Experiment - 8

AIM:
a) Add animations to UI elements using flutter's animation framework.
b) Experiment with different types of animations like fade, slide, etc.

DESCRIPTION:
Flutter's animation framework offers both implicit and explicit animations to enhance UI elements.

- **Implicit animations:** are suitable for simple property changes and when you prioritize ease of use. Examples: AnimatedContainer, AnimatedOpacity, AnimatedPositioned, AnimatedCrossFade).
- **Explicit animations:** are necessary for complex, custom animations, sequential animations, or when you need precise control over the animation's behaviour. Explicit animations are also prebuilt animation effects, but require an Animation object in order to work.
  Examples: SizeTransition, ScaleTransition or PositionedTransition.

Key Components:

- **Animation**: is a class that represents a running or stopped animation, and is composed of a value representing the target value the animation is running to, and the status, which represents the current value the animation is displaying on screen at any given time. It is a subclass of ***Listenable***, and notifies its listeners when the status changes while the animation is running.
- **AnimationController:** Manages the animation's duration, playback (forward, reverse, repeat), and status. Requires a *TickerProviderStateMixin* in the StatefulWidget.
- **Tween:** Defines the range of values an animated property will transition between (e.g., Tween<double>(begin: 0.0, end: 1.0)).
- **CurvedAnimation:** Applies a non-linear curve to the animation's progress, creating different easing effects (e.g., Curves.easeOut).
- **AnimatedBuilder** or **AnimatedWidget:** Rebuilds the UI based on the current animation value without rebuilding the entire widget tree, optimizing performance.

Experiment - 8(a)

AIM: Add animations to UI elements using flutter's animation framework.

SOLUTION:

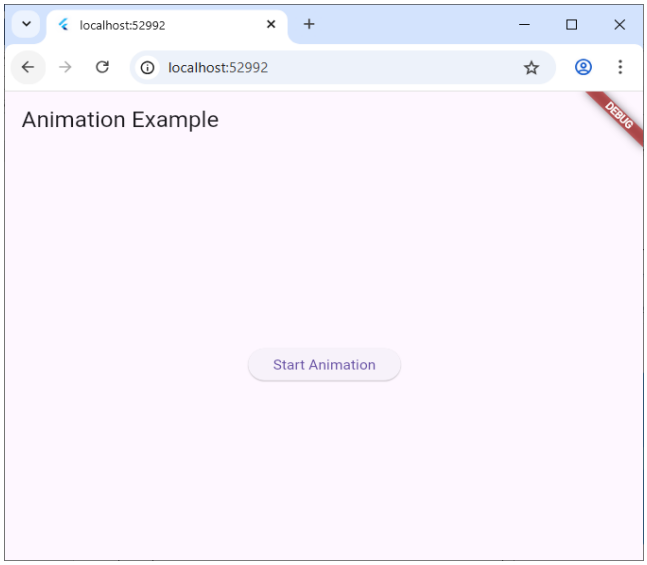
```
import 'package:flutter/material.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget
{
  @override
  Widget build(BuildContext context)
  {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Animation Example'),
        ),
        body: AnimationWidget(),
      ),
    );
  }
```

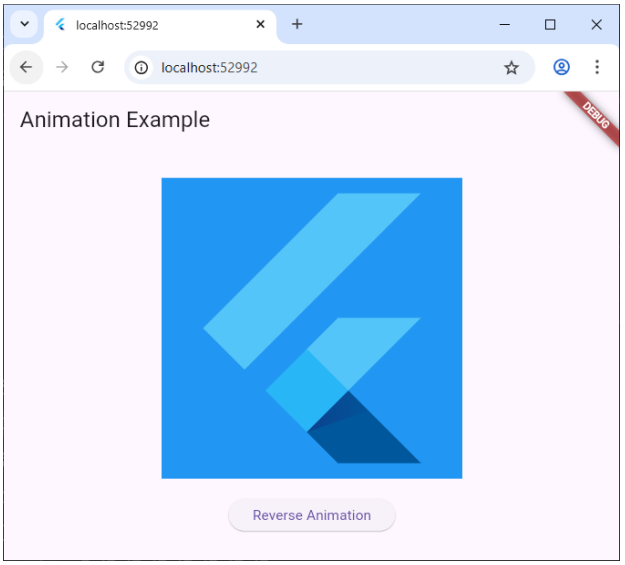
```
}
class AnimationWidget extends StatefulWidget
{
  @override
  _AnimationWidgetState createState() => _AnimationWidgetState();
}

class _AnimationWidgetState extends State<AnimationWidget>
    with SingleTickerProviderStateMixin {
  late AnimationController _controller;
  late Animation<double> _animation;
  @override
  void initState()
  {
    super.initState();
    _controller = AnimationController(
      duration: Duration(seconds: 1),
      vsync: this,
    );
    _animation = Tween<double>(begin: 0, end: 300).animate(_controller)
      ..addListener(()
      {
        setState(() {}); // Trigger rebuild when animation value changes
      });
}
@override
Widget build(BuildContext context)
{
  return Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
        Container(
          width:  _animation.value,
          height: _animation.value,
          color: Colors.blue,
          child: FlutterLogo(size: 100),
        ),
        SizedBox(height: 20),
        ElevatedButton(
          onPressed: ()
          {
            if (_controller.status == AnimationStatus.completed)
            { // Restart animation
              _controller.reverse();
            }
            else
            { // Start animation
              _controller.forward();
            }
          },
          child: Text(
            _controller.status == AnimationStatus.completed
```

```
            ? 'Reverse Animation'
            : 'Start Animation',
          ),
        ),
      ],
    ),
  );
}
@override
void dispose()
{ // Clean up controller when widget is disposed
  _controller.dispose();
  super.dispose();
}
}
```

Output:





Experiment - 8(b)

AIM: Experiment with different types of animations like fade, slide, etc.

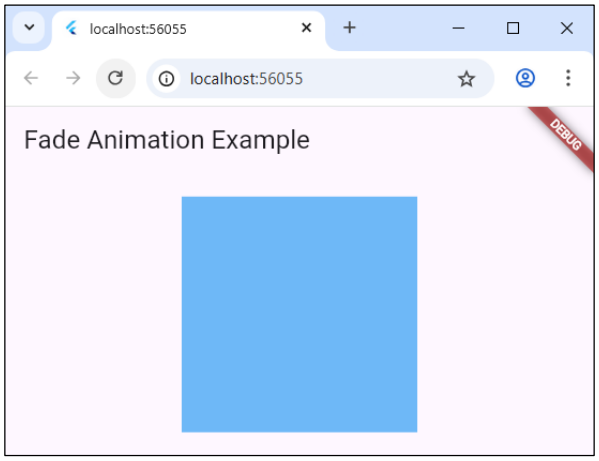
SOLUTION:

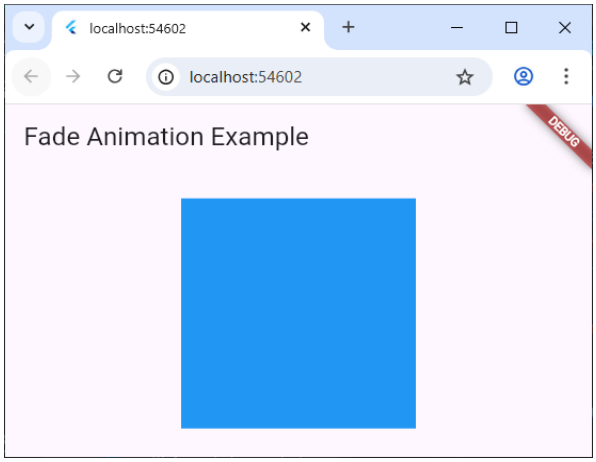
Fade Animation:

```
import 'package:flutter/material.dart';
void main()
{
  runApp(MyApp());
}
class MyApp extends StatelessWidget
{
  @override
  Widget build(BuildContext context)
  {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Fade Animation Example'),
        ),
        body: FadeAnimation(),
      ),
    );
  }
}
class FadeAnimation extends StatefulWidget
{
  @override
  _FadeAnimationState createState() => _FadeAnimationState();
}
class _FadeAnimationState extends State<FadeAnimation>
    with SingleTickerProviderStateMixin {
  late AnimationController _controller;
  late Animation<double> _animation;
  @override
  void initState()
  {
    super.initState();
    _controller = AnimationController(
      duration: Duration(seconds: 2),
      vsync: this,
    );
    _animation = Tween<double>( begin: 0, end: 2).animate(_controller);
    _controller.forward(); // Start animation automatically
  }
  @override
  Widget build(BuildContext context)
  {
    return
      Center(
        child: FadeTransition(
          opacity: _animation,
```

```
          child: Container(
            width: 200,
            height: 200,
            color: Colors.blue,
          ),
        ),
      );
  }
  @override
  void dispose()
  { // Clean up controller when widget is disposed
    _controller.dispose();
    super.dispose();
  }
}
```

Output:



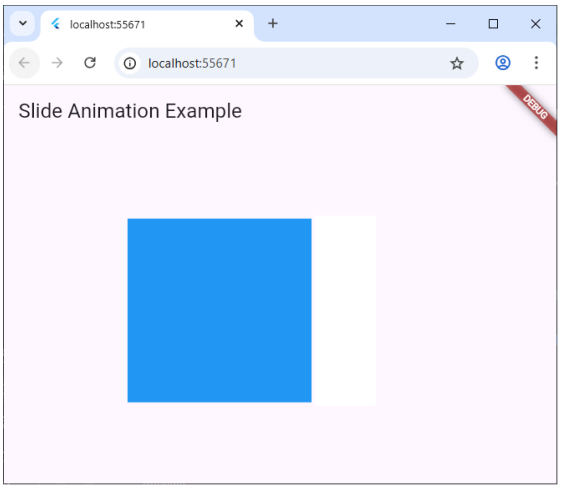


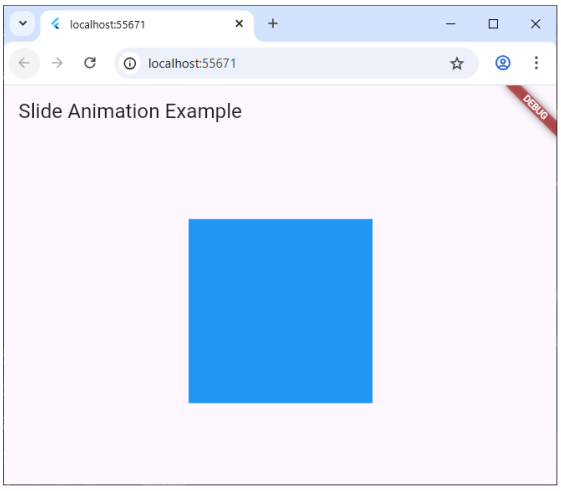
Slide Animation:

```
import 'package:flutter/material.dart';
void main()
{
  runApp(MyApp());
}
class MyApp extends StatelessWidget
{
  @override
  Widget build(BuildContext context)
  {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Slide Animation Example'),
        ),
        body: SlideAnimation(),
      ),
    );
  }
}
class SlideAnimation extends StatefulWidget
{
  @override
  _SlideAnimationState createState() => _SlideAnimationState();
}
class _SlideAnimationState extends State<SlideAnimation>
    with SingleTickerProviderStateMixin {
  late AnimationController _controller;
  late Animation<Offset> _animation;
  @override
  void initState()
  {
    super.initState();
    _controller = AnimationController(
      duration: Duration(seconds: 2),
      vsync: this,
    );
    _animation = Tween<Offset>(
        begin: Offset(-1.0, 0.0),
        end: Offset(0.0, 0.0)
    ).animate(_controller);
    _controller.forward(); // Start animation automatically
  }
  @override
  Widget build(BuildContext context)
  {
    return
      Center(
        child: SlideTransition(
          position: _animation,
          child: Container(
            width: 200,
```

```
            height: 200,
            color: Colors.blue,
          ),
        ),
      );
  }
  @override
  void dispose()
  { // Clean up controller when widget is disposed
    _controller.dispose();
    super.dispose();
  }
}
```

Output:





Scale Animation:

```
import 'package:flutter/material.dart';
void main()
{
  runApp(MyApp());
}
class MyApp extends StatelessWidget
{
  @override
  Widget build(BuildContext context)
  {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Scale Animation Example'),
        ),
        body: ScaleAnimation(),
      ),
    );
  }
}
class ScaleAnimation extends StatefulWidget
{
  @override
  _ScaleAnimationState createState() => _ScaleAnimationState();
}
class _ScaleAnimationState extends State<ScaleAnimation>
    with SingleTickerProviderStateMixin {
  late AnimationController _controller;
  late Animation<double> _animation;
  @override
  void initState()
  {
    super.initState();
    _controller = AnimationController(
      duration: Duration(seconds: 2),
      vsync: this,
    );
    _animation = Tween<double>( begin: 0.0, end: 1.0).animate(_controller);
    _controller.forward(); // Start animation automatically
  }
  @override
  Widget build(BuildContext context)
  {
    return Center(
      child: ScaleTransition(
        scale: _animation,
        child: Container(
          width: 200,
          height: 200,
          color: Colors.blue,
        ),
      ),
```

```
    );
  }
  @override
  void dispose()
  { // Clean up controller when widget is disposed
    _controller.dispose();
    super.dispose();
  }
}
```

Output: