Bài 1: widget, statelesswidget

Widget – là thành phần giao diện cơ bản (thẻ)

Có thể viết các widget tự mình tao ra nhưng phải kế thừa từ lớp statefulwidget / statelesswidgetState: thuộc tính bên trong cái thẻ

Statelesswidget: Giống private – gắn với giao diện, mỗi khi thay đổi thì phải gọi lại hàm build

Bài 2: Statefulwidget và textfield

* Một statefulwidget được tạo thành từ hai class sau:

+ Một class public được kế thừa statefulwidget

+ Một class private để vẽ giao diện (qua hàm build)

State: thay đổi nội bộ trong thẻ, khi nó thay đổi thì hàm build reload lại giao diện

* Ở trong hàm build sẽ có những hàm thay đổi State và sự thay đổi đó đến từ class public
* Muốn truy cập thuộc tính ở class public thì ta dùng cú pháp: this.widget.tên thuộc tính.
* Thẻ Column (cột): có thành phần children (mảng) có thể chứa nhiều thành phần khác

+

mainAxisAlignment

Theo chiều nào của column

crossAxisAlignment

Theo chiều hang

State trong hàm build: cho người dùng thay đổi hoặc thêm 1 thuộc tính nào đó bên trong thẻ textfield

* TextField: hiện thông tin ra cho chỉnh sửa. Nên gói lại trong một Container vì trong đó có chứa thuộc tính padding (có thuộc tính tính khoảng cách nội dung theo 4 phía giữa các thẻ).

+ Padding “vertical” = padding Top + Bottom

+ Padding “horizontal” = padding Left + Right

-BoderRadius trong thuộc tính border (bo tròn góc lại) vd:  borderRadius: BorderRadius.all(Radius.circular(30)))

- State thay đổi -> Hàm build tự động được gọi -> Giao diện (UI) của app được reload

- Cách lấy thông tin nhập vào ở textfield: dùng TextEdittingController. Tạo một đối tượng TextEdittingController để theo dõi sự thay đổi ở trên thẻ Textfield

- Tiếp đến dùng hàm onChanged mỗi khi dòng text điền vào thay đổi thì ở trong onChanged cũng thay đổi theo

- Synstax: this.setState((){Thay đổi giá trị state tại đây.})

-Phương thức setState () thường được gọi từ Flutter framwork và từ phía developer.

-Nó được sử dụng để thông báo rằng "dữ liệu đã thay đổi" và widget ở build context này nên được rebuilt.

08 WidgetsBindingObserver, vòng đời(lifecycle) 1 Widget

* Cú pháp: Class A + with “protocol X” => Class A phải thực thi các phương thức trong protocal
* WidgetBindingObsever chứa các phương thức liên quan đến vòng đời của một thẻ
* Đăng kí Observer cho các phương thức liên quan đến lifecycle trong WidgetsBindingObserver
* Đăng kí Obsever là gì ? là để sử dụng các phương thức liên quan đến lifecycle trong WidgetsBindingObserver

09: thêm package và tuỳ biến ngày tháng DateTime với intl

10: Layout với SafeArea,Column, FlatButton, TextField

Safearea(): tạo một khoảng cách giữa vùng an toàn so với phần trên của điện thoại (vd: phần tai thỏ của iphone)

+ minimum: tạo khoảng cách giữa safearea với điện thoại

* Mỗi thẻ TextField (nên) có một TexEditingController riêng, sau đó gán thuộc tính đó vào controller của mỗi TextField tương ứng

-FlatButton: phiên bản mới dùng TextButton

Onchanged sẽ được gọi khi gõ, xóa ở TextField, để gán text vừa gõ vào thuộc tính sẽ gọi thêm setState trong hàm onChanged

Nơi gọi Scaffold.of(context) phải dưới 1 thẻ Scaffold -> fix gọi đối tượng

GlobalKey<ScaffoldMessengerKey>();

Sau đó muốn hiên thì gọi: \_(tên thuộc tính vừa đặt).ScaffoldState!.showSnackBar(SnackBar)

11:Đưa dữ liệu state lên ListTile, sử dụng hàm map

* Xóa dữ liệu text field sau khi add một đối tượng vào list
* Khởi tạo lại đối tượng thành rỗng

ButtonTheme(height:…, width:…,child: 1 button)

12: Đưa ListTile vào bên trong thẻ Card

* Có 2 loại thẻ:

+ Thẻ có thuộc tính child: một thẻ đấy có một thẻ khác

+ Thẻ có thuộc tính childrent: một thẻ chưa nhiều thẻ khác

* Trong thẻ card có thể tùy biến các thuộc tính: shape, set backgrounds color, elevation (bóng), onTap

13: Tách riêng các Widget và tạo ListView

ListView (children:<Widget>) => nên dùng khi load những cái hữu hạn (vì nó load hết)

ListView(itemCount:…,itemBuilder:…)=> Chỉ load những thứ mà mình nhìn thấy(dùng khi độ dài lớn)

* Khi gọi bàn phím lên thì bàn phím sẽ co làm cho app bị báo lỗi vì bị co màn hình -> sử dụng SingleChildScrollView để có thể cuộn lên khi màn hình bị co

14: Tuỳ biến AppBar và FloatingActionButton

* AppBar là một thuộc tính trong thẻ Scaffold bao gồm ác thuộc tính: tittle, actions[]

15 Tuỳ chỉnh ListView item sử dụng Column và Row

Row: là thẻ con chứa các widget theo dạng từ trái sang phải

* Thẻ expanded: Chiếm phần không gian còn lại của thẻ chứa nó \

16: Hiển thị và tuỳ biến Modal Bottom Sheet

* Giống alert
* This.context: tham chiếu đến widget hiện tại
* Builder: giao diện của Modal Sheet
* Modal Sheet gồm 2 phần chính: input và các button ok, cancel

17:Flutter cơ bản-Thực thi các Actions trong ModalBottomSheet

-Ẩn modalbotoomsheet bằng cách sử dụng Navigator

18-Flutter cơ bản-Tuỳ biến Theme và Fonts

* Theme – kiểu dự liệu ThemeData nằm trong material app
* Primarycolor: màu chủ đạo (vd: AppBar, floatingactionbutton)
* Để truy xuất floatingactionbutton dùng cú pháp: theme.of(context).primaryColor,

20-Ứng dụng FoodApp-Danh sách với GridView và Grid Item

* Gradient: màu thay đổi từ màu A -> màu B

22-Ứng dụng FoodApp-Truyền dữ liệu giữa 2 màn hình dùng hàm Constructor

* InkWell: có thể bấm (tap) như Button
* splashColor: màu nháy lên khi nhấn
* MaterialPageRoute: Build Route from Widget