// Automatic FlutterFlow imports

import '/backend/schema/structs/index.dart';

import '/flutter\_flow/flutter\_flow\_theme.dart';

import '/flutter\_flow/flutter\_flow\_util.dart';

import '/custom\_code/widgets/index.dart'; // Imports other custom widgets

import '/custom\_code/actions/index.dart'; // Imports custom actions

import '/flutter\_flow/custom\_functions.dart'; // Imports custom functions

import 'package:flutter/material.dart';

// Begin custom widget code

// DO NOT REMOVE OR MODIFY THE CODE ABOVE!

import 'dart:ui';

class DashedContainer extends StatefulWidget {

  const DashedContainer({

    super.key,

    this.width,

    this.height,

    this.borderColor,

    this.borderWidth,

    this.borderStyle = 'dashed',

    this.borderRadius,

    this.backgroundColor,

    this.paddingLeft,

    this.paddingRight,

    this.paddingTop,

    this.paddingBottom,

    this.titleText,

    this.titleTextColor,

    this.titleTextFontSize,

    this.titleTextFontWeight,

    this.titleTextAlign,

    this.describText,

    this.describTextColor,

    this.describTextFontSize,

    this.describTextFontWeight,

    this.describTextAlign,

    this.marginBetweenTextAndImage,

    this.marginBetweenTitleAndDescrib,

    this.image,

    this.imageWidth,

    this.imageHeight,

    this.imageBorderRadius,

    this.imageFit,

    this.marginLeft,

    this.marginRight,

    this.marginTop,

    this.marginBottom,

  });

  final double? width;

  final double? height;

  final Color? borderColor;

  final double? borderWidth;

  final String borderStyle;

  final double? borderRadius;

  final Color? backgroundColor;

  final double? paddingLeft;

  final double? paddingRight;

  final double? paddingTop;

  final double? paddingBottom;

  final String? titleText;

  final Color? titleTextColor;

  final double? titleTextFontSize;

  final String? titleTextFontWeight;

  final String? titleTextAlign;

  final String? describText;

  final Color? describTextColor;

  final double? describTextFontSize;

  final double? describTextFontWeight;

  final String? describTextAlign;

  final double? marginBetweenTextAndImage;

  final double? marginBetweenTitleAndDescrib;

  final String? image;

  final double? imageWidth;

  final double? imageHeight;

  final double? imageBorderRadius;

  final String? imageFit;

  final double? marginLeft;

  final double? marginRight;

  final double? marginTop;

  final double? marginBottom;

  @override

  State<DashedContainer> createState() => \_DashedContainerState();

}

class \_DashedContainerState extends State<DashedContainer> {

  @override

  Widget build(BuildContext context) {

    return Container(

      width: widget.width ?? 300,

      height: widget.height ?? 150,

      margin: EdgeInsets.only(

        left: widget.marginLeft ?? 16,

        right: widget.marginRight ?? 16,

        top: widget.marginTop ?? 8,

        bottom: widget.marginBottom ?? 8,

      ),

      padding: EdgeInsets.only(

        left: widget.paddingLeft ?? 16,

        right: widget.paddingRight ?? 16,

        top: widget.paddingTop ?? 16,

        bottom: widget.paddingBottom ?? 16,

      ),

      decoration: BoxDecoration(

        color: widget.backgroundColor ?? Colors.white,

        borderRadius: BorderRadius.circular(widget.borderRadius ?? 8),

      ),

      child: CustomPaint(

        painter: \_BorderPainter(

          color: widget.borderColor ?? Colors.grey,

          width: widget.borderWidth ?? 1.0,

          style: widget.borderStyle,

          radius: widget.borderRadius ?? 8,

        ),

        child: Column(

          mainAxisAlignment: MainAxisAlignment.center,

          crossAxisAlignment: CrossAxisAlignment.center,

          children: [

            if (widget.image != null)

              ClipRRect(

                borderRadius:

                    BorderRadius.circular(widget.imageBorderRadius ?? 0),

                child: Image.asset(

                  widget.image!,

                  width: widget.imageWidth ?? 40,

                  height: widget.imageHeight ?? 40,

                  fit: \_getBoxFit(widget.imageFit),

                ),

              )

            else

              Icon(

                Icons.cloud\_upload,

                size: widget.imageWidth ?? 40,

                color: widget.borderColor ?? Colors.grey,

              ),

            SizedBox(height: widget.marginBetweenTextAndImage ?? 8),

            Text(

              widget.titleText ?? 'Upload File',

              textAlign:

                  \_getTextAlign(widget.titleTextAlign) ?? TextAlign.center,

              style: TextStyle(

                fontSize: widget.titleTextFontSize ?? 16,

                color: widget.titleTextColor ?? Colors.black,

                fontWeight: \_getFontWeight(widget.titleTextFontWeight) ??

                    FontWeight.bold,

              ),

            ),

            SizedBox(height: widget.marginBetweenTitleAndDescrib ?? 4),

            Text(

              widget.describText ??

                  'Drag and drop or click here\n(.JPG, .JPEG, .PNG)',

              textAlign:

                  \_getTextAlign(widget.describTextAlign) ?? TextAlign.center,

              style: TextStyle(

                fontSize: widget.describTextFontSize ?? 14,

                color: widget.describTextColor ?? Colors.grey,

                fontWeight: FontWeight.normal,

              ),

            ),

          ],

        ),

      ),

    );

  }

  TextAlign? \_getTextAlign(String? align) {

    switch (align) {

      case 'center':

        return TextAlign.center;

      case 'left':

        return TextAlign.left;

      case 'right':

        return TextAlign.right;

      default:

        return null;

    }

  }

  FontWeight? \_getFontWeight(String? weight) {

    switch (weight) {

      case 'bold':

        return FontWeight.bold;

      case 'normal':

        return FontWeight.normal;

      default:

        return null;

    }

  }

  BoxFit \_getBoxFit(String? fit) {

    switch (fit) {

      case 'contain':

        return BoxFit.contain;

      case 'cover':

        return BoxFit.cover;

      case 'fill':

        return BoxFit.fill;

      default:

        return BoxFit.cover;

    }

  }

}

class \_BorderPainter extends CustomPainter {

  final Color color;

  final double width;

  final String style;

  final double radius;

  \_BorderPainter({

    required this.color,

    required this.width,

    required this.style,

    required this.radius,

  });

  @override

  void paint(Canvas canvas, Size size) {

    final Paint paint = Paint()

      ..color = color

      ..strokeWidth = width

      ..style = PaintingStyle.stroke;

    final RRect rect = RRect.fromRectAndRadius(

      Offset.zero & size,

      Radius.circular(radius),

    );

    if (style == 'dashed') {

      \_drawDashedBorder(canvas, rect, paint);

    } else if (style == 'dotted') {

      \_drawDottedBorder(canvas, rect, paint);

    } else {

      canvas.drawRRect(rect, paint);

    }

  }

  void \_drawDashedBorder(Canvas canvas, RRect rect, Paint paint) {

    const double dashWidth = 6;

    const double dashSpace = 4;

    final Path path = Path()..addRRect(rect);

    for (final PathMetric metric in path.computeMetrics()) {

      double distance = 0.0;

      while (distance < metric.length) {

        final Path segment = metric.extractPath(distance, distance + dashWidth);

        canvas.drawPath(segment, paint);

        distance += dashWidth + dashSpace;

      }

    }

  }

  void \_drawDottedBorder(Canvas canvas, RRect rect, Paint paint) {

    const double dotRadius = 2;

    const double dotSpacing = 6;

    final Path path = Path()..addRRect(rect);

    for (final PathMetric metric in path.computeMetrics()) {

      double distance = 0.0;

      while (distance < metric.length) {

        final Offset offset = metric.getTangentForOffset(distance)!.position;

        canvas.drawCircle(offset, dotRadius, paint);

        distance += dotSpacing;

      }

    }

  }

  @override

  bool shouldRepaint(CustomPainter oldDelegate) => true;

}