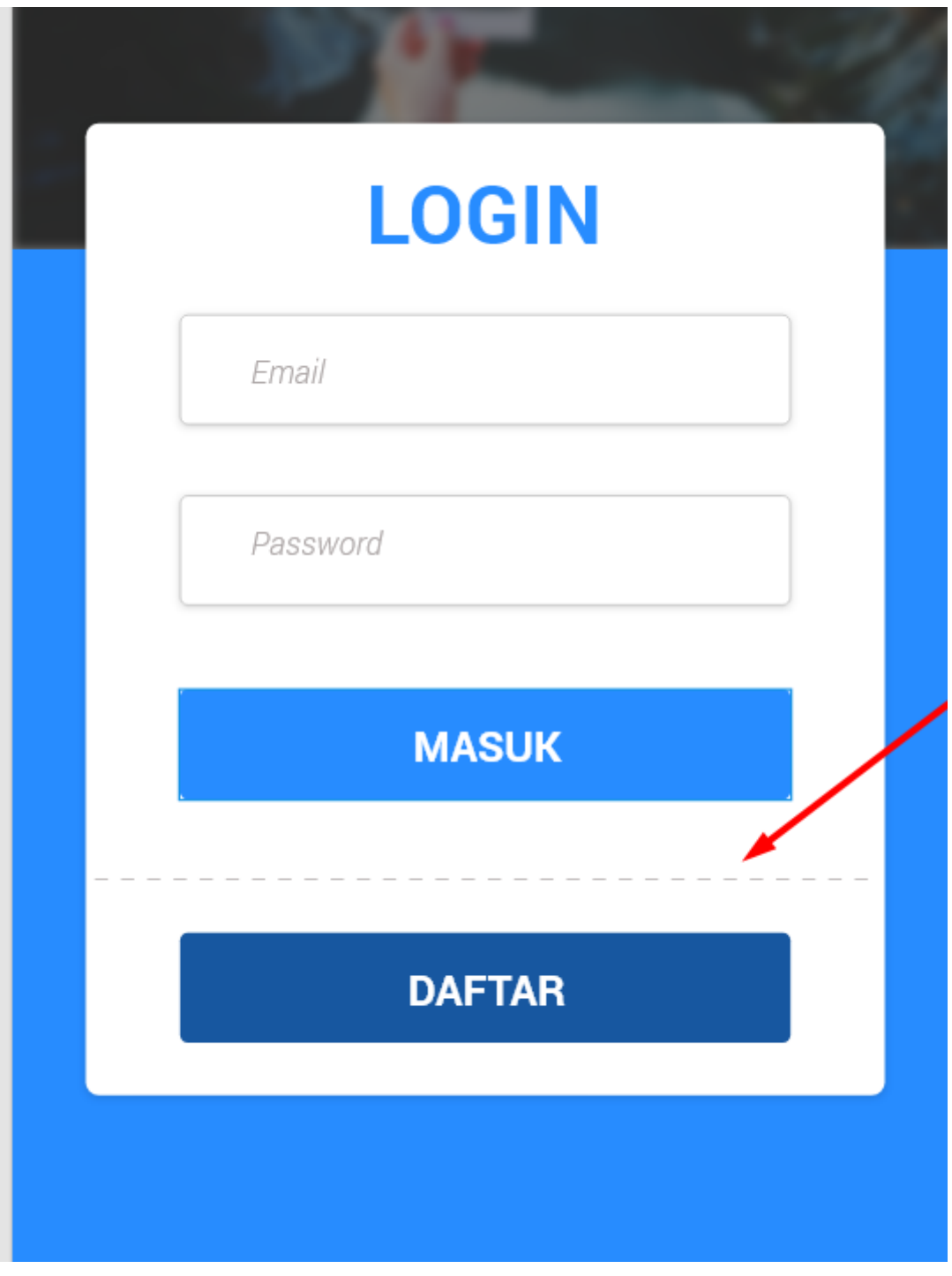
How to add line dash in Flutter

Asked 2 years, 11 months ago Active 2 months ago Viewed 34k times

32

8

How to make a line dash in Flutter like this?



flutter flutter-layout

Share Follow

edited Jul 3 '19 at 18:41
CopsOnRoad
147k 42 473 340

asked Jan 3 '19 at 9:48
Badai Ardiat
457 1 5 16

Did you try something? Can you add that thing too? – surajs1n Jan 3 '19 at 9:59

3 github.com/flutter/flutter/issues/4858 – Günter Zöchbauer Jan 3 '19 at 10:07

4 try Text('----------') – Shyju M Jan 4 '19 at 4:23

1 if you want to draw it as a dashed path on the canvas use this package: pub.dartlang.org/packages/path_drawing – Tarek360 Jan 4 '19 at 7:28

18 Answers

Active Oldest Votes

50

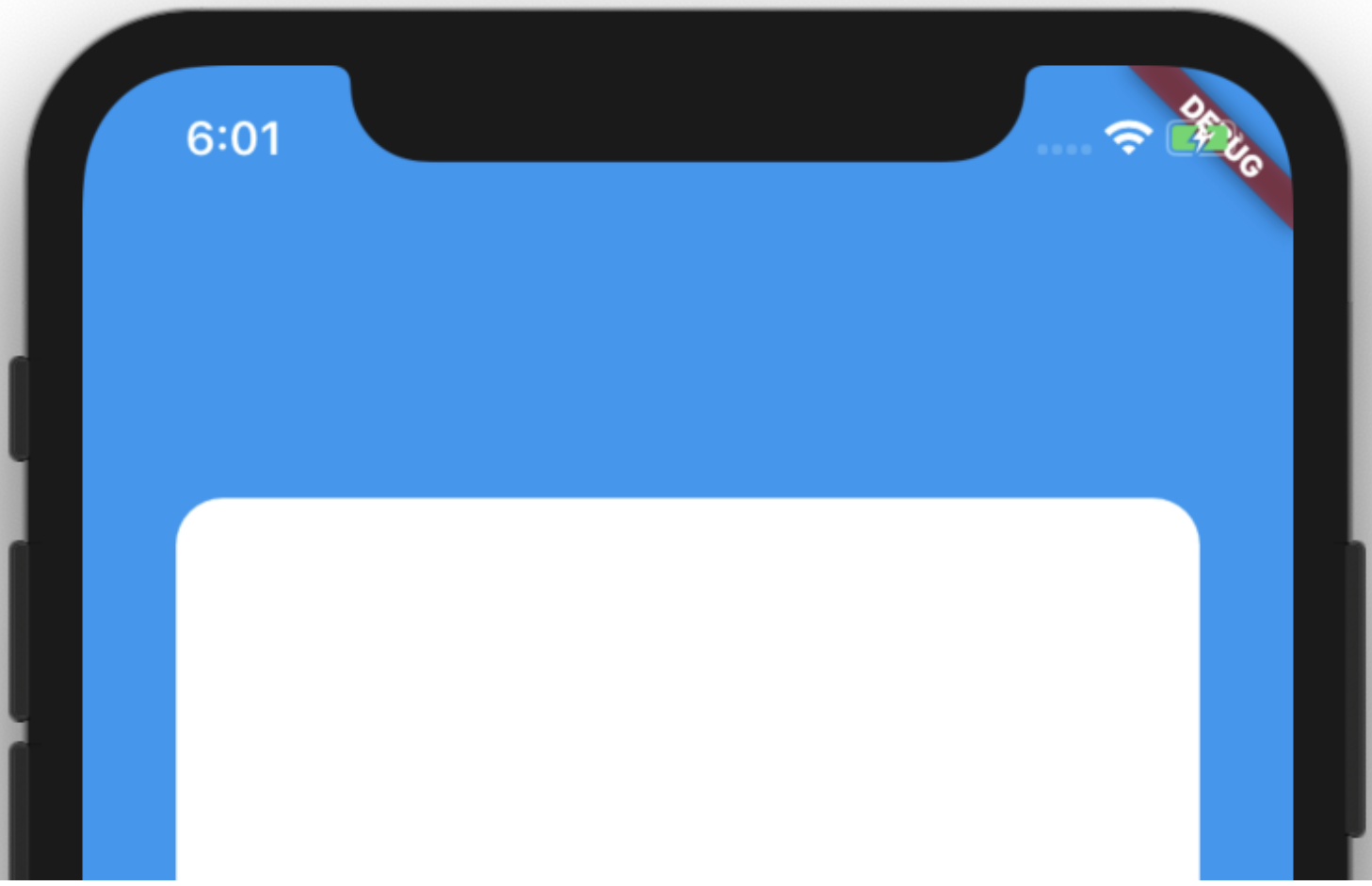
As a workaround, in your case, you can do something like this

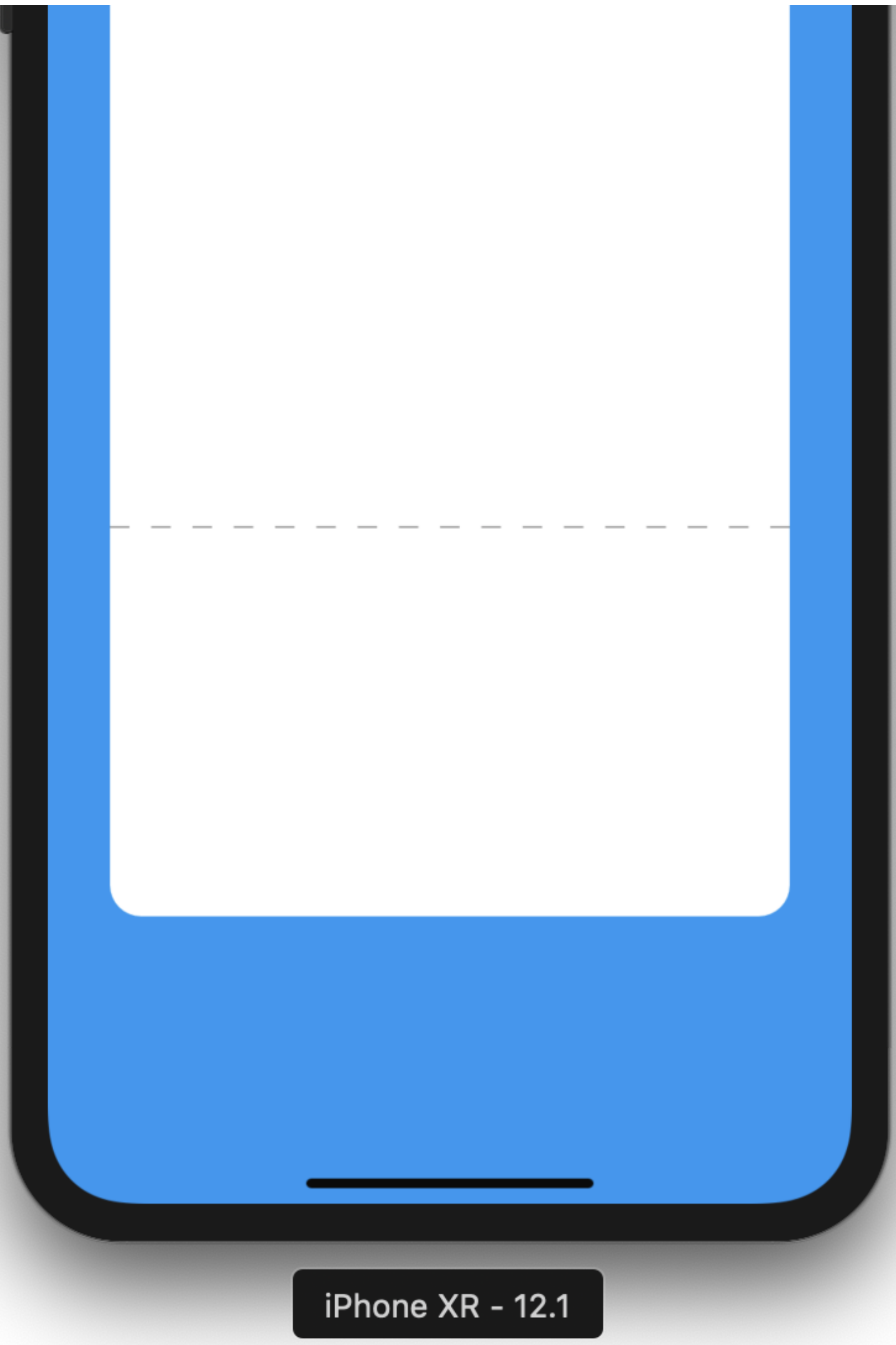
```
class MySeparator extends StatelessWidget {
  final double height;
  final Color color;

  const MySeparator({this.height = 1, this.color = Colors.black});

  @override
  Widget build(BuildContext context) {
    return LayoutBuilder(
      builder: (BuildContext context, BoxConstraints constraints) {
        final boxWidth = constraints.constrainWidth();
        final dashWidth = 10.0;
```

```
        final dashHeight = height;
        final dashCount = (boxWidth / (2 * dashWidth)).floor();
        return Flex(
          children: List.generate(dashCount, (_) {
            return SizedBox(
              width: dashWidth,
              height: dashHeight,
              child: DecoratedBox(
                decoration: BoxDecoration(color: color),
              ),
            );
          }),
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          direction: Axis.horizontal,
        );
      },
    );
  }
}
```

and use it `const MySeparator()`

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Material(
        child: Container(
          color: Colors.blue,
          child: Center(
            child: Container(
              height: 600, width: 350,
              decoration: BoxDecoration(
                color: Colors.white,
                borderRadius: BorderRadius.all(Radius.circular(16.0)),
              ),
              child: Flex(
                direction: Axis.vertical,
                children: [
                  Expanded(child: Container()),
                  const MySeparator(color: Colors.grey),
                  Container(height: 200),
                ],
              ),
            ),
          ),
        ),
      ),
    );
  }
}
```





Share Follow

edited Jan 3 '19 at 15:05

answered Jan 3 '19 at 14:58

maksimr
2,984 1 20 19

26

```
class DashedLinePainter extends CustomPainter {
  @override
  void paint(Canvas canvas, Size size) {
    double dashWidth = 9, dashSpace = 5, startX = 0;
    final paint = Paint()
      ..color = Colors.grey
      ..strokeWidth = 1;
    while (startX < size.width) {
      canvas.drawLine(Offset(startX, 0), Offset(startX + dashWidth, 0), paint);
      startX += dashWidth + dashSpace;
    }
  }

  @override
```

```
  bool shouldRepaint(CustomPainter oldDelegate) => false;
}
```

Share Follow

answered Nov 13 '19 at 14:10
Anton Duzenko
1,902 1 19 22

Use it like CustomPaint(painter: DashedLinePainter()); – B.shruti Apr 6 at 5:51

15

```
// garis putus putus
Row(
children: List.generate(150~/10, (index) => Expanded(
 child: Container(
  color: index%2==0?Colors.transparent
  :Colors.grey,
  height: 2,
 ),
 )),
),
```

Share Follow

answered Mar 11 '20 at 5:04
malik kurosaki
602 6 8

1 While this code may answer the question, providing additional context regarding why and/or how this code answers the question improves its long-term value. – Igor F. Mar 11 '20 at 9:21

this is the simplest answer to implement without any dependency. – prem pattnaik May 10 '20 at 3:26

1 Looks good trick but why 150~/10 ? why not you used just a number like 15, 16, 17.... ? – Deven Nov 1 at 11:50

@Deven for example, you want to apply the width of the widget or screen, then divide by how many lines you want and the "~" sign to give an integer value, and if you want it to be even simpler, you can use your suggestion with fixed numbers like 5, 6,10 and so on – malik kurosaki Nov 3 at 11:06

12

CustomPainter can help here as well. In this example is a vertical dash line but could be changed easily.

```
class LineDashedPainter extends CustomPainter {
  @override
  void paint(Canvas canvas, Size size) {
    var paint = Paint()..strokeWidth = 2;
    var max = 35;
    var dashWidth = 5;
    var dashSpace = 5;
    double startY = 0;
    while (max >= 0) {
      canvas.drawLine(Offset(0, startY), Offset(0, startY + dashWidth), paint);
      final space = (dashSpace + dashWidth);
      startY += space;
      max -= space;
    }
  }

  @override
  bool shouldRepaint(CustomPainter oldDelegate) => false;
}
```

and that use CustomPaint Widget:

```
CustomPaint(painter: LineDashedPainter())
```

Share Follow

answered May 5 '19 at 15:06
etzuk
338 3 9

11

I have written flutter_dash library for drawing that dash. Just one line and you should have a dash :D

```
Dash(length: 200, dashColor: Colors.red)
```

Give it a try!

Share Follow

edited Dec 1 '19 at 9:45

answered Sep 12 '19 at 2:03
Lê Vũ Huy

556 5 16

Grateful for me! Thanks a lot :) – Andres Paladines Feb 15 '20 at 22:27

1 I don't recomend, this library is not well optimized and will cause your app to be slower to load screens – Soufiane Ghzal Apr 19 '20 at 21:06

@SoufianeGhzal please give me some advices to optimize it, thank you – Lê Vũ Huy Apr 21 '20 at 11:51

1 @LêVũHuy I don't really know what's the issue. go to github please github.com/huy-lv/flutter_dash/issues/2 – Soufiane Ghzal Apr 21 '20 at 12:54

2

Vertical dashed line:

I modifed maksimr's example:

```
class DashedLine extends StatelessWidget {
  final double height;
  final double heightContainer;
  final Color color;

  const DashedLine({this.height = 3, this.color = Colors.black, this.heightContainer = 70});

  @override
  Widget build(BuildContext context) {
    return Container(
      height: heightContainer,
      child: LayoutBuilder(
        builder: (BuildContext context, BoxConstraints constraints) {
          final boxHeight = constraints.constrainHeight();
          final dashWidth = 10.0;
          final dashHeight = height;
          final dashCount = (boxHeight / (2 * dashHeight)).floor();
          return Flex(
            children: List.generate(dashCount, (_) {
              return SizedBox(
                width: dashWidth,
                height: dashHeight,
                child: DecoratedBox(
                  decoration: BoxDecoration(color: color),
                ),
              );
            }),
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            direction: Axis.vertical,
          );
        },
      ),
    );
  }
}
```

Share Follow

answered Jul 9 '20 at 21:14

Paweł Zawiślak

87 1 4

2

I created a `CustomPainter` by integrating the solution here and the math from here. This `CustomPainter` allows to draw a solid line or a dashed line by specifying the length of the dash and the length of the space between dashes. But the best thing is you can even draw the solid or dashed line in all directions. I mean horizontal, vertical, and even diagonal!

This is the code for the `CustomPainter` :

```
import 'dart:math';

import 'package:flutter/material.dart';

class LinePainter extends CustomPainter {
  final Offset firstOffset;
  final Offset secondOffset;
  final Color color;
  final double strokeWidth;
  final double dashLength;
  final double dashSpace;

  const LinePainter({
    required this.firstOffset,
    required this.secondOffset,
    this.color = Colors.black,
    this.strokeWidth = 2.0,
    this.dashLength = 4.0,
    this.dashSpace = 4.0,
  });
```

```
@override
void paint(Canvas canvas, Size size) {
  final paint = Paint()
    ..color = color
    ..strokeWidth = strokeWidth;
  _drawDashedLine(
      dashLength, dashSpace, firstOffset, secondOffset, canvas, size, paint);
}

@override
bool shouldRepaint(covariant CustomPainter oldDelegate) {
  return false;
}

void _drawDashedLine(double dashLength, double dashSpace, Offset firstOffset,
    Offset secondOffset, Canvas canvas, Size size, Paint paint) {
  var startOffset = firstOffset;

  var intervals = _getDirectionVector(firstOffset, secondOffset).length /
      (dashLength + dashSpace);

  for (var i = 0; i < intervals; i++) {
    var endOffset = _getNextOffset(startOffset, secondOffset, dashLength);

    /// Draw a small line.
    canvas.drawLine(startOffset, endOffset, paint);

    /// Update the starting offset.
    startOffset = _getNextOffset(endOffset, secondOffset, dashSpace);
  }
}

Offset _getNextOffset(
  Offset firstOffset,
  Offset secondOffset,
  double smallVectorLength,
) {
  var directionVector = _getDirectionVector(firstOffset, secondOffset);

  var rescaleFactor = smallVectorLength / directionVector.length;
  if (rescaleFactor.isNaN || rescaleFactor.isInfinite) {
    rescaleFactor = 1;
  }

  var rescaledVector = Offset(directionVector.vector.dx * rescaleFactor,
      directionVector.vector.dy * rescaleFactor);

  var newOffset = Offset(
      firstOffset.dx + rescaledVector.dx, firstOffset.dy + rescaledVector.dy);

  return newOffset;
}

DirectionVector _getDirectionVector(Offset firstVector, Offset secondVector) {
  var directionVector = Offset(
      secondVector.dx - firstVector.dx, secondVector.dy - firstVector.dy);

  var directionVectorLength =
          sqrt(pow(directionVector.dx, 2) + pow(directionVector.dy, 2));

    return DirectionVector(
      vector: directionVector,
      length: directionVectorLength,
    );
  }
}

class DirectionVector {
  final Offset vector;
  final double length;

  const DirectionVector({
    required this.vector,
    required this.length,
  });
}
```

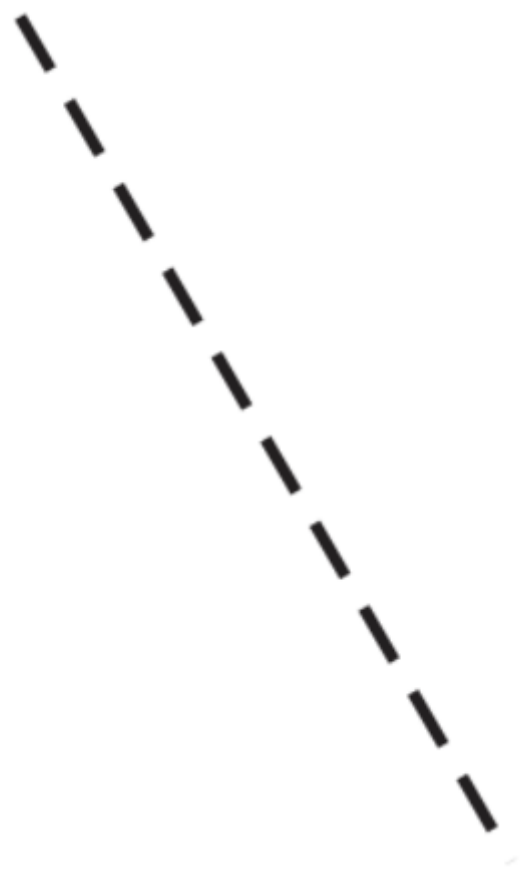
Run code snippet Expand snippet

You can use this `CustomPainter` by setting up the `painter` parameter of a `CustomPaint` widget, like this:

```
CustomPaint(
  painter: LinePainter(
    firstOffset: Offset(0, 0),
    secondOffset: Offset(10, 10),
  ),
),
```

Run code snippet Expand snippet

The result is shown in the following image:



Share Follow

answered May 22 at 19:04
Abel Rodríguez
156 3

1

Here is the code for horizontal dashed line, like your image. **CustomPaint** is highly recommended by flutter team for stuff like this. It is fast and efficient for rendering also. You can play with **Offset** to change the direction.

```
class MyClass extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return Center(
   child: CustomPaint(
    painter: MyLinePainter(),
   ),
  );
 }
}

class MyLinePainter extends CustomPainter {
 @override
 void paint(Canvas canvas, Size size) {
  var max = 100;
  var dashWidth, dashSpace = 5;
  double startX = 0;
  final paint = Paint()..color = Colors.grey;
  while (max >= 0) {
   canvas.drawLine(Offset(startX, 0), Offset(startX + dashWidth, 0), paint..strokeWidth = 1);
   final space = (dashSpace + dashWidth);
   startX += space;
   max -= space;
  }
 }
```

Share Follow

answered Aug 5 '19 at 8:21
Simran Singh
647 4 8

1

Create this class:

```
class DotWidget extends StatelessWidget {
  final double totalWidth, dashWidth, emptyWidth, dashHeight;

  final Color dashColor;

  const DotWidget({
    this.totalWidth = 300,
    this.dashWidth = 10,
    this.emptyWidth = 5,
    this.dashHeight = 2,
    this.dashColor = Colors.black,
    Key key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Row(
      mainAxisSize: MainAxisSize.min,
      children: List.generate(
        totalWidth ~/ (dashWidth + emptyWidth),
            (_) => Container(
          width: dashWidth,
          height: dashHeight,
          color: dashColor,
          margin: EdgeInsets.only(left: emptyWidth / 2, right: emptyWidth / 2),
        ),
      ),
    );
  }
}
```

Usage:

Use it like any other widget

```
child: DotWidget(
  dashColor: Colors.black,
  dashHeight: 2,
  dashWidth: 100,
)
```

Share Follow

edited Feb 29 '20 at 12:07

answered Jul 3 '19 at 18:39

CopsOnRoad
147k 42 473 340

1

You can use **CustomPainter** with a **linear gradient dashed shader** for your lines.

```
// GradientRotation(3.14 / 2) — for vertical lines with dashes
// GradientRotation(0) — for horizontal lines with dashes
// .createShader(Rect.fromLTWH(0, 0, 10, 10) — 10 is the size of repeated shaders part
// This method can be tricky if you need a line oriented by some angle.

Paint()..shader = LinearGradient(
                colors: [Colors.blue, Colors.transparent],
                stops: [0.5, 0.5],
                tileMode: TileMode.repeated,
                transform: GradientRotation(3.14 / 2))
            .createShader(Rect.fromLTWH(0, 0, 10, 10))
        ..style = PaintingStyle.stroke
        ..strokeWidth = 6
```

Share Follow

answered Mar 11 at 19:26

Dmitry_Kovalov
1,176 8 9

You can use this:

1

```
Widget dashedHorizontalLine(){
  return Row(
    children: [
      for (int i = 0; i < 20; i++)
        Expanded(
          child: Row(
            children: [
              Expanded(
                child: Divider(
                  color: AppColors.darkGreen,
                  thickness: 2,
                ),
              ),
              Expanded(
                child: Container(),
              ),
            ],
          ),
        ),
    ],
  );
}
```

Share Follow

answered Mar 16 at 19:14

Hrvoje Čukman
71 2 8

1

```
Container(
            color: Colors.white,
            height: 40.0,
            child: Center(
              child: Text(
                "--------------------------------------------------------------------------",
                maxLines: 1,
                style: typoNormalTextRegular.copyWith(
                    color: colorABGray),
              ),
            ),
          ),
```

Only use Text Widget, easy solution

Share Follow

answered Jul 16 at 7:18

Bao Bao
121 7

1

Thank to marksimr answer, here is the code for both vertical and horizontal dash line.

Horizontal usage:

```
DashLineView(
  fillRate: 0.7,
),
```

Vertical usage:

```
DashLineView(
  fillRate: 0.7,
  direction: Axis.vertical,
),
```

Full code:

```
class DashLineView extends StatelessWidget {
  final double dashHeight;
  final double dashWith;
  final Color dashColor;
  final double fillRate; // [0, 1] totalDashSpace/totalSpace
  final Axis direction;

  DashLineView(
      {this.dashHeight = 1,
      this.dashWith = 8,
      this.dashColor = Colors.black,
```

```
    this.fillRate = 0.5,
    this.direction = Axis.horizontal});

  @override
  Widget build(BuildContext context) {
    return LayoutBuilder(
      builder: (BuildContext context, BoxConstraints constraints) {
        final boxSize = direction == Axis.horizontal
            ? constraints.constrainWidth()
            : constraints.constrainHeight();
        final dCount = (boxSize * fillRate / dashWith).floor();
        return Flex(
          children: List.generate(dCount, (_) {
            return SizedBox(
              width: direction == Axis.horizontal ? dashWith : dashHeight,
              height: direction == Axis.horizontal ? dashHeight : dashWith,
              child: DecoratedBox(
                decoration: BoxDecoration(color: dashColor),
              ),
            );
          }),
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          direction: direction,
        );
      },
    );
  }
}
```

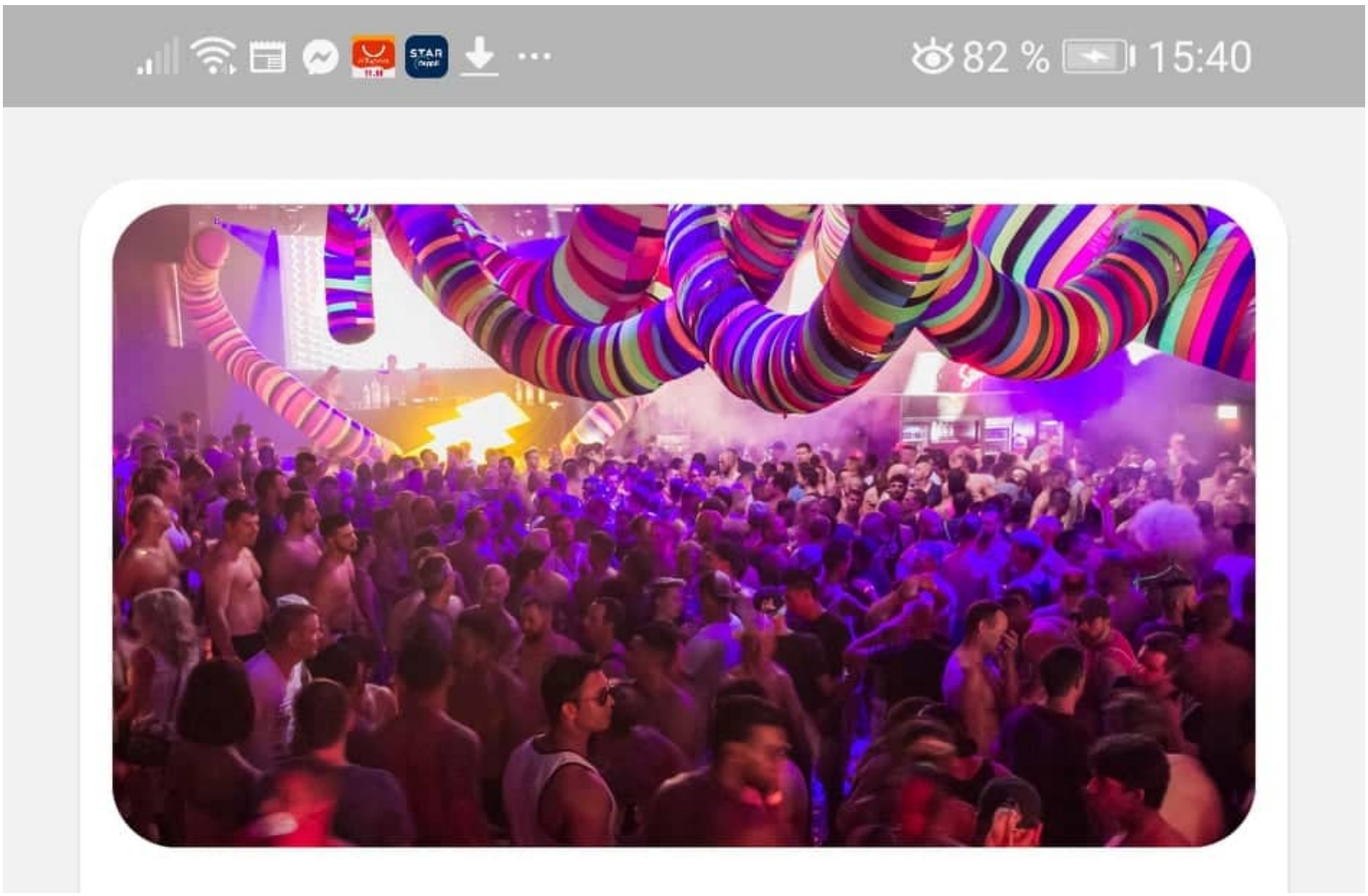
Share Follow

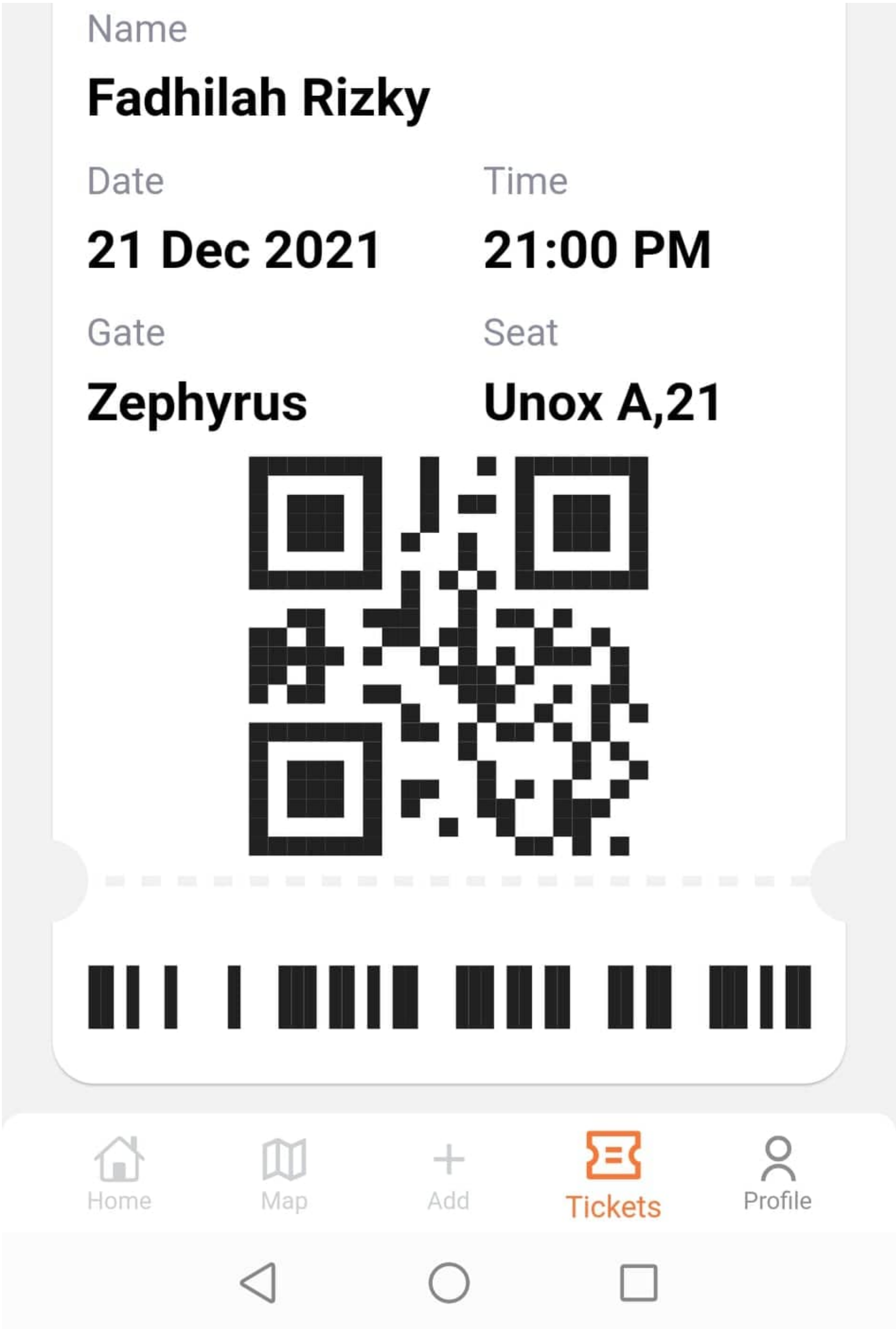
answered Oct 14 at 7:38

Liar
1,155 1 8 19

1

```
Row(
  children: List.generate(20, (index) {
            return Expanded(
              child: Padding(
                padding: const EdgeInsets.only(left: 8.0),
                child: Container(
                  height: 5,
                  width: 10,
                  color: Color(0XFFf2f2f2),
                ),
              ),
            );
          }),
        )
```





Share Follow

answered Oct 29 at 13:51

Tokiniaina Eddy Andriamiandris

624 7 6

0

Try this,

```
class DotDivider extends StatelessWidget {
  final double width;
  final double height;
  final double gap;
  final Color color;
  final double lineHeight;

  const DotDivider(
      {this.height = 1.0,
      this.color = Colors.black,
      this.width = 2.0,
      this.gap = 2.0,
      this.lineHeight = 10.0});

  @override
  Widget build(BuildContext context) {
    return LayoutBuilder(
      builder: (BuildContext context, BoxConstraints constraints) {
        final boxWidth = constraints.constrainWidth();
        final dashWidth = width;
        final dashHeight = height;
        final dashCount = (boxWidth / dashWidth).floor();
        return Container(
          height: (lineHeight * 2) + height,
          child: ListView.builder(
            physics: NeverScrollableScrollPhysics(),
            scrollDirection: Axis.horizontal,
            itemCount: dashCount,
            itemBuilder: (BuildContext context, int index) => Center(
              child: Container(
                width: dashWidth,
                height: dashHeight,
                margin:
                    EdgeInsets.symmetric(vertical: lineHeight, horizontal: gap),
                decoration: BoxDecoration(color: color),
              ),
            ),
          ),
        );
      },
    );
  }
}
```

Share Follow

answered Oct 11 '19 at 7:26

Raj Yadav
6,772 3 30 27

0

You should prefer using **CustomPainter** because it's more performance and suitable for such issues.

```
class DashLine extends StatelessWidget {
  const DashLine({
    Key key,
    this.color,
    this.dashWidth,
    this.dashSpace,
    this.strokeWidth,
  }) : super(key: key);

  final Color color;
  final double dashWidth;
  final double dashSpace;
  final double strokeWidth;

  @override
  Widget build(BuildContext context) {
    return CustomPaint(
      painter: _DashLinePainter(
        color: color,
        dashWidth: dashWidth,
        dashSpace: dashSpace,
        strokeWidth: strokeWidth,
      ),
    );
  }
}

class _DashLinePainter extends CustomPainter {
  _DashLinePainter({
```

```
  Color color,
  double dashWidth,
  double dashSpace,
  double strokeWidth,
})  : _color = color ?? Colors.red,
      _dashWidth = dashWidth ?? 5.0,
      _dashSpace = dashSpace ?? 5.0,
      _strokeWidth = strokeWidth ?? 1.0;

final Color _color;
final double _dashWidth;
final double _dashSpace;
final double _strokeWidth;

@override
void paint(Canvas canvas, Size size) {
  final paint = Paint()
    ..color = _color
    ..strokeWidth = _strokeWidth;

  var max = size.width;
  var startX = 0.0;
  while (max >= 0) {
    canvas.drawLine(Offset(startX, 0), Offset(startX + _dashWidth, 0), paint);
    final space = (_dashSpace + _dashWidth);
    startX += space;
    max -= space;
  }
}

@override
bool shouldRepaint(_DashLinePainter oldDelegate) {
  return _color != oldDelegate._color ||
      _dashWidth != oldDelegate._dashWidth ||
      _dashSpace != oldDelegate._dashSpace ||
      _strokeWidth != oldDelegate._strokeWidth;
}
}
```

Share Follow

answered Sep 17 '20 at 13:44
Denis Chuvasov
1

0

Use dotted_line: ^3.0.0 lib which provides dashed lines and many more link

```
import 'package:dotted_line/dotted_line.dart';

DottedLine(
  direction: Axis.horizontal,
  lineLength: double.infinity,
  lineThickness: 1.0,
  dashLength: 4.0,
  dashColor: Colors.grey,
  dashRadius: 0.0,
  dashGapLength: 4.0,
  dashGapColor: Colors.transparent,
  dashGapRadius: 0.0,
)
```

Output:

Share Follow

answered May 26 at 18:00
Jitesh Mohite
20.8k 8 99 107

0

I came up with this solution.

```
Row(                    // Dashed line
    children: [
      for (int i = 0; i < 25; i++)
        Container(
          width: 5,
          height: 1,
          decoration: BoxDecoration(
            border: Border(
              bottom: BorderSide(
                width: 1,
```

```
                color: i % 2 == 0
                    ? const Color.fromRGBO(214, 211, 211, 1)
                    : Colors.transparent,
              ),
            ),
          ),
        ),
      ],
    ),
```

Output:

Share Follow

answered Sep 27 at 6:24

Ritik Saxena
31 5