

# Flutter: Quản lý trạng thái

Bùi Võ Quốc Bảo

Khoa CNTT | Trường CNTT-TT | Đại học Cần Thơ



# Tài liệu tham khảo

- Chương 6-13, **Flutter Apprentice** by Vincenzo Guzzi, Kevin D Moore, Vincent Ngo and Michael Katz
- <https://suragch.medium.com/flutter-state-management-for-minimalists-4c71a2f2f0c1>
- <https://docs.flutter.dev/development/data-and-backend/state-mgmt/declarative>
- <https://docs.flutter.dev/cookbook/forms>



# Xây dựng UI dạng khai báo



- Flutter cho phép nhà phát triển tạo UI cho ứng dụng *dạng khai báo (declarative)*: Flutter xây dựng UI để phản chiếu trạng thái của ứng dụng

$$\text{UI} = f(\text{state})$$

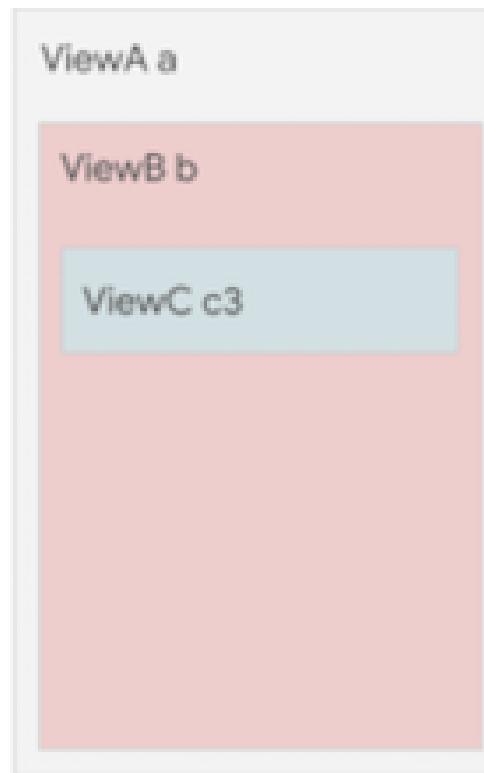
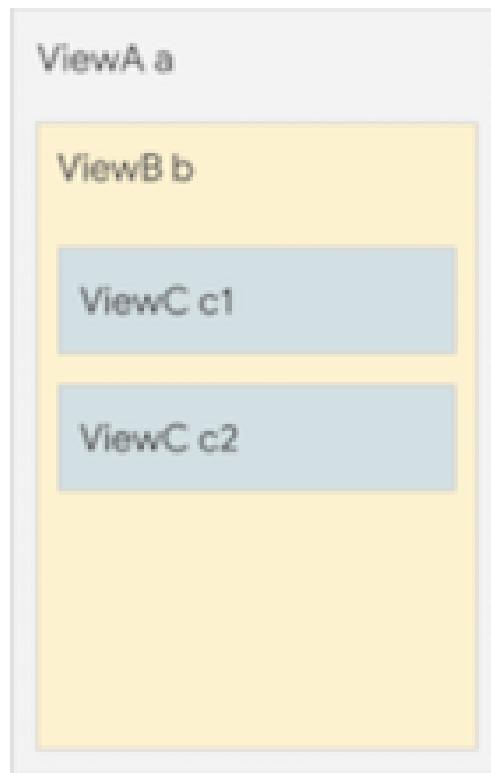
The layout  
on the screen

Your  
build  
methods

The application state

- Trạng thái ứng dụng thay đổi kích hoạt việc vẽ lại/tái tạo lại UI

# Xây dựng UI dạng khai báo



```
// Imperative style  
b.setColor(red)  
b.clearChildren()  
ViewC c3 = new ViewC(...)  
b.add(c3)
```

```
// Declarative style  
return ViewB(  
    color: red,  
    child: ViewC(...),  
)
```

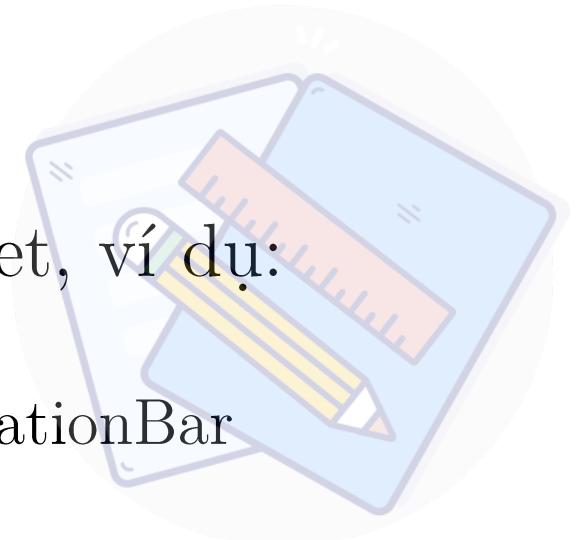
# Trạng thái ứng dụng

- Theo nghĩa rộng: trạng thái ứng dụng là mọi thứ tồn tại trong bộ nhớ khi ứng dụng chạy
- Định nghĩa hữu ích hơn cho nhà phát triển: *bất kỳ dữ liệu nào cần thiết để xây dựng lại UI*
- Có thể được chia thành hai loại:
  - Trạng thái cục bộ (local state/UI state/ephemeral state)
  - Trạng thái chia sẻ (shared state/app state)



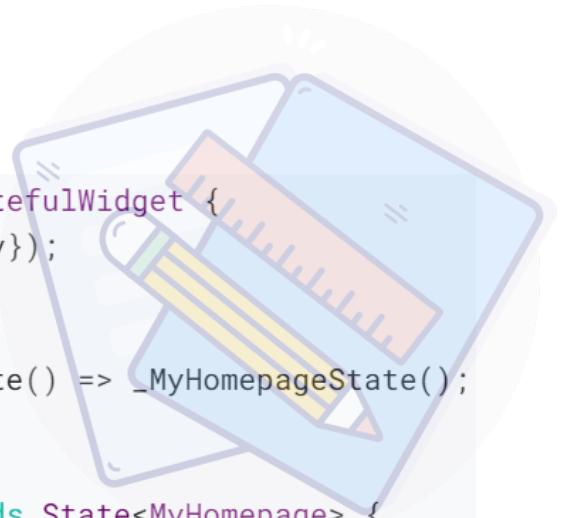
# Trạng thái cục bộ

- Trạng thái có thể được chứa gọn trong một widget, ví dụ:
  - Tiết trình hiện tại của một hoạt ảnh phức tạp
  - Thẻ đang được chọn hiện tại trong một BottomNavigationBar
  - Trang hiện thời trong một PageView, ...
- Các phần khác trong cây widget ít khi cần đến trạng thái này
- Chỉ cần sử dụng một **StatefulWidget** để quản lý



# Trạng thái cục bộ

- Trong ví dụ này, `_index` là trạng thái cục bộ
- Các phần khác trong ứng dụng không truy cập `_index`
- Giá trị `_index` chỉ thay đổi bên trong `MyHomePage`
- Không quan tâm `_index = 0` khi ứng dụng được đóng và khởi động lại



```
class MyHomepage extends StatefulWidget {
  const MyHomepage({super.key});

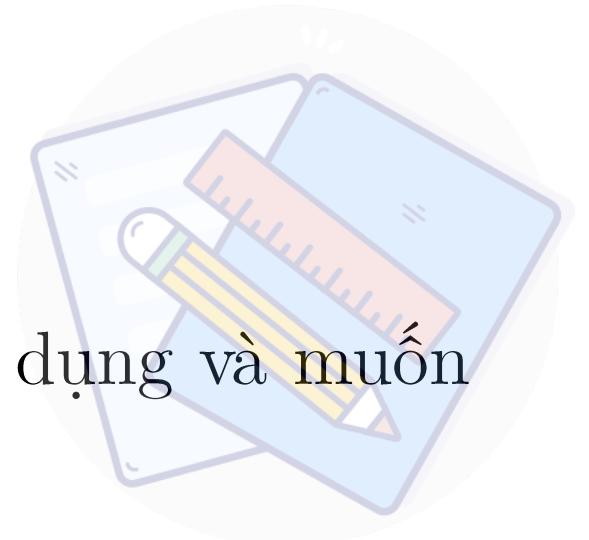
  @override
  _MyHomepageState createState() => _MyHomepageState();
}

class _MyHomepageState extends State<MyHomepage> {
  int _index = 0;

  @override
  Widget build(BuildContext context) {
    return BottomNavigationBar(
      currentIndex: _index,
      onTap: (newIndex) {
        setState(() {
          _index = newIndex;
        });
      },
      // ... items ...
    );
  }
}
```

# Trạng thái chia sẻ

- Còn gọi là trạng thái ứng dụng (app state)
- Trạng thái được sử dụng tại nhiều nơi trong ứng dụng và muôn được lưu giữ giữa các phiên làm việc
  - Thông tin thiết lập của người dùng (user preferences)
  - Thông tin đăng nhập
  - Trạng thái đọc/chưa đọc của các bài đăng trong một ứng dụng tin tức
  - Các thông báo trong một ứng dụng mạng xã hội
  - Giỏ hàng mua sắm trong một ứng dụng thương mại điện tử, ...



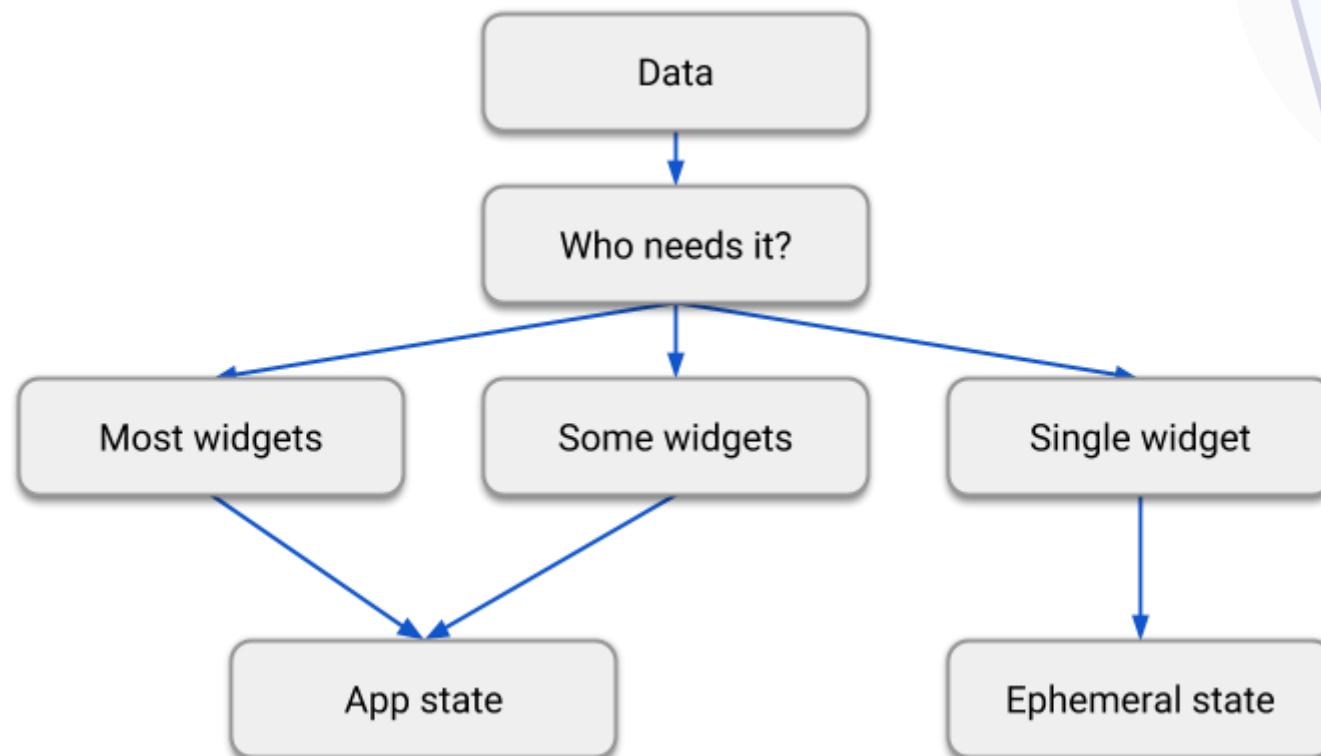
# Trạng thái chia sẻ

- Có nhiều giải pháp quản lý tùy thuộc ứng dụng
  - Trong trường hợp đơn giản, widget cha có thể gửi trạng thái, callback cập nhật trạng thái (kết hợp với setState) xuống cho widget con
  - Lựa chọn giải pháp phù hợp và giúp cho mã nguồn đơn giản, dễ hiểu



# Trạng thái cục bộ vs. Trạng thái chia sẻ

Lưu ý: Không có ranh giới rõ ràng giữa hai loại trạng thái!

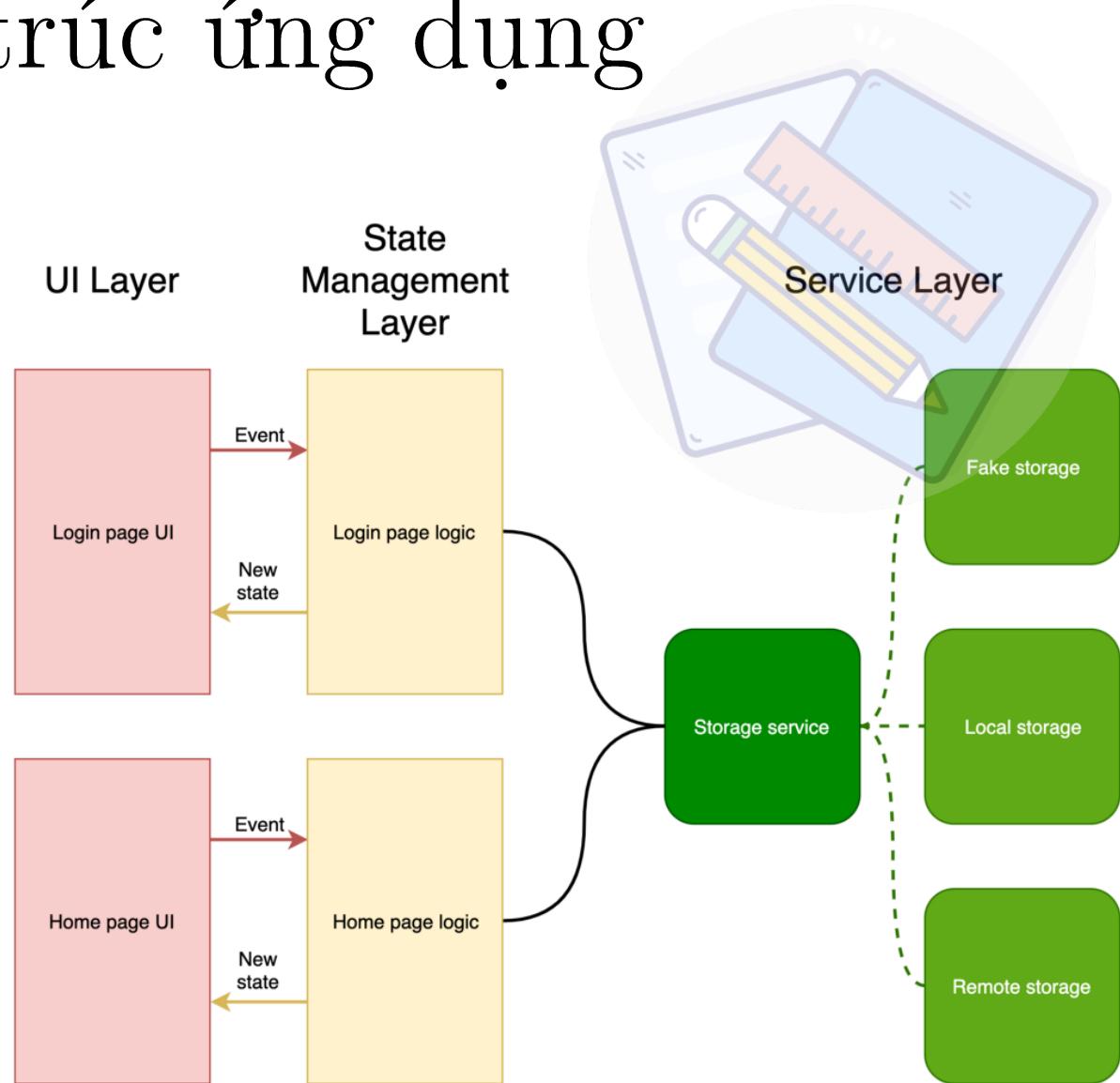




# Quản lý trạng thái đơn giản

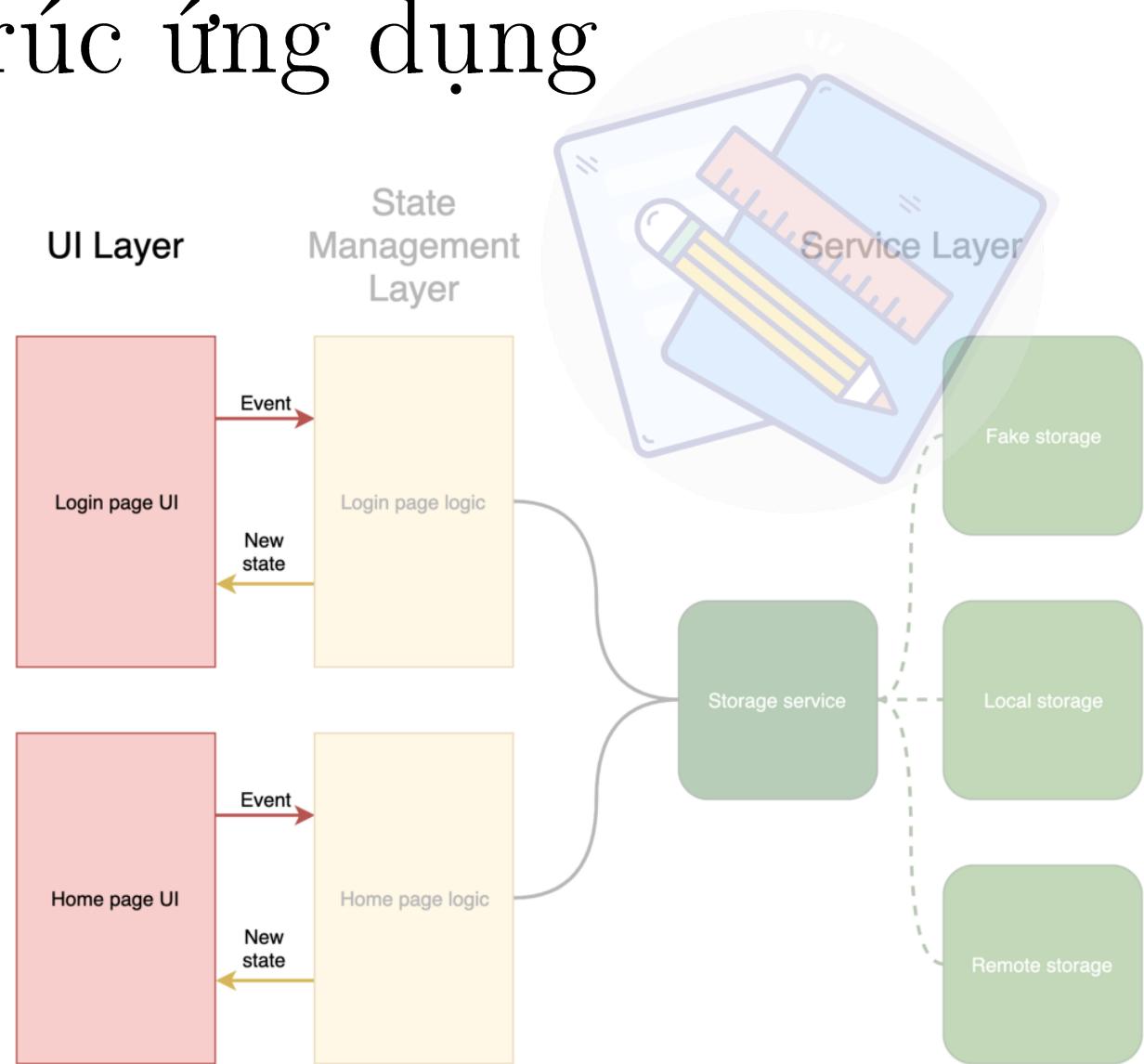
# Tổng quan về kiến trúc ứng dụng

- Kiến trúc ứng dụng Flutter có thể được tổ chức thành 3 layer: layer UI, layer quản lý trạng thái, và layer dịch vụ
  - Mỗi khối màu đại diện cho một lớp (class), một tập tin hoặc thư mục



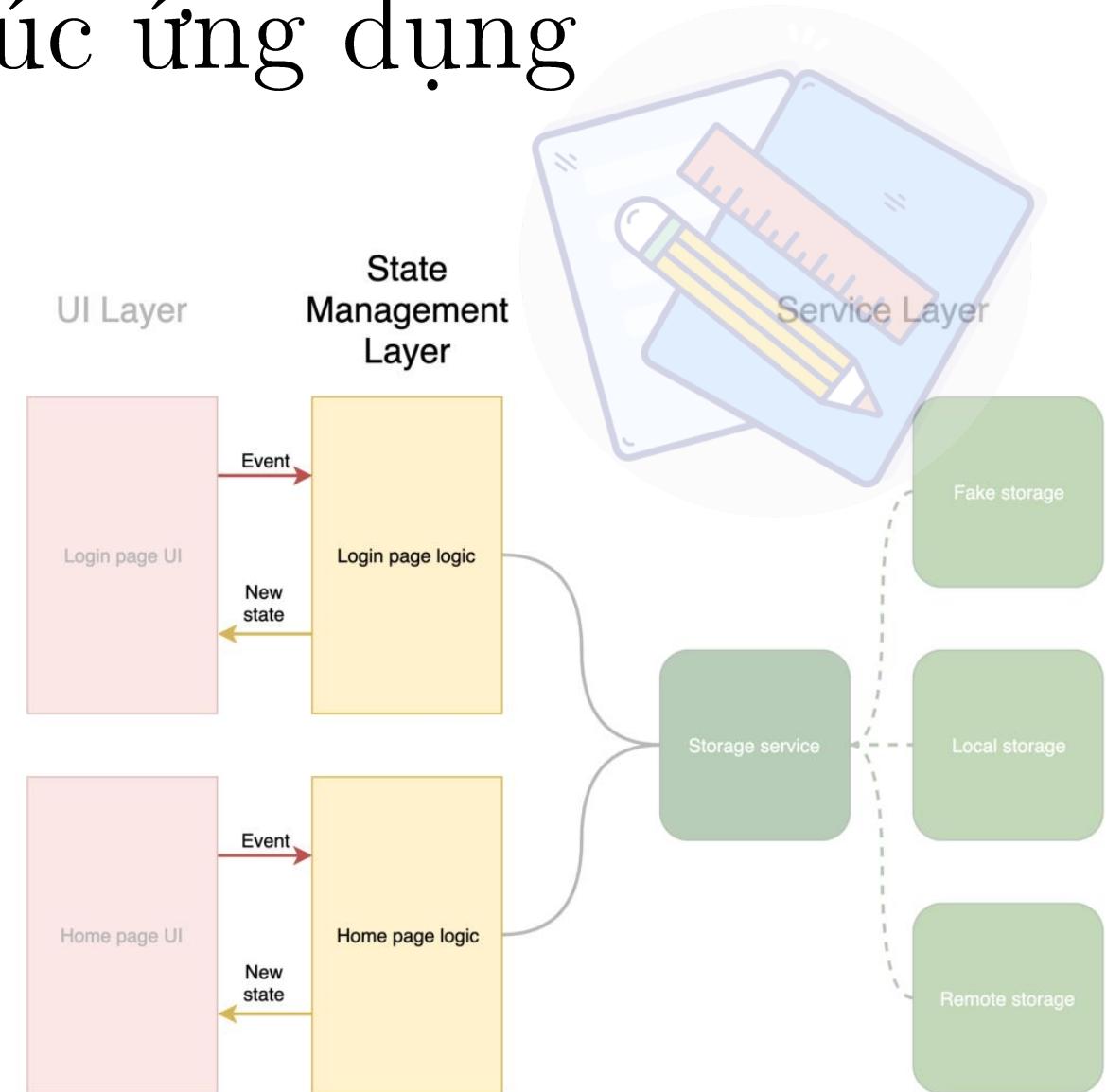
# Tổng quan về kiến trúc ứng dụng

- Layer UI: gồm các widget hợp thành bộ cục của ứng dụng
  - **Nhiệm vụ duy nhất** của layer UI là hiển thị trạng thái ứng dụng cho người dùng
  - Cần lắng nghe các thay đổi của trạng thái ứng dụng
  - Widget lắng nghe trạng thái và xây dựng lại UI khi có thay đổi thường được gọi là các widget builder



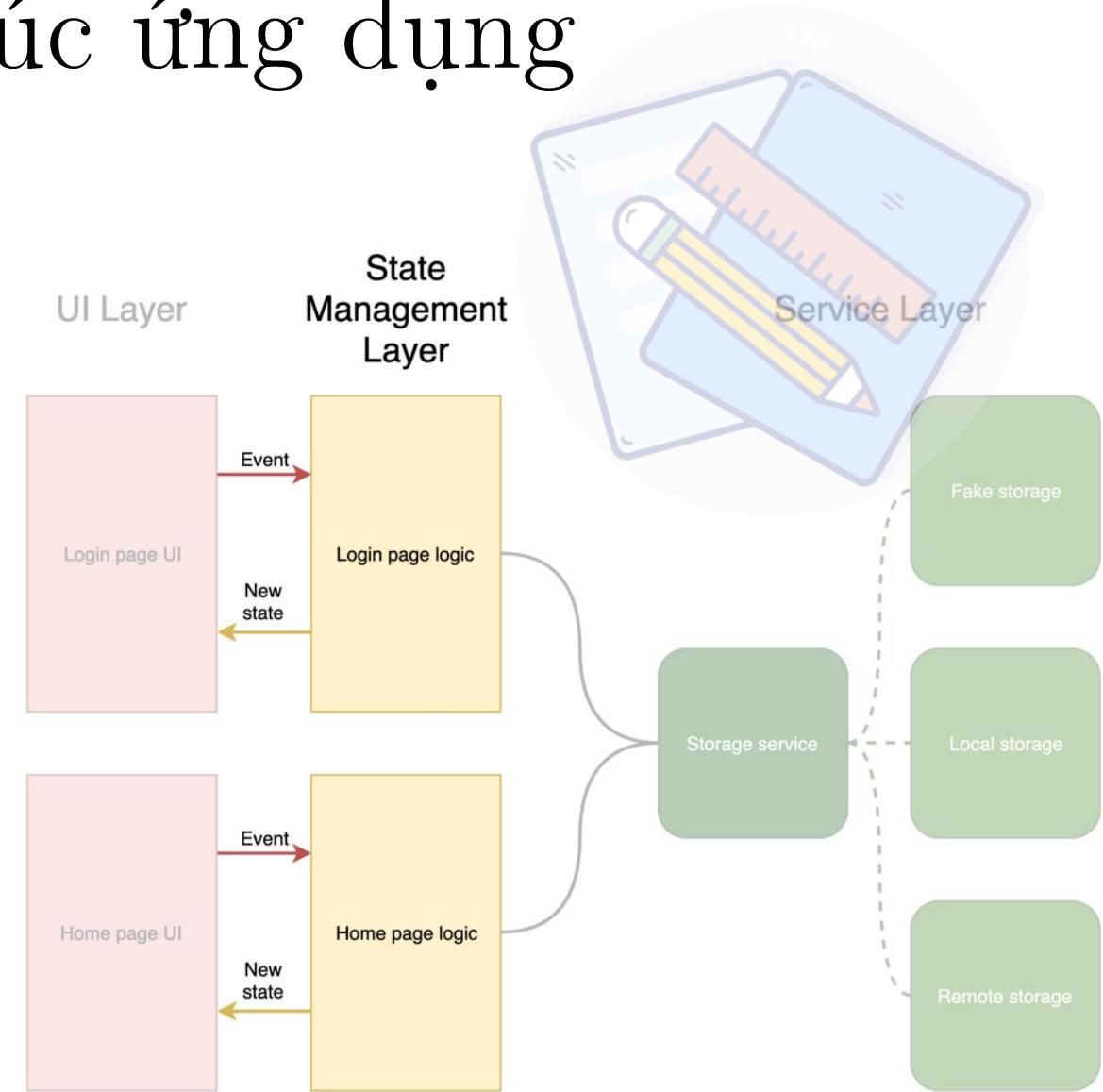
# Tổng quan về kiến trúc ứng dụng

- Layer quản lý trạng thái: lưu giữ trạng thái và các xử lý cập nhật trạng thái ứng dụng dựa trên các sự kiện
  - Nhiều tên gọi khác nhau: Bloc, Cubit, ViewModel, Model, Controller, Manager, ...



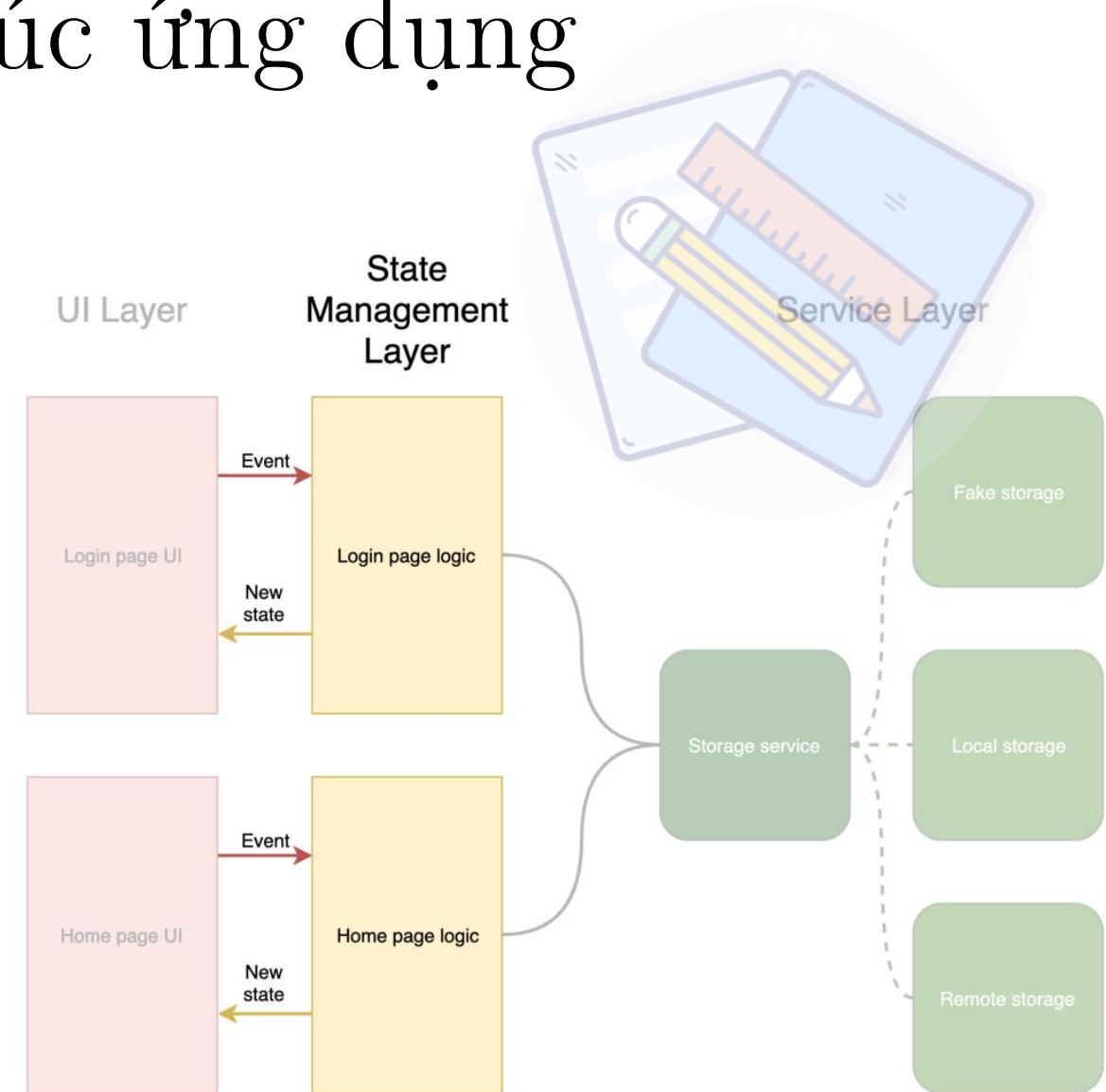
# Tổng quan về kiến trúc ứng dụng

- Layer quản lý trạng thái: lưu giữ trạng thái và các xử lý cập nhật trạng thái ứng dụng dựa trên các sự kiện
  - Sự kiện (event): một hành động của người dùng trên ứng dụng, đọc dữ liệu từ xa thành công, timer bị đáo hạn, ...
  - Đối với các sự kiện UI, layer UI báo cho layer quản lý trạng thái sự kiện xảy ra (ví dụ như gọi phương thức trên đối tượng quản lý trạng thái)



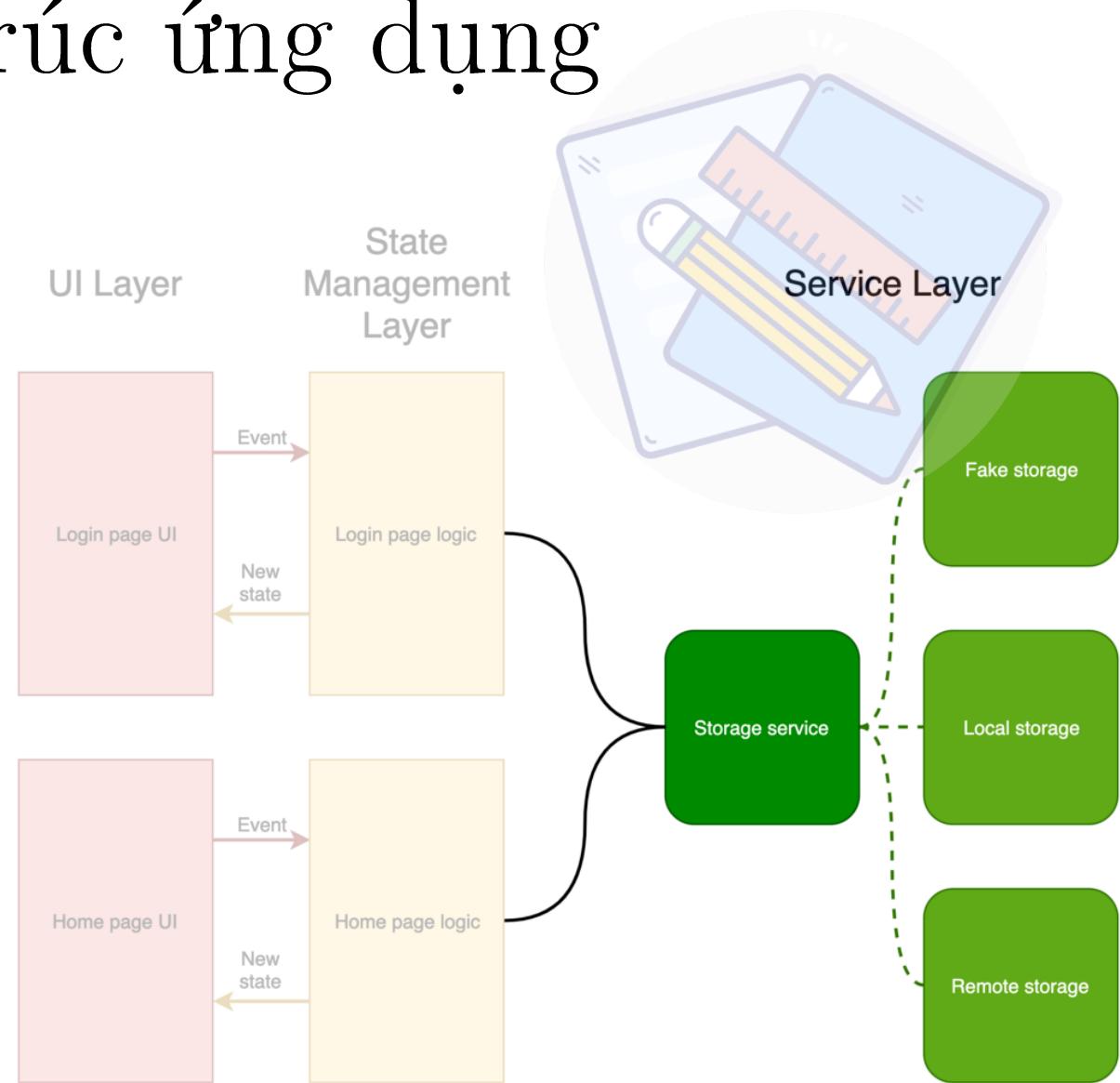
# Tổng quan về kiến trúc ứng dụng

- Layer quản lý trạng thái: lưu giữ trạng thái và các xử lý cập nhật trạng thái ứng dụng dựa trên các sự kiện
  - Cung cấp cơ chế để cho layer UI có thể lắng nghe các thay đổi của trạng thái ứng dụng
  - Layer quản lý trạng thái không nên biết gì về layer UI



# Tổng quan về kiến trúc ứng dụng

- Layer dịch vụ: còn gọi là kho dữ liệu (data repository), cung cấp dữ liệu cho nhiều nơi trong ứng dụng
  - Thực hiện những xử lý như truy xuất cơ sở dữ liệu, gọi API đến các máy chủ từ xa
  - Layer quản lý trạng thái tương tác với layer dịch vụ thông qua giao diện (interface) dịch vụ



# Quản lý trạng thái

- Một giải pháp quản lý trạng thái cần:
  1. Cung cấp cho layer UI một tham chiếu đến layer trạng thái
  2. Báo hiệu cho layer trạng thái biết về các sự kiện xảy ra
  3. Cung cấp cho layer UI cách lắng nghe thay đổi về trạng thái
  4. Tái tạo lại UI sau khi trạng thái thay đổi



# Quản lý trạng thái đơn giản

1. Cung cấp cho layer UI một tham chiếu đến layer trạng thái
  - Sử dụng [Provider](#) (thư viện wrapper của InheritedWidget)
  - Hoặc có thể sử dụng một [Service Locator](#) như [GetIt](#). GetIt cho phép đăng ký các đối tượng và truy xuất chúng về sau

```
final getIt = GetIt.instance;

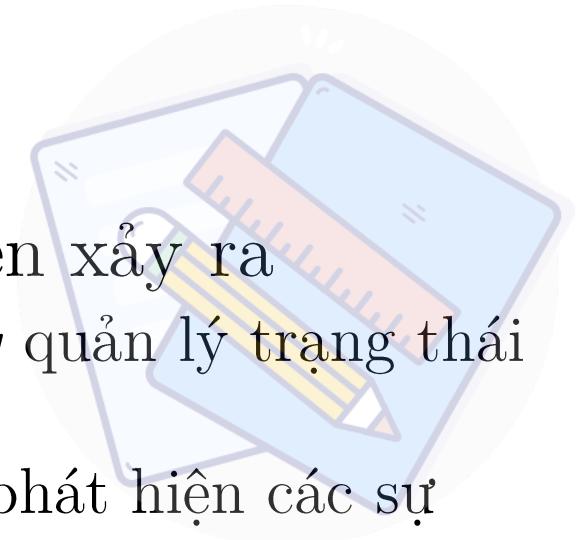
void setup() {
    // Đăng ký các đối tượng
    getIt.registerLazySingleton<HomePageManager>(() => HomePageManager());
    getIt.registerLazySingleton<StorageService>(() => SharedPreferencesStorage());
}

// Truy xuất đối tượng đã đăng ký trước đó
final HomePageManager = getIt<HomePageManager>();
```



# Quản lý trạng thái đơn giản

2. Báo hiệu cho layer trạng thái biết về các sự kiện xảy ra
  - Có thể chỉ đơn giản là *gọi phương thức trên đối tượng* quản lý trạng thái khi phát hiện các sự kiện
    - + Dùng các widget hỗ trợ tương tác người dùng để phát hiện các sự kiện trên UI
    - + Ví dụ, khi người dùng nhấn nút đăng nhập:  
`loginPageManager.submitUserInfo(username, password);`
  - Một số giải pháp quản lý trạng thái như flutter bloc dùng stream để báo hiệu các sự kiện trong ứng dụng



# Quản lý trạng thái đơn giản



3. Cung cấp cho layer UI cách lắng nghe thay đổi về trạng thái
  - Có thể sử dụng ValueNotifier<T> hoặc ChangeNotifier của Flutter để lưu giữ trạng thái ứng dụng
  - **ValueNotifier<T>**: lưu giữ một giá trị kiểu T. Khi giá trị thay đổi thì báo hiệu cho các đối tượng lắng nghe (i.e., các widget builder của lớp UI)

```
// Tạo một giá trị int mà các đối tượng khác có thể lắng nghe các thay đổi
final myStateNotifier = ValueNotifier<int>(1);
```

```
// Thực hiện thay đổi giá trị, các đối tượng lắng nghe sẽ được báo hiệu
myStateNotifier.value = 2;
```

# Quản lý trạng thái đơn giản

3. Cung cấp cho layer UI cách lắng nghe thay đổi về trạng thái
  - **ValueNotifier<T>**: lưu giữ một giá trị đơn kiểu T. Khi giá trị thay đổi thì báo hiệu cho các đối tượng lắng nghe

```
class FavoriteNotifier extends ValueNotifier<bool> {  
    // set initial value to false  
    FavoriteNotifier() : super(false);  
  
    // get reference to service layer  
    final _storageService = getIt<StorageService>();  
  
    // a method to call from the outside  
    void toggleFavoriteStatus(Song song) {  
        value = !value;  
        _storageService.updateFavoriteSong(song, value);  
    }  
}
```



# Quản lý trạng thái đơn giản



3. Cung cấp cho layer UI cách lắng nghe thay đổi về trạng thái
  - **ChangeNotifier:** dùng để thừa kế hoặc [mixin](#), cung cấp một API báo hiệu cho phép các đối tượng đăng ký nhận các báo hiệu

```
class CartModel extends ChangeNotifier {  
    final List<Item> _items = [];  
  
    UnmodifiableListView<Item> get items => UnmodifiableListView(_items);  
    int get totalPrice => _items.length * 42;  
  
    void add(Item item) {  
        _items.add(item);  
        // This call tells the widgets that are listening to this model to rebuild.  
        notifyListeners();  
    }  
  
    void removeAll() {  
        _items.clear();  
        // This call tells the widgets that are listening to this model to rebuild.  
        notifyListeners();  
    }  
}
```

# Quản lý trạng thái đơn giản

3. Cung cấp cho layer UI cách lắng nghe thay đổi về trạng thái
  - Bên cạnh dùng **ValueNotifier<T>**, **ChangeNotifier** để lưu giữ trạng thái ứng dụng, có thể sử dụng **Future** hoặc **Stream** do Flutter có hỗ trợ các widget builder cho Future, Stream



# Quản lý trạng thái đơn giản

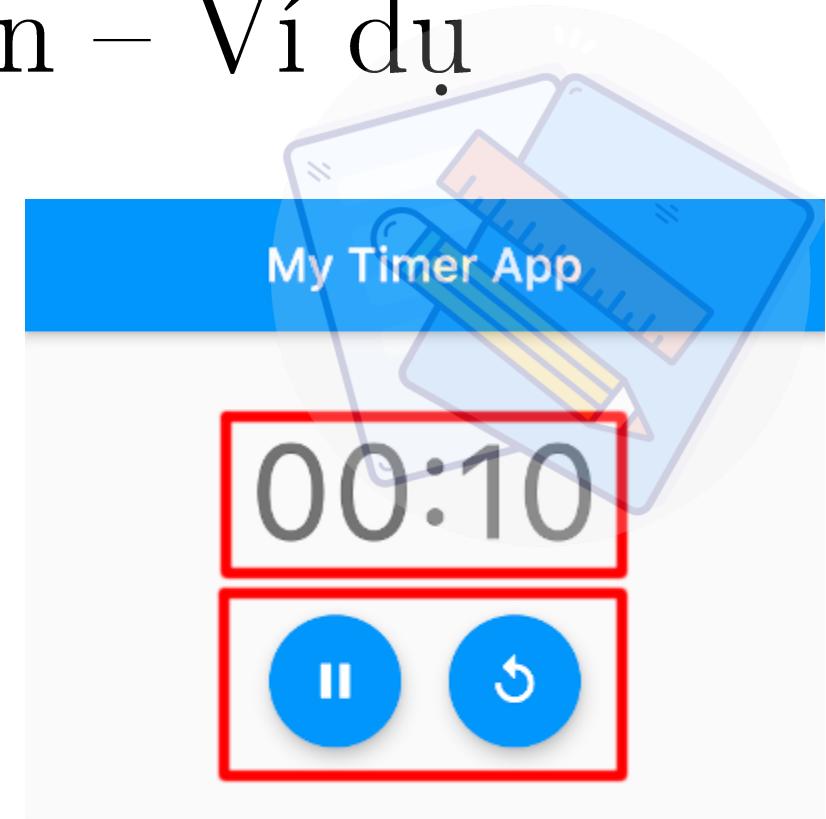
4. Tái tạo lại UI sau khi trạng thái thay đổi
  - Flutter cung cấp các widget \*Builder lắng nghe các thay đổi trạng thái và tái tạo lại UI
  - Tùy thuộc vào cách lưu giữ trạng thái ứng dụng
    - + [ValueListenableBuilder](#) dùng với **ValueNotifier**
    - + [ListenableBuilder](#) dùng với **ChangeNotifier** (và **ValueNotifier**)
    - + [FutureBuilder](#) dùng với **Future**
    - + [StreamBuilder](#) dùng với **Stream**



# Quản lý trạng thái đơn giản – Ví dụ

Ứng dụng Timer: gồm hai đơn vị trạng thái cần quản lý

- Chuỗi thời gian (kiểu String)
- Cấu hình các nút điều khiển (kiểu enum)



initial



started



paused

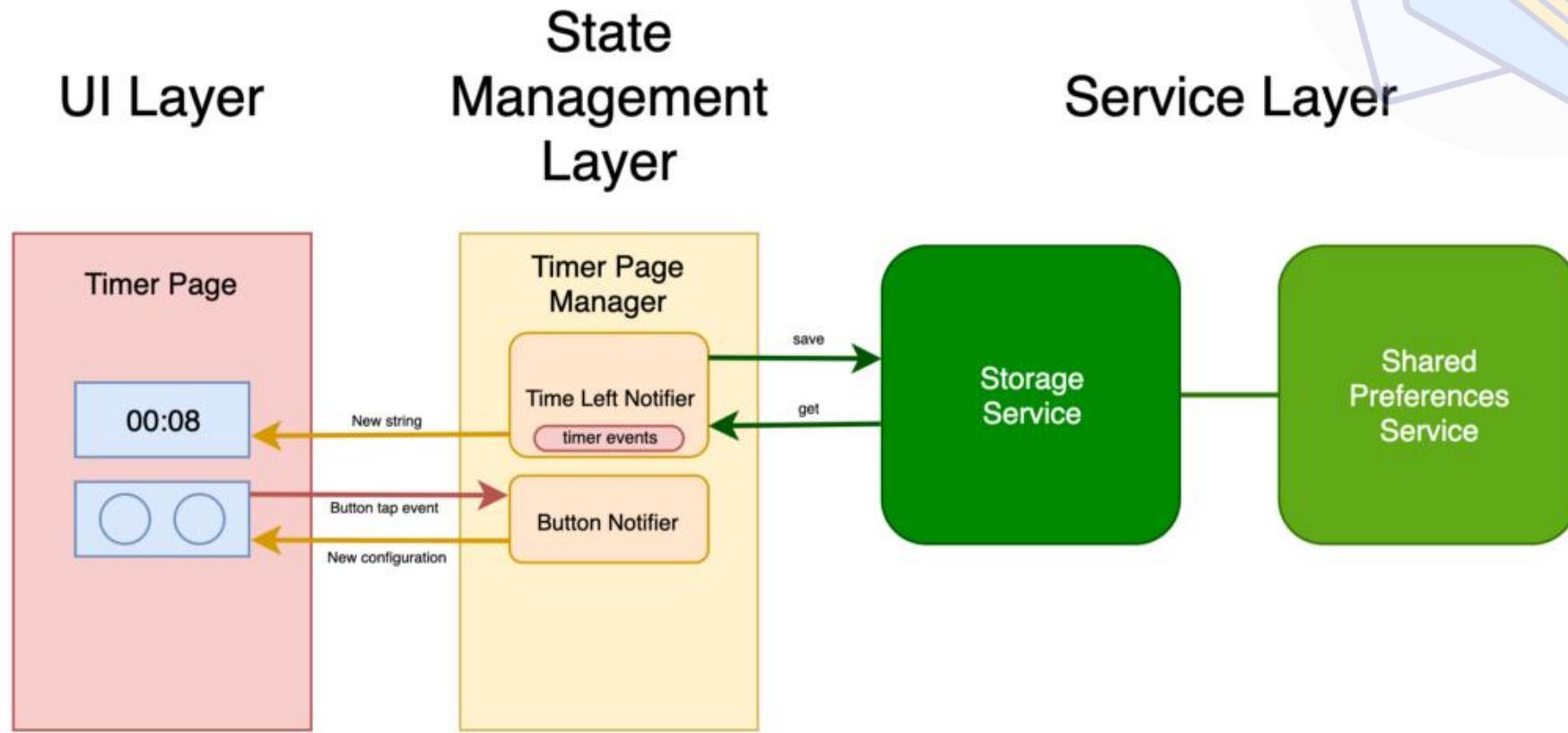
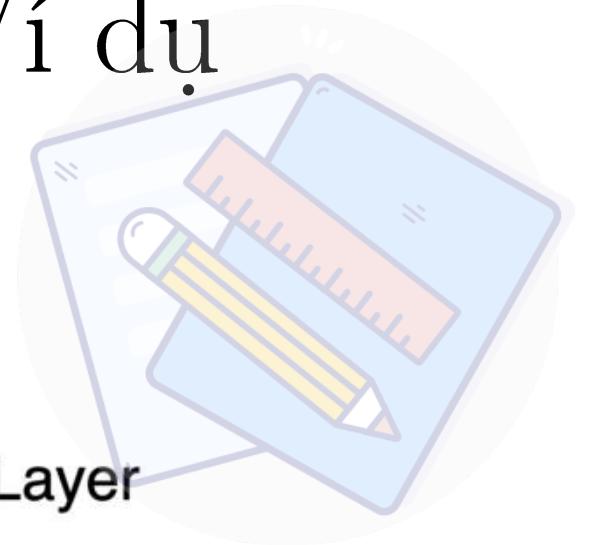


finished



# Quản lý trạng thái đơn giản – Ví dụ

Ứng dụng Timer: kiến trúc của ứng dụng

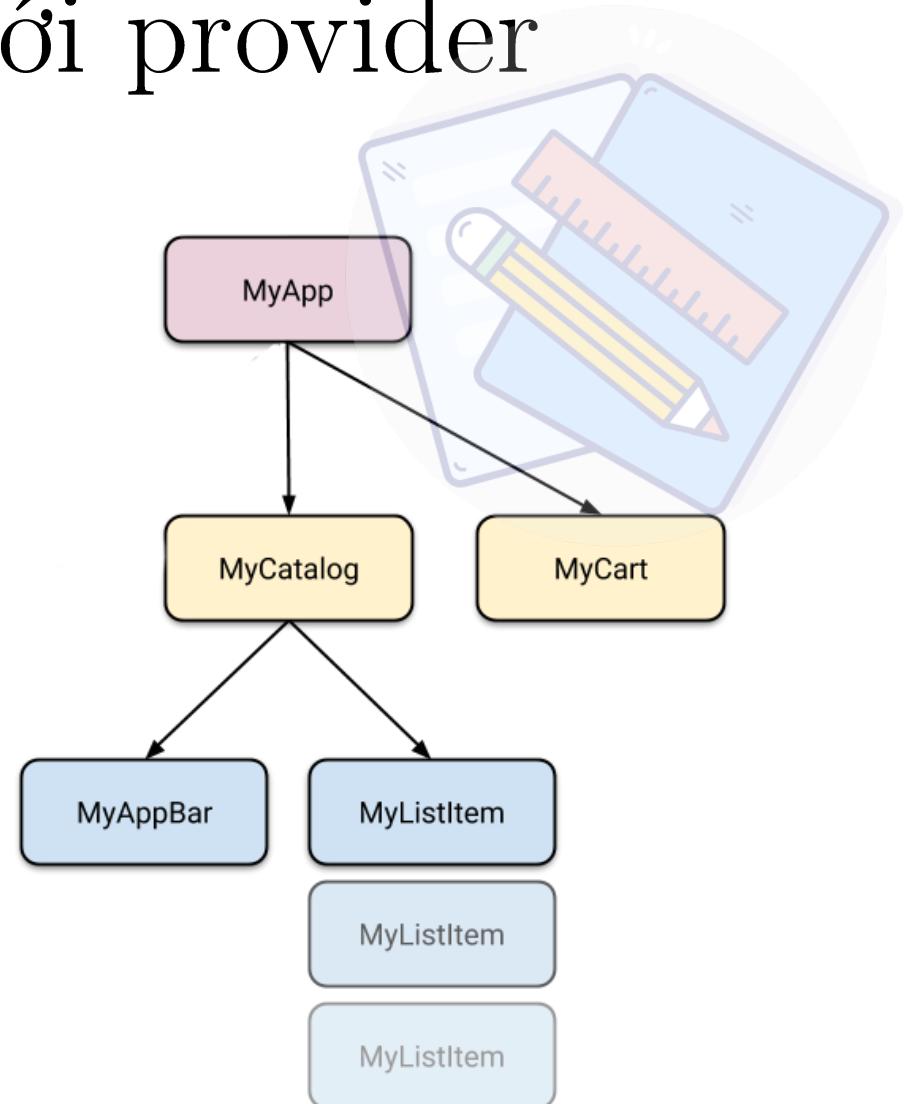


# Truyền nhận trạng thái trong cây widget với provider



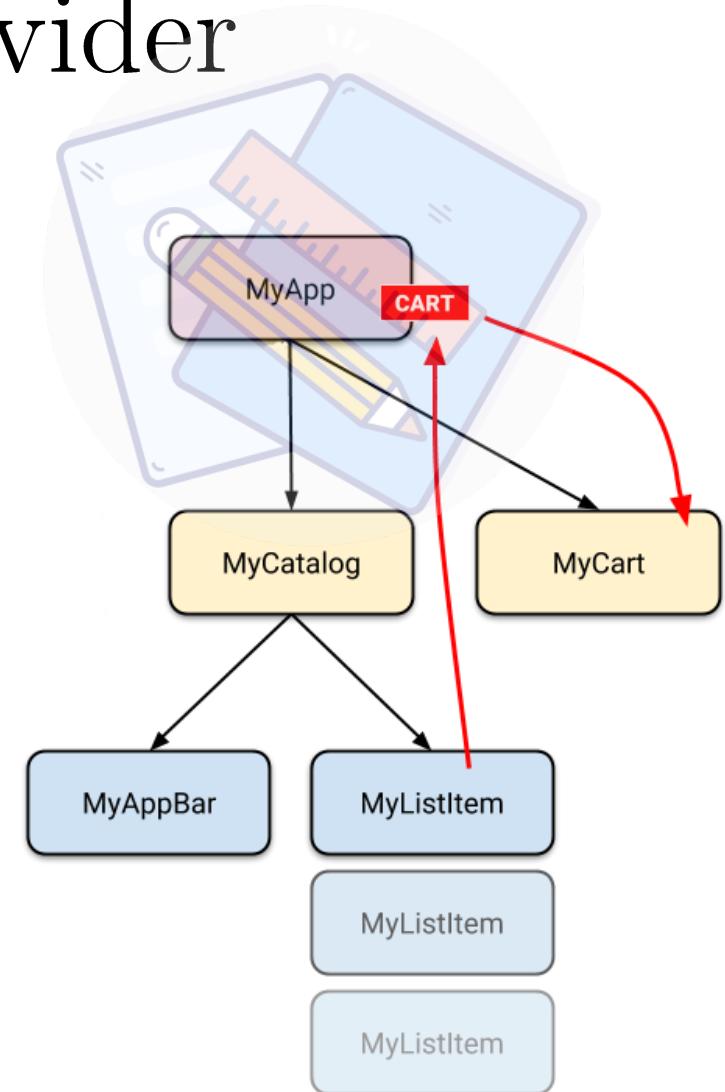
# Truyền nhận trạng thái với provider

- Xét ứng dụng ví dụ với cây widget như trong hình
  - Lưu trữ trạng thái hiện của giỏ hàng (MyCart) tại đâu trong cây widget?



# Truyền nhận trạng thái với provider

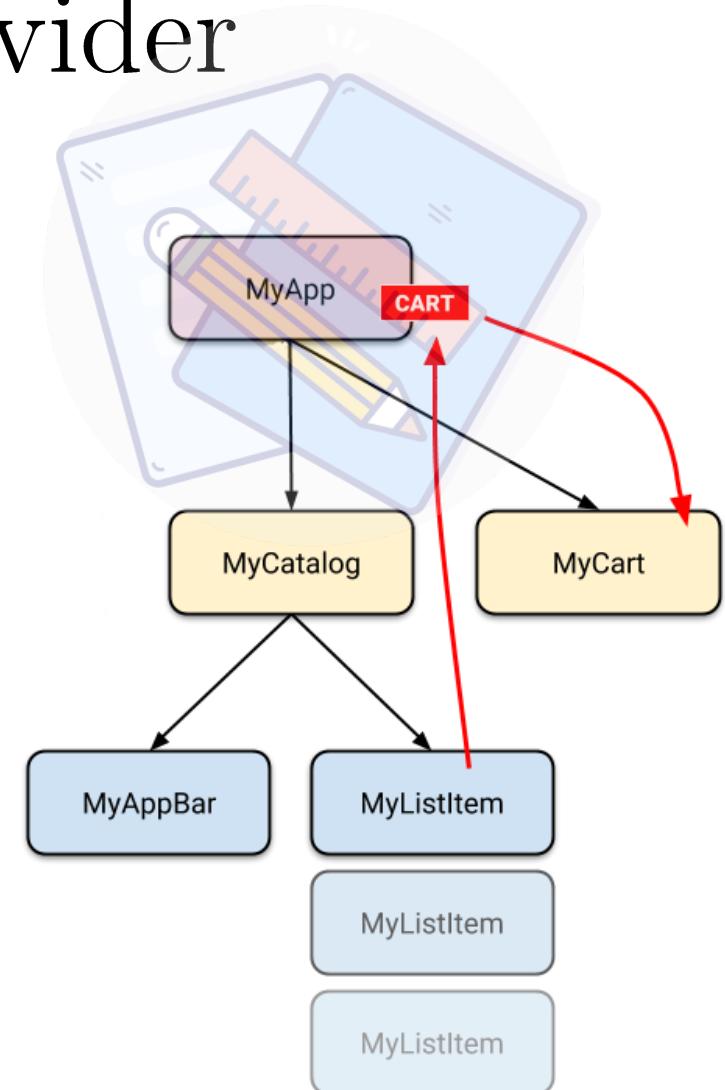
- **Nâng trạng thái lên:** giữ trạng thái phía trên các widget sử dụng trạng thái đó
- Tại sao?
  - Flutter là framework dạng khai báo: nội dung của widget thay đổi thì cần tái tạo lại widget đó (thay vì cập nhật)
  - Widget được tạo trong phương thức build của widget cha -> đặt trạng thái trong widget cha/tổ tiên



# Truyền nhận trạng thái với provider

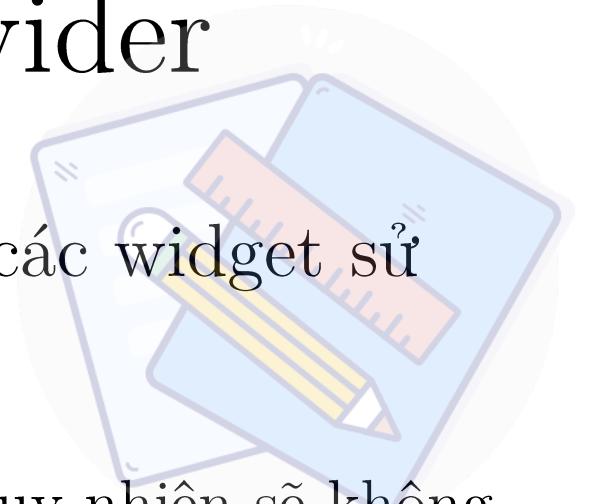
- **Nâng trạng thái lên:** giữ trạng thái phía trên các widget sử dụng trạng thái đó

```
Widget build(BuildContext context) {  
  var cartModel = somehowGetMyCartModel(context);  
  return SomeWidget(  
    // Just construct the UI once, using the current state of the cart.  
    // ...  
  );  
}
```



# Truyền nhận trạng thái với provider

- **Nâng trạng thái lên:** giữ trạng thái phía trên các widget sử dụng trạng thái đó
  - Làm thế nào truy cập trạng thái?
  - Widget cha có thể gửi trạng thái xuống widget con, tuy nhiên sẽ không khả thi khi cây widget lớn
- Flutter cung cấp cơ chế cho phép các widget cung cấp dữ liệu và các dịch vụ cho các widget con của chúng: InheritedWidget
- Nhà phát triển thường không làm việc một cách trực tiếp với InheritedWidget mà sử dụng các gói thư viện widget wrapper ở mức cao như gói **provider**



# Truyền nhận trạng thái với provider

pubspec.yaml

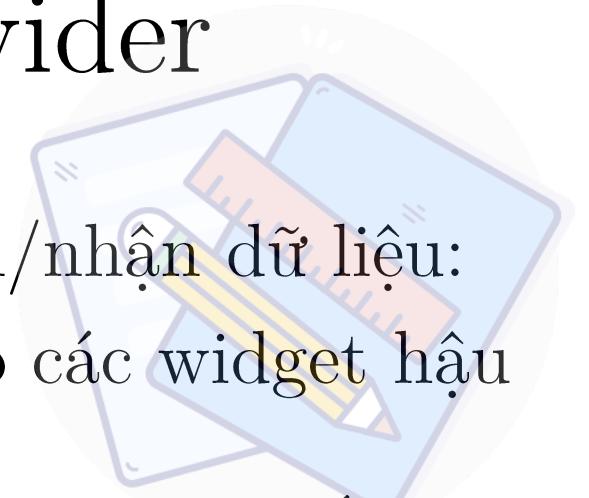
```
dependencies:  
  flutter:  
    sdk: flutter  
  provider: ^6.0.3
```

Khai báo sử dụng gói provider với:

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



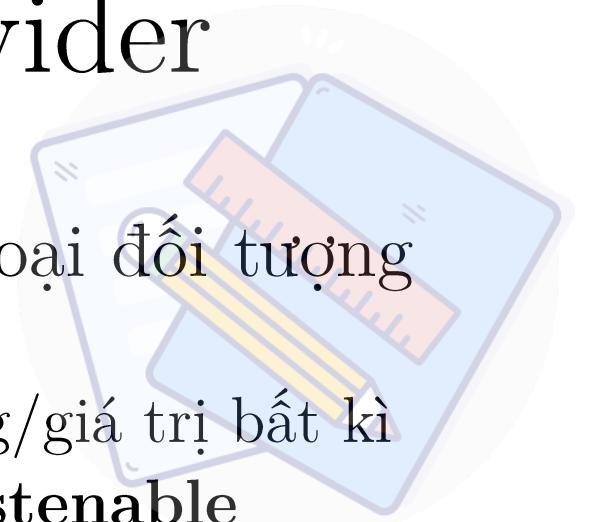
# Truyền nhận trạng thái với provider



Gói provider cung cấp các widget để hỗ trợ truyền/nhận dữ liệu:

- **Provider:** widget cung cấp đối tượng/giá trị cho các widget hậu duệ truy xuất
  - Provider không chỉ cung cấp mà còn có thể tạo, lắng nghe, hủy đối tượng
  - Đặt Provider phía trên các widget cần truy xuất đối tượng
  - Chỉ nên đặt Provider ở vị trí cao vừa đủ
- **Consumer/Selector:** widget nhận đối tượng từ Provider và chuyển cho builder (hàm tái tạo UI)
  - Consumer/Selector phải là hậu duệ của Provider
  - builder được gọi mỗi khi đối tượng từ Provider gửi báo hiệu thay đổi
  - Nên đặt Consumer/Selector ở vị trí thấp nhất có thể

# Truyền nhận trạng thái với provider

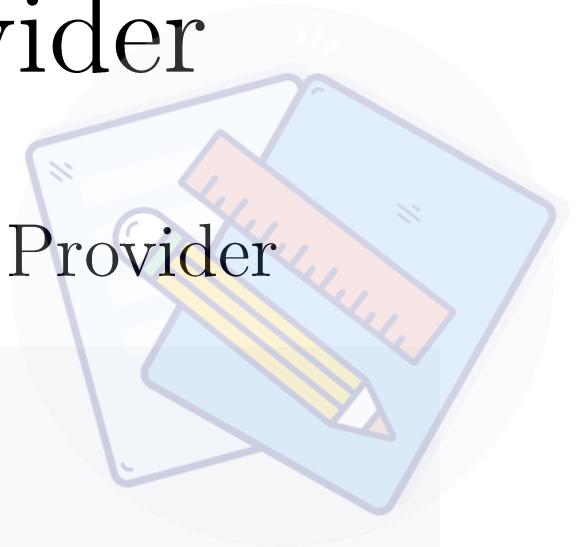


- Gói provider hỗ trợ nhiều loại Provider cho các loại đối tượng khác nhau
  - **Provider**: dạng cơ bản nhất, cung cấp một đối tượng/giá trị bất kì
  - **ListenableProvider**: cung cấp và *lắng nghe* một **Listenable**
  - **ChangeNotifierProvider**: cung cấp và *lắng nghe* một **ChangeNotifier**
  - **ValueListenableProvider**: *lắng nghe* một **ValueListenable** và cung cấp giá trị hiện thời của nó
  - **StreamProvider**: *lắng nghe* một **Stream** và cung cấp nội dung của nó
  - **FutureProvider**: *lắng nghe* một **Future** và cung cấp kết quả của nó
- Các Provider có phύc thức xây dựng **\*Provider.value()** thay vì tạo mới đối tượng, tái sử dụng một đối tượng có sẵn

# Truyền nhận trạng thái với provider

- **MultiProvider**: widget cho phép kết hợp nhiều Provider

```
void main() {  
  runApp(  
    MultiProvider(  
      providers: [  
        ChangeNotifierProvider(create: (context) => CartModel()),  
        Provider(create: (context) => SomeOtherClass()),  
      ],  
      child: const MyApp(),  
    ),  
  );  
}
```



# Truyền nhận trạng thái với provider

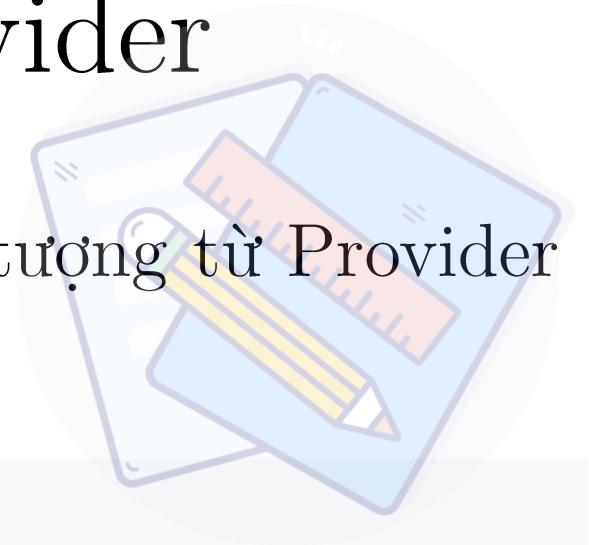


- **\*ProxyProvider<T, R>**: cung cấp và đồng bộ hóa đối tượng kiểu R với giá trị ngoài thuộc kiểu T
  - Được dùng khi đối tượng cần cung cấp phụ thuộc vào giá trị khác cần theo dõi thay đổi

```
providers: [
    // In this sample app, CatalogModel never changes, so a simple Provider
    // is sufficient.
    Provider(create: (context) => CatalogModel()),
    // CartModel is implemented as a ChangeNotifier, which calls for the use
    // of ChangeNotifierProvider. Moreover, CartModel depends
    // on CatalogModel, so a ProxyProvider is needed.
    ChangeNotifierProxyProvider<CatalogModel, CartModel>(
        create: (context) => CartModel(),
        update: (context, catalog, cart) {
            if (cart == null) throw ArgumentError.notNull('cart');
            cart.catalog = catalog;
            return cart;
        },
    ), // ChangeNotifierProxyProvider
],
```

# Truyền nhận trạng thái với provider

- Consumer<T>/Selector<T, R>: nhận đối tượng từ Provider và chuyển cho builder



```
return Consumer<CartModel>(
    builder: (context, cart, child) {
        return Text('Total price: ${cart.totalPrice}');
    },
);
```

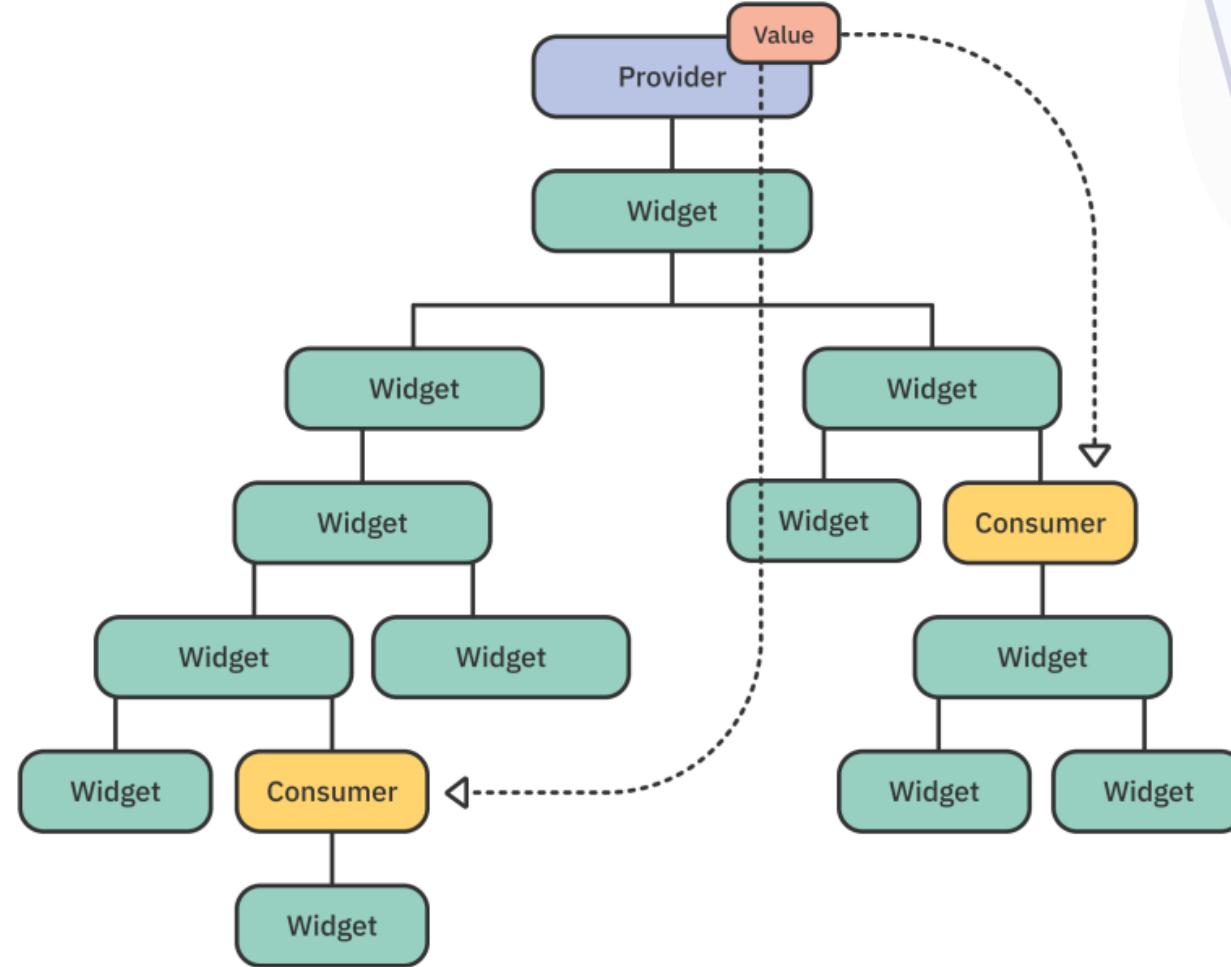
# Truyền nhận trạng thái với provider

- **Consumer<T>/Selector<T, R>**: nhận đối tượng từ Provider và chuyển cho builder

```
return Consumer<CartModel>(
    builder: (context, cart, child) => Stack(
        children: [
            // Use SomeExpensiveWidget here, without rebuilding every time.
            if (child != null) child,
            Text('Total price: ${cart.totalPrice}'),
        ],
    ),
    // Build the expensive widget here.
    child: const SomeExpensiveWidget(),
);
```

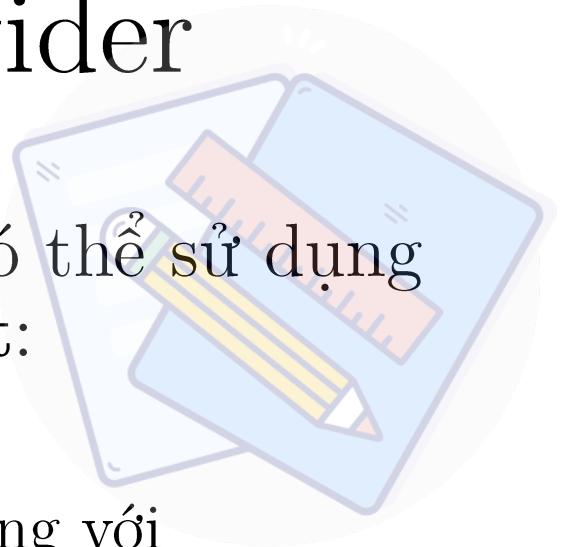


# Truyền nhận trạng thái với provider



# Truyền nhận trạng thái với provider

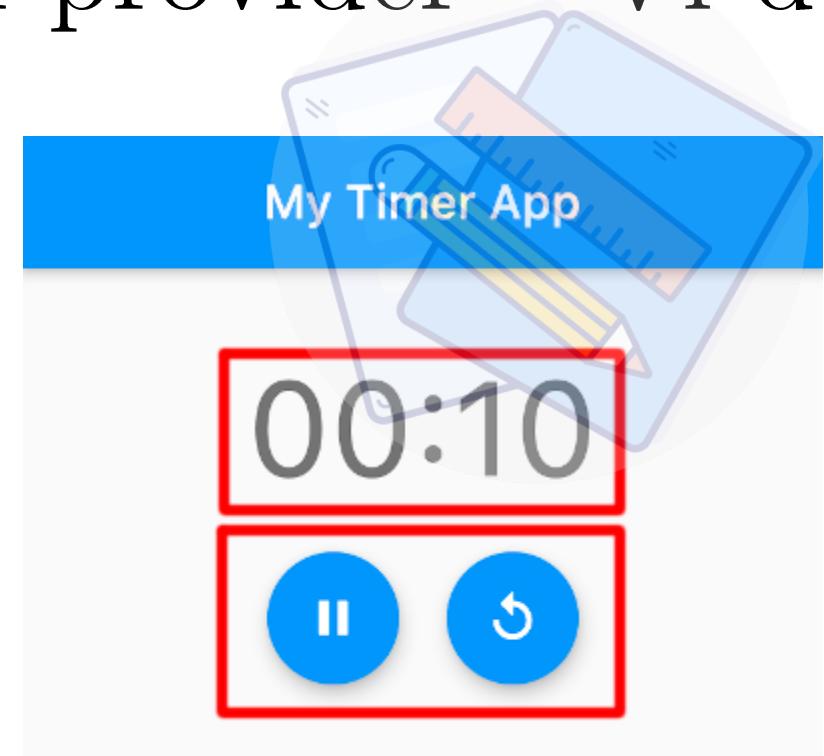
- Thay vì sử dụng widget **Consumer/Selector**, có thể sử dụng các phương thức mở rộng sau của **BuildContext**:
  - **context.read<T>()**: đọc ra đối tượng T, tương đương với **Provider.of<T>(context, listen: false)**
  - **context.watch<T>()**: đọc và *lắng nghe* các thay đổi từ T, tương đương với **Provider.of<T>(context, listen: true)**
  - **context.select<T, R>(R cb(T value))**: đọc và *lắng nghe* thay đổi của giá trị kiểu R dẫn xuất từ T



# Truyền nhận trạng thái với provider – Ví dụ

Ứng dụng Timer: gồm hai đơn vị trạng thái cần quản lý

- Chuỗi thời gian (kiểu String)
- Cấu hình các nút điều khiển (kiểu enum)



initial



started



paused

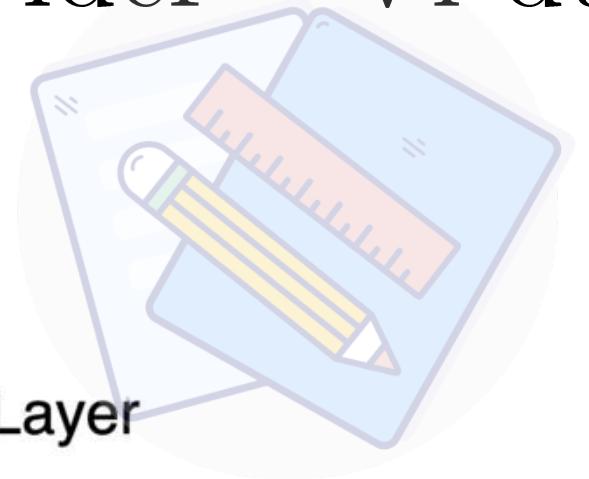
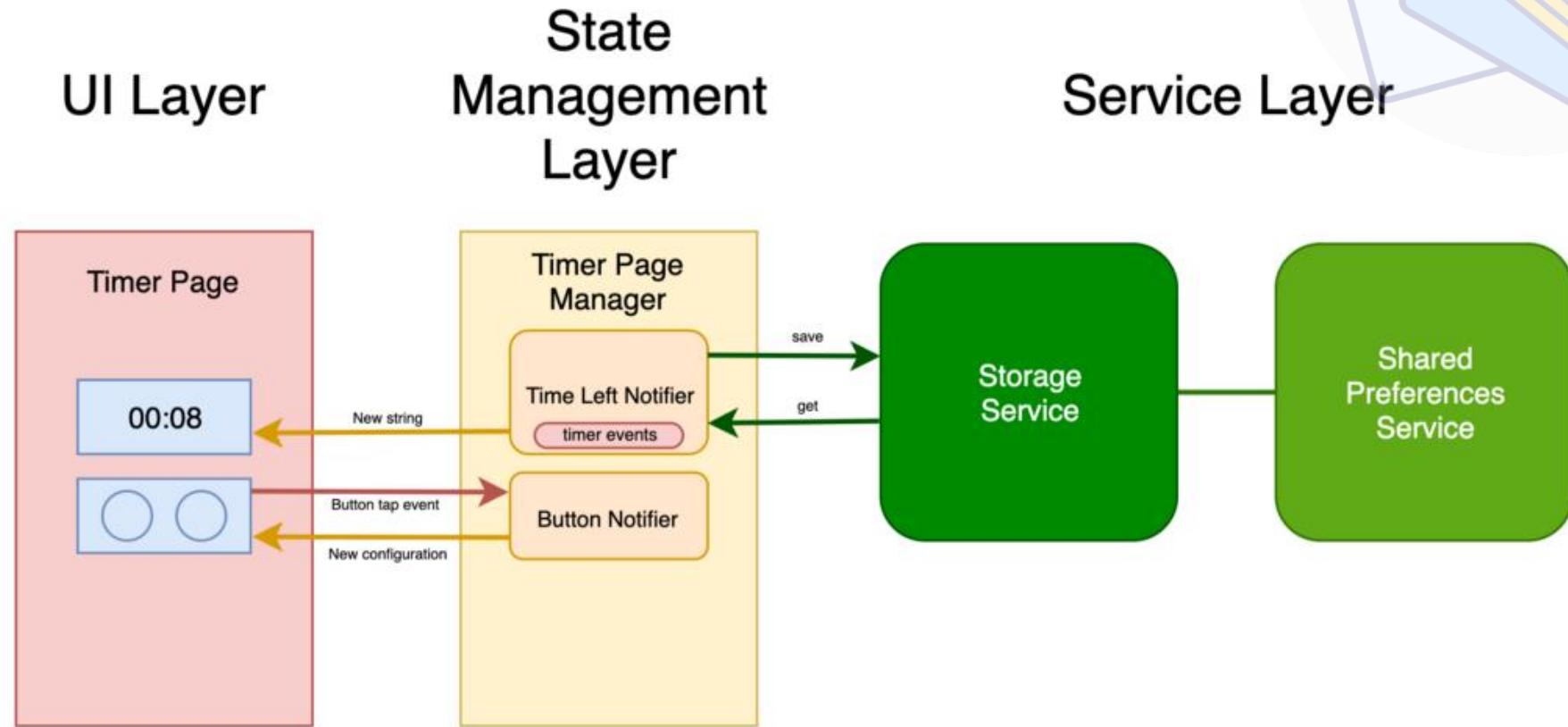


finished



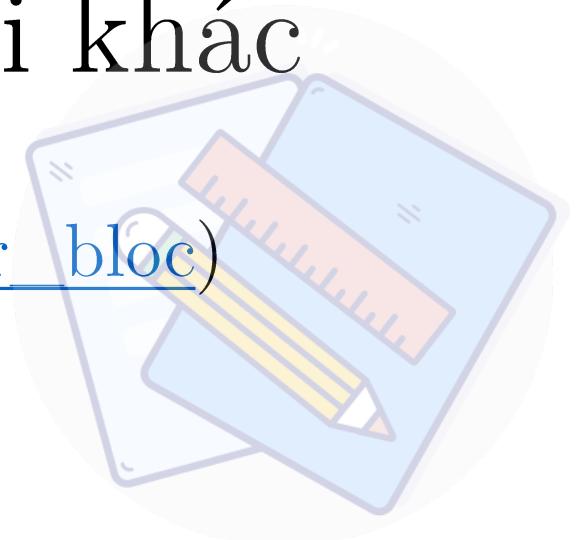
# Truyền nhận trạng thái với provider – Ví dụ

Ứng dụng Timer: kiến trúc của ứng dụng



# Các giải pháp quản lý trạng thái khác

- **BLoC/Cubit** ([https://pub.dev/packages/flutter\\_bloc](https://pub.dev/packages/flutter_bloc))
- Riverpod (<https://riverpod.dev/>)
- MobX (<https://github.com/mobxjs/mobx.dart>)
- Redux ([https://pub.dev/packages/flutter\\_redux](https://pub.dev/packages/flutter_redux))
- ...



# Quản lý trạng thái – Tổng kết

- Quản lý trạng thái cục bộ
  - setState (cây widget được tái xây dựng khi setState được gọi)
- Quản lý trạng thái ứng dụng đơn giản
  - Lưu giữ các biến trạng thái trong các kiểu dữ liệu sau: **ValueNotifier**, **ChangeNotifier**, **Future**, **Stream**
  - Chia sẻ các biến trạng thái với **provider** hoặc **get\_it**
  - Dùng các widget builder tương ứng với kiểu biến trạng thái sử dụng:  
**ValueListenableBuilder**, **ListenableBuilder**, **FutureBuilder**,  
**StreamBuilder**, **Consumer/Selector** (của provider)





# Các widget tương tác

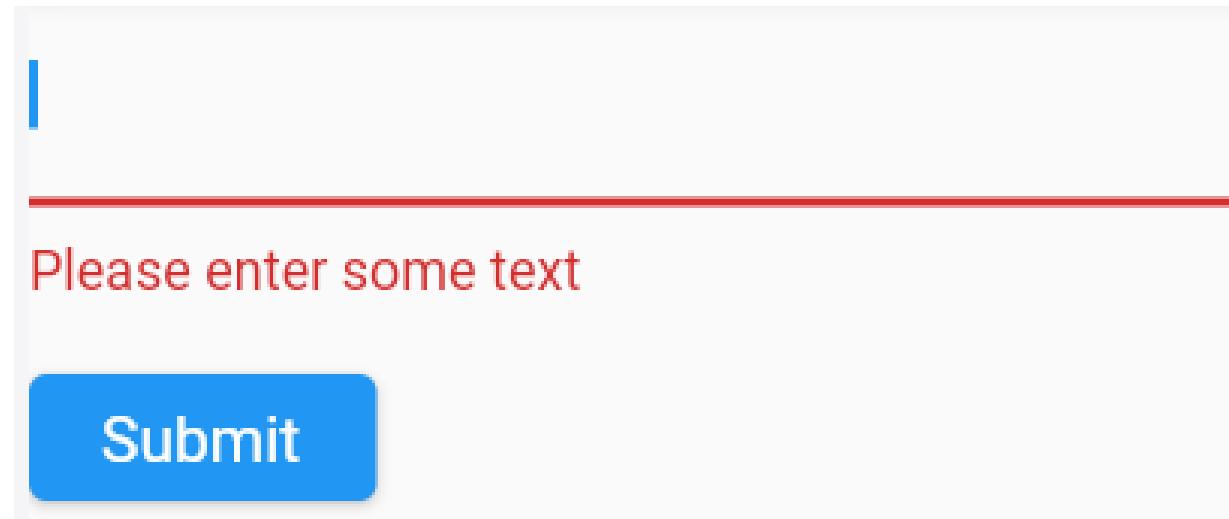
# TextField và TextFormField

- **TextField**: widget cho phép người dùng nhập dữ liệu từ bàn phím
  - **decoration**: điều khiển cách hiển thị TextField
  - **keyboardType**: loại bàn phím nhập liệu
  - **obscureText**: giấu/ẩn văn bản nhập liệu
  - **maxLines**: số dòng hiển thị tối đa
  - **autofocus**, **focusNode**: điều khiển focus
  - **inputFormatters**: kiểm tra và định dạng nhập liệu
  - **textInputAction**: điều khiển hành xử của action button (phím Enter)
  - **controller**: đối tượng **TextEditingController** điều khiển nhập liệu
  - **onChanged**: hàm được gọi khi văn bản nhập liệu thay đổi
  - **onSubmitted**: hàm được gọi khi kết thúc nhập liệu



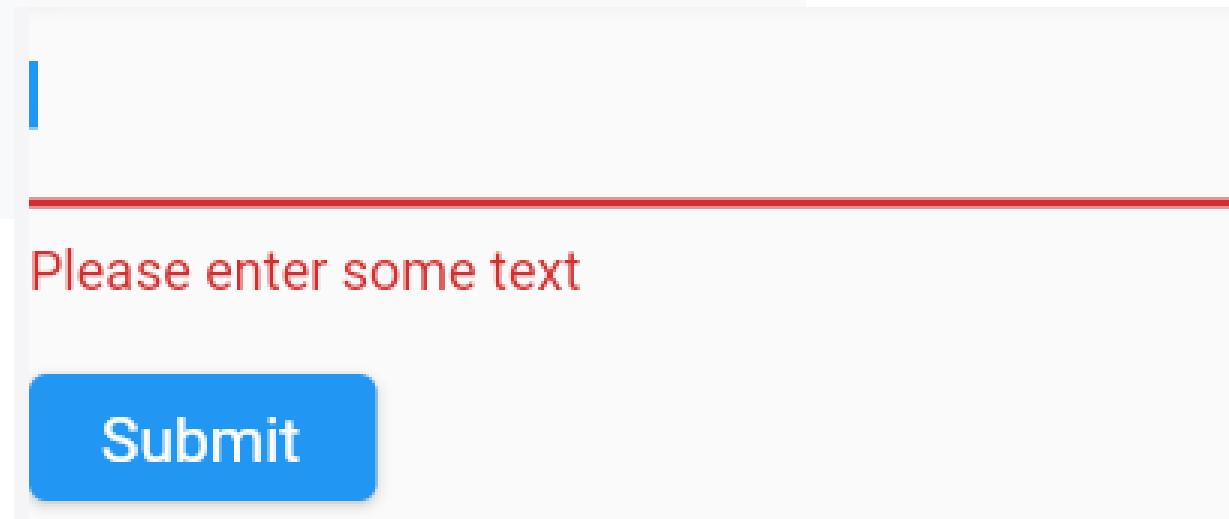
# TextField và TextFormField

- TextFormField: widget kết hợp TextField vào một FormField
  - Hỗ trợ lưu (save), đặt lại (reset), kiểm tra (validate) giá trị TextField và có thể hiển thị lỗi nếu có
  - Thường được sử dụng với widget Form – một widget gom nhóm các widget FormField lại với nhau



# TextField và TextFormField

```
TextFormField(  
    // The validator receives the text that the user has entered.  
    validator: (value) {  
        if (value == null || value.isEmpty) {  
            return 'Please enter some text';  
        }  
        return null;  
    },  
) ,
```



The screenshot shows a TextFormField with a red underline below it, indicating an error. The placeholder text 'Please enter some text' is displayed in red. A blue 'Submit' button is at the bottom.

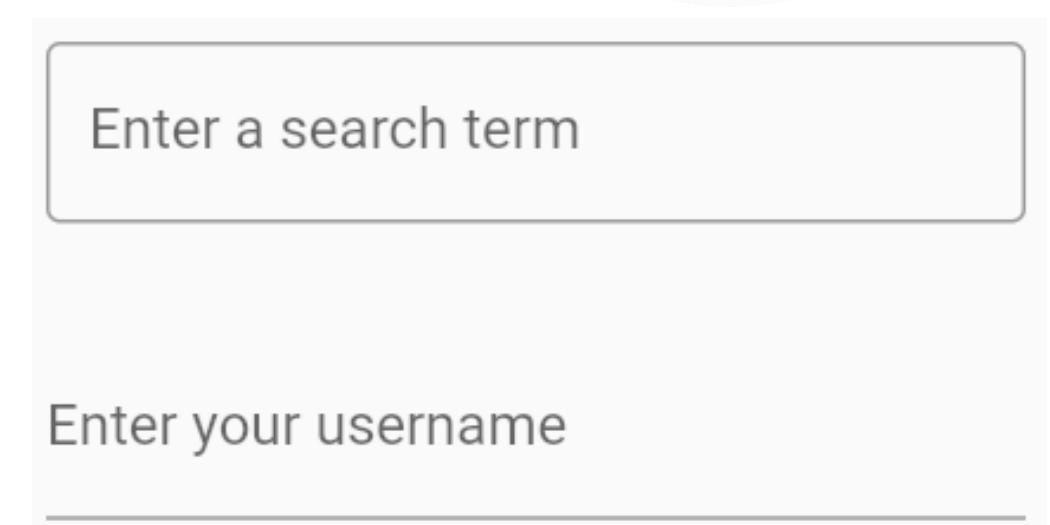


# TextField và TextFormField

- TextField và TextFormField có thể nhận vào một InputDecoration (thuộc tính decoration) để tạo đường viền, thêm nhãn, biểu tượng, văn bản gợi ý, văn bản lỗi, ...

```
decoration: InputDecoration(  
  border: OutlineInputBorder(),  
  hintText: 'Enter a search term',  
,
```

```
decoration: const InputDecoration(  
  border: UnderlineInputBorder(),  
  labelText: 'Enter your username',  
,
```



# TextField và TextFormField

- Xử lý dữ liệu nhập liệu với **TextField/TextFormField**:
  - Dùng callback [onChanged](#), được gọi mỗi khi dữ liệu nhập liệu thay đổi
  - Hoặc sử dụng một [TextEditingController](#) cho phép widget cha tương tác với trạng thái của TextField (e.g., đọc giá trị nhập liệu)

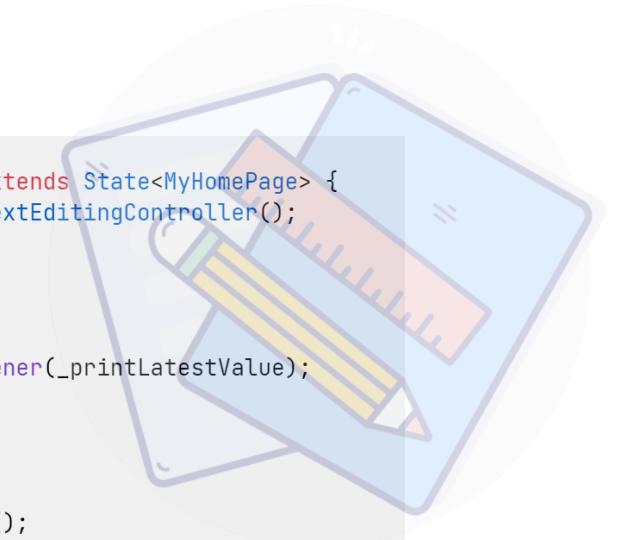
```
class _MyHomePageState extends State<MyHomePage> {
  final myController = TextEditingController();

  @override
  void initState() {
    super.initState();
    myController.addListener(_printLatestValue);
  }

  @override
  void dispose() {
    myController.dispose();
    super.dispose();
  }

  void _printLatestValue() {
    print('Second text field: ${myController.text}');
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Retrieve Text Input'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: TextFormField(
          controller: myController,
        ),
      );
    }
}
```



# TextField và TextFormField

- Khi TextField/TextFormField được chọn và sẵn sàng nhận nhập liệu, ta nói rằng nó đang có focus
- Có thể điều khiển focus của TextField/TextFormField bằng cách:
  - Dùng thuộc tính autofocus

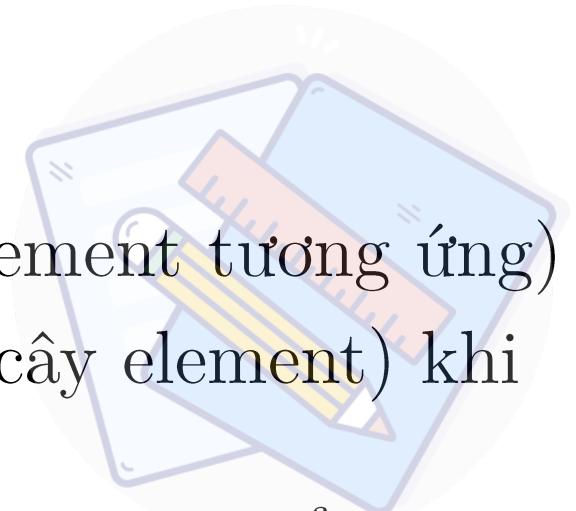
```
TextField(  
    autofocus: true,  
);
```

- Sử dụng [FocusNode](#)



# Thuộc tính key của widget

- Các key là các định danh cho các widget (và element tương ứng)
- Thuộc tính **key** giúp duy trì trạng thái (trong cây element) khi các widget di chuyển trong cây widget
- *Key thường được sử dụng khi cần thêm, xóa hoặc thay đổi thứ tự các widget trong tập hợp các widget cùng loại, có trạng thái*
- Có hai loại key:
  - [LocalKey](#) (ObjectKey, UniqueKey, ValueKey): các key phải là duy nhất trong tập các con cùng cha
  - [GlobalKey](#): các key phải là duy nhất trong ứng dụng



# GlobalKey và Form

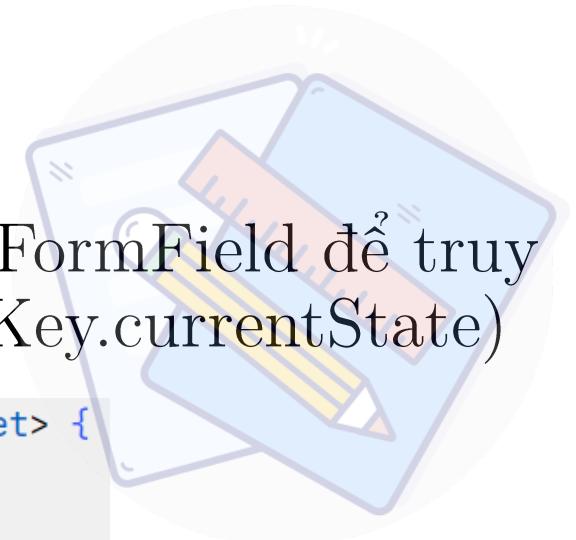
- GlobalKey: các key phải là duy nhất trong ứng dụng
- Có hai trường hợp sử dụng cho GlobalKey:
  - Cho phép truy xuất trạng thái của widget. GlobalKey thường được sử dụng với widget Form/FormField để truy cập trạng thái FormState/FormFieldState (GlobalKey.currentState)
  - Cho phép widget thay đổi cha mà không mất trạng thái (i.e., cần hiển thị một widget ở hai màn hình khác nhau với cùng một trạng thái)



# GlobalKey và Form

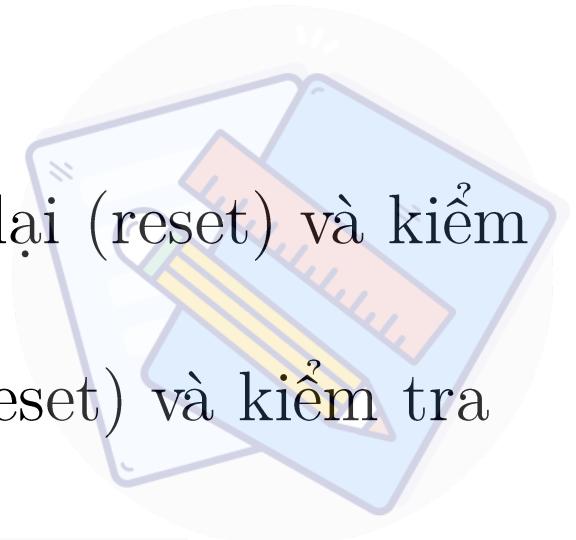
- GlobalKey thường được sử dụng với widget Form/FormField để truy cập trạng thái FormState/FormFieldState (GlobalKey.currentState)

```
class _My StatefulWidget extends State<My StatefulWidget> {  
    final _formKey = GlobalKey<FormState>();  
  
    @override  
    Widget build(BuildContext context) {  
        return Form(  
            key: _formKey,  
            child: Column(  
                children: <Widget>[  
                    // Add TextFormField and ElevatedButton here.  
                ],  
            ),  
        );  
    }  
}
```



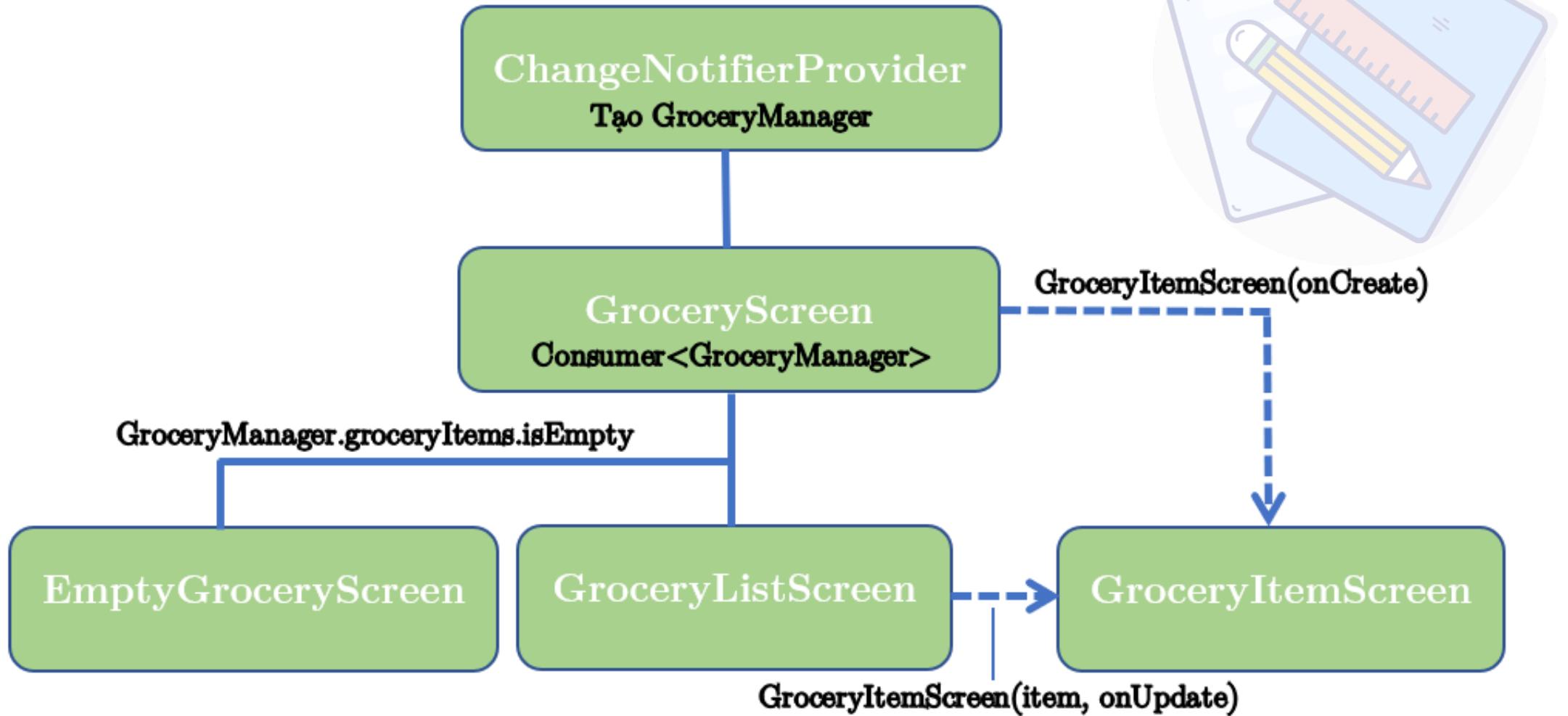
# GlobalKey và Form

- FormFieldState có thể được dùng để lưu (save), đặt lại (reset) và kiểm tra (validate) TextFormField
- FormState có thể được dùng để lưu (save), đặt lại (reset) và kiểm tra (validate) các TextFormField hậu duệ của Form



```
ElevatedButton(  
    onPressed: () {  
        // Validate returns true if the form is valid, or false otherwise.  
        if (_formKey.currentState!.validate()) {  
            // If the form is valid, display a snackbar. In the real world,  
            // you'd often call a server or save the information in a database.  
            ScaffoldMessenger.of(context).showSnackBar(  
                const SnackBar(content: Text('Processing Data')),  
            );  
        }  
    },  
    child: const Text('Submit'),  
,
```

# Ứng dụng Danh sách mua hàng



# Mô hình dữ liệu

```
enum Importance {  
    low,  
    medium,  
    high,  
}  
  
class GroceryItem {  
    final String id;  
    final String name;  
    final Importance importance;  
    final Color color;  
    final int quantity;  
    final DateTime date;  
    final bool isComplete;  
  
    GroceryItem({  
        required this.id,  
        required this.name,  
        required this.importance,  
        required this.color,  
        required this.quantity,  
        required this.date,  
        this.isComplete = false,  
    });
```

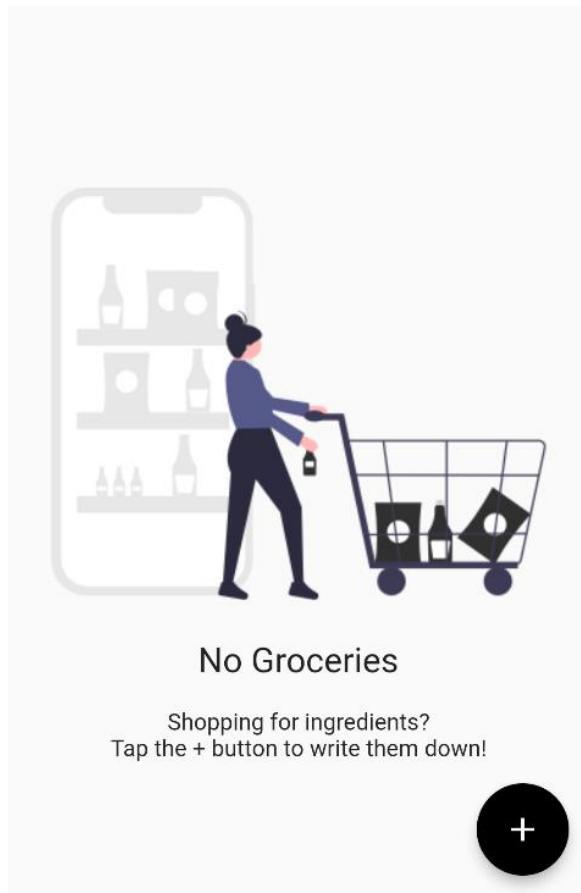
```
class GroceryManager extends ChangeNotifier {  
    final _groceryItems = <GroceryItem>[];  
  
    List<GroceryItem> get groceryItems => List.unmodifiable(_groceryItems);  
  
    void deleteItem(int index) {  
        _groceryItems.removeAt(index);  
        notifyListeners();  
    }  
  
    void addItem(GroceryItem item) {  
        _groceryItems.add(item);  
        notifyListeners();  
    }  
  
    void updateItem(GroceryItem item, int index) {  
        _groceryItems[index] = item;  
        notifyListeners();  
    }  
  
    void completeItem(int index, bool change) {  
        final item = _groceryItems[index];  
        _groceryItems[index] = item.copyWith(isComplete: change);  
        notifyListeners();  
    }  
}
```



# Thiết kế các trang ứng dụng

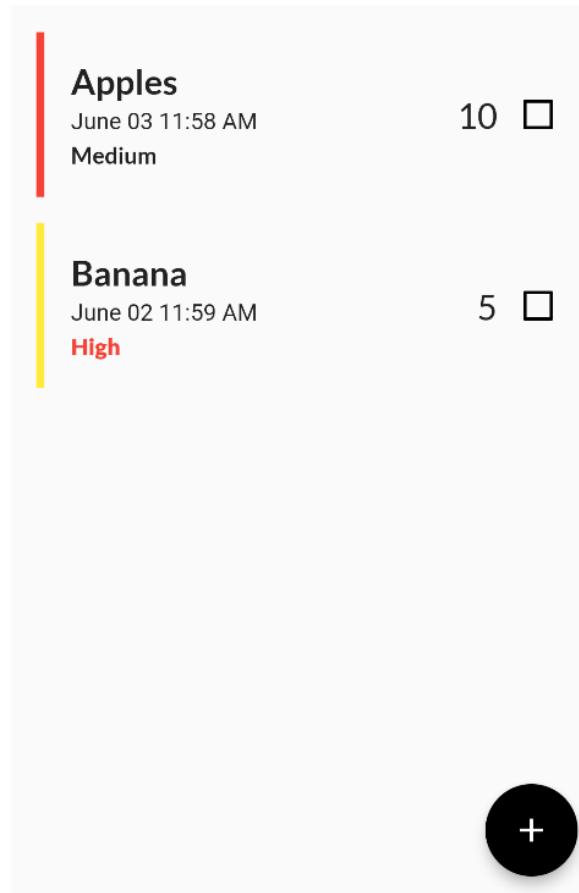
GroceryEmptyScreen

Grocery List



GroceryListScreen

Grocery List



GroceryItemScreen

←

Grocery Item

Item Name

Banana

Importance

low

medium

high

Date

02-06-2022

Select

Time of Day

11:59 AM

Select

Color

Select

Quantity 5



Banana  
June 02 11:59 AM

5

# Hàm build chính của ứng dụng

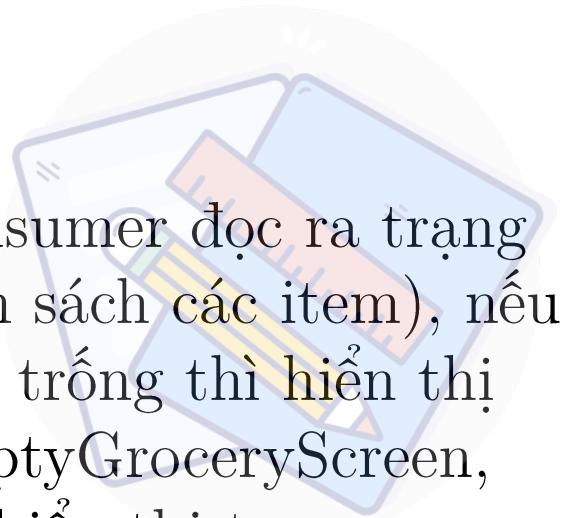
Dùng ChangeNotifierProvider  
gửi trạng thái (danh sách các  
item) xuống cây widget

```
class Foderlich extends StatelessWidget {  
  const Foderlich({Key? key}) : super(key: key);  
  @override  
  Widget build(BuildContext context) {  
    final theme = FoderlichTheme.light();  
    return MaterialApp(  
      debugShowCheckedModeBanner: false,  
      theme: theme,  
      title: 'Foderlich',  
      home: ChangeNotifierProvider(  
        create: (context) => GroceryManager(),  
        child: const GroceryScreen(),  
      ), // ChangeNotifierProvider  
    ); // MaterialApp  
  }  
}
```



# GroceryScreen

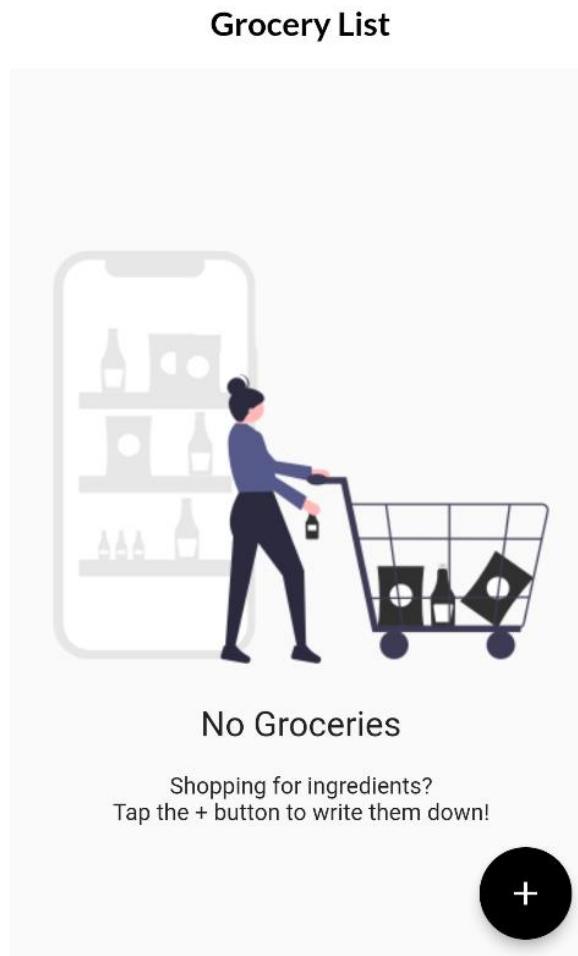
```
Widget buildGroceryScreen() {  
    return Consumer<GroceryManager>(  
        builder: (context, manager, child) {  
            if (manager.groceryItems.isNotEmpty) {  
                return GroceryListScreen(manager: manager);  
            } else {  
                return const EmptyGroceryScreen();  
            }  
        },  
    ); // Consumer  
}
```



Dùng Consumer đọc ra trạng thái (danh sách các item), nếu danh sách trống thì hiển thị trang EmptyGroceryScreen, ngược lại hiển thị trang GroceryListScreen

# EmptyGroceryScreen và GroceryListScreen

GroceryEmptyScreen



GroceryListScreen

Grocery List

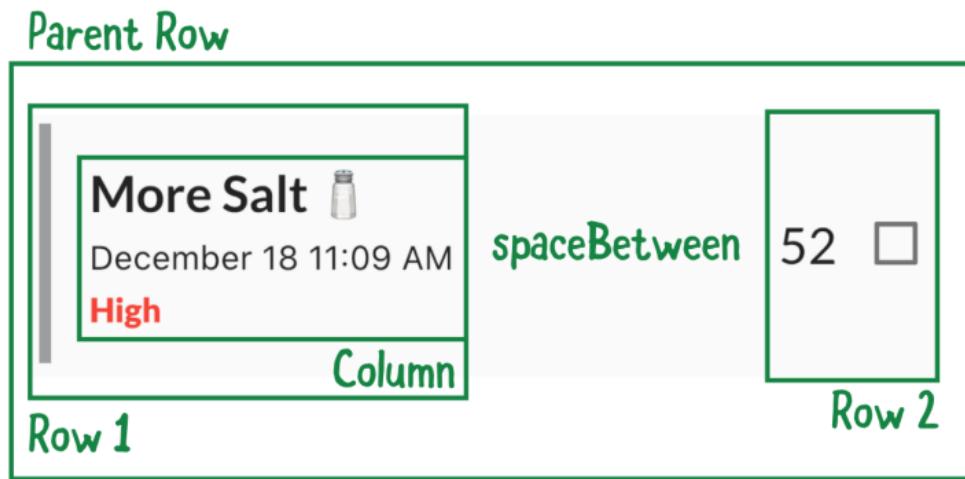
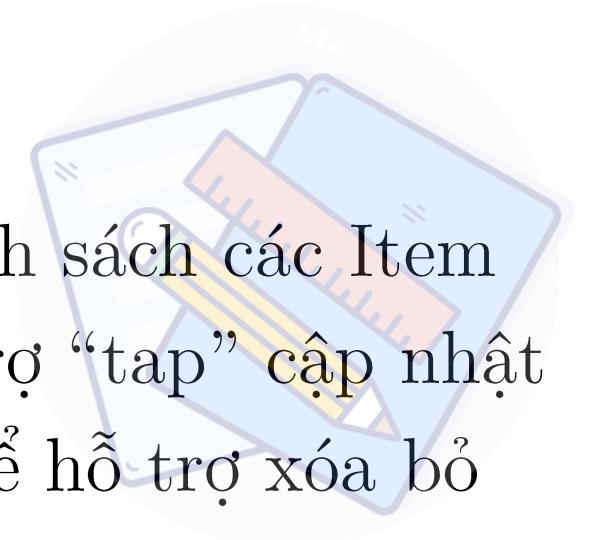
<b>Apples</b> June 03 11:58 AM Medium	10 <input type="checkbox"/>
<b>Banana</b> June 02 11:59 AM High	5 <input type="checkbox"/>

+



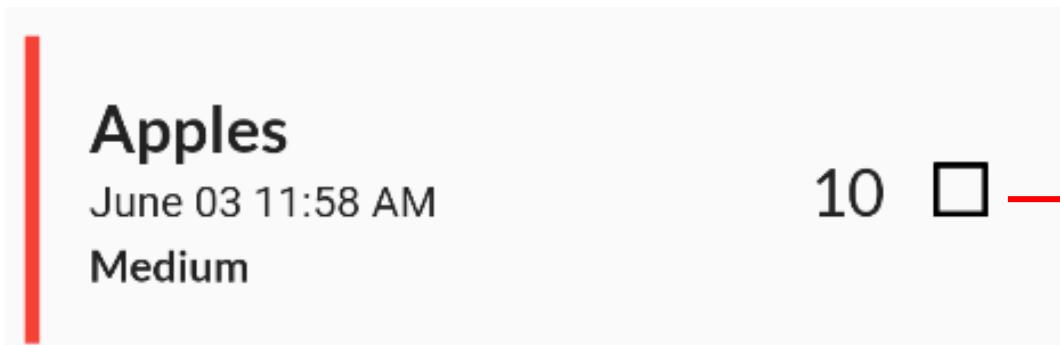
# GroceryListScreen

- Tạo danh sách cuộn các GroceryTile từ một danh sách các Item
- Bao GroceryTitle trong widget InkWell để hỗ trợ “tap” cập nhật
- Bao widget InkWell trong widget Dismissible để hỗ trợ xóa bỏ

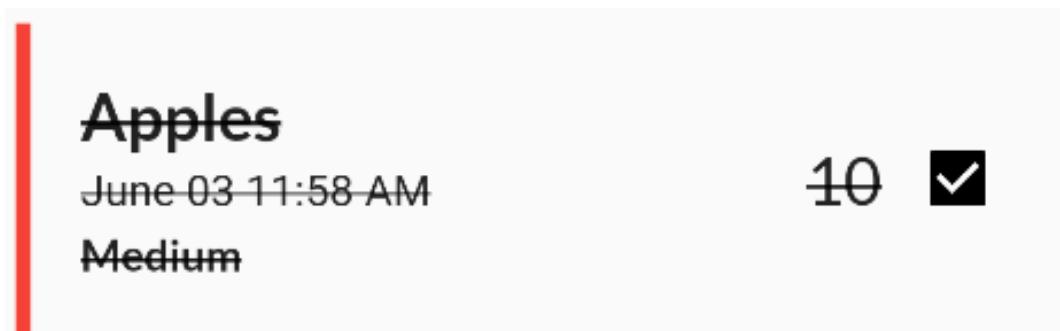


# GroceryTile

- Hiển thị thông tin một Item



```
Widget buildCheckbox() {  
    return Checkbox(  
        value: item.isComplete,  
        onChanged: onComplete,  
    );  
}
```



```
child: InkWell(  
    child: GroceryTile(  
        key: Key(item.id),  
        item: item,  
        onComplete: (change) {  
            if (change != null) {  
                manager.completeItem(index, change);  
            }  
        },  
    ), // GroceryTile
```

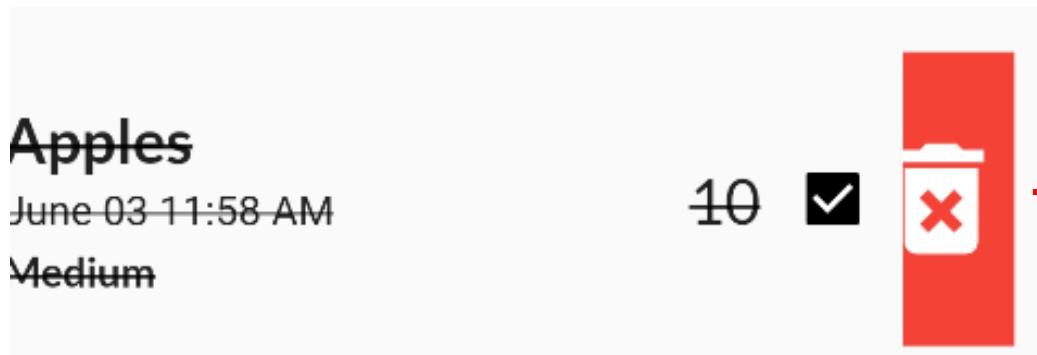
# Dismissible

- Widget có thể được loại bỏ bằng cách vuốt theo hướng chỉ định
  - **key**: thuộc tính dùng định danh widget
  - **direction**: chỉ định hướng vuốt widget (DismissDirection)
  - **background**: widget nền cho widget con, trường hợp secondaryBackground được chỉ định, background làm widget nền cho hướng vuốt xuống (down) hoặc sang phải (startToEnd)
  - **secondaryBackground** (chỉ khi background được chỉ định): widget nền cho widget con cho hướng vuốt lên (up) hoặc sang trái (endToStart)
  - **confirmDismiss**: hàm được gọi để xác nhận loại bỏ, trả về true widget sẽ được loại bỏ, ngược lại quay về vị trí cũ
  - **onDismissed**: hàm được gọi sau khi widget đã được loại bỏ



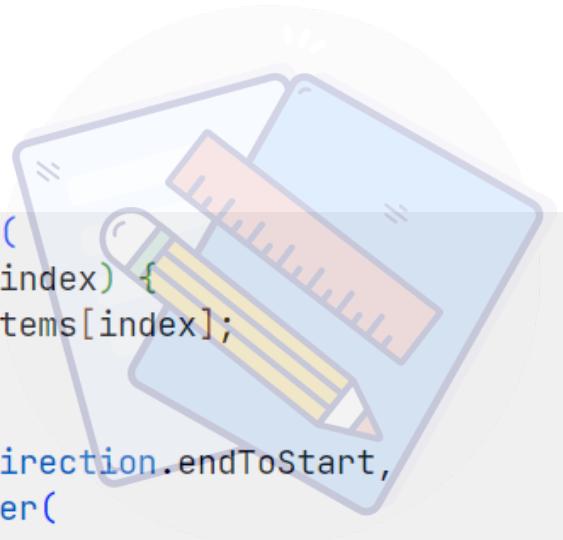
# GroceryListScreen

Dismissible bắt buộc cung cấp giá trị cho key

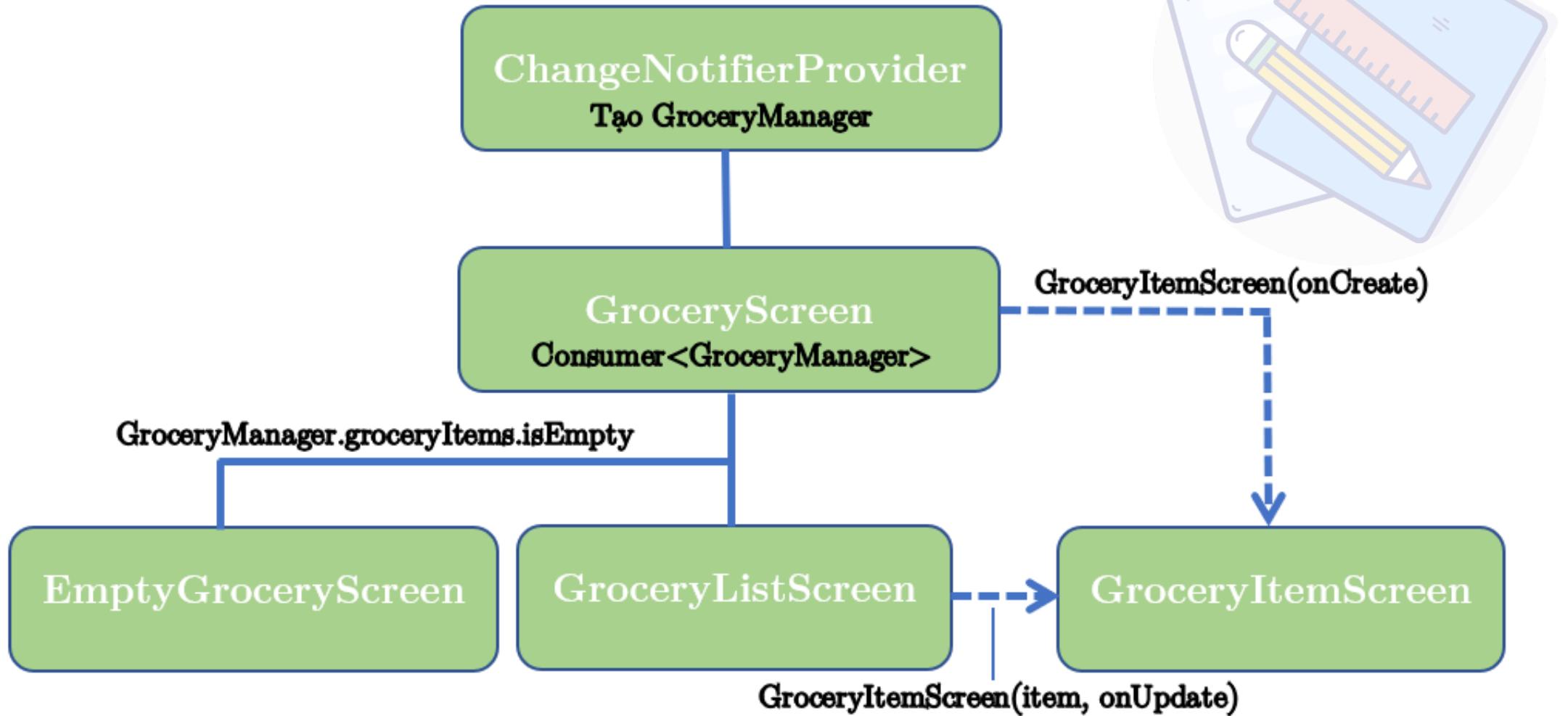


Xóa item khi vuốt (swipe)

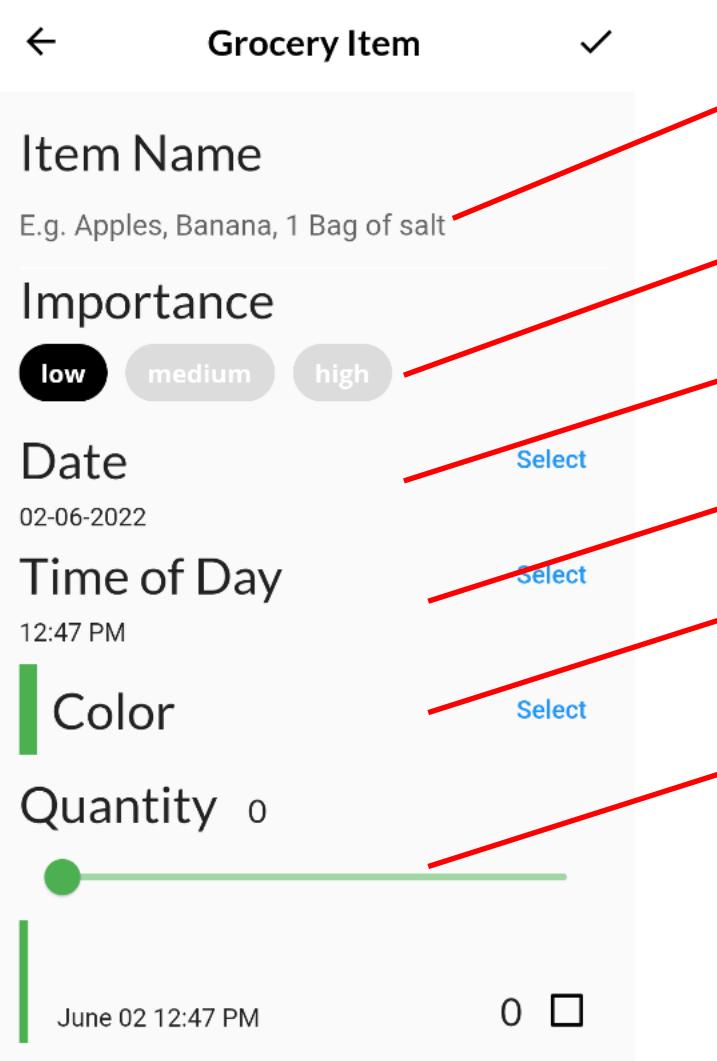
```
child: ListView.separated(  
    itemBuilder: (context, index) {  
        final item = groceryItems[index];  
        return Dismissible(  
            key: Key(item.id),  
            direction: DismissDirection.endToStart,  
            background: Container(  
                color: Colors.red,  
                alignment: Alignment.centerRight,  
                child: const Icon(Icons.delete_forever,  
                    color: Colors.white, size: 50), // Icon  
            ), // Container  
            onDismissed: (direction) {  
                manager.deleteItem(index);  
                ScaffoldMessenger.of(context).showSnackBar(SnackBar(  
                    content: Text('${item.name} dismissed'),  
                )); // SnackBar  
            },  
            child: InkWell(  
                child: GroceryTile(  
                    name: item.name,  
                    quantity: item.quantity,  
                    rating: item.rating,  
                    id: item.id  
                )  
            )  
        );  
    }  
)
```



# Ứng dụng Danh sách mua hàng



# GroceryItemScreen



The screenshot shows a mobile application screen titled "Grocery Item". The screen has a back arrow, a title bar with "Grocery Item" and a checkmark icon, and a floating action button with a plus sign. Below the title are six input fields:

- Item Name:** "E.g. Apples, Banana, 1 Bag of salt" with a placeholder "Item Name".
- Importance:** A radio button group with "low" (selected), "medium", and "high" options.
- Date:** "02-06-2022" with a "Select" button.
- Time of Day:** "12:47 PM" with a "Select" button.
- Color:** A color swatch with a green square and a "Select" button.
- Quantity:** "0" with a slider and a text input field containing "0".

Red arrows point from each of the six input fields to the corresponding code snippets in the code editor on the right.

```
Widget buildNameField() { ... }

Widget buildImportanceField() { ... }

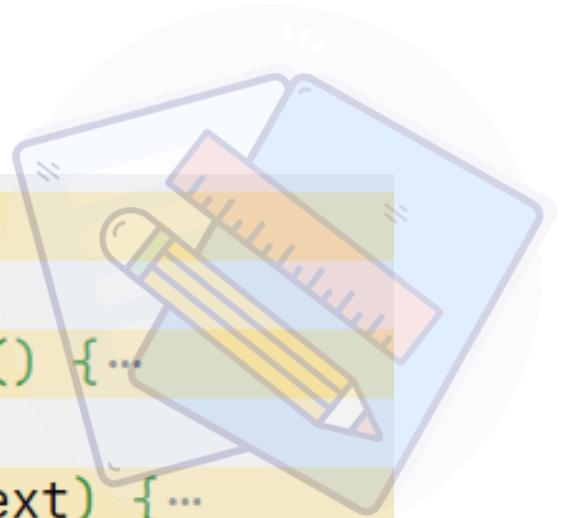
Widget buildDateField(BuildContext context) { ... }

Widget buildTimeField(BuildContext context) { ... }

Widget buildColorPicker(BuildContext context) { ... }

Widget buildQuantityField() { ... }

@override
Widget build(BuildContext context) { ... }
```



# GroceryItemScreen

←      **Grocery Item**      ✓

**Item Name**  
E.g. Apples, Banana, 1 Bag of salt

**Importance**  
**low**   medium   high

**Date**  
02-06-2022      Select

**Time of Day**  
12:47 PM      Select

**Color**      Select

**Quantity** 0  


June 02 12:47 PM      0 □

```
class GroceryItemScreen extends StatefulWidget {
  const GroceryItemScreen({
    Key? key,
    required this.onCreate,
    required this.onUpdate,
    this.originalItem,
  }) : isUpdating = (originalItem != null),
        || super(key: key);

  final Function(GroceryItem) onCreate;
  final Function(GroceryItem) onUpdate;
  final GroceryItem? originalItem;
  final bool isUpdating;

  class _GroceryItemScreenState extends State<GroceryItemScreen> {
    // local state
    final _nameController = TextEditingController();
    String _name = '';
    Importance _importance = Importance.low;
    DateTime _dueDate = DateTime.now();
    TimeOfDay _timeOfDay = TimeOfDay.now();
    Color _currentColor = Colors.green;
    int _currentSliderValue = 0;
```



# GroceryItemScreen

- Khởi tạo, hủy các trạng thái cục bộ trong `initState`, `dispose`

```
@override  
void initState() {  
    final originalItem = widget.originalItem;  
    if (originalItem != null) {  
        _nameController.text = originalItem.name;  
        _name = originalItem.name;  
        _currentSliderValue = originalItem.quantity;  
        _importance = originalItem.importance;  
        _currentColor = originalItem.color;  
        final date = originalItem.date;  
        _timeOfDay = TimeOfDay(hour: date.hour, minute: date.minute);  
        _dueDate = date;  
    }  
    _nameController.addListener(() {  
        setState(() {  
            _name = _nameController.text;  
        });  
    });  
  
    super.initState();  
}
```

```
@override  
void dispose() {  
    _nameController.dispose();  
    super.dispose();  
}
```



# GroceryItemScreen

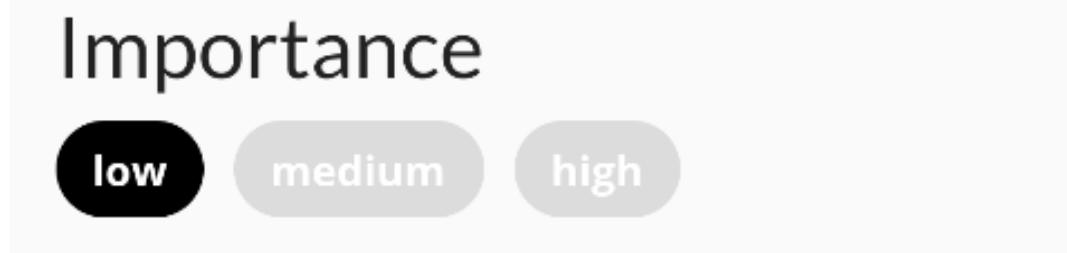
Item Name

E.g. Apples, Banana, 1 Bag of salt

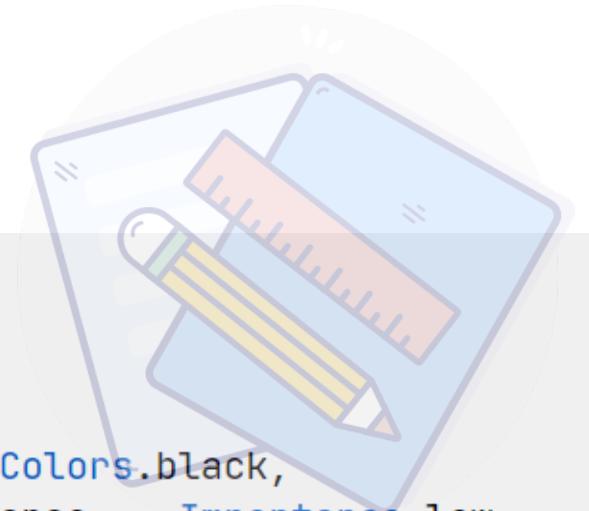
```
TextField(  
  controller: _nameController,  
  cursorColor: _currentColor,  
  decoration: InputDecoration(  
    hintText: 'E.g. Apples, Banana, 1 Bag of salt',  
    enabledBorder: const UnderlineInputBorder(  
      borderSide: BorderSide(color: Colors.white),  
    ), // UnderlineInputBorder  
    focusedBorder: UnderlineInputBorder(  
      borderSide: BorderSide(color: _currentColor),  
    ), // UnderlineInputBorder  
    border: UnderlineInputBorder(  
      borderSide: BorderSide(color: _currentColor),  
    ), // UnderlineInputBorder  
  ), // InputDecoration  
, // TextField
```



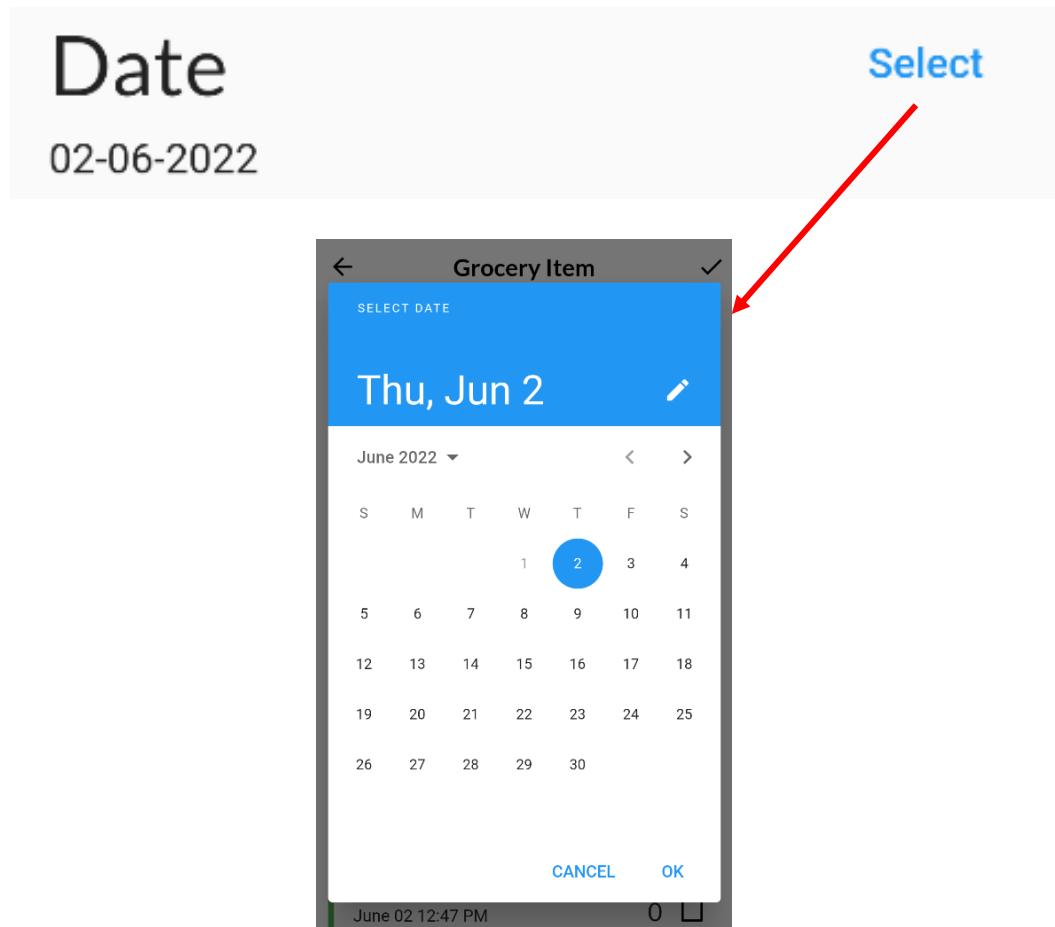
# GroceryItemScreen



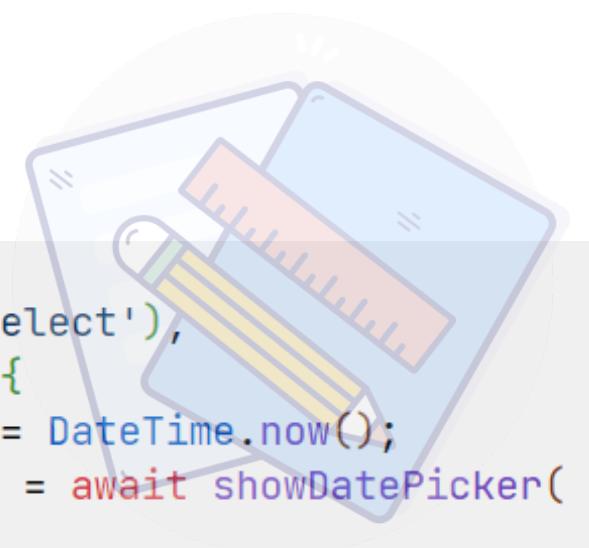
```
Wrap(
  spacing: 10,
  children: [
    ChoiceChip(
      selectedColor: Colors.black,
      selected: _importance == Importance.low,
      label: const Text(
        'low',
        style: TextStyle(
          color: Colors.white,
        ), // TextStyle
      ), // Text
      onSelected: (selected) {
        setState(() => _importance = Importance.low);
      },
    ), // ChoiceChip
    ChoiceChip(
```



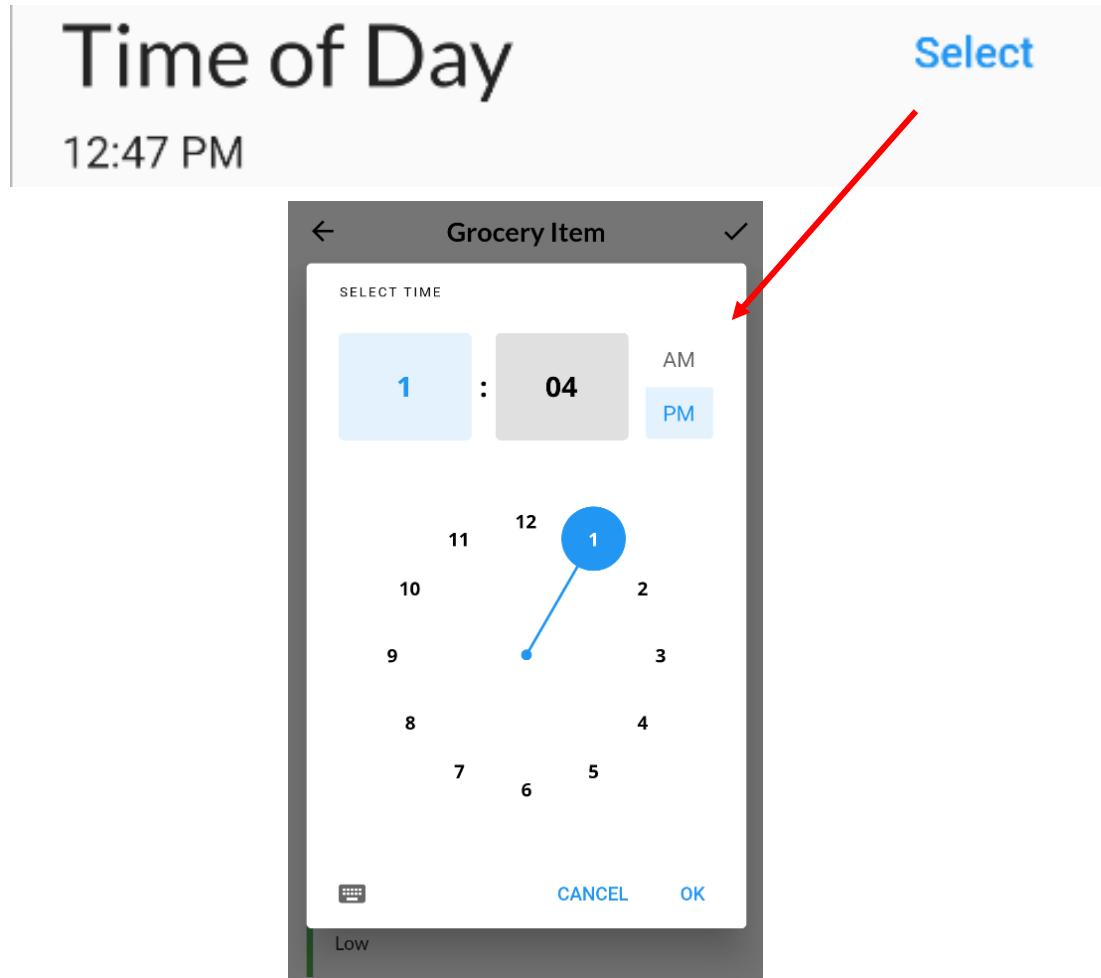
# GroceryItemScreen



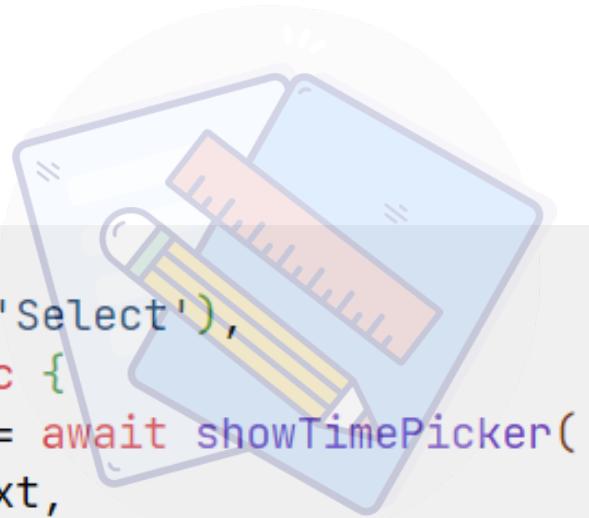
```
TextButton(  
    child: const Text('Select'),  
    onPressed: () async {  
        final currentDate = DateTime.now();  
        final selectedDate = await showDatePicker(  
            context: context,  
            initialDate: currentDate,  
            firstDate: currentDate,  
            lastDate: DateTime(currentDate.year + 5),  
        );  
        setState(() {  
            if (selectedDate != null) {  
                _dueDate = selectedDate;  
            }  
        });  
    },  
) // TextButton
```



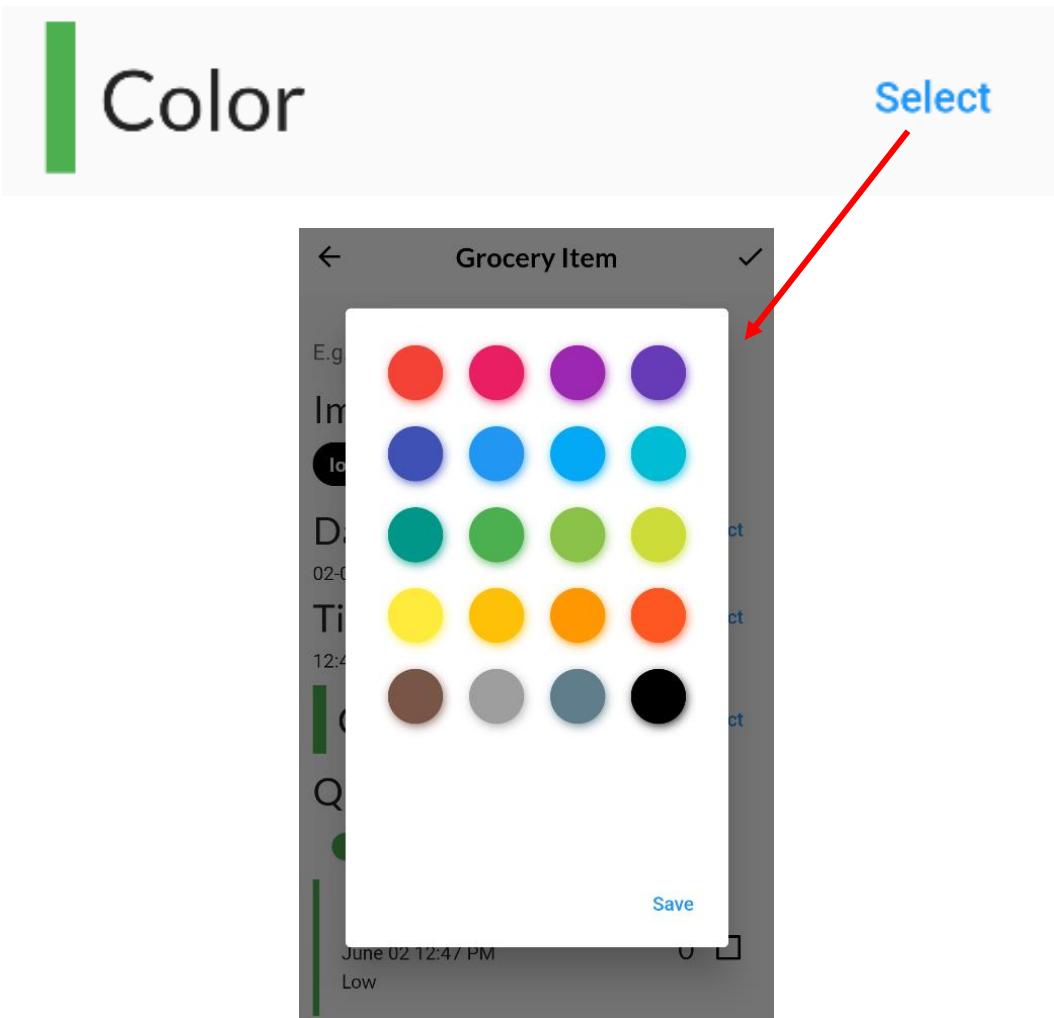
# GroceryItemScreen



```
TextButton(  
    child: const Text('Select'),  
    onPressed: () async {  
        final timeOfDay = await showTimePicker(  
            context: context,  
            initialTime: TimeOfDay.now(),  
        );  
        setState(  
            () {  
                if (timeOfDay != null) {  
                    _timeOfDay = timeOfDay;  
                }  
            },  
        );  
    },  
) // TextButton
```

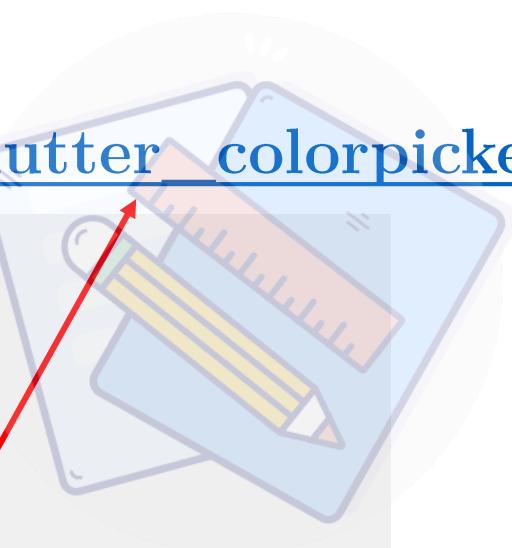


# GroceryItemScreen



Cần khai báo gói [flutter\\_colorpicker](#)

```
TextButton(  
    child: const Text('Select'),  
    onPressed: () {  
        showDialog(  
            context: context,  
            builder: (context) {  
                return AlertDialog(  
                    content: BlocPicker(  
                        pickerColor: Colors.white,  
                        onColorChanged: (color) {  
                            setState(() => _currentColor = color);  
                        },  
                    ), // BlocPicker  
                    actions: [  
                        TextButton(  
                            child: const Text('Save'),  
                            onPressed: () {  
                                Navigator.pop(context);  
                            }, // TextButton  
                        ), // AlertDialog  
                    ],  
                );  
            }  
        );  
    }  
);
```

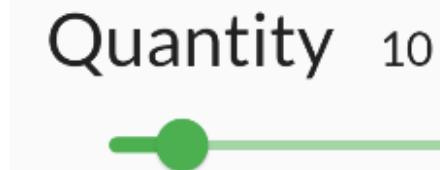


# showDialog

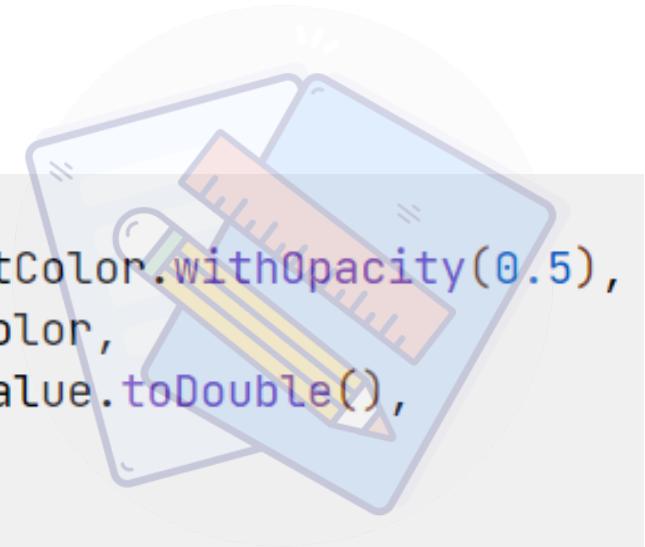


- Hiển thị một hội thoại (dialog) phía trên nội dung hiện thời
- [showDialog](#)(context, builder)
  - builder: hàm tạo một widget Dialog
  - Hai dạng widget Dialog phổ biến: [AlertDialog](#) (thông tin + xác nhận) và [SimpleDialog](#) (danh sách các tùy chọn)
- Các thuộc tính quan trọng của một **AlertDialog**
  - **title**: tựa đề hội thoại
  - **content**: nội dung chính hội thoại
  - **actions**: dòng các nút tương tác với hội thoại

# GroceryItemScreen

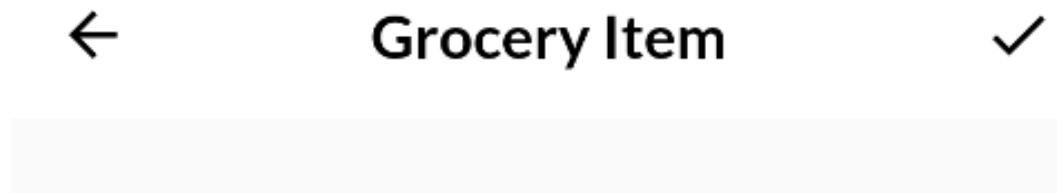


```
Slider(  
    inactiveColor: _currentColor.withOpacity(0.5),  
    activeColor: _currentColor,  
    value: _currentSliderValue.toDouble(),  
    min: 0,  
    max: 100,  
    divisions: 100,  
    label: _currentSliderValue.toInt().toString(),  
    onChanged: (double value) {  
        setState(  
            () {  
                _currentSliderValue = value.toInt();  
            },  
        );  
    },  
) // Slider
```



# GroceryItemScreen

- Xử lý lưu các thông tin nhập liệu: tạo đối tượng groceryItem từ các thông tin nhập liệu và gọi callback tương ứng

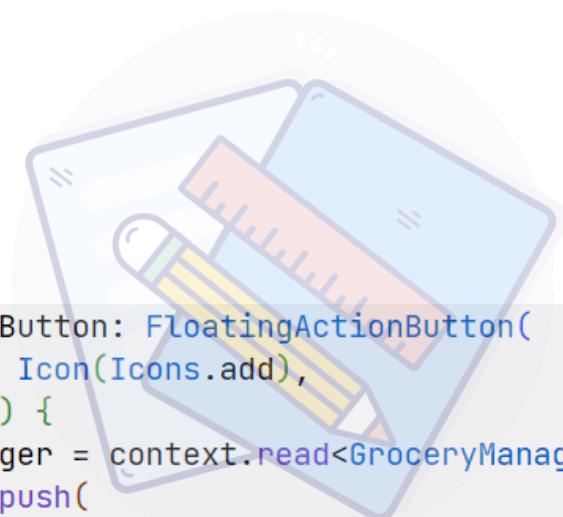
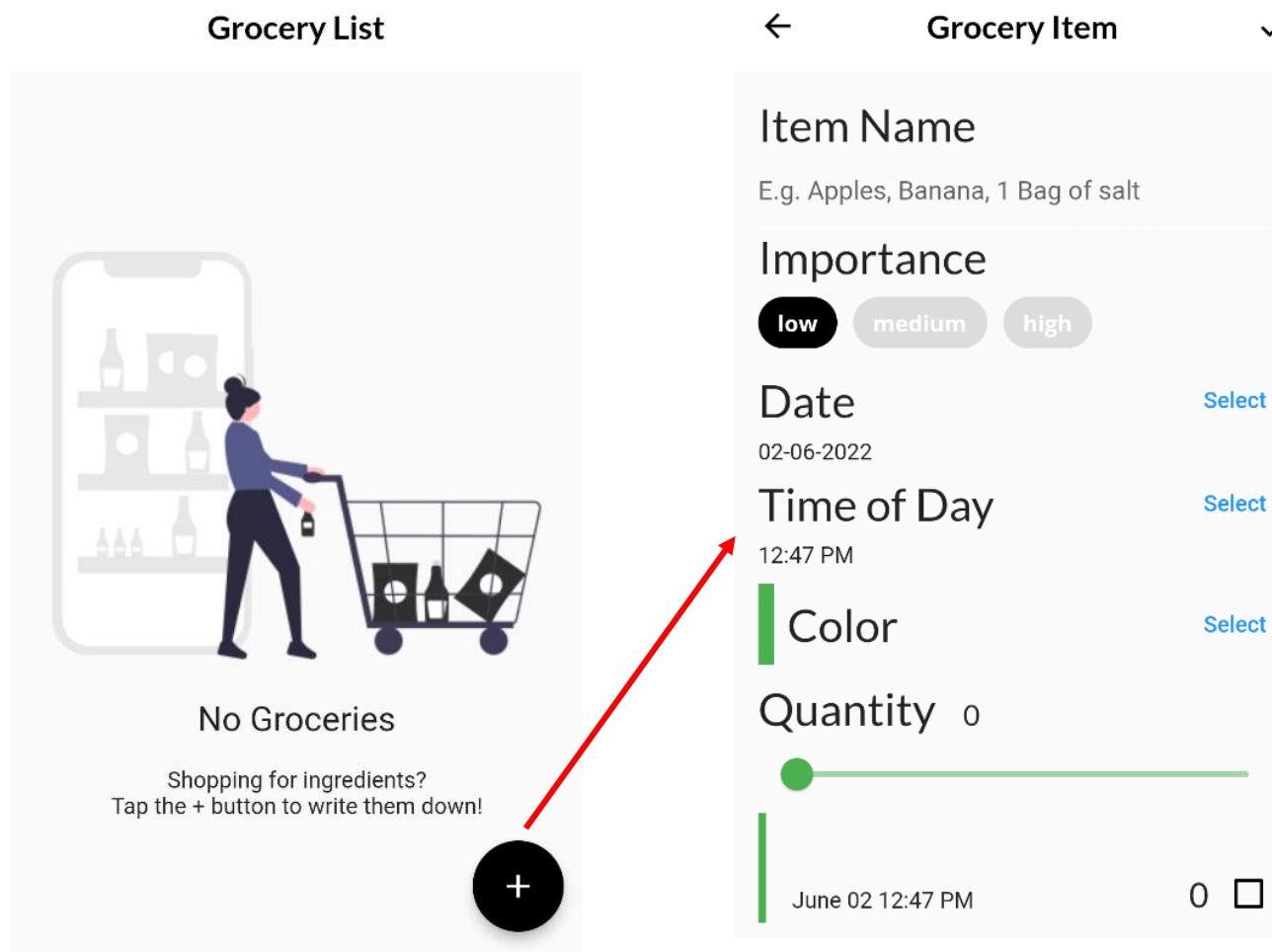


```
appBar: AppBar(
  actions: [
    IconButton(
      onPressed: () {
        final groceryItem = GroceryItem(
          id: widget.originalItem?.id ?? const Uuid().v1(),
          name: _nameController.text,
          importance: _importance,
          color: _currentColor,
          quantity: _currentSliderValue,
          date: DateTime(
            _dueDate.year, _dueDate.month, _dueDate.day,
            _timeOfDay.hour, _timeOfDay.minute,
          ), // DateTime
        ); // GroceryItem

        if (widget.isUpdating) {
          widget.onUpdate(groceryItem);
        } else {
          widget.onCreate(groceryItem);
        }
      },
      icon: const Icon(Icons.check),
    ), // IconButton
  ],
)
```

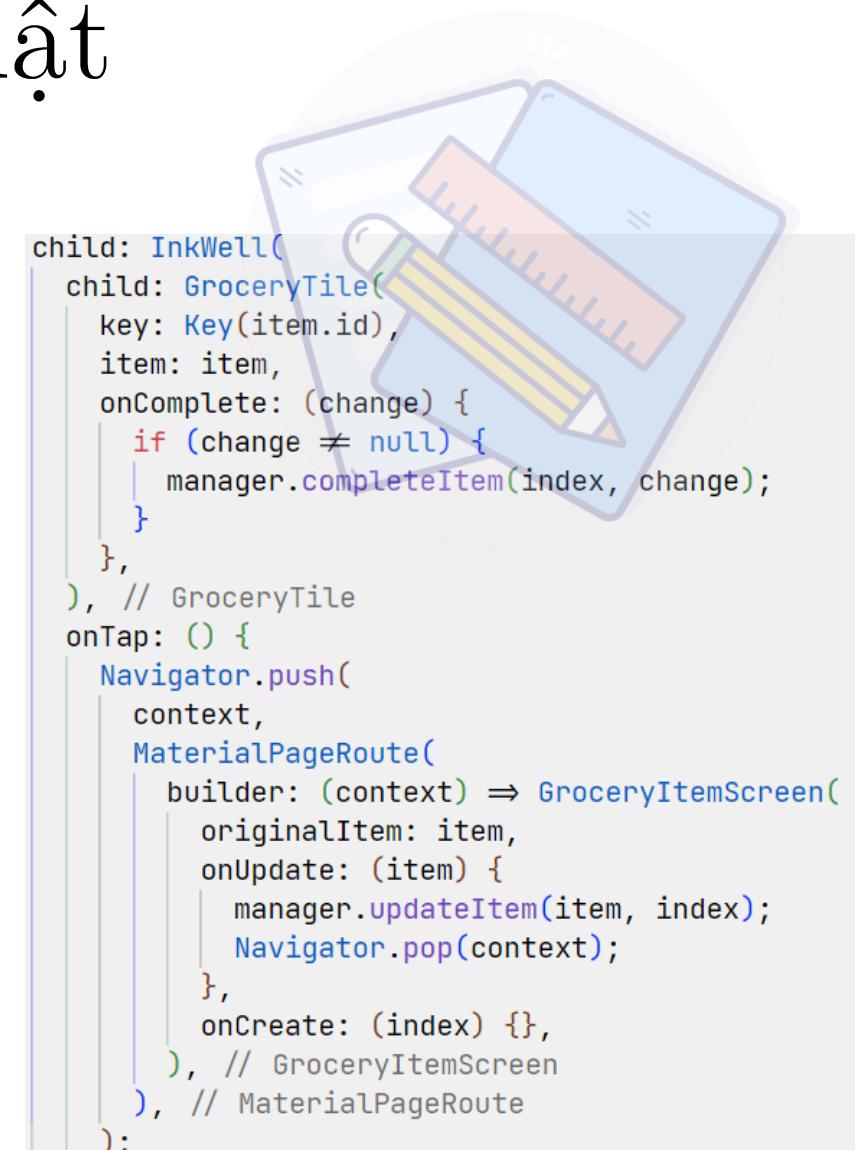
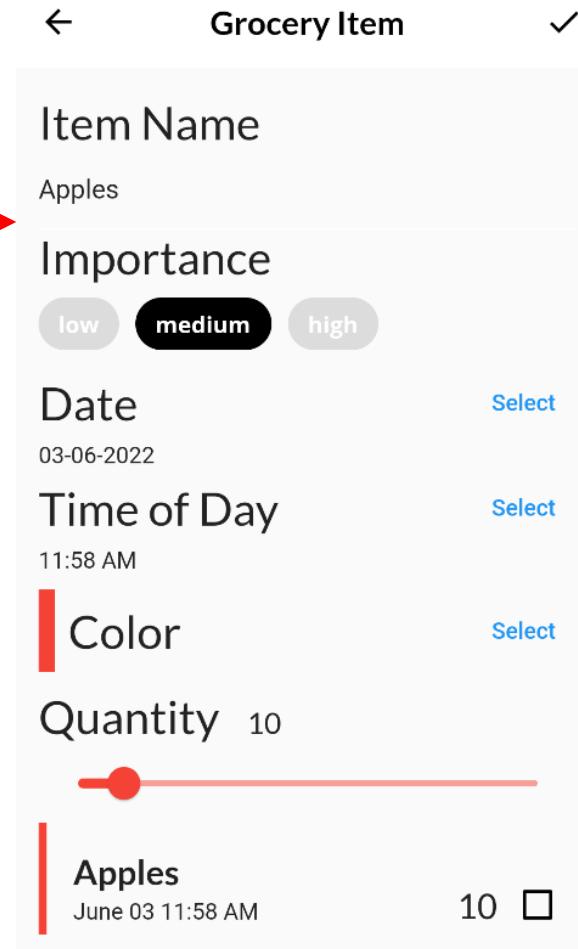
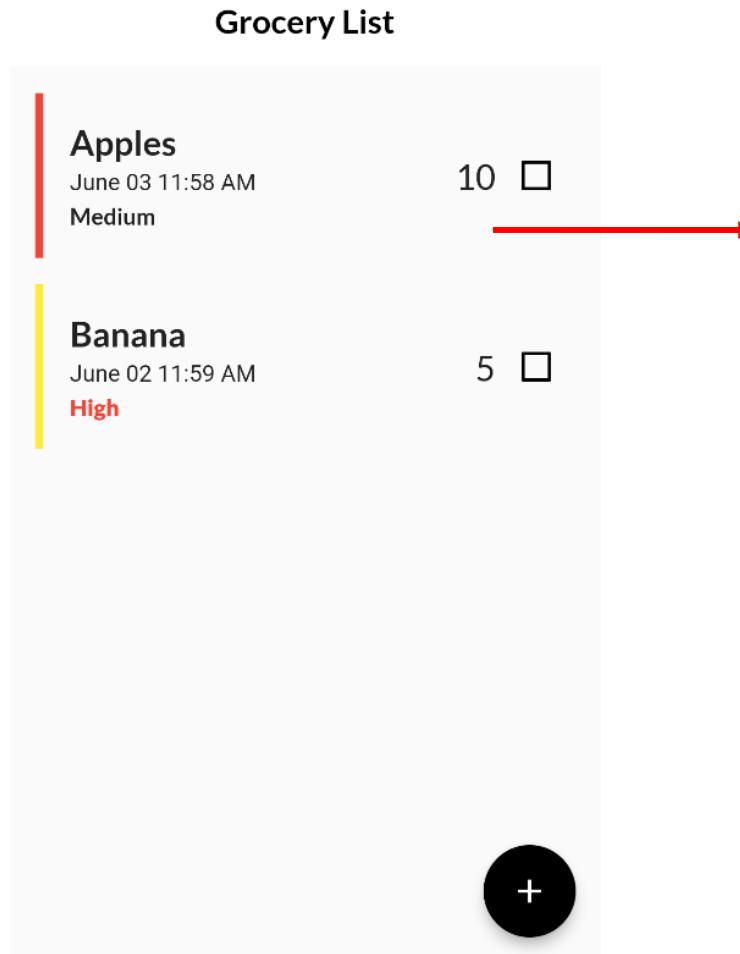


# GroceryItemScreen: Tạo mới



```
floatingActionButton: FloatingActionButton(
  child: const Icon(Icons.add),
  onPressed: () {
    final manager = context.read<GroceryManager>();
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => GroceryItemScreen(
          onCreate: (item) {
            manager.addItem(item);
            Navigator.pop(context);
          },
          onUpdate: (item) {},
        ),
      ),
    );
  },
);
```

# GroceryItemScreen: Cập nhật



# Câu hỏi?

