**ScopedModel trong Flutter**

Trong bài trước chúng ta đã tìm hiểu cách quản lý trạng thái của widget bằng **SatefullWidget**, trong bài hôm nay chúng ta tìm hiểu cách quản lý trạng thái trong ứng dụng bằng **ScopedModel**

Flutter cung cấp một phương pháp đơn giản để quản lý trạng thái của ứng dụng sử dụng *scoped\_model* package. Flutter package đơn giản là một thư viện với những phương thức được sử dụng nhiều lần. Chúng ta sẽ tìm hiểu kỹ về package trong  Flutter ở các bài học tới.

**scoped\_model** cung cấp 3 class chính cho phép quản lý trạng thái của ứng dụng một cách mạnh mẽ:sdf

Model

Model đóng gói trạng thái của một ứng dụng. Chúng ta có thể sử dụng nhiều Model (bằng việc kế thừa Model class) để quản lý trạng thái của ứng dụng. Model có một phương thức duy nhất là *notifyListeners*, nó được gọi bất cứ khi nào trạng thái của Model thay đổi. *notifyListeners* sẽ thực hiện các công việc cần thiết để cập nhật giao diện.

Ví dụ ta tạo một model là Product

class Product extends Model {

final String name;

final String description;

final int price;

final String image;

int rating;

Product(this.name, this.description, this.price, this.image, this.rating);

factory Product.fromMap(Map<String, dynamic> json) {

return Product(

json['name'],

json['description'],

json['price'],

json['image'],

json['rating'],

);

}

void updateRating(int myRating) {

rating = myRating; notifyListeners();

}

}

ScopedModel

ScopedModel là một widget, chúng ta hiểu đơn giản nó là một tiện ích để chúng ta có thể dễ dàng chuyển Data Model từ widget cha xuống các widget con, cháu của nó. Ngoài ra nó còn có nhiệm vụ rebuild lại các widget con giữ các model mà trong trường hợp model này được cập nhật. Nếu cần nhiều hơn một Data Model thì chúng ta có thể sử dụng lồng ScopeModel. Dưới đây là hai dạng ScopedModel :

Single model :

ScopedModel<Product>(

model: item, child: AnyWidget()

)

Multiple model

ScopedModel<Product>(

model: item1,

child: ScopedModel<Product>(

model: item2, child: AnyWidget(),

),

)

ScopeModel.of là một phương thức dùng để lấy Data Model dưới ScopeModel. Và nó có thể được sử dụng khi Data Model thay đổi kể cá khi giao diện (UI) không thay đổi. Dưới đây là ví dụ khi ta thay đổi UI( đánh giá ) của một sản phẩm

ScopedModel.of<Product>(context).updateRating(2);

ScopedModelDescendant

ScopedModelDescendant là một widget, nó lấy Data Model từ lớp cha và build lại UI bất kí khi nào Data Model thay đổi.

ScopedModelDescendant có 2 thuộc tính là builder và child. Child là phần UI không bị thay đổi và sẽ được chuyển cho hàm builder  . Hàm buider sẽ  nhận 3 đối số:

1. Content :ScopedModelDescendant chuyển sang context của ứng dụng
2. Child : Một phần của UI và không thay đổi dựa trên Data Model
3. Model : Dưới đây là ví dụ tường minh

return ScopedModelDescendant<ProductModel>(

builder: (context, child, cart) => { ... Actual UI ... },

child: PartOfTheUI(),

);

Bây giờ chúng ta sẽ sử dụng các ví dụ từ các bài học trước và sử dụng ScopeModel thay vì StatefulWidget :

1. Tạo một ứng dụng Flutter mới với tên  project tùy ý bạn, ở đây tôi sẽ sử dụng tên là product\_scoped\_model\_app
2. Sau đó thay thế các dòng code mặc định trong hàm main.dart bằng product\_state\_app code nhé
3. Coppy các hình ảnh từ assets mà đã sử dụng trong các bài trước, tôi sẽ để link lại đây nhé ( <https://vncoder.vn/bai-hoc/layout-trong-flutter-225> ) sau đó vào pubspec.yaml file tìm đến mục assets và dán vào

flutter:

assets:

- assets/floppy.jpg

- assets/iphone.jpg

- assets/laptop.jpg

- assets/pendrive.jpg

- assets/pixel.jpg

- assets/tablet.jpg

Bây giờ chúng ta phải sử dụng gói thư viện của bên thứ ba vì nó không có trong Framework của flutter

Bằng cách thêm Scope\_model vào*pubspec.yaml* ởphần *dependencies*

dependencies: scoped\_model: ^1.0.1

Oke, bạn nên sử dụng version mới nhất nhé, cập nhật tại đây : <https://pub.dev/packages/scoped_model>

Bây giờ bạn hãy thay thế đoạn code mặc định(main.dart) bằng đoạn code của chúng tôi nhé

import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {

// This widget is the root of your application.

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Flutter Demo',

theme: ThemeData(primarySwatch: Colors.blue,),

home: MyHomePage(title: 'Product state demo home page'),

);

}

}

class MyHomePage extends StatelessWidget {

MyHomePage({Key key, this.title}) : super(key: key);

final String title;

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text(this.title),

),

body: Center(

child: Text( 'Hello World', )

),

);

}

}

Bạn nhớ import Scope\_model vào main.dart nhé ^^

import 'package:scoped\_model/scoped\_model.dart';

Giờ chúng ta sẽ tạo lớp Product quen thuộc, Product.dart là lớp chứa thông tin của sản phẩm Product

import 'package:scoped\_model/scoped\_model.dart';

class Product extends Model {

final String name;

final String description;

final int price;

final String image;

int rating;

Product(this.name, this.description, this.price, this.image, this.rating);

factory Product.fromMap(Map<String, dynamic> json) {

return Product(

json['name'],

json['description'],

json['price'],

json['image'],

json['rating'],

);

}

void updateRating(int myRating) {

rating = myRating;

notifyListeners();

}

}

Tốt rồi, giờ chúng ta sẽ sử dụng thuộc tính notifyListeners để lắng nghe sự thay đổi của UI khi mà người dùng đánh giá

Chúng ta sẽ viết phương thức getProduct để tạo ra nội dung cho lớp Product

import product.dart in main.dart

import 'Product.dart';

static List<Product> getProducts() {

List<Product> items = <Product>[];

items.add(

Product(

"Pixel",

"Pixel is the most feature-full phone ever", 800,

"pixel.jpg", 0

)

);

items.add(

Product(

"Laptop", "Laptop is most productive development tool", 2000,

"laptop.jpg", 0

)

);

items.add(

Product(

"Tablet",

"Tablet is the most useful device ever for meeting", 1500,

"tablet.jpg", 0

)

);

items.add(

Product(

"Pendrive",

"Pendrive is useful storage medium",

100, "pendrive.jpg", 0

)

);

items.add(

Product(

"Floppy Drive",

"Floppy drive is useful rescue storage medium", 20,

"floppy.jpg", 0

)

);

return items;

}

OK, tiếp tục chúng ta sẽ tạo một widget mới có tên là RatingBox và sử dụng Scope\_model để support nhé

class RatingBox extends StatelessWidget {

RatingBox({Key key, this.item}) : super(key: key);

final Product item;

Widget build(BuildContext context) {

double \_size = 20;

print(item.rating);

return Row(

mainAxisAlignment: MainAxisAlignment.end,

crossAxisAlignment: CrossAxisAlignment.end,

mainAxisSize: MainAxisSize.max,

children: <Widget>[

Container(

padding: EdgeInsets.all(0),

child: IconButton(

icon: (

item.rating >= 1

? Icon( Icons.star, size: \_size, )

: Icon( Icons.star\_border, size: \_size, )

), color: Colors.red[500],

onPressed: () => this.item.updateRating(1),

iconSize: \_size,

),

),

Container(

padding: EdgeInsets.all(0),

child: IconButton(

icon: (item.rating >= 2

? Icon(

Icons.star,

size: \_size,

) : Icon(

Icons.star\_border,

size: \_size,

)

),

color: Colors.red[500],

onPressed: () => this.item.updateRating(2),

iconSize: \_size,

),

),

Container(

padding: EdgeInsets.all(0),

child: IconButton(

icon: (

item.rating >= 3? Icon(

Icons.star,

size: \_size,

)

: Icon(

Icons.star\_border,

size: \_size,

)

),

color: Colors.red[500],

onPressed: () => this.item.updateRating(3),

iconSize: \_size,

),

),

],

);

}

}

ở đây chúng ta sử dụng StatelessWidget thay vì StatefulWidget.Cũng như vậy, chúng ta đã sử dụng phương thức Product model’s updateRating để set giá trị phần đánh giá

tiếp theo chúng ta sẻ điều chỉnh widget ProductBox để làm việc với lớp Product, ScopedModel và ScopedModelDescendant

class ProductBox extends StatelessWidget {

ProductBox({Key key, this.item}) : super(key: key);

final Product item;

Widget build(BuildContext context) {

return Container(

padding: EdgeInsets.all(2),

height: 140,

child: Card(

child: Row(

mainAxisAlignment: MainAxisAlignment.spaceEvenly,

children: <Widget>[

Image.asset("assets/" + this.item.image),

Expanded(

child: Container(

padding: EdgeInsets.all(5),

child: ScopedModel<Product>(

model: this.item,

child: Column(

mainAxisAlignment: MainAxisAlignment.spaceEvenly,

children: <Widget>[

Text(this.item.name,

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

Text(this.item.description),

Text("Price: " +

this.item.price.toString()),

ScopedModelDescendant<Product>(

builder: (context, child, item)

{ return RatingBox(item: item); }

)

],

)

)

)

)

]

),

)

);

}

}

Các bạn có thể thấy chúng ta đã đóng gói widget RatingBox bên trong ScopedModel và ScopedModelDecendant

Bây giờ chúng ta sẽ thay đổi widget MyHomePage để sử dụng widget ProductBox như sau

class MyHomePage extends StatelessWidget {

MyHomePage({Key key, this.title}) : super(key: key);

final String title;

final items = Product.getProducts();

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text("Product Navigation")),

body: ListView.builder(

itemCount: items.length,

itemBuilder: (context, index) {

return ProductBox(item: items[index]);

},

)

);

}

}

Ở đây chúng ta đã sử dụng ListView.builder để xây dựng động danh sách Product

Vậy là xong, dưới đây là toàn bộ code được sử dụng :

Product.dart

import 'package:scoped\_model/scoped\_model.dart';

class Product extends Model {

final String name;

final String description;

final int price;

final String image;

int rating;

Product(this.name, this.description, this.price, this.image, this.rating);

factory Product.fromMap(Map<String, dynamic> json) {

return Product(

json['name'],

json['description'],

json['price'],

json['image'],

json['rating'],

);n

} void cn "Laptop is most productive development tool", 2000, "laptop.jpg", 0));

items.add(

Product(

"Tablet"cnvn,

"Tablet is the most useful device ever for meeting", 1500,

"tablet.jpg", 0

)

);

items.add(

Product(

"Pendrive",

"Pendrive is useful storage medium", 100,

"pendrive.jpg", 0

)

);

items.add(

Product(

"Floppy Drive",

"Floppy drive is useful rescue storage medium", 20,

"floppy.jpg", 0

)

)

; return items;

}

main.dart

import 'package:flutter/material.dart';

import 'package:scoped\_model/scoped\_model.dart';

import 'Product.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {

// This widget is the root of your application

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Flutter Demo',

theme: ThemeData(

primarySwatch: Colors.blue,

),

home: MyHomePage(title: 'Product state demo home page'),

);

}

}

class MyHomePage extends StatelessWidget {

MyHomePage({Key key, this.title}) : super(key: key);

final String title;

final items = Product.getProducts();

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text("Product Navigation")),

body: ListView.builder(

itemCount: items.length,

itemBuilder: (context, index) {

return ProductBox(item: items[index]);

},

)

);

}

}

class RatingBox extends StatelessWidget {

RatingBox({Key key, this.item}) : super(key: key);

final Product item;

Widget build(BuildContext context) {

double \_size = 20;

print(item.rating);

return Row(

mainAxisAlignment: MainAxisAlignment.end,

crossAxisAlignment: CrossAxisAlignment.end,

mainAxisSize: MainAxisSize.max,

children: <Widget>[

Container(

padding: EdgeInsets.all(0),

child: IconButton(

icon: (

item.rating >= 1? Icon( Icons.star, size: \_size, )

: Icon( Icons.star\_border, size: \_size, )

),

color: Colors.red[500],

onPressed: () => this.item.updateRating(1),

iconSize: \_size,

),

),

Container(

padding: EdgeInsets.all(0),

child: IconButton(

icon: (item.rating >= 2

? Icon(

Icons.star,

size: \_size,

)

: Icon(

Icons.star\_border,

size: \_size,

)

),

color: Colors.red[500],

onPressed: () => this.item.updateRating(2),

iconSize: \_size,

),

),

Container(

padding: EdgeInsets.all(0),

child: IconButton(

icon: (

item.rating >= 3 ?

Icon( Icons.star, size: \_size, )

: Icon( Icons.star\_border, size: \_size, )

),

color: Colors.red[500],

onPressed: () => this.item.updateRating(3),

iconSize: \_size,

),

),

],

);

}

}

class ProductBox extends StatelessWidget {

ProductBox({Key key, this.item}) : super(key: key);

final Product item;

Widget build(BuildContext context) {

return Container(

padding: EdgeInsets.all(2),

height: 140,

child: Card(

child: Row(

mainAxisAlignment: MainAxisAlignment.spaceEvenly,

children: <Widget>[

Image.asset("assets/" + this.item.image),

Expanded(

child: Container(

padding: EdgeInsets.all(5),

child: ScopedModel<Product>(

model: this.item, child: Column(

mainAxisAlignment: MainAxisAlignment.spaceEvenly,

children: <Widget>[

Text(

this.item.name, style: TextStyle(

fontWeight: FontWeight.bold

)

),

Text(this.item.description),

Text("Price: " + this.item.price.toString()),

ScopedModelDescendant<Product>(

builder: (context, child, item) {

return RatingBox(item: item);

}

)

],

)

)

)

)

]

),

)

);

}

}

Các bạn hãy thử chạy và xem kết quả. Nó cơ bản hoạt động giống ví dụ trước nhưng ở đây có sử dụng thêm khái niệm  Scope\_model.

*Scope\_model - chúng ta có thể hiểu đơn giản nó là một framework , ví dụ khi bạn code "thô", lúc data model thay đổi, ở hàm setState() thay vì rebuild lại tất cả các widget (Btn, txt, ..) thì sẽ phạm phải quy tắc Single Responsibility thì Scope\_model được sinh ra với mục đích chỉ rebuild data model bị thay đỏi thay vì rebuild tất cả widget*