Лекция № 3

Widgets

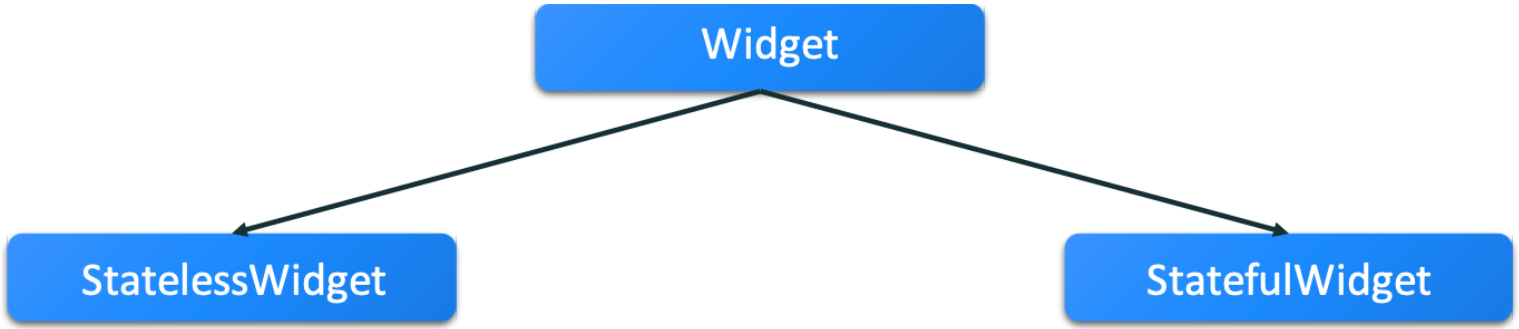
Widget

В Flutter **все является виджетом**. Начиная с кнопки, экрана и заканчивая целым приложением.

Виджеты - это центральная иерархия классов в фреймворке Flutter.

Виджет - это **неизменяемое** описание части пользовательского интерфейса.

Виджеты в Flutter можно разделить на две основные категории: без состояния (наследуются от StatelessWidget) и с сохранением состояния (наследуются от StatefulWidget и содержат объект состояния State).



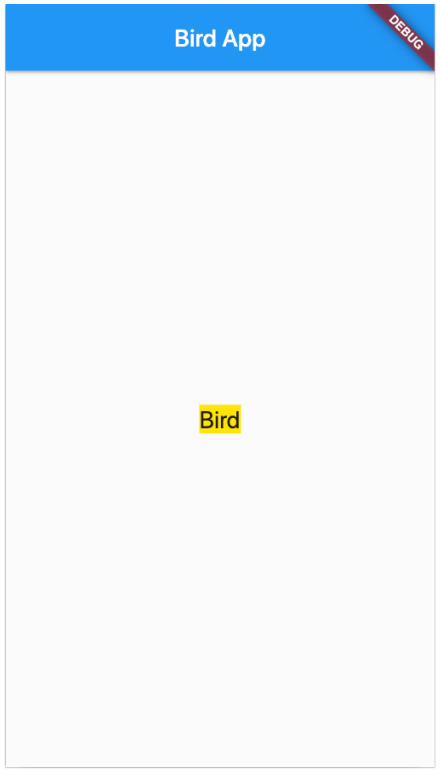
Widget without state
Ex.:
- Text()
- Container()
- TextButton()
- Etc.

Widget with some state
Ex.:
- Image()
- Checkbox()
- Form()
- Slider()
- Etc.

StatelessWidget

Виджеты без состояния являются неизменяемыми и не имеют никакого внутреннего состояния. Они просто отображают свои свойства и перестраиваются при изменении этих свойств.

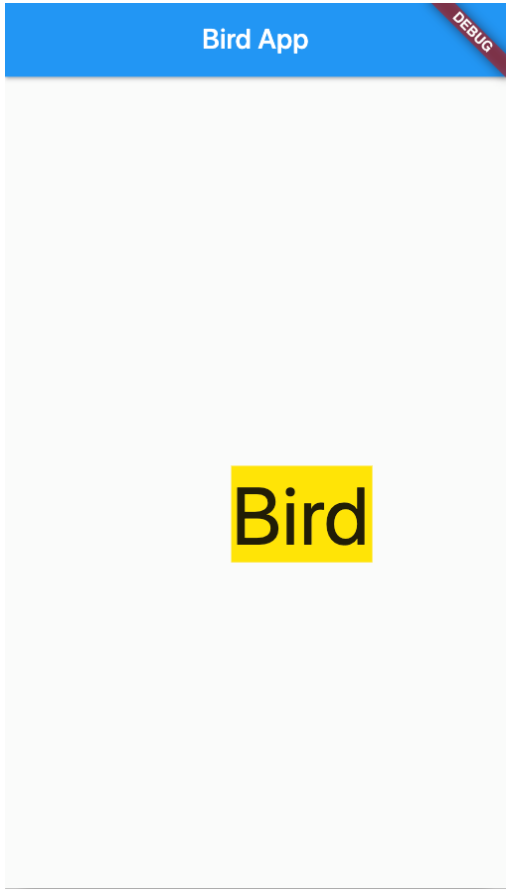
```
void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Bird App',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Bird App'),
        ),
        body: Center(
          child: Bird(),
        ),
      ),
    );
  }
}
```



StatefulWidget

Виджеты с сохранением состояния, с другой стороны, имеют изменяемое состояние, которое может изменяться в течение срока службы виджета.

```
class Bird extends StatefulWidget {
  const Bird({
    Key? key,
    this.color = const Color(0xFFFFE306),
    this.child = const Text('Bird', style: TextStyle(fontSize: 20)),
  }) : super(key: key);

  final Color;
  final Widget? child;
  @override
  State<Bird> createState() => _BirdState();
}

class _BirdState extends State<Bird> {
  double _size = 1.0;

  void grow() {
    setState(() {
      _size += 0.1;
    });
  }
  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: grow,
      child: Container(
        color: widget.color,
        transform: Matrix4.diagonal3Values(_size, _size, 1.0),
        child: widget.child,
```



Пояснение

Класс Bird управляет своим собственным состоянием, поэтому он переопределяет createState() для создания объекта State. Фреймворк вызывает createState(), когда хочет построить виджет. В этом примере createState() возвращает экземпляр _BirdState.

```
class Bird extends StatefulWidget {
  const Bird({
    Key? key,
    this.color = const Color(0xFFFFE306),
    this.child = const Text('Bird', style: TextStyle(fontSize: 20)),
  }) : super(key: key);

  final Color;
  final Widget? child;
  @override
  State<Bird> createState() => _BirdState();
}
```

Класс _BirdState хранит изменяемые данные (размер _size), которые могут меняться в течение всего времени существования виджета.

```
class _BirdState extends State<Bird> {
  double _size = 1.0;

  void grow() {
    setState(() {
      _size += 0.1;
    });
  }

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: grow,
      child: Container(
        color: widget.color,
        transform: Matrix4.diagonal3Values(_size, _size, 1.0),
        child: widget.child,
      ),
    );
  }
}
```

Рекомендации

Используйте StatelessWidget всегда когда можете обойтись без StatefulWidget

Минимизируйте количество childs в StatefulWidget

Widget types

Кроме StatelessWidget и StatefulWidget во Flutter есть несколько других типов виджетов, включая унаследованные виджеты, виджеты Material Design и виджеты из Купертино.

Унаследованные виджеты позволяют передавать данные по дереву виджетов без перестройки всех виджетов, в то время как виджеты Material Design и Cupertino предоставляют набор компонентов пользовательского интерфейса, которые соответствуют рекомендациям Material Design и дизайну iOS соответственно.

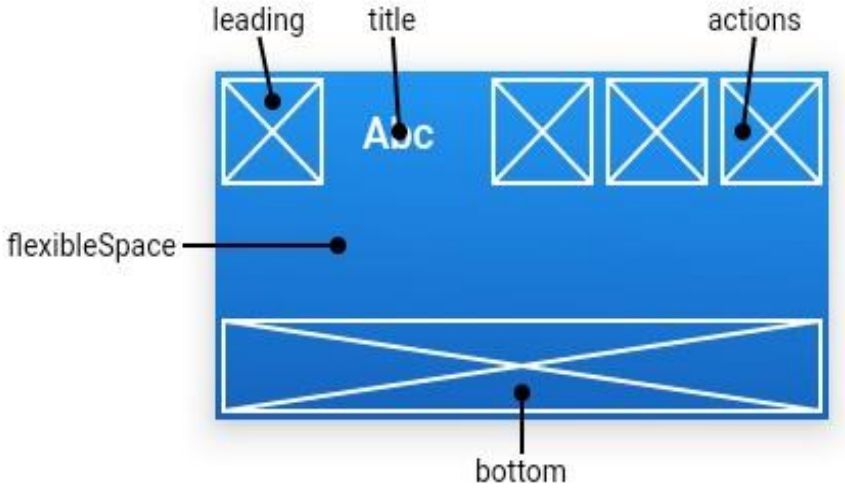
Basic Widgets

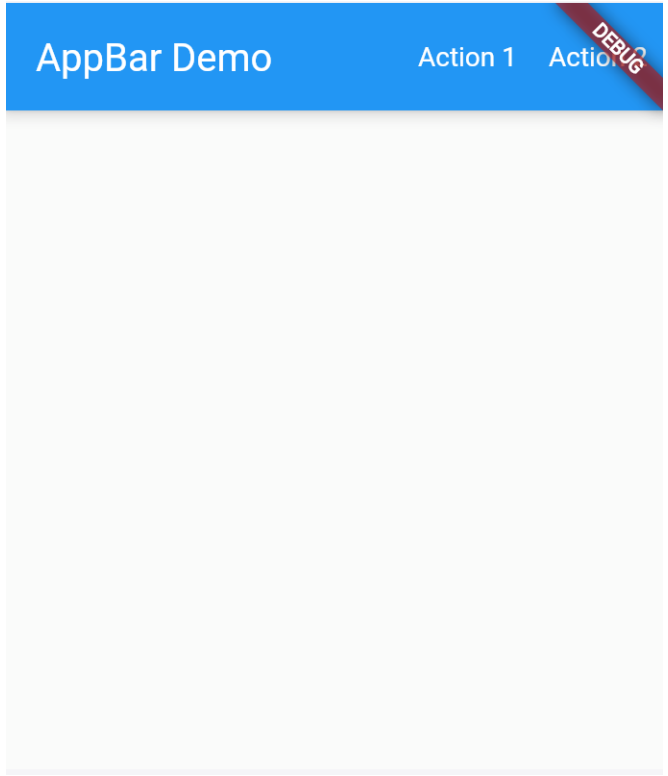
- Appbar
- Column
- Container
- FlutterLogo
- Icon
- Image
- Placeholder
- RaisedButton
- Row
- Scaffold
- Text

Appbar

Панель приложения состоит из панели инструментов и потенциально других виджетов.

```
appBar: AppBar(
      title: const Text('AppBar Demo'),
      actions: <Widget>[
        TextButton(
          style: style,
          onPressed: () {},
          child: const Text('Action 1'),
        ),
        TextButton(
          style: style,
          onPressed: () {},
          child: const Text('Action 2'),
        ),
      ],
    )
```



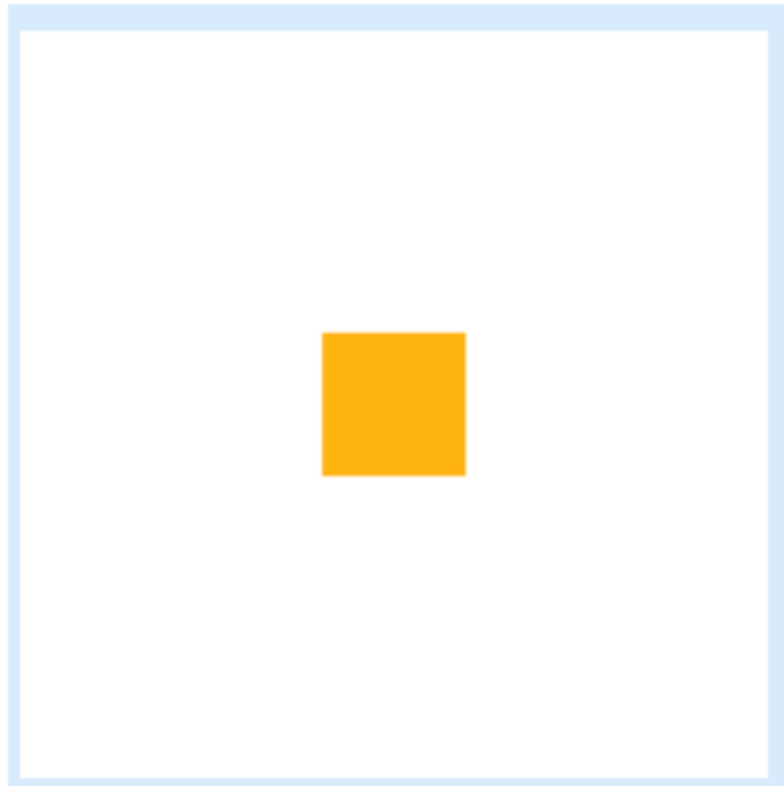


Column

```
Column(
  children: const <Widget>[
    Text('Deliver features faster'),
    Text('Craft beautiful UIs'),
    Expanded(
      child: FittedBox(
        child: FlutterLogo(),
      ),
    ),
  ],
)
```

Container

```
Center(
  child: Container(
    margin: const EdgeInsets.all(10.0),
    color: Colors.amber[600],
    width: 48.0,
    height: 48.0,
  ),
)
```



Контейнер сначала окружает дочерний элемент отступом, а затем применяет дополнительные ограничения к заполненному экстенту. Затем контейнер окружается дополнительным пустым пространством, описанным от края.

Image

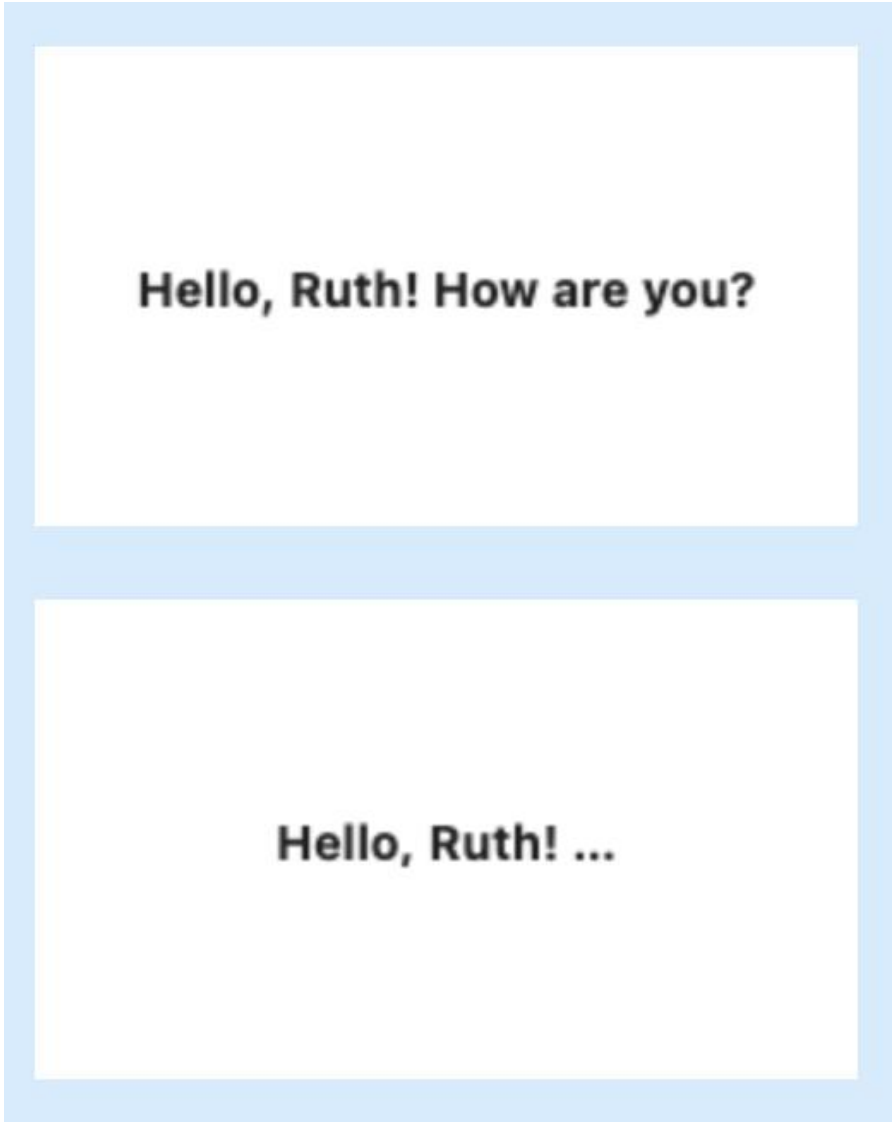
```
const Image(
  image: NetworkImage('https://flutter.github.io/assets-for-api-docs/assets/widgets/owl.jpg'),
)
```

Row

```
Row(
  children: const <Widget>[
    Expanded(
      child: Text('Deliver features faster', textAlign: TextAlign.center),
    ),
    Expanded(
      child: Text('Craft beautiful UIs', textAlign: TextAlign.center),
    ),
    Expanded(
      child: FittedBox(
        child: FlutterLogo(),
      ),
    ),
  ],
)
```

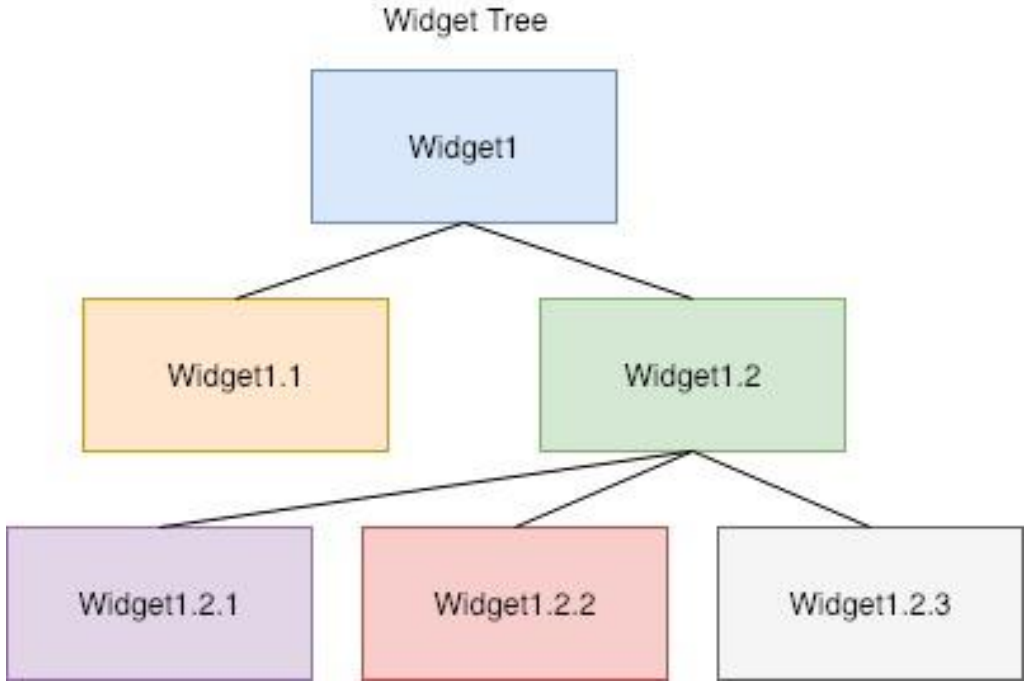
Text

```
Text(
  'Hello, $_name! How are you?',
  textAlign: TextAlign.center,
  overflow: TextOverflow.ellipsis,
  style: const TextStyle(fontWeight: FontWeight.bold),
)
```



Дерево виджетов и иерархия

Каждый элемент приложения Flutter — это виджет, включая кнопки, текстовые поля, изображения и даже само приложение. Эти виджеты организованы в виде древовидной структуры с одним корневым виджетом наверху и другими виджетами, ответвляющимися от него. Каждый виджет имеет определенное назначение и может иметь дочерние виджеты, которые сами могут иметь дочерние виджеты и так далее.



Все в основном сводится к переопределению build() методов конструируемых нами виджетов. Первое что нужно понять — это то, что при изменении состояния вызывается этот метод виджета и создает все Widget-объекты по новой!

Т.е. виджеты являются иммутабельными и при изменении состояния просто создаются новые.

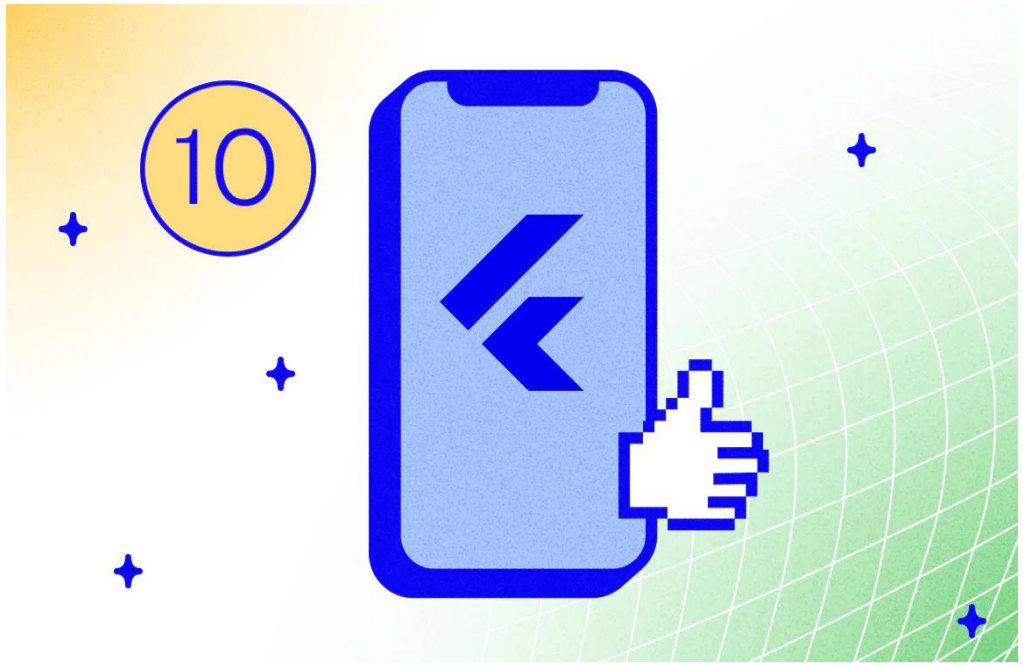
```
class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp();
```

Widget — это лишь описание отображаемых на экране объектов. Если объект уже создан и отрисован на основании какого-то виджета, то при изменении этого виджета, скорее всего нам не нужно пересоздавать этот объект заново, а возможно даже и изменять его вовсе. А мы помним, что виджеты пересоздаются всегда при смене состояния, даже если их структура не изменилась.

Для андроид программистов можно провести аналогию с RecyclerView — не нужно пересоздавать ViewHolder’ы (аналогия наших тяжелый объектов), если они уже созданы и отрисованы на основании какой то декларативной сущности (аналогия с Widget). При отображении новых сущностей у ViewHolder’ов лишь изменяются параметры без создания новых инстансов (в идеале), а возможно и вовсе остаются нетронутыми.

Организация виджетов для повышения производительности

Организация виджетов для повышения производительности во Flutter относится к процессу структурирования и управления иерархией виджетов таким образом, чтобы упростить разработку, отладку и поддержку кодовой базы. Правильная организация виджетов может помочь повысить производительность, сделав код более модульным, удобным для чтения и снизив риск ошибок и багов. Всегда полезно следовать хорошим практикам и максимально оптимизировать наше приложение.



1. Не выносите виджеты в методы класса

Когда у нас есть сложное представление, чтобы реализовать в один виджет, то обычно мы разделяем его на виджеты поменьше, которые помещаем в методы класса. В следующем примере представлен виджет, содержащий заголовок, основной контент и "подвал".

```
class MyHomePage extends StatelessWidget {
    Widget _buildHeaderWidget(…) { return … }
    Widget _buildMainWidget(…) { return … }
    Widget _buildFooterWidget(…) { return … }
    @override
    Widget build(…) { return
            _buildHeaderWidget(),
            _ buildMainWidget(),
            _buildFooterWidget()
}}
```

То, что мы увидели выше, – это антипаттерн. Почему так? Всё потому, что, когда мы вносим изменения и обновляем MyHomePage виджет, то виджеты, которые у нас вынесены методах, также обновляются, даже если в этом нет никакой необходимости.

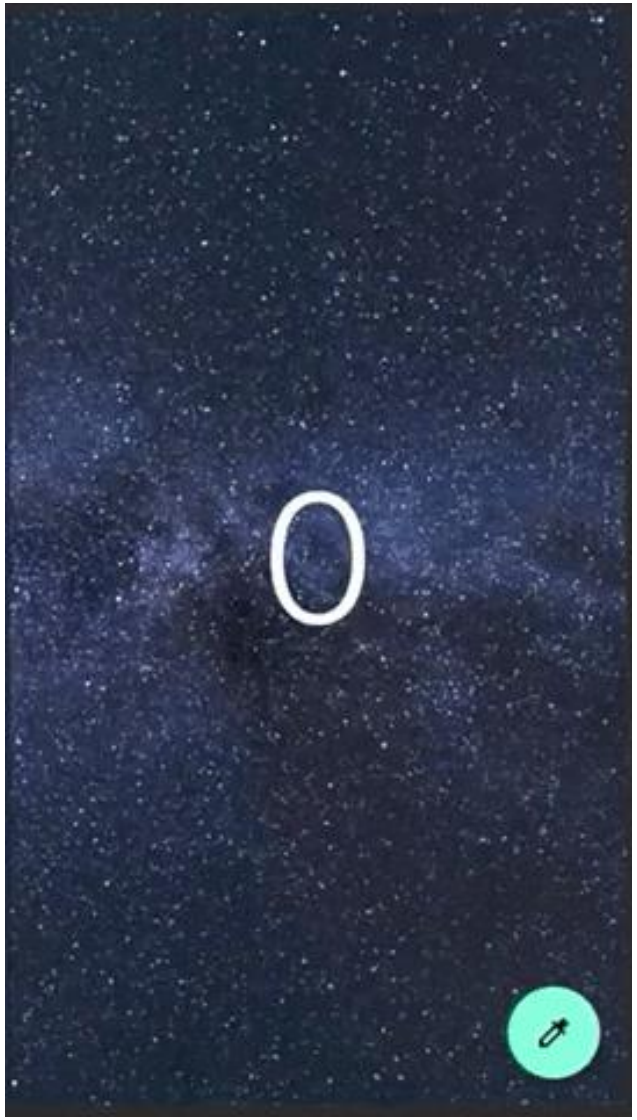
У Stateful/Stateless виджетов есть специальный механизм "кэширования", учитывающий ключ, тип виджета и его атрибуты, который позволяет не перестраивать виджет без необходимости. Кроме того, это помогает нам инкапсулировать и рефакторировать наши виджеты.

```
class MyHomePage extends StatelessWidget {
  @override
  Widget build(…) { return HeaderWidget(), MainWidget(), FooterWidget() … }
}
class HeaderWidget extends StatelessWidget {
  @override
  Widget build(…) { return … }
}
class FooterWidget extends StatelessWidget { … }
class MainWidget extends StatelessWidget { … }
```

2. Используйте ‘const’

Рекомендуется использовать ключевое слово const для значений, которые возможно инициализировать во время компиляции, а также при вызове конструктора виджета (если он поддерживает const, конечно), что позволяет работать с одним и тем же каноническим экземпляром, тем самым избегая повторных вычислений.

Давайте используем пример с setState, но в этот раз мы добавим счетчик, который будет увеличивать значение на 1 каждый раз при нажатии на кнопку.



У нас снова 2 вывода в консоли, один из которых относится к основнову виджету, а другой – к BackgroundWidget. Каждый раз, когда мы нажимаем кнопку, мы видим, что дочерний виджет также перестраивается, хотя его содержимое никак не меняется.

```
body: Stack(
  children: [
    Positioned.fill(
      child: const BackgroundWidget(),
    ),
    Center(
```

Теперь после клика по кнопке мы видим вывод только для основного виджета и избегаем перестроения виджета, вызванного с const.

3. Используйте ‘itemExtent’ в ‘ListView’ для больших списков

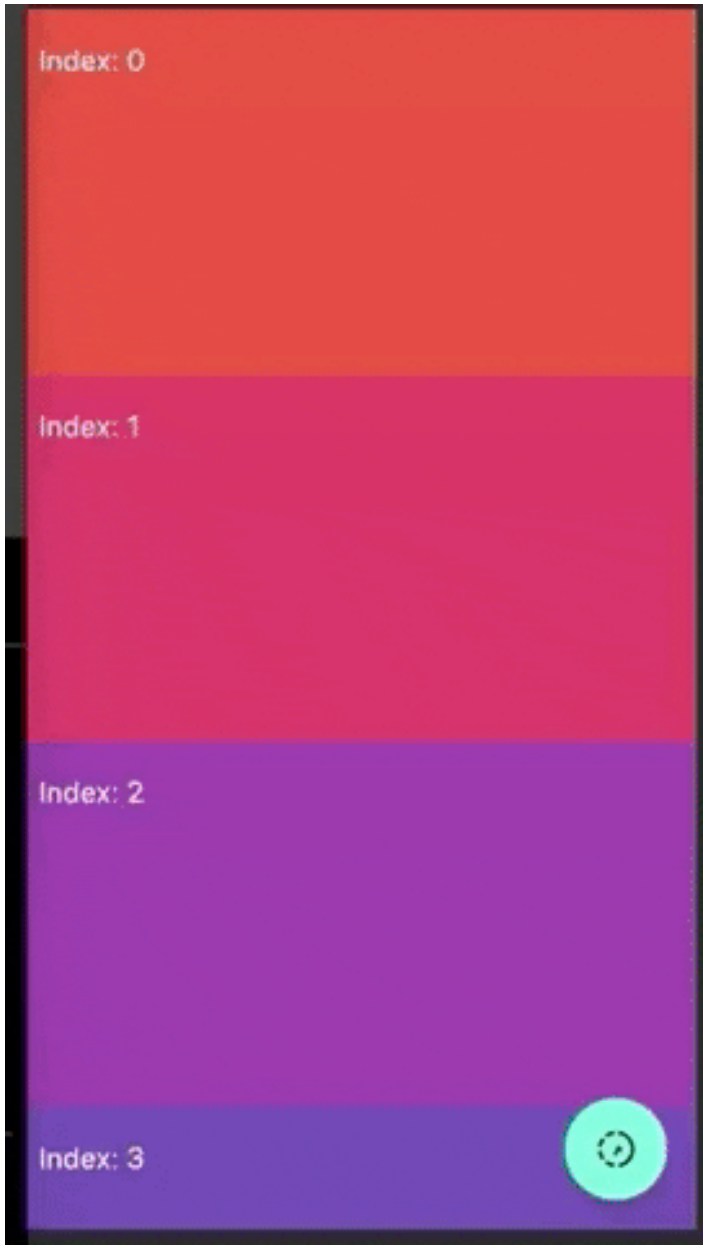
Иногда, когда у нас есть очень длинный список, и мы хотим быстро переместиться по нему, например, в самый конец, очень важно использовать itemExtent. Давайте рассмотрим простой пример. У нас есть список из 10 тысяч элементов. При нажатии на кнопку мы перейдем к последнему элементу. В этом примере мы не будем использовать itemExtent и позволим элементам списка самим определить свой размер.

Как можно увидеть на анимации выше, переход происходит очень долго (~10 секунд). Так получается из-за того, что дочерние элементы сами определяют свой размер. Это даже блокирует UI!

Чтобы избежать этого, мы должны использовать свойство itemExtent, благодаря которому при прокрутке не совершается лишней работы по расчету позиции скролла, так как размеры элементов заранее известны.

```
);
body: ListView(
  controller: _scrollController,
  children: widgets,
  itemExtent: 200,
),
```

С этим небольшим изменением мы мгновенно переходим в самый низ без каких-либо задержек.



С более подробными советами для оптимизации можно ознакомиться тута:

https://habr.com/ru/post/502882/

https://flutter.dev/docs/perf/rendering/best-practices

https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html#performance-considerations

https://api.flutter.dev/flutter/widgets/Opacity-class.html#transparent-image

https://api.flutter.dev/flutter/widgets/Opacity-class.html#transparent-image

RenderObjects, Elements and BuildContext

Render Objects

RenderObject — тот самый тяжелый объект.

RenderObject содержит всю логику для отображения фактического виджета и является довольно дорогим для создания инстанса. Он заботится о расчете layout’а, отрисовки и hit-testing’а. Хорошей идеей будет хранить эти объекты в памяти как можно дольше и, возможно, даже переиспользовать их (так как их создание довольно дорого). В отличии от Widget — объекты RenderObject являются мутабельными. RenderObject’ы также можно представить в виде дерева отображаемых объектов.

ПРИМЕР. *Имеется экран со множеством текстовых полей. Их может быть как 5, так и 30. Между ними могут находиться различные виджеты.*

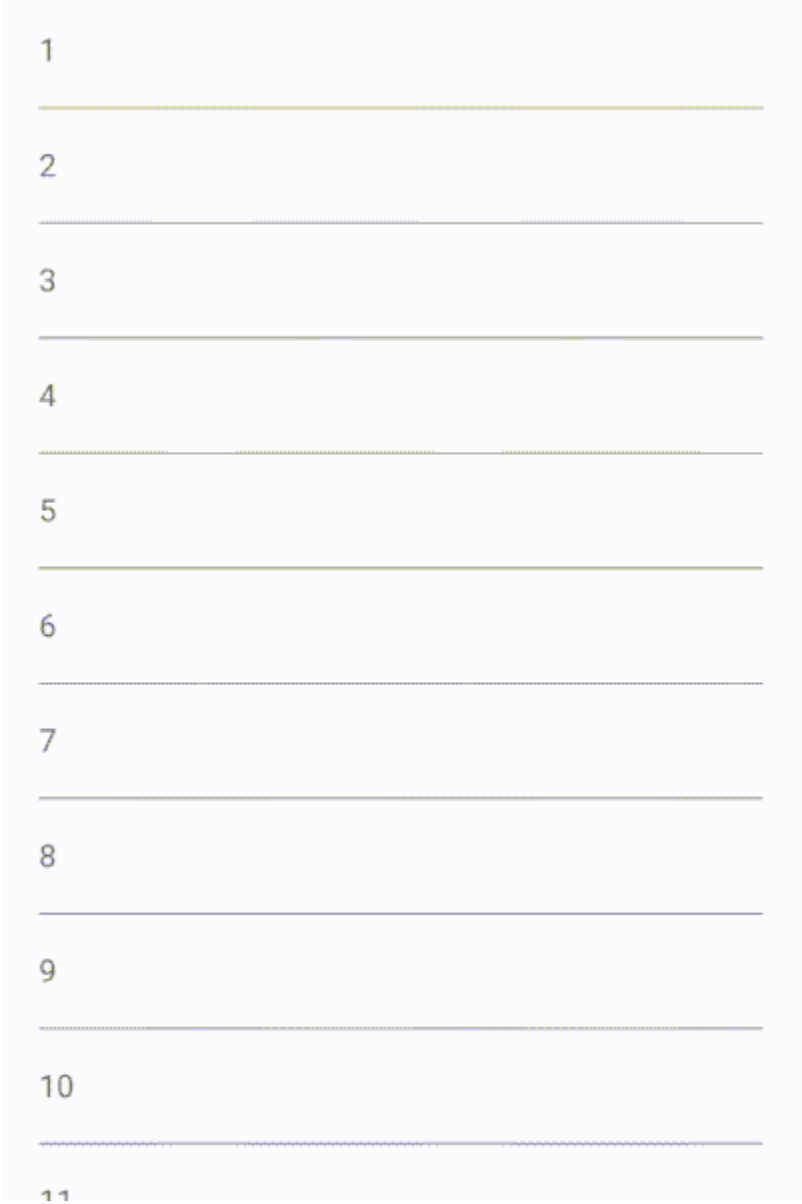
Задача:

• Поместить над клавиатурой блок с кнопкой «Далее» для переключения на следующее поле.

• При смене фокуса подскролливать поле к блоку с кнопкой «Далее».

Проблема:

Блок с кнопкой перекрывает текстовое поле. Нужно реализовать автоматический скролл на размер перекрываемого пространства текстового поля.



RenderObject позволяет решить данную проблему.

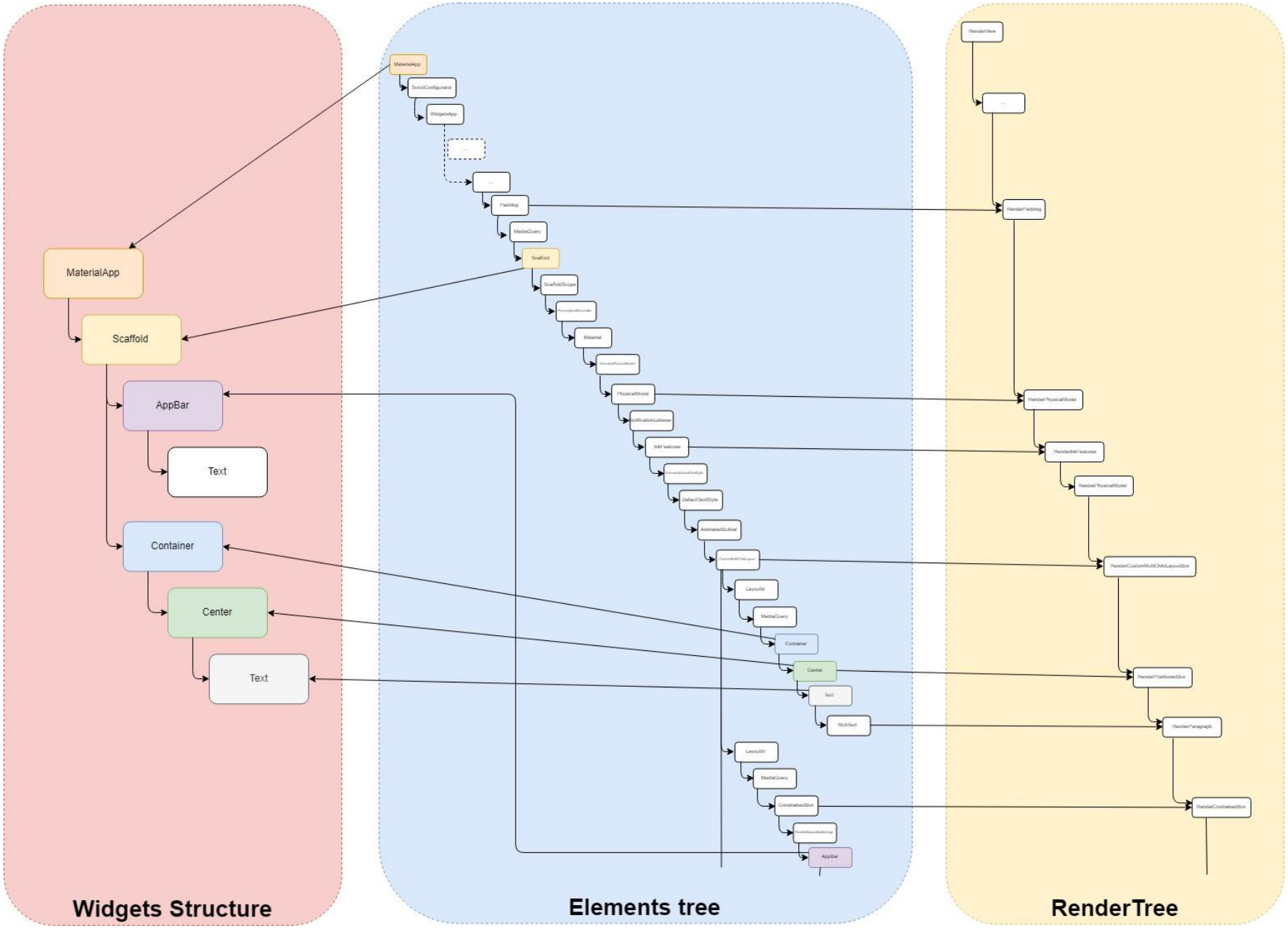
Копнув глубже уровня виджетов, с помощью небольших манипуляций со слоем рендера получили полезную функциональность, которая позволяет писать более сложные UI и логику. Иногда нужно знать размеры динамических виджетов, их позицию или сравнить перекрывающие друг на друга виджеты.

Elements

Элемент — некоторое представление виджета в определенном месте дерева.

Виджет описывает конфигурацию некоторой части пользовательского интерфейса, но как мы уже знаем, один и тот же виджет может использоваться в разных местах дерева. Каждое такое место будет представлено соответствующим элементом. Но со временем, виджет, который связан с элементом может поменяться. Это означает, что элементы более живучие и продолжают использоваться, лишь обновляя свои связи.

Это довольно рациональное решение. Как мы уже определили выше, виджеты — неизменяемая конфигурация, которая просто описывает определенную часть интерфейса, а значит они должны быть очень легковесными. А элементы, в зоне которых управление, являются намного более тяжеловесными, но они не пересоздаются без надобности.


MaterialApp
Scaffold
AppBar
Text
Container
Center
Text
Widgets Structure
Elements tree
RenderTree

Чтобы понять каким образом это осуществляется, рассмотрим жизненный цикл элемента:

Элемент создаётся посредством вызова метода Widget.createElement и конфигурируется экземпляром виджета, у которого был вызван метод.

- С помощью метода mount созданный элемент добавляется в заданную позицию родительского элемента. При вызове данного метода также ассоциируются дочерние виджеты и элементам сопоставляются объекты дерева рендеринга.
- Виджет становится активным и должен появиться на экране.
- В случае изменения виджета, связанного с элементом (например, если родительский элемент изменился), есть несколько вариантов развития событий. Если новый виджет имеет такой же runtimeType и key, то элемент связывается с ним. В противном случае, текущий элемент удаляется из дерева, а для нового виджета создаётся и ассоциируется с ним новый элемент.

- В случае, если родительский элемент решит удалить дочерний элемент, или промежуточный между ними, это приведет к удалению объекта рендеринга и переместит данный элемент в список неактивных, что приведет к деактивации элемента (вызов метода deactivate).
- Когда элемент считается неактивным, он не находится на экране. Элемент может находиться в неактивном состоянии только до конца текущего фрейма, если за это время он остается неактивным, он демонтируется (unmount), после этого считается несуществующим и больше не будет включен в дерево.
- При повторном включении в дерево элементов, например, если элемент или его предки имеют глобальный ключ, он будет удален из списка неактивных элементов, будет вызван метод activate, и рендер объект, сопоставленный данному элементу, снова будет встроен в дерево рендеринга. Это означает, что элемент должен снова появиться на экране.

Build Context

BuildContext — это фундаментальная концепция, которая представляет расположение виджета в дереве виджетов. Он обеспечивает доступ к широкому спектру услуг и информации, такой как размер и положение виджета, текущая тема и данные медиа-запроса.

BuildContext передается виджетам в качестве параметра во время метода сборки и используется для доступа к свойствам виджета и его родительских виджетов. Он также используется для создания новых виджетов и навигации по дереву виджетов.

Одной из наиболее важных функций BuildContext является предоставление доступа к InheritedWidget. InheritedWidget — это тип виджета, который позволяет обмениваться данными между виджетами без необходимости сверления реквизита. Используя BuildContext, разработчики могут легко получить доступ к InheritedWidget и получить необходимые им данные.

BuildContext также предоставляет доступ к навигатору, который используется для навигации между экранами в приложении. С помощью Navigator разработчики могут выталкивать и выталкивать экраны в дереве виджетов, позволяя пользователям перемещаться по приложению.

Наконец, BuildContext предоставляет доступ к MediaQuery, который предоставляет информацию о размере экрана устройства и плотности пикселей. Эта информация имеет решающее значение для создания адаптивных приложений, которые могут адаптироваться к разным размерам и ориентации экрана.

В целом, BuildContext — это фундаментальная концепция Flutter, и она широко используется во всем фреймворке. Понимая, как эффективно использовать BuildContext, разработчики могут создавать высокопроизводительные, отзывчивые и масштабируемые приложения, обеспечивающие превосходное взаимодействие с пользователем.

Создание пользовательских виджетов

Мы создаем **пользовательские виджеты,** когда хотим, чтобы наше приложение выглядело и работало, как нам этого хочется, и мы знаем, что будет повторение определенного виджета. Мы можем создать пользовательский виджет в новом файле Flutter и определить параметры, которые нам нужны в конструкторе.

Создадим первый пользовательский виджет в файле custom_container.dart

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

class CustomContainer extends StatelessWidget {
  CustomContainer(
      {@required this.child, this.height, this.width, this.onTap, this.color});
  final Function onTap;
  final Widget child;
  final double height;
  final double width;
  final Color color;

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: onTap,
      child: Container(
        height: height,
        width: width,
        padding: EdgeInsets.all(12),
        decoration: BoxDecoration(
            color: color, borderRadius: BorderRadius.all(Radius.circular(8))),
        child: child,
      ),
    );
  }
}
```

Еще один пользовательсикй виджет создадим в custom_button.dart.

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

class CustomButton extends StatelessWidget {
  CustomButton({this.onTap, this.color = Colors.white30, this.icon});
  final Function onTap;
  final Color color;
  final IconData icon;
  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: onTap,
      child: Container(
        height: 50,
        width: 50,
        decoration: BoxDecoration(
            borderRadius: BorderRadius.circular(30), color: color),
        child: Icon(icon),
      ),
    );
  }
}
```

Последний виджет мы создадим в custom_column.dart

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

class CustomColumn extends StatelessWidget {
  CustomColumn({this.text, this.child});
  final String text;
  final Widget child;
  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        Text(
          text,
          style: TextStyle(fontSize: 18),
        ),
        child
      ],
    );
  }
}
```

- Далее мы можем импортировать наши виджеты в любой другой файл и использовать их там

```
import 'package:custom_widget_demo/widgets/custom_button.dart';
import 'package:custom_widget_demo/widgets/custom_column.dart';
import 'package:custom_widget_demo/widgets/custom_container.dart';
import 'package:flutter/material.dart';
```

Объединение виджетов для создания пользовательских интерфейсов

Используем ранее созданные нами виджеты для создания пользовательского интерфейса. Создадим виджет Home.

Виджет Home — это виджет с состоянием, и у нас есть все свойства, такие как *activeColor* , *inactiveColor* , *isSelected* для выбора пола, рост, вес и другие для расчета индекса массы тела.

Мы импортировали все наши пользовательские виджеты в файл home.dart. Помимо этого мы также создали перечисление для пола **g.**

```
enum g { m, f }



class Home extends StatefulWidget {
  @override
  _HomeState createState() => _HomeState();
}

class _HomeState extends State<Home> {
  final activeColor = Colors.white30;
  final inactiveColor = Colors.white12;
  g isSelected;
  int height = 160;
  int weight = 60;
  int age = 25;
  String bmi = '';
```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('GFG Custom Widget'),
    ),
    body: SafeArea(
      child: Container(
        padding: EdgeInsets.all(12),
        child: Column(
          children: [
            Row(
              children: [
                Expanded(
                  child: CustomContainer(
                    color: isSelected == g.m ? activeColor : inactiveColor,
                    onTap: () {
                      setState(() {
                        isSelected = g.m;
                      });
                    },
                    child: Padding(
                      padding: const EdgeInsets.symmetric(vertical: 20.0),
                      child: Text(
                        'FEMALE',
                        textAlign: TextAlign.center,
                        style: TextStyle(fontSize: 18),
                      ),
                    ),
                  ),
                ),
                SizedBox(
                  width: 10,
                ),
                Expanded(
                  child: CustomContainer(
                    color: isSelected == g.f ? activeColor : inactiveColor,
                    onTap: () {
                      setState(() {
                        isSelected = g.f;
                      });
                    },
                    child: Padding(
                      padding: const EdgeInsets.symmetric(vertical: 20.0),
                      child: Text(
                        'MALE',
                        textAlign: TextAlign.center,
                        style: TextStyle(fontSize: 18),
                      ),
                    ),
                  ),
                ),
              ],
            ),
          ],
        ),
      ),
    );
}
```

Сначала мы определили Scaffold, в котором мы определили AppBar, *a* затем, как дочерний элемент, мы определили *SafeArea* . Теперь, переходя к компонентам, мы определили столбец, который содержит все компоненты экрана. Первый дочерний элемент столбца — это виджет строки, который содержит два виджета *CustomContainer* с функцией *onTap* для выбора пола и изменения цвета контейнера по мере того, как мы это делаем.

```
SizedBox(
        height: 10,
      ),
      CustomContainer(
        color: inactiveColor,
        height: 100,
        child: CustomColumn(
          text: 'HEIGHT $height cm',
          child: SliderTheme(
            data: SliderTheme.of(context).copyWith(
              activeTrackColor: Colors.white,
              thumbColor: Colors.green,
              overlayColor: Color(0x2900ff00),
              thumbShape:
                  RoundSliderThumbShape(enabledThumbRadius: 15.0),
              overlayShape:
                  RoundSliderOverlayShape(overlayRadius: 25.0),
            ),
            child: Slider(
              value: height.toDouble(),
              min: 120.0,
              max: 220.0,
              onChanged: (double newValue) {
                setState(() {
                  height = newValue.floor();
                });
              },
            ),
          ),
        ),
      ),
```

После установки высоты мы снова определили CustomContainer *с* неактивным цветом с помощью *CustomColumn* , который принимает текст как высоту с динамически изменяющейся высотой с помощью определенного нами слайдера. Мы установили слайдеру некоторые свойства, чтобы он выглядел в соответствии с нашим приложением.

```
SizedBox(
        height: 10,
      ),
      Row(
        children: [
          Expanded(
            child: CustomContainer(
              color: inactiveColor,
              child: CustomColumn(
                text: 'WEIGHT $weight',
                child: Padding(
                  padding: const EdgeInsets.all(8.0),
                  child: Row(
                    mainAxisAlignment: MainAxisAlignment.center,
                    children: [
                      CustomButton(
                        onTap: () {
                          setState(() {
                            weight = weight - 1;
                          });
                        },
                        icon: Icons.arrow_downward,
                      ),
                      SizedBox(
                        width: 10,
                      ),
                      CustomButton(
                        onTap: () {
                          setState(() {
                            weight = weight + 1;
                          });
                        },
                        icon: Icons.arrow_upward,
                      )
                    ],
                  ),
                ),
              ),
            ),
          ),
```

```
          SizedBox(
            width: 10,
          ),
          Expanded(
            child: CustomContainer(
              color: inactiveColor,
              child: CustomColumn(
                text: 'AGE $age',
                child: Padding(
                  padding: const EdgeInsets.all(8.0),
                  child: Row(
                    mainAxisAlignment: MainAxisAlignment.center,
                    children: [
                      CustomButton(
                        onTap: () {
                          setState(() {
                            age = age - 1;
                          });
                        },
                        icon: Icons.arrow_downward,
                      ),
                      SizedBox(
                        width: 10,
                      ),
                      CustomButton(
                        onTap: () {
                          setState(() {
                            age = age + 1;
                          });
                        },
                        icon: Icons.arrow_upward,
                      )
                    ],
                  ),
                ),
              ),
            ),
          ),
        ],
      ),
```

Установив высоту, мы определили строку с двумя *CustomContainer* , которые оба принимают *CustomColumn* с текстом Weight и Age. Оба этих контейнера имеют две кнопки в строке как дочерний элемент CustomColumn. Функционал этих кнопок заключается в увеличении или уменьшении значения веса и возраста.

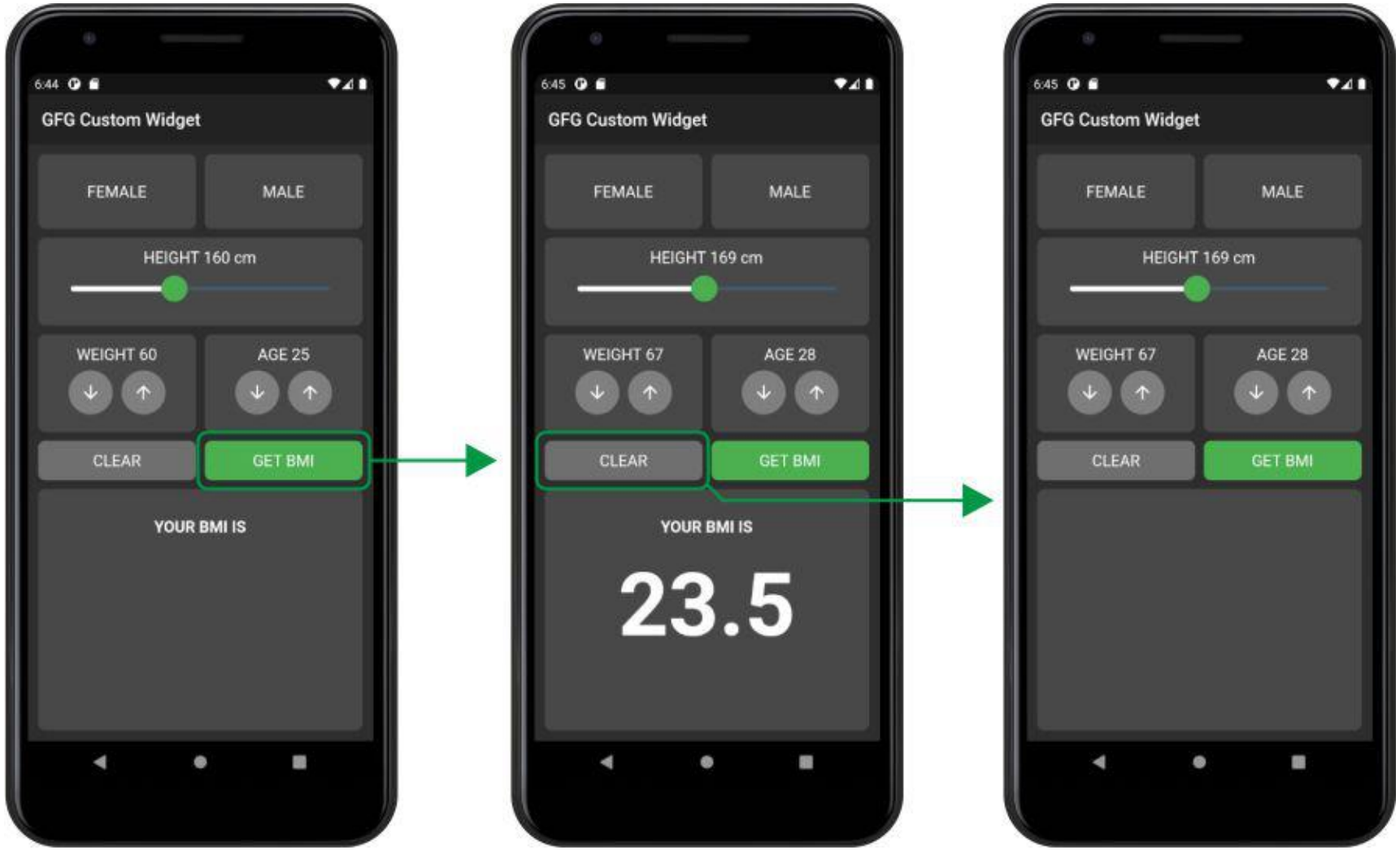
```
SizedBox(
        height: 10,
      ),
      Row(
        children: [
          Expanded(
            child: CustomContainer(
              onTap: () {
                setState(() {
                  bmi = '';
                });
              },
              width: double.infinity,
              child: Text(
                'CLEAR',
                style: TextStyle(
                  fontSize: 18,
                ),
                textAlign: TextAlign.center,
              ),
              color: activeColor,
            ),
          ),
          SizedBox(
            width: 10,
          ),
          Expanded(
            child: CustomContainer(
              onTap: () {
                double _bmi = weight / pow(height / 100, 2);

                setState(() {
                  bmi = _bmi.toStringAsFixed(1);
                });
              },
              width: double.infinity,
              child: Text(
                'GET BMI',
                style: TextStyle(
                  fontSize: 18,
                ),
                textAlign: TextAlign.center,
              ),
              color: Colors.green,
            ),
          ),
        ],
      ),
```

Здесь мы определили две кнопки с помощью нашего *CustomContainer* . Первый используется для очистки отображаемого вывода, а другой показывает вывод на экране.

```
SizedBox(
                height: 10,
              ),
              Expanded(
                child: CustomContainer(
                  width: double.infinity,
                  child: Column(
                    children: [
                      SizedBox(
                        height: 20,
                      ),
                      Text(
                        'YOUR BMI IS',
                        style: TextStyle(
                            fontSize: 18, fontWeight: FontWeight.bold),
                      ),
                      SizedBox(
                        height: 20,
                      ),
                      Text(
                        bmi,
                        style: TextStyle(
                            fontSize: 100, fontWeight: FontWeight.bold),
                      )
                    ],
                  ),
                  color: inactiveColor,
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

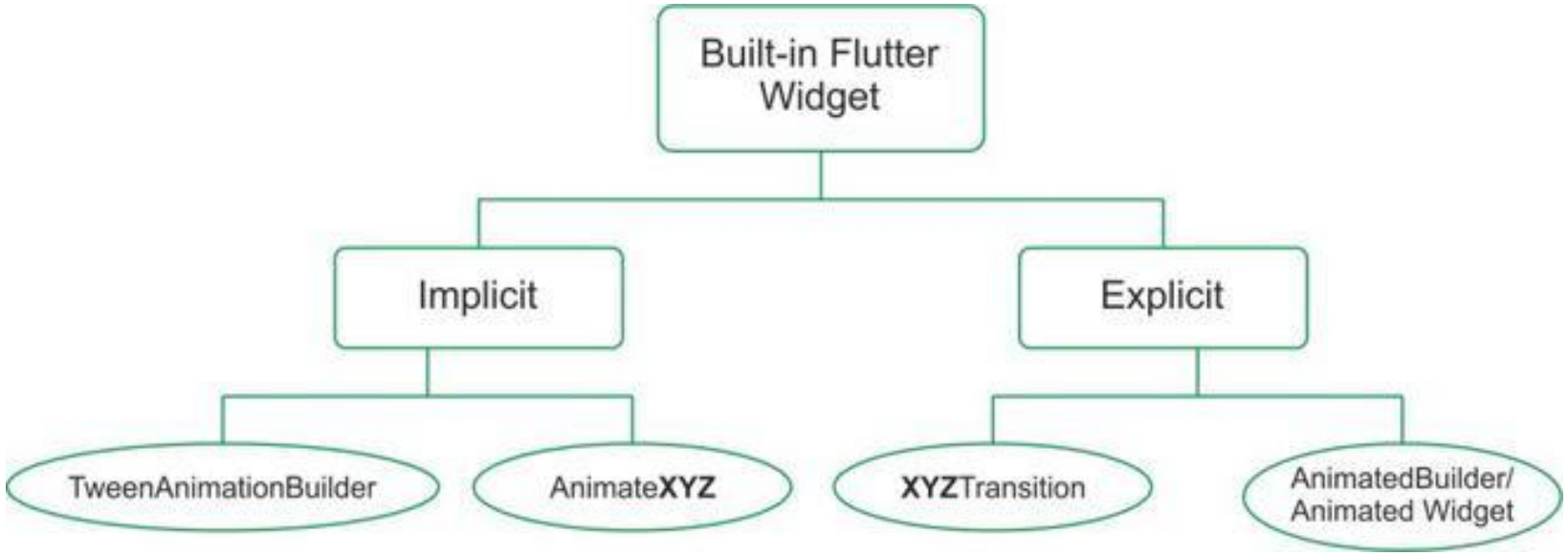
Последним компонентом приложения также является *CustomContainer*, который используется для отображения рассчитанного индекса массы тела на экране.

Результат


6:44
GFG Custom Widget
FEMALE
MALE
HEIGHT 160 cm
WEIGHT 60
AGE 25
CLEAR
GET BMI
YOUR BMI IS
6:45
GFG Custom Widget
FEMALE
MALE
HEIGHT 169 cm
WEIGHT 67
AGE 28
CLEAR
GET BMI
YOUR BMI IS
23.5
6:45
GFG Custom Widget
FEMALE
MALE
HEIGHT 169 cm
WEIGHT 67
AGE 28
CLEAR
GET BMI

Анимация во Flutter

Всякий раз, когда вы создаете приложение, анимация играет жизненно важную роль в разработке пользовательского опыта. Людям, как правило, нравятся приложения с плавным ходом и привлекательным дизайном. Пакет Flutter предоставляет множество методов для создания и использования анимации в нашем приложении. Мы обсудим встроенные виджеты Flutter для управления анимацией.



Как показано на блок-схеме, для управления анимацией во Flutter фреймворк предоставляет виджеты различной реализации. Основные свойства, присутствующие во всех виджетах анимации, — это **Duration** и **Curve** . **Duration** — это время, в течение которого виджет анимируется, а **Curve** определяет способ анимации объекта от начала до конца (поток анимации от начала до конца). Встроенные виджеты анимации во флаттере можно разделить на две основные категории: неявные и явные.

- **Неявные виджеты**

- Это самый простой виджет, предоставляемый Flutter. Эти виджеты могут быть реализованы без особых усилий со стороны разработчика. Это очень простые методы анимации, поэтому у них не так много стилей, которые можно изменить. У них есть односторонняя анимация, которая не является непрерывной. Неявные виджеты, в свою очередь, можно разделить на две категории:
- **AnimatedXYZ** : Здесь XYZ — это конкретный виджет, который можно анимировать. Это анимированные версии основных виджетов, доступных во Flutter. Вот некоторые из неявных AnimatedXYZ существующих виджетов XYZ.
  1. Align → AnimatedAlign
  2. Container → AnimatedContainer
  3. DefaultTextStyle → AnimatedDefaultTextStyle
  4. Padding → AnimatedPadding
  5. Positioned → AnimatedPositioned
- **TweenAnimationBuilder** : эти виджеты анимируют данный виджет от начального значения (*Tween.begin*) до конечного значения (*Tween.end*). Этот виджет можно использовать для анимации пользовательского виджета для простой анимации. Он принимает свойство, которое создает анимацию на основе значения, указанного в ее параметре. Мы также можем указать, что нужно сделать, когда анимация завершится, с помощью обратного вызова *onEnd* .

- **Явные виджеты**

- Эти виджеты обеспечивают более детальное управление анимированным виджетом. У них есть свойства для управления повторением и перемещением виджета. Эти виджеты требуют AnimationController для детального управления, которое они предоставляют. Этот контроллер можно определить в состоянии **initState** и **избавиться** от состояний для лучшего использования. Явный виджет можно классифицировать как
- **XYZTransition** : здесь XYZ — это особый виджет, доступный как переход. Это встроенный переход, который обеспечивает больший контроль над неявной анимацией. Их можно рассматривать как расширение виджета *AnimatedXYZ* . Некоторые доступные явные *XYZTransition :*
    1. *SizeTransition*
    2. *FadeTransition*
    3. *AlignTransition*
    4. *RotationTransition*
    5. *PositionedTransition*
    6. *DecoratedBoxTransition*
- **AnimatedBuilder/ AnimatedWidget** : когда нет доступных виджетов из предопределенного *XYZTransition* , которые уже определяют различные анимации, мы можем использовать *AnimatedBuilder / AnimatedWidget* . Они применяются к пользовательскому виджету, который мы хотим явно анимировать. Если мы можем определить анимацию в том же виджете, мы можем использовать AnimatedBuilder, в противном случае, если мы определим новый виджет, мы сможем расширить его с помощью AnimatedWidget.

Теперь создадим несколько анимаций.

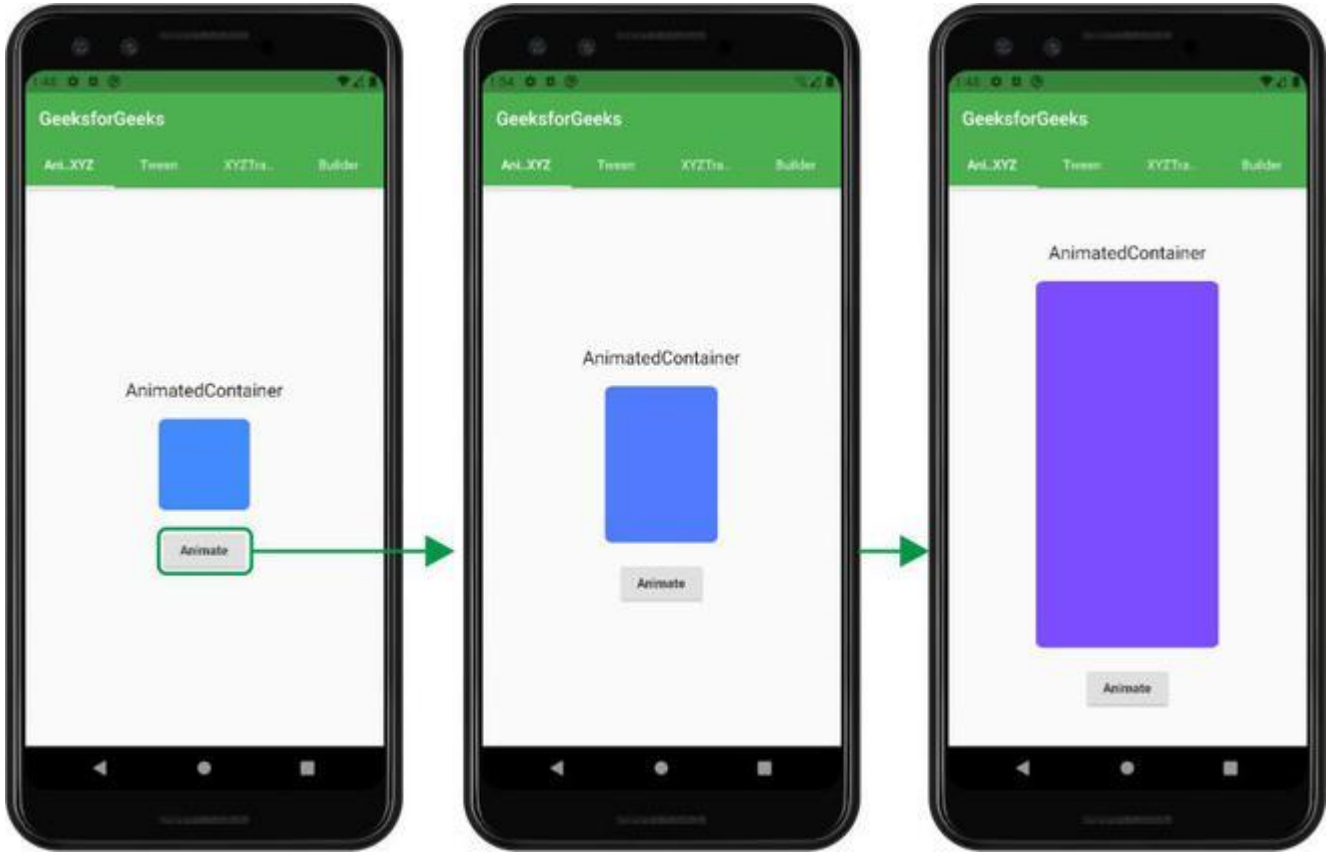
Чтобы использовать изображения, добавим в ***pubspec.yaml***

```
assets:
    - images/ #Add
```

Начнем с файла main.dart, в который импортируем, созданные нами анимации, и сформируем разметку.

```dart
import 'package:flutter/material.dart';
import 'builder_animation.dart';
import 'xyz_transition.dart';
import 'tween_animation.dart';
import 'animated_xyz.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Animation Demo',
      theme: ThemeData(
        primarySwatch: Colors.green,
      ),
      home: Home(),
    );
  }
}

class Home extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return DefaultTabController(
      length: 4,
      child: Scaffold(
        // define appbar here
        appBar: AppBar(

          // add tabs to the app
          bottom: TabBar(
            tabs: [
              Tab(text: 'Ani..XYZ'),
              Tab(text: 'Tween'),
              Tab(text: 'XYZTra..'),
              Tab(text: 'Builder'),
            ],
          ),
          title: Text('GeeksforGeeks'),
        ),
        body: TabBarView(

          // animations
          children: [
            AnimatedXYZ(),
            TweenAnimation(),
            XYZTransition(),
            BuilderAnimation(),
          ],
        ),
      ),
    );
  }
}
```

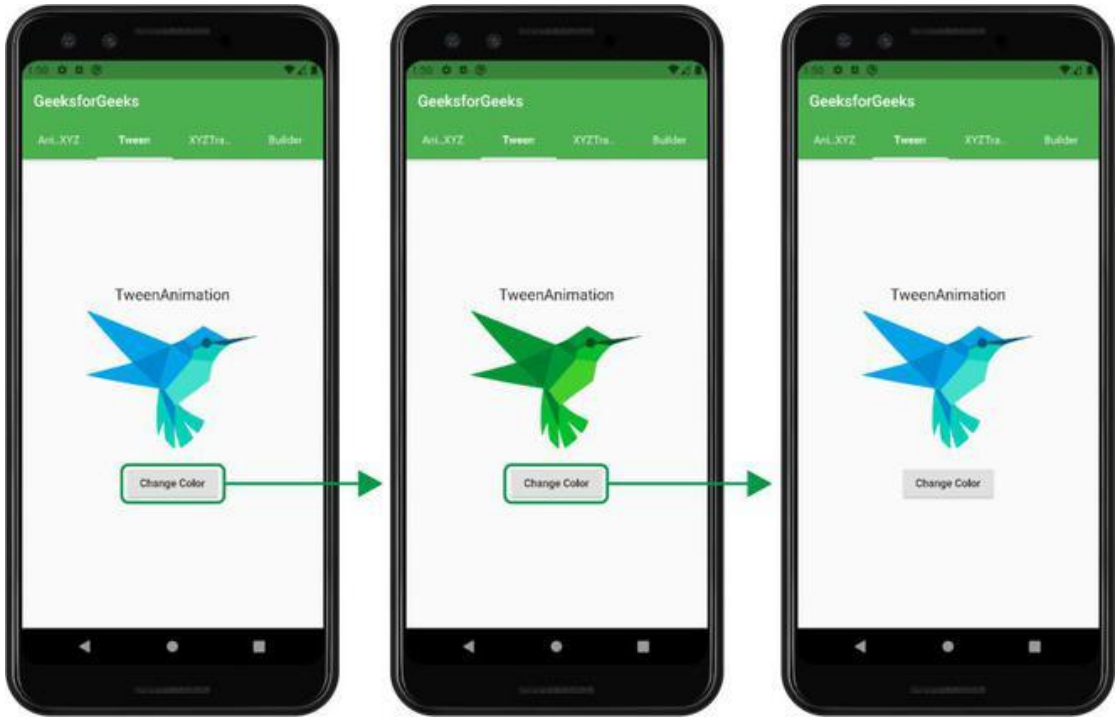
В этом файле есть столбец, содержащий различные виджеты, и AnimatedContainer , который помимо обычных функций также имеет кривую и определенную продолжительность. При нажатии кнопки размер и цвет Контейнера изменяются, как показано на рисунке:




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

class AnimatedXYZ extends StatefulWidget {
  @override
  _AnimatedXYZState createState() => _AnimatedXYZState();
}

// building the container class
class _AnimatedXYZState extends State<AnimatedXYZ> {
  bool _toggle = true;

  @override
  Widget build(BuildContext context) {
    return SafeArea(
      child: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Padding(
              padding: const EdgeInsets.all(20),
              child: Text(
                'AnimatedContainer',
                style: TextStyle(fontSize: 20),
              ),
            ),

            // using the AnimatedContainer widget
            AnimatedContainer(
              decoration: BoxDecoration(
                color: _toggle == true
                    ? Colors.blueAccent
                    : Colors.deepPurpleAccent,
                borderRadius: BorderRadius.all(Radius.circular(8)),
              ),
              curve: Curves.easeInOutBack,
              duration: Duration(seconds: 1),
              height: _toggle == true ? 100 : 400,
              width: _toggle == true ? 100 : 200,
            ),
            SizedBox(
              height: 20,
            ),
            RaisedButton(
              onPressed: () {
                setState(() {
                  _toggle = !_toggle;
                });
              },
              child: Text('Animate'),
            )
          ],
        ),
      ),
    );
  }
}
```

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

class TweenAnimation extends StatefulWidget {
  @override
  _TweenAnimationState createState() => _TweenAnimationState();
}

class _TweenAnimationState extends State<TweenAnimation> {
  Color c1 = Colors.white;
  Color c2 = Colors.yellow;
  @override
  Widget build(BuildContext context) {
    return SafeArea(
      child: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(
              'TweenAnimation',
              style: TextStyle(fontSize: 20),
            ),
            SizedBox(
              height: 10,
            ),

            // Using TweenAnimationBuilder
            TweenAnimationBuilder(
              tween: ColorTween(begin: c1, end: c2),
              duration: Duration(seconds: 1),
              builder: (_, Color color, __) {
                return ColorFiltered(

                  // image assets
                  child: Image.asset(
                    'images/bird.png',
                    height: 180,
                  ),
                  colorFilter: ColorFilter.mode(color, BlendMode.modulate),
                );
              },
            ),
            SizedBox(
              height: 20,
            ),

            // button
            RaisedButton(
              onPressed: () {
                setState(() {
                  c1 = c1 == Colors.white ? Colors.yellow : Colors.white;
                  c2 = c2 == Colors.yellow ? Colors.white : Colors.yellow;
                });
              },
              child: Text('Change Color'),
            )
          ],
        ),
      ),
    );
  }
}
```

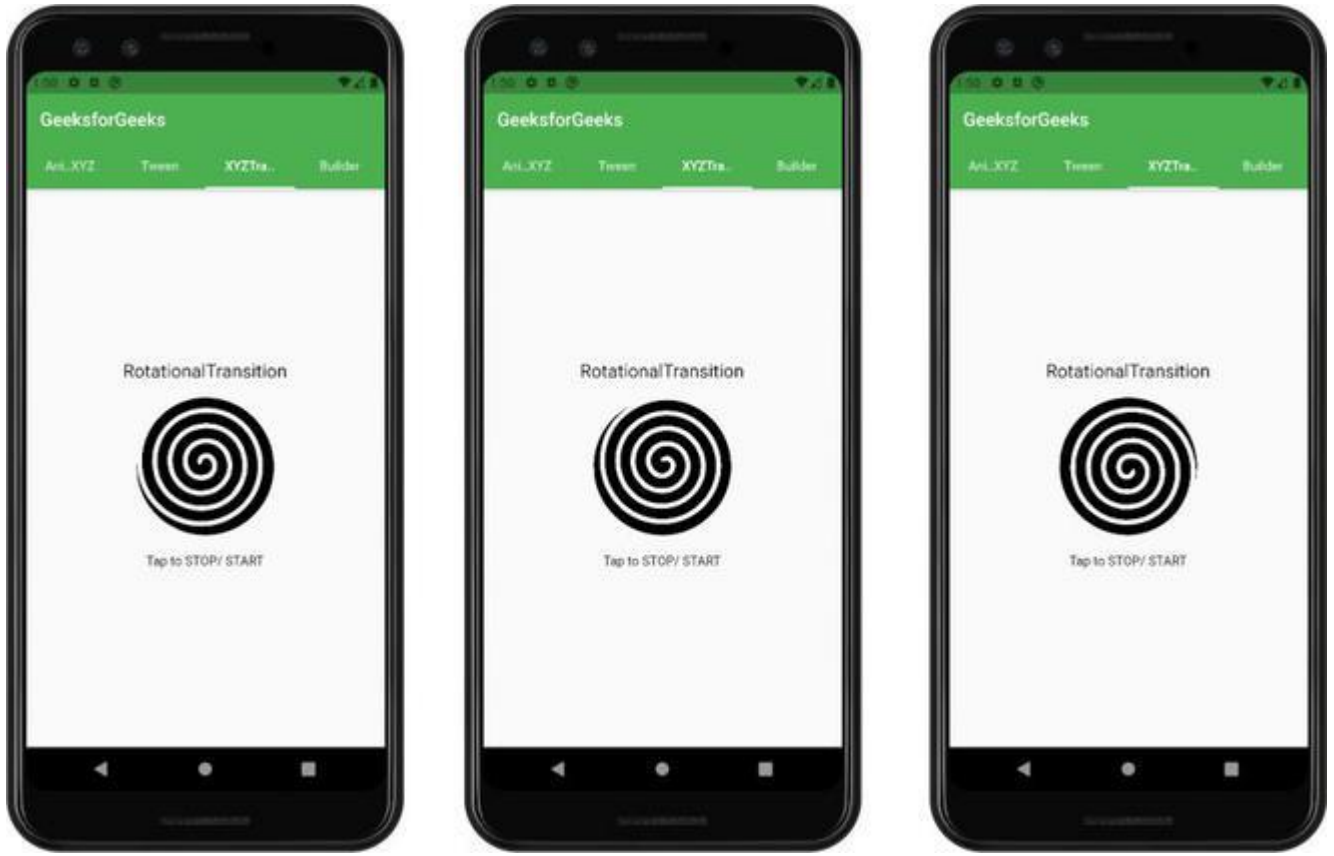
В этом файле мы просто определили столбец, который содержит различные виджеты, и помимо них имеет TweenAnimationBuilder, *который* приним ает тип твина (здесь мы использовали *ColorTween*) и продолжительность анимации. У него также есть свойство builder, которое строит виджет, предоставленный в соответствии с анимацией.



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

class XYZTransition extends StatefulWidget {
  @override
  _XYZTransitionState createState() => _XYZTransitionState();
}

class _XYZTransitionState extends State<XYZTransition>
    with SingleTickerProviderStateMixin {
  AnimationController _animationController;

  @override
  void initState() {
    super.initState();
    _animationController = AnimationController(
      vsync: this,
      duration: Duration(seconds: 3),
    )..repeat();
  }

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SafeArea(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(
              'RotationalTransition',
              style: TextStyle(fontSize: 20),
            ),
            SizedBox(
              height: 10,
            ),

            // assign action to gestures
            GestureDetector(
              onTap: () {
                _animationController.isAnimating
                    ? _animationController.stop()
                    : _animationController.repeat();
              },
              child: Padding(
                padding: const EdgeInsets.all(8.0),
                child: Center(
                  child: Column(
                    mainAxisAlignment: MainAxisAlignment.center,
                    children: [

                      // defining the animation type
                      RotationTransition(
                        child: Image.asset('images/hypno.png',
                            height: 150, width: 150),
                        alignment: Alignment.center,
                        turns: _animationController,
                      ),
                      SizedBox(
                        height: 20,
                      ),
                      Text('Tap to STOP/ START')
                    ],
                  ),
                ),
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

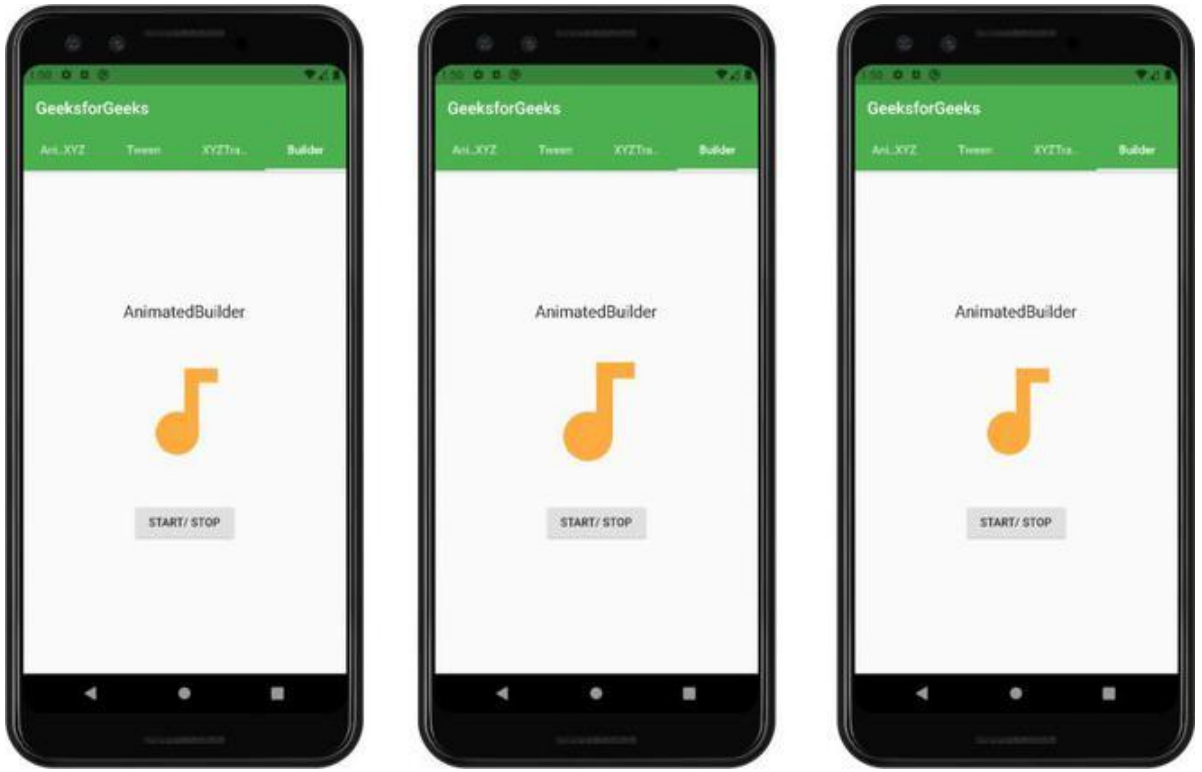
Явные виджеты предоставляют нам больше ручного управления. Чтобы получить доступ к этим элементам управления, нам нужен контроллер. Поэтому мы определяем для этого объект *AnimationController* . Нам нужно инициализировать анимацию при построении экрана, а также избавиться от нее при переходе на другой экран. Здесь мы использовали *RotationTransition* для бесконечного вращения данного изображения. Вращение можно остановить/запустить снова, нажав на изображение, как указано на экране. Переход принимает объект *AnimationController* в качестве ходов. Мы определили повторение анимации после ее завершения в *initState* .

Результат вращения



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

class BuilderAnimation extends StatefulWidget {
  @override
  _BuilderAnimationState createState() => _BuilderAnimationState();
}

class _BuilderAnimationState extends State<BuilderAnimation>
    with TickerProviderStateMixin {
  Animation _starAnimation;
  AnimationController _starAnimationController;

  bool toggle = false;

  // animation controller
  @override
  void initState() {
    super.initState();
    _starAnimationController =
        AnimationController(vsync: this, duration: Duration(milliseconds: 500));
    _starAnimation = Tween(begin: 140.0, end: 160.0).animate(CurvedAnimation(
        curve: Curves.elasticInOut, parent: _starAnimationController));

    _starAnimationController.addStatusListener((AnimationStatus status) {
      if (status == AnimationStatus.completed) {
        _starAnimationController.repeat();
      }
    });
  }

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

  @override
  Widget build(BuildContext context) {
    return SafeArea(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          Text(
            'AnimatedBuilder',
            style: TextStyle(fontSize: 20),
          ),
          SizedBox(
            height: 10,
          ),

          // animated container
          // goes as a child
          Container(
            height: 200,
            width: 200,
            child: AnimatedBuilder(
              animation: _starAnimationController,
              builder: (context, child) {
                return Center(
                  child: Container(
                    child: Center(
                      child: Icon(
                        Icons.audiotrack,
                        color: Colors.orangeAccent,
                        size: _starAnimation.value,
                      ),
                    ),
                  ),
                );
              },
            ),
          ),
          SizedBox(
            height: 10,
          ),

          // button
          RaisedButton(
            child: Text('START/ STOP'),
            onPressed: () {
              toggle = !toggle;
              toggle == true
                  ? _starAnimationController.forward()
                  : _starAnimationController.stop();
            },
          ),
        ],
      ),
    );
  }
}
```

- Этот виджет показывает значок музыкальной дорожки, который постоянно увеличивается. Большинство деталей остаются такими же, как в явном *XYZTransition,* определенном выше. Но здесь мы должны передать билдер с дочерним элементом и контекстом. Здесь мы должны передать контроллер в свойство анимации *AnimatedBuilder* . Кнопка используется для запуска и остановки анимации.



Итоговый результат