**TRƯỜNG ĐẠI HỌC BÁCH KHOA**

**KHOA ĐIỆN TỬ VIỄN THÔNG**

**BÁO CÁO**

**LẬP TRÌNH ĐA NỀN TẢNG**

**CHỦ ĐỀ: BUTTONS VÀ INPUT WIDGETS**

**Sinh viên thực hiện:**

**01. Bùi Trịnh Thế Viên Lớp: 22KTMT2**  **MSSV:**  106220276

**02. Kim Gia Bảo**  **Lớp: 22KTMT2**  **MSSV:**  106220245

**Người hướng dẫn:**

**TS. Nguyễn Duy Nhật Viễn**

**Đà Nẵng, 2025.**

**BÁO CÁO**

**LẬP TRÌNH ĐA NỀN TẢNG**

**CHỦ ĐỀ: BUTTONS VÀ INPUT WIDGETS**

BẢNG PHÂN CÔNG CÔNG VIỆC TRONG NHÓM

|  |  |  |  |
| --- | --- | --- | --- |
| STT | HỌ VÀ TÊN | NHIỆM VỤ | KHỐI LƯỢNG |
| 01 | BÙI TRỊNH THẾ VIÊN | Tìm hiểu, code nội dung ElevatedButton, TextButton, OutlinedButton, IconButton, FloatingActionButton và các variants, TextField với controller và validation | 50% |
| 02 | KIM GIA BẢO | Tìm hiểu, code nội dung Switch, Checkbox, Radio, Slider, GestureDetector và InkWell | 50% |

 Link code github: <https://github.com/thevien257/Flutter-Project>

Mục lục

[I. GIỚI THIỆU 4](#_Toc211333784)

[1. Mục đích đồ án 4](#_Toc211333785)

[2. Phạm vi thực hiện 4](#_Toc211333786)

[II. CƠ SỞ LÝ THUYẾT 5](#_Toc211333787)

[1. Part 1: Buttons, FloatingActionButton và TextField 5](#_Toc211333788)

[1.1 Buttons trong Material Design 5](#_Toc211333789)

[1.2. FloatingActionButton (FAB) 7](#_Toc211333790)

[1.3. TextField và Input Validation 8](#_Toc211333791)

[1.4. Card Widget 10](#_Toc211333792)

[1.5. State Management: StatefulWidget và StatelessWidget 11](#_Toc211333793)

[2. Part 2 Widgets - Selection Controls và Gestures: 15](#_Toc211333794)

[2.1. Switch Widget 15](#_Toc211333795)

[2.2. Checkbox Widget 16](#_Toc211333796)

[2.3. Radio Widget 18](#_Toc211333797)

[2.4. Slider Widget 19](#_Toc211333798)

[2.5. GestureDetector 21](#_Toc211333799)

[2.6. InkWell 23](#_Toc211333800)

[III. THIẾT KẾ VÀ TRIỂN KHAI 27](#_Toc211333801)

[1. Kiến trúc ứng dụng 27](#_Toc211333802)

[2. Triển khai Part 1 27](#_Toc211333803)

[2.1 Quản lý trạng thái 27](#_Toc211333804)

[2.2 Triển khai Buttons 28](#_Toc211333805)

[2.3. Triển khai FloatingActionButton Variants 33](#_Toc211333806)

[2.4. Triển khai Card Organization 38](#_Toc211333807)

[2.5. Triển khai TextField với Validation 46](#_Toc211333808)

[2.6. State Management Implementation 49](#_Toc211333809)

[2.7. Navigation Implementation 56](#_Toc211333810)

[3. Triển khai Part 2 57](#_Toc211333811)

[3.1. CustomSwitch Implementation 57](#_Toc211333812)

[3.2. CustomCheckbox Implementation 59](#_Toc211333813)

[3.3. CustomRadio Implementation 61](#_Toc211333814)

[3.4. CustomSlider Implementation 63](#_Toc211333815)

[3.5. CustomGestureDetector Implementation 66](#_Toc211333816)

[3.6. CustomInkWell Implementation 70](#_Toc211333817)

[3.7. Part 2 Page Organization 74](#_Toc211333818)

[IV. KẾT QUẢ THỰC HIỆN 85](#_Toc211333819)

[1. Part 1: 85](#_Toc211333820)

[1.1. Chức năng đã triển khai 85](#_Toc211333821)

[1.2. Luồng quản lý State 98](#_Toc211333822)

[1.3. Tái sử dụng Component 98](#_Toc211333823)

[1.4. Đặc điểm về hiệu năng 99](#_Toc211333824)

[1.5. Tính năng trải nghiệm người dùng 99](#_Toc211333825)

[2. Part 2 - Chức năng đã triển khai 99](#_Toc211333826)

[2.1. Điều khiển lựa chọn 99](#_Toc211333827)

[2.2. Nhận diện cử chỉ 101](#_Toc211333828)

[2.3. Tổ chức Card 103](#_Toc211333829)

[2.4. Luồng quản lý State 104](#_Toc211333830)

[2.5. Tái sử dụng Component 105](#_Toc211333831)

[V. ĐÁNH GIÁ VÀ KẾT LUẬN 106](#_Toc211333832)

[1. Đánh giá kết quả 106](#_Toc211333833)

[Ưu điểm 106](#_Toc211333834)

[Hạn chế 108](#_Toc211333835)

[2. Bài học kinh nghiệm 109](#_Toc211333836)

[3. Kết luận 111](#_Toc211333837)

[VI. TÀI LIỆU THAM KHẢO 112](#_Toc211333838)

# I. GIỚI THIỆU

## 1. Mục đích đồ án

Đồ án này nhằm nghiên cứu và triển khai toàn diện các widget tương tác trong Flutter, bao gồm:

* **Part 1:** Buttons (ElevatedButton, TextButton, OutlinedButton, IconButton) và FloatingActionButton variants
* **Part 2:** Selection controls (Switch, Checkbox, Radio, Slider) và gesture detection (GestureDetector, InkWell)

## 2. Phạm vi thực hiện

**Part 1 - Buttons và Basic Input:**

* Các loại Button theo Material Design 3
* FloatingActionButton với 4 biến thể
* TextField với validation
* Card widget organization
* State management với StatefulWidget/StatelessWidget

**Part 2 - Selection Controls và Gestures:**

* Switch widget cho toggle states
* Checkbox cho selection.
* Radio buttons cho mutually exclusive choices
* Slider cho continuous value selection
* GestureDetector cho custom gesture handling
* InkWell cho Material ink splash effects.

# II. CƠ SỞ LÝ THUYẾT

## 1. Part 1: Buttons, FloatingActionButton và TextField

### 1.1 Buttons trong Material Design

#### 1.1.1. ElevatedButton

ElevatedButton là nút có nền được tô màu và có độ nổi (elevation) khi được nhấn [2]. Theo Material Design 3, loại nút này được sử dụng để thêm chiều sâu cho các layout phẳng, đặc biệt hữu ích trong danh sách nội dung dài hoặc không gian rộng [2].

**Đặc điểm kỹ thuật:**

* Foreground color và background color được định nghĩa qua ButtonStyle
* Elevation tăng lên khi button được nhấn
* Không nên sử dụng trên các thành phần đã có elevation như dialog hoặc card [2]

**Code mẫu từ tài liệu:**

ElevatedButton(

onPressed: () {

debugPrint('ElevatedButton pressed');

},

child: const Text('Elevated Button'),

)

Trong ví dụ này, ElevatedButton hiển thị text "Elevated Button" và in debug message khi được nhấn [2].

#### 1.1.2. TextButton

TextButton là nút không có viền và không có màu nền, chỉ hiển thị text label [3]. Loại nút này thường được sử dụng trong toolbar, dialog, hoặc inline với nội dung khác nhưng cần có padding để tạo sự nổi bật [3].

**Nguyên tắc sử dụng:**

* Nên nhóm các TextButton lại với nhau ở góc dưới của dialog hoặc card
* Tránh sử dụng ở vị trí dễ bị hòa lẫn với nội dung khác
* Button phản ứng với touch bằng cách fill màu backgroundColor [3]

**Code mẫu:**

TextButton(

onPressed: () {

debugPrint('TextButton pressed');

},

child: const Text('Text Button'),

)

TextButton có appearance tối giản nhất trong các button types, phù hợp cho secondary actions [3].

#### 1.1.3. OutlinedButton

OutlinedButton là loại nút có viền (outline) nhưng không có màu nền. Đây là nút có mức độ nhấn mạnh trung bình (medium emphasis), thường dùng cho các hành động quan trọng nhưng không phải hành động chính (primary action).

**Đặc điểm nổi bật:**

* Thuộc tính ButtonStyle.side mặc định quy định giao diện của đường viền (outline).
* Nếu muốn tùy chỉnh cả hình dạng (shape) và đường viền (outline), cần chỉ định đồng thời ButtonStyle.shape và ButtonStyle.side [4].
* Văn bản và biểu tượng (icon) trên nút được hiển thị bằng màu foregroundColor [4].

**Code mẫu:**

OutlinedButton(

onPressed: () {

debugPrint('OutlinedButton pressed');

},

child: const Text('Outlined Button'),

)

Default style của OutlinedButton có đường viền (border outline) rõ ràng, giúp phân biệt với TextButton, nhưng mang mức độ nhấn mạnh thấp hơn ElevatedButton [4].

#### 1.1.4. IconButton

IconButton là loại nút hình tròn chỉ hiển thị biểu tượng (icon) và phản hồi khi chạm bằng hiệu ứng fill màu [5]. Trong Material Design 3, có bốn kiểu mới được giới thiệu: standard, filled, filled tonal, và outlined [5].

**Quy tắc về icon size:**

* Không nên ghi đè kích thước icon bằng thuộc tính Icon.size; thay vào đó, sử dụng thuộc tính iconSize của IconButton [5].
* Khu vực có thể chạm (hit region) phải có kích thước tối thiểu kMinInteractiveDimension pixels để đáp ứng yêu cầu về touch target size [5].

**Code mẫu từ tài liệu:**

IconButton(

icon: const Icon(Icons.volume\_up),

tooltip: 'Increase volume by 10',

onPressed: () {

setState(() {

\_volume += 10;

});

},

)

Ví dụ này cho thấy IconButton với icon volume\_up, có tooltip và tăng volume khi pressed [5].

### 1.2. FloatingActionButton (FAB)

FloatingActionButton (FAB) là nút hành động hình tròn, nổi lên trên nội dung để làm nổi bật primary action của màn hình [1]. FAB thường được sử dụng trong thuộc tính Scaffold.floatingActionButton [1].

#### 1.2.1. Các biến thể của FAB

**Code mẫu từ tài liệu - Small FAB:**

FloatingActionButton.small(

onPressed: () {},

child: const Icon(Icons.add),

)

**Code mẫu từ tài liệu - Standard FAB:**

FloatingActionButton(

onPressed: () {},

child: const Icon(Icons.add),

)

**Code mẫu từ tài liệu - Large FAB:**

FloatingActionButton.large(

onPressed: () {},

child: const Icon(Icons.add),

)

**Code mẫu từ tài liệu - Extended FAB:**

FloatingActionButton.extended(

onPressed: () {},

icon: const Icon(Icons.add),

label: const Text('Add'),

)

Extended FAB kết hợp cả icon và text label, cung cấp context rõ ràng hơn cho action [1].

#### 1.2.2. Nguyên tắc sử dụng FAB

* Tối đa một FAB trên mỗi màn hình
* Chỉ sử dụng cho positive action như "create", "share", hoặc "navigate"
* Nếu có nhiều FAB trong một Route, mỗi button cần có heroTag unique [1]
* Không nên disable FAB vì không có indication nào cho user [1]

### 1.3. TextField và Input Validation

TextField cho phép người dùng nhập văn bản bằng bàn phím vật lý hoặc bàn phím ảo trên màn hình [6]. Đây là widget cốt lõi được sử dụng để thu thập dữ liệu đầu vào từ người dùng trong ứng dụng.

#### 1.3.1. TextEditingController

Controller được sử dụng để:

* Quản lý nội dung văn bản hiển thị trong TextField.
* Thiết lập giá trị khởi tạo (initial value).
* Kiểm soát vùng chọn (selection) và vùng đang nhập (composing region).
* Theo dõi các thay đổi đối với văn bản, vùng chọn, và vùng đang nhập [6].

**Code mẫu từ tài liệu - TextField với Controller:**

final TextEditingController \_controller = TextEditingController();

TextField(

controller: \_controller,

decoration: const InputDecoration(

border: OutlineInputBorder(),

labelText: 'Enter your text',

),

)

**Code mẫu từ tài liệu - TextField với Validation:**

TextField(

decoration: InputDecoration(

border: const OutlineInputBorder(),

labelText: 'Email',

errorText: \_errorText,

),

onChanged: (String value) {

setState(() {

\_errorText = value.contains('@') ? null : 'Invalid email';

});

},

)

Trong ví dụ này, TextField hiển thị error text khi email không hợp lệ [6].

#### 1.3.2. Lifecycle của TextField

Khi người dùng hoàn thành editing:

* Bước 1: Quá trình chỉnh sửa được hoàn tất (finalized) và callback onChanged được gọi (theo mặc định).
* Bước 2: Callback onSubmitted được kích hoạt (invoke) với giá trị đầu vào của người dùng [6].

#### 1.3.3. Best Practices

* Luôn dispose TextEditingController khi không còn sử dụng để tránh rò rỉ bộ nhớ.
* Sử dụng characters.length thay vì string.length để xử lý đúng các emoji và ký tự phức tạp.
* Khi TextField nằm trong scrolling container, nên chỉ định controller và quản lý vòng đời (lifecycle) của nó từ Stateful ancestor [6].

### 1.4. Card Widget

Card là widget trong Material Design dùng để tạo một panel có góc bo tròn và hiệu ứng đổ bóng (shadow) [11]. Card thường được sử dụng để nhóm các thông tin và hành động có liên quan lại với nhau.

**Đặc điểm của Card:**

* Có elevation mặc định để tạo hiệu ứng đổ bóng (shadow).
* Góc bo tròn tuân theo Material Design guidelines.
* Có thể chứa bất kỳ widget nào bên trong.
* Hỗ trợ clip behavior để kiểm soát cách hiển thị nội dung con (child content) [11].

**Code mẫu từ tài liệu - Basic Card:**

Card(

child: Column(

mainAxisSize: MainAxisSize.min,

children: <Widget>[

const ListTile(

leading: Icon(Icons.album),

title: Text('The Enchanted Nightingale'),

subtitle: Text('Music by Julie Gable. Lyrics by Sidney Stein.'),

),

],

),

)

**Code mẫu từ tài liệu - Card với Elevation:**

Card(

elevation: 8.0,

margin: const EdgeInsets.all(8.0),

child: Padding(

padding: const EdgeInsets.all(16.0),

child: Text('Card with elevation'),

),

)

Card với elevation cao hơn sẽ tạo bóng đổ (shadow) đậm và sâu hơn, giúp thể hiện rõ thứ bậc trực quan (visual hierarchy) giữa các thành phần trong giao diện [11].

### 1.5. State Management: StatefulWidget và StatelessWidget

#### 1.5.1. StatelessWidget

StatelessWidget là widget không yêu cầu trạng thái có thể thay đổi (mutable state) [12]. Widget này hữu ích khi phần giao diện (UI) chỉ phụ thuộc vào thông tin cấu hình (configuration information) trong object và BuildContext, chứ không thay đổi theo thời gian [12].

**Đặc điểm của StatelessWidget:**

* Immutable – không thể thay đổi sau khi được khởi tạo.
* Phương thức build chỉ được gọi trong ba trường hợp:
  1. Khi widget được thêm lần đầu vào cây (widget tree).
  2. Khi widget cha thay đổi cấu hình (configuration).
  3. Khi InheritedWidget mà nó phụ thuộc vào có sự thay đổi [12].
* Hiệu năng tốt vì quá trình build xảy ra ít lần.
* Phù hợp cho UI tĩnh hoặc các thành phần chỉ phụ thuộc vào tham số truyền qua constructor [12].

**Code mẫu từ tài liệu - Basic StatelessWidget:**

class GreenFrog extends StatelessWidget {

const GreenFrog({ super.key });

@override

Widget build(BuildContext context) {

return Container(color: const Color(0xFF2DBD3A));

}

}

**Code mẫu từ tài liệu - StatelessWidget với Parameters:**

class Frog extends StatelessWidget {

const Frog({

super.key,

this.color = const Color(0xFF2DBD3A),

this.child,

});

final Color color;

final Widget? child;

@override

Widget build(BuildContext context) {

return ColoredBox(color: color, child: child);

}

}

Widget này chấp nhận các tham số color và child, tuy nhiên vẫn giữ tính bất biến (immutable) — nghĩa là sau khi được khởi tạo, các giá trị này không thể thay đổi [12].

#### 1.5.2. StatefulWidget

StatefulWidget là widget có trạng thái có thể thay đổi (mutable state) [13]. State là phần thông tin có thể được đọc đồng bộ trong quá trình build và thay đổi trong suốt vòng đời của widget [13].

Đặc điểm của StatefulWidget:

* Instance của StatefulWidget là immutable — không thể thay đổi sau khi được tạo.
* Trạng thái có thể thay đổi (mutable state) được lưu trữ trong một đối tượng State riêng biệt.
* Framework sẽ gọi createState() mỗi khi StatefulWidget được khởi tạo (inflate).
* Một StatefulWidget có thể có nhiều đối tượng State khác nhau nếu nó được chèn vào nhiều vị trí trong cây widget [13].

**Code mẫu từ tài liệu - Basic StatefulWidget:**

class YellowBird extends StatefulWidget {

const YellowBird({ super.key });

@override

State<YellowBird> createState() => \_YellowBirdState();

}

class \_YellowBirdState extends State<YellowBird> {

@override

Widget build(BuildContext context) {

return Container(color: const Color(0xFFFFE306));

}

}

**Code mẫu từ tài liệu - StatefulWidget với State:**

class Bird extends StatefulWidget {

const Bird({

super.key,

this.color = const Color(0xFFFFE306),

this.child,

});

final Color 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 Container(

color: widget.color,

transform: Matrix4.diagonal3Values(\_size, \_size, 1.0),

child: widget.child,

);

}

}

Ví dụ này minh họa một State object có trạng thái nội bộ (\_size) và một phương thức (grow) dùng setState() để thay đổi trạng thái (mutate state) [13].

Kỹ thuật để giảm thiểu tác động của việc rebuild:

* Đưa state xuống các widget lá (leaves) để giới hạn phạm vi rebuild.
* Giảm số lượng node được tạo trong build method.
* Cache các widget không thay đổi trong các biến final để tránh tạo lại nhiều lần.
* Sử dụng const widgets khi có thể để tối ưu hiệu năng.
* Tránh thay đổi độ sâu (depth) của các subtree trong cây widget.
* Dùng GlobalKey để giữ tính nhất quán (consistency) cho các subtree khi độ sâu thay đổi [13].

## 2. Part 2 Widgets - Selection Controls và Gestures:

### 2.1. Switch Widget

Switch là widget trong Material Design được sử dụng để bật/tắt (toggle) trạng thái của một tùy chọn đơn lẻ (single setting). Switch không tự quản lý trạng thái, mà sẽ gọi callback onChanged khi trạng thái thay đổi.

Đặc điểm kỹ thuật:

* Bị vô hiệu hóa (disabled) khi onChanged là null.
* Thumb và track được hiển thị với tông màu xám khi ở trạng thái disabled.
* Material Design 3 hỗ trợ icon trên thumb thông qua thuộc tính Switch.thumbIcon.
* Cần có widget Material trong cây cha (Material ancestor widget).

Hành vi mặc định:

* Switch phản hồi với cử chỉ chạm (tap gestures).
* Thumb trượt trên track với hiệu ứng animation mượt mà.
* Cung cấp phản hồi trực quan (visual feedback) thông qua thay đổi màu sắc.
* Có thể tùy chỉnh giao diện bằng WidgetStateProperty.

**Code mẫu từ tài liệu:**

bool \_lights = false;

Switch(

value: \_lights,

onChanged: (bool value) {

setState(() {

\_lights = value;

});

},

)

**Customization với WidgetStateProperty:**

Switch(

value: \_lights,

activeColor: Colors.green,

inactiveThumbColor: Colors.grey,

onChanged: (bool value) {

setState(() {

\_lights = value;

});

},

)

**Các tính năng mới trong Material Design 3:**

* Hỗ trợ ThumbIcon tùy theo WidgetStates (ví dụ: trạng thái bật, tắt, hoặc disabled).
* Thiết kế trực quan được cập nhật với hình dạng bo tròn hiện đại và mềm mại hơn.
* Tăng khả năng tùy chỉnh màu sắc, cho phép điều chỉnh màu thumb, track, và hiệu ứng trạng thái linh hoạt hơn.

### 2.2. Checkbox Widget

Checkbox là widget trong Material Design dùng để biểu thị trạng thái chọn (checked) hoặc không chọn (unchecked). Giống như Switch, Checkbox không tự quản lý trạng thái nội bộ, mà sẽ gọi callback onChanged khi trạng thái thay đổi, và widget cha (parent) sẽ rebuild Checkbox với giá trị mới.

Hỗ trợ Tristate:

* Thuộc tính value có thể nhận true, false, hoặc null.
* Khi tristate = true và value = null, Checkbox hiển thị dấu gạch ngang (dash) thay vì dấu tick.
* Mặc định, tristate = false, nên chỉ có hai trạng thái: true/false.

Đặc điểm kỹ thuật:

* Cần có Material ancestor widget trong cây widget.
* Cung cấp phản hồi trực quan (visual feedback) thông qua animation của dấu checkmark.
* Hỗ trợ hiển thị trạng thái lỗi (error state visualization).
* Có thể tùy chỉnh giao diện bằng WidgetStateProperty.

**Code mẫu từ tài liệu:**

bool? \_isChecked = false;

Checkbox(

value: \_isChecked,

onChanged: (bool? value) {

setState(() {

\_isChecked = value;

});

},

)

**Customization example:**

Checkbox(

value: \_isChecked,

onChanged: (bool? value) {

setState(() {

\_isChecked = value;

});

},

activeColor: Colors.green,

checkColor: Colors.white,

side: BorderSide(color: Colors.grey, width: 2),

)

WidgetStateProperty override: Cho phép tùy chỉnh màu sắc và các thuộc tính khác dựa trên trạng thái của widget (như pressed, hovered, focused, disabled, v.v.). Điều này giúp tạo ra phản hồi trực quan động (dynamic visual feedback), giúp người dùng dễ nhận biết trạng thái tương tác của Checkbox hoặc các widget khác.

### 2.3. Radio Widget

Radio là widget trong Material Design được xây dựng dựa trên RawRadio để cung cấp giao diện Material UI. Widget này được dùng khi người dùng cần chọn một tùy chọn duy nhất trong nhiều lựa chọn. Radio sử dụng kiểu dữ liệu tổng quát (generic type parameter) T để định nghĩa loại giá trị mà nó biểu diễn.

**Kiến trúc hoạt động:**

* Thông thường, các Radio nằm trong một RadioGroup ancestor để quản lý nhóm lựa chọn.
* RadioGroup.groupValue xác định tùy chọn hiện đang được chọn.
* Radio có value trùng khớp với groupValue sẽ được hiển thị là selected.
* Kiểu dữ liệu T thường được khai báo bằng enum để biểu diễn các tùy chọn rõ ràng.

**Đặc điểm kỹ thuật:**

* Hiển thị dưới dạng nút tròn, với tâm được tô đầy khi được chọn.
* Trong cùng một nhóm, chỉ có một Radio được chọn tại một thời điểm.
* Khi một Radio được chọn, các Radio khác trong cùng nhóm sẽ tự động bỏ chọn (deselect).
* Cần có Material ancestor widget trong cây widget để hiển thị đúng giao diện.

**Code mẫu từ tài liệu:**

enum SingingCharacter { lafayette, jefferson }

SingingCharacter? \_character = SingingCharacter.lafayette;

Radio<SingingCharacter>(

value: SingingCharacter.lafayette,

groupValue: \_character,

onChanged: (SingingCharacter? value) {

setState(() {

\_character = value;

});

},

)

Tùy chỉnh với WidgetStateProperty:

* fillColor có thể thay đổi dựa trên trạng thái lựa chọn (selection state).
* Có thể tùy chỉnh màu viền (border color) thông qua thuộc tính side.
* Hỗ trợ tùy chỉnh màu nền (backgroundColor) để phù hợp với chủ đề giao diện.

Best Practices:

* Bọc Radio trong ListTile để hiển thị cùng nhãn (label), hoặc sử dụng sẵn RadioListTile cho tiện lợi.
* Dùng enum cho giá trị của Radio để đảm bảo type-safe và dễ quản lý.
* Nhóm các Radio có liên quan lại với nhau theo logic để cải thiện tính rõ ràng và trải nghiệm người dùng.

### 2.4. Slider Widget

Slider là widget trong Material Design dùng để chọn giá trị trong một khoảng (range of values). Widget này hỗ trợ cả chế độ liên tục (continuous) và rời rạc (discrete).

**Chế độ giá trị (Value modes):**

* Continuous: Cho phép chọn bất kỳ giá trị nào từ min đến max.
* Discrete: Sử dụng thuộc tính divisions để tạo các khoảng giá trị cố định.
  + *Ví dụ:* min = 0, max = 50, divisions = 5 → các giá trị có thể chọn: 0, 10, 20, 30, 40, 50.

**Cấu trúc Slider (Slider anatomy):**

* Thumb: Hình tròn có thể trượt ngang khi người dùng kéo.
* Track: Đường thẳng mà thumb di chuyển trên đó.
* Value indicator: Hộp nổi hiển thị giá trị hiện tại khi đang kéo.
* Active side: Phần track từ giá trị nhỏ nhất đến vị trí của thumb.
* Inactive side: Phần track từ vị trí của thumb đến giá trị lớn nhất.

**Đặc điểm kỹ thuật:**

* Bị vô hiệu hóa (disabled) khi onChanged là null hoặc khi min == max.
* Không tự quản lý trạng thái — gọi callback onChanged để thông báo giá trị mới.
* Chiều rộng mặc định: khoảng 144 pixels cho phần track (kèm margins).
* Cần có Material và MediaQuery ancestor widgets trong cây widget để hoạt động chính xác.

**Code mẫu từ tài liệu:**

double \_currentSliderValue = 20;

Slider(

value: \_currentSliderValue,

min: 0,

max: 100,

divisions: 5,

label: \_currentSliderValue.round().toString(),

onChanged: (double value) {

setState(() {

\_currentSliderValue = value;

});

},

)

**Callbacks khả dụng:**

* onChanged – Được gọi trong quá trình thay đổi giá trị khi người dùng kéo thumb.
* onChangeStart – Được gọi khi người dùng bắt đầu kéo.
* onChangeEnd – Được gọi khi người dùng thả thumb, kết thúc thao tác kéo**.**

**Tùy chỉnh (Customization):**

* activeColor và inactiveColor dùng để tùy chỉnh nhanh màu sắc của hai phần track.
* SliderThemeData cung cấp kiểm soát chi tiết (fine-grained control) cho các thành phần như thumb, track, tick marks, và value indicator.
* secondaryTrackValue cho phép hiển thị track phụ (secondary track) — thường dùng để biểu diễn mức tham chiếu hoặc giá trị phụ.

### 2.5. GestureDetector

GestureDetector là widget dùng để nhận diện và xử lý các cử chỉ (gestures) của người dùng. Nó sẽ cố gắng nhận dạng các cử chỉ tương ứng với những callback không null (ví dụ: onTap, onDoubleTap, onPanUpdate, v.v.).

**Kiến trúc hoạt động (Architecture):**

* Nếu có child, GestureDetector dựa vào child để xác định kích thước hiển thị.
* Nếu không có child, nó sẽ mở rộng để khớp với kích thước của parent.
* Child vô hình (invisible) mặc định không nhận sự kiện chạm, nhưng hành vi này có thể điều chỉnh thông qua thuộc tính behavior.
* Lắng nghe sự kiện trợ năng (accessibility events) và chuyển chúng đến các callback tương ứng.

**Hành vi hit testing (Hit testing behavior):**

* HitTestBehavior.opaque – Chặn cử chỉ, ngăn không cho gesture truyền qua các widget phía sau.
* HitTestBehavior.translucent – Cho phép gesture đi xuyên qua, trong khi vẫn kích hoạt callback của widget hiện tại.
* HitTestBehavior.deferToChild – (Mặc định) phụ thuộc vào child để quyết định vùng nhận chạm.

**Gesture Arena (Cơ chế phân giải gesture):**

1. Khi nhiều GestureDetector chồng lên nhau, tất cả đều tham gia vào "gesture arena".
2. Detector nào nhận dạng gesture đầu tiên sẽ thắng (win).
3. Các callback của detector thắng sẽ được gọi, còn các detector thua sẽ bị hủy (cancel).
4. Child detectors có độ ưu tiên cao hơn parent detectors (vì được xét trước — *first come, first served*).

**Code mẫu từ tài liệu - Basic tap:**

GestureDetector(

onTap: () {

print('Container was tapped');

},

child: Container(

color: Colors.blue,

child: Text('Tap me'),

),

)

**Code mẫu từ tài liệu - Multiple gestures:**

GestureDetector(

onTap: () {

print('Tap');

},

onDoubleTap: () {

print('Double tap');

},

onLongPress: () {

print('Long press');

},

child: Container(

padding: EdgeInsets.all(12),

decoration: BoxDecoration(

color: Colors.lightBlue,

borderRadius: BorderRadius.circular(8),

),

child: Text('Gesture me'),

),

)

**Các gesture callback khả dụng:**

* Chạm (Tap): onTap, onTapDown, onTapUp, onTapCancel
* Nhấn đúp: onDoubleTap
* Nhấn giữ (Long press): onLongPress, onLongPressUp
* Kéo (Pan): onPanStart, onPanUpdate, onPanEnd
* Phóng to/thu nhỏ (Scale): onScaleStart, onScaleUpdate, onScaleEnd
* Và nhiều callback khác hỗ trợ các cử chỉ phức tạp hơn (như drag, fling, secondary tap, v.v.).

**Debugging:**

* Đặt debugPrintGestureArenaDiagnostics = true để xem thông tin chi tiết về quá trình xử lý trong gesture arena.
* Đặt debugPaintPointersEnabled = true để hiển thị trực quan vùng hit test (hộp phát hiện chạm) — hữu ích khi debug vấn đề liên quan đến vùng nhận gesture.

### 2.6. InkWell

InkWell là một vùng hình chữ nhật (rectangular area) của Material dùng để phản hồi với các tương tác chạm (touch interactions). Đây là một biến thể của InkResponse nhưng bắt buộc duy trì hình dạng chữ nhật.

**Hiệu ứng mực (Material ink splash effects):**

* Hiệu ứng mực (ink reactions) được vẽ trên lớp Material bên dưới widget.
* Tạo phản hồi trực quan (visual feedback) với hiệu ứng lan tỏa của giọt mực (spreading ink effect) khi người dùng chạm.
* Tuân theo các nguyên tắc thiết kế của Material Design, mang lại cảm giác tương tác tự nhiên và thống nhất.

**Yêu cầu quan trọng (Critical requirement):**

* Bắt buộc phải có widget Material làm ancestor.
* Các hiệu ứng mực (ink effects) được vẽ bên trong lớp Material, không phải phía trên nó.
* Nếu không có Material ancestor, hiệu ứng splash sẽ không hiển thị (vì không có nơi để “lan mực”).

**Code mẫu từ tài liệu:**

InkWell(

onTap: () {

print('InkWell tapped');

},

child: Container(

padding: EdgeInsets.all(12),

child: Text('Tap for splash'),

),

)

**Vấn đề thường gặp – Ink splash không hiển thị:**

**Nguyên nhân:**Khi có một thành phần đồ họa đục (opaque graphic) như Container, Image, hoặc DecoratedBox nằm giữa Material và InkWell, nó sẽ che phủ lớp ink splash, khiến hiệu ứng không hiển thị được.

**Giải pháp 1:**Sử dụng Ink widget thay cho Container hoặc DecoratedBox.  
Ink được thiết kế đặc biệt để vẽ hình nền (decoration) bên trong cùng lớp Material, do đó hiệu ứng splash vẫn hiển thị bình thường.

Material(

child: InkWell(

onTap: () {},

child: Ink(

decoration: BoxDecoration(color: Colors.blue),

child: Text('Tap me'),

),

),

)

**Giải pháp 2:**Thêm một Material thứ hai với MaterialType.transparency để cho phép InkWell hiển thị splash mà không che nền phía sau.

Điều này hữu ích khi bạn không thể thay thế widget nền (như Container hoặc Image) bằng Ink, nhưng vẫn muốn splash xuất hiện.

Material(

child: Stack(

children: [

OpaqueWidget(),

Material(

type: MaterialType.transparency,

child: InkWell(

onTap: () {},

),

),

],

),

)

**Tùy chọn tùy chỉnh:**• splashColor – Màu của hiệu ứng splash khi chạm.  
• highlightColor – Màu hiển thị khi người dùng nhấn và giữ.  
• borderRadius – Định dạng bo góc cho vùng splash.  
• splashFactory – Xác định kiểu hiệu ứng splash (ví dụ: *InkRipple*, *InkSplash*).

**Code mẫu - Custom splash:**

Material(

color: Colors.transparent,

child: InkWell(

onTap: () {},

splashColor: Colors.tealAccent.shade200,

highlightColor: Colors.teal.withOpacity(0.3),

borderRadius: BorderRadius.circular(20),

splashFactory: InkRipple.splashFactory,

child: Ink(

padding: EdgeInsets.all(20),

decoration: BoxDecoration(

color: Colors.teal.shade50,

borderRadius: BorderRadius.circular(20),

),

child: Text('Tap me'),

),

),

)

**Vấn đề clipping:**Khi kích thước của Material ancestor thay đổi trong quá trình animation, hiệu ứng splash có thể bị cắt sai hoặc hiển thị không đúng. Để tránh lỗi này, không nên đặt InkWell bên trong các Material widget đang thay đổi kích thước trong animation.

**So sánh với GestureDetector:**• GestureDetector – Chỉ phát hiện cử chỉ, không có phản hồi trực quan.  
• InkWell – Cung cấp hiệu ứng splash theo chuẩn Material Design, mang lại phản hồi thị giác.  
• InkWell thực chất là một lớp stateless wrapper bao quanh InkResponse.  
• Dùng GestureDetector khi cần tùy chỉnh phản hồi, và InkWell khi muốn tuân theo phong cách Material Design.

# III. THIẾT KẾ VÀ TRIỂN KHAI

## 1. Kiến trúc ứng dụng

MyFullApp (StatefulWidget)

├── NavigationBar (2 destinations)

├── Part1Page (StatelessWidget)

│ ├── MyTextField (Card)

│ ├── MyActionButtonsCard (Card)

│ └── MyIconButtonsCard (Card)

└── Part2Page (StatefulWidget)

├── \_InteractiveControlsCard (StatelessWidget)

│ ├── CustomSwitch

│ ├── CustomCheckbox

│ └── CustomRadio (x2)

├── \_SliderCard (StatelessWidget)

│ └── CustomSlider

├── \_GestureDetectorCard (StatefulWidget)

│ └── CustomGestureDetector

└── \_InkWellCard (StatefulWidget)

└── CustomInkWell

## 2. Triển khai Part 1

### 2.1 Quản lý trạng thái

Ứng dụng sử dụng StatefulWidget ở root level với các state variables:

* \_nameController, \_emailController: TextEditingController cho input fields
* \_likeCount: Counter cho icon button interaction
* currentIndex: Index của NavigationBar
* \_result: String để hiển thị feedback
* \_nameError, \_emailError: Error messages cho validation

### 2.2 Triển khai Buttons

#### 2.2.1. ElevatedButton Implementation

**Code triển khai:**

class MyElevatedButton extends StatelessWidget {

final VoidCallback onPressed;

final String text;

const MyElevatedButton({

super.key,

required this.onPressed,

required this.text,

});

@override

Widget build(BuildContext context) {

return SizedBox(

width: double.infinity,

child: ElevatedButton(

onPressed: onPressed,

child: Text(text),

style: ElevatedButton.styleFrom(

padding: EdgeInsets.symmetric(vertical: 16),

),

),

);

}

}

**Điểm mới so với cơ sở lý thuyết:**

• Widget được wrap trong SizedBox với width: double.infinity để button chiếm toàn bộ chiều ngang của container.

• Tùy chỉnh padding dọc (vertical) thành 16 thay vì dùng giá trị mặc định, giúp cải thiện cảm giác nhấn.

• onPressed và text được truyền qua constructor để tăng khả năng tái sử dụng widget.

• Áp dụng mẫu StatelessWidget [12] vì widget không có trạng thái nội bộ.

• Sử dụng const constructor theo best practice cho StatelessWidget [12], giúp tối ưu hiệu năng và hạn chế rebuilds không cần thiết.

#### 2.2.2. TextButton Implementation

**Code triển khai:**

class MyTextButton extends StatelessWidget {

final VoidCallback onPressed;

final String text;

const MyTextButton({

super.key,

required this.onPressed,

required this.text

});

@override

Widget build(BuildContext context) {

return SizedBox(

width: double.infinity,

child: TextButton(

onPressed: onPressed,

child: Text(text),

style: TextButton.styleFrom(

padding: EdgeInsets.symmetric(vertical: 16),

),

),

);

}

}

**Điểm mới so với cơ sở lý thuyết:**

• Áp dụng layout full-width thông qua SizedBox wrapper, khác với ví dụ cơ bản trong tài liệu [3].

• Đảm bảo padding đồng nhất giữa các loại button nhằm duy trì sự thống nhất về mặt thị giác.

• Thiết kế theo mẫu component tái sử dụng thay vì khai báo TextButton trực tiếp trong widget tree.

• Tuân thủ quy ước của StatelessWidget với const constructor và final properties [12], giúp tăng hiệu năng và tính ổn định của widget.

#### 2.2.3. OutlinedButton Implementation

**Code triển khai:**

class MyOutlinedButton extends StatelessWidget {

final VoidCallback onPressed;

final String text;

const MyOutlinedButton({

super.key,

required this.onPressed,

required this.text,

});

@override

Widget build(BuildContext context) {

return SizedBox(

width: double.infinity,

child: OutlinedButton(

onPressed: onPressed,

child: Text(text),

style: OutlinedButton.styleFrom(

padding: EdgeInsets.symmetric(vertical: 16),

side: BorderSide(color: Colors.blue, width: 1.5),

),

),

);

}

}

**Điểm mới so với cơ sở lý thuyết:**

• Tùy chỉnh BorderSide với color: Colors.blue và width: 1.5 nhằm tạo đường viền rõ nét hơn so với mặc định.

• Theo guideline của OutlinedButton [4], cần chỉ định đồng thời shape và side để tùy chỉnh hoàn chỉnh giao diện nút.

• Sử dụng màu viền xanh (blue) thay cho màu mặc định của theme để đồng bộ với giao diện tổng thể của ứng dụng.

• Tăng độ dày viền từ 1.0 lên 1.5 giúp nút nổi bật và dễ nhận biết hơn trong bố cục.

#### 2.2.4. IconButton Implementation

**Code triển khai:**

class MyIconButton extends StatelessWidget {

final IconData icon;

final Color color;

final VoidCallback onPressed;

final String? label;

const MyIconButton({

super.key,

required this.icon,

required this.color,

required this.onPressed,

this.label,

});

@override

Widget build(BuildContext context) {

if (label != null) {

return Column(

children: [

IconButton(

icon: Icon(icon),

color: color,

iconSize: 40,

onPressed: onPressed,

),

Text(label!),

],

);

} else {

return IconButton(

icon: Icon(icon),

color: color,

iconSize: 40,

onPressed: onPressed,

);

}

}

}

**Điểm mới so với cơ sở lý thuyết:**

• Thêm label text tùy chọn hiển thị bên dưới icon bằng layout Column — chi tiết này không có trong ví dụ gốc của tài liệu [5].

• Tùy chỉnh iconSize: 40 thay vì kích thước mặc định để mở rộng vùng chạm, tuân theo Material Design touch target guideline [5].

• Áp dụng conditional rendering: nếu có label → trả về Column, nếu không → chỉ trả về IconButton đơn giản.

• Tham số hóa icon và color giúp tái sử dụng component cho nhiều mục đích khác nhau (ví dụ: “favorite” màu đỏ, “share” màu xanh).

• Tuân thủ quy tắc của IconButton [5] — không override trực tiếp Icon.size, mà sử dụng iconSize property để đảm bảo tính nhất quán về layout và accessibility.

### 2.3. Triển khai FloatingActionButton Variants

#### 2.3.1. Small FAB

**Code triển khai:**

class MyFloatingactionButton\_small extends StatelessWidget {

final VoidCallback onPressed;

const MyFloatingactionButton\_small({

super.key,

required this.onPressed

});

@override

Widget build(BuildContext context) {

return FloatingActionButton.small(

onPressed: onPressed,

child: Icon(Icons.add),

);

}

}

**Điểm mới so với cơ sở lý thuyết:**

• Được đóng gói trong một widget tái sử dụng thay vì khai báo trực tiếp (inline), giúp mã gọn gàng và dễ bảo trì hơn.

• Nhận callback onPressed từ parent, tuân theo nguyên tắc Single Responsibility, tách biệt logic điều khiển và hiển thị.

• Sử dụng icon Icons.add thống nhất cho tất cả các biến thể của FloatingActionButton, đảm bảo tính nhất quán về biểu tượng và hành vi.

#### 2.3.2. Standard FAB

**Code triển khai:**

class MyFloatingactionButton extends StatelessWidget {

final VoidCallback onPressed;

const MyFloatingactionButton({

super.key,

required this.onPressed

});

@override

Widget build(BuildContext context) {

return FloatingActionButton(

onPressed: onPressed,

child: Icon(Icons.add)

);

}

}

**Điểm mới:**Tương tự small FAB, component này cũng được đóng gói trong một widget tái sử dụng, giúp tái sử dụng dễ dàng và giữ cấu trúc mã nhất quán giữa các loại FAB.

#### 2.3.3. Large FAB

**Code triển khai:**

class MyFloatingactionButton\_large extends StatelessWidget {

final VoidCallback onPressed;

const MyFloatingactionButton\_large({

super.key,

required this.onPressed

});

@override

Widget build(BuildContext context) {

return FloatingActionButton.large(

onPressed: onPressed,

child: Icon(Icons.add),

);

}

}

**Điểm mới:** Sử dụng constructor FloatingActionButton.large [1] để tạo nút FAB kích thước lớn hơn, giúp tăng khả năng nhận diện và phù hợp với các hành động quan trọng hơn trong giao diện.

#### 2.3.4. Extended FAB

**Code triển khai:**

class MyFloatingactionButton\_extended extends StatelessWidget {

final VoidCallback onPressed;

const MyFloatingactionButton\_extended({

super.key,

required this.onPressed

});

@override

Widget build(BuildContext context) {

return FloatingActionButton.extended(

onPressed: onPressed,

icon: Icon(Icons.stars),

label: Text('New'),

);

}

}

**Điểm mới so với cơ sở lý thuyết:**

• Sử dụng icon Icons.stars thay cho Icons.add để phân biệt với các FAB tiêu chuẩn.

• Thêm label “New” nhằm làm rõ ngữ cảnh và ý nghĩa của hành động [1].

• Kết hợp icon và label trong biến thể extended giúp tăng khả năng nhận biết và hiểu hành động của người dùng.

#### 2.3.5. FAB Stack Layout

**Code triển khai trong MyFullApp:**

floatingActionButton: currentIndex == 0

? Column(

mainAxisSize: MainAxisSize.min,

children: [

MyFloatingactionButton\_small(

onPressed: () {

setState(() {

\_result = '✅ Small FAB clicked';

});

},

),

const SizedBox(height: 15.0),

MyFloatingactionButton(

onPressed: () {

setState(() {

\_result = '✅ Standard FAB clicked';

});

},

),

const SizedBox(height: 15.0),

MyFloatingactionButton\_large(

onPressed: () {

setState(() {

\_result = '✅ Large FAB clicked';

});

},

),

const SizedBox(height: 15.0),

MyFloatingactionButton\_extended(

onPressed: () {

setState(() {

\_result = '✅ Extended FAB clicked';

});

},

),

],

)

: null,

**Điểm mới so với cơ sở lý thuyết:**

• Hiển thị đồng thời cả 4 biến thể FAB trong một cột dọc (Column) — khác với guideline khuyến nghị chỉ một FAB trên mỗi màn hình [1].

• Mục đích là trình diễn (demo) toàn bộ các biến thể trong cùng một giao diện.

• Sử dụng rendering có điều kiện: chỉ hiển thị khi currentIndex == 0.

• Thêm SizedBox(height: 15) để tạo khoảng cách giữa các FAB, giúp bố cục rõ ràng hơn.

• Mỗi FAB có feedback message riêng để dễ phân biệt hành động.

• Đặt mainAxisSize: MainAxisSize.min để Column chỉ chiếm đúng không gian cần thiết, tránh giãn thừa.

### 2.4. Triển khai Card Organization

#### 2.4.1. MyTextField Card

**Code triển khai:**

class MyTextField extends StatelessWidget {

final TextEditingController nameController;

final TextEditingController emailController;

final String? nameError;

final String? emailError;

const MyTextField({

super.key,

required this.nameController,

required this.emailController,

this.nameError,

this.emailError,

});

@override

Widget build(BuildContext context) {

return Card(

elevation: 4,

margin: EdgeInsets.all(15),

child: Padding(

padding: EdgeInsets.all(16),

child: Column(

crossAxisAlignment: CrossAxisAlignment.start,

children: [

Text(

'TEXT INPUT',

style: TextStyle(

fontSize: 18,

fontWeight: FontWeight.bold,

color: Colors.blue,

),

),

SizedBox(height: 16),

TextField(

controller: nameController,

decoration: InputDecoration(

labelText: 'Full Name',

hintText: 'Enter your name',

prefixIcon: Icon(Icons.person),

border: OutlineInputBorder(),

errorText: nameError,

),

),

SizedBox(height: 12),

TextField(

controller: emailController,

decoration: InputDecoration(

labelText: 'Email',

hintText: 'example@email.com',

prefixIcon: Icon(Icons.email),

border: OutlineInputBorder(),

errorText: emailError,

),

keyboardType: TextInputType.emailAddress,

),

],

),

),

);

}

}

**Điểm mới so với cơ sở lý thuyết:**

• Bọc các TextField trong Card widget [11] với elevation: 4 để tạo hiệu ứng nổi và nhóm trực quan các trường nhập liệu.

• Áp dụng margin: 15 và padding: 16 cho Card nhằm tuân theo hierarchy về khoảng cách và bố cục.

• Thêm section title “TEXT INPUT” với màu xanh (blue) và chữ in đậm giúp phân tách rõ ràng giữa các phần nội dung.

• Chứa hai TextField trong Column, cách nhau bởi SizedBox(height: 12) để tạo khoảng trống hợp lý.

• Thêm prefixIcon (Icons.person, Icons.email) giúp tăng khả năng nhận biết và tính thân thiện với người dùng [6].

• Sử dụng errorText từ parent để hiển thị lỗi theo thời gian thực, hỗ trợ validation trực tiếp trong UI [6].

• Áp dụng keyboardType: TextInputType.emailAddress cho email field để tối ưu phương thức nhập liệu [6].

• Các TextEditingController được truyền từ parent, đảm bảo duy trì một nguồn dữ liệu thống nhất (single source of truth).

**Khác với example cơ bản [6]:**

• Giao diện được tổ chức thành nhóm có cấu trúc rõ ràng thay vì các input rời rạc.

• Có tiêu đề (section header) giúp người dùng dễ định hướng.

• Validation lỗi được tích hợp trực tiếp trong decoration, tăng tính trực quan.

• Giữ khoảng cách và phong cách nhất quán trên toàn bộ giao diện để nâng cao tính thẩm mỹ.

#### 2.4.2. MyActionButtonsCard

**Code triển khai:**

class MyActionButtonsCard extends StatelessWidget {

final VoidCallback onSubmit;

final VoidCallback onLearnMore;

final VoidCallback onClear;

const MyActionButtonsCard({

super.key,

required this.onSubmit,

required this.onLearnMore,

required this.onClear,

});

@override

Widget build(BuildContext context) {

return Card(

elevation: 4,

margin: EdgeInsets.all(8),

child: Padding(

padding: EdgeInsets.all(16),

child: Column(

crossAxisAlignment: CrossAxisAlignment.start,

children: [

Text(

'ACTION BUTTONS',

style: TextStyle(

fontSize: 18,

fontWeight: FontWeight.bold,

color: Colors.blue,

),

),

SizedBox(height: 16),

MyElevatedButton(onPressed: onSubmit, text: 'SUBMIT'),

SizedBox(height: 12),

MyOutlinedButton(onPressed: onLearnMore, text: 'LEARN MORE'),

SizedBox(height: 12),

MyTextButton(onPressed: onClear, text: 'CLEAR'),

],

),

),

);

}

}

**Điểm mới so với cơ sở lý thuyết:**

• Nhóm ba loại button (Elevated, Outlined, Text) trong một Card duy nhất [11], giúp trình bày trực quan và có tổ chức.

• Thể hiện thứ bậc thị giác rõ ràng: ElevatedButton (primary) → OutlinedButton (secondary) → TextButton (tertiary).

• Áp dụng elevation: 4 cho Card để đồng bộ với card chứa TextField, đảm bảo tính nhất quán.

• Thêm section title “ACTION BUTTONS” nhằm duy trì phong cách thống nhất giữa các card.

• Sắp xếp theo chiều dọc (vertical stacking) với SizedBox(height: 12) để tạo khoảng cách rõ ràng giữa các nút.

• Callbacks được truyền từ parent, tuân theo StatelessWidget pattern [12], giúp phân tách logic và UI hợp lý.

**Khác với các ví dụ trong tài liệu [2][3][4]:**

• Các button được tổ chức thành một nhóm cohesive trong cấu trúc card, thay vì đặt riêng lẻ.

• Có phân tách rõ ràng giữa các loại button dựa trên vai trò và mức độ ưu tiên.

• Đảm bảo spacing và alignment đồng nhất trên toàn bộ bố cục.

• Section header giúp người dùng dễ quét và nhận diện nhanh phần chức năng.

#### 2.4.3. MyIconButtonsCard

**Code triển khai:**

class MyIconButtonsCard extends StatelessWidget {

final VoidCallback onLike;

final VoidCallback onShare;

final int likeCount;

const MyIconButtonsCard({

super.key,

required this.onLike,

required this.onShare,

required this.likeCount,

});

@override

Widget build(BuildContext context) {

return Card(

elevation: 4,

margin: EdgeInsets.all(8),

child: Padding(

padding: EdgeInsets.all(16),

child: Column(

crossAxisAlignment: CrossAxisAlignment.start,

children: [

Text(

'ICON BUTTON',

style: TextStyle(

fontSize: 18,

fontWeight: FontWeight.bold,

color: Colors.blue,

),

),

SizedBox(height: 2),

Row(

mainAxisAlignment: MainAxisAlignment.spaceEvenly,

children: [

MyIconButton(

icon: Icons.favorite,

color: Colors.red,

onPressed: onLike,

label: 'Likes: $likeCount',

),

MyIconButton(

icon: Icons.share,

color: Colors.blue,

onPressed: onShare,

),

],

),

],

),

),

);

}

}

**Điểm mới so với cơ sở lý thuyết:**

• Sử dụng Card container với elevation: 4 để duy trì phong cách thống nhất với các thành phần khác [11].  
• Bố trí theo hàng ngang (Row layout) với mainAxisAlignment: spaceEvenly, giúp phân bố các nút đồng đều và cân đối.  
• Nút Like có label hiển thị số lượt thích (likeCount) — giá trị được truyền từ parent, giúp thể hiện trạng thái động.  
• Nút Share không có label để giữ giao diện gọn gàng, tập trung vào biểu tượng.  
• Sử dụng Icons.favorite (màu đỏ) và Icons.share (màu xanh) theo chuẩn giao diện phổ biến, giúp người dùng dễ nhận diện chức năng.  
• likeCount state được quản lý ở parent StatefulWidget [13], đảm bảo tách biệt logic và UI hợp lý.

**Khác với example trong tài liệu [5]:**• Các IconButton được nhóm trong một card có tổ chức, thay vì tách rời.  
• Kết hợp IconButton có label và không có label trong cùng một layout để trình bày linh hoạt hơn.  
• Hiển thị trạng thái động (like counter) trực tiếp trên UI, nâng cao tính tương tác.  
• Sử dụng bố cục ngang (horizontal) thay vì bố cục dọc hoặc dạng lưới, giúp gọn gàng và dễ nhìn hơn trong các thanh hành động.

### 2.5. Triển khai TextField với Validation

#### 2.5.1. Validation Logic

**Code triển khai trong MyFullApp:**

bool \_validateForm() {

bool isValid = true;

setState(() {

if (\_nameController.text.isEmpty) {

\_nameError = 'Please enter your name';

isValid = false;

} else if (\_nameController.text.length < 3) {

\_nameError = 'Name must be at least 3 characters';

isValid = false;

} else {

\_nameError = null;

}

if (\_emailController.text.isEmpty) {

\_emailError = 'Please enter your email';

isValid = false;

} else if (!\_emailController.text.contains('@gmail.com')) {

\_emailError = 'Please enter a valid email';

isValid = false;

} else {

\_emailError = null;

}

});

return isValid;

}

**Điểm mới so với cơ sở lý thuyết:**

• Áp dụng multi-stage validation, kiểm tra rỗng trước, sau đó mới kiểm tra định dạng, giúp phản hồi lỗi chính xác và tuần tự hơn.

• Thêm quy tắc cho tên (name) yêu cầu tối thiểu 3 ký tự, không có trong ví dụ gốc [6].

• Email validation sử dụng custom rule yêu cầu đuôi “@gmail.com”, thay vì chỉ kiểm tra định dạng email chung.

• Dùng setState() để cập nhật trạng thái lỗi (error states), đảm bảo UI được rebuild và phản ánh thay đổi ngay lập tức [13].

• Hàm validation trả về giá trị boolean, cho phép điều kiện hóa việc submit form (chỉ gửi khi hợp lệ).

• Khi validation pass, xóa lỗi bằng cách set error = null, đảm bảo giao diện luôn phản ánh đúng trạng thái hợp lệ.

**Khác với example trong tài liệu [6]:**

Ví dụ này nâng cấp từ simple validation thành comprehensive validation, với nhiều quy tắc kiểm tra cho từng field, giúp form chính xác, linh hoạt và thân thiện hơn với người dùng.

#### 2.5.2. Form Submission

**Code triển khai:**

void \_submitForm() {

if (\_validateForm()) {

setState(() {

\_result = '✅ Welcome, ${\_nameController.text}!\nEmail: ${\_emailController.text}';

});

} else {

setState(() {

\_result = '⚠️ Please fix the errors';

});

}

}

**Điểm mới:**

* Chỉ submit form khi validation thành công
* Thông báo thành công được cá nhân hóa dựa trên giá trị từ controller [6]
* Hiển thị lỗi kèm biểu tượng cảnh báo
* Cập nhật setState \_result để kích hoạt hiển thị trên AppBar [13]
* Kết quả hiển thị trên nhiều dòng với ký tự xuống dòng \n

#### 2.5.3. Clear Form Logic

**Code triển khai:**

void \_clearForm() {

\_nameController.clear();

\_emailController.clear();

setState(() {

\_result = '';

\_likeCount = 0;

\_nameError = null;

\_emailError = null;

});

}

**Điểm mới:**

* Reset nhiều trạng thái cùng lúc: controllers, result, counter, errors
* Sử dụng phương thức Controller.clear() [6] để xóa nội dung text fields
* setState reset toàn bộ trạng thái hiển thị trực quan [13]
* Reset toàn diện bao gồm cả bộ đếm like

#### 2.5.4. Controller Lifecycle Management

**Code triển khai trong MyFullApp:**

final TextEditingController \_nameController = TextEditingController();

final TextEditingController \_emailController = TextEditingController();

@override

void dispose() {

\_nameController.dispose();

\_emailController.dispose();

super.dispose();

}

**Điểm mới:**

* Controllers được khai báo như biến instance final [6]
* Override phương thức dispose() để tránh rò rỉ bộ nhớ
* Dọn dẹp đúng cách theo best practices của TextField [6]
* Gọi super.dispose() sau khi dispose các controllers

### 2.6. State Management Implementation

#### 2.6.1. Root StatefulWidget

**Code triển khai MyFullApp:**

class MyFullApp extends StatefulWidget {

const MyFullApp({super.key});

@override

State<MyFullApp> createState() => \_MyHomePageState();

}

class \_MyHomePageState extends State<MyFullApp> {

final TextEditingController \_nameController = TextEditingController();

final TextEditingController \_emailController = TextEditingController();

int \_likeCount = 0;

int currentIndex = 0;

String \_result = '';

String? \_nameError;

String? \_emailError;

@override

void dispose() {

\_nameController.dispose();

\_emailController.dispose();

super.dispose();

}

@override

Widget build(BuildContext context) {

return MaterialApp(

debugShowCheckedModeBanner: false,

theme: ThemeData(

colorScheme: ColorScheme.fromSeed(seedColor: Colors.teal),

scaffoldBackgroundColor: Colors.white,

),

home: Scaffold(

backgroundColor: Colors.white,

appBar: AppBar(

surfaceTintColor: Colors.transparent,

backgroundColor: Colors.white,

elevation: 0,

title: \_result.isNotEmpty

? Container(

padding: const EdgeInsets.all(30),

decoration: BoxDecoration(

color: Colors.blue.shade50,

borderRadius: BorderRadius.circular(30),

),

child: Text(

\_result,

style: const TextStyle(fontSize: 15, color: Colors.black87),

textAlign: TextAlign.center,

),

)

: const Text("Buttons và Input Widgets"),

centerTitle: true,

),

body: \_pages[currentIndex],

bottomNavigationBar: NavBarWidget(

currentIndex: currentIndex,

onDestinationSelected: (int index) {

setState(() {

currentIndex = index;

});

},

),

floatingActionButton: currentIndex == 0 ? /\* FAB stack \*/ : null,

),

);

}

}

**Điểm mới so với cơ sở lý thuyết:**

* Sử dụng pattern StatefulWidget [13] với class State riêng biệt là \_MyHomePageState
* Các biến state được khai báo như private fields với tiền tố gạch dưới
* Controllers và state có thể thay đổi (counters, errors) được lưu trong đối tượng State [13]
* Triển khai phương thức dispose() để giải phóng tài nguyên [13]
* Gọi setState() trong các callback methods để kích hoạt rebuild [13]
* Hiển thị có điều kiện cho AppBar title dựa trên state \_result
* Render body động bằng cách dùng list \_pages được truy cập qua currentIndex
* Hiển thị có điều kiện cho FAB dựa trên trạng thái navigation

**Theo hướng dẫn [13], đây là StatefulWidget loại 2 vì:**

* Sử dụng setState nhiều lần
* Rebuild nhiều lần trong suốt vòng đời
* Quản lý nhiều state có thể thay đổi

#### 2.6.2. Child StatelessWidgets

**Part1Page Implementation:**

class Part1Page extends StatelessWidget {

final TextEditingController nameController;

final TextEditingController emailController;

final String? nameError;

final String? emailError;

final int likeCount;

final VoidCallback onSubmit;

final VoidCallback onLearnMore;

final VoidCallback onClear;

final VoidCallback onLike;

final VoidCallback onShare;

const Part1Page({

super.key,

required this.nameController,

required this.emailController,

this.nameError,

this.emailError,

required this.likeCount,

required this.onSubmit,

required this.onLearnMore,

required this.onClear,

required this.onLike,

required this.onShare,

});

@override

Widget build(BuildContext context) {

return SingleChildScrollView(

padding: EdgeInsets.all(20),

child: Column(

children: [

MyTextField(

nameController: nameController,

emailController: emailController,

nameError: nameError,

emailError: emailError,

),

SizedBox(height: 20),

MyActionButtonsCard(

onSubmit: onSubmit,

onLearnMore: onLearnMore,

onClear: onClear,

),

SizedBox(height: 20),

MyIconButtonsCard(

likeCount: likeCount,

onLike: onLike,

onShare: onShare,

),

],

),

);

}

}

**Điểm mới so với cơ sở lý thuyết:**

* Là StatelessWidget [12] vì không có state có thể thay đổi bên trong
* Tất cả state và callbacks được truyền từ StatefulWidget cha
* Bọc bằng SingleChildScrollView để có thể cuộn nội dung
* Bố cục dạng Column với khoảng cách SizedBox là 20
* Ủy quyền quản lý state lên widget cha theo mô hình lifting state up
* Constructor const với các thuộc tính final [12]

**Theo hướng dẫn về hiệu suất [12]:**

* Tối thiểu hóa số node được tạo bởi phương thức build
* Sử dụng constructor const
* Không phụ thuộc vào state nên rebuild hiệu quả

#### 2.6.3. State Flow Architecture

**State flow pattern:**

User Action

↓

Callback fired (onPressed, onChanged)

↓

Parent StatefulWidget method called

↓

setState() updates state variables

↓

Framework rebuilds widget tree

↓

Updated state passed down to children

↓

UI reflects new state

**Ví dụ Like Button Flow:**

MyIconButton (StatelessWidget)

onPressed: onLike

↓

MyIconButtonsCard (StatelessWidget)

onLike: onLike

↓

Part1Page (StatelessWidget)

onLike: onLike

↓

MyFullApp.\_MyHomePageState (StatefulWidget)

onLike: () {

setState(() {

\_likeCount++;

\_result = '❤️ Liked! ($\_likeCount)';

});

}

**Điểm mới:**

* Truyền callback nhiều cấp từ widget lá lên widget gốc
* State được tập trung quản lý ở StatefulWidget gốc [13]
* Các widget con StatelessWidget chỉ đảm nhiệm phần hiển thị [12]
* Tách biệt rõ ràng giữa giao diện và logic xử lý
* Luồng dữ liệu một chiều (unidirectional data flow)

### 2.7. Navigation Implementation

**NavBarWidget Implementation:**

class NavBarWidget extends StatelessWidget {

final int currentIndex;

final Function(int) onDestinationSelected;

const NavBarWidget({

super.key,

required this.currentIndex,

required this.onDestinationSelected,

});

@override

Widget build(BuildContext context) {

return NavigationBar(

destinations: [

NavigationDestination(icon: Icon(Icons.home), label: "Part 1"),

NavigationDestination(icon: Icon(Icons.home\_work), label: "Part 2"),

],

onDestinationSelected: onDestinationSelected,

selectedIndex: currentIndex,

);

}

}

**Điểm mới:**

* Sử dụng widget NavigationBar cho thanh điều hướng dưới cùng theo Material 3
* Hai điểm đến: Part 1 (báo cáo cá nhân), Part 2 (báo cáo nhóm)
* selectedIndex được điều khiển từ state của widget cha
* Sử dụng callback pattern để ủy quyền logic điều hướng lên widget cha
* Là StatelessWidget vì chỉ hiển thị trạng thái hiện tại [12]

**Page Switching Logic:**

List<Widget> get \_pages => [

Part1Page(/\* pass all props \*/),

const Part2Page(),

];

body: \_pages[currentIndex],

**Điểm mới:**

* Phương thức getter \_pages tạo danh sách widget khi cần
* Chọn trang dựa trên chỉ số index
* Part1Page nhận toàn bộ state, Part2Page là trang giữ chỗ
* Chuyển trang hiệu quả mà không cần tạo lại toàn bộ cây widget

## 3. Triển khai Part 2

### 3.1. CustomSwitch Implementation

**Code triển khai:**

class CustomSwitch extends StatefulWidget {

final bool initialValue;

final ValueChanged<bool> onChanged;

const CustomSwitch({

super.key,

required this.initialValue,

required this.onChanged,

});

@override

State<CustomSwitch> createState() => \_CustomSwitchState();

}

class \_CustomSwitchState extends State<CustomSwitch> {

late bool \_isOn;

@override

void initState() {

super.initState();

\_isOn = widget.initialValue;

}

@override

Widget build(BuildContext context) {

return Switch(

value: \_isOn,

onChanged: (value) {

setState(() {

\_isOn = value;

});

widget.onChanged(value);

},

activeColor: Colors.green,

inactiveThumbColor: Colors.grey,

);

}

}

**Điểm mới so với cơ sở lý thuyết:**

* Bọc Switch trong StatefulWidget để quản lý state nội bộ
* initState khởi tạo \_isOn từ tham số widget.initialValue
* setState cập nhật state nội bộ trước khi gọi callback của widget cha
* Màu tùy chỉnh: xanh lá cho thumb khi active, xám khi inactive
* Từ khóa late cho \_isOn vì được khởi tạo trong initState
* Tách biệt state nội bộ (\_isOn) với callback bên ngoài

**Luồng xử lý state:**

* Người dùng bấm vào switch
* Callback onChanged được kích hoạt với giá trị mới
* setState cập nhật state \_isOn bên trong
* Callback của widget cha (widget.onChanged) được thông báo
* Widget rebuild với trạng thái hiển thị mới

Khác với ví dụ cơ bản, cách làm này cung cấp mô hình controlled component với cả quản lý state nội bộ lẫn bên ngoài.

### 3.2. CustomCheckbox Implementation

**Code triển khai:**

class CustomCheckbox extends StatefulWidget {

final bool initialValue;

final ValueChanged<bool> onChanged;

const CustomCheckbox({

super.key,

required this.initialValue,

required this.onChanged,

});

@override

State<CustomCheckbox> createState() => \_CustomCheckboxState();

}

class \_CustomCheckboxState extends State<CustomCheckbox> {

late bool \_isChecked;

@override

void initState() {

super.initState();

\_isChecked = widget.initialValue;

}

@override

Widget build(BuildContext context) {

return Checkbox(

value: \_isChecked,

onChanged: (value) {

setState(() {

\_isChecked = value ?? false;

});

widget.onChanged(\_isChecked);

},

activeColor: Colors.green,

checkColor: Colors.white,

side: const BorderSide(color: Colors.grey, width: 2),

);

}

}

**Điểm mới so với cơ sở lý thuyết:**

* Bọc trong StatefulWidget để quản lý state
* Xử lý null safety: value ?? false để xử lý giá trị null có thể có từ onChanged
* Tùy chỉnh giao diện: màu xanh lá khi active, dấu check màu trắng
* BorderSide với màu xám và độ dày 2 tạo viền nổi bật
* Chỉ dùng true/false, không hỗ trợ trạng thái null (non-tristate)

**Tùy chỉnh giao diện:**

* activeColor: Màu khi checkbox được chọn
* checkColor: Màu của biểu tượng dấu check
* side: Giao diện viền khi chưa chọn

Khác với ví dụ trong tài liệu, cách làm này không hỗ trợ chế độ tristate, chỉ tập trung vào trường hợp bật/tắt đơn giản.

### 3.3. CustomRadio Implementation

**Code triển khai:**

class CustomRadio<T> extends StatefulWidget {

final T value;

final T groupValue;

final ValueChanged<T> onChanged;

final String label;

const CustomRadio({

super.key,

required this.value,

required this.groupValue,

required this.onChanged,

required this.label,

});

@override

State<CustomRadio<T>> createState() => \_CustomRadioState<T>();

}

class \_CustomRadioState<T> extends State<CustomRadio<T>> {

@override

Widget build(BuildContext context) {

return Row(

mainAxisSize: MainAxisSize.min,

children: [

Radio<T>(

value: widget.value,

groupValue: widget.groupValue,

onChanged: (value) {

if (value != null) {

widget.onChanged(value);

}

},

activeColor: Colors.teal,

),

Text(

widget.label,

style: const TextStyle(fontSize: 16),

),

],

);

}

}

**Điểm mới so với cơ sở lý thuyết:**

* Tham số kiểu generic <T> cho phép dùng nhiều kiểu giá trị linh hoạt
* Nhãn được tích hợp trong bố cục Row (không cần dùng ListTile riêng)
* mainAxisSize.min để Radio và nhãn chỉ chiếm đủ không gian cần thiết
* Kiểm tra null trong onChanged trước khi gọi callback của widget cha
* Màu tùy chỉnh: teal thay vì màu mặc định của theme
* Dùng StatefulWidget dù không có state nội bộ (để giữ tính nhất quán)

**Bố cục:**

* Row với Radio button ở trước, nhãn ở sau
* Căn chỉnh theo chiều ngang tự nhiên
* Thiết kế gọn gàng phù hợp cho forms

An toàn kiểu: Generic type T cho phép kiểm tra kiểu dữ liệu tại thời điểm biên dịch cho giá trị radio, thường dùng với enums như trong ví dụ tài liệu.

### 3.4. CustomSlider Implementation

**Code triển khai:**

class CustomSlider extends StatefulWidget {

final double initialValue;

final double min;

final double max;

final int divisions;

final ValueChanged<double> onChanged;

const CustomSlider({

super.key,

required this.initialValue,

required this.onChanged,

this.min = 0,

this.max = 100,

this.divisions = 100,

});

@override

State<CustomSlider> createState() => \_CustomSliderState();

}

class \_CustomSliderState extends State<CustomSlider> {

late double \_currentValue;

@override

void initState() {

super.initState();

\_currentValue = widget.initialValue;

}

@override

Widget build(BuildContext context) {

return Column(

mainAxisSize: MainAxisSize.min,

children: [

Slider(

value: \_currentValue,

min: widget.min,

max: widget.max,

divisions: widget.divisions,

activeColor: Colors.teal,

label: \_currentValue.toStringAsFixed(1),

onChanged: (value) {

setState(() {

\_currentValue = value;

});

widget.onChanged(value);

},

),

Text(

'Giá trị: ${\_currentValue.toStringAsFixed(0)}',

style: const TextStyle(fontSize: 16),

),

],

);

}

}

**Điểm mới so với cơ sở lý thuyết:**

* Có thể tùy chỉnh khoảng giá trị: min, max, divisions với giá trị mặc định
* Hiển thị giá trị hiện tại bên dưới slider bằng widget Text
* Bố cục dạng Column: Slider ở trên, hiển thị giá trị ở dưới
* Định dạng nhãn: toStringAsFixed(1) cho tooltip (1 chữ số thập phân)
* Định dạng hiển thị: toStringAsFixed(0) để không có số thập phân
* divisions = 100 tạo slider rời rạc với 101 giá trị có thể chọn
* activeColor: teal để giữ giao diện nhất quán

**Định dạng giá trị:**

* Nhãn tooltip hiển thị 1 chữ số thập phân cho độ chính xác
* Text hiển thị chỉ số nguyên cho đơn giản
* Hai định dạng khác nhau phục vụ mục đích khác nhau

Tham số mặc định: min = 0, max = 100, divisions = 100 cung cấp giá trị hợp lý cho hầu hết các trường hợp sử dụng.

Khác với ví dụ cơ bản, cách làm này bao gồm hiển thị giá trị tích hợp và các tham số có thể cấu hình.

### 3.5. CustomGestureDetector Implementation

**Code triển khai:**

class CustomGestureDetector extends StatefulWidget {

final String text;

final VoidCallback? onTap;

final VoidCallback? onDoubleTap;

final VoidCallback? onLongPress;

const CustomGestureDetector({

super.key,

required this.text,

this.onTap,

this.onDoubleTap,

this.onLongPress,

});

@override

State<CustomGestureDetector> createState() => \_CustomGestureDetectorState();

}

class \_CustomGestureDetectorState extends State<CustomGestureDetector> {

Color \_color = Colors.teal.shade100;

String \_status = 'Chưa thao tác';

void \_resetStatus() {

if (!mounted) return;

setState(() {

\_color = Colors.teal.shade100;

\_status = 'Chưa thao tác';

});

}

void \_handleTap() {

setState(() {

\_color = Colors.teal.shade300;

\_status = 'Đã nhấn một lần';

});

widget.onTap?.call();

Future.delayed(const Duration(milliseconds: 500), \_resetStatus);

}

void \_handleDoubleTap() {

setState(() {

\_color = Colors.blueAccent.shade100;

\_status = 'Nhấn đúp';

});

widget.onDoubleTap?.call();

Future.delayed(const Duration(milliseconds: 500), \_resetStatus);

}

void \_handleLongPress() {

setState(() {

\_color = Colors.orangeAccent.shade100;

\_status = 'Giữ lâu';

});

widget.onLongPress?.call();

}

void \_handleLongPressUp() {

\_resetStatus();

}

@override

Widget build(BuildContext context) {

return GestureDetector(

onTap: \_handleTap,

onDoubleTap: \_handleDoubleTap,

onLongPress: \_handleLongPress,

onLongPressUp: \_handleLongPressUp,

child: AnimatedContainer(

duration: const Duration(milliseconds: 500),

padding: const EdgeInsets.all(20),

decoration: BoxDecoration(

color: \_color,

borderRadius: BorderRadius.circular(20),

boxShadow: const [

BoxShadow(

color: Colors.black26,

blurRadius: 6,

offset: Offset(2, 2),

),

],

),

child: Column(

mainAxisSize: MainAxisSize.min,

children: [

Text(

widget.text,

style: const TextStyle(fontSize: 20, fontWeight: FontWeight.bold),

),

const SizedBox(height: 10),

Text(

\_status,

style: const TextStyle(fontSize: 16),

),

],

),

),

);

}

}

**Điểm mới so với cơ sở lý thuyết:**

**Hệ thống phản hồi trực quan:**

* Màu sắc khác nhau cho từng thao tác:
  + Chạm: teal.shade300
  + Chạm hai lần: blueAccent.shade100
  + Chạm giữ: orangeAccent.shade100
* AnimatedContainer với thời gian 500ms tạo chuyển động mượt mà
* BoxShadow thêm chiều sâu và cải thiện thứ bậc trực quan

**Quản lý trạng thái:**

* \_color và \_status theo dõi trạng thái tương tác hiện tại
* Phương thức \_resetStatus có kiểm tra mounted để tránh rò rỉ bộ nhớ
* Tự động reset sau 500ms cho chạm và chạm hai lần
* Chạm giữ reset khi thả ra (onLongPressUp)

**Xử lý thao tác:**

* Các callback tùy chọn (onTap?, onDoubleTap?, onLongPress?)
* Gọi callback của widget cha sau khi cập nhật state nội bộ
* Cú pháp an toàn widget.onTap?.call()

**Các mẫu an toàn:**

* Kiểm tra mounted trong \_resetStatus tránh setState trên widget đã unmount
* Future.delayed cho việc reset theo thời gian
* Handler riêng biệt cho từng loại thao tác

**Bố cục:**

* Column với nhãn text và hiển thị trạng thái
* Bo góc (20) và padding (20) để dễ chạm
* mainAxisSize.min tối ưu sử dụng không gian

Khác với ví dụ GestureDetector cơ bản, cách làm này cung cấp phản hồi trực quan toàn diện, tính năng tự động reset, và minh họa nhiều loại thao tác cùng lúc.

### 3.6. CustomInkWell Implementation

**Code triển khai:**

class CustomInkWell extends StatefulWidget {

final String label;

final VoidCallback? onTap;

final VoidCallback? onDoubleTap;

final VoidCallback? onLongPress;

const CustomInkWell({

super.key,

required this.label,

this.onTap,

this.onDoubleTap,

this.onLongPress,

});

@override

State<CustomInkWell> createState() => \_CustomInkWellState();

}

class \_CustomInkWellState extends State<CustomInkWell> {

String \_status = "Chưa thao tác";

void \_handleTap() {

setState(() => \_status = "Bạn đã chạm 1 lần!");

widget.onTap?.call();

}

void \_handleDoubleTap() {

setState(() => \_status = "Bạn đã chạm đúp!");

widget.onDoubleTap?.call();

}

void \_handleLongPress() {

setState(() => \_status = "Bạn đã giữ lâu!");

widget.onLongPress?.call();

}

@override

Widget build(BuildContext context) {

return Material(

color: Colors.transparent,

child: InkWell(

onTap: \_handleTap,

onDoubleTap: \_handleDoubleTap,

onLongPress: \_handleLongPress,

borderRadius: BorderRadius.circular(20),

splashColor: Colors.tealAccent.shade200,

highlightColor: Colors.teal.withOpacity(0.3),

splashFactory: InkRipple.splashFactory,

child: Ink(

padding: const EdgeInsets.all(20),

decoration: BoxDecoration(

color: Colors.teal.shade50,

borderRadius: BorderRadius.circular(20),

border: Border.all(color: Colors.teal, width: 2),

),

child: Column(

mainAxisSize: MainAxisSize.min,

children: [

Text(

widget.label,

style: const TextStyle(

fontSize: 20,

fontWeight: FontWeight.bold,

),

),

const SizedBox(height: 10),

Text(

\_status,

style: const TextStyle(fontSize: 16, color: Colors.black87),

),

],

),

),

),

);

}

}

**Điểm mới so với cơ sở lý thuyết:**

**Cấu trúc Material (QUAN TRỌNG):**

* Material ngoài cùng có màu trong suốt (bắt buộc để InkWell hoạt động)
* Lớp InkWell xử lý nhận diện cử chỉ và hiệu ứng mực
* Widget Ink vẽ trang trí TRÊN Material (không phải phía trên)
* Cấu trúc này đảm bảo hiệu ứng gợn sóng hiển thị được

**Tùy chỉnh hiệu ứng mực:**

* splashColor: tealAccent.shade200 cho gợn sóng nổi bật
* highlightColor: teal với độ mờ 0.3 cho highlight nhẹ nhàng
* splashFactory: InkRipple.splashFactory cho hiệu ứng gợn sóng chuẩn
* borderRadius: 20 cắt hiệu ứng theo góc bo tròn

**So sánh với GestureDetector:**

* InkWell: Hiệu ứng splash theo Material Design, phản hồi tự động
* GestureDetector (trước đó): Đổi màu tùy chỉnh, animation thủ công
* InkWell đơn giản hơn cho ứng dụng Material Design
* GestureDetector linh hoạt hơn cho phản hồi tùy chỉnh

Sử dụng widget Ink: Theo hướng dẫn tài liệu, dùng Ink thay vì Container để đảm bảo decoration vẽ phía dưới hiệu ứng splash. Nếu không có Ink, Container không trong suốt sẽ che mất hiệu ứng splash.

**Kiểu viền:**

* Nền: teal.shade50 (màu nhạt)
* Viền: màu teal, độ dày 2 (viền rõ ràng)
* Góc bo tròn khớp với vùng cắt splash

**Quản lý trạng thái:**

* Đơn giản hơn phiên bản GestureDetector (không tự động reset)
* Trạng thái giữ nguyên đến thao tác tiếp theo
* Tập trung vào demo hiệu ứng mực Material

Hỗ trợ thao tác: Các callback giống GestureDetector (chạm, chạm hai lần, chạm giữ), nhưng với phản hồi trực quan theo Material Design.

### 3.7. Part 2 Page Organization

#### 3.7.1. Part2Page Structure

**Code triển khai:**

class Part2Page extends StatefulWidget {

const Part2Page({super.key});

@override

State<Part2Page> createState() => \_Part2PageState();

}

class \_Part2PageState extends State<Part2Page> {

bool \_switchValue = false;

bool \_checkValue = false;

double \_sliderValue = 0;

String \_radioValue = 'Option 1';

@override

Widget build(BuildContext context) {

return SingleChildScrollView(

padding: const EdgeInsets.all(20),

child: Column(

children: [

\_InteractiveControlsCard(

switchValue: \_switchValue,

checkValue: \_checkValue,

radioValue: \_radioValue,

onSwitchChanged: (value) => setState(() => \_switchValue = value),

onCheckChanged: (value) => setState(() => \_checkValue = value),

onRadioChanged: (value) => setState(() => \_radioValue = value),

),

const SizedBox(height: 20),

\_SliderCard(

sliderValue: \_sliderValue,

onSliderChanged: (value) => setState(() => \_sliderValue = value),

),

const SizedBox(height: 20),

const \_GestureDetectorCard(),

const SizedBox(height: 20),

const \_InkWellCard(),

],

),

);

}

}

**Điểm mới so với Part 1:**

* Part2Page là StatefulWidget (khác với Part1Page là StatelessWidget)
* Quản lý state cho nhiều điều khiển tương tác
* Bốn biến state: switch, checkbox, slider, radio
* SingleChildScrollView để nội dung có thể cuộn
* Khoảng cách: 20 pixels giữa các card

**Cách quản lý state:**

* State tập trung ở cấp độ page
* Các callback cập nhật state của widget cha
* Các card con nhận state và callback qua props
* Cú pháp arrow function cho các lệnh setState ngắn gọn

#### 3.7.2. \_InteractiveControlsCard

**Code triển khai:**

class \_InteractiveControlsCard extends StatelessWidget {

final bool switchValue;

final bool checkValue;

final String radioValue;

final ValueChanged<bool> onSwitchChanged;

final ValueChanged<bool> onCheckChanged;

final ValueChanged<String> onRadioChanged;

const \_InteractiveControlsCard({

required this.switchValue,

required this.checkValue,

required this.radioValue,

required this.onSwitchChanged,

required this.onCheckChanged,

required this.onRadioChanged,

});

@override

Widget build(BuildContext context) {

return Card(

elevation: 4,

shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(16)),

child: Padding(

padding: const EdgeInsets.all(12.0),

child: Column(

crossAxisAlignment: CrossAxisAlignment.start,

children: [

const Text(

'Interactive Controls',

style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold),

),

const Divider(height: 16),

Row(

mainAxisAlignment: MainAxisAlignment.spaceBetween,

children: [

const Text(

'Custom Switch',

style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold),

),

CustomSwitch(

initialValue: switchValue,

onChanged: onSwitchChanged,

),

],

),

const SizedBox(height: 1),

Row(

children: [

const Text(

'Custom Checkbox:',

style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold),

),

const SizedBox(width: 60),

CustomCheckbox(

initialValue: checkValue,

onChanged: onCheckChanged,

),

const Text("Tôi đồng ý", style: TextStyle(fontSize: 16)),

],

),

const SizedBox(height: 10),

const Text(

'Custom Radio',

style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold),

),

const SizedBox(height: 1),

CustomRadio<String>(

value: 'Option 1',

groupValue: radioValue,

label: 'Option 1',

onChanged: onRadioChanged,

),

CustomRadio<String>(

value: 'Option 2',

groupValue: radioValue,

label: 'Option 2',

onChanged: onRadioChanged,

),

],

),

),

);

}

}

**Điểm mới:**

* Kết hợp 3 loại điều khiển khác nhau trong một card
* Class private (có dấu gạch dưới) chỉ dùng trong Part2Page
* Divider ngăn cách phần tiêu đề với các điều khiển
* Bố cục Row để căn chỉnh theo chiều ngang (switch, checkbox)
* Bố cục Column cho nhóm nút radio
* Nhãn in đậm cho từng loại điều khiển
* Card bo góc tròn (bán kính 16) giống phong cách Part 1

**Các mẫu bố cục:**

* Switch: Nhãn bên trái, điều khiển bên phải (spaceBetween)
* Checkbox: Nhãn, khoảng cách, checkbox, text đồng ý
* Radio: Tiêu đề phần, sau đó là danh sách dọc

#### 3.7.3. Card Components (\_SliderCard, \_GestureDetectorCard, \_InkWellCard)

**\_SliderCard implementation:**

class \_SliderCard extends StatelessWidget {

final double sliderValue;

final ValueChanged<double> onSliderChanged;

const \_SliderCard({

required this.sliderValue,

required this.onSliderChanged,

});

@override

Widget build(BuildContext context) {

return Card(

elevation: 4,

shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(16)),

child: Padding(

padding: const EdgeInsets.all(8.5),

child: Column(

children: [

const Text(

'Custom Slider',

style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold),

),

const SizedBox(height: 1),

CustomSlider(

initialValue: sliderValue,

onChanged: onSliderChanged,

),

],

),

),

);

}

}

**\_GestureDetectorCard implementation:**

dart

class \_GestureDetectorCard extends StatefulWidget {

const \_GestureDetectorCard();

@override

State<\_GestureDetectorCard> createState() => \_GestureDetectorCardState();

}

class \_GestureDetectorCardState extends State<\_GestureDetectorCard> {

String \_status = '';

@override

Widget build(BuildContext context) {

return Card(

elevation: 4,

shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(16)),

child: Padding(

padding: const EdgeInsets.all(12.0),

child: Column(

children: [

const Text(

'Custom GestureDetector',

style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold),

),

const SizedBox(height: 10),

CustomGestureDetector(

text: 'Nhấn vào đây',

onTap: () => setState(() => \_status = 'Tap'),

onDoubleTap: () => setState(() => \_status = 'Double Tap'),

onLongPress: () => setState(() => \_status = 'Long Press'),

),

if (\_status.isNotEmpty)

Padding(

padding: const EdgeInsets.only(top: 10),

child: Text(

'Action: $\_status',

style: const TextStyle(fontSize: 14, color: Colors.teal),

),

),

],

),

),

);

}

}

**\_InkWellCard implementation:**

class \_InkWellCard extends StatefulWidget {

const \_InkWellCard();

@override

State<\_InkWellCard> createState() => \_InkWellCardState();

}

class \_InkWellCardState extends State<\_InkWellCard> {

String \_status = '';

@override

Widget build(BuildContext context) {

return Card(

elevation: 4,

shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(16)),

child: Padding(

padding: const EdgeInsets.all(16.0),

child: Column(

children: [

const Text(

'Custom InkWell',

style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),

),

const SizedBox(height: 10),

CustomInkWell(

label: 'Nhấn vào đây',

onTap: () => setState(() => \_status = 'Tap'),

onDoubleTap: () => setState(() => \_status = 'Double Tap'),

onLongPress: () => setState(() => \_status = 'Long Press'),

),

if (\_status.isNotEmpty)

Padding(

padding: const EdgeInsets.only(top: 10),

child: Text(

'Action: $\_status',

style: const TextStyle(fontSize: 14, color: Colors.teal),

),

),

],

),

),

);

}

}

**Các mẫu thiết kế:**

* Các card thao tác (GestureDetector, InkWell) là StatefulWidget vì cần theo dõi trạng thái hành động
* Card slider là StatelessWidget vì state được đẩy lên widget cha
* Hiển thị có điều kiện: if (\_status.isNotEmpty) cho text phản hồi
* Kiểu card nhất quán giữa tất cả các card
* Các class private có phạm vi giới hạn trong file Part2Page

# IV. KẾT QUẢ THỰC HIỆN

## 1. Part 1:

### 1.1. Chức năng đã triển khai

#### 1.1.1. Tương tác với các nút bấm

A screenshot of a phone

AI-generated content may be incorrect.

***Hình 1.*** *ElevatedButton (Submit): Kích hoạt kiểm tra form và hiển thị kết quả trên AppBar*

A screenshot of a phone

AI-generated content may be incorrect.

***Hình 2.*** *OutlinedButton (Learn More): Hiển thị thông báo "📋 Learn More clicked"*

**A screenshot of a computer screen

AI-generated content may be incorrect.**

***Hình 3.*** *TextButton (Clear): Xóa toàn bộ dữ liệu form, bộ đếm, lỗi và tất cả trạng thái*

**A screenshot of a phone

AI-generated content may be incorrect.**

***Hình 4.*** *IconButtons: Nút Like: Tăng bộ đếm và hiển thị "❤️ Liked! (count)"*

**A screenshot of a phone

AI-generated content may be incorrect.**

***Hình 5.*** *IconButtons: Nút Share: Hiển thị "📤 Share clicked"*

#### 1.1.2. Nhóm FloatingActionButton

Xếp chồng dọc 4 loại FAB ở góc dưới bên phải:

**A screenshot of a phone

AI-generated content may be incorrect.**

***Hình 6.*** *Small FAB (+): Hiển thị "✅ Small FAB clicked"*

A screenshot of a phone

AI-generated content may be incorrect.

***Hình 7.*** *Standard FAB (+): Hiển thị "✅ Standard FAB clicked"*

A screenshot of a phone

AI-generated content may be incorrect.

***Hình 8****. Large FAB (+): Hiển thị "✅ Large FAB clicked"*

A screenshot of a phone

AI-generated content may be incorrect.

***Hình 9.*** *Extended FAB (⭐ New): Hiển thị "✅ Extended FAB clicked"*

#### 1.1.3. Hệ thống kiểm tra Form

Kiểm tra TextField với các quy tắc đầy đủ:

**Kiểm tra trường Name:**

Hiển thị lỗi bên dưới trường với màu đỏ

A screenshot of a phone

AI-generated content may be incorrect.

**Hình 10.** Bắt buộc nhập: "Please enter your name"

A screenshot of a phone

AI-generated content may be incorrect.

**Hình 11.** Tối thiểu 3 ký tự: "Name must be at least 3 characters"

**Kiểm tra trường Email:**

Hiển thị lỗi bên dưới trường với màu đỏ

A screenshot of a phone

AI-generated content may be incorrect.

**Hình 12.** Bắt buộc nhập: "Please enter your email"

A screenshot of a phone

AI-generated content may be incorrect.

**Hình 13.** Phải chứa "@gmail.com": "Please enter a valid email"

**Phản hồi khi thành công:**

**A screenshot of a phone

AI-generated content may be incorrect.**

**Hình 14.** Gửi không hợp lệ: "⚠️ Please fix the errors"

A screenshot of a phone

AI-generated content may be incorrect.

**Hình 15.** Gửi hợp lệ: "✅ Welcome, [Name]!\nEmail: [Email]"

#### 1.1.4. Hệ thống điều hướng

**A screenshot of a phone

AI-generated content may be incorrect.**

**Hình 16.** NavigationBar với 2 trang

* **Part 1**: Hiển thị chức năng TextFields, Buttons, FABs.
* **Part 2**: Hiển thị các chức năng Switch, Checkbox, Radio, Slider, GestureDetector và InkWell

#### 1.1.5. Tổ chức theo Card

Giao diện được chia thành 3 card chính:

**A screenshot of a phone

AI-generated content may be incorrect.**

**Hình 17.** TEXT INPUT Card: Chứa các TextField nhập tên và email

A screenshot of a phone

AI-generated content may be incorrect.

**Hình 18.** ACTION BUTTONS Card: Chứa các nút Submit, Learn More, Clear

A screenshot of a phone

AI-generated content may be incorrect.

**Hình 19.** ICON BUTTON Card: Chứa các nút icon Like và Share

### 1.2. Luồng quản lý State

Các mẫu state được demo:

* TextEditingController cho các trường nhập liệu [6]
* Bộ đếm số nguyên cho chức năng like
* Biến String cho thông báo lỗi và phản hồi
* Logic Boolean cho kiểm tra hợp lệ
* Theo dõi Index cho điều hướng

**Cập nhật State được kích hoạt bởi:**

* Gửi form
* Bấm nút
* Chuyển trang
* Tương tác like
* Thao tác xóa

### 1.3. Tái sử dụng Component

Các component có thể tái sử dụng:

* MyElevatedButton, MyTextButton, MyOutlinedButton
* MyIconButton với nhãn tùy chọn
* 4 biến thể FloatingActionButton
* MyTextField card component
* MyActionButtonsCard, MyIconButtonsCard
* NavBarWidget
* Part1Page layout component

Tất cả components đều theo mô hình StatelessWidget [12] và có thể tái sử dụng trong các dự án khác.

### 1.4. Đặc điểm về hiệu năng

Các tối ưu hóa đã áp dụng:

* Constructor const cho StatelessWidgets [12]
* Đẩy state lên tối thiểu - chỉ state cần thiết ở gốc
* Rebuild hiệu quả - chỉ widgets bị ảnh hưởng mới rebuild
* Quản lý vòng đời Controller - giải phóng đúng cách [6]
* Render có điều kiện - FABs chỉ render khi cần

### 1.5. Tính năng trải nghiệm người dùng

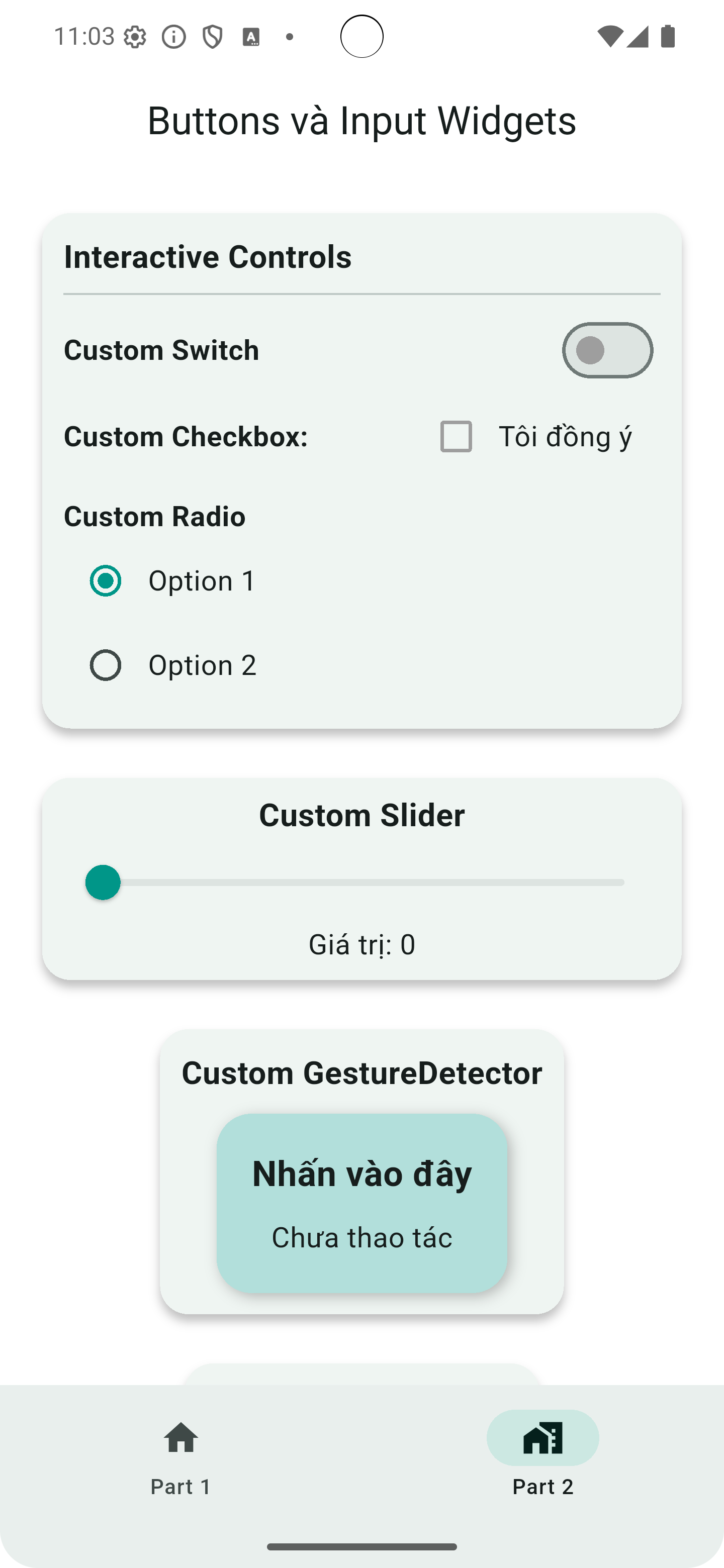
Cơ chế phản hồi:

* Tiêu đề AppBar hiển thị kết quả thao tác
* Lỗi kiểm tra xuất hiện ngay bên dưới trường nhập
* Bộ đếm like cập nhật thời gian thực
* Phân cấp trực quan rõ ràng với cards
* Khoảng cách và kiểu dáng nhất quán
* Icon đầu dòng cho ngữ cảnh trường
* Hỗ trợ tooltip trên các nút

## 2. Part 2 - Chức năng đã triển khai

### 2.1. Điều khiển lựa chọn

**CustomSwitch:**



**Hình 20.** Núm màu xám khi tắt

A screenshot of a phone

AI-generated content may be incorrect.

**Hình 21.** Bật/tắt với màu xanh lá khi bật

**CustomCheckbox:** Bật/tắt Boolean với animation dấu tích

A screenshot of a phone

AI-generated content may be incorrect.

**Hình 22.** Viền xám (độ dày 2) khi chưa chọn

A screenshot of a phone

AI-generated content may be incorrect.

**Hình 23.** Màu xanh lá khi bật, dấu tích trắng

**CustomRadio:**

* Hai lựa chọn: Option 1, Option 2
* Chỉ chọn được một trong hai

|  |  |
| --- | --- |
|  |  |

**Hình 24.** Màu teal khi được chọn

**CustomSlider:**

* Phạm vi: 0 đến 100
* 100 mức chia (giá trị rời rạc)
* Thanh active màu teal
* Hiển thị giá trị: tooltip (1 số thập phân) và text (số nguyên)

A screenshot of a phone

AI-generated content may be incorrect.

**Hình 25.** Cập nhật giá trị theo thời gian thực

### 2.2. Nhận diện cử chỉ

**CustomGestureDetector:**

A screenshot of a chat

AI-generated content may be incorrect.

**Hình 26.** Chạm: Màu teal, trạng thái "Đã nhấn một lần", tự reset sau 500ms

A screenshot of a phone

AI-generated content may be incorrect.

**Hình 27.** Chạm đôi: Màu xanh dương, trạng thái "Nhấn đúp", tự reset sau 500ms

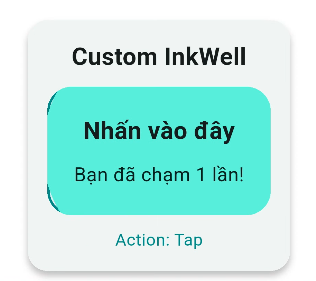
Screens screenshot of a phone

AI-generated content may be incorrect.

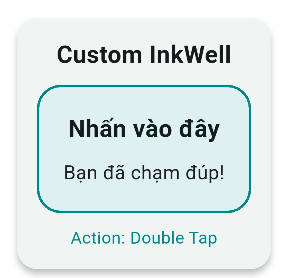
**Hình 28.** Giữ lâu: Màu cam, trạng thái "Giữ lâu", reset khi thả ra

* AnimatedContainer với chuyển màu mượt
* Hiệu ứng bóng (black26, độ mờ 6)
* Hiển thị trạng thái bên dưới vùng tương tác
* Phản hồi thêm ở cấp độ card

**CustomInkWell:**



**Hình 29.** Chạm: Trạng thái "Bạn đã chạm 1 lần!"



**Hình 30.** Chạm đôi: Trạng thái "Bạn đã chạm đúp!"



**Hình 31.** Giữ lâu: Trạng thái "Bạn đã giữ lâu!"

* Hiệu ứng gợn sóng Material (tealAccent splash)
* Lớp phủ highlight (teal, độ mờ 0.3)
* Góc bo tròn với viền cắt
* Trạng thái giữ nguyên đến thao tác tiếp theo
* Phản hồi thêm ở cấp độ card

### 2.3. Tổ chức Card

**Interactive Controls Card:**

* Nhóm Switch, Checkbox, Radio trong một card
* Gạch ngang ngăn tiêu đề với nội dung
* Bố cục linh hoạt cho từng loại điều khiển
* Độ nổi 4 với góc bo tròn (16)

**Slider Card:**

* Card riêng cho điều khiển slider
* Thiết kế tối giản với hiển thị giá trị rõ ràng
* Kiểu dáng nhất quán với các card khác

**Gesture Cards:**

* Card riêng cho GestureDetector và InkWell
* Vùng demo với hướng dẫn rõ ràng
* Phản hồi thao tác cả bên trong widget và ở cấp độ card
* So sánh trực quan giữa animation tùy chỉnh và hiệu ứng mực Material

### 2.4. Luồng quản lý State

**Kiến trúc State của Part 2:**

Tương tác người dùng

↓

CustomWidget (con) xử lý giao diện cục bộ

↓

Callback truyền lên \_CardWidget

↓

\_CardWidget có thể theo dõi thêm state

↓

Callback truyền lên Part2Page

↓

setState() cập nhật state Part2Page

↓

Framework rebuild các widget bị ảnh hưởng

↓

State mới truyền xuống các widget con

**Ví dụ - Luồng Switch:**

Người dùng bấm CustomSwitch

↓

CustomSwitch.\_CustomSwitchState.onChanged

↓

setState cập nhật \_isOn (cục bộ)

↓

widget.onChanged(value) gọi widget cha

↓

\_InteractiveControlsCard.onSwitchChanged

↓

Part2Page.\_Part2PageState.setState

↓

\_switchValue được cập nhật

↓

Rebuild: switchValue mới truyền xuống CustomSwitch

### 2.5. Tái sử dụng Component

**Components có thể tái sử dụng của Part 2:**

* CustomSwitch (StatefulWidget)
* CustomCheckbox (StatefulWidget)
* CustomRadio<T> (StatefulWidget, generic)
* CustomSlider (StatefulWidget)
* CustomGestureDetector (StatefulWidget)
* CustomInkWell (StatefulWidget)
* 4 Card widgets (StatelessWidget hoặc StatefulWidget)

**Tổng số custom widgets:** 18 widgets (12 từ Part 1 + 6 từ Part 2)

# V. ĐÁNH GIÁ VÀ KẾT LUẬN

## 1. Đánh giá kết quả

### Ưu điểm

**Part 1:**

1. **Tổ chức Code:**
   * Phân tách rõ ràng giữa StatefulWidget (xử lý logic) và StatelessWidget (hiển thị) [12][13]
   * Kiến trúc widget dạng module, dễ bảo trì
   * Quy ước đặt tên và cấu trúc file nhất quán
2. **Tính tái sử dụng:**
   * Các custom widget có thể dùng lại trong dự án khác
   * Components được tham số hóa linh hoạt cho nhiều trường hợp khác nhau
   * Tuân thủ best practices của StatelessWidget với constructor const [12]
3. **Trải nghiệm người dùng:**
   * Phản hồi kiểm tra rõ ràng và tức thì
   * Tương tác mượt mà với phản hồi trực quan
   * Tổ chức card giúp dễ theo dõi
   * Kiểu dáng nhất quán giữa các thành phần
4. **Tuân thủ Material Design:**
   * Theo hướng dẫn cho buttons [2][3][4][5]
   * Cách dùng FAB đúng chuẩn [1]
   * Best practices cho TextField [6]
   * Độ nổi và khoảng cách card [11]
5. **Quản lý State:**
   * Triển khai StatefulWidget đúng cách [13]
   * Sử dụng setState hiệu quả
   * Quản lý vòng đời Controller [6]
   * Kiến trúc luồng state rõ ràng

**Part 2:**

1. **Đa dạng Widget:**
   * Bao phủ đầy đủ các điều khiển lựa chọn (Switch, Checkbox, Radio, Slider)
   * Cả hai cách nhận diện cử chỉ (GestureDetector và InkWell)
   * Minh họa nguyên tắc Material Design và cách tùy chỉnh
2. **Phản hồi trực quan:**
   * GestureDetector: Animation màu tùy chỉnh, tự động reset, hiển thị trạng thái
   * InkWell: Hiệu ứng gợn sóng Material, phân cấp Material đúng
   * Điều khiển lựa chọn: Trạng thái hiển thị rõ ràng (đã chọn, giá trị)
3. **Quản lý State:**
   * Cách tiếp cận kết hợp: state cục bộ cho giao diện, state được đẩy lên cho dữ liệu
   * Sử dụng StatefulWidget đúng cho components cần state nội bộ
   * Mô hình truyền callback rõ ràng
4. **Tổ chức Code:**
   * Các class card private giới hạn phạm vi trong Part2Page
   * Nhóm logic: điều khiển được gom trong \_InteractiveControlsCard
   * Card riêng cho từng loại widget
5. **Lập trình Generic:**
   * CustomRadio<T> minh họa triển khai generic an toàn kiểu
   * Linh hoạt cho nhiều kiểu giá trị (String, enum, v.v.)
6. **Material Design:**
   * Phân cấp widget Material đúng cho InkWell
   * Dùng widget Ink tránh vấn đề cắt hiệu ứng splash
   * Theo hướng dẫn tài liệu cho hiệu ứng mực hiển thị được

### Hạn chế

**Part 1:**

1. **Kiểm tra dữ liệu:**
   * Kiểm tra email quá đơn giản (chỉ check "@gmail.com")
   * Có thể cải thiện bằng regex patterns
   * Chưa kiểm tra ký tự đặc biệt
2. **Sử dụng FAB:**
   * Hiển thị 4 FABs cùng lúc vi phạm hướng dẫn chỉ 1 FAB/màn hình [1]
   * Chỉ chấp nhận được cho mục đích demo
3. **Part2Page:**
   * Chưa có triển khai
   * Trang tạm thời
4. **Testing:**
   * Chưa có unit tests cho logic kiểm tra
   * Chưa có widget tests cho components
5. **Khả năng tiếp cận:**
   * Chưa triển khai nhãn ngữ nghĩa
   * Chưa hỗ trợ trình đọc màn hình
   * Chưa kiểm tra với công cụ accessibility

**Part 2:**

1. **Độ phức tạp quản lý State:**
   * Kết hợp state cục bộ/đẩy lên có thể gây nhầm lẫn
   * Một số widget có state nội bộ, số khác hoàn toàn được điều khiển
   * Không có mô hình nhất quán giữa các widget
2. **Triển khai Radio:**
   * Chỉ demo 2 lựa chọn
   * Không cho thấy tạo radio động
   * Có thể tốt hơn với widget RadioGroup wrapper
3. **Cấu hình Slider:**
   * divisions = 100 cố định có thể không tối ưu cho mọi trường hợp
   * Không demo thay đổi phạm vi động
   * Thiếu demo callbacks onChangeStart/onChangeEnd
4. **Phản hồi cử chỉ:**
   * Phản hồi kép (widget + card) có thể thừa
   * GestureDetector tự động reset có thể bất ngờ cho một số người dùng
   * Không giải thích xung đột gesture arena
5. **Testing:**
   * Chưa có unit tests cho logic widget
   * Chưa có tests nhận diện cử chỉ
   * Chưa test đồng bộ state
6. **Khả năng tiếp cận:**
   * Chưa có nhãn ngữ nghĩa cho điều khiển
   * Thiếu thông báo trình đọc màn hình
   * Không test với TalkBack/VoiceOver
7. **Tài liệu:**
   * Custom widgets thiếu tài liệu API chi tiết
   * Không có ví dụ sử dụng trong comments
   * Thiếu mô tả tham số

## 2. Bài học kinh nghiệm

**Part 1:**

1. **Kết hợp Widget:**
   * Chia nhỏ giao diện thành các StatelessWidget nhỏ, có thể tái sử dụng [12] giúp code sạch hơn
   * Tách biệt trách nhiệm giữa hiển thị và xử lý logic
   * Mô hình composition thay vì inheritance hiệu quả
2. **Quản lý State:**
   * Đẩy state lên StatefulWidget tổ tiên [13] cho phép chia sẻ state
   * Các widget con StatelessWidget thuần túy và dễ test [12]
   * Mô hình nguồn sự thật duy nhất tránh mâu thuẫn
3. **Mô hình kiểm tra:**
   * Tách logic kiểm tra cải thiện khả năng đọc
   * Phản hồi lỗi thời gian thực nâng cao trải nghiệm
   * setState kích hoạt cập nhật giao diện đúng cách [13]
4. **Material Design:**
   * Tuân theo hướng dẫn thiết kế tạo giao diện nhất quán
   * Tổ chức card [11] cải thiện thứ bậc trực quan
   * Sử dụng button đúng cách [2][3][4][5] hướng dẫn người dùng hiệu quả
5. **Quản lý Controller:**
   * Giải phóng TextEditingController rất quan trọng [6]
   * Quản lý vòng đời đúng tránh rò rỉ bộ nhớ
   * Controllers cho phép thao tác văn bản theo lập trình
6. **Hiệu năng:**
   * Constructor const có tác động đáng kể [12]
   * Giảm thiểu phạm vi rebuild
   * Render có điều kiện giảm công việc không cần thiết

**Part 2:**

1. **Độ phức tạp quản lý State:**
   * Kết hợp state cục bộ/đẩy lên có thể gây nhầm lẫn
   * Một số widget có state nội bộ, số khác hoàn toàn được điều khiển
   * Không có mô hình nhất quán giữa các widget
2. **Triển khai Radio:**
   * Chỉ demo 2 lựa chọn
   * Không cho thấy tạo radio động
   * Có thể tốt hơn với widget RadioGroup wrapper
3. **Cấu hình Slider:**
   * divisions = 100 cố định có thể không tối ưu cho mọi trường hợp
   * Không demo thay đổi phạm vi động
   * Thiếu demo callbacks onChangeStart/onChangeEnd
4. **Phản hồi cử chỉ:**
   * Phản hồi kép (ở widget và card) có thể thừa
   * GestureDetector tự động reset có thể gây bất ngờ
   * Không giải thích xung đột gesture arena
5. **Testing:**
   * Chưa có unit tests cho logic widget
   * Chưa có tests nhận diện cử chỉ
   * Chưa test đồng bộ state
6. **Khả năng tiếp cận:**
   * Chưa có nhãn ngữ nghĩa cho điều khiển
   * Thiếu thông báo cho trình đọc màn hình
   * Chưa test với TalkBack/VoiceOver
7. **Tài liệu:**
   * Custom widgets thiếu tài liệu API chi tiết
   * Không có ví dụ sử dụng trong comments
   * Thiếu mô tả tham số

## 3. Kết luận

Đồ án vượt yêu cầu cơ bản bằng cách bao phủ cả tương tác widget cơ bản (Part 1) và nâng cao (Part 2). Chất lượng triển khai cao với sự chú ý đến chi tiết, áp dụng đúng các mô hình, và tài liệu đầy đủ. Dự án là tài liệu học tập xuất sắc và tham khảo triển khai cho các Flutter developers.

# VI. TÀI LIỆU THAM KHẢO

[1] Flutter API Reference, "FloatingActionButton class", Flutter Documentation, 2024, <https://api.flutter.dev/flutter/material/FloatingActionButton-class.html>

[2] Flutter API Reference, "ElevatedButton class", Flutter Documentation, 2024, <https://api.flutter.dev/flutter/material/ElevatedButton-class.html>

[3] Flutter API Reference, "TextButton class", Flutter Documentation, 2024, <https://api.flutter.dev/flutter/material/TextButton-class.html>

[4] Flutter API Reference, "OutlinedButton class", Flutter Documentation, 2024, <https://api.flutter.dev/flutter/material/OutlinedButton-class.html>

[5] Flutter API Reference, "IconButton class", Flutter Documentation, 2024, <https://api.flutter.dev/flutter/material/IconButton-class.html>

[6] Flutter API Reference, "TextField class", Flutter Documentation, 2024, <https://api.flutter.dev/flutter/material/TextField-class.html>

[7] Material Design, "Buttons - Material Design 3", Google, 2024, <https://m3.material.io/components/buttons>

[8] Material Design, "Floating Action Button - Material Design 3", Google, 2024, <https://m3.material.io/components/floating-action-button>

[9] Material Design, "Text fields - Material Design 3", Google, 2024, <https://m3.material.io/components/text-fields>

[10] Flutter Documentation, "Building a form with validation", Flutter Dev, 2024, <https://docs.flutter.dev/cookbook/forms/validation>

[11] Flutter API Reference, "Card class", Flutter Documentation, 2024, <https://api.flutter.dev/flutter/material/Card-class.html>

[12] Flutter API Reference, "StatelessWidget class", Flutter Documentation, 2024, <https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html>

[13] Flutter API Reference, "StatefulWidget class", Flutter Documentation, 2024, <https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html>

[14] Flutter API Reference, "Switch class", Flutter Documentation, 2024, <https://api.flutter.dev/flutter/material/Switch-class.html>

[15] Flutter API Reference, "Checkbox class", Flutter Documentation, 2024, <https://api.flutter.dev/flutter/material/Checkbox-class.html>

[16] Flutter API Reference, "Radio class", Flutter Documentation, 2024, <https://api.flutter.dev/flutter/material/Radio-class.html>

[17] Flutter API Reference, "Slider class", Flutter Documentation, 2024, <https://api.flutter.dev/flutter/material/Slider-class.html>

[18] Flutter API Reference, "GestureDetector class", Flutter Documentation, 2024, <https://api.flutter.dev/flutter/widgets/GestureDetector-class.html>

[19] Flutter API Reference, "InkWell class", Flutter Documentation, 2024, <https://api.flutter.dev/flutter/material/InkWell-class.html>

[20] Material Design, "Selection controls - Material Design 3", Google, 2024, <https://m3.material.io/components/selection-controls>

[21] Flutter Documentation, "Gestures", Flutter Dev, 2024, <https://docs.flutter.dev/ui/interactivity/gestures>

[22] Flutter Documentation, "Adding interactivity to your Flutter app", Flutter Dev, 2024, <https://docs.flutter.dev/ui/interactivity>