ใบงานการทดลองที่ 6

เรื่อง Widgets

1. จุดประสงค์

1. ใช้ Icon widget ได้
2. ใช้ TextButton widget ได้
3. ใช้ ElevatedButton widget ได้
4. ใช้ OutlinedButton widget ได้
5. ใช้ Container widget ได้

2. ทฤษฎี

บทที่ 3.2 Widgets พื้นฐาน

3.2.5 Icon

Icon ใน Flutter คือ Widget ที่ใช้แสดงไอคอนหรือรูปสัญลักษณ์ต่างๆ ภายในแอปพลิเคชัน ไอคอนสามารถแสดงได้จากชุดไอคอนที่ต่างกัน เช่น Material Design Icons, Cupertino Icons หรือไอคอนที่กำหนดเองได้ โดยไอคอนมักถูกนำมาใช้เพื่อเพิ่มการเข้าใจและประกอบกับข้อความหรือแสดงสถานะต่าง ๆ ในแอปพลิเคชัน Flutter มันมีความยืดหยุ่นในการใช้งานและปรับแต่งไอคอนให้เข้ากับการออกแบบของแอปพลิเคชันได้ตามต้องการในแต่ละส่วนของหน้าจอหรือการใช้งานต่าง ๆ ในแอป

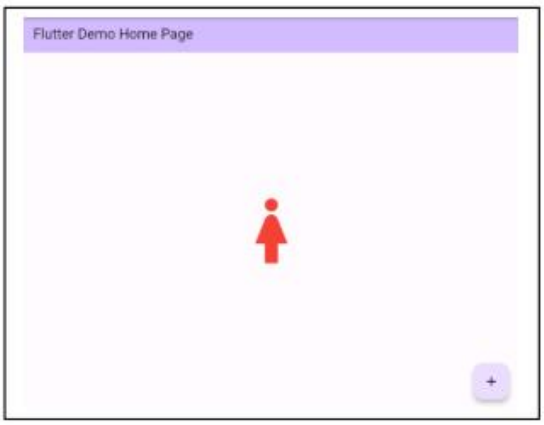
คุณสมบัติ (property) ที่สำคัญของ Icon ได้แก่

- icon: ระบุไอคอนที่ต้องการแสดง เช่น Icons.favorite จาก Material Design Icons หรือ CupertinoIcons.heart จาก Cupertino Icons
- size: กำหนดขนาดของไอคอน
- color: ระบุสีของไอคอน

ตัวอย่างการใช้โค้ด

```
Icon(
    Icons.woman,
    size: 120,
    color: Colors.red
)
```

ผลลัพธ์



จากตัวอย่างข้างต้นเป็นการสร้าง Icon woman ที่แสดงสัญลักษณ์รูปผู้หญิง ที่มีขนาด 120 และเป็นสีแดง

3.2.6 TextButton

TextButton ใน Flutter คือ Widget ที่ใช้สร้างปุ่มที่มีข้อความบนพื้นที่ที่สามารถคลิกหรือแตะได้ เป็นวิดเจ็ตที่สามารถกำหนดข้อความและรูปแบบต่าง ๆ ของปุ่มได้ตามต้องการ เช่น สีพื้นหลังของปุ่ม สีข้อความ และรูปแบบการเรียงตำแหน่งข้อความภายในปุ่ม เมื่อผู้ใช้คลิกหรือแตะปุ่ม TextButton จะส่งการแจ้งเตือนหรือเปิดหน้าจอต่าง ๆ ตามที่ได้กำหนดไว้ในโค้ดของแอปพลิเคชัน Flutter

คุณสมบัติ (property) ที่สำคัญของ TextButton ได้แก่

- onPressed: คุณสมบัติที่ใช้กำหนดฟังก์ชันหรือการกระทำที่ต้องการให้เกิดขึ้นเมื่อปุ่มถูกคลิกหรือแตะ
- child: Widget ภายในปุ่มที่จะแสดง เช่น Text, Icon, Image, หรือ Widgets อื่นๆ
- style: ใช้กำหนดรูปแบบของปุ่ม เช่น สีพื้นหลัง, สีของตัวอักษร, ขนาดและรูปแบบตัวอักษร, การเข้ารหัสของปุ่ม หรือตกแต่งเพิ่มเติมในส่วนอื่นๆ

ตัวอย่างการใช้โค้ด

```
TextButton(
    style: TextButton.styleFrom(
        foregroundColor: Colors.green,
        textStyle: const TextStyle(fontSize: 46),
    ),
        onPressed: _incrementCounter,
        child: Text('Increase'),
)
```

ผลลัพธ์

จากตัวอย่างข้างต้นเป็นการสร้างปุ่มที่มีตัวหนังสือชื่อ “Increase” เป็นปุ่มที่เมื่อกดแล้วจะเรียกใช้ฟังก์ชัน _incrementCounter ที่เป็นฟังก์ชันเพิ่มขึ้นที่ละหนึ่งนั่นเอง

3.2.7 ElevatedButton

ElevatedButton เป็น Widget ใน Flutter ซึ่ง มักจะมีพื้นหลังสีที่เน้นขึ้นเพื่อช่วยให้ผู้ใช้งานสามารถระบุได้ว่านี่คือปุ่มที่สำคัญหรือสำหรับการกระทำที่สำคัญต่างๆ ในแอปพลิเคชัน ปุ่ม ElevatedButton มักจะมีการเพิ่ม shadow หรือเงาด้านล่างเพื่อเน้นให้มีลักษณะเป็นปุ่มที่ยื่นออกมาเป็นวิธีที่ดีในการทำให้ปุ่มนั้นๆ มีความโดดเด่นและตอบสนองต่อการกระทำของผู้ใช้งานในแอปพลิเคชัน Flutter และมันมีคุณสมบัติต่างๆ ที่สามารถกำหนดได้ เช่น สีพื้นหลัง (background color) หรือสีของตัวอักษร (text color) และคำแนะนำ (tooltip) ซึ่งช่วยให้สามารถปรับแต่งรูปแบบของปุ่มให้เข้ากับการออกแบบและโครงสร้างของแอปพลิเคชันได้ตามต้องการ

คุณสมบัติ (property) ที่สำคัญของ ElevatedButton ได้แก่

- onPressed: ระบุฟังก์ชันหรือการกระทำที่จะเกิดขึ้นเมื่อปุ่มถูกกด
- child: Widget ที่จะแสดงในปุ่ม เช่น Text, Icon, หรือ Widget อื่นๆ
- style: ใช้กำหนดรูปแบบการแสดงผลของปุ่ม เช่น สีพื้นหลัง (background color), สีของตัวอักษร (text color), ขนาด (size), และรายละเอียดการแสดงผลอื่นๆ ของปุ่ม

ตัวอย่างการใช้โค้ด

```
ElevatedButton(
    style: ElevatedButton.styleFrom(
        foregroundColor: Colors.white,
        backgroundColor: Colors.green,
        padding: EdgeInsets.all(30),
        textStyle: const TextStyle(fontSize: 36),
    ),
        onPressed: _incrementCounter,
        child: Text('Increase'),
)
```

ผลลัพธ์

จากตัวอย่างข้างต้นเป็นการสร้างปุ่มโดยใช้ ElevatedButton ที่มีตัวหนังสือชื่อ “Increase” เป็นปุ่มที่เมื่อกดแล้วจะเรียกใช้ฟังก์ชัน_incrementCounter ที่เป็นฟังก์ชันเพิ่มขึ้นทีละหนึ่งนั่นเอง

3.2.8 OutlinedButton

OutlinedButton ใน Flutter เป็น Widget ที่ใช้สร้างปุ่มที่มีเส้นขอบ (outline) โดยปุ่มจะไม่มีพื้นหลังสี แต่จะมีเฉพาะเส้นขอบที่แสดงออกมาเป็นลักษณะของปุ่ม ส่วนในส่วนกลางของปุ่มจะเป็นพื้นที่โปร่งใสหรือสีพื้นหลังของพื้นหลังที่ประกาศไว้ (ถ้ามี)
โดยมีคุณสมบัติที่คล้ายกับ ElevatedButton โดยสามารถกำหนดรูปแบบการแสดงผลของปุ่มได้เช่นกัน เช่น สีของเส้นขอบ, สีของตัวอักษร, ฟังก์ชันหรือการกระทำที่เกิดขึ้นเมื่อปุ่มถูกกด และวิดเจ็ตภายในปุ่ม เช่น Text, Icon, หรือ Widget อื่นๆ ก็สามารถนำมาแสดงในปุ่มได้ การใช้งาน OutlinedButton เหมาะสำหรับกรณีที่ต้องการปุ่มที่มีเส้นขอบเป็นลักษณะหลักและต้องการให้พื้นหลังเป็นสีโปร่งใสหรือสะท้อนสีพื้นหลังของส่วนที่ประกาศไว้ เพื่อให้มีลักษณะที่โดดเด่นแต่ยังคงสามารถระบุได้ชัดเจนว่านี่คือปุ่มที่สามารถกดได้ในแอปพลิเคชันของคุณใน Flutter โดยมีความสามารถในการปรับแต่งและการใช้งานที่ยืดหยุ่นตามความต้องการของผู้พัฒนา

คุณสมบัติ (property) ที่สำคัญของ OutlinedButton ได้แก่

- onPressed: ฟังก์ชันหรือการกระทำที่จะเกิดขึ้นเมื่อปุ่มถูกกด
- child: Widget ที่จะแสดงภายในปุ่ม เช่น Text, Icon, หรือ Widget อื่น ๆ
- style: ชุดของสไตล์เพื่อกำหนดรูปแบบการแสดงผลของปุ่ม เช่น สีของเส้นขอบ, สีของตัวอักษร, ขนาดของเส้นขอบ, และลักษณะการแสดงผลอื่น ๆ

ตัวอย่างการใช้โค้ด

```
OutlinedButton(
    style: OutlinedButton.styleFrom(
        padding: EdgeInsets.all(30),
        textStyle: const TextStyle(fontSize: 36),
    ),
        onPressed: _incrementCounter,
        child: Text('Increase'),
)
```

ผลลัพธ์

จากตัวอย่างข้างต้นเป็นการสร้างปุ่มโดยใช้ ElevatedButton ที่มีตัวหนังสือชื่อ “Increase” เป็นปุ่มที่เมื่อกดแล้วจะเรียกใช้ฟังก์ชัน_incrementCounter ที่เป็นฟังก์ชันเพิ่มขึ้นที่ละหนึ่งนั่นเอง

3.2.9 Container

Container เป็น Widget ที่มีหน้าที่ในการจัดการพื้นที่และการจัดวาง (layout) ขององค์ประกอบต่าง ๆ ในแอปพลิเคชัน โดย Container สามารถทำหน้าที่เป็นบริเวณสี่เหลี่ยมผืนผ้าที่มีความยืดหยุ่นในการกำหนดขนาด (size) และคุณสมบัติต่าง ๆ ของการแสดงผล (presentation properties) ได้เป็นอย่างดี นอกจากนี้ยังสามารถใช้งานเพื่อใส่ Widget อื่น ๆ เข้าไปด้านในได้ด้วย เช่น Text, Image, หรือ Widget อื่น ๆ ที่ต้องการจัดวาง โดย Container เป็นองค์ประกอบสำคัญที่ช่วยให้การจัดการพื้นที่และการแสดงผลใน Flutter มีความยืดหยุ่นและควบคุมได้ตามต้องการของแอปพลิเคชัน และสามารถปรับแต่งลักษณะแสดงผลตามความต้องการได้อย่างหลากหลาย

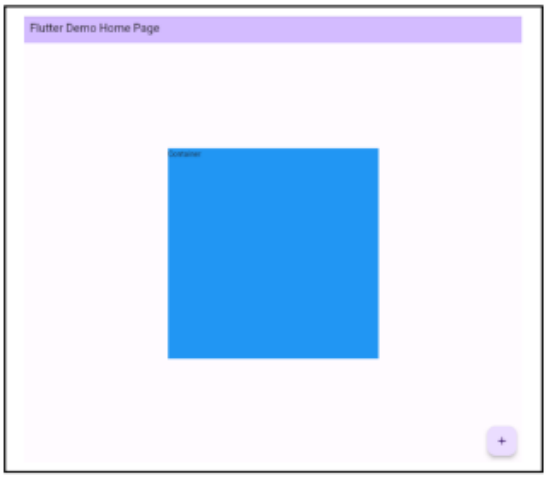
คุณสมบัติ (property) ที่สำคัญของ Container Widget ได้แก่

- width: ขนาดความกว้าง
- height: ขนาดความสูง
- color: สีของพื้นหลัง
- child: Widget ที่จะแสดงภายในปุ่ม เช่น Text, Icon, หรือ Widget อื่น ๆ

ตัวอย่างการใช้โค้ด

```
Container(
    width: 400,
    height: 400,
    color: Colors.blue,
    child: Text('Container'),
)
```

ผลลัพธ์



จากตัวอย่างข้างต้นเป็นการสร้าง Container ที่มีพื้นหลังสีฟ้าขนาดกว้าง 400 สูง 400 มีข้อความด้านในกล่องเขียนว่า “Container”

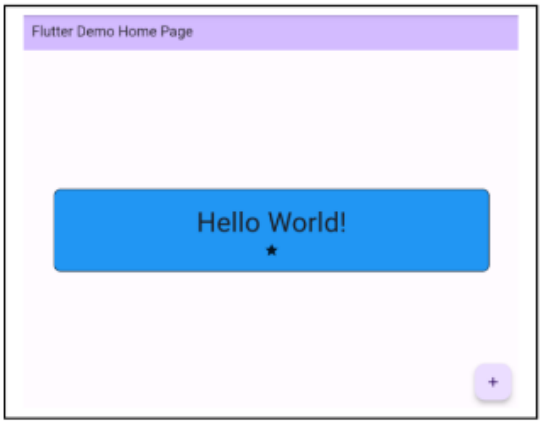
คุณสมบัติ (property) เพิ่มเติม Container Widget ได้แก่

- alignment: การจัดวางของ Widget ภายใน Container
- padding: การกำหนดระยะห่าง (padding) รอบขอบของ Container
- margin: การกำหนดระยะห่าง (margin) ระหว่าง Container กับ Widget ที่อยู่รอบ ๆ
- decoration: ใช้กำหนดรูปแบบการแสดงผลเพิ่มเติม เช่น เส้นขอบ, ภาพพื้นหลัง หรือ สีพื้นหลัง

ตัวอย่างการใช้โค้ด

```
Container(
    alignment: Alignment.center,
    margin: EdgeInsets.all(50.0),
    padding: EdgeInsets.symmetric(horizontal: 100,
vertical: 20),
    decoration: BoxDecoration(
        color: Colors.blue,
        border: Border.all(color: Colors.black),
        borderRadius: BorderRadius.circular(10.0),
    ),
    child: Column(
        children: [
            Text('Hello World!',
                    style: TextStyle(fontSize: 42)),
            Icon(Icons.star),
        ],
    ),
)
```

ผลลัพธ์



จากตัวอย่างข้างต้นเป็นการสร้าง Container ที่มีพื้นหลังสีฟ้าเส้นขอบสีดำขนาดกว้าง 100 สูง 20 ภายในประกอบไปด้วยตัวหนังสือ “Hello World!” และสัญลักษณ์รูปดาว

บทที่ 3.3 Stateless Widget

Stateless Widget ใน Flutter คือประเภทหนึ่งของ Widget ซึ่งไม่สามารถเปลี่ยนแปลงข้อมูลภายในตัวมันเองได้หลังจากถูกสร้างขึ้นแล้ว หมายความว่า มันไม่มีสถานะ (state) ที่สามารถเปลี่ยนแปลงได้ แต่สามารถถูกสร้างและแสดงผลบนหน้าจอได้ เมื่อข้อมูลที่ใช้ในการสร้าง Widget เปลี่ยนแปลง การสร้าง Stateless Widget ใหม่จะเกิดขึ้นเพื่อแสดงผลข้อมูลใหม่นั้นแทน การที่มันไม่มีสถานะทำให้การทำงานของ Stateless Widget เร็วและมีประสิทธิภาพ เนื่องจากไม่ต้องจัดการกับการเปลี่ยนแปลงของข้อมูลภายในตัวมันเอง

3.3.1 ลักษณะที่สำคัญของ Stateless Widget

- **ไม่มีสถานะ (Stateless):** ไม่สามารถเปลี่ยนแปลงข้อมูลภายในตัวมันเองได้หลังจากถูกสร้างขึ้นแล้ว ซึ่งทำให้มันมีประสิทธิภาพและเร็วเนื่องจากไม่ต้องจัดการกับการเปลี่ยนแปลงของข้อมูล
- **ไม่มีการเปลี่ยนแปลงค่า (Immutable):** ค่าที่ใช้ในการสร้าง Stateless Widget จะถูกส่งผ่านเป็นพารามิเตอร์และจะไม่มีการเก็บค่าไว้ในตัวมันเอง
- **สร้าง UI ตามข้อมูลที่ได้รับ:** ใช้ข้อมูลที่ได้รับเพื่อสร้าง UI และแสดงผลบนหน้าจอ
- **ไม่สามารถเปลี่ยนแปลงการแสดงผลเองได้:** เมื่อ Stateless Widget ถูกสร้างแล้ว มันไม่สามารถเปลี่ยนแปลงการแสดงผลของตัวเองได้ ถ้าข้อมูลเปลี่ยนแปลง มันจะถูกสร้างใหม่และแสดงผลใหม่ตามข้อมูลใหม่นั้น
- **เหมาะสำหรับ UI ที่ไม่ต้องการการเปลี่ยนแปลงบ่อยๆ:** ใช้สำหรับส่วนของ UI ที่สถานะไม่มีการเปลี่ยนแปลงบ่อยๆ เช่น การแสดงข้อมูลที่ไม่ต้องการการอัพเดทในเวลาเรียลไทม์

3.3.2 ตัวอย่างที่เหมาะกับการใช้ Stateless Widget

- **แสดงข้อมูลส่วนตัว:** สามารถใช้ Stateless widget เพื่อแสดงข้อมูลส่วนตัว เช่น ชื่อ, อายุ, ที่อยู่, ข้อมูลติดต่อ เป็นต้น ซึ่งข้อมูลเหล่านี้มักจะเป็นคงที่และไม่มีการเปลี่ยนแปลงบ่อยครั้ง
- **แสดงผลผลงาน (Portfolio)**: Stateless widget เหมาะสำหรับการแสดงผลงานต่าง ๆ ของตนเอง ซึ่งอาจเป็นรูปภาพ, วีดีโอ, หรือข้อมูลเกี่ยวกับงานที่ทำได้ เนื่องจากข้อมูลเหล่านี้มักจะเป็นคงที่และไม่มีการเปลี่ยนแปลงบ่อยครั้ง
- **ส่วนเกี่ยวกับการศึกษาหรือประสบการณ์**: สามารถใช้ Stateless widget เพื่อแสดงส่วนของการศึกษา หรือประสบการณ์ทางการศึกษาหรือการทำงาน ซึ่งข้อมูลในส่วนนี้มักจะเป็นคงที่และไม่มีการเปลี่ยนแปลง
- **ข้อมูลการทำงานหรือโปรเจ็กต์**: สร้าง Stateless widget เพื่อแสดงรายละเอียดเกี่ยวกับโปรเจ็กต์หรืองานที่ทำได้ อาจมีการแสดงรายละเอียดเกี่ยวกับส่วนของงาน หรือคำอธิบายเกี่ยวกับโปรเจ็กต์

3.3.3 Method ที่สำคัญของ Stateless Widget

- **build:** เป็น method ที่บังคับให้ implement ใน Stateless widget ซึ่งใช้สำหรับสร้าง UI หรือ Widget tree โดยที่มันจะถูกเรียกทุกครั้งที่ state ของ widget ถูกเปลี่ยนแปลง หรือตอนที่มีการ rebuild UI นั่นเอง

3.3.4 การใช้งาน Stateless Widget

ขั้นตอนพื้นฐานในการใช้งาน Stateless Widget มีดังนี้

- **สร้างคลาส Stateless Widget:** สร้างคลาสใหม่ที่ extends จาก StatelessWidget และให้การสร้าง UI ที่ต้องการแสดงผลในเมธอด build() ภายในคลาส Stateless Widget

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


class MyStatelessWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      // สร้าง UI ตามที่ต้องการ
    );
  }
}
```

- **การนำไปใช้ใน Widget Tree:** นำ MyStatelessWidget ที่สร้างมาใช้งานใน Widget Tree ของแอปพลิเคชั่น Flutter โดยการเรียกใช้งานเหมือนกับ Widget ปกติ

- **การนำไปใช้ใน Widget Tree:** นำ MyStatelessWidget ที่สร้างมาใช้งานใน Widget Tree ของแอปพลิเคชั่น Flutter โดยการเรียกใช้งานเหมือนกับ Widget ปกติ

```
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('Stateless Widget Example')),
        body: Center(
          child: MyStatelessWidget(), // เรียกใช้ MyStatelessWidget ที่สร้างไว้
        ),
      ),
    );
  }
}
```

บทที่ 3.4 Stateful Widget

ใน Flutter, Stateful Widget เป็นหนึ่งในประเภทของ Widget ซึ่งสามารถเก็บ state หรือสถานะของตัวเองไว้ได้ นั่นหมายถึง Widget ที่สามารถเปลี่ยนแปลงข้อมูลภายในตัวมันเองได้ โดย Stateful Widget จะประกอบด้วยสองส่วนหลัก คือ StatefulWidget ซึ่งเป็น immutable และ State ที่เป็น mutable

- **StatefulWidget:** เป็นส่วนที่ไม่สามารถเปลี่ยนแปลงได้หลังจากสร้าง เพราะฉะนั้นสิ่งที่เปลี่ยนแปลงจะต้องเกิดขึ้นใน State ซึ่งเป็น class แยกที่เก็บสถานะและการเปลี่ยนแปลงของ Widget
- **State:** เป็นส่วนที่สามารถเปลี่ยนแปลงได้ และเก็บข้อมูลที่มีความสำคัญต่อการแสดงผลของ Widget โดย State จะถูกผูกติดกับ StatefulWidget และจะมีการ rebuild หรือสร้าง Widget ใหม่เมื่อมีการเปลี่ยนแปลง state

การใช้ Stateful Widget ช่วยให้สามารถอัปเดต UI ของแอปพลิเคชัน Flutter ตามการเปลี่ยนแปลงของข้อมูลหรือสถานะที่ถูกเก็บไว้ภายใน Widget ได้ง่ายขึ้นและมีประสิทธิภาพมากยิ่งขึ้น

3.4.1 ลักษณะที่สำคัญของ Stateful Widget

- **Mutable State:** Stateful Widget มีความสามารถในการเก็บ state หรือสถานะภายในตัวมันเอง ซึ่งสามารถเปลี่ยนแปลงได้ตลอดการทำงานของแอปพลิเคชัน ซึ่งจะส่งผลในการอัปเดตหน้าจอหรือ UI ในกรณีที่ state เปลี่ยนแปลง
- **setState()**: ฟังก์ชัน setState() เป็นเครื่องมือสำคัญที่ช่วยในการแจ้งให้ Flutter ทราบว่า state ของ Widget ได้เปลี่ยนแปลง ซึ่งจะทำให้ Flutter ทำการ rebuild หรือสร้าง Widget ใหม่เพื่อให้สามารถแสดงผล UI ใหม่ตาม state ที่เปลี่ยนแปลง
- **Lifecycle Methods**: Stateful Widget มี lifecycle methods ที่ช่วยในการจัดการตั้งแต่ขั้นตอนการสร้าง Widget จนถึงการทำลาย Widget ซึ่งช่วยให้นักพัฒนาสามารถจัดการกับการเปลี่ยนแปลงต่าง ๆ และการทำความสะอาดหรือปรับปรุงข้อมูลก่อนที่ Widget จะถูกทำลาย
- **Rebuild ตาม State ที่เปลี่ยนแปลง**: เมื่อ state ของ Stateful Widget เปลี่ยนแปลง การเรียกใช้งาน setState() จะทำให้ Flutter ทำการ rebuild หรือสร้าง Widget ใหม่เพื่อให้แสดงผล UI ใหม่ตาม state ที่เปลี่ยนแปลง
- **มีความซับซ้อนเมื่อเทียบกับ Stateless Widget**: Stateful Widget มักมีความซับซ้อนกว่า Stateless Widget เนื่องจากมีการจัดการ state และการเปลี่ยนแปลง UI ที่ต้องติดต่อกับ state ซึ่งอาจทำให้มันมีขนาดของโค้ดมากขึ้นบ้าง

3.4.2 ตัวอย่างที่เหมาะกับการใช้ Stateful Widget

- **แอปพลิเคชันที่มีการอัปเดต state อย่างต่อเนื่อง (Real-time Updates):** เช่น แอปพลิเคชันที่มีการแสดงผลข้อมูลแบบ real-time หรือข้อมูลที่มีการอัปเดตอย่างต่อเนื่อง เช่น แอปพลิเคชันการแชท, แอปพลิเคชันข่าวสารที่มีการอัปเดตข้อมูลอยู่ตลอดเวลา
- **ฟอร์มและอินพุตผู้ใช้ (User Input)**: Stateful widget เหมาะสำหรับการสร้างฟอร์มหรืออินพุตที่ผู้ใช้สามารถป้อนข้อมูลเข้ามา และควบคุมการแสดงผลตามข้อมูลที่ถูกป้อน ตัวอย่างเช่น ฟอร์มการลงทะเบียน, ฟอร์มการเข้าสู่ระบบ, หรือหน้าจอการสั่งซื้อสินค้า
- **การจัดการอนิเมชันและการเคลื่อนไหว (Animations and Transitions)**: Stateful widget สามารถใช้ในการจัดการอนิเมชันและการเคลื่อนไหว โดยการอัปเดต state เพื่อสร้างการเปลี่ยนแปลงใน UI หรือการทำอนิเมชันต่าง ๆ เช่น การเลือกภาพในแกลเลอรีที่มีการแสดงภาพโต้ตอบ

3.4.3 Method ที่สำคัญของ Stateful Widget

- **build:** เมธอดนี้จะถูกเรียกทุกครั้งเมื่อ state object ต้องการที่จะแสดงผล UI ที่ต้องการ rebuild โดยทั่วไปจะคืนค่าเป็น Widget ที่ต้องการแสดงผล
- **createState():** เมื่อ stateful widget ถูกสร้างขึ้น แม่แบบจะเรียกเมธอดนี้เพื่อสร้างอ็อบเจกต์ state สำหรับ widget นั้น ๆ
- **setState():** เมธอดที่ใช้สำหรับอัปเดต state ของ stateful widget และสั่งให้ Flutter ทำการ rebuild UI ใหม่ เมื่อ state เปลี่ยนแปลง

3.4.4 การใช้งาน Stateful Widget

ขั้นตอนพื้นฐานในการใช้งาน Stateful Widget มีดังนี้

- **สร้าง Stateful Widget:** เริ่มต้นโดยการสร้าง Stateful Widget โดยใช้ StatefulWidget และ State class แยกกัน เช่น

```
class MyStatefulWidget extends StatefulWidget {
 @override
 _MyStatefulWidgetState createState() => _MyStatefulWidgetState();
}

class _MyStatefulWidgetState extends State<MyStatefulWidget> {
 // ในส่วนนี้คุณสามารถกำหนด state และเมธอดสำหรับการเปลี่ยนแปลง state ได้
}
```

- **การจัดการ State:** ใน _MyStatefulWidgetState class, คุณสามารถกำหนดตัวแปร state และเมธอดสำหรับการเปลี่ยนแปลง state ได้ เช่น

```
class _MyStatefulWidgetState extends State<MyStatefulWidget> {
 String message = 'Initial message';

 void updateMessage() {
  setState(() {
   message = 'Updated message';
  });
 }

 @override
 Widget build(BuildContext context) {
  // ในส่วนนี้คุณสามารถใช้ state ในการสร้าง UI
  return Container(
   child: Column(
    children: [
     Text(message),
     ElevatedButton(
      onPressed: () {
       updateMessage(); // เรียกใช้เมธอดเพื่อเปลี่ยนแปลง state
      },
      child: Text('Update Message'),
     ),
    ],
   ),
  );
 }
}
```

- **การใช้งาน setState():** เมื่อมีการเปลี่ยนแปลง state ใน _MyStatefulWidgetState ให้ใช้ setState() เพื่อบอก Flutter ให้ rebuild UI ตาม state ใหม่ เมื่อ state ถูกเปลี่ยนแปลง

```
void updateMessage() {
 setState(() {
  message = 'Updated message';
 });
}
```

- **การใช้ Stateful Widget ในแอปพลิเคชัน:** ในการใช้งานในแอปพลิเคชัน, คุณสามารถนำ Stateful Widget ที่สร้างไปใช้งานได้เหมือนกับ Widget อื่น ๆ โดยเรียกใช้ MyStatefulWidget() เช่น

```
void main() {
  runApp(MaterialApp(
    home: Scaffold(
      appBar: AppBar(title: Text('Stateful Widget Example')),
      body: MyStatefulWidget(), // นำ Stateful Widget มาใช้งานในโค้ดหลัก
    ),
  ));
}
```

3. เครื่องมือและอุปกรณ์การทดลอง

3.1 คอมพิวเตอร์ 1 เครื่อง

3.2 ใบงานที่ 1 เรื่อง คำสั่งเลือกเงื่อนไข

4. ลำดับขั้นการทดลอง

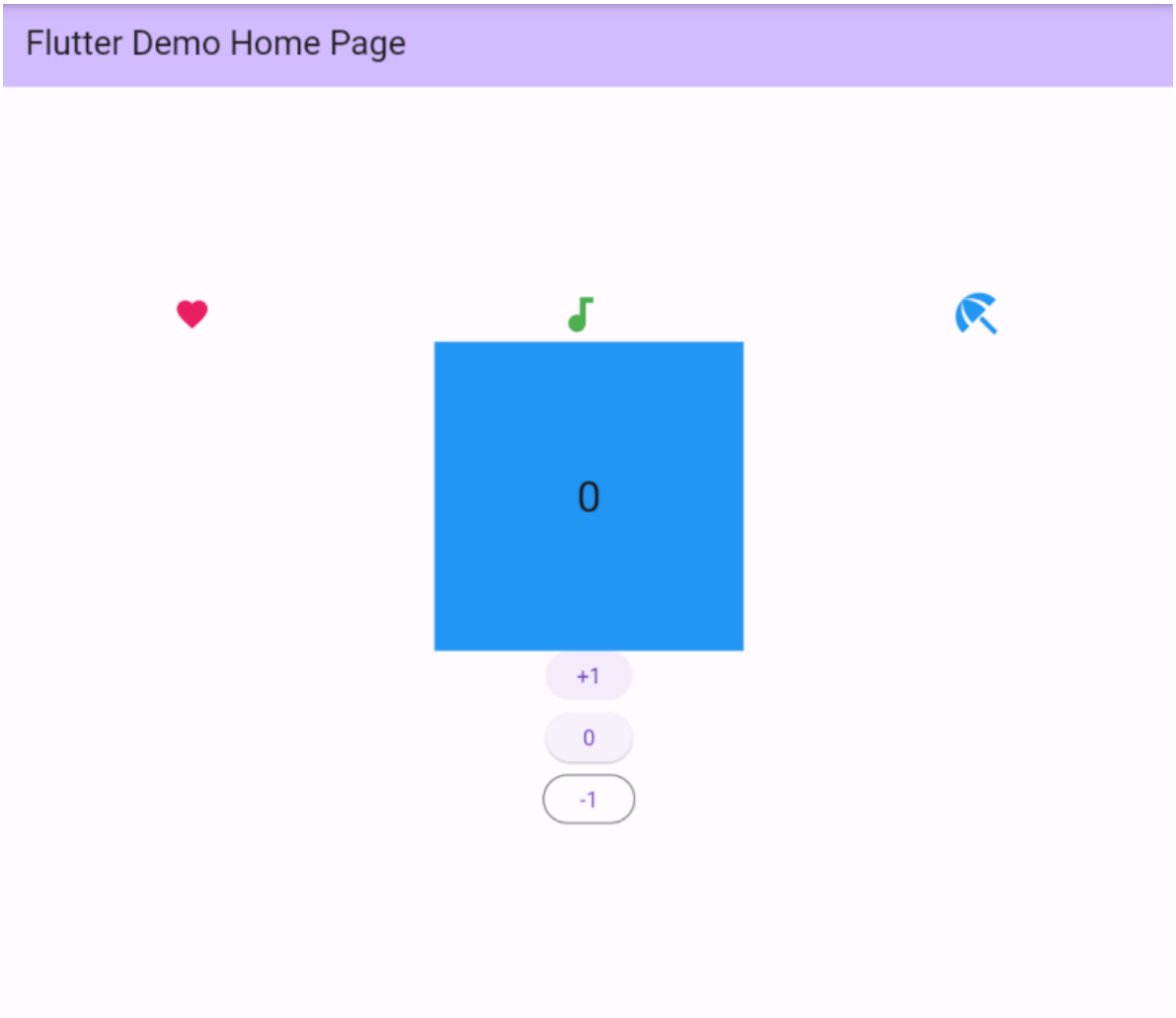
4.1 นักเรียนศึกษาเนื้อหา เรื่อง คำสั่งเลือกเงื่อนไข

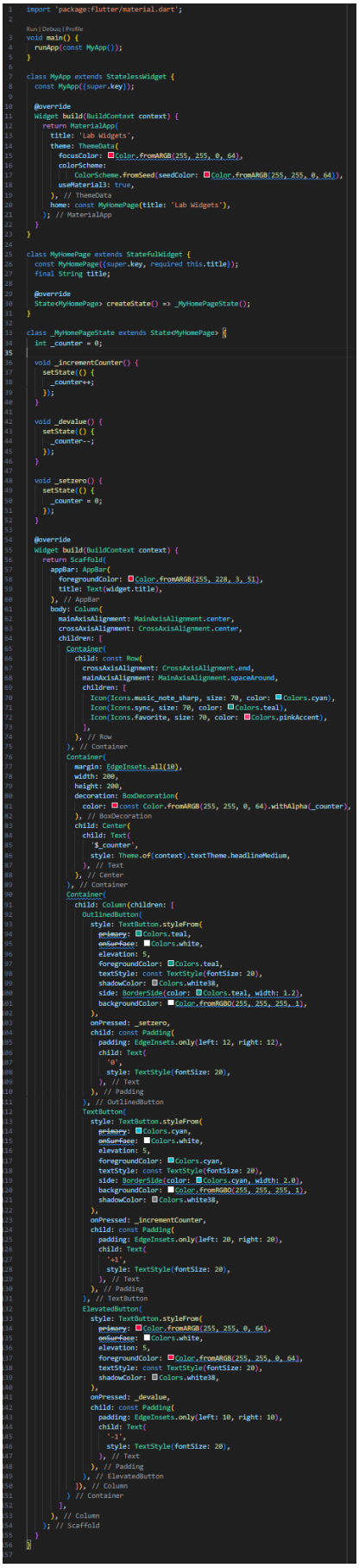
4.2 ให้นักเรียนตอบคำถาม ลงในใบงานที่ 1

4.3 ส่งงานครูหลังจากเสร็จเรียบร้อย

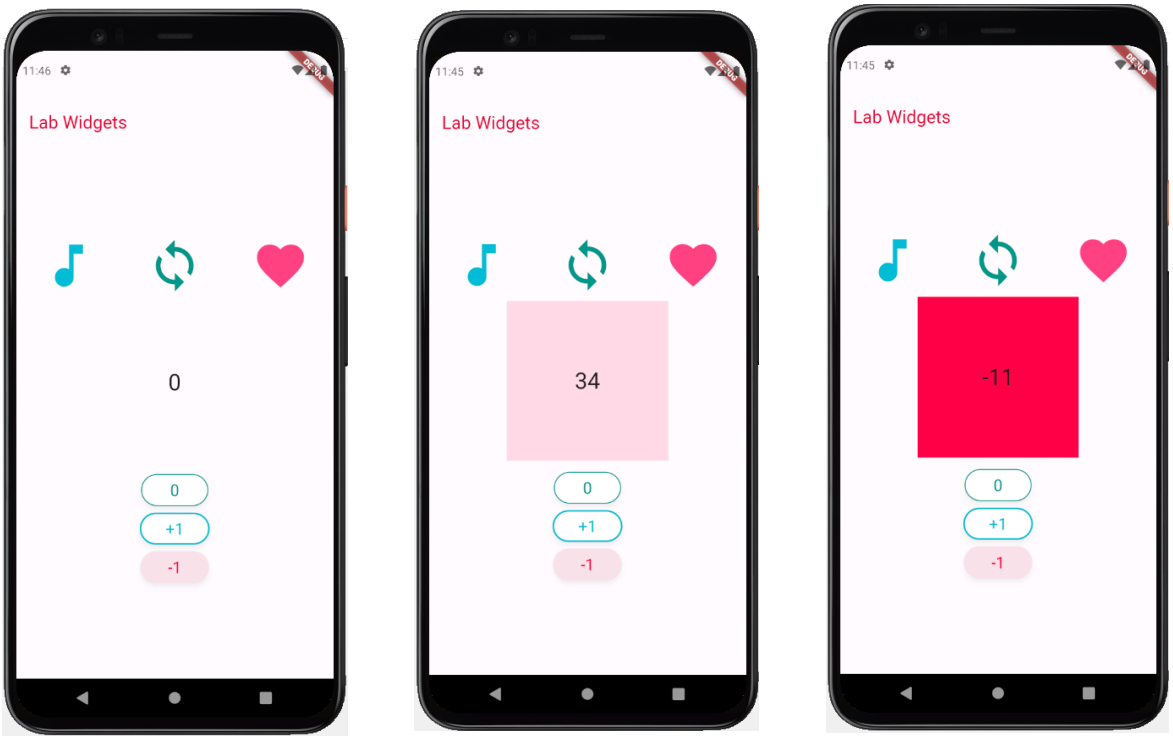
คำสั่ง ให้นักศึกษาเขียนคำตอบตามที่โจทย์กำหนดให้ถูกต้อง (สามารถแนบรูปโค้ดและผลลัพธ์คำตอบของโปรแกรมได้)

1. ใช้ Icon widget , TextButton widget , ElevatedButton widget, OutlinedButton widget และ Container widget ประกอบกัน โดยกันแปลง code Counter ให้มีตัว Icon 3 อันโดยมีการใส่สีและระบุขนาดของ Icon ให้เรียบร้อย text ตัวเลข Counter ให้อยู่ใน Container ที่มีการระบุขนาดและสี มีปุ่ม TextButton widget , ElevatedButton widget, OutlinedButton widget ให้เป็นปุ่ม +1 0 -1 ของตัวเลข Counterตามลำดับ โองอิงจากรูปด้านล่าง





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

Run | Debug | Profile
void main() {
  runApp(const MyApp());
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Lab Widgets',
      theme: ThemeData(
        focusColor: Color.fromARGB(255, 255, 0, 64),
        colorScheme:
            ColorScheme.fromSeed(seedColor: Color.fromARGB(255, 255, 0, 64)),
        useMaterial3: true,
      ), // ThemeData
      home: const MyHomePage(title: 'Lab Widgets'),
    ); // MaterialApp
  }
}

class MyHomePage extends StatefulWidget {
  const MyHomePage({super.key, required this.title});
  final String title;

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  void _devalue() {
    setState(() {
      _counter--;
    });
  }

  void _setzero() {
    setState(() {
      _counter = 0;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        foregroundColor: Color.fromARGB(255, 228, 3, 51),
        title: Text(widget.title),
      ), // AppBar
      body: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        crossAxisAlignment: CrossAxisAlignment.center,
        children: [
          Container(
            child: const Row(
              crossAxisAlignment: CrossAxisAlignment.end,
              mainAxisAlignment: MainAxisAlignment.spaceAround,
              children: [
                Icon(Icons.music_note_sharp, size: 70, color: Colors.cyan),
                Icon(Icons.sync, size: 70, color: Colors.teal),
                Icon(Icons.favorite, size: 70, color: Colors.pinkAccent),
              ],
            ), // Row
          ), // Container
          Container(
            margin: EdgeInsets.all(10),
            width: 200,
            height: 200,
            decoration: BoxDecoration(
              color: const Color.fromARGB(255, 255, 0, 64).withAlpha(_counter),
            ), // BoxDecoration
            child: Center(
              child: Text(
                '$_counter',
                style: Theme.of(context).textTheme.headlineMedium,
              ), // Text
            ), // Center
          ), // Container
          Container(
            child: Column(children: [
              OutlinedButton(
                style: TextButton.styleFrom(
                  primary: Colors.teal,
                  onSurface: Colors.white,
                  elevation: 5,
                  foregroundColor: Colors.teal,
                  textStyle: const TextStyle(fontSize: 20),
                  shadowColor: Colors.white38,
                  side: BorderSide(color: Colors.teal, width: 1.2),
                  backgroundColor: Color.fromRGBO(255, 255, 255, 1),
                ),
                onPressed: _setzero,
                child: const Padding(
                  padding: EdgeInsets.only(left: 12, right: 12),
                  child: Text(
                    '0',
                    style: TextStyle(fontSize: 20),
                  ), // Text
                ), // Padding
              ), // OutlinedButton
              TextButton(
                style: TextButton.styleFrom(
                  primary: Colors.cyan,
                  onSurface: Colors.white,
                  elevation: 5,
                  foregroundColor: Colors.cyan,
                  textStyle: const TextStyle(fontSize: 20),
                  side: BorderSide(color: Colors.cyan, width: 2.0),
                  backgroundColor: Color.fromRGBO(255, 255, 255, 1),
                  shadowColor: Colors.white38,
                ),
                onPressed: _incrementCounter,
                child: const Padding(
                  padding: EdgeInsets.only(left: 20, right: 20),
                  child: Text(
                    '+1',
                    style: TextStyle(fontSize: 20),
                  ), // Text
                ), // Padding
              ), // TextButton
              ElevatedButton(
                style: TextButton.styleFrom(
                  primary: Color.fromARGB(255, 255, 0, 64),
                  onSurface: Colors.white,
                  elevation: 5,
                  foregroundColor: Color.fromARGB(255, 255, 0, 64),
                  textStyle: const TextStyle(fontSize: 20),
                  shadowColor: Colors.white38,
                ),
                onPressed: _devalue,
                child: const Padding(
                  padding: EdgeInsets.only(left: 10, right: 10),
                  child: Text(
                    '-1',
                    style: TextStyle(fontSize: 20),
                  ), // Text
                ), // Padding
              ), // ElevatedButton
            ]), // Column
          ) // Container
        ],
      ), // Column
    ); // Scaffold
  }
}
```


11:46
Lab Widgets
0
0
+1
-1
11:45
Lab Widgets
34
0
+1
-1
11:45
Lab Widgets
-11
0
+1
-1

5. สรุปผลการทดลอง

จากการทดลองสรุปได้ว่าการใช้งาน dark เป็นการใช้งานในการสร้างแอปโดยจะมีการอำนวยความสะดวกในเรื่องของ icon และปุ่มต่างที่เราไม่จำเป็นต้องโหลดหรือนำรูปใดๆมาตกแต่งเพราะใน dark ไม่ให้อยู่แล้ว ต่อมาเป็นการใช้ container เป็นการเหมือนเรามี contanner เพื่อจัดเก็บสิ่งต่างๆ โดยเราสามารถย้ายปรับเปลี่ยน สิ่งที่อยู่ในนั้นก็จะปรับเปลี่ยนตามต่อมาเป็นการสร้าง class ในการทำงานเพื่อให้รู้ว่าค่าแต่ละอย่างสามารถจำเป็นต้องกำหนดค่าหรือไม่เป็นการแยกการจัดการให้มีการใช้งานที่เหมาะสมต่องานของเรา

6. คำถามหลังการทดลอง

6.1 StatefulWidget และ StatelessWidget แตกต่างกันอย่างไร

StatefulWidget และ StatelessWidget เป็นสองประเภทหลักของ Widget ใน Flutter
โดยความแตกต่างที่สำคัญระหว่างทั้งสองประเภทคือความสามารถในการเปลี่ยนแปลงค่าหรือ state

StatelessWidget
ไม่สามารถเปลี่ยนแปลงค่าหรือ state ของ Widget ได้
เมื่อสร้าง Instance ของ StatelessWidget แล้วจะไม่สามารถเปลี่ยนแปลงค่าหรือ state ของ Widget ได้เลย
เหมาะสำหรับ Widget ที่ไม่มีการเปลี่ยนแปลงค่าหรือ state เช่น Text, Image, Container

StatefulWidget
สามารถเปลี่ยนแปลงค่าหรือ state ของ Widget ได้
เมื่อสร้าง Instance ของ StatefulWidget แล้วจะสามารถสร้าง State object เพื่อเก็บค่าหรือ state ของ Widget ได้
เหมาะสำหรับ Widget ที่มีการการเปลี่ยนแปลงค่าหรือ state เช่น Checkbox, Switch, Slider