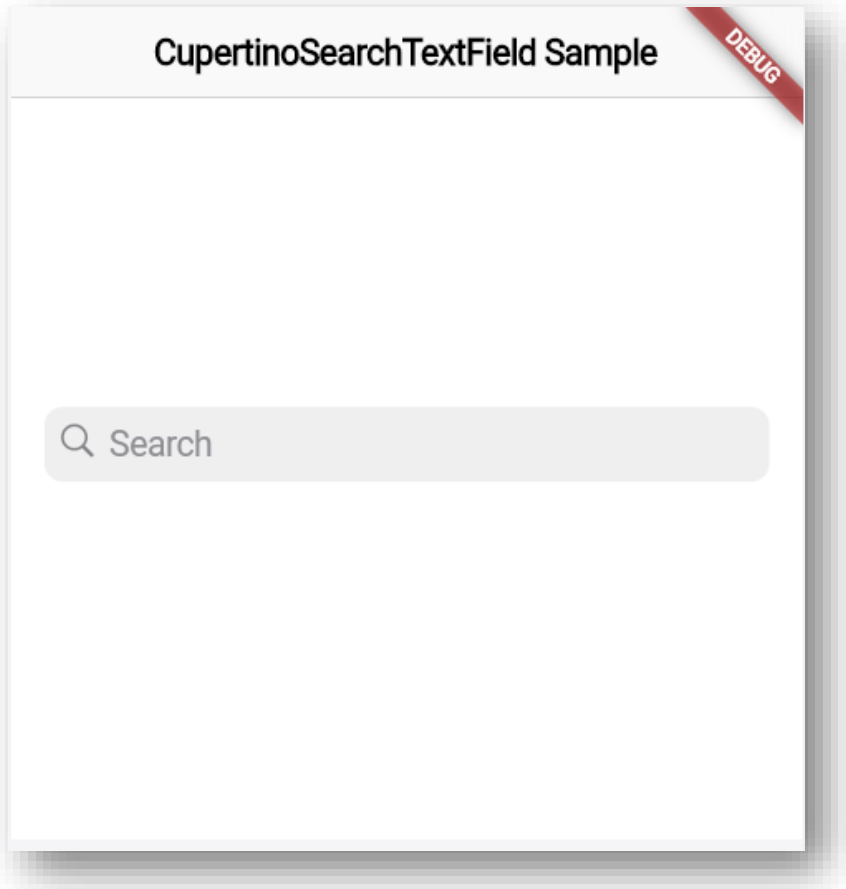
CupertinoSearchTextField

```dart
class _SearchTextFieldExampleState extends State<SearchTextFieldExample> {
 late TextEditingController textController;

 @override
 void initState() {
  super.initState();
  textController = TextEditingController(text: 'initial text');
 }

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

 @override
 Widget build(BuildContext context) {
  return CupertinoPageScaffold(
   navigationBar: const CupertinoNavigationBar(
    middle: Text('CupertinoSearchTextField Sample'),
   ),
   child: Center(
    child: Padding(
     padding: const EdgeInsets.all(16.0),
     child: CupertinoSearchTextField(
      controller: textController,
      placeholder: 'Search',
     ),
    ),
   ),
  );
 }
}
```



autocorrect → bool
autofocus → bool
onChanged → ValueChanged<T>
onSubmitted → ValueChanged<T>
enabled → bool?

Flutter Platform Widgets

```
if(Platform.isIOS){
return CupertinoButton();
}
else if(Platform.IsAndroid){
return ElevatedButton();
}
```

```
import 'package:flutter/widgets.dart';
import 'package:flutter_adapt_style_platform/selector_screen.dart';
import 'package:flutter_adapt_style_platform/styles.dart';
import 'package:flutter_platform_widgets/flutter_platform_widgets.dart';

class App extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return PlatformProvider(
      builder: (BuildContext context) => PlatformApp(
        title: 'Flutter platform style sample',
        material: (_, __) => MaterialAppData(
          theme: materialTheme, debugShowCheckedModeBanner: false),
        cupertino: (_, __) => CupertinoAppData(
          theme: cupertinoTheme, debugShowCheckedModeBanner: false),
        home: SelectorScreen()));
  }
}
```

Flutter SDK предлагает два набора виджетов для стилей: Material и Cupertino Widget. Виджет Material реализует Material Design Language для iOS, Android, веб-приложений и десктопных приложений. С другой стороны, виджеты Cupertino используются для реализации текущего iOS design language на основе рекомендаций Apple – Human Design.

Итак, возникает вопрос, зачем нужны виджеты Cupertino для разработки iOS-приложений, когда есть виджет Material, которые можно использовать для любой платформы, включая iOS?

flutter pub add flutter_platform_widgets