

BỘ GIÁO DỤC & ĐÀO TẠO
TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN

----



**BÁO CÁO MÔN HỌC
CÁC CÔNG NGHỆ LẬP TRÌNH HIỆN ĐẠI**

**ĐỀ TÀI
TÌM HIỂU FLUTTER**

Nhóm 17

3120410457	Lê Bảo Tài
3120410467	Võ Minh Tân
3120410297	Trần Nguyên Lộc
3120410471	Trịnh Hùng Thái

Giảng viên phụ trách ThS. PHẠM THỊ VƯƠNG

TP.Hồ Chí Minh, tháng 5 năm 2024

MỤC LỤC

LỜI CẢM ƠN	vi
LỜI CAM ĐOAN	vii
DANH MỤC HÌNH ẢNH	viii
LỜI MỞ ĐẦU	1
CHƯƠNG 1. TỔNG QUAN VỀ FLUTTER.....	2
1.1. Flutter là gì	2
1.2. Lịch sử phát triển của Flutter.....	3
1.3. Các tính năng chính của Flutter.....	4
1.4. Kiến trúc của Flutter.....	7
1.5. Ưu điểm và nhược điểm của Flutter.....	11
1.6. Một số thông tin liên quan khác	12
1.6.1. Các nguồn tài liệu khoá học Flutter	12
1.6.1.1. Khoá học trả phí	12
1.6.1.2. Khoá học miễn phí	15
1.6.2. Các thông tin tuyển dụng về Flutter.....	17
1.7. Cài đặt và cấu hình cần thiết cho việc lập trình với Flutter.....	19
1.7.1. Flutter SDK	19
1.7.2. IDE	22
1.7.3. Khởi chạy chương trình đầu tiên.....	27
CHƯƠNG 2. CHI TIẾT VỀ FLUTTER	32
2.1. Những kiến thức cơ bản về Flutter	32
2.1.1. Tổng quan về Widgets và một số Widgets cần phải biết.....	32
2.1.1.1. Khái niệm về Widget trong Flutter	32

2.1.1.2. Các loại Widget con	34
2.1.1.3. Phân loại Widget State	44
2.1.2. Bố cục giao diện (Layout).....	48
2.1.2.1. Khái niệm bố cục và bố trí trong Flutter	48
2.1.2.2. Một số Widget bố cục	53
2.1.2.3. Xây dựng thử một bố cục phức tạp	68
2.1.3. Cử chỉ giao diện (Gestures).....	74
2.1.3.1. Con trỏ.....	74
2.1.3.2. Cử chỉ	75
2.1.3.3. Bắt sự kiện cử chỉ.....	77
2.1.4. Quản lý trạng thái (State)	80
2.1.4.1. Khái niệm	81
2.1.4.2. Trạng thái tức thời (Ephemeral State).....	82
2.1.4.3. Trạng thái ứng dụng (App State)	83
2.1.4.4. Quản lí trạng thái (State) bằng thư viện Provider	84
2.1.4.5. So sánh Quản lý trạng thái bằng cách thông thường và Quản lý trạng thái bằng thư viện Provider	90
2.1.5. Điều hướng màn hình.....	92
2.1.5.1. Điều hướng bằng Routes	92
2.1.5.2. Điều hướng bằng RouteName	96
2.1.5.3. So sánh giữa Route và RouteName	97
2.2. Một số Widgets thông dụng cần phải biết	98
2.2.1. Widget Scaffold	98
2.2.2. Widget Container	101
2.2.3. Widget Row và Column	103
2.2.4. Widget Text.....	104

2.2.5. Widget TextField	106
2.2.6. Widget Button	108
2.2.7. Widget Stack	110
2.2.8. Widget Form	112
2.2.9. Widget Alert Dialogs	113
2.2.10. Widget Icon.....	117
2.2.11. Widget Image	119
2.2.12. Widget Card	122
2.2.13. Widget Tabbar.....	124
2.2.14. Widget Drawer.....	126
2.2.15. Widget ListView	128
2.2.16. Widget GridView	130
2.2.17. Widget CheckBox	131
2.2.18. Widget RadioButtons	133
2.2.19. Widget ProgressBar	134
2.2.20. Widget Snackbar	135
2.2.21. Widget Tooltip	137
2.2.22. Widget Slider	140
2.2.23. Widget Switch.....	143
2.2.24. Widget Charts.....	145
2.2.25. Widget Bottom Navigation Bar	148
2.2.26. Widget Theme.....	150
2.2.27. Widget Table	151
2.2.28. Widget Animation.....	152
CHƯƠNG 3. XÂY DỰNG MỘT SỐ CHƯƠNG TRÌNH FLUTTER CƠ BẢN...154	

3.1. Ứng dụng Máy Tính (Calculator App).....	154
3.1.1. Giới thiệu.....	154
3.1.2. Cấu trúc chương trình	154
3.1.2.1. Phân loại thư mục.....	154
3.1.2.2. Các hàm mặc định.....	156
3.1.2.3. Màn hình hiển thị	158
3.1.3. Demo chương trình	166
3.2. Ứng dụng Ghi Chú (Todo App)	167
3.2.1. Giới thiệu.....	167
3.2.2. Cấu trúc chương trình	168
3.2.3. Demo chương trình	169
3.3. Trò chơi TicTacToe (TicTacToe)	175
3.3.1. Giới thiệu.....	175
3.3.2. Cấu trúc chương trình	175
3.3.2.1. Phân loại thư mục.....	175
3.3.2.2. Các hàm mặc định.....	176
3.3.2.3. Màn hình hiển thị	178
3.3.3. Demo chương trình	187
3.4. Ứng dụng Tính toán chỉ số BMI (BMI Calculator)	188
3.4.1. Giới thiệu.....	188
3.4.2. Cấu trúc chương trình	189
3.4.2.1. Phân loại thư mục.....	189
3.4.2.2. Thiết lập màu sắc và chủ đề	191
3.4.2.3. Thiết lập GetX Controller	194
3.4.2.4. Phân tích giao diện người dùng	196

3.4.3. Demo chương trình	198
KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	201
DANH MỤC THAM KHẢO.....	202

LỜI CẢM ƠN

Để hoàn thành được báo cáo cáo cho môn Lập trình hiện đại này, chúng em xin chân thành cảm ơn thầy ThS.Phạm Thị Vương khoa Công Nghệ Thông Tin trường Đại học Sài Gòn đã tạo cơ hội cho chúng em được học tập, nghiên cứu, hợp tác làm việc nhóm và tích lũy kiến thức để thực hiện và hoàn thành báo cáo đồ án này. Xin cảm ơn thầy đã luôn nhận xét và theo dõi tiến trình trong suốt thời gian làm bài báo cáo.

Do kiến thức của các thành viên còn nhiều hạn chế và thiếu kinh nghiệm nên nội dung báo cáo khó tránh khỏi những thiếu sót. Chúng em rất mong nhận được những lời góp ý thêm đến từ thầy để được hoàn thiện hơn.

LỜI CAM ĐOAN

Nhóm chúng em xin cam đoan đê tài “Tìm hiểu Flutter” là công trình nghiên cứu và tìm hiểu của riêng nhóm chúng em.

Mọi trích dẫn sử dụng trong báo cáo đều được ghi rõ nguồn tài liệu tham khảo theo đúng quy định.

Nhóm xin hoàn toàn chịu trách nhiệm và chịu mọi hình thức kỷ luật theo quy định nếu có bất kì hành vi vi phạm, gian trá nào.

Thành phố Hồ Chí Minh, tháng 5 năm 2024

Nhóm 17

DANH MỤC HÌNH ẢNH

Hình 1. 1. Flutter là gì?	2
Hình 1. 2. Nguồn gốc hình thành và điểm độc đáo của Flutter	4
Hình 1. 3. Các tính năng chính của Flutter	7
Hình 1. 4. Kiến trúc tổng thể của của Flutter.....	9
Hình 1. 5. Mô hình giải phẫu của một ứng dụng Flutter.....	10
Hình 1. 6. Một số khoá học Flutter trả phí trên Udemy	14
Hình 1. 7. Khoá học Flutter trả phí trên CodeGym.....	14
Hình 1. 8. Khoá học lập trình Flutter miễn phí trên Youtube: TinCoder	15
Hình 1. 9. Khoá học lập trình Flutter miễn phí trên Youtube: Dear Programmer	16
Hình 1. 10. Khoá học lập trình Flutter miễn phí trên web: CafeDev.....	16
Hình 1. 11. Một số tuyển dụng lập trình viên Flutter trên trang Topdev.vn.....	17
Hình 1. 12. Một số tuyển dụng lập trình viên Flutter trên trang topcv.vn	18
Hình 1. 13. Tải gói Flutter SDK mới nhất từ trang chủ chính thức của Flutter.....	19
Hình 1. 14. Giải nén và cài đặt gói Flutter SDK tại một đường dẫn bất kỳ.....	20
Hình 1. 15. Truy cập Enviroment Variables từ thanh tìm kiếm.....	20
Hình 1. 16. Chọn Edit Path của User Variables.....	21
Hình 1. 17. Thiết lập đường dẫn tới thư mục bin của Flutter SDK	21
Hình 1. 18. Kiểm tra bằng lệnh Flutter doctor	22
Hình 1. 19. Truy cập vào trang chủ android studio để cài đặt IDE	23
Hình 1. 20. Cài đặt các Plugin cần thiết trên Android Studio.....	23
Hình 1. 21. Kiểm tra Android SDK	24
Hình 1. 22. Kiểm tra Dart SDK.....	24
Hình 1. 23. Án vào nút Create Virtual Device để tạo máy ảo	25

Hình 1. 24. Tuỳ chọn kích cỡ màn hình máy ảo	25
Hình 1. 25. Tuỳ chọn phiên bản Android	26
Hình 1. 26. Các tuỳ chọn còn lại và ấn Finish	26
Hình 1. 27. Tạo mới một dự án Flutter	27
Hình 1. 28. Thiết lập tên dự án, đường dẫn, mô tả, loại dự án và các thiết lập khác cho dự án Flutter đầu tiên.....	27
Hình 1. 29. Những dòng code mặc định của một dự án Flutter.....	28
Hình 1. 30. Khởi tạo máy ảo cho ứng dụng Flutter đầu tiên.....	28
Hình 1. 31. Android Studio biên dịch code và cài đặt ứng dụng Flutter lên máy ảo	30
Hình 1. 32. Máy ảo khởi động chương trình Flutter đầu tiên	31
Hình 1. 33. Android Studio đặt chương trình đang chạy vào trạng thái Hot Reload	31
Hình 2. 1. Một ví dụ về cây Widgets trong Flutter	32
Hình 2. 2. Hình ảnh minh họa cho Widget Text.....	34
Hình 2. 3. Hình ảnh minh họa cho một số Widget Button.....	36
Hình 2. 4. Đường dẫn hình ảnh cấu hình trong pubspec.yaml trong dự án thực tế ..	37
Hình 2. 5. Hình ảnh minh họa cho Widget Image	38
Hình 2. 6. Hình ảnh minh họa cho Widget Icon	39
Hình 2. 7. Hình ảnh minh họa cho Widget Column	41
Hình 2. 8. Hình ảnh minh họa cho Widget Row.....	43
Hình 2. 9. Sắp xếp bố cục trong Flutter	49
Hình 2. 10. Minh họa về cây bố cục trong Flutter	49
Hình 2. 11. Một ví dụ về xây dựng bố cục của một ứng dụng trong thực tế	53
Hình 2. 12. Ví dụ về cách sử dụng Widget FittedBox	59
Hình 2. 13. Ví dụ về cách sử dụng Widget OverflowBox	62
Hình 2. 14. Kết hợp giữa Widget Row và Column.....	65

Hình 2. 15. Một ví dụ minh họa của Widget Stack được sử dụng như thế nào	68
Hình 2. 16. Bố cục của ví dụ ProductBox Widget.....	73
Hình 2. 17. Kết quả chạy thử giả lập trong ví dụ bố cục	73
Hình 2. 18. Hình ảnh minh họa cử chỉ trong Flutter.....	76
Hình 2. 19. Giao diện bắt sự kiện cử chỉ.....	80
Hình 2. 20. Một giải nghĩa cho Quản lý trạng thái trên Flutter	81
Hình 2. 21. Ví dụ minh họa cho trạng thái ứng dụng	84
Hình 2. 22. Demo minh họa cho quản lý trạng thái (State) bằng thư viện Provider	90
Hình 2. 23. Màn hình thứ 1 có nút bấm để di chuyển tới màn hình thứ 2	95
Hình 2. 24. Màn hình thứ 2 có nút bấm để quay ngược về màn hình thứ 1	96
Hình 2. 25. Một ví dụ của Widget Scaffold	101
Hình 2. 26. Một ví dụ của Widget Container.....	103
Hình 2. 27. Một ví dụ của Widget Column.....	104
Hình 2. 28. Một ví dụ của Widget Row	104
Hình 2. 29. Một ví dụ của Widget Text	106
Hình 2. 30. Một ví dụ của Widget Text Field	107
Hình 2. 31. Một số ví dụ về các loại Widget Button.....	109
Hình 2. 32. Một ví dụ về Widget Stack	111
Hình 2. 33. Một ví dụ khác về Widget Stack khi được ứng dụng vào xây dựng giao diện	111
Hình 2. 34. Một ví dụ về Widget Form được sử dụng để xác thực thông tin nhập vào của các trường	113
Hình 2. 35. Khi ta nhập sai trong Form thì trường thông báo lỗi	113
Hình 2. 36. Dạng Confirmation AlertDialog	116
Hình 2. 37. Dạng TextField AlertDialog	116
Hình 2. 38. Dạng Select Dialog	116

Hình 2. 39. AlertDialog cơ bản	116
Hình 2. 40. Một số ví dụ về Widget Icon.....	118
Hình 2. 41. Cấu hình pubspec.yaml cho Image 1	120
Hình 2. 42. Cấu hình pubspec.yaml cho Image 2	120
Hình 2. 43. Demo hiển thị hình ảnh trong Flutter.....	121
Hình 2. 44. Cấu hình widget Image hiển thị ảnh từ Internet.....	121
Hình 2. 45. Demo hiển thị hình ảnh từ Internet	122
Hình 2. 46. Demo hiển thị cho widget Card	123
Hình 2. 47. Tạo First Screen và Second Screen cho ví dụ Tabbar	124
Hình 2. 48. Tạo DefaultTabController cho ví dụ Tabbar.....	124
Hình 2. 49. Tạo Tabbar trong màn hình chính.....	125
Hình 2. 50. Cấu hình đường dẫn tới First Screen và Second Screen cho ví dụ Tabbar	125
Hình 2. 51. Demo hoàn chỉnh cho ví dụ Tabbar	125
Hình 2. 52. Tạo một Drawer trong ứng dụng Flutter vừa khởi tạo	126
Hình 2. 53. Thêm nội dung vào trong Drawer	127
Hình 2. 54. Khởi động máy ảo trong ví dụ Drawer	127
Hình 2. 55. Demo chương trình Flutter sử dụng Widget Drawer	128
Hình 2. 56. Một giao diện Flutter Cơ bản có sử dụng ListView.....	129
Hình 2. 57. Một giao diện Flutter Cơ bản có sử dụng GridView	131
Hình 2. 58. Đoạn code demo cho checkbox	132
Hình 2. 59. Một giao diện Flutter Cơ bản có sử dụng Checkbox	132
Hình 2. 60. Đoạn code demo cho radioButton.....	133
Hình 2. 61. Một giao diện Flutter Cơ bản sử dụng radioButton	134
Hình 2. 62. Một giao diện Flutter Cơ bản sử dụng cả 2 dạng ProgressBar	135
Hình 2. 63. Một ví dụ minh họa cho Widget SnackBar.....	137

Hình 2. 64. Một ví dụ minh họa cho Widget Tooltip.....	140
Hình 2. 65. Một ví dụ minh họa cho Widget Slider.....	142
Hình 2. 66. Một ví dụ minh họa cho Widget Switch	145
Hình 2. 67. Cài đặt thư viện fl_chart để sử dụng Widget Chart	147
Hình 2. 68. Demo chương trình hiển thị một biểu đồ bằng cách sử dụng Widget Chart	148
Hình 2. 69. Một ví dụ minh họa cho Widget BottomNavigationBar.....	150
Hình 2. 70. Một ví dụ minh họa cho Widget Theme	151
Hình 2. 71. Một ví dụ minh họa cho Widget Table	152
Hình 2. 72. Một ví dụ minh họa cho Widget Animation	153
Hình 3. 1. Cây thư mục chương trình Máy tính (Calculator App).....	155
Hình 3. 2. Giao diện chương trình Máy tính được chia làm 2 phần khu vực hiển thị	161
Hình 3. 3. Lấy ví dụ đây là một nút bấm trong khu vực bàn phím	164
Hình 3. 4. Demo chương trình máy tính	167
Hình 3. 5. Cấu trúc thư mục chương trình todo	168
Hình 3. 6. Menu của ứng dụng.....	169
Hình 3. 7. Màn hình chính khi mở ứng dụng.....	170
Hình 3. 8. Màn hình các công việc đã xong.....	171
Hình 3. 9. Các thành phần của một item	172
Hình 3. 10. Giao diện sửa chữa công việc	172
Hình 3. 11. Giao diện các item đã xoá	173
Hình 3. 12. Thành phần của delete item	174
Hình 3. 13. Sau khi hoàn tác xoá công việc	174
Hình 3. 14. Thư mục chương trình TicTacToe	175

Hình 3. 15. Khởi tạo trò chơi TicTacToe.....	176
Hình 3. 16. Sử dụng GridView để hiện thị các ô trong TicTacToe	177
Hình 3. 17. InkWell để xử lý sự kiện khi người dùng nhấn vào ô.....	177
Hình 3. 18. Danh sách các vị trí có thẻ chiến thắng.....	178
Hình 3. 19. Hiển thị tiêu đề và lượt chơi người hiện tại	184
Hình 3. 20. Hiển thị vùng chơi Tic Tac Toe và xử lý hành động người chơi.....	185
Hình 3. 21. Hiển thị vùng chơi Tic Tac Toe và xử lý hành động người chơi.....	186
Hình 3. 22. Hiển thị thông báo kết thúc trò chơi.....	186
Hình 3. 23. Hình ảnh demo trò chơi TicTacToe	187
Hình 3. 24. Cấu trúc thư mục chương trình tính toán BMI	189
Hình 3. 25. Chương trình BMI Calculator chủ đề tối	194
Hình 3. 26. Chương trình BMI Calculator chủ đề sáng	194
Hình 3. 27. Từng thành phần được đánh số trong màn hình chính home_screen của chương trình BMI Calculator	196
Hình 3. 28. Xây dựng giao diện đảm bảo code tối giản và dễ hiểu	197
Hình 3. 29. Màn hình hiển thị kết quả của chương trình tính toán BMI.....	198
Hình 3. 30. Demo chương trình BMI Calculator với Input như trên hình.....	199
Hình 3. 31. Demo chương trình BMI Calculator với Output như trên hình	200

LỜI MỞ ĐẦU

Lý do chọn đề tài

Hiện nay số lượng điện thoại ngày càng đông và công nghệ cũng ngày càng phát triển. Trong mỗi ngày thì mỗi người dành thời gian khá nhiều cho điện thoại từ giải trí cho đến làm việc. Từ đó ứng dụng phần mềm trên điện thoại cũng phát triển nhanh chóng. Từ đó nhóm em có quyết định chọn đề tài nghiên cứu về framework Fluter để thực hiện việc phát triển phần mềm trên điện thoại di động.

Mục đích

Học tập và phát triển bản thân với ngôn ngữ lập trình Dart và Framework Flutter. Xây dựng các giao diện cơ bản phục vụ cho mục đích học tập và tìm hiểu Flutter.

Đối tượng và phạm vi

Framework Flutter và ngôn ngữ lập trình Flutter

Phương pháp thực hiện

Sử dụng công nghệ Flutter, ngôn ngữ lập trình Dart.

Các công cụ như: Visual Studio Code, Android Studio.

Cấu trúc báo cáo

Cấu trúc của báo cáo gồm 3 phần chính:

- ❖ Chương 1. Tổng quan về Flutter
- ❖ Chương 2. Chi tiết về Flutter
- ❖ Chương 3. Các chương trình Demo

CHƯƠNG 1. TỔNG QUAN VỀ FLUTTER

1.1. Flutter là gì

Nói chung, tạo một ứng dụng di động là một công việc rất phức tạp và đầy thử thách. Có rất nhiều framework có sẵn, cung cấp các tính năng tuyệt vời để phát triển các ứng dụng di động. Để phát triển các ứng dụng dành cho thiết bị di động, Android cung cấp một framework gốc dựa trên ngôn ngữ Java và Kotlin, trong khi iOS cung cấp một framework dựa trên ngôn ngữ Objective-C / Swift. Vì vậy, chúng ta cần hai ngôn ngữ và framework khác nhau để phát triển ứng dụng cho cả hai hệ điều hành. Ngày nay, để khắc phục sự phức tạp này, có một số framework đã được giới thiệu hỗ trợ cả hệ điều hành cùng với các ứng dụng dành cho máy tính để bàn. Những loại framework này được gọi là công cụ phát triển đa nền tảng.



Hình 1. 1. Flutter là gì?

Framework phát triển đa nền tảng có khả năng viết một code và có thể triển khai trên nhiều nền tảng khác nhau (Android, iOS và Máy tính để bàn). Nó tiết kiệm rất nhiều thời gian và nỗ lực phát triển của các nhà phát triển. Có một số công cụ có

sẵn để phát triển đa nền tảng, bao gồm các công cụ dựa trên web, chẳng hạn như Ionic từ Drifty Co. vào năm 2013, Phonegap từ Adobe, Xamarin từ Microsoft và React Native form của Facebook. Mỗi framework này có mức độ thành công khác nhau trong ngành công nghiệp di động. Gần đây, một framework công tác mới đã được giới thiệu trong họ phát triển đa nền tảng có tên là Flutter được phát triển từ Google.

Flutter là một bộ công cụ giao diện người dùng để tạo các ứng dụng nhanh, đẹp, được biên dịch nguyên bản cho thiết bị di động, web và máy tính để bàn với một ngôn ngữ lập trình và cơ sở code duy nhất. Nó là miễn phí và code nguồn mở. Ban đầu nó được phát triển từ Google và bây giờ được quản lý theo tiêu chuẩn ECMA . Ứng dụng Flutter sử dụng ngôn ngữ lập trình Dart để tạo ứng dụng. Các phi tiêu chương trình có phiếu một số tính năng tương tự như ngôn ngữ lập trình khác, chẳng hạn như Kotlin và Swift, và có thể xuyên biên dịch thành code JavaScript.

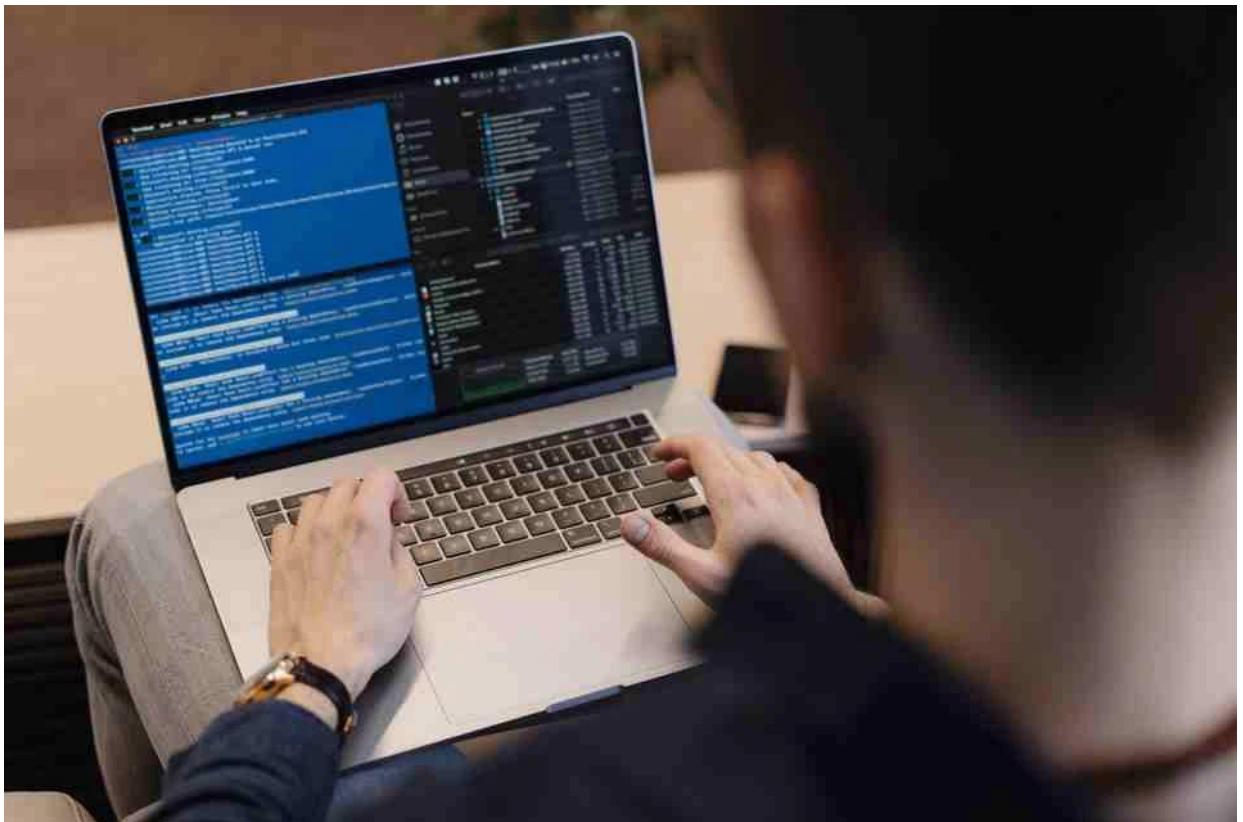
Flutter chủ yếu được tối ưu hóa cho các ứng dụng di động 2D có thể chạy trên cả nền tảng Android và iOS. Chúng ta cũng có thể sử dụng nó để xây dựng các ứng dụng đầy đủ tính năng, bao gồm máy ảnh, bộ nhớ, vị trí địa lý, mạng, SDK của bên thứ ba, v.v.

1.2. Lịch sử phát triển của Flutter

Flutter đã trở thành framework nổi tiếng có tác dụng phát triển ứng dụng di động và web từ Google. Nền tảng được ra mắt lần đầu vào năm 2017 tại hội nghị Dart Developer Summit. Ban đầu, Flutter được thiết kế để giúp các nhà phát triển xây dựng ứng dụng di động đẹp và tương thích trên nhiều nền tảng với một code nguồn duy nhất. Từ đó tiết kiệm thời gian và công sức cho người dùng trong quá trình phát triển.\

Ban đầu nó được biết đến với tên code “Sky” và có thể chạy trên hệ điều hành Android. Sau khi công bố Flutter, phiên bản Flutter Alpha đầu tiên (v-0.06) đã được phát hành vào tháng 5 năm 2017 .Sau đó, trong bài phát biểu quan trọng của ngày Nhà phát triển Google tại Thượng Hải, Google đã tung ra bản xem trước thứ hai của Flutter vào tháng 9 năm 2018 , đây là bản phát hành lớn cuối cùng trước phiên bản

Flutter 1.0. Vào ngày 4 tháng 12 năm 2018 , phiên bản ổn định đầu tiên của framework Flutter đã được phát hành tại sự kiện Flutter Live, ký hiệu là Flutter 1.0. Bản phát hành ổn định hiện tại của framework là Flutter v1.9.1 + hotfix.6 vào ngày 24 tháng 10 năm 2019.



Hình 1. 2. Nguồn gốc hình thành và điểm đặc đáo của Flutter

Tính đến thời điểm hiện tại, Flutter đã phát triển với tốc độ chóng mặt thông qua việc ra mắt các phiên bản mới. Công nghệ cung cấp các tính năng cải tiến và sửa lỗi để người dùng có được trải nghiệm tốt nhất khi phát triển ứng dụng di động và web.

1.3. Các tính năng chính của Flutter

Hot Reload (Tải lại nóng):

Một trong những tính năng nổi bật của Flutter là khả năng Hot Reload. Các nhà phát triển có thể ngay lập tức xem những thay đổi họ thực hiện trên code được

phản ánh trên giao diện ứng dụng, mà không cần phải build lại toàn bộ ứng dụng. Điều này giúp tăng tốc đáng kể quá trình phát triển và khuyến khích thử nghiệm cũng như lặp lại.

Widget (Các khối dựng):

Flutter được xây dựng dựa trên khái niệm widget, là những khối xây dựng để tạo các thành phần UI. Mọi thứ trong Flutter đều là một widget, từ nút bấm và văn bản đến bộ cục và hoạt ảnh. Cách tiếp cận dựa trên widget này giúp đơn giản hóa việc phát triển UI và cho phép các nhà phát triển dễ dàng tạo ra các giao diện người dùng phức tạp.

Phát triển đa nền tảng:

Flutter cho phép các nhà phát triển viết một codebase duy nhất chạy trên nhiều nền tảng, bao gồm iOS, Android, web và desktop. Điều này giúp loại bỏ sự cần thiết phải duy trì các codebase riêng biệt cho các nền tảng khác nhau, tiết kiệm thời gian và công sức.

Giao diện người dùng biểu cảm:

Bộ widget được thiết kế sẵn phong phú và các thành phần có thể tùy chỉnh của Flutter cho phép các nhà phát triển tạo ra giao diện người dùng đẹp mắt và tương tác. Điều này giúp tạo ra các ứng dụng có giao diện người dùng nhất quán trên các nền tảng.

Hiệu suất nhanh:

Ứng dụng Flutter được biên dịch thành code ARM gốc, đảm bảo hiệu suất cao và giảm thiểu chi phí. Việc không có cầu nối giữa ứng dụng và nền tảng cũng góp phần tăng tốc độ thực thi.

Widget Material Design và Cupertino:

Flutter cung cấp cả widget Material Design (dành cho Android) và widget Cupertino (dành cho iOS) để đảm bảo các ứng dụng tuân theo các nguyên tắc thiết kế riêng cho từng nền tảng. Điều này giúp đạt được trải nghiệm người dùng giống như bản địa.

Hỗ trợ hoạt ảnh phong phú:

Flutter cung cấp một thư viện hoạt ảnh mạnh mẽ cho phép các nhà phát triển dễ dàng tạo các hoạt ảnh mượt mà và phức tạp. Tính năng này rất cần thiết để nâng cao sự tương tác của người dùng và tạo giao diện ứng dụng năng động.

Truy cập vào các tính năng gốc:

Mặc dù Flutter là một framework độc lập, nó cũng cung cấp các plugin cho phép các nhà phát triển truy cập vào các tính năng của thiết bị gốc như camera, vị trí, cảm biến, v.v. Điều này đảm bảo rằng các nhà phát triển có thể tận dụng toàn bộ khả năng của nền tảng bên dưới.

Hỗ trợ cộng đồng mạnh mẽ:

Flutter có một cộng đồng các nhà phát triển đang phát triển nhanh chóng, những người tích cực đóng góp vào sự phát triển của nó. Điều này dẫn đến vô số tài nguyên, gói và plugin sẵn có, giúp việc giải quyết các nhu cầu phát triển khác nhau trở nên dễ dàng hơn.

Quốc tế hóa và bản địa hóa:

Flutter cung cấp hỗ trợ tích hợp cho quốc tế hóa và bản địa hóa, cho phép các ứng dụng dễ dàng được dịch sang nhiều ngôn ngữ và thích ứng với các khu vực khác nhau.

Công cụ kiểm thử và gỡ lỗi:

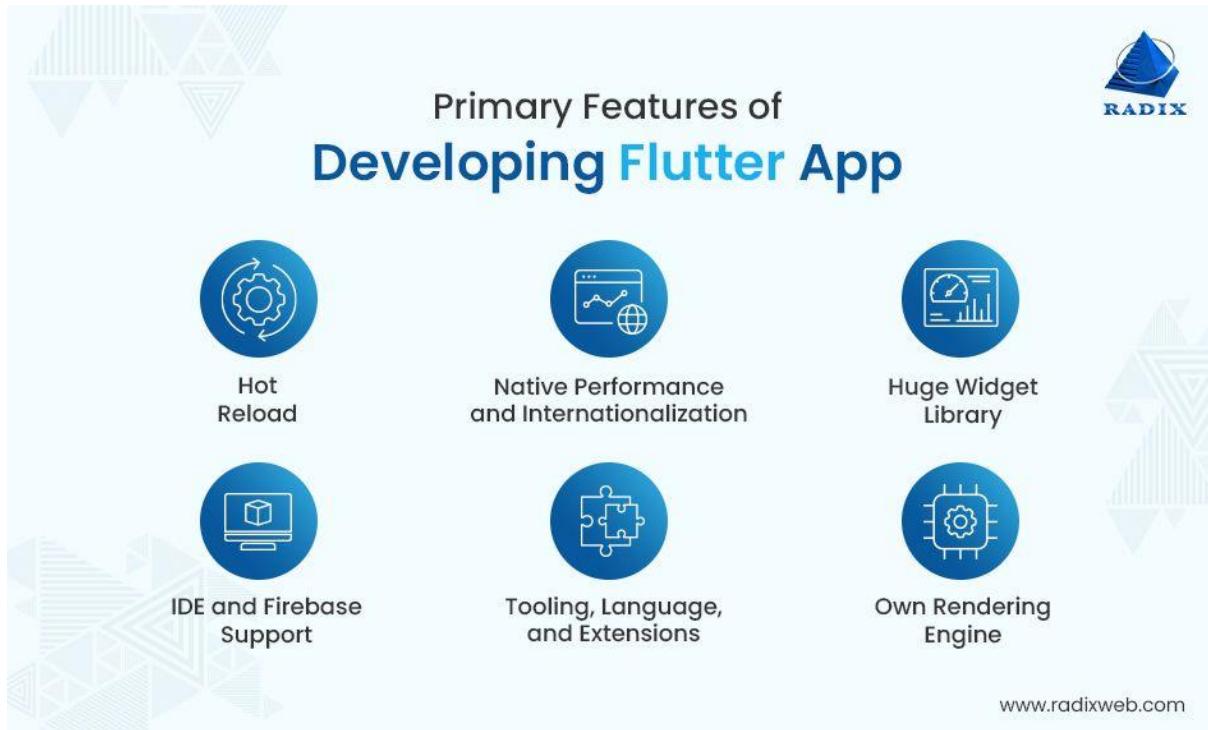
Flutter bao gồm các công cụ kiểm thử và gỡ lỗi mạnh mẽ giúp các nhà phát triển xác định các vấn đề và tối ưu hóa code của họ. Kết hợp với Hot Reload, điều này dẫn đến phát triển hiệu quả và khắc phục lỗi nhanh hơn.

Khả năng mở rộng và bảo trì:

Kiến trúc mô-đun và dựa trên widget của Flutter cho phép các nhà phát triển xây dựng các codebase có khả năng mở rộng và bảo trì. Điều này đặc biệt hữu lợi cho các dự án yêu cầu cập nhật và cải tiến liên tục.

Kết luận:

Các tính năng của Flutter giúp các nhà phát triển tạo ra các ứng dụng đa nền tảng chất lượng cao với hiệu suất ấn tượng và giao diện người dùng tuyệt đẹp. Tính linh hoạt, dễ sử dụng và cộng đồng hỗ trợ mạnh mẽ của nó biến Flutter trở thành lựa chọn lý tưởng cho các nhà phát triển đang tìm kiếm các giải pháp hiệu quả và thiết thực cho phát triển ứng dụng hiện đại. Khi framework Flutter tiếp tục phát triển, nó có khả năng giới thiệu nhiều tính năng thú vị hơn nữa, giúp nâng cao hơn nữa trải nghiệm phát triển và khả năng



Hình 1. 3. Các tính năng chính của Flutter

1.4. Kiến trúc của Flutter

Kiến trúc Flutter gồm 3 thành phần chính:

1. Framework

- Là nền tảng cơ bản cung cấp các API, widget và công cụ cốt lõi để xây dựng ứng dụng Flutter.
- Trách nhiệm phát triển và duy trì ứng dụng.
- Cung cấp các widget để xây dựng giao diện người dùng.

- Bao gồm các thư viện hỗ trợ cho các chức năng như:
 - i. Networking
 - ii. State Management
 - iii. Animations
 - iv. Material Design
 - v. Cupertino Design

2. Engine

- Là trình kết xuất chịu trách nhiệm hiển thị giao diện người dùng (UI) của ứng dụng.
- Phân tích code Dart và tạo các lệnh đồ họa tương ứng với nền tảng hệ điều hành cụ thể.
- Cung cấp khả năng render giao diện người dùng.
- Viết bằng C++ để đảm bảo hiệu suất cao.
- Có thể sử dụng GPU để tăng tốc độ render.

3. Platform

- Cung cấp các API để truy cập các chức năng của thiết bị.
- Là cầu nối giữa framework và nền tảng hệ điều hành cụ thể. Cho phép Flutter hoạt động trên nhiều nền tảng khác nhau như iOS, Android, Web, Windows, macOS, Linux

Mối quan hệ tương tác:

1. Framework tương tác với Engine:

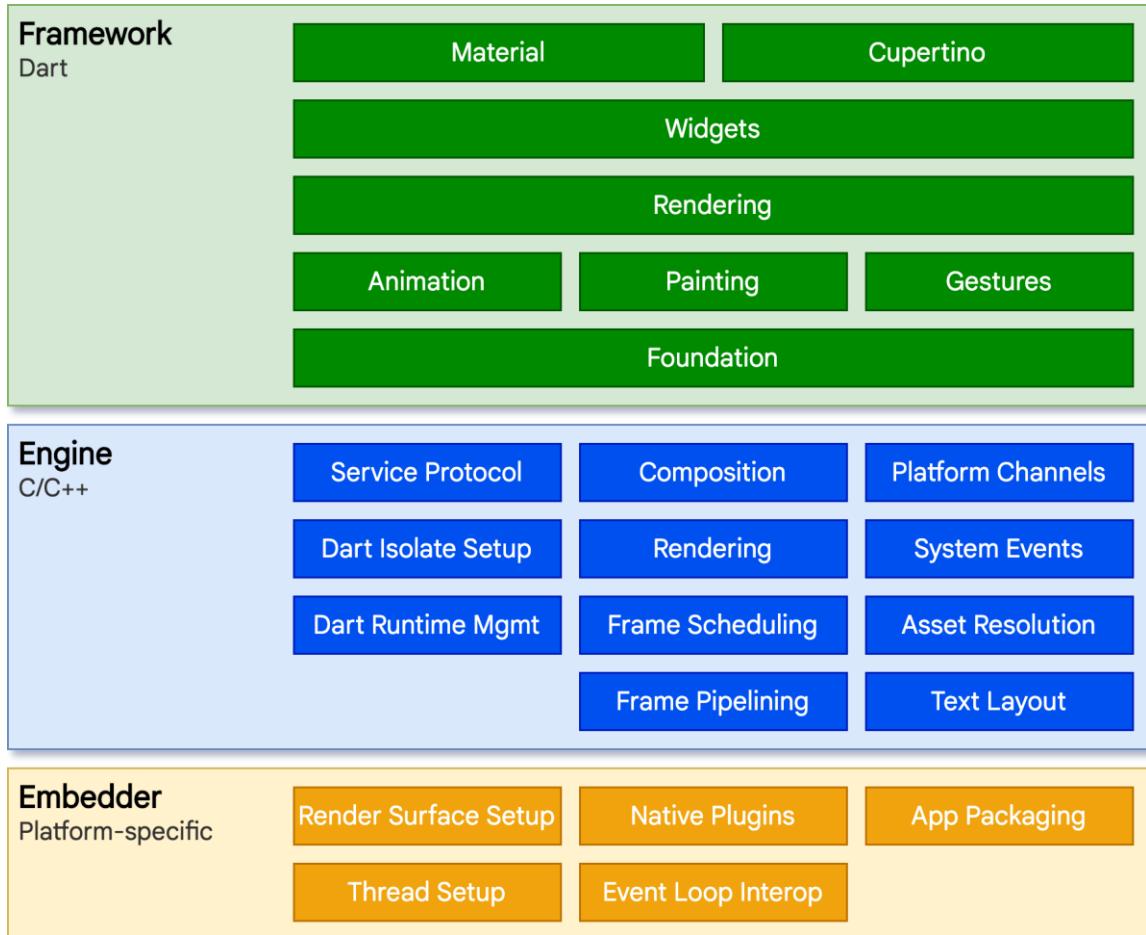
- Framework cung cấp các widget và API để định nghĩa UI của ứng dụng.
- Engine phân tích các widget này và tạo các lệnh đồ họa tương ứng.

2. Engine tương tác với Embedder:

- Engine gửi các lệnh đồ họa cho Embedder để hiển thị trên màn hình thiết bị.
- Embedder sử dụng các API hệ điều hành để hiển thị các lệnh đồ họa này.

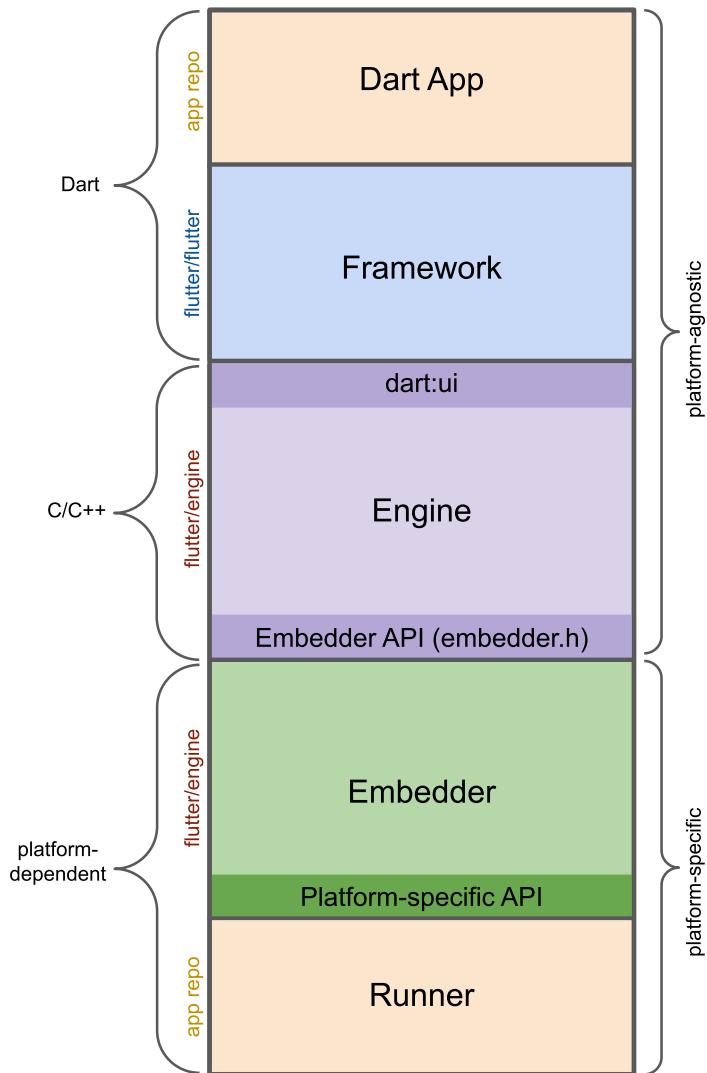
3. Embedder tương tác với hệ điều hành:

- Embedder truy cập các chức năng hệ điều hành thông qua các API được cung cấp.
- Ví dụ: truy cập camera, bộ nhớ, cảm ứng, v.v.



Hình 1.4. Kiến trúc tổng thể của Flutter

Bên cạnh đó, ta còn có cả mô hình giải phẫu của một ứng dụng Flutter được mô tả như sau:



Hình 1.5. Mô hình giải phẫu của một ứng dụng Flutter

1. Dart App

- Ghép các widget thành giao diện người dùng mong muốn.
- Thực thi logic nghiệp vụ.
- Thuộc sở hữu của nhà phát triển ứng dụng.

2. Framework

- Cung cấp API cấp cao để xây dựng các ứng dụng chất lượng cao (ví dụ: widget, kiểm tra nhán, nhận dạng cử chỉ, khả năng truy cập, nhập văn bản).
- Ghép cây widget của ứng dụng thành một cảnh.

3. Engine

- Chịu trách nhiệm raster hóa các cảnh được hợp thành.
- Cung cấp triển khai cấp thấp của các API cốt lõi của Flutter (ví dụ: đồ họa, bộ cục văn bản, runtime Dart).
- Công khai chức năng của nó cho framework bằng cách sử dụng API *dart:ui*.
- Tích hợp với một nền tảng cụ thể bằng cách sử dụng API *Embedder* của Engine

4. Runner

- Ghép các thành phần lại và nhúng thành các gói để có thể chạy trên được nền tảng đích
- Là một phần của mẫu ứng dụng được tạo bởi *Flutter create*, thuộc sở hữu của nhà phát triển ứng dụng.

1.5. Ưu điểm và nhược điểm của Flutter

❖ Ưu điểm

Flutter đáp ứng các nhu cầu và yêu cầu tùy chỉnh để phát triển các ứng dụng di động. Nó cũng cung cấp nhiều lợi thế, được liệt kê dưới đây.

- Nó làm cho quá trình phát triển ứng dụng cực kỳ nhanh chóng vì tính năng tải lại nóng (Hot Reload). Tính năng này cho phép người lập trình thay đổi hoặc cập nhật code được phản ánh ngay sau khi các thay đổi được thực hiện.
- Nó cung cấp trải nghiệm cuộn mượt mà và liền mạch khi sử dụng ứng dụng mà không bị treo hoặc cắt nhiều, giúp chạy ứng dụng nhanh hơn so với các framework phát triển ứng dụng dành cho thiết bị di động khác.
- Flutter làm giảm thời gian và nỗ lực kiểm tra. Như chúng ta đã biết, các ứng dụng rung là đa nền tảng, do đó người thử nghiệm không cần phải luôn

chạy cùng một nhóm thử nghiệm trên các nền tảng khác nhau cho cùng một ứng dụng.

- ⊕ Nó có giao diện người dùng tuyệt vời vì nó sử dụng tiện ích tập trung vào thiết kế, các công cụ phát triển cao, API nâng cao và nhiều tính năng khác.
- ⊕ Nó tương tự như một framework phản ứng trong đó các nhà phát triển không cần cập nhật nội dung giao diện người dùng theo cách thủ công.
- ⊕ Nó phù hợp với các ứng dụng MVP (Sản phẩm khả thi tối thiểu) vì quá trình phát triển nhanh chóng và tính chất đa nền tảng của nó

❖ Nhược điểm

Trước đó chúng ta đã thấy rằng Flutter có nhiều ưu điểm, nhưng nó cũng chưa một số nhược điểm, được đưa ra dưới đây.

- ⊕ Flutter là một ngôn ngữ tương đối mới cần được hỗ trợ tích hợp liên tục thông qua việc duy trì các tập lệnh.
- ⊕ Nó cung cấp quyền truy cập rất hạn chế vào các thư viện SDK. Nó có nghĩa là một nhà phát triển không có nhiều chức năng để tạo một ứng dụng di động. Các loại chức năng như vậy cần được phát triển bởi chính nhà phát triển Flutter.
- ⊕ Nó sử dụng lập trình Dart để viết code, vì vậy một nhà phát triển cần phải học các công nghệ mới. Tuy nhiên, nó rất dễ học đối với các nhà phát triển.

1.6. Một số thông tin liên quan khác

1.6.1. Các nguồn tài liệu khoá học Flutter

1.6.1.1. Khoá học trả phí

Một số khoá học về Flutter trả phí nhưng đi kèm là sự đầy đủ và chất lượng như Udemy, CodeGym... Tuy mất phí nhưng sự đánh đổi lại không hề nhỏ, lợi ích mà chúng ta nhận được qua những khoá học trả phí như sau:

- ⊕ **Nội dung chuyên sâu và cập nhật:** Khóa học trả phí thường được đầu tư kỹ lưỡng về nội dung, với các bài giảng chi tiết, cập nhật và đi sâu vào các chủ đề chuyên sâu.
- ⊕ **Giảng viên uy tín:** Khóa học trả phí thường được giảng dạy bởi các giảng viên uy tín, có nhiều kinh nghiệm trong lĩnh vực giảng dạy và thực hành.
- ⊕ **Hỗ trợ và tương tác:** Khóa học trả phí thường cung cấp dịch vụ hỗ trợ và tương tác tốt hơn, giúp ta giải đáp thắc mắc và trao đổi với giảng viên và học viên khác.
- ⊕ **Chứng chỉ:** Một số khóa học trả phí cấp chứng chỉ sau khi hoàn thành, giúp ta nâng cao năng lực và tăng cơ hội việc làm.

Có nhiều loại khóa học trả phí khác nhau, bao gồm:

1. **Khóa học trực tuyến trên các nền tảng online:** Có thể học mọi lúc mọi nơi với các bài giảng video, bài tập thực hành và tài liệu học tập.
2. **Khóa học trực tiếp tại trung tâm:** Tham gia học tại một địa điểm cụ thể với giảng viên và các học viên khác.

Một số khoá học Flutter trả phí được nhiều người lựa chọn:

- Udemy: <https://www.udemy.com/courses/search/?src=ukw&q=Flutter>
- CodeGym: <https://codegym.vn/khoa-hoc/Flutter/>

3.394 kết quả cho “Flutter”

Flutter & Dart - The Complete Guide [2024 Edition]
A Complete Guide to the Flutter SDK & Flutter Framework for building native iOS and Android apps
Academind by Maximilian Schwarzmüller, Maximilian Schwarzmüller
4.6 ★★★★☆ (74.574)
Tổng số 30 giờ • 305 bài giảng • Tất cả trình độ
Bán chạy nhất

Flutter Bloc cho Mobile, Web, PC - Tôi chọn bạn
Đa nền tảng, tiết kiệm chi phí, hiệu quả cao
Tien Bui Duc
4.7 ★★★★☆ (28)
Tổng số 21,5 giờ • 133 bài giảng • Tất cả trình độ
Thịnh hành & mới

The Complete Flutter Development Bootcamp with Dart
Officially created in collaboration with the Google Flutter team.
Dr. Angela Yu
4.6 ★★★★☆ (53.157)
Tổng số 29 giờ • 217 bài giảng • Tất cả trình độ

Complete Flutter Guide 2024: Build Android, iOS and Web apps
The Complete Flutter SDK. Flutter Framework, Dart guide to develop fast, production-grade apps for Android, iOS and Web
Sagnik Bhattacharya, Paulina Knop
4.6 ★★★★☆ (685)
Tổng số 24 giờ • 241 bài giảng • Tất cả trình độ

Hình 1. 6. Một số khóa học Flutter trả phí trên Udemy

TÌM HIỂU

LỘ TRÌNH ĐÀO TẠO KHÓA HỌC FLUTTER

TẢI ĐỀ CƯƠNG CHI TIẾT

DART VÀ FLUTTER CĂN BẢN

- Trình bày được tổng quan về Flutter và mô tả được các thành phần trong Flutter.
- Thành thạo ngôn ngữ lập trình Dart và cách sử dụng Dart trong Flutter.
- Thành thạo tạo giao diện người dùng với Flutter.
- Hiểu cách quản lý trạng thái trong ứng dụng Flutter.
- Phát hành ứng dụng lên Play Store (Google), App Store (Apple).
- Quản lý mã nguồn với Git

LẬP TRÌNH FLUTTER NÂNG CAO

- Có khả năng xây dựng ứng dụng di động đa nền tảng phức tạp với Flutter.
- Thành thạo về quản lý trạng thái nâng cao
- Thành thạo các kỹ thuật tối ưu hóa hiệu suất ứng dụng
- Làm việc với API, cơ sở dữ liệu và tích hợp các dịch vụ bên ngoài (Google Maps, Firebase, Thanh toán trực tuyến,...)
- Tự động hóa quy trình phát hành ứng dụng (áp dụng các kỹ thuật CI/CD)

Hình 1. 7. Khoa học Flutter trả phí trên CodeGym

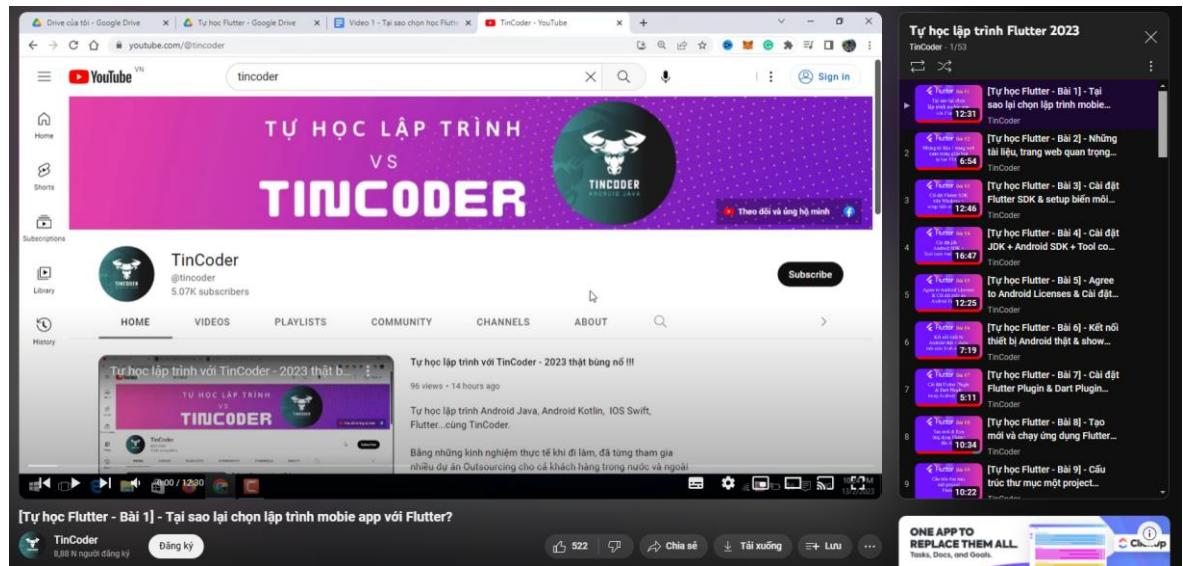
1.6.1.2. Khoá học miễn phí

Ngoài việc phải trả phí để học lập trình ra thì ta cũng có sự lựa chọn khác là lựa chọn các khoá học lập trình Flutter miễn phí trên các nền tảng khác, ví dụ: Youtube, Các trang web phi lợi nhuận. Ưu điểm của những khoá học này là:

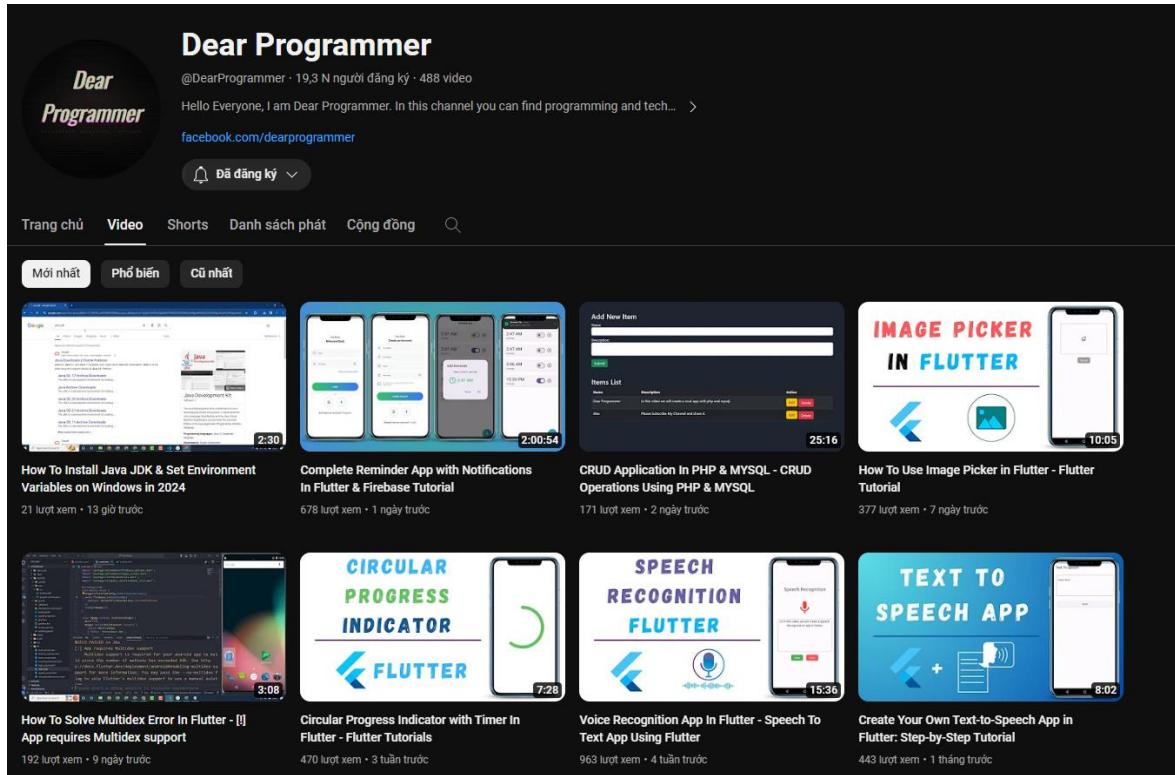
- ⊕ **Tiếp cận kiến thức dễ dàng:** Có thể học tập mọi lúc mọi nơi mà không cần lo lắng về chi phí.
- ⊕ **Thử nghiệm lĩnh vực mới:** Có thể khám phá các lĩnh vực mới mà không cần đầu tư nhiều tiền.
- ⊕ **Phát triển kỹ năng:** Có thể trau dồi kỹ năng và kiến thức mới để nâng cao bản thân.

Một số khoá học Flutter miễn phí:

- TinCoder: <https://www.youtube.com/@tincoder>
- DearProgrammer: <https://www.youtube.com/@DearProgrammer>
- CafeDev: <https://cafedev.vn/series-tu-hoc-Flutter-tu-co-ban-toi-nang-cao>



Hình 1. 8. Khoá học lập trình Flutter miễn phí trên Youtube: TinCoder



Hình 1. 9. Khoá học lập trình Flutter miễn phí trên Youtube: Dear Programmer

The screenshot shows the CafeDev website. At the top, there are navigation links: 'TRANG CHỦ', 'CHIA SẺ', 'HỌC LẬP TRÌNH FREE' (which is underlined in red), and a search icon. Below this, there's a section titled 'MIỄN PHÍ 100% | Series tự học Flutter từ cơ bản tới nâng cao (cập nhật liên tục...)'. It describes Flutter as a powerful UI toolkit used to build natively compiled mobile, web, and desktop applications using a single codebase. It highlights that the series covers basic to advanced topics in Flutter development. Below this text, there's a list of course components: 'Sách, video và tài liệu học git', 'Lộ trình học Flutter từ cơ bản tới nâng cao', 'Thực hành từng widget cơ bản và các vấn đề nâng cao với ví dụ chi tiết', a 'Giới thiệu mọi thứ về Cafedev tại đây' link, and a 'Kho khoá học lập trình online chọn lọc(Cập nhật liên tục...)' link. At the bottom, there's a table with two columns: 'Phần mở đầu' and 'Sách, video và tài liệu'. The table rows are: 0.0 (Kho sách Flutter), 0.1 (Nơi đăng ký nhận ebook lập trình, ebook công nghệ thông tin tại đây), and 0.2 (Video học Flutter(Đang cập nhật...)).

Hình 1. 10. Khoá học lập trình Flutter miễn phí trên web: CafeDev

1.6.2. Các thông tin tuyển dụng về Flutter

Một số mức lương mà các nhà tuyển dụng đưa ra để tuyển dụng các lập trình viên Flutter trên các nền tảng như sau:

❖ Topdev.vn

The screenshot displays five job listings from the Topdev.vn website, each featuring a company logo, job title, salary range, location, experience level, and relevant technologies.

- Senior Mobile Developer (Flutter)** at One Mount Group. Salary: 1.000 USD to 2.500 USD. Location: Quận Hai Bà Trưng, Hà Nội (In Office). Experience: Senior. Duration: 13 months salary fixed. Technologies: Android, iOS, Mobile, API, Flutter. Posted 3 days ago.
- Chuyên viên Lập trình (C#/ASP.NET)** at Công ty CP Tư vấn và Quản lý BĐS Saigon Center Real. Salary: 18.000.000 VND to 25.000.000 VND. Location: Quận 1, Hồ Chí Minh (In Office). Experience: Middle. Technologies: ASP.NET, C#, Flutter. Posted 5 days ago.
- Chuyên Viên Tư Vấn Giải Pháp - Front end/ Back end (Flutter)** at CÔNG TY CỔ PHẦN GIẢI PHÁP CÔNG NGHỆ HCTECH. Salary: 13.000.000 VND to 20.000.000 VND. Location: Thành phố Thủ Dầu Một, Bình Dương (In Office). Experience: Senior. Technologies: Tester, IT Presales, Data Analyst, Triển Khai Phần Mềm. Posted 1 week ago.
- Software Developer (VueJS/ Flutter)** at CUBE SYSTEM VIETNAM CO., LTD. Salary: Thưởng lượng. Location: Quận 12, Hồ Chí Minh (In Office). Experience: Middle. Technologies: Python, AWS, VueJS, Flutter. Posted 1 week ago.
- MOBILE DEVELOPER** at CÔNG TY CỔ PHẦN CYFEER. Salary: 15.000.000 VND to 25.000.000 VND. Location: Quận Thanh Xuân, Hà Nội (In Office). Experience: Middle. Technologies: Android, iOS, Mobile, Flutter. Posted 1 week ago.

Hình 1. 11. Một số tuyển dụng lập trình viên Flutter trên trang Topdev.vn

❖ Topcv.vn

Tuyển dụng 53 việc làm Flutter tại Hà Nội 2024

Trang chủ > Việc làm flutter > Tuyển dụng 53 việc làm Flutter tại Hà Nội 2024

Gợi ý địa điểm: **Hà Nội (53)** Cầu Giấy (16) Nam Từ Liêm (14) Thanh Xuân (6) Đống Đa (4) Tây Hồ (2)

Ưu tiên hiển thị theo: Liên quan Ngày đăng Ngày cập nhật Cần tuyển gấp

Nhân Viên Lập Trình Mobile (Flutter) - Tại Hà Nội - Thu Nhập 15 - 20 Triệu **15 - 20 triệu**

Công ty CP Dược Phẩm Norway Pharmatech As

Hà Nội Còn 30 ngày để ứng tuyển Cập nhật 2 giờ trước **Ứng tuyển**

Flutter Developer (Salary Up To 40M) ✓ **Tới 40 triệu**

CÔNG TY CỔ PHẦN SYSTEMATIC FUNCTIONS

Hà Nội Còn 25 ngày để ứng tuyển Cập nhật 3 giờ trước **Ứng tuyển**

Lập Trình Viên Mobile Flutter (18-22M/ Tháng) Tại Hà Đông , Hà Nội ✓ **18 - 22 triệu**

CÔNG TY CỔ PHẦN TẦM NHÌN QUỐC TẾ ALADDIN

Hà Nội Còn 11 ngày để ứng tuyển Cập nhật 3 ngày trước **Ứng tuyển**

Lập Trình Viên Mobile (Flutter) **Tới 30 triệu**

Công ty Cổ phần Đầu tư và Giải pháp VietIS

Hà Nội Còn 26 ngày để ứng tuyển Cập nhật 5 ngày trước **Ứng tuyển**

Lập Trình Viên Flutter **25 - 30 triệu**

Công ty Cổ phần Đầu tư và Giải pháp VietIS

Hà Nội Còn 24 ngày để ứng tuyển Cập nhật 6 ngày trước **Ứng tuyển**

Hình 1. 12. Một số tuyển dụng lập trình viên Flutter trên trang topcv.vn

Nhìn chung, mức lương của các lập trình viên Flutter tương đối cao và nhu cầu tuyển dụng lập trình viên tăng cao. Mức lương dao động từ 15 – 40 triệu tuỳ vào kinh nghiệm và năng lực, ta có một bảng lương trung bình từng Level như sau:

Level (Kinh nghiệm)	Mức lương trung bình (VND)
Junior Developer (0-1 năm kinh nghiệm)	10.000.000 - 15.000.000
Mid-level Developer (1-3 năm kinh nghiệm)	15.000.000 - 25.000.000
Senior Developer (3-5 năm kinh nghiệm)	25.000.000 - 35.000.000
Team Lead/Architect (5+ năm kinh nghiệm)	35.000.000 - 50.000.000

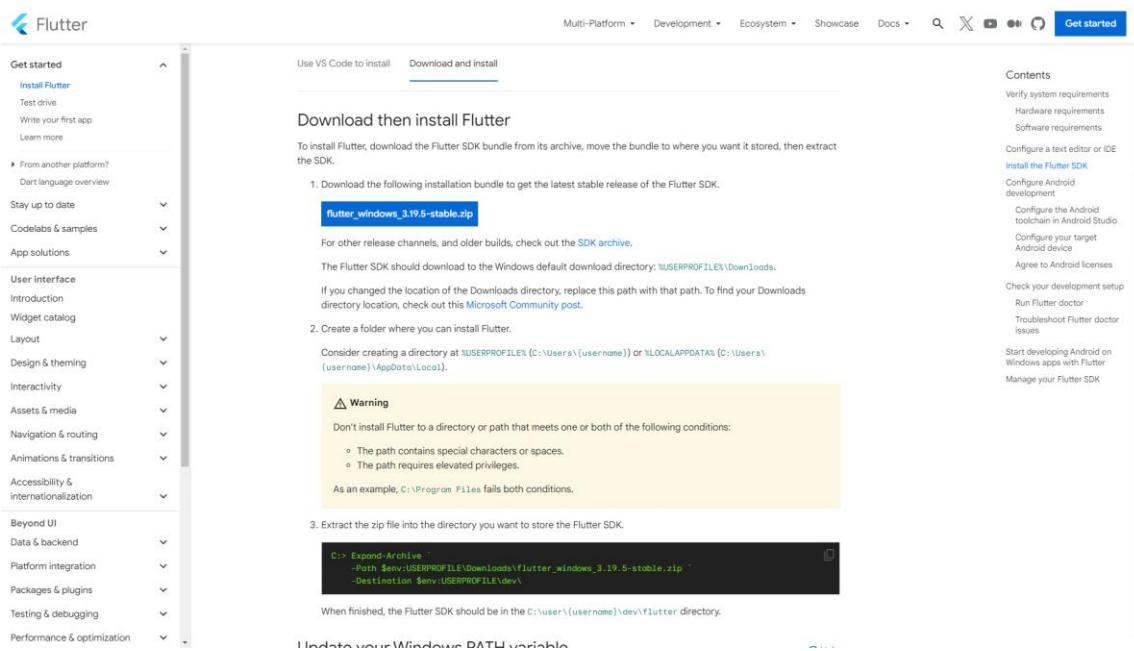
1.7. Cài đặt và cấu hình cần thiết cho việc lập trình với Flutter

1.7.1. Flutter SDK

Trước tiên ta cần cài đặt Flutter SDK. Vậy Flutter SDK là gì ? Flutter SDK là bộ công cụ phát triển phần mềm (SDK) code nguồn mở, miễn phí được tạo bởi Google để xây dựng các ứng dụng di động đa nền tảng chất lượng cao với hiệu suất cao và giao diện người dùng đẹp mắt.

Các bước cài đặt được mô tả như sau (Lưu ý: Các bước hướng dẫn chỉ được áp dụng trên máy tính Windows, không bao gồm MacOS và Linux):

Bước 1: Truy cập vào trang web docs.Flutter.dev để tải Flutter SDK



Hình 1. 13. Tải gói Flutter SDK mới nhất từ trang chủ chính thức của Flutter

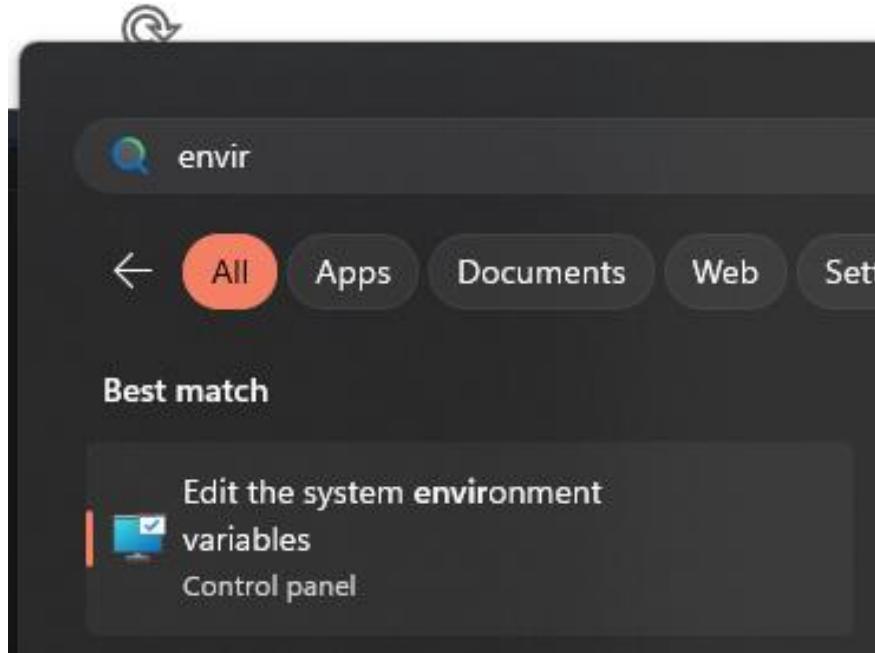
Bước 2: Sau khi tải xong, giải nén và đặt nó ở một thư mục đường dẫn bất kỳ mà ta mong muốn (Lưu ý: Không nên đặt Flutter SDK ở nơi cần sự cho phép (cấp quyền permission từ người dùng))

	Name	Date modified	Type	Size
📁	.git	3/7/2024 9:30 PM	File folder	
📁	.github	3/7/2024 9:23 PM	File folder	
📁	.idea	3/7/2024 9:23 PM	File folder	
📁	.vscode	3/7/2024 9:23 PM	File folder	
📁	bin	3/7/2024 9:24 PM	File folder	
📁	dev	3/7/2024 9:24 PM	File folder	
📁	examples	3/7/2024 9:24 PM	File folder	
📁	packages	3/7/2024 9:24 PM	File folder	
📄	.ci.yaml	3/7/2024 9:18 PM	YAML File	177 KB
📄	.gitattributes	3/7/2024 9:18 PM	txtfile	1 KB
📄	.gitignore	3/7/2024 9:18 PM	txtfile	3 KB
📄	analysis_options.yaml	3/7/2024 9:18 PM	YAML File	12 KB
📄	AUTHORS	3/7/2024 9:18 PM	File	5 KB
📄	CODE_OF_CONDUCT.md	3/7/2024 9:18 PM	MD File	3 KB
📄	CODEOWNERS	3/7/2024 9:18 PM	File	1 KB
📄	CONTRIBUTING.md	3/7/2024 9:18 PM	MD File	12 KB
📄	dartdoc_options.yaml	3/7/2024 9:18 PM	YAML File	2 KB
📄	flutter_console.bat	3/7/2024 9:18 PM	Windows Batch File	2 KB
📄	flutter_root.iml	3/7/2024 9:18 PM	IML File	1 KB
📄	LICENSE	3/7/2024 9:18 PM	File	2 KB
📄	PATENT_GRANT	3/7/2024 9:18 PM	File	2 KB
📄	README.md	3/7/2024 9:18 PM	MD File	7 KB
📄	TESTOWNERS	3/7/2024 9:18 PM	File	28 KB
📄	version	3/7/2024 9:18 PM	File	1 KB

Hình 1. 14. Giải nén và cài đặt gói Flutter SDK tại một đường dẫn bất kỳ

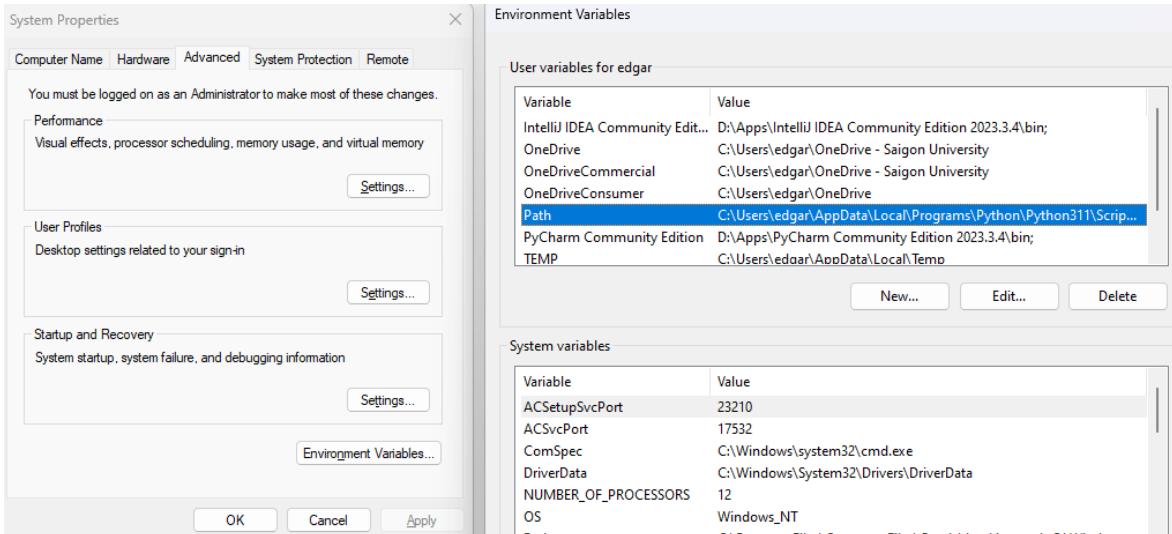
Bước 3: Cấu hình biến môi trường (Environment Variables)

- Ta truy cập Environment Variables từ thanh tìm kiếm Windows



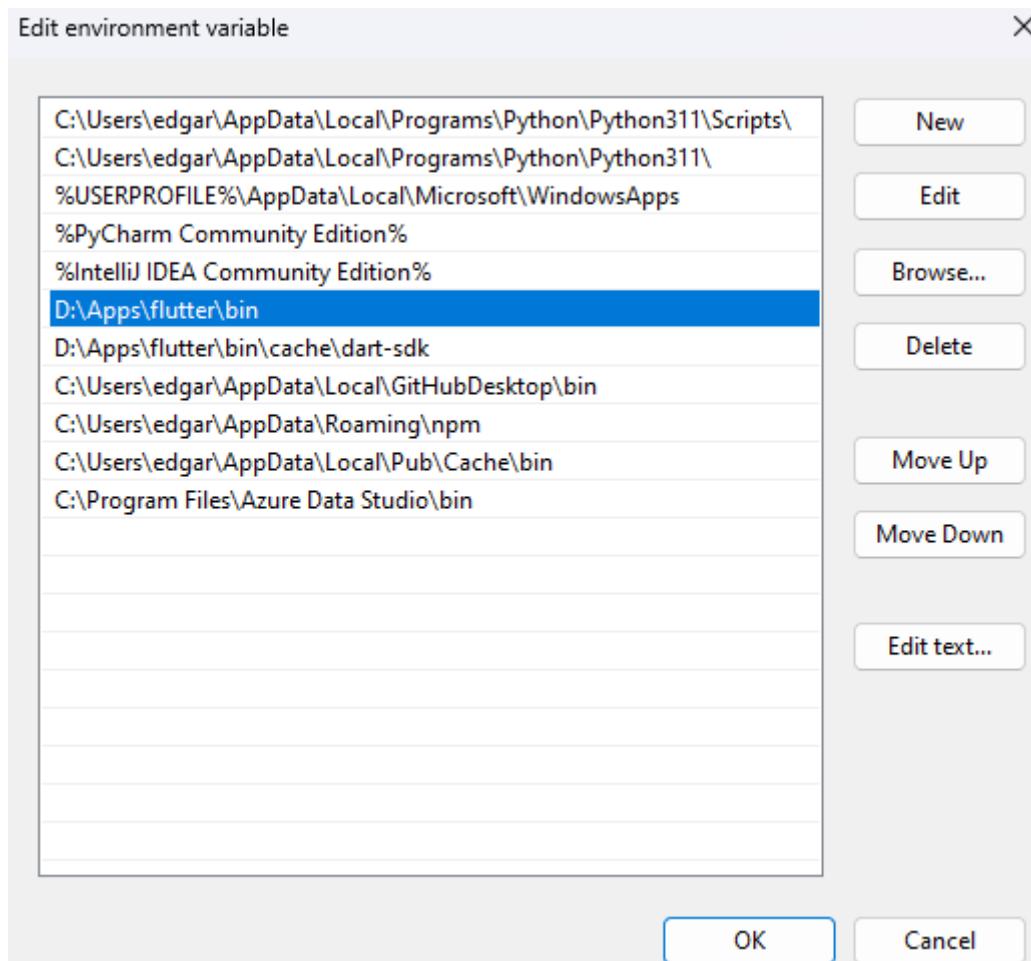
Hình 1. 15. Truy cập Environment Variables từ thanh tìm kiếm

- Chọn vào Environment Variables -> ấn vào Nút Edit của User Variables



Hình 1. 16. Chọn Edit Path của User Variables

- Thiết lập đường dẫn Flutter vào thư mục bin của Flutter SDK như minh hoa



Hình 1. 17. Thiết lập đường dẫn tới thư mục bin của Flutter SDK

- Kiểm tra tất cả trạng thái trước bằng cách chạy lệnh **Flutter doctor** trên CMD hoặc Windows Terminal để đảm bảo rằng tất cả đã được cài đặt thành công và có thể thực hiện lập trình Flutter (Lưu ý: Chỉ cần tích xanh đầy đủ như ở hình dưới là đã có thể lập trình Flutter rồi)

```

Windows PowerShell

PS C:\Users\edgar> flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.19.5, on Microsoft Windows [Version 10.0.22631.3296], locale en-US)
[✓] Windows Version (Installed version of Windows is version 10 or higher)
[✓] Android toolchain - develop for Android devices (Android SDK version 34.0.0)
[✓] Chrome - develop for the web
[✗] Visual Studio - develop Windows apps
  X Visual Studio not installed; this is necessary to develop Windows apps.
    Download at https://visualstudio.microsoft.com/downloads/.
    Please install the "Desktop development with C++" workload, including all of its default components
[✓] Android Studio (version 2023.2)
[✓] Connected device (3 available)
[✓] Network resources

! Doctor found issues in 1 category.

PS C:\Users\edgar>

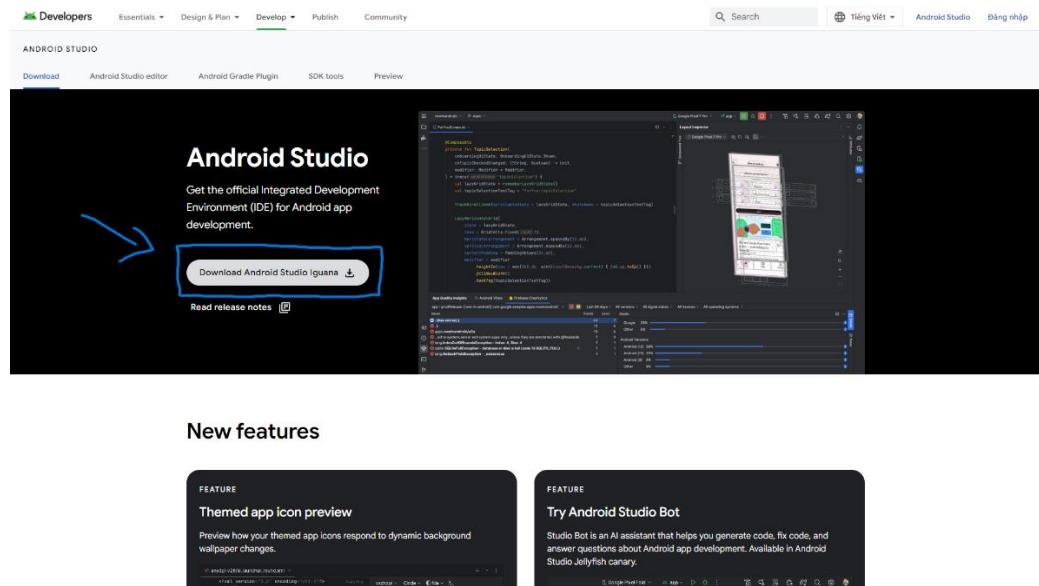
```

Hình 1. 18. Kiểm tra bằng lệnh *Flutter doctor*

1.7.2. IDE

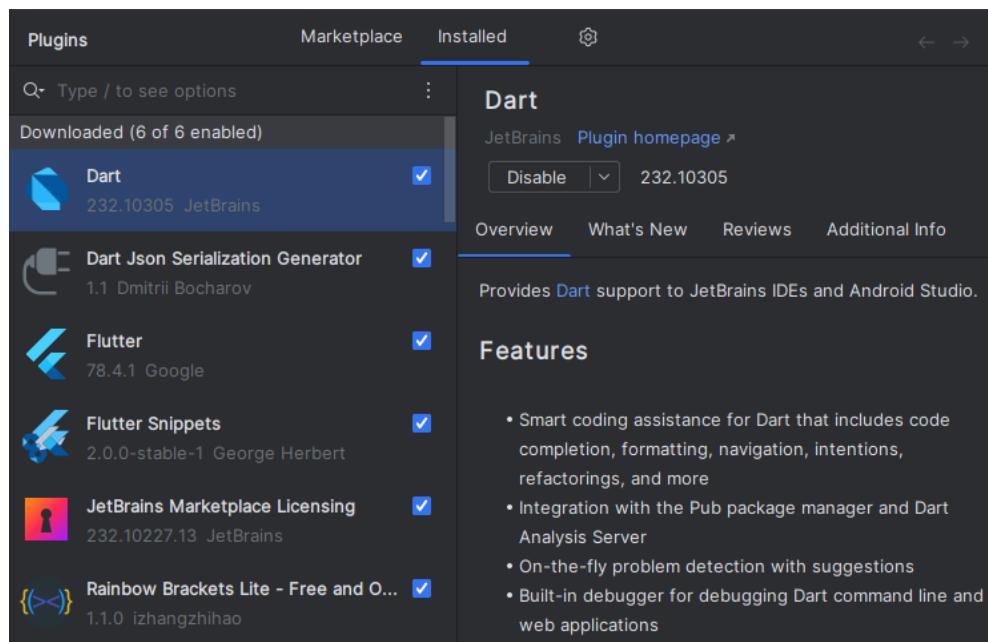
Việc lập trình thì không thể thiếu IDE hoặc Code Editor. Trong bài báo cáo này, chúng em chỉ sử dụng **Android Studio** để lập trình Flutter. Sau đây là các bước hướng dẫn cho việc thiết lập **Android Studio** để có thể lập trình được Flutter:

Bước 1: Truy cập vào trang chủ developer.android.com để cài đặt Android Studio



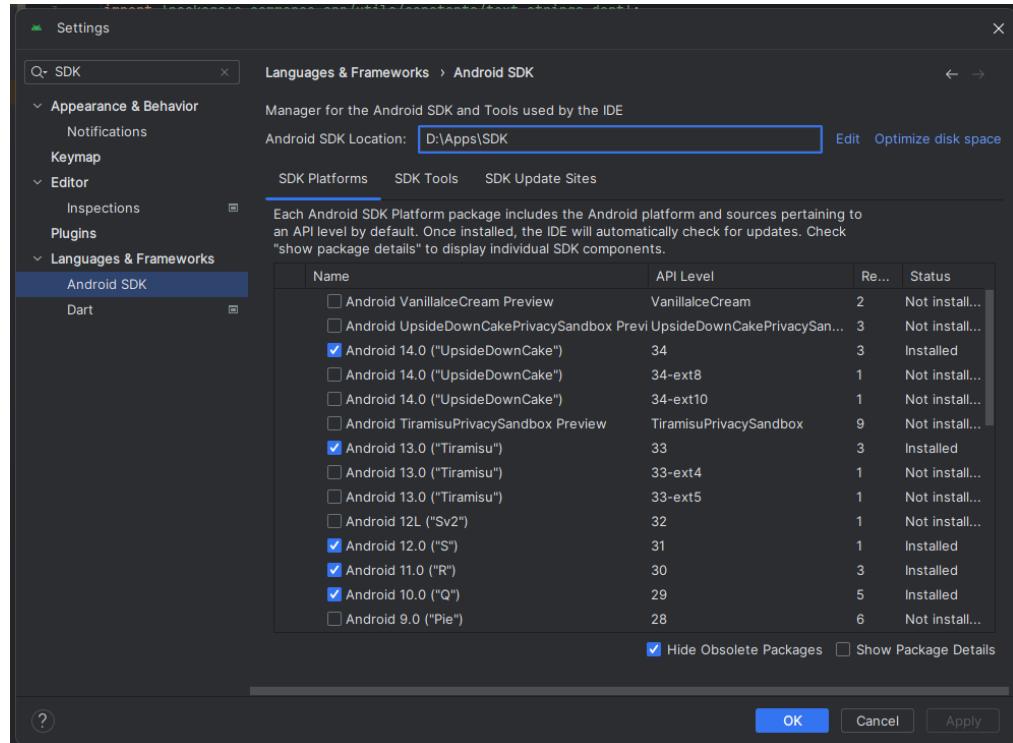
Hình 1. 19. Truy cập vào trang chủ android studio để cài đặt IDE

Bước 2: Sau khi cài đặt IDE xong, ta tiến hành cài đặt các plugin cần thiết như: Dart, Flutter.....

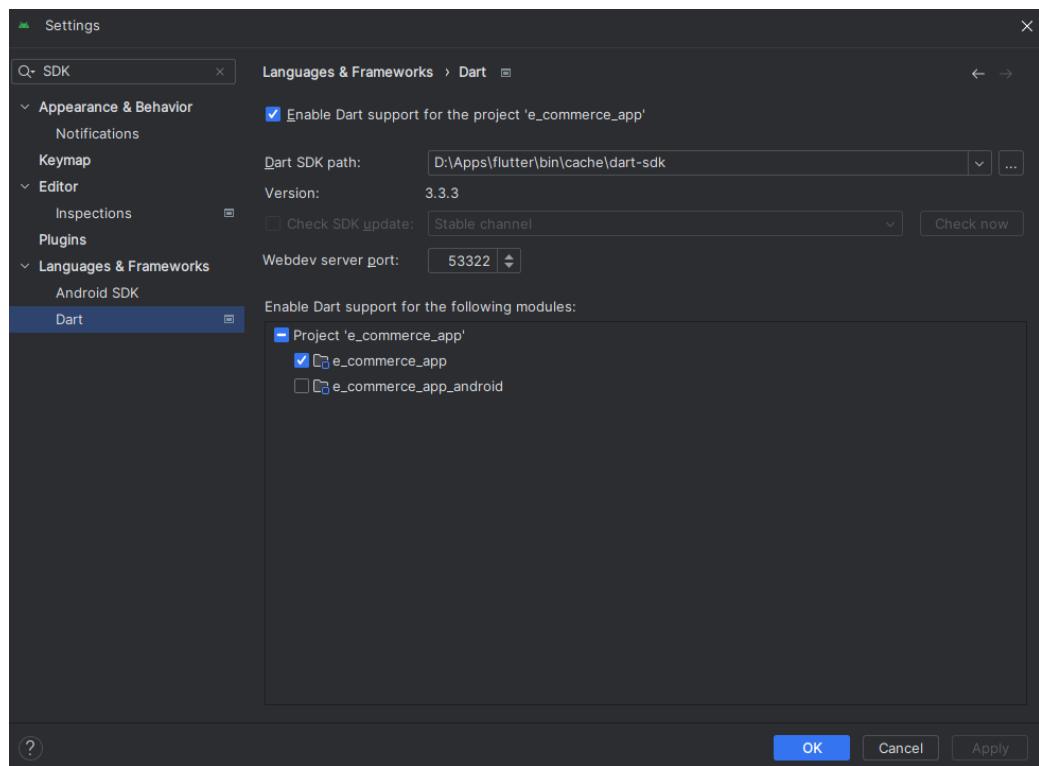


Hình 1. 20. Cài đặt các Plugin cần thiết trên Android Studio

Bước 3: Kiểm tra Android SDK và Dart SDK đã được thiết lập trên Android Studio hay chưa. Setting -> Android SDK, Setting -> Dart



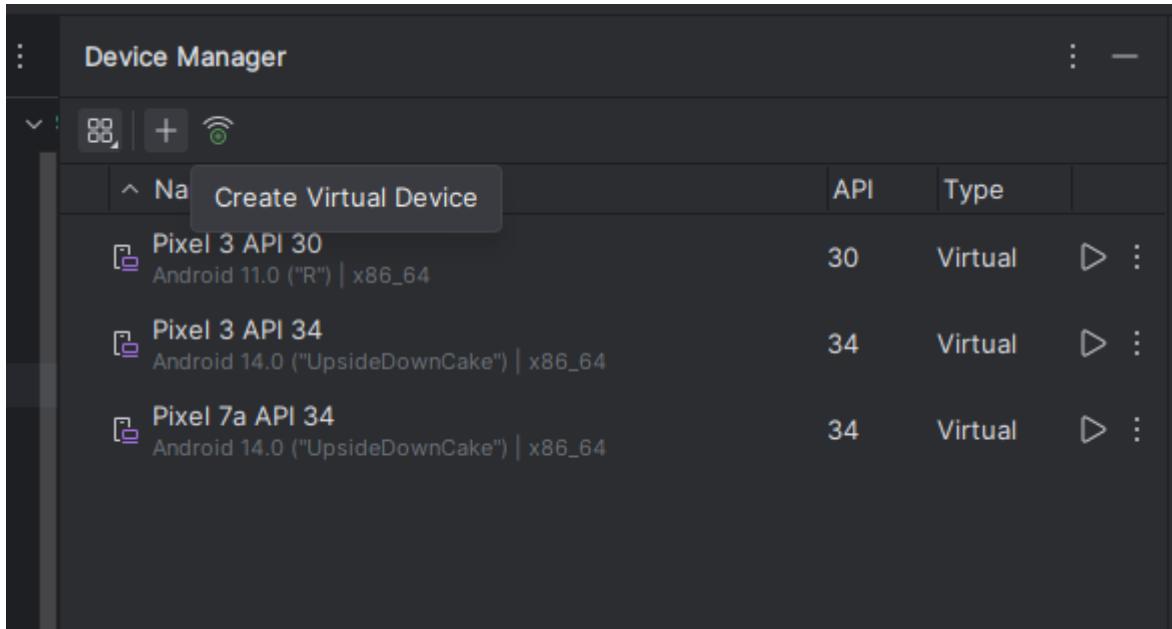
Hình 1. 21. Kiểm tra Android SDK



Hình 1. 22. Kiểm tra Dart SDK

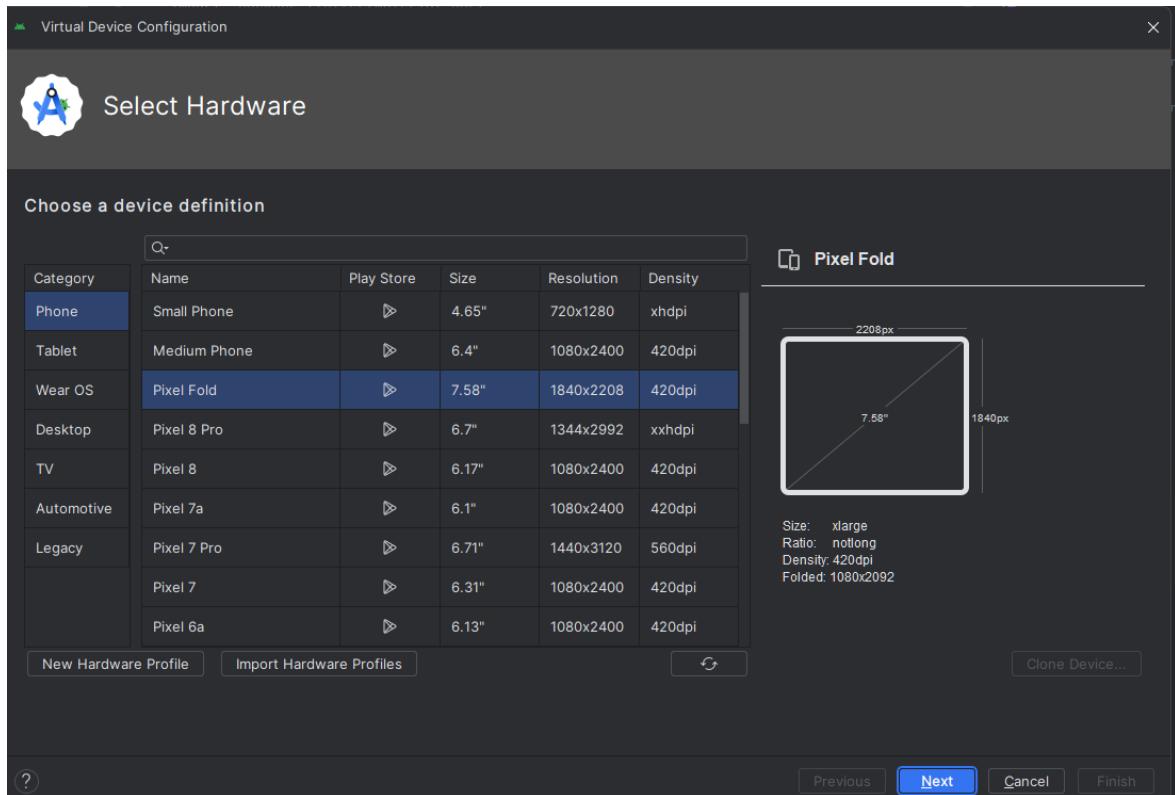
Bước 4: Tạo thử máy ảo Emulator trên Android Studio

- Ta tạo một máy ảo bằng cách ấn vào nút *Create Virtual Device*

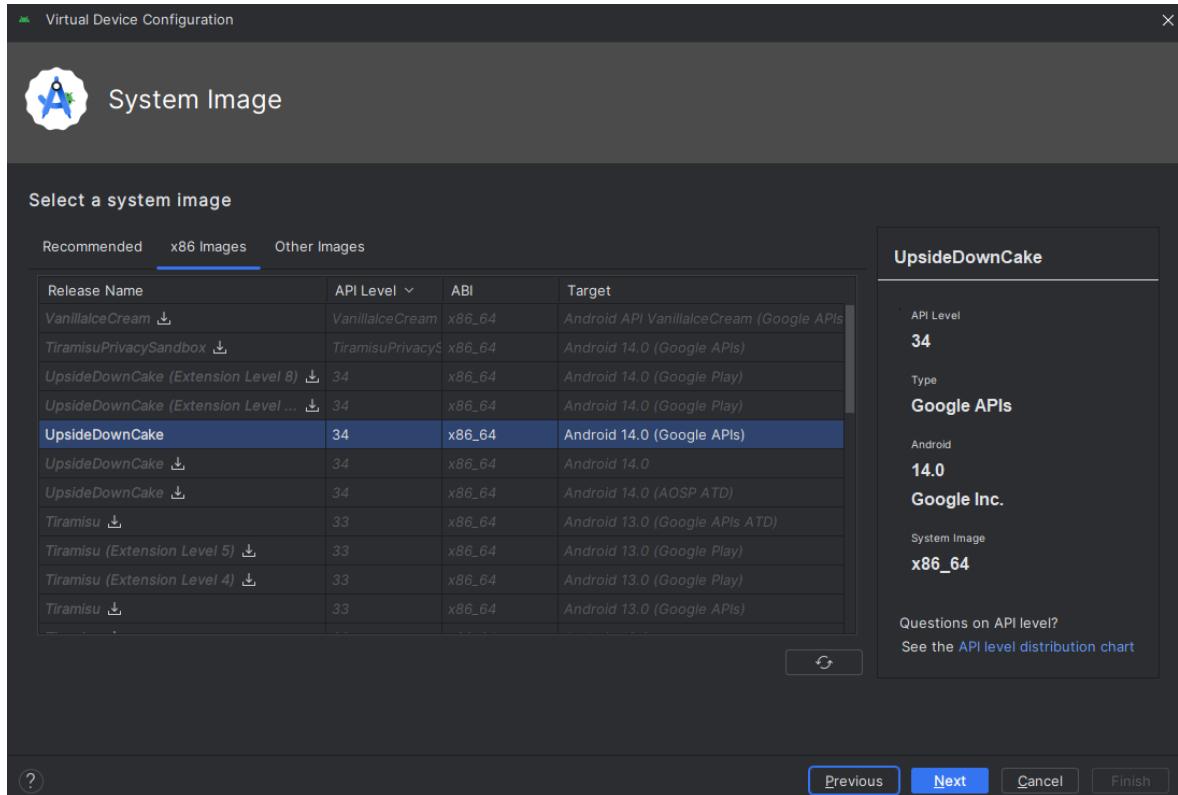


Hình 1. 23. Án vào nút Create Virtual Device để tạo máy ảo

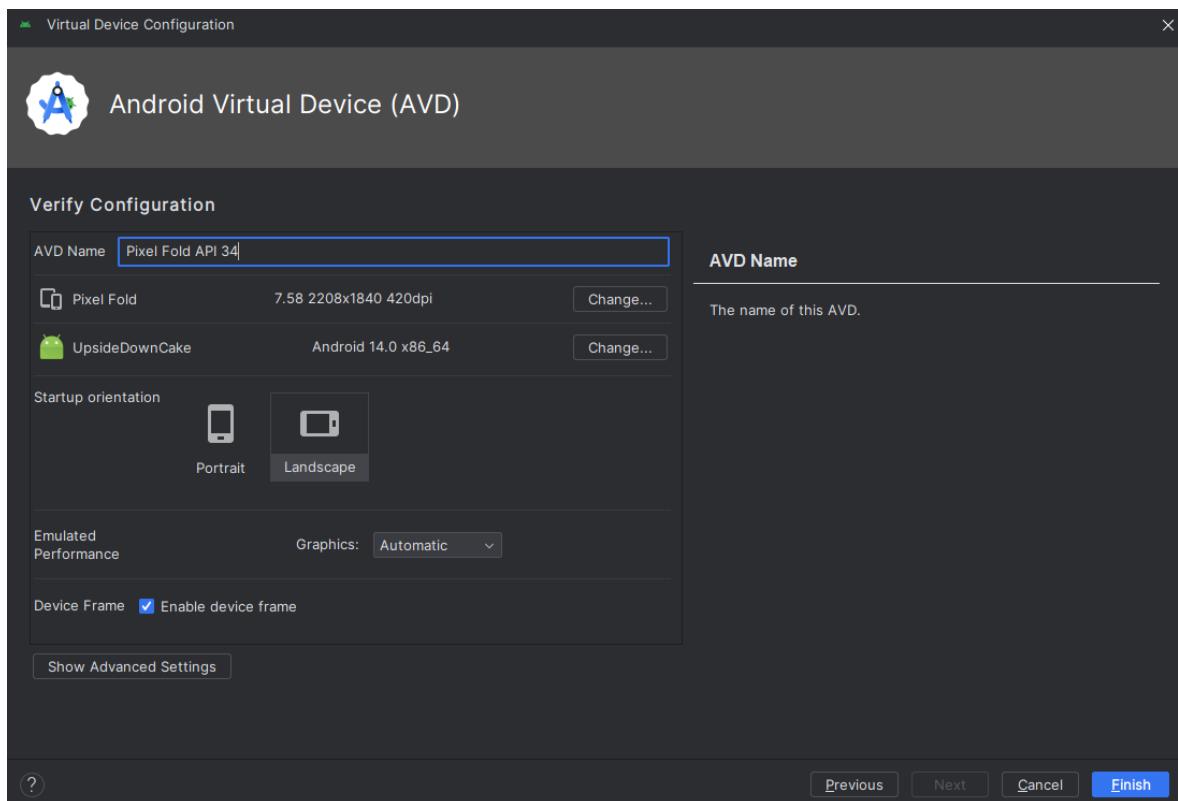
- Tiếp sau đó ta chọn kích thước màn hình, phiên bản android của máy ảo, các tùy biến khác và ánh Finish để hoàn thành



Hình 1. 24. Tùy chọn kích cỡ màn hình máy ảo



Hình 1. 25. Tuỳ chọn phiên bản Android

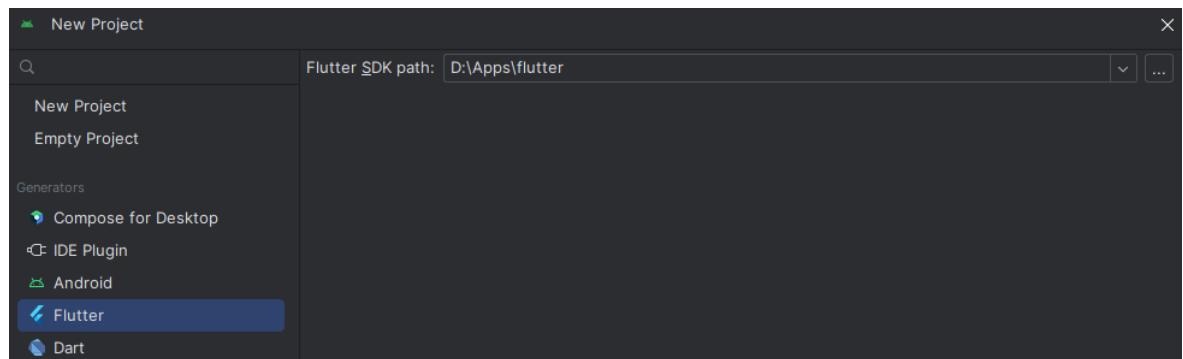


Hình 1. 26. Các tuỳ chọn còn lại và ấn Finish

1.7.3. Khởi chạy chương trình đầu tiên

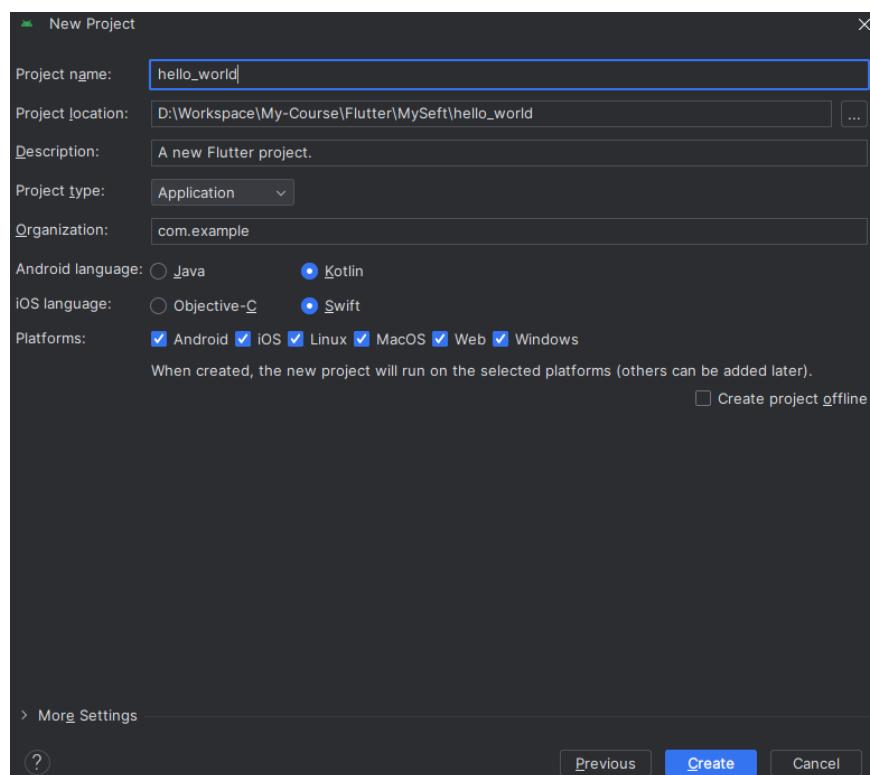
Sau khi hoàn tất quá trình thiết lập môi trường và IDE cho Flutter. Tiếp sau đó ta tiến hành khởi tạo thử chương trình đầu tiên để kiểm tra xem mọi thứ có hoạt động ổn định hay không. Các bước thực hiện được mô tả như sau:

Bước 1: Án vào nút tạo mới Project và chọn vào mục Flutter. Sau đó là bấm next



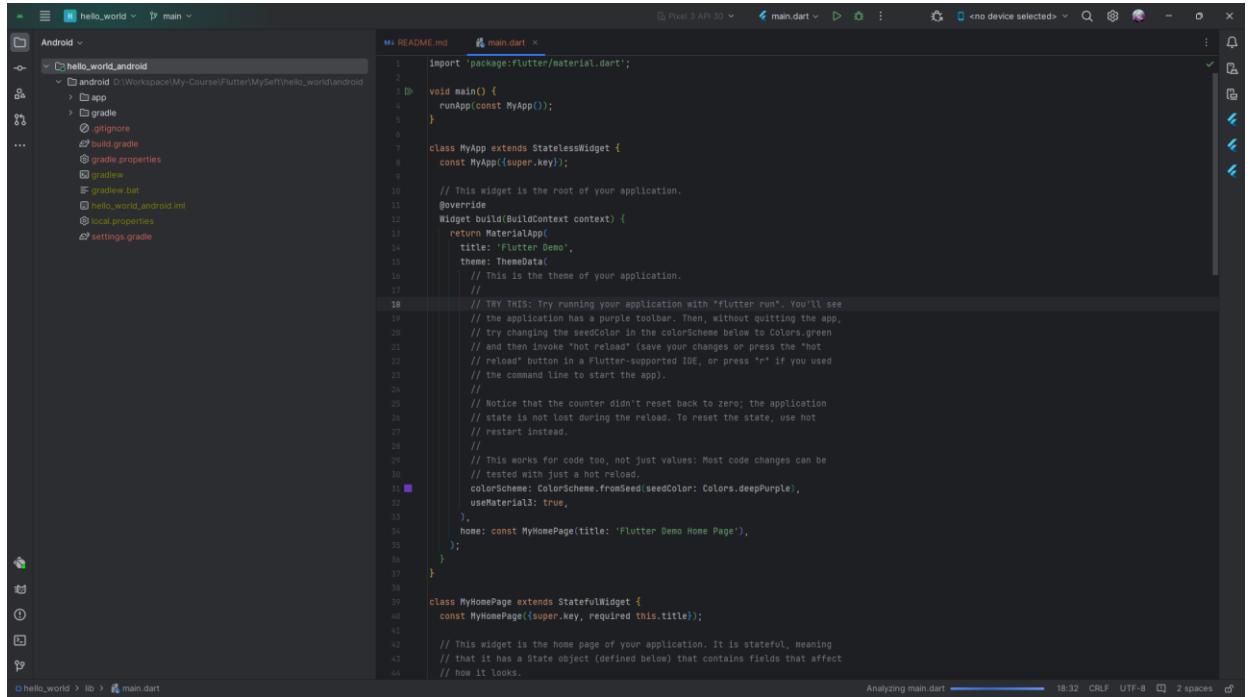
Hình 1. 27. Tạo mới một dự án Flutter

Bước 2: Thiết lập tên dự án, đường dẫn, mô tả loại dự án và các thiết lập khác



Hình 1. 28. Thiết lập tên dự án, đường dẫn, mô tả, loại dự án và các thiết lập khác cho dự án Flutter đầu tiên

Bước 3: Sau khi khởi tạo dự án thành công, ta truy cập vào dự án và thử chạy dự án đó trên máy ảo mà chúng ta đã tạo trước đó

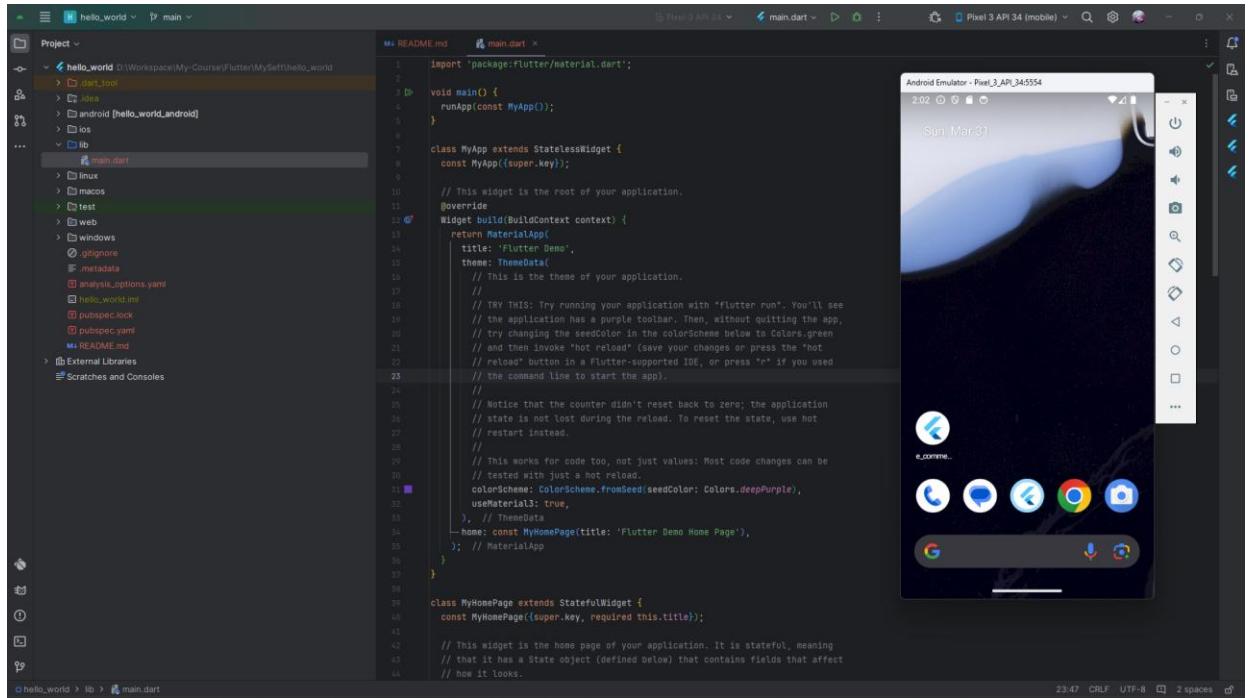


```

1 import 'package:flutter/material.dart';
2
3 void main() {
4   runApp(const MyApp());
5 }
6
7 class MyApp extends StatelessWidget {
8   const MyApp({super.key});
9
10 // This widget is the root of your application.
11 @override
12 Widget build(BuildContext context) {
13   return MaterialApp(
14     title: 'Flutter Demo',
15     theme: ThemeData(
16       // This is the theme of your application.
17       //
18       // TRY THIS: Try running your application with "flutter run". You'll see
19       // the application has a purple toolbar. Then, without quitting the app,
20       // try changing the seedColor in the colorScheme below to Colors.green
21       // and then invoke "hot reload" (save your changes or press the "hot
22       // reload" button in a Flutter-supported IDE, or press "r" if you used
23       // the command line to start the app).
24       //
25       // Notice that the counter didn't reset back to zero; the application
26       // state is not lost during the reload. To reset the state, use hot
27       // restart instead.
28       //
29       // This works for code too, not just values: Most code changes can be
30       // tested with just a hot reload.
31       colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
32       useMaterial3: true,
33     ),
34     home: const MyHomePage(title: 'Flutter Demo Home Page'),
35   );
36 }
37
38 class MyHomePage extends StatefulWidget {
39   const MyHomePage({super.key, required this.title});
40
41   // This widget is the home page of your application. It is stateful, meaning
42   // that it has a State object (defined below) that contains fields that affect
43   // how it looks.
44 }

```

Hình 1. 29. Những dòng code mặc định của một dự án Flutter



```

1 import 'package:flutter/material.dart';
2
3 void main() {
4   runApp(const MyApp());
5 }
6
7 class MyApp extends StatelessWidget {
8   const MyApp({super.key});
9
10 // This widget is the root of your application.
11 @override
12 Widget build(BuildContext context) {
13   return MaterialApp(
14     title: 'Flutter Demo',
15     theme: ThemeData(
16       // This is the theme of your application.
17       //
18       // TRY THIS: Try running your application with "flutter run". You'll see
19       // the application has a purple toolbar. Then, without quitting the app,
20       // try changing the seedColor in the colorScheme below to Colors.green
21       // and then invoke "hot reload" (save your changes or press the "hot
22       // reload" button in a Flutter-supported IDE, or press "r" if you used
23       // the command line to start the app).
24       //
25       // Notice that the counter didn't reset back to zero; the application
26       // state is not lost during the reload. To reset the state, use hot
27       // restart instead.
28       //
29       // This works for code too, not just values: Most code changes can be
30       // tested with just a hot reload.
31       colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
32       useMaterial3: true,
33     ),
34     home: const MyHomePage(title: 'Flutter Demo Home Page'),
35   );
36 }
37
38 class MyHomePage extends StatefulWidget {
39   const MyHomePage({super.key, required this.title});
40
41   // This widget is the home page of your application. It is stateful, meaning
42   // that it has a State object (defined below) that contains fields that affect
43   // how it looks.
44 }

```

Hình 1. 30. Khởi tạo máy ảo cho ứng dụng Flutter đầu tiên

Để có thể chạy một ứng dụng Flutter thì điều tiên quyết phải có đó là các dòng lệnh như sau:

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

void main() {
  runApp(const MyApp());
}
```

Giải thích từng dòng lệnh như sau:

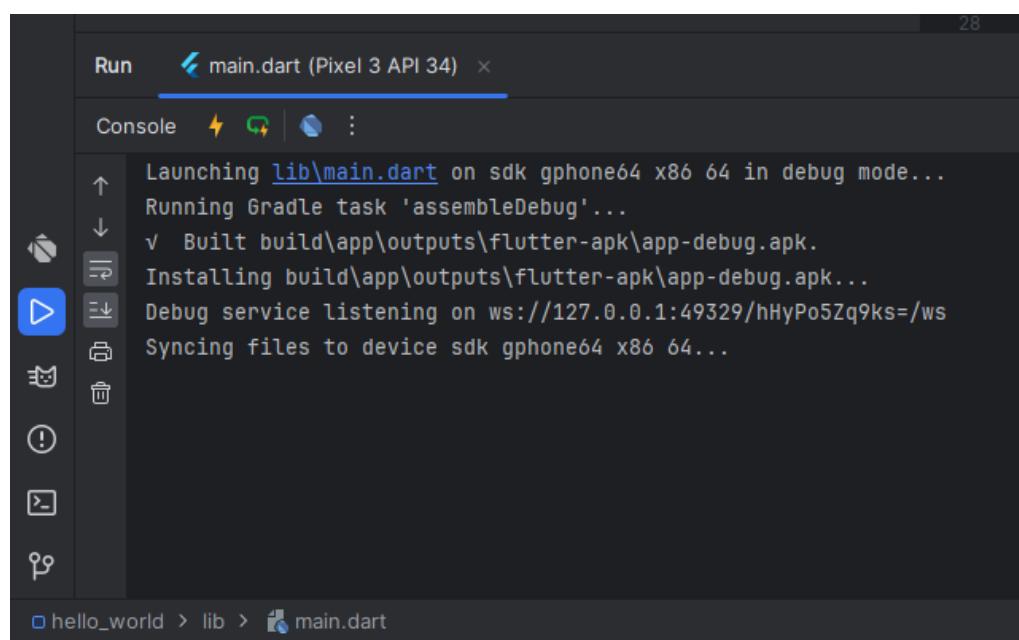
- Dòng lệnh **import 'package:flutter/material.dart'** giúp import thư viện Material vào ứng dụng Flutter. Thư viện Material cung cấp các widget cơ bản để xây dựng giao diện người dùng cho ứng dụng Flutter, bao gồm:
 - o Nút bấm
 - o Text
 - o Hộp thoại
 - o Thanh cuộn
 - o V.v.
- Hàm **main()** là điểm khởi đầu của mọi ứng dụng Flutter. Hàm này thực hiện hai nhiệm vụ chính:
 - o Khởi tạo widget gốc của ứng dụng.
 - o Gọi hàm **runApp()** để hiển thị widget gốc trên màn hình.
- Widget **MyApp()** là widget gốc của ứng dụng Flutter. Widget này chịu trách nhiệm hiển thị giao diện người dùng chính của ứng dụng.

❖ Lý do cần thiết:

- ✚ Các dòng lệnh này cung cấp cho ứng dụng Flutter các thành phần và chức năng cần thiết để hoạt động.
- ✚ Dòng lệnh import thư viện Material cung cấp các widget cơ bản để xây dựng giao diện người dùng.
- ✚ Hàm **main()** là điểm khởi đầu của ứng dụng và chịu trách nhiệm hiển thị giao diện người dùng.
- ✚ Widget **MyApp()** là widget gốc của ứng dụng và hiển thị giao diện người dùng chính.

- Do đó, nếu không có các dòng lệnh này, ứng dụng Flutter sẽ không thể chạy

Để khởi chạy được các dòng lệnh dart trên Android Studio để chúng thực thi lên máy ảo thì ta cần ấn tổ hợp phím **Shift + F10** hoặc ấn vào nút  trên góc phải màn hình. Tiếp sau đó, Android Studio sẽ tự động thực thi biên dịch code và tiến hành cài đặt ứng dụng lên máy ảo:



The screenshot shows the Android Studio interface with the 'Run' tab selected. The title bar says 'main.dart (Pixel 3 API 34)'. The main area is a 'Console' tab showing the following output:

```

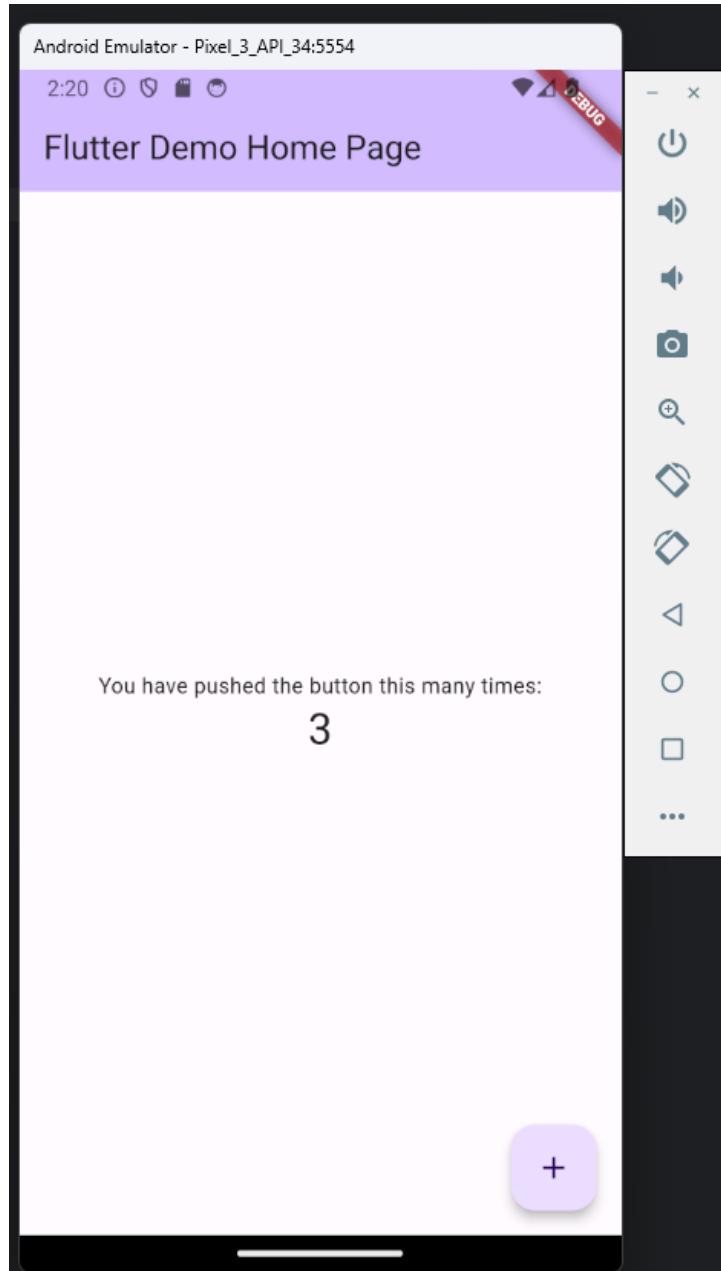
Launching lib\main.dart on sdk gphone64 x86 64 in debug mode...
Running Gradle task 'assembleDebug'...
✓ Built build\app\outputs\flutter-apk\app-debug.apk.
Installing build\app\outputs\flutter-apk\app-debug.apk...
Debug service listening on ws://127.0.0.1:49329/hHyPo5Zq9ks=/ws
Syncing files to device sdk gphone64 x86 64...

```

The left sidebar has various icons for file operations like run, stop, and refresh.

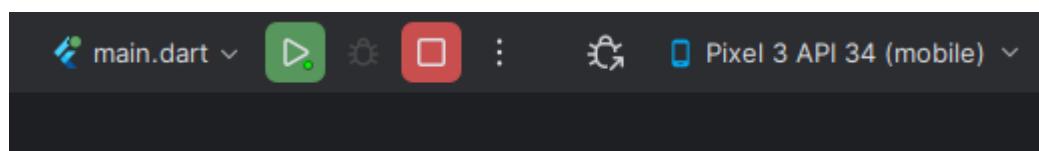
Hình 1. 31. Android Studio biên dịch code và cài đặt ứng dụng Flutter lên máy ảo

Sau khi quá trình biên dịch hoàn tất, máy ảo sẽ khởi động ứng dụng:



Hình 1. 32. Máy ảo khởi động chương trình Flutter đầu tiên

Sau quá trình đó, Android Studio sẽ đặt máy ảo và chương trình trong trạng thái Hot Reload, mỗi khi ta thay đổi dòng code và ấn Save, thì giao diện chương trình cũng sẽ tự động load lại theo:



Hình 1. 33. Android Studio đặt chương trình đang chạy vào trạng thái Hot Reload

CHƯƠNG 2. CHI TIẾT VỀ FLUTTER

2.1. Những kiến thức cơ bản về Flutter

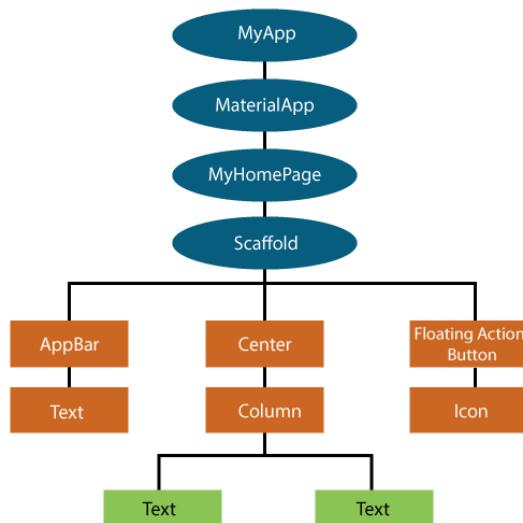
2.1.1. Tổng quan về Widgets và một số Widgets cần phải biết

2.1.1.1. Khái niệm về Widget trong Flutter

Bất cứ khi nào ta viết code để xây dựng bất cứ thứ gì trong Flutter, thì cái mà ta xây dựng được sẽ nằm trong một *Widgets*. Mục đích chính là xây dựng ứng dụng từ các widget. Nó mô tả chế độ xem ứng dụng của ta trông như thế nào với cấu hình và trạng thái hiện tại của chúng. Khi ta thực hiện bất kỳ thay đổi nào trong code, widget con sẽ xây dựng lại mô tả của nó bằng cách tính toán sự khác biệt của widget con hiện tại và trước đó để xác định những thay đổi tối thiểu đối với việc hiển thị trong giao diện người dùng của ứng dụng.

Các widget được lồng vào nhau để xây dựng ứng dụng. Nó có nghĩa là thư mục gốc của ứng dụng tự nó là một widget, và tất cả các cách nhìn xuống cũng là một widget. Ví dụ: một widget có thể hiển thị một thứ gì đó, có thể xác định thiết kế, có thể xử lý tương tác, v.v.

Hình ảnh dưới đây mô tả trực quan đơn giản của một cây Widgets:



Hình 2. 1. Một ví dụ về cây Widgets trong Flutter

Chúng ta có thể tạo một Widget trong Flutter như sau:

```
Class ImageWidget extends StatelessWidget {  
    // Class Body  
}
```

Một ví dụ khác, ví dụ về Hello World:

```
import 'package:flutter/material.dart';  
  
class MyHomePage extends StatelessWidget {  
  
    MyHomePage({Key key, this.title}) : super(key: key);  
  
    // This widget is the home page of your application.  
  
    final String title;  
  
  
    @override  
  
    Widget build(BuildContext context) {  
  
        return Scaffold(  
  
            appBar: AppBar(  
  
                title: Text(this.title),  
  
            ),  
  
            body: Center(  
  
                child: Text('Hello World'),  
  
            ),  
  
        );  
    }  
}
```

2.1.1.2. Các loại Widget con

Ngoài ra, trong Flutter chúng ta có thể chia widget Flutter thành 2 dạng:

1. **Hiển thị** (Visible) (Đầu ra, đầu vào)
2. **Vô hình** (Invisible) (Bố cục và kiểm soát – Layout and Control)

Dạng 1: Một số Widget hiển thị cơ bản có sử dụng đầu ra, đầu vào cho dữ liệu bao gồm:

❖ Text

Widget Text giữ một số văn bản để hiển thị trên màn hình. Chúng ta có thể căn chỉnh widget văn bản bằng cách sử dụng thuộc tính *textAlign* và thuộc tính *style* cho phép tùy chỉnh Text bao gồm phông chữ, độ đậm của phông chữ, kiểu phông chữ, khoảng cách giữa các chữ cái, màu sắc và nhiều hơn nữa. Chúng ta có thể sử dụng nó như các đoạn code dưới đây.

```
// Ví dụ về widget Text
Text(
  'Hello World!',
  textAlign: TextAlign.center,
  style: new TextStyle(fontWeight: FontWeight.bold),
)
```

Hình ảnh minh họa:



Hình 2. 2. Hình ảnh minh họa cho Widget Text

❖ Button

Widget này cho phép ta thực hiện một số hành động khi nhấp chuột. Flutter có sẵn các phiên bản Button khác nhau tuỳ thuộc vào mục đích sử dụng của chúng ta đối

với giao diện mà chúng ta đang xây dựng, Ví dụ như: TextButton, ElevatedButton, OutlinedButton và IconButton. Chúng ta có thể sử dụng nó như các đoạn code dưới đây.

```
// Một số loại widget Button thường được sử dụng

TextButton(
    onPressed: () {
        // Xử lý khi nút được bấm
    },
    child: Text('Nút TextButton'),
)

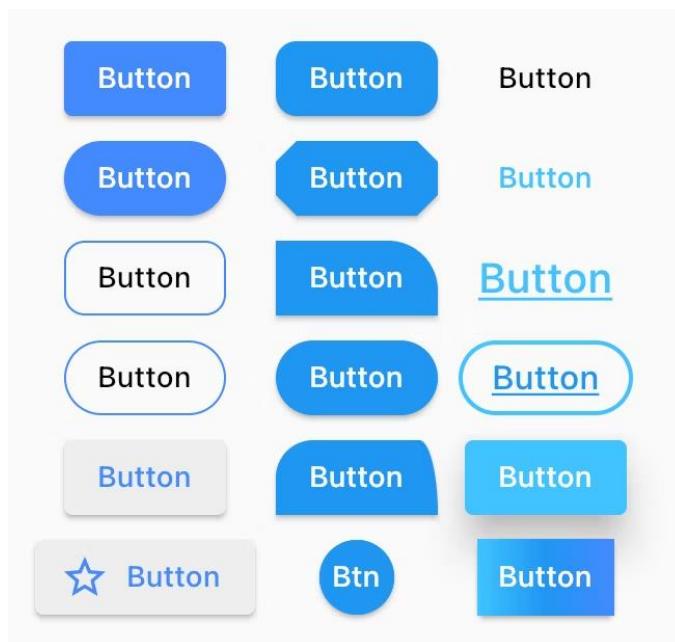
ElevatedButton(
    onPressed: () {
        // Xử lý khi nút được bấm
    },
    child: Text('Nút ElevatedButton'),
)

OutlinedButton(
    onPressed: () {
        // Xử lý khi nút được bấm
    },
    child: Text('Nút OutlinedButton'),
)
```

```
IconButton(
    onPressed: () {
        // Xử lý khi nút được bấm
    },
    icon: Icon(Icons.add),
)
```

Trong đoạn code ví dụ trên, các widget Button luôn luôn có thuộc tính *onPressed()* để thực hiện hành động sau khi ta ấn vào Button và một thuộc tính child hoặc thuộc tính tương ứng với loại Button để chứa nội dung hiển thị của Button đó.

Hình ảnh minh họa:



Hình 2. 3. Hình ảnh minh họa cho một số Widget Button

❖ Image

Widget này giữ hình ảnh có thể tìm nạp hình ảnh từ nhiều nguồn như từ thư mục nội dung hoặc trực tiếp từ URL. Nó cung cấp nhiều hàm tạo để tải hình ảnh, được đưa ra dưới đây:

- Hình ảnh(Image): Đây là một trình tải hình ảnh chung, được sử dụng bởi *ImageProvider*.

- Tài sản(asset): Nó tải hình ảnh từ thư mục tài sản dự án của ta.
- tệp(file): Nó tải hình ảnh từ thư mục hệ thống.
- bộ nhớ(memory): Nó tải hình ảnh từ bộ nhớ.
- mạng(network): Nó tải hình ảnh từ mạng.

Để thêm hình ảnh vào dự án, trước tiên ta cần tạo một thư mục nội dung nơi ta lưu giữ hình ảnh của mình và sau đó thêm dòng bên dưới vào tệp *pubspec.yaml* như sau:

```
assets:
  - <Đường dẫn tới thư mục chứa ảnh>
```

```
45 ----- LOCAL ASSETS -----
46 assets:
47   - assets/logos/
48   - assets/icons/brands/
49   - assets/images/on_boarding_images/
50   - assets/images/animations/
51
52 ----- LOCAL FONTS -----
53 fonts:
54   - family: Poppins
55     fonts:
56       - asset: assets/fonts/Poppins-Light.ttf
57         weight: 300
58       - asset: assets/fonts/Poppins-Regular.ttf
59         weight: 400
60       - asset: assets/fonts/Poppins-Italic.ttf
```

Hình 2. 4. Đường dẫn hình ảnh cấu hình trong *pubspec.yaml* trong dự án thực tế

Code nguồn hoàn chỉnh để thêm hình ảnh được hiển thị bên dưới trong ví dụ hello world:

```
class MyHomePage extends StatelessWidget {
  MyHomePage({Key key, this.title}) : super(key: key);
  // This widget is the home page of your application.
  final String title;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
```

```
appBar: AppBar(  
    title: Text(this.title),  
,  
    body: Center(  
        child: Image.asset('assets/computer.png'),  
,  
    );  
,  
},  
)
```

Khi ta chạy ứng dụng, ta sẽ nhận được kết quả như sau:



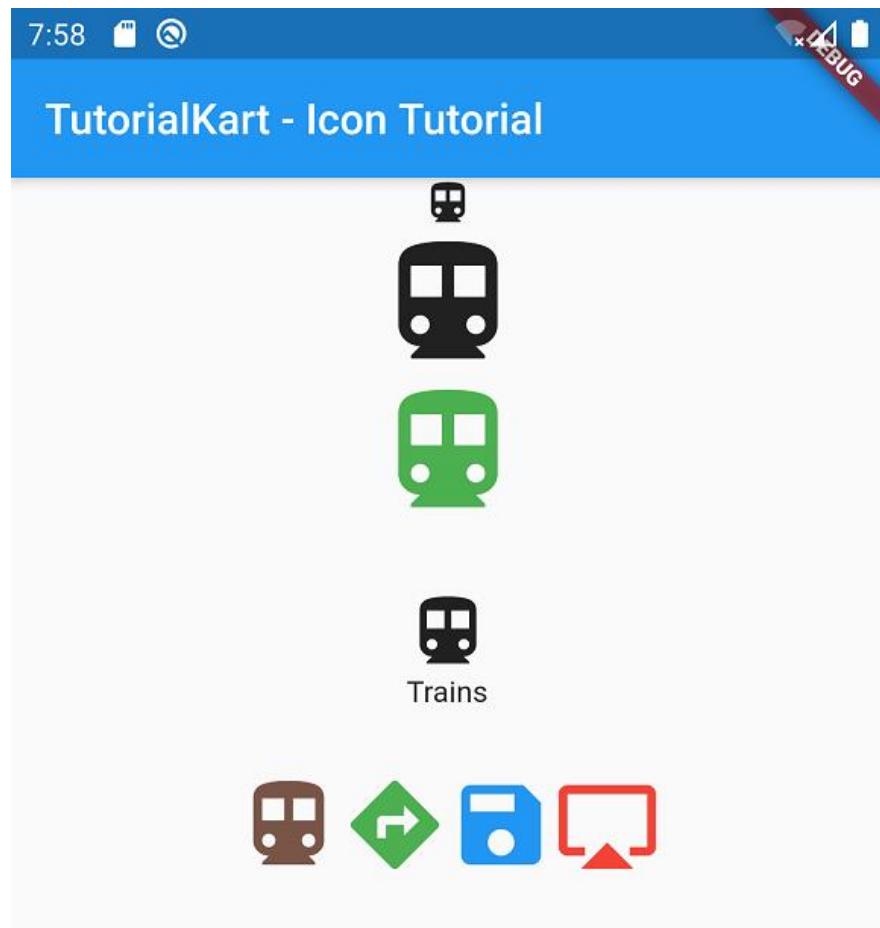
Hình 2. 5. Hình ảnh minh họa cho Widget Image

❖ Image

Widget này hoạt động như một thùng chứa để lưu trữ Biểu tượng trong Flutter. Đoạn code sau đây giải thích rõ ràng hơn.

```
Icon(  
  Icons.add,  
  size: 34.0,  
)
```

Hình ảnh minh họa:



Hình 2. 6. Hình ảnh minh họa cho Widget Icon

Dạng 2: Một số Widget dạng ẩn, liên quan đến việc bố trí và hiển thị các dạng bố cục lên màn hình:

❖ Column

Widget Column là một widget layout cơ bản trong Flutter, dùng để sắp xếp các widget con theo chiều dọc, từ trên xuống dưới. Nó đóng vai trò quan trọng trong

việc xây dựng giao diện người dùng (UI) cho các ứng dụng Flutter. Ta có đoạn code ví dụ như sau đây:

```
Column(
    mainAxisAlignment: MainAxisAlignment.center,
    crossssAxisAlignment: CrossAxisAlignment.center,
    mainAxisSize: MainAxisSize.minContent,
    children: [
        Text("Item1"),
        Text("Item2"),
        Text("Item3"),
    ],
),
```

Các thuộc tính quan trọng của Column như sau:

- **children:** Danh sách các widget con được sắp xếp theo chiều dọc.
- **mainAxisAlignment:** Xác định cách sắp xếp các widget con theo chiều dọc.

Các giá trị phổ biến bao gồm:

- **start:** Sắp xếp các widget con ở đầu (top) của Column.
- **center:** Sắp xếp các widget con ở giữa (center) của Column.
- **end:** Sắp xếp các widget con ở cuối (bottom) của Column.
- **spaceBetween:** Tạo khoảng cách đều giữa các widget con.
- **spaceAround:** Tạo khoảng cách bằng nhau giữa các widget con và hai đầu của Column

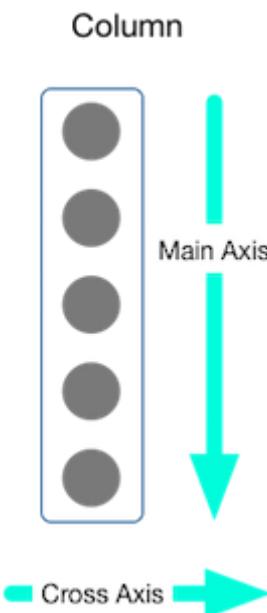
- **crossAxisAlignment:** Xác định cách sắp xếp các widget con theo chiều ngang.

Các giá trị phổ biến bao gồm:

- **start:** Sắp xếp các widget con ở cạnh trái (start) của Column.
- **center:** Sắp xếp các widget con ở giữa (center) của Column.
- **end:** Sắp xếp các widget con ở cạnh phải (end) của Column.

- **stretch:** Mở rộng các widget con để lấp đầy chiều ngang của Column.
- **mainAxisSize:** Xác định kích thước tối đa của Column theo chiều dọc. Các giá trị phổ biến bao gồm:
 - **MainAxisSize.minContent:** Kích thước tối thiểu cần thiết để chứa các widget con.
 - **MainAxisSize.maxContent:** Mở rộng để chứa tất cả các widget con.
 - **MainAxisSize.fill:** Mở rộng để lấp đầy chiều cao có sẵn.

Hình ảnh minh họa:



Hình 2. 7. Hình ảnh minh họa cho Widget Column

❖ Row

Widget Row là một widget layout cơ bản trong Flutter, dùng để sắp xếp các widget con theo chiều ngang, từ trái sang phải. Nó đóng vai trò quan trọng trong việc xây dựng giao diện người dùng (UI) cho các ứng dụng Flutter. Ta có đoạn code ví dụ như sau đây:

```
Row (
  mainAxisSize: MainAxisSize.center,
  crossssAxisAlignment: CrossAxisAlignment.center,
```

```
mainAxisSize: MainAxisSize.minContent,
children: [
    Text("Item1"),
    Text("Item2"),
    Text("Item3"),
],
),
```

Các thuộc tính quan trọng của Row như sau:

- **children:** Danh sách các widget con được sắp xếp theo chiều ngang.
- **mainAxisAlignment:** Xác định cách sắp xếp các widget con theo chiều ngang.

Các giá trị phổ biến bao gồm:

- **start:** Sắp xếp các widget con ở đầu (top) của Row.
- **center:** Sắp xếp các widget con ở giữa (center) của Row.
- **end:** Sắp xếp các widget con ở cuối (bottom) của Row.
- **spaceBetween:** Tạo khoảng cách đều giữa các widget con.
- **spaceAround:** Tạo khoảng cách bằng nhau giữa các widget con và hai đầu của Row

- **crossAxisAlignment:** Xác định cách sắp xếp các widget con theo chiều dọc.

Các giá trị phổ biến bao gồm:

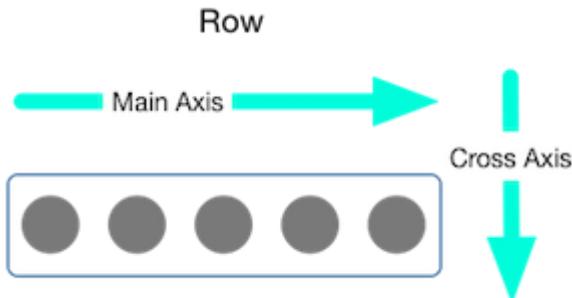
- **start:** Sắp xếp các widget con ở cạnh trái (start) của Row.
- **center:** Sắp xếp các widget con ở giữa (center) của Row.
- **end:** Sắp xếp các widget con ở cạnh phải (end) của Row.
- **stretch:** Mở rộng các widget con để lấp đầy chiều ngang của Row.

- **mainAxisSize:** Xác định kích thước tối đa của Row theo chiều dọc. Các giá trị phổ biến bao gồm:

- **MainAxisSize.minContent:** Kích thước tối thiểu cần thiết để chứa các widget con.
- **MainAxisSize.maxContent:** Mở rộng để chứa tất cả các widget con.

- **MainAxisSize.fill:** Mở rộng để lấp đầy chiều cao có sẵn.

Hình ảnh minh họa:



Hình 2. 8. Hình ảnh minh họa cho Widget Row

Ngoài ra, ta còn có các Widget thường được sử dụng cho việc căn chỉnh, đệm, căn lề cho các Widget con như: **Padding, Margin, Center...** Ta có các dòng code ví dụ như sau:

```
// Một số cách để sử dụng Widget dạng căn chỉnh

// Cách 1: Sử dụng bên trong 1 Widget cha
Center(
    child: Container(
        padding: EdgeInsets.all(10),
        margin: EdgeInsets.all(10),
        child: Text('Hello World'),
    ),
);

// Cách 2: Sử dụng như một Widget thông thường
// Có thể áp dụng cho cả Margin và Center
Padding(
    padding: EdgeInsets.all(10),
```

```

        child: Text('Hello World'),
    )
}

```

Bên cạnh các Widget vừa được giới thiệu trên thì Flutter còn có rất nhiều dạng Widget khác liên quan tới việc căn chỉnh các Widget con như **Align**, **SizedBox**, **Stack**, **CustomPaint**.... Do bài báo cáo này có giới hạn nên không thể trình bày chi tiết hết tất cả, có thể tìm hiểu thêm tại <https://docs.flutter.dev/>.

2.1.1.3. Phân loại Widget State

State là một phần cốt lõi của Flutter, đóng vai trò quan trọng trong việc tạo ra các giao diện người dùng (UI) động và phản hồi. State đại diện cho dữ liệu mà widget hiển thị và thay đổi theo thời gian. Việc quản lý state hiệu quả là chìa khóa để xây dựng các ứng dụng Flutter linh hoạt và mượt mà.

Hai phương pháp chính để quản lý state trong Flutter:

1. StatelessWidget
2. StatefulWidget

❖ StatefulWidget

StatefulWidget có thông tin state. Nó chủ yếu chứa hai lớp: **state object** và **widget**. Nó là động vì nó có thể thay đổi dữ liệu bên trong suốt thời gian tồn tại của widget. widget con này không có phương thức **build()**. Nó có phương thức **createState()**, trả về một lớp mở rộng Lớp **state** của Flutter. Các ví dụ của StatefulWidget là *Checkbox*, *Radio*, *Slider*, *InkWell*, *Form* và *TextField*.

StatefulWidget thường được sử dụng cho các widget có state thay đổi theo thời gian, cần cập nhật giao diện khi state thay đổi. Nó đồng thời cung cấp phương thức **setState()** để cập nhật state của widget, dẫn đến việc xây dựng lại widget và hiển thị thay đổi trên giao diện

```

// Một ví dụ cho StatefulWidget

class CounterWidget extends StatefulWidget {
    @override

```

```
_CounterWidgetState createState() => _CounterWidgetState();  
}  
  
class _CounterWidgetState extends State<CounterWidget> {  
    int _counter = 0;  
  
    void incrementCounter() {  
        setState(() {  
            _counter++;  
        });  
    }  
  
    @override  
    Widget build(BuildContext context) {  
        return Column(  
            children: [  
                Text('Số lượng: $_counter'),  
                ElevatedButton(  
                    onPressed: incrementCounter,  
                    child: Text('Tăng'),  
                ),  
            ],  
        );  
    }  
}
```

Khởi code trên sẽ tạo ra một nút bấm và một đoạn chữ với giá trị `_counter` tương ứng với số lần được ấn vào trong nút bấm, mỗi lần ta bấm vào nút bấm thì sẽ gọi lại hàm `incrementCounter()` và `setState()` sẽ kích hoạt việc xây dựng lại Widget. Do đó phương thức `build` sẽ được gọi lại và giá trị `_counter` cập nhật lại trong widget text (mỗi lần bấm sẽ tăng biến counter lên 1).

❖ StatelessWidget

Stateless Widget là một widget trong Flutter không lưu trữ trạng thái và không thay đổi giao diện theo thời gian. Nó chỉ hiển thị dữ liệu dựa trên thông tin được truyền vào. Stateless widget được sử dụng cho các trường hợp hiển thị nội dung tĩnh, không cần cập nhật theo tương tác của người dùng hoặc thay đổi dữ liệu bên ngoài.

Đặc điểm chính của Stateless Widget:

- Không lưu trữ trạng thái: Stateless widget không có biến lưu trữ trạng thái thay đổi theo thời gian.
- Giao diện tĩnh: Giao diện của stateless widget chỉ được xác định dựa trên dữ liệu được truyền vào và không thay đổi khi widget được xây dựng lại.
- Hiệu quả: Stateless widget thường hiệu quả hơn stateful widget vì nó không cần cập nhật giao diện thường xuyên.
- Dễ sử dụng: Stateless widget dễ sử dụng và hiệu hơn stateful widget vì nó không cần quản lý trạng thái.

Cách sử dụng Stateless Widget:

1. **Khai báo lớp:** Tạo một lớp kế thừa từ `StatelessWidget`.
2. **Constructor:** Định nghĩa constructor để nhận dữ liệu đầu vào.
3. **Phương thức build:** Phân phối widget con và xác định giao diện widget.
4. **Sử dụng:** Tạo instance của stateless widget và truyền dữ liệu vào constructor.

```
// Một ví dụ cho StatelessWidget
```

```
class My StatelessWidget extends StatelessWidget {  
  final String title;  
  final String content;  
  
  const My StatelessWidget({  
    Key? key,  
    required this.title,  
    required this.content,  
  }) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    return Column(  
      children: [  
        Text(  
          title,  
          style: const TextStyle(fontSize: 24, fontWeight: FontWeight.bold),  
        ),  
        const SizedBox(height: 16),  
        Text(  
          content,  
          style: const TextStyle(fontSize: 16),  
        ),  
      ],  
    );  
  }  
}
```

```

    }
}
```

Khối code trên sẽ tạo cho ta một giao diện tĩnh gồm một cột **Column** và 2 đoạn **Text**. So với StatelessWidget thì ta có một bảng so sánh như sau:

Đặc điểm	StatelessWidget	StatefulWidget
Trạng thái	Không lưu trữ	Lưu trữ và quản lý
Giao diện	Tĩnh	Thay đổi theo thời gian
Hiệu quả	Hiệu quả hơn	Kém hiệu quả hơn
Độ khó	Dễ sử dụng	Khó sử dụng hơn

Vậy sau khi trình bày được cả 2 cách quản lý state, ta đúc kết được như sau:

✚ Kết luận:

- Sử dụng **StatelessWidget** cho các trường hợp hiển thị nội dung tĩnh, không cần cập nhật theo thời gian.
- Sử dụng **StatefulWidget** cho các trường hợp cần hiển thị nội dung thay đổi theo thời gian, ví dụ như bộ đếm số, danh sách thay đổi ...

2.1.2. Bộ cục giao diện (Layout)

2.1.2.1. Khái niệm bộ cục và bố trí trong Flutter

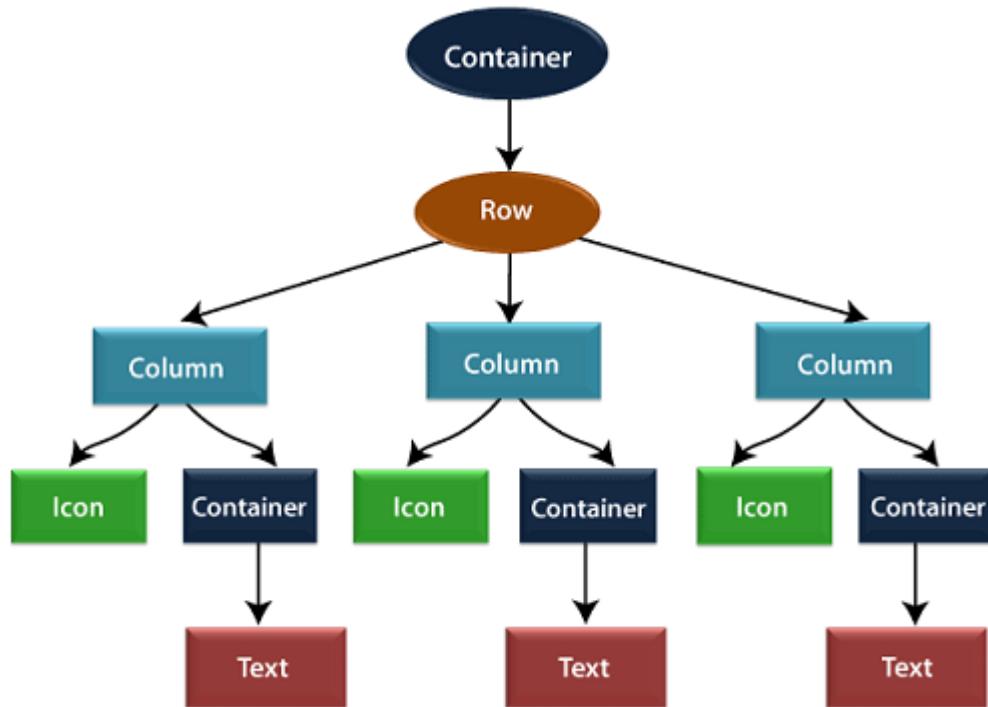
Khái niệm chính của cơ chế bộ trí là widget. Chúng ta biết rằng sự flutter giả định mọi thứ như một widget. Vì vậy, hình ảnh, biểu tượng, văn bản và thậm chí cả bộ cục(layout) của ứng dụng của ta đều là widget. Ở đây, một số thứ ta không thấy trên giao diện người dùng ứng dụng của mình, chẳng hạn như các hàng (row), cột (column) và lưới sắp xếp (grid layout), ràng buộc (constant) và căn chỉnh các widget hiển thị cũng là các widget.

Flutter cho phép chúng ta tạo bố cục bằng cách soạn nhiều widget để xây dựng các widget phức tạp hơn. Ví dụ , chúng ta có thể thấy hình ảnh dưới đây hiển thị ba biểu tượng với nhãn bên dưới mỗi biểu tượng:



Hình 2. 9. Sắp xếp bố cục trong Flutter

Trong hình ảnh thứ hai, chúng ta có thể thấy bố cục trực quan của hình ảnh trên. Hình ảnh này hiển thị một hàng gồm ba cột và các cột này chứa một biểu tượng và nhãn:



Hình 2. 10. Minh họa về cây bố cục trong Flutter

Trong hình trên, vùng chứa là một lớp widget cho phép chúng ta tùy chỉnh widget con. Nó chủ yếu được sử dụng để thêm đường viền, đệm, lề, màu nền và nhiều thứ khác. Tại đây, widget văn bản nằm dưới vùng chứa để thêm lề. Toàn bộ hàng cũng được đặt trong một vùng chứa để thêm lề và phần đệm xung quanh hàng. Ngoài

ra, phần còn lại của giao diện người dùng được kiểm soát bởi các thuộc tính như màu sắc, kiểu văn bản, v.v.

Các bước sau đây cho biết cách bố trí Widget trong Flutter:

- **Bước 1:** Đầu tiên, ta cần chọn một Bố cục widget.
 - Bước đầu tiên là lựa chọn loại bố cục widget phù hợp cho giao diện ta muốn tạo. Flutter cung cấp nhiều widget bố cục khác nhau để sắp xếp và định vị các widget con, ví dụ như:
 - **Row:** Sắp xếp widget con theo chiều ngang (từ trái sang phải).
 - **Column:** Sắp xếp widget con theo chiều dọc (từ trên xuống dưới).
 - **Stack:** Xếp chồng các widget con lên nhau.
 - **Wrap:** Bọc các widget con trong một dòng hoặc cột, tự điều chỉnh kích thước theo không gian có sẵn.
 - **Center:** Đặt widget con ở trung tâm của widget cha.
 - Lựa chọn bố cục phù hợp phụ thuộc vào nhu cầu cụ thể và cấu trúc giao diện ta muốn xây dựng.
- **Bước 2:** Tiếp theo, tạo một widget hiển thị.
 - Sau khi chọn bố cục, ta cần tạo một hoặc nhiều widget để hiển thị nội dung mong muốn. Flutter cung cấp nhiều loại widget khác nhau để hiển thị văn bản, hình ảnh, nút bấm, v.v.
 - Ví dụ: sử dụng widget **Text** để hiển thị văn bản, **Image** để hiển thị hình ảnh, **ElevatedButton** để tạo nút bấm.
 - Mỗi widget có các thuộc tính riêng để định cấu hình giao diện và hành vi của nó.
 - Ta cần thiết lập các thuộc tính phù hợp để widget hiển thị nội dung và chức năng như mong muốn.
- **Bước 3:** Sau đó, thêm widget hiển thị vào widget layout.

- Sau khi tạo widget hiển thị, ta cần thêm chúng vào widget bố cục đã chọn ở Bước 1.
- Mỗi widget bố cục có phương thức hoặc thuộc tính để thêm widget con
 - Ví dụ: *children* cho **Row**, **Column**, **Stack**, **Wrap**, v.v.
- **Bước 4:** Cuối cùng, thêm widget bố cục vào trang mà ta muốn hiển thị.
 - Bước cuối cùng là thêm widget bố cục chứa các widget hiển thị vào trang mà ta muốn hiển thị.
 - Flutter cung cấp nhiều loại widget trang khác nhau, ví dụ như **Scaffold**, **MaterialApp**, v.v.
 - Widget trang thường là widget gốc của giao diện ứng dụng.
 - Thêm widget bố cục vào widget trang sẽ hiển thị nội dung của widget con trên màn hình

Ta có một đoạn code minh họa như sau:

```
// Widget hiển thị văn bản "Hello, Flutter!"

Text(
  'Hello, Flutter!',
  style: TextStyle(fontSize: 24),
)

// Widget bố cục Column

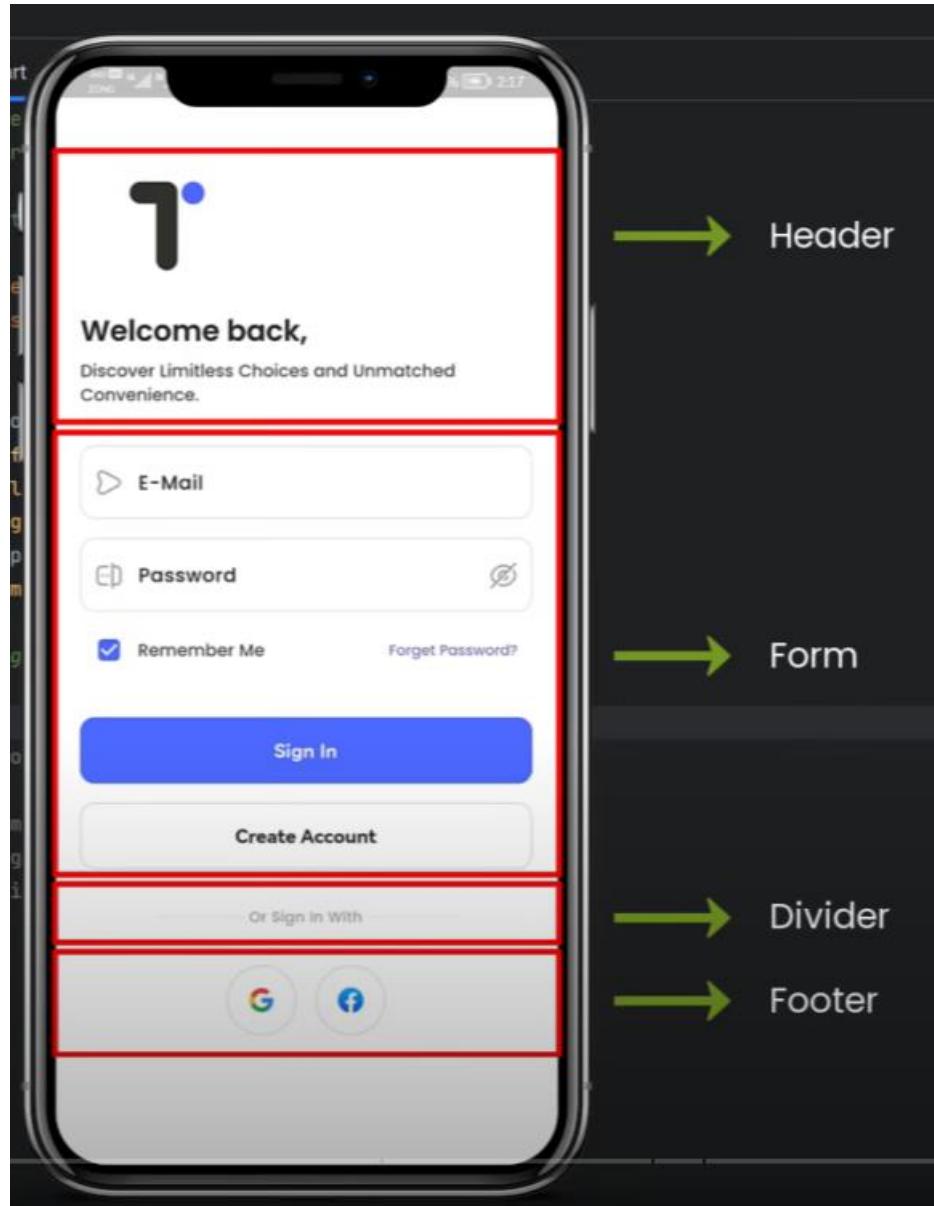
Column(
  children: [
    // Widget hiển thị văn bản "Hello, Flutter!" được thêm vào bố cục
    Text(
      'Hello, Flutter!',
      style: TextStyle(fontSize: 24),
    )
  ],
)
```

```
),
],
)

// Widget trang Scaffold hiển thị widget bố cục Column
Scaffold(
  appBar: AppBar(
    title: Text('Ví dụ Flutter'),
  ),
  body: Column(
    children: [
      // Widget bố cục Column được thêm vào widget trang Scaffold
      Column(
        children: [
          Text(
            'Hello, Flutter!',
            style: TextStyle(fontSize: 24),
          ),
        ],
      ),
    ],
  ),
)
```

Tóm lại, việc tạo và hiển thị widget đơn giản trong Flutter bao gồm các bước cơ bản: chọn bố cục widget, tạo widget hiển thị, thêm widget hiển thị vào bố cục, và

thêm bộ cục widget vào trang. Hiểu rõ các bước này và sử dụng các widget phù hợp sẽ giúp ta xây dựng giao diện người dùng Flutter hiệu quả và đáp ứng nhu cầu thiết kế.



Hình 2. 11. Một ví dụ về xây dựng bộ cục của một ứng dụng trong thực tế

2.1.2.2. Một số Widget bộ cục

Trong Flutter, ta phân loại Widget bộ cục thành 2 loại:

1. widget đơn
2. widget đa

❖ Widget đơn

Widget bô cục con duy nhất là một loại widget, có thể chỉ có **một widget** bên trong widget bô cục mẹ. Các widget này cũng có thể chứa chức năng bô cục đặc biệt. Flutter cung cấp cho chúng ta nhiều widget con để làm cho giao diện người dùng của ứng dụng trở nên hấp dẫn. Nếu chúng ta sử dụng các widget này một cách thích hợp, nó có thể tiết kiệm thời gian của chúng ta và làm cho code ứng dụng dễ đọc hơn. Danh sách các loại widget đơn lẻ khác nhau là:

1. **Container:** Đây là widget bô cục phổ biến nhất cung cấp các tùy chọn có thể tùy chỉnh để đặt màu, định vị và định cỡ các widget.

```
Center(
    child: Container(
        margin: const EdgeInsets.all(15.0),
        color: Colors.blue,
        width: 42.0,
        height: 42.0,
    ),
)
```

2. **Padding:** Nó là một widget được sử dụng để sắp xếp widget con của nó theo khoảng đệm đã cho. Nó chứa *EdgeInsets* là một lớp trong Flutter đại diện cho các khoảng cách (padding và margin) xung quanh một widget, cho phia mong muốn mà ta muốn cung cấp đệm.

```
const Greetings(
    child: Padding(
        padding: EdgeInsets.all(14.0),
        child: Text('Hello Cafedev!'),
    ),
)
```

```
)
```

3. **Center:** widget này cho phép ta căn giữa widget trong chính nó.
4. **Align:** Đây là một widget, căn chỉnh widget của nó trong chính nó và định kích thước nó dựa trên kích thước của đối tượng con. Nó cung cấp nhiều quyền kiểm soát hơn để đặt widget ở vị trí chính xác mà ta cần.

```
Center(
```

```
    child: Container(
```

```
        height: 110.0,
```

```
        width: 110.0,
```

```
        color: Colors.blue,
```

```
        child: Align(
```

```
            alignment: Alignment.topLeft,
```

```
            child: FlutterLogo(
```

```
                size: 50,
```

```
            ),
```

```
        ),
```

```
        ),
```

```
)
```

5. **SizedBox:** widget này cho phép ta cung cấp kích thước được chỉ định cho widget thông qua tất cả các màn hình.

```
SizedBox(
```

```
    width: 300.0,
```

```
    height: 450.0,
```

```
    child: const Card(child: Text('Hello Cafedev!')),
```

```
)
```

6. **AspectRatio:** widget này cho phép ta giữ kích thước của widget theo một tỷ lệ khung hình được chỉ định.

```
AspectRatio(
  aspectRatio: 5/3,
  child: Container(
    color: Colors.blue,
  ),
),
```

7. **Baseline:** widget này thay đổi widget theo đường cơ sở của widget con bên trong.

```
child: Baseline(
  baseline: 30.0,
  baselineType: TextBaseline.alphabetic,
  child: Container(
    height: 60,
    width: 50,
    color: Colors.blue,
  ),
)
```

8. **ConstrainedBox:** Đây là một widget cho phép ta buộc các ràng buộc bổ sung lên widget con của nó. Nó có nghĩa là ta có thể buộc widget con có một ràng buộc cụ thể mà không làm thay đổi các thuộc tính của widget con.

```
ConstrainedBox(
  constraints: new BoxConstraints(
    minHeight: 150.0,
```

```

minWidth: 150.0,
maxHeight: 300.0,
maxWidth: 300.0,
),
child: new DecoratedBox(
decoration: new BoxDecoration(color: Colors.red),
),
),
),

```

9. **CustomSingleChildLayout:** Nó là một widget, chuyển từ bô cục của con đơn thành một delegate. Người được ủy quyền (delegate) quyết định vị trí của widget và cũng được sử dụng để xác định kích thước của widget.
10. **FittedBox:** Nó chia tỷ lệ và định vị widget theo sự phù hợp được chỉ định.

```

import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {
  // It is the root widget of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Multiple Layout Widget',
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        // This is the theme of your application.

```

```
primarySwatch: Colors.green,  
)  
home: MyHomePage(),  
);  
}  
}  
  
class MyHomePage extends StatelessWidget {  
  
@override  
Widget build(BuildContext context) {  
return Scaffold(  
appBar: AppBar(title: Text("FittedBox Widget")),  
body: Center(  
child: FittedBox(child: Row(  
children: <Widget>[  
Container(  
child: Image.asset('assets/computer.png'),  
),  
Container(  
child: Text("This is a widget"),  
)  
],  
),  
fit: BoxFit.contain,  
)
```

```
    ),  
    );  
}  
}
```



Hình 2. 12. Ví dụ về cách sử dụng Widget FittedBox

11. **FractionallySizedBox:** Nó là một widget cho phép kích thước của widget con của nó theo phần nhỏ của không gian có sẵn.
12. **LimitedBox:** widget này cho phép chúng ta giới hạn kích thước của nó chỉ khi nó không bị giới hạn.
13. **Offstage:** Nó được sử dụng để đo kích thước của một widget mà không cần đưa nó lên màn hình.

14. **OverflowBox:** Nó là một widget, cho phép áp đặt các ràng buộc khác nhau đối với widget con của nó so với nó nhận được từ cha mẹ. Nói cách khác, nó cho phép con làm tràn widget cha.

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

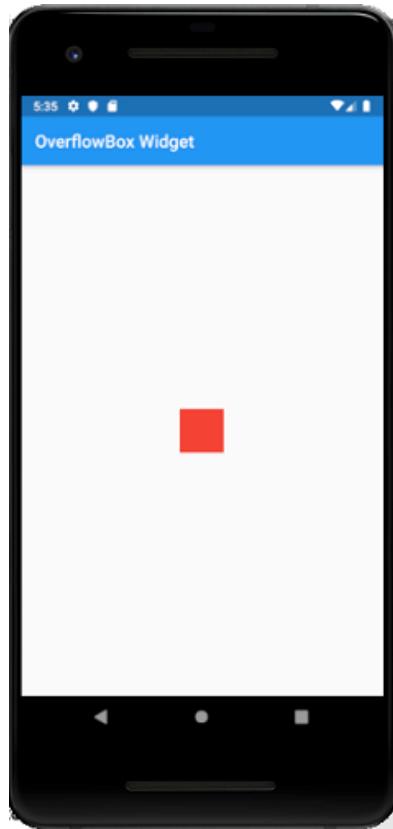
void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
    // It is the root widget of your application.

    @override
    Widget build(BuildContext context) {
        return MaterialApp(
            title: 'Single Layout Widget',
            debugShowCheckedModeBanner: false,
            theme: ThemeData(
                // This is the theme of your application.
                primarySwatch: Colors.blue,
            ),
            home: MyHomePage(),
        );
    }
}

class MyHomePage extends StatelessWidget {
    // This class is the home screen of your application.
    // It is stateless.
    @override
}
```

```
Widget build(BuildContext context) {  
  return Scaffold(  
    appBar: AppBar(  
      title: Text("OverflowBox Widget"),  
    ),  
    body: Center(  
      child: Container(  
        height: 50.0,  
        width: 50.0,  
        color: Colors.red,  
        child: OverflowBox(  
          minHeight: 70.0,  
          minWidth: 70.0,  
          child: Container(  
            height: 50.0,  
            width: 50.0,  
            color: Colors.blue,  
          ),  
        ),  
      ),  
    ),  
  );  
}
```



Hình 2. 13. Ví dụ về cách sử dụng Widget OverflowBox

❖ Widget đa

Đa widget là một loại widget chứa **nhiều hơn một widget con bên trong** và cách bố trí của các widget này là **đuy nhất**. Ví dụ: widget Row bố trí widget theo hướng ngang và widget Column bố trí widget theo hướng dọc. Nếu chúng ta kết hợp widget Row và Column, thì nó có thể xây dựng bất kỳ cấp độ nào của widget phức tạp.

Ta có đoạn code minh họa như sau:

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

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

class MyApp extends StatelessWidget {
```

```
// It is the root widget of your application.

@Override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Multiple Layout Widget',
    debugShowCheckedModeBanner: false,
    theme: ThemeData(
      // This is the theme of your application.
      primarySwatch: Colors.blue,
    ),
    home: MyHomePage(),
  );
}

class MyHomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center(
      child: Container(
        alignment: Alignment.center,
        color: Colors.white,
        child: Row(
          children: <Widget>[
            Expanded(
              child: Text('Peter', textAlign: TextAlign.center),
            )
          ],
        ),
      ),
    );
  }
}
```

```
 ),  
 Expanded(  
   child: Text('John', textAlign: TextAlign.center ),  
  
 ),  
 Expanded(  
   child: FittedBox(  
     fit: BoxFit.contain, // otherwise the logo will be tiny  
     child: const FlutterLogo(),  
   ),  
   ),  
   ],  
 ),  
 ),  
 );  
 }  
 }
```



Hình 2. 14. Kết hợp giữa Widget Row và Column

Danh sách các Widget đa gồm:

- ❖ **Row:** Nó cho phép sắp xếp các widget con của nó theo hướng nằm ngang.
- ❖ **Column:** Nó cho phép sắp xếp các widget con của nó theo hướng dọc.
- ❖ **ListView:** Đây là widget cuộn phổ biến nhất cho phép chúng ta sắp xếp các widget con của nó lần lượt theo hướng cuộn.
- ❖ **GridView:** Nó cho phép chúng ta sắp xếp các widget con của nó dưới dạng một mảng widget 2D, có thể cuộn được. Nó bao gồm một mô hình lặp lại của các ô được sắp xếp theo bố cục ngang và dọc.
- ❖ **Expanded:** Nó cho phép tạo các con của widget Hàng và Cột chiếm diện tích tối đa có thể.
- ❖ **Table:** Nó là một widget cho phép chúng ta sắp xếp các con của nó trong một widget dựa trên bảng.

- ❖ **Flow:** Nó cho phép chúng ta triển khai widget dựa trên luồng.
- ❖ **Stack:** Đây là một widget thiết yếu, chủ yếu được sử dụng để chồng lên một số widget. Nó cho phép ta đặt nhiều lớp lên màn hình. Ví dụ sau đây giúp ta hiểu điều đó.

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

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

class MyApp extends StatelessWidget {
    // It is the root widget of your application.

    @override
    Widget build(BuildContext context) {
        return MaterialApp(
            title: 'Multiple Layout Widget',
            debugShowCheckedModeBanner: false,
            theme: ThemeData(
                // This is the theme of your application.
                primarySwatch: Colors.blue,
            ),
            home: MyHomePage(),
        );
    }
}

class MyHomePage extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return Center(
            child: Container(
                alignment: Alignment.center,
```

```
color: Colors.white,  
child: Stack(  
    children: <Widget>[  
        // Max Size  
        Container(  
            color: Colors.blue,  
        ),  
        Container(  
            color: Colors.pink,  
            height: 400.0,  
            width: 300.0,  
        ),  
        Container(  
            color: Colors.yellow,  
            height: 220.0,  
            width: 200.0,  
        )  
    ],  
),  
(  
);  
}  
}
```



Hình 2. 15. Một ví dụ minh họa của Widget Stack được sử dụng như thế nào

2.1.2.3. Xây dựng thử một bô cục phức tạp

Trong phần này, chúng ta sẽ tìm hiểu cách có thể tạo giao diện người dùng phức tạp bằng cách sử dụng cả widget bô cục đơn và nhiều widget. Khung bô cục cho phép ta tạo bô cục giao diện người dùng phức tạp bằng cách lồng các hàng và cột vào bên trong các hàng và cột. Bây giờ, trong bài báo cáo này sẽ tạo một ví dụ về giao diện người dùng phức tạp bằng cách **tạo danh sách sản phẩm**. Ta có đoạn code mẫu như sau:

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

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

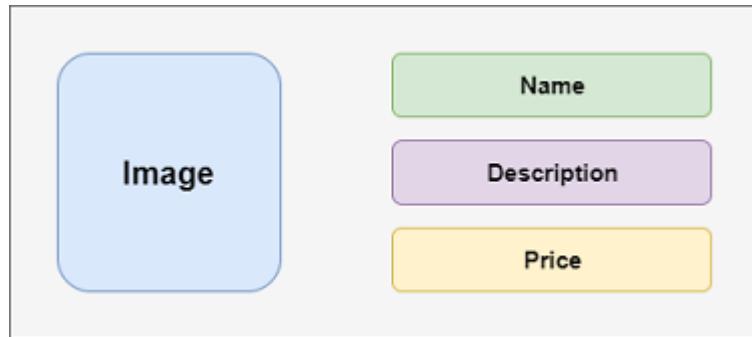
```
class MyApp extends StatelessWidget {  
    // It is the root widget of your application.  
  
    @override  
  
    Widget build(BuildContext context) {  
  
        return MaterialApp(  
            title: 'Flutter Demo Application', theme: ThemeData(  
                primarySwatch: Colors.green,),  
            home: MyHomePage(title: 'Complex layout example'),  
        );  
    }  
}  
  
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("Product List")),  
            body: ListView(  
                padding: const EdgeInsets.fromLTRB(3.0, 12.0, 3.0, 12.0),  
                children: <Widget>[  
                    ProductBox(  
                        name: "iPhone",  
                    ),  
                    ProductBox(  
                        name: "Samsung",  
                    ),  
                    ProductBox(  
                        name: "Huawei",  
                    ),  
                    ProductBox(  
                        name: "Oppo",  
                    ),  
                    ProductBox(  
                        name: "Vivo",  
                    ),  
                ],  
            ),  
        );  
    }  
}
```

```
description: "iPhone is the top branded phone ever",
    price: 55000,
    image: "iphone.png"
),
ProductBox(
    name: "Android",
    description: "Android is a very stylish phone",
    price: 10000,
    image: "android.png"
),
ProductBox(
    name: "Tablet",
    description: "Tablet is a popular device for official meetings",
    price: 25000,
    image: "tablet.png"
),
ProductBox(
    name: "Laptop",
    description: "Laptop is most famous electronic device",
    price: 35000,
    image: "laptop.png"
),
ProductBox(
    name: "Desktop",
    description: "Desktop is most popular for regular use",
```

```
    price: 10000,  
    image: "computer.png"  
,  
],  
)  
);  
}  
}  
  
class ProductBox extends StatelessWidget {  
  
ProductBox({Key key, this.name, this.description, this.price, this.image}) :  
    super(key: key);  
  
final String name;  
final String description;  
final int price;  
final String image;  
  
  
Widget build(BuildContext context) {  
  
    return Container(  
        padding: EdgeInsets.all(2),  
        height: 110,  
        child: Card(  
            child: Row(  
                mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
                children: <Widget>[  
                    Image.asset("assets/" + image),  
                ]  
            )  
        )  
    );  
}
```

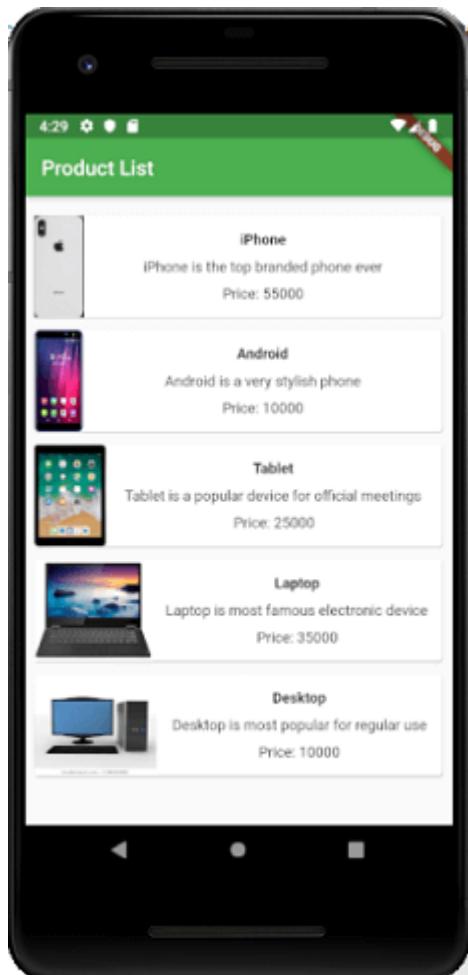
```
Expanded(  
    child: Container(  
        padding: EdgeInsets.all(5),  
        child: Column(  
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
            children: <Widget>[  
                Text(  
                    this.name, style: TextStyle(  
                        fontWeight: FontWeight.bold  
                ),  
                Text(this.description), Text(  
                    "Price: " + this.price.toString()  
                ),  
                ],  
            ),  
        ),  
    );  
}
```

Trong đoạn code trên, chúng ta tạo **ProductBox** widget chứa các chi tiết của sản phẩm, chẳng hạn như hình ảnh, tên, giá và mô tả. Trong widget ProductBox, sử dụng các widget sau: Container, Row, Column, Expanded, Card, Text, Image, v.v. widget này có bố cục sau:



Hình 2. 16. Bố cục của ví dụ ProductBox Widget

Sau khi chạy trên giả lập android, ta có kết quả như sau:



Hình 2. 17. Kết quả chạy thử giả lập trong ví dụ bố cục

2.1.3. Cử chỉ giao diện (Gestures)

Cử chỉ (Gestures) là một tính năng thú vị trong Flutter cho phép chúng ta tương tác với ứng dụng di động (hoặc bất kỳ thiết bị dựa trên cảm ứng). Nói chung, cử chỉ xác định bất kỳ hành động hoặc chuyển động vật lý nào của người dùng nhằm mục đích kiểm soát thiết bị di động. Một số ví dụ về cử chỉ là:

- Khi màn hình di động bị khóa, ta trượt ngón tay trên màn hình để mở khóa.
- Nhấn vào một nút trên màn hình điện thoại di động của ta và nhấn và giữ biểu tượng ứng dụng trên thiết bị dựa trên cảm ứng để kéo biểu tượng đó qua các màn hình.

Flutter chia hệ thống cử chỉ thành hai lớp khác nhau, được đưa ra dưới đây:

1. Con trỏ (Pointers)
2. Cử chỉ (Gestures)

2.1.3.1. Con trỏ

Con trỏ (Pointers) là lớp đầu tiên đại diện cho dữ liệu thô về tương tác của người dùng. Nó có các sự kiện, mô tả **vị trí** và **chuyển động** của các con trỏ như chạm, chuột và kiểu trên màn hình. Flutter không cung cấp bất kỳ cơ chế nào để hủy hoặc dừng các sự kiện con trỏ được gửi đi thêm. Flutter cung cấp một widget **Listener** để lắng nghe các sự kiện con trỏ trực tiếp từ lớp widget. Con trỏ sự kiện được phân loại thành bốn loại chủ yếu:

- **PointerDownEvents:** Nó cho phép con trỏ tiếp xúc với màn hình tại một vị trí cụ thể.
- **PointerMoveEvents:** Nó cho phép con trỏ di chuyển từ vị trí này đến vị trí khác trên màn hình.
- **PointerUpEvents:** Nó cho phép con trỏ dừng tiếp xúc với màn hình.
- **PointerCancelEvents:** Sự kiện này được gửi khi tương tác với con trỏ bị hủy.

- **Ví dụ sử dụng:** Có thể sử dụng để theo dõi vị trí ngón tay của người dùng khi di chuyển trên màn hình.

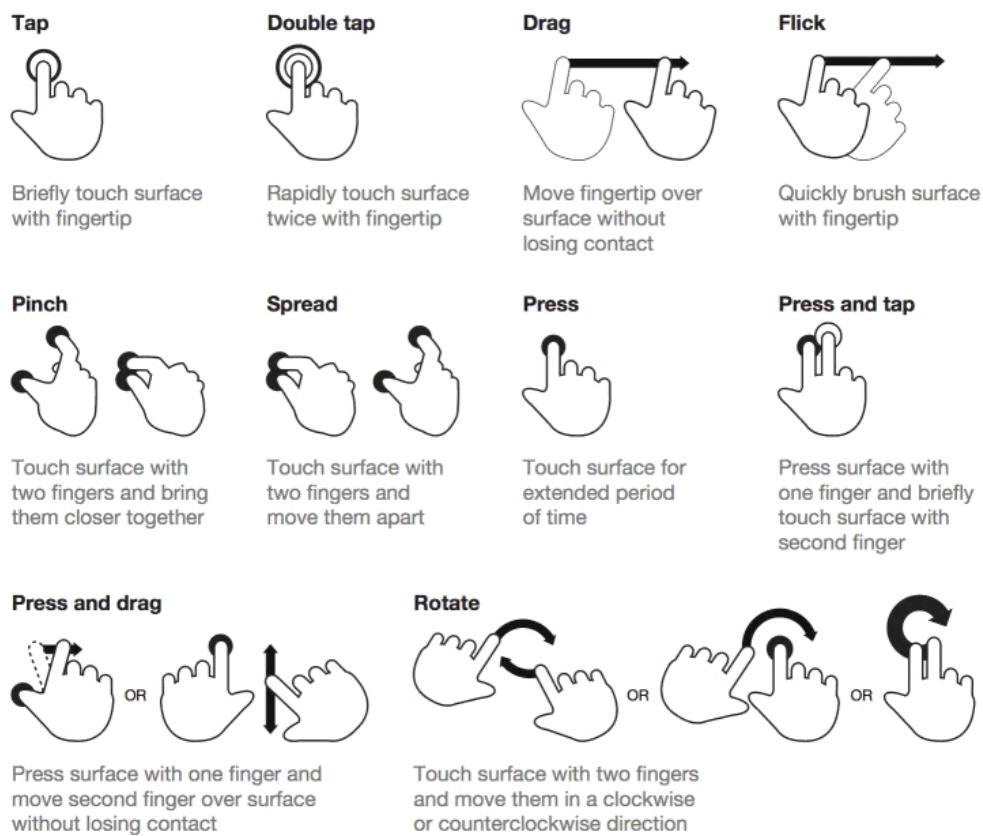
2.1.3.2. Cử chỉ

Đây là lớp thứ hai đại diện cho các hành động ngữ nghĩa như chạm, vuốt, kéo, phóng to, thu nhỏ, v.v. được nhận dạng từ nhiều sự kiện con trỏ riêng lẻ. Nó cũng có thể gửi nhiều sự kiện tương ứng với vòng đời cử chỉ như kéo bắt đầu, kéo cập nhật và kéo kết thúc. Một số cử chỉ được sử dụng phổ biến được liệt kê dưới đây:

1. **Tap (Chạm):** Có nghĩa là chạm vào bề mặt màn hình từ đầu ngón tay trong một thời gian ngắn rồi thả chúng ra. Cử chỉ này chứa các sự kiện sau:
 - a. onTapDown
 - b. onTapUp
 - c. onTap
 - d. onTapCancel
2. **Double Tap (Nhấn kép):** Nó tương tự như cử chỉ Nhấn, nhưng ta cần nhấn hai lần trong thời gian ngắn. Cử chỉ này chứa các sự kiện sau:
 - a. onDoubleTap
3. **Drag (Kéo):** Nó cho phép chúng ta chạm vào bề mặt của màn hình bằng đầu ngón tay và di chuyển nó từ vị trí này sang vị trí khác rồi thả chúng ra. Flutter phân loại kéo thành hai loại:
 - a. **Kéo ngang:** Cử chỉ này cho phép con trỏ di chuyển theo hướng ngang. Nó chứa các sự kiện sau:
 - i. onHorizontalDragStart
 - ii. onHorizontalDragUpdate
 - iii. onHorizontalDragEnd
 - b. **Kéo dọc:** Cử chỉ này cho phép con trỏ di chuyển theo hướng thẳng đứng. Nó chứa các sự kiện sau:
 - i. onVerticalDragStart
 - ii. onVerticalDragStart

iii. onVerticalDragStart

4. **Long Press (Nhấn lâu):** Có nghĩa là chạm vào bề mặt của màn hình tại một vị trí cụ thể trong một thời gian dài. Cử chỉ này chứa các sự kiện sau:\n
 - a. onLongPress
5. **Pan (Di chuyển):** Có nghĩa là chạm vào bề mặt của màn hình bằng đầu ngón tay, có thể di chuyển theo bất kỳ hướng nào mà không cần nhả đầu ngón tay. Cử chỉ này chứa các sự kiện sau:\n
 - a. onPanStart
 - b. onPanUpdate
 - c. onPanEnd
6. **Pinch (Chụm):** Có nghĩa là chụm (di chuyển ngón tay và ngón cái của một người hoặc đưa chúng lại gần nhau trên màn hình cảm ứng) bề mặt của màn hình bằng cách sử dụng hai ngón tay để phóng to hoặc thu nhỏ màn hình.



Hình 2. 18. Hình ảnh minh họa cử chỉ trong Flutter

2.1.3.3. Bắt sự kiện cử chỉ

Flutter cung cấp một tiện ích hỗ trợ tuyệt vời cho tất cả các loại cử chỉ bằng cách sử dụng tiện ích **GestureDetector**. GestureWidget là các widget không trực quan, chủ yếu được sử dụng để phát hiện cử chỉ của người dùng. Ý tưởng cơ bản của bộ phát hiện cử chỉ là một tiện ích **không trạng thái** có chứa các tham số trong hàm tạo của nó cho các sự kiện chạm khác nhau.

Trong một số tình huống, có thể có nhiều bộ phát hiện cử chỉ tại một vị trí cụ thể trên màn hình và sau đó, khung sẽ xác định cử chỉ nào sẽ được gọi. Widget GestureDetector quyết định cử chỉ nào sẽ nhận ra dựa trên lệnh gọi lại nào của nó là không rỗng.

Hãy để chúng ta tìm hiểu cách chúng ta có thể sử dụng các cử chỉ này trong ứng dụng của mình với sự kiện **onTap()** đơn giản và xác định cách GestureDetector xử lý điều này. Ở đây, chúng ta sẽ tạo **một widget hộp**, thiết kế nó theo đặc điểm kỹ thuật mong muốn của chúng ta và sau đó thêm hàm onTap() vào nó. Ta có code như sau:

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

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

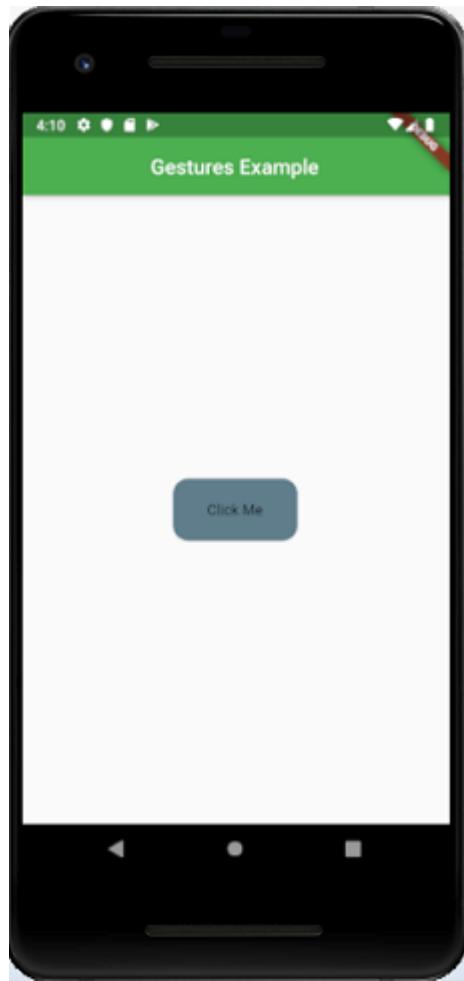
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
```

```
title: 'Flutter Demo Application',  
theme: ThemeData(  
    primarySwatch: Colors.green,  
,  
home: MyHomePage(),  
);  
}  
}  
  
class MyHomePage extends StatefulWidget {  
const MyHomePage({super.key});  
  
@override  
MyHomePageState createState() => MyHomePageState();  
}  
  
class MyHomePageState extends State<MyHomePage> {  
@override  
Widget build(BuildContext context) {  
return Scaffold(  
    appBar: AppBar(  
        title: const Text('Gestures Example'),  
        centerTitle: true,  
,  
    ),  
    body: Center(  
)
```

```
child: GestureDetector(  
    onTap: () {  
        print('Box Clicked');  
    },  
    child: Container(  
        height: 60.0,  
        width: 120.0,  
        padding: const EdgeInsets.all(10.0),  
        decoration: BoxDecoration(  
            color: Colors.blueGrey,  
            borderRadius: BorderRadius.circular(15.0),  
        ),  
        child: const Center(  
            child: Text('Click Me'),  
        ),  
    ),  
),  
(  
);  
}  
}
```

Sau đó, ta khởi động máy ảo và tiến hành build chương trình, ta có kết quả như sau:



Hình 2. 19. Giao diện bắt sự kiện cử chỉ

Khi ta ấn vào Container xám có dòng chữ **Click me** sẽ có dòng console hiển thị lên kết quả rằng “Box Clicked” cho thấy khu vực Container đó đã được ấn vào. Và khi ta ấn vào khu vực trắng thì sẽ không có chuyện gì xảy ra trên console.

2.1.4. Quản lý trạng thái (State)

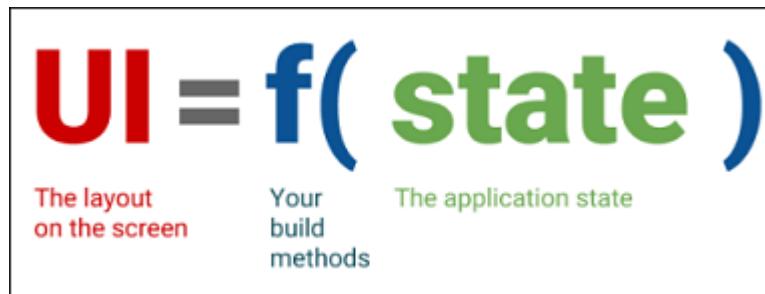
Tiếp đến trong phần này, ta sẽ tìm hiểu về quản lý trạng thái (State) và cách chúng ta sử lý nó trong Flutter. Chúng ta biết rằng trong Flutter, mọi thứ đều là một widget. Có thể phân loại widget này thành hai loại, một là **widget Không trạng thái (Stateless Widget)** và một là **widget có Trạng thái (Stateful Widget)**. Widget không trạng thái không có bất kỳ trạng thái bên trong nào. Nó có nghĩa là một khi nó được xây dựng, chúng ta không thể thay đổi hoặc sửa đổi nó cho đến khi chúng được khởi tạo lại. Mặt khác, Widget Stateful là động và có trạng thái. Nó có nghĩa là chúng

ta có thể sửa đổi nó một cách dễ dàng trong suốt vòng đời của nó mà không cần khởi động lại nó một lần nữa.

2.1.4.1. Khái niệm

Trạng thái là thông tin có thể **đọc** được khi widget được tạo và có thể **thay đổi hoặc sửa đổi** trong suốt thời gian tồn tại của ứng dụng. Nếu ta muốn thay đổi widget của mình, ta cần cập nhật đối tượng trạng thái, có thể được thực hiện bằng cách sử dụng hàm **setState()** có sẵn cho các **Widget Stateful**. Hàm setState() cho phép chúng ta thiết lập các thuộc tính của đối tượng trạng thái kích hoạt vẽ lại giao diện người dùng.

Quản lý trạng thái (State) là một trong những quy trình phổ biến và cần thiết nhất trong vòng đời của một ứng dụng. Theo tài liệu chính thức, Flutter mang tính chất khai báo. Điều đó có nghĩa là Flutter xây dựng giao diện người dùng của mình bằng cách phản ánh trạng thái hiện tại của ứng dụng của ta. Hình sau giải thích rõ hơn về nơi ta có thể xây dựng giao diện người dùng từ trạng thái ứng dụng.



Hình 2. 20. Một giải nghĩa cho Quản lý trạng thái trên Flutter

Chúng ta hãy lấy một ví dụ đơn giản để hiểu khái niệm quản lý trạng thái (State). Giả sử ta đã tạo danh sách khách hàng hoặc sản phẩm trong ứng dụng của mình. Nay ta đã thêm một khách hàng mới vào danh sách đó. Sau đó, cần phải làm mới danh sách để xem mục mới được thêm vào bản ghi. Vì vậy, bất cứ khi nào ta thêm một mục mới, ta cần phải làm mới danh sách. Kiểu lập trình này yêu cầu quản lý trạng thái (State) xử lý tình huống như vậy để cải thiện hiệu suất. Đó là bởi vì mỗi khi ta thực hiện một thay đổi hoặc cập nhật giống nhau, trạng thái sẽ được làm mới.

Trong Flutter , quản lý trạng thái (State) phân thành hai loại khái niệm, được đưa ra dưới đây:

1. Trạng thái tức thời (Ephemeral State)
2. Trạng thái ứng dụng (App State)

2.1.4.2. Trạng thái tức thời (Ephemeral State)

Trạng thái này còn được gọi là Trạng thái giao diện người dùng hoặc trạng thái địa phương. Nó là một loại trạng thái có liên quan đến **widget cụ thể**, hoặc ta có thể nói rằng nó là một trạng thái chúa trong một widget duy nhất. Trong loại trạng thái này, ta không cần sử dụng các kỹ thuật quản lý state(trạng thái):

- Liên quan đến dữ liệu thay đổi theo thời gian nhanh chóng, không liên quan đến trạng thái tổng thể của ứng dụng.
- Ví dụ: vị trí con trỏ chuột, giá trị trong một trường nhập liệu, kết quả tìm kiếm tức thời.
- Không cần lưu trữ hoặc khôi phục khi ứng dụng thay đổi trạng thái.

Ví dụ phổ biến của trạng thái này là Text Field:

```
class MyHomepage extends StatefulWidget {
  @override
  MyHomepageState createState() => MyHomepageState();
}

class MyHomepageState extends State<MyHomepage> {
  String _name = "Peter";
  @override
  Widget build(BuildContext context) {
```

```

return RaisedButton(
    child: Text(_name),
    onPressed: () {
        setState(() {
            _name = _name == "Peter" ? "John" : "Peter";
        });
    },
);
}
}

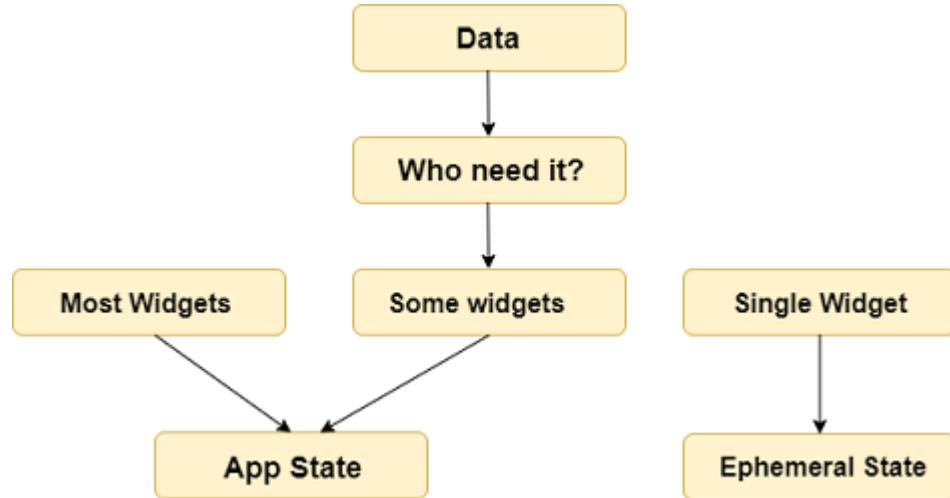
```

Trong ví dụ trên, biến `_name` là một trạng thái tạm thời. Ở đây, chỉ có hàm `setState()` bên trong lớp của `StatefulWidget` mới có thể truy cập vào `_name`. Phương thức xây dựng gọi một hàm `setState()`, hàm này thực hiện sửa đổi các biến trạng thái. Khi phương thức này được thực thi, đối tượng widget sẽ được thay thế bằng đối tượng mới, mang lại giá trị biến được sửa đổi.

2.1.4.3. Trạng thái ứng dụng (App State)

Nó khác với trạng thái tức thời. Đó là một loại trạng thái mà chúng ta muốn **chia sẻ** trên các phần khác nhau của ứng dụng và muốn giữ lại giữa các phiên của người dùng. Do đó, loại trạng thái này có thể được sử dụng trên toàn cầu. Đôi khi nó còn được gọi là trạng thái ứng dụng hoặc trạng thái chia sẻ. Một số ví dụ về trạng thái này là Tùy chọn người dùng, Thông tin đăng nhập, thông báo trong ứng dụng mạng xã hội, giỏ hàng trong ứng dụng thương mại điện tử, trạng thái đã đọc / chưa đọc của các bài báo trong ứng dụng tin tức, v.v.

Sơ đồ sau giải thích sự khác biệt giữa trạng thái tạm thời và trạng thái ứng dụng một cách phù hợp hơn.



Hình 2. 21. Ví dụ minh họa cho trạng thái ứng dụng

2.1.4.4. Quản lý trạng thái (State) bằng thư viện Provider

Ví dụ đơn giản nhất về quản lý trạng thái ứng dụng có thể được học bằng cách sử dụng **thư viện Provider**. Việc quản lý trạng thái (State) với thư viện Provider rất dễ hiểu và ít phải viết code. Provider là thư viện của **bên thứ ba**. Ở đây, chúng ta cần hiểu ba khái niệm chính để sử dụng thư viện này:

1. ChangeNotifier
2. ChangeNotifierProvider
3. Consumer

❖ ChangeNotifier

ChangeNotifier là một lớp đơn giản, cung cấp thông báo thay đổi cho người nghe của nó. Nó dễ hiểu, dễ thực hiện và được tối ưu hóa cho một số lượng nhỏ người nghe. Nó được sử dụng để người nghe quan sát một mô hình để thay đổi. Trong điều này, chúng ta chỉ sử dụng phương thức **notifyListener()** để thông báo cho người nghe.

Ví dụ: chúng ta hãy xác định một mô hình dựa trên ChangeNotifier. Trong mô hình này, **Lớp Counter** được mở rộng (extend) với lớp **ChangeNotifier**, được sử dụng để thông báo cho người nghe của nó khi chúng ta gọi **InformListeners()**. Đây là phương thức duy nhất cần triển khai trong mô hình ChangeNotifier. Trong ví dụ này, chúng ta đã khai báo hai hàm là tăng

và giảm, được sử dụng để tăng và giảm giá trị. Chúng ta có thể gọi phương thức `notifyListeners()` bất kỳ lúc nào mô hình thay đổi theo cách có thể thay đổi giao diện người dùng của ứng dụng.

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

class Counter with ChangeNotifier {
    int _counter;

    Counter(this._counter);

    getCounter() => _counter; // getter

    setCounter(int counter) => _counter = counter; // setter

    void increment() {
        _counter++;
        notifyListeners();
    }

    void decrement() {
        _counter--;
        notifyListeners();
    }
}
```

❖ **ChangeNotifierProvider**

`ChangeNotifierProvider` là widget cung cấp một phiên bản của `ChangeNotifier` cho con của nó. Nó đến từ thư viện `Provider`. Các đoạn code sau đây giúp hiểu khái niệm về `ChangeNotifierProvider`.

Ở đây, chúng ta đã định nghĩa một người xây dựng (`builder`) những người sẽ tạo một đối tượng mới của **Lớp Counter**. `ChangeNotifierProvider`

không xây dựng lại lớp Counter trừ khi có sự kiện yêu cầu cho việc này.

Nó cũng sẽ tự động gọi phương thức **dispose()** trên mô hình Counter khi thể hiện không còn cần thiết nữa.

```
class MyApp extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return MaterialApp(
            theme: ThemeData(
                primarySwatch: Colors.indigo,
            ),
            home: ChangeNotifierProvider<CounterModel>(
                builder: (_) => CounterModel(),
                child: CounterView(),
            ),
        );
    }
}
```

Nếu có nhu cầu cung cấp nhiều hơn một lớp, ta có thể sử dụng **MultiProvider**.

MultiProvider là một widget trong Flutter cho phép ta cung cấp nhiều Provider khác nhau cho các widget con của nó. Nó hoạt động như một danh sách tất cả các Provider đang được sử dụng trong phạm vi của nó. Chúng ta có thể hình dung thông qua đoạn code như sau:

```
void main() {
    runApp(

```

```

MultiProvider(
  providers: [
    ChangeNotifierProvider(builder: (context) => Counter()),
    Provider(builder: (context) => SomeOtherClass()),
  ],
  child: MyApp(),
),
);
}

```

❖ Consumer

Consumer là một widget trong Flutter được sử dụng để truy cập dữ liệu từ một Provider. Nó cho phép ta hiển thị dữ liệu được cung cấp bởi Provider trong giao diện widget. Chúng ta có thể hình dung thông qua đoạn code như sau:

```

return Consumer<Counter>(
  builder: (context, count, child) {
    return Text("Total price: ${count.total}");
  },
);

```

Tóm lại: Từ các khối code trên, ta có một khối code ví dụ hoàn chỉnh cho việc sử dụng Provider để quản lý trạng thái (State) như sau:

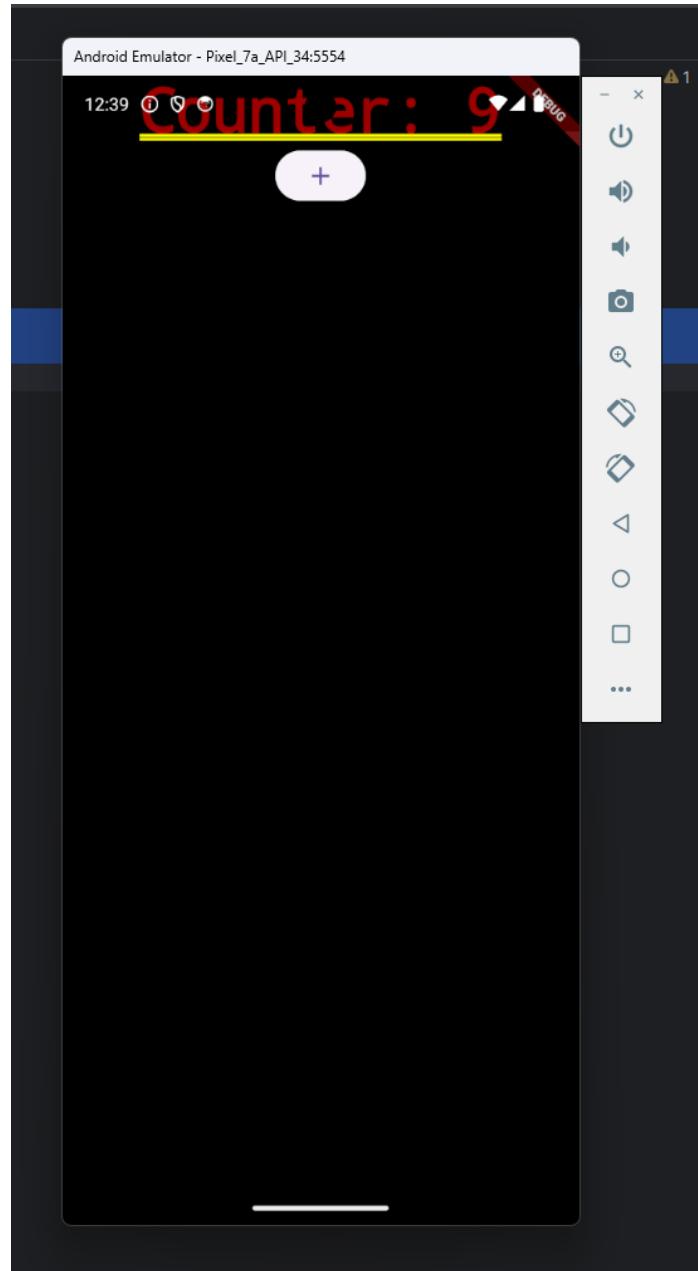
```

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

```

```
void main() => runApp(const MyApp());  
  
class MyApp extends StatelessWidget {  
  const MyApp({super.key});  
  
  // This widget is the root of your application.  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      title: 'Flutter Demo Application',  
      home: MyWidget(),  
    );  
  }  
}  
  
class CounterProvider extends ChangeNotifier {  
  int _counter = 0;  
  
  int get counter => _counter;  
  
  void increment() {  
    _counter++;  
  }  
}
```

```
notifyListeners();  
}  
}  
  
class MyWidget extends StatelessWidget {  
  
  @override  
  
  Widget build(BuildContext context) {  
  
    return ChangeNotifierProvider(  
  
      create: (context) => CounterProvider(),  
  
      child: Consumer<CounterProvider>(builder: (context, provider, child) {  
  
        return Column(  
  
          children: [  
  
            Text('Counter: ${provider.counter}'),  
  
            ElevatedButton(  
  
              onPressed: () => provider.increment(),  
  
              child: const Icon(Icons.add),  
  
            ),  
          ],  
        );  
      }),  
    );  
  }  
}
```



Hình 2. 22. Demo minh họa cho quản lý trạng thái (State) bằng thư viện Provider

2.1.4.5. So sánh Quản lý trạng thái bằng cách thông thường và Quản lý trạng thái bằng thư viện Provider

- ❖ Quản lý state bằng cách thông thường:
 - Sử dụng **setState** trong widget để cập nhật state của widget đó.
 - Phù hợp cho việc quản lý state đơn giản trong một widget.
 - Có thể dẫn đến code lộn xộn và khó bảo trì khi ứng dụng phức tạp hơn.

❖ Quản lý state bằng Provider:

- Sử dụng Provider để chia sẻ state giữa các widget.
- Dễ sử dụng và tái sử dụng cho các state phức tạp.
- Giúp tách biệt logic quản lý state khỏi giao diện widget.
- Cung cấp giải pháp quản lý state hiệu quả và linh hoạt cho các ứng dụng Flutter.

Ta có một bảng so sánh như sau:

Tính năng	Cách thông thường	Provider
Phù hợp cho	State đơn giản trong một widget	State phức tạp chia sẻ giữa nhiều widget
Dễ sử dụng	Khó khăn khi state phức tạp	Dễ dàng
Tái sử dụng	Khó khăn	Dễ dàng
Bảo trì	Khó khăn khi ứng dụng phức tạp	Dễ dàng
Tách biệt	Logic quản lý state không tách biệt khỏi giao diện	Logic quản lý state tách biệt khỏi giao diện
Giải pháp	Giải pháp đơn giản cho state đơn giản	Giải pháp hiệu quả và linh hoạt cho state phức tạp

Kết luận:

1. Nên sử dụng quản lý state bằng cách thông thường cho state đơn giản trong một widget.
2. Nên sử dụng Provider cho state phức tạp chia sẻ giữa nhiều widget.
3. Provider cung cấp giải pháp quản lý state hiệu quả và linh hoạt hơn cho các ứng dụng Flutter.

2.1.5. Điều hướng màn hình

Điều hướng và định tuyến (Navigation and Routing) là một số khái niệm cốt lõi của tất cả các ứng dụng di động, cho phép người dùng di chuyển giữa các trang khác nhau. Chúng ta biết rằng mọi ứng dụng di động đều chứa một số màn hình để hiển thị các loại thông tin khác nhau. Ví dụ: một ứng dụng có thể có màn hình chứa nhiều sản phẩm khác nhau. Khi người dùng chạm vào sản phẩm đó, ngay lập tức nó sẽ hiển thị thông tin chi tiết về sản phẩm đó.

Trong Flutter, các màn hình và trang được gọi là **các tuyến** và các tuyến này chỉ là một widget. Trong Android, một tuyến tương tự như **Activity**, trong khi trong iOS, nó tương đương với **ViewController**.

2.1.5.1. Điều hướng bằng Routes

Trong bất kỳ ứng dụng di động nào, điều hướng đến các trang khác nhau xác định quy trình làm việc của ứng dụng và cách xử lý điều hướng được gọi là định tuyến. Flutter cung cấp một lớp định tuyến cơ bản **MaterialPageRoute** và hai phương thức **Navigator.push()** và **Navigator.pop()** cho biết cách điều hướng giữa hai tuyến đường. Các bước sau là bắt buộc để bắt đầu điều hướng trong ứng dụng:

1. Đầu tiên, ta cần tạo hai tuyến đường. Ví dụ: Màn hình A, Màn hình B.
2. Sau đó, điều hướng đến một tuyến đường từ một tuyến đường khác bằng cách sử dụng phương thức **Navigator.push()**. Ví dụ: Di chuyển từ màn hình A sang màn hình B.
3. Cuối cùng, điều hướng đến tuyến đường đầu tiên bằng cách sử dụng phương thức **Navigator.pop()**. Ví dụ: Quay lại từ màn hình B về màn hình A

Ta có một ví dụ như sau:

```
class FirstRoute extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
```

```
return Scaffold(  
    appBar: AppBar(  
        title: Text('First Route'),  
    ),  
    body: Center(  
        child: RaisedButton(  
            child: Text('Open route'),  
            onPressed: () {  
                // Navigate to second route when tapped.  
                onPressed: () {  
                    Navigator.push(  
                        context,  
                        MaterialPageRoute(builder: (context) => SecondRoute()),  
                    );  
                }  
            },  
        ),  
    ),  
);  
}  
  
}  
  
class SecondRoute extends StatelessWidget {  
    @override  
    Widget build(BuildContext context) {
```

```

return Scaffold(
    appBar: AppBar(
        title: Text("Second Route"),
    ),
    body: Center(
        child: RaisedButton(
            onPressed: () {
                // Navigate back to first route when tapped.
                onPressed: () {
                    Navigator.pop(context);
                }
            },
            child: Text('Go back!'),
        ),
    ),
);
}
}

```

Giải thích khái niệm code trên: 2 class trên tương ứng với 2 màn hình được hiển thị lên máy ảo. Trong Màn hình thứ 1 có nút bấm để di chuyển qua màn hình thứ 2 bằng phương thức **Navigator.push()** và màn hình thứ 2 có nút bấm để quay ngược lại màn hình thứ 1 là **Navigator.pop()**.

Phương thức **Navigator.push()** được sử dụng để điều hướng / chuyển sang một tuyến đường / trang / màn hình mới. Ở đây, phương thức **push()** thêm một trang / tuyến đường trên ngăn xếp và sau đó quản lý nó bằng cách sử dụng **Navigator**. Một

lần nữa, chúng ta sử dụng lớp **MaterialPageRoute** cho phép chuyển đổi giữa các tuyến bằng cách sử dụng hoạt ảnh dành riêng cho nền tảng.

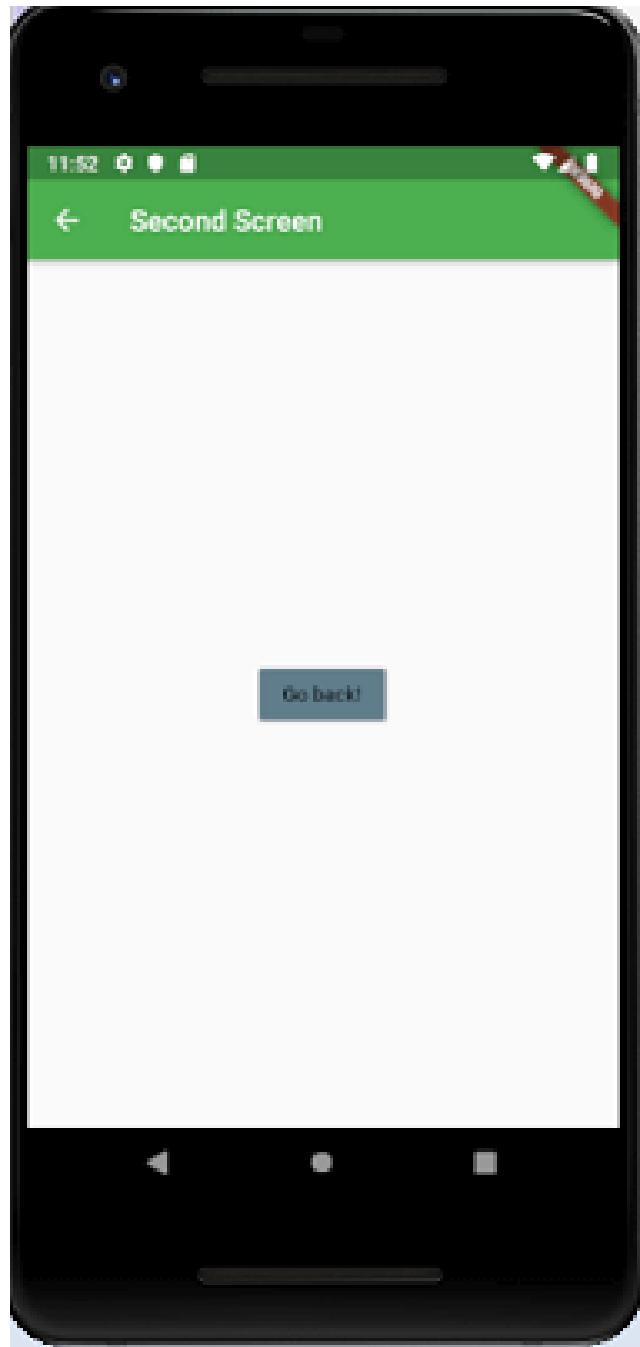
Và khi chúng ta cần quay ngược trở lại, chúng ta sử dụng phương thức **Navigator.pop()** để đóng tuyến thứ hai và quay lại tuyến đầu tiên. Phương thức **pop()** cho phép chúng ta loại bỏ tuyến đường hiện tại khỏi ngăn xếp, được quản lý bởi Bộ điều hướng.

Chương trình Demo như sau:



Hình 2. 23. Màn hình thứ 1 có nút bấm để di chuyển tới màn hình thứ 2

Sau khi chúng ta ấn vào nút “Click Here”, hệ thống sẽ bắt sự kiện và di chuyển chúng ta đến màn hình thứ 2 có nút “Go Back”:



Hình 2. 24. Màn hình thứ 2 có nút bấm để quay ngược về màn hình thứ 1

2.1.5.2. Điều hướng bằng RouteName

Cũng giống như điều hướng bằng Route, RouteName cho phép chúng ta cấu hình các tuyến đường mặc định trước bằng các “KeyName” mà chúng ta đặt ra. Nó sử dụng lớp **Navigator** cùng với việc đặt tên cho các màn hình (route) để thực hiện

việc điều hướng. Về cơ bản nó giống với Route nhưng lại có cách thức hoạt động khác hơn nhiều.

Trong hàm **main()** của ứng dụng Flutter, ta cần định nghĩa các **named route**. Mỗi named route bao gồm tên **route** và **widget** tương ứng sẽ được hiển thị khi điều hướng đến route đó. Ta có đoạn code ví dụ như sau:

```
MaterialApp(
  initialRoute: '/',
  routes: {
    '/': (context) => FirstScreen(),
    '/second': (context) => SecondScreen(),
  },
);
```

Điều hướng bằng route name: Sử dụng phương thức **Navigator.pushNamed()** để điều hướng đến route đã được định nghĩa. Ta chỉ cần truyền tên route vào phương thức này:

```
Navigator.pushNamed(context, '/second'); // Điều hướng đến route '/second'
```

Truyền tham số (tùy chọn): Ta có thể truyền tham số cho route bằng cách sử dụng thuộc tính **arguments** của phương thức **Navigator.pushNamed()**. Tham số sẽ được truyền vào widget tương ứng của route dưới dạng **Map<String, dynamic>**:

```
Navigator.pushNamed(
  context,
  '/second',
  arguments: {'data': 'Hello from FirstScreen'},
);
```

2.1.5.3. So sánh giữa Route và RouteName

1. Route:

a. Ưu điểm:

- i. Dễ dàng sử dụng và hiệu.
- ii. Linh hoạt, cho phép ta truyền dữ liệu và tùy chỉnh route.

b. Nhược điểm:

- i. Có thể trở nên lộn xộn khi ứng dụng phức tạp với nhiều route.
- ii. Khó quản lý route name.

2. RouteName:

a. Ưu điểm:

- i. Giữ cho code gọn gàng và dễ đọc.
- ii. Dễ dàng quản lý route name.
- iii. Thích hợp cho ứng dụng lớn với nhiều route.

b. Nhược điểm:

- i. Cần thêm bước định nghĩa named routes.
- ii. Ít linh hoạt hơn so với điều hướng bằng route.

Bảng so sánh tổng quan:

Tính năng	Routes	Route Name
Cách thức hoạt động	Sử dụng widget	Sử dụng tên route
Ưu điểm	Dễ sử dụng, linh hoạt	Gọn gàng, dễ quản lý
Nhược điểm	Lộn xộn khi phức tạp	Cần định nghĩa named routes
Thích hợp cho	Ứng dụng đơn giản	Ứng dụng lớn

2.2. Một số Widgets thông dụng cần phải biết

2.2.1. Widget Scaffold

Widget Scaffold là dạng widget chịu trách nhiệm chính trong việc tạo cơ sở cho màn hình ứng dụng mà trên đó các widget con giữ và hiển thị trên màn hình. Nó

cung cấp nhiều widget hoặc API để hiển thị **Drawer**, **SnackBar**, **BottomNavigationBar**, **AppBar**, **FloatingActionButton**...

Hàm tạo và thuộc tính của Widget Scaffold:

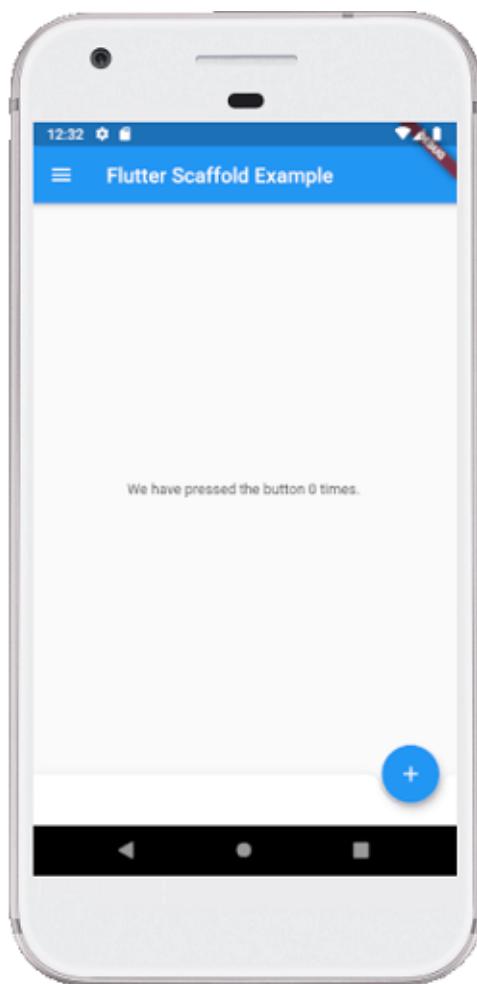
```
const Scaffold({  
  Key key,  
  this.appBar,  
  this.body,  
  this.floatingActionButton,  
  this.floatingActionButtonLocation,  
  this.persistentFooterButtons,  
  this.drawer,  
  this.endDrawer,  
  this.bottomNavigationBar,  
  this.bottomSheet,  
  this.floatingActionButtonAnimator,  
  this.backgroundColor,  
  this.resizeToAvoidBottomPadding = true,  
  this.primary = true,  
})
```

Trong đó, mô tả của các thuộc tính trong đó như sau:

- **appBar**: Nó là một **thanh ngang** chủ yếu được hiển thị ở **trên cùng** của widget Scaffold.
- **body**: Là thuộc tính chính và bắt buộc khác của widget này, nó sẽ **hiển thị nội dung chính trong Scaffold**.
- **drawer**: Nó là một **bảng điều khiển trượt** được hiển thị ở bên cạnh của body. Thông thường, nó bị ẩn trên thiết bị di động, nhưng người dùng có thể vuốt nó từ trái sang phải hoặc từ phải sang trái để truy cập menu **drawer**.
- **floatActionButton**: Là nút hiển thị ở góc dưới cùng bên phải và nổi phía trên phần thân. Nó là một nút biểu tượng hình tròn nổi trên nội dung của

màn hình tại một vị trí cố định để thúc đẩy một hành động chính trong ứng dụng.

- **backgroundColor:** Thuộc tính này được sử dụng để đặt màu nền của toàn bộ widget Scaffold.
- **primary:** Nó được sử dụng để cho biết liệu Scaffold có được hiển thị ở trên cùng của màn hình hay không. Giá trị mặc định của nó là **true**, nghĩa là chiều cao của AppBar được mở rộng bằng chiều cao của thanh trạng thái của màn hình.
- **secureFooterButton:** Đó là danh sách các nút được hiển thị ở cuối widget Scaffold. Các mục thuộc tính này luôn hiển thị, thậm chí chúng ta đã cuộn phần thân của Scaffold. Nó luôn được bao bọc trong một **widget ButtonBar**. Chúng được hiển thị bên dưới phần thân nhưng ở phía trên bottomNavigationBar.
- **bottomNavigationBar:** Thuộc tính này giống như một **menu hiển thị thanh điều hướng** ở cuối Scaffold. Nó có thể được nhìn thấy trong hầu hết các ứng dụng di động.
- **endDrawer:** Nó tương tự như thuộc tính drawer, nhưng chúng được hiển thị ở bên phải màn hình theo mặc định. Nó có thể được vuốt từ phải sang trái hoặc từ trái sang phải.
- **resizeToAvoidBottomInset:** Nếu **đúng**, phần thân và các widget floating của Scaffold nên tự điều chỉnh kích thước của chúng để tránh bàn phím ảo.
- **floatActionButtonLocation:** Theo mặc định, nó được đặt ở góc dưới cùng bên phải của màn hình. Nó được sử dụng để xác định vị trí của floatActionButton.



Hình 2. 25. Một ví dụ của Widget Scaffold

2.2.2. Widget Container

Container (Vùng chứa) trong Flutter là một widget mẹ có thể chứa nhiều widget con và quản lý chúng một cách hiệu quả thông qua chiều rộng, chiều cao, khoảng đệm, màu nền, v.v. Nó là một widget kết hợp vẽ, định vị và định cỡ thông thường của các widget con.

Hàm tạo và thuộc tính của Widget Container:

```
Container({
  Key key,
  AlignmentGeometry alignment,
  EdgeInsetsGeometry padding,
  Color color,
```

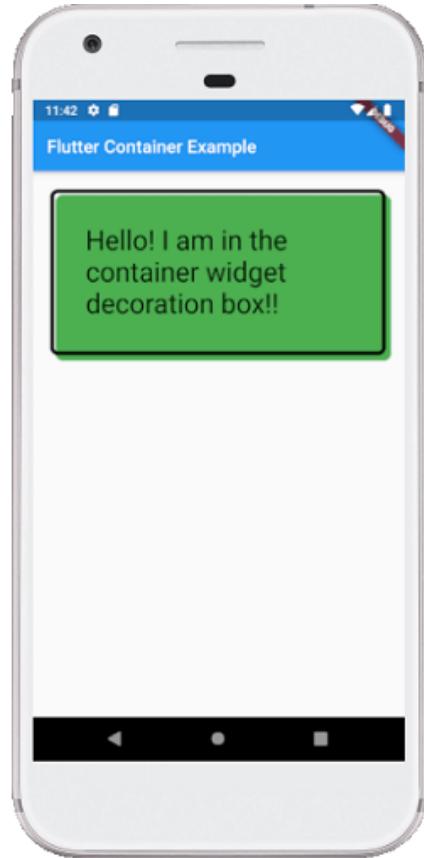
```

    double width,
    double height,
    Decoration decoration,
    Decoration foregroundDecoration,
    BoxConstraints constraints,
    Widget child,
    Clip clipBehavior: Clip.none
);

```

Trong đó, mô tả của các thuộc tính như sau:

- **child:** Thuộc tính này được sử dụng để lưu trữ widget con của Container.
- **color:** Thuộc tính này được sử dụng để đặt **màu nền** của Text.
- **height and width:** Thuộc tính này được sử dụng để thiết lập chiều cao và chiều rộng của container theo nhu cầu của chúng ta.
- **margin:** Thuộc tính này được sử dụng để bao quanh **không gian** empty xung quanh Container.
- **padding:** Thuộc tính này được sử dụng để **đặt khoảng cách** giữa đường viền của Container (cả bốn hướng) và widget con của nó.
- **alignment:** Thuộc tính này được sử dụng để **thiết lập vị trí** của con trong Container.
- **Decoration:** Thuộc tính này cho phép nhà phát triển **thêm trang trí** trên **widget**. Nó trang trí hoặc vẽ các phụ tùng phía sau widget con.
- **transform:** Thuộc tính biến đổi cho phép các nhà phát triển **xoay** Container.
- **constraints:** Thuộc tính này được sử dụng khi chúng ta muốn **thêm các ràng buộc bổ sung** cho widget con



Hình 2. 26. Một ví dụ của Widget Container

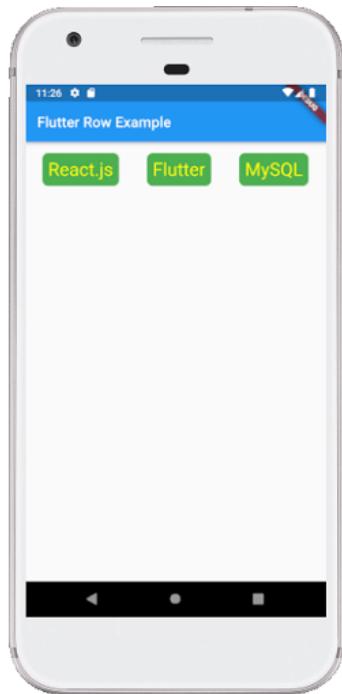
2.2.3. Widget Row và Column

Row và Column là hai widget thiết yếu trong Flutter cho phép các nhà phát triển căn chỉnh widget con theo chiều ngang và chiều dọc theo nhu cầu của chúng ta.

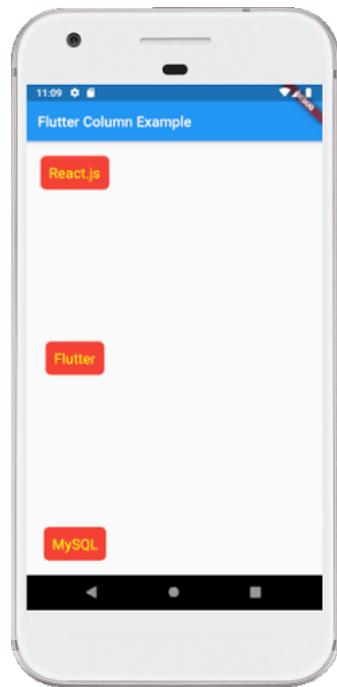
Hàm tạo và thuộc tính của 2 Widget này gần như tương tự nhau:

- **start**: Nó sẽ đặt các con từ điểm bắt đầu của trực chính.
- **end**: Nó sẽ đặt các con ở cuối trực chính.
- **center**: Nó sẽ đặt các con ở giữa trực chính.
- **spaceBetween**: Nó sẽ đặt không gian trống giữa các con một cách đồng đều.
- **spaceAround**: Nó sẽ đặt không gian trống giữa các con một cách đồng đều và một nửa không gian đó trước và sau widget con đầu tiên và cuối cùng.

- **spaceEvenly:** Nó sẽ đặt không gian trống giữa các con một cách đồng đều và trước và sau widget con đầu tiên và cuối cùng.



Hình 2. 28. Một ví dụ của Widget Row



Hình 2. 27. Một ví dụ của Widget Column

2.2.4. Widget Text

Text (Văn bản) là một widget con trong Flutter cho phép chúng ta hiển thị một chuỗi Text với một dòng duy nhất trong ứng dụng của chúng ta.

Hàm tạo và thuộc tính của Widget Text:

```
const Text(String data,{  
  Key key,  
  TextStyle style,  
  StrutStyle strutStyle,  
  TextAlign textAlign,  
  TextDirection textDirection,  
  TextOverflow overflow,  
  bool softWrap,  
  double textScaleFactor,
```

```

int maxLines,
String semanticsLabel,
TextWidthBasis textWidthBasis,
TextHeightBehavior textHeightBehavior
})

```

Trong đó, mô tả của các thuộc tính như sau:

- **TextAlign:** Nó được sử dụng để chỉ định cách Text của chúng ta được căn chỉnh theo chiều ngang. Nó cũng kiểm soát vị trí Text.
- **TextDirection:** Nó được sử dụng để xác định cách các giá trị textAlign kiểm soát bối cảnh của Text của chúng ta. Thông thường, chúng ta viết Text từ trái sang phải, nhưng chúng ta có thể thay đổi nó bằng cách sử dụng tham số này.
- **Overflow:** Nó được sử dụng để xác định khi nào Text sẽ không vừa với không gian có sẵn. Nó có nghĩa là chúng ta đã chỉ định nhiều Text hơn không gian có sẵn.
- **TextScaleFactor:** Nó được sử dụng để xác định tỷ lệ của Text được hiển thị bởi widget Text. Giả sử chúng ta đã chỉ định hệ số tỷ lệ Text là 1,5, thì Text của chúng ta sẽ lớn hơn 50 phần trăm so với kích thước phông chữ được chỉ định.
- **SoftWrap:** Nó được sử dụng để xác định có hay không hiển thị tất cả nội dung widget Text khi không còn đủ dung lượng. Nếu nó là sự thật, nó sẽ hiển thị tất cả nội dung. Nếu không, nó sẽ không hiển thị tất cả nội dung.
- **MaxLines:** Nó được sử dụng để xác định số dòng tối đa được hiển thị trong widget Text.
- **TextWidthBasis:** Nó được sử dụng để kiểm soát cách xác định chiều rộng Text.
- **TextHeightBehavior:** Nó được sử dụng để kiểm soát cách đoạn văn xuất hiện giữa dòng đầu tiên và phần cuối của dòng cuối cùng.

- **Style:** Đây là thuộc tính phổ biến nhất của widget con này cho phép các nhà phát triển tạo kiểu dáng cho Text của họ. Nó có thể tạo kiểu bằng cách chỉ định màu nền và nền trước, cỡ chữ, độ đậm của phông chữ, khoảng cách giữa các chữ và từ, ngôn ngữ, bóng, v.v.



Hình 2. 29. Một ví dụ của Widget Text

2.2.5. Widget TextField

TextField là một widget cơ bản trong Flutter được sử dụng để nhập văn bản từ người dùng. Nó có thể được sử dụng để tạo các trường nhập cho nhiều loại thông tin khác nhau, như tên, email, số điện thoại, v.v.

Hàm tạo và thuộc tính của Widget TextField:

```
TextField ( 
  decoration: InputDecoration(
```

```

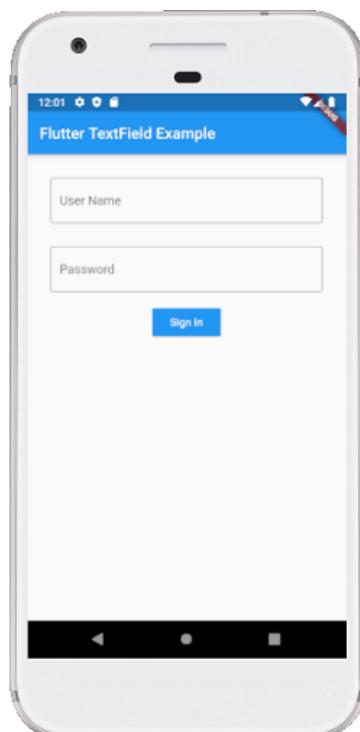
border: InputBorder.none,
labelText: 'Enter Name',
hintText: 'Enter Your Name'
),
);

```

Trong đó, mô tả các thuộc tính như sau:

- **decoration**: Nó được sử dụng để hiển thị decoration xung quanh TextField.
- **border**: Nó được sử dụng để tạo một đường viền hình chữ nhật tròn mặc định xung quanh TextField.
- **labelText**: Nó được sử dụng để hiển thị văn bản nhãn trên vùng chọn TextField.
- **hintText**: Nó được sử dụng để hiển thị văn bản gợi ý bên trong TextField.
- **icon**: Nó được sử dụng để thêm các biểu tượng trực tiếp vào TextField.

Ngoài ra, còn có dạng một phiên bản widget textField khác là **TextField** bao gồm các tính năng tương tự nhưng có nhiều tính năng kiểm soát đầu vào và xác thực đầu vào hơn so với TextField



Hình 2. 30. Một ví dụ của Widget Text Field

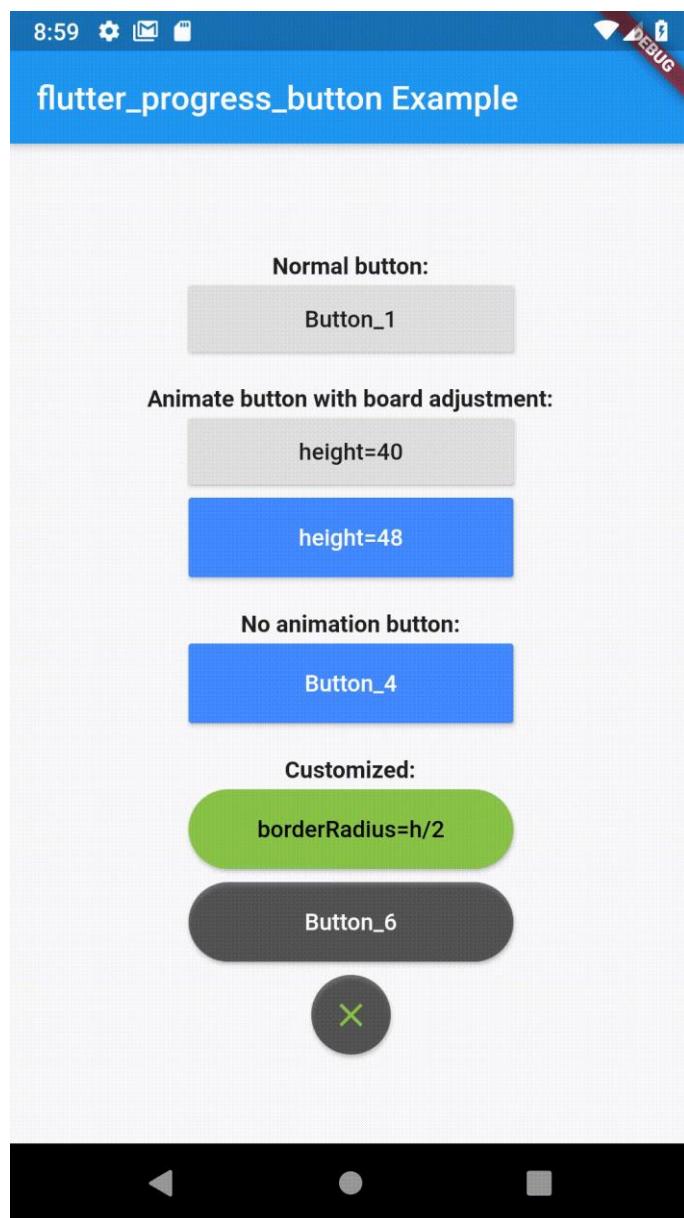
2.2.6. Widget Button

Nút (Button) là phần tử điều khiển đồ họa cung cấp cho người dùng kích hoạt một sự kiện như thực hiện hành động, lựa chọn, tìm kiếm mọi thứ, v.v.

Có một số loại nút sau đây:

- Nút phẳng (Flat Button): Nó là một **nút nhẵn vẫn bẳng** không có nhiều trang trí và hiển thị **mà không có bất kỳ độ cao nào**. Nút phẳng có hai thuộc tính bắt buộc là: **child** và **onPressed()**.
- Nút nâng (Raised Button): Nó là một nút, dựa trên vật liệu vật liệu và có **thân hình chữ nhật**. Nó tương tự như một nút phẳng, nhưng nó có **độ cao(elevation)** sẽ tăng lên khi nút được nhấn.
- Nút nổi (Floating Button): Nút FAB là một **nút biểu tượng hình tròn** kích hoạt hành động chính trong ứng dụng của chúng ta. Nó là nút được sử dụng nhiều nhất trong các ứng dụng hiện nay.
 - **FloatingActionButton**: Nó tạo một nút nổi hình tròn đơn giản với một widget con bên trong nó. Nó phải có một tham số **child** để hiển thị một widget.
 - **FloatingActionButton.extended**: Nó tạo ra một nút nổi rộng cùng với một biểu tượng và nhãn bên trong nó. Thay vì một child, nó sử dụng các nhãn và các thông số biểu tượng.
- Nút thả xuống (Drop Down Button): Một nút thả xuống được sử dụng để tạo một lớp phủ đẹp mắt trên màn hình cho phép người dùng chọn bất kỳ mục nào từ nhiều tùy chọn.
- Nút biểu tượng (Icon Button): IconButton là một **hình ảnh được in** trên widget Material. Nó là một widget hữu ích mang lại cho giao diện người dùng Flutter một cảm giác thiết kế material design.
- Nút Inkwell: Nút InkWell là một khái niệm thiết kế material design, được sử dụng để **phản hồi cảm ứng**.

- Nút PopupMenu: Nó là một nút **hiển thị menu** khi nó được nhấn và sau đó gọi phương thức **onSelected**, menu bị loại bỏ. Đó là vì mục từ nhiều tùy chọn được chọn.
- Nút phác thảo (Outline Button): Nó tương tự như nút phẳng, nhưng nó có một đường viền mỏng hình chữ nhật tròn màu xám.



Hình 2. 31. Một số ví dụ về các loại Widget Button

2.2.7. Widget Stack

Stack là một widget cơ bản trong Flutter được sử dụng để xếp chồng các widget lên nhau. Nó cho phép bạn tạo các bộ cục giao diện người dùng phức tạp bằng cách đặt các widget ở các vị trí khác nhau và điều chỉnh thứ tự hiển thị của chúng.

Đặc điểm chính:

- Sắp xếp các widget con theo thứ tự chồng dồn.
- Widget ở vị trí cuối cùng sẽ được hiển thị ở trên cùng.
- Có thể sử dụng thuộc tính Positioned để điều chỉnh vị trí chính xác của các widget con.
- Thích hợp để tạo các hiệu ứng như:
 - Hiệu ứng chồng lớp (ví dụ: thanh điều hướng ở dưới cùng màn hình)
 - Hiển thị popup hoặc tooltip
 - Tạo các hiệu ứng chuyển động

❖ **Lợi ích:**

- Đơn giản và dễ sử dụng.
- Linh hoạt và mạnh mẽ.
- Có thể tạo nhiều loại bộ cục giao diện người dùng khác nhau.

❖ **Hạn chế:**

- Có thể khiến giao diện người dùng trở nên phức tạp và khó quản lý nếu sử dụng quá nhiều widget con.
- Có thể ảnh hưởng đến hiệu suất nếu sử dụng nhiều widget con động.

❖ **Thay thế:**

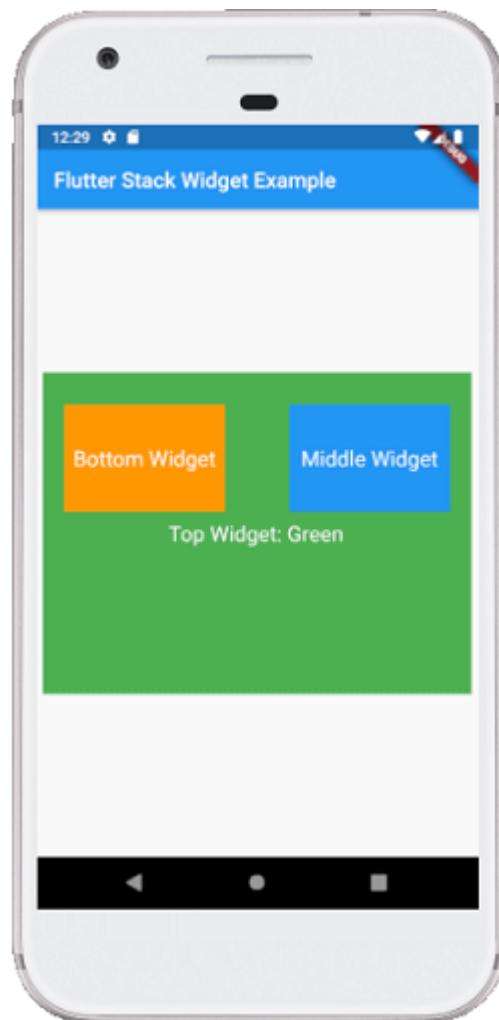
- Row: Widget để xếp các widget theo hàng ngang.
- Column: Widget để xếp các widget theo hàng dọc.
- Flexible: Widget để điều chỉnh kích thước của widget con trong một Row hoặc Column.

Kết luận: Widget Stack là một công cụ hữu ích để tạo các bộ cục giao diện người dùng phức tạp và linh hoạt trong Flutter. Với khả năng xếp chồng các widget

và điều chỉnh vị trí của chúng, Stack có thể được sử dụng để tạo nhiều hiệu ứng giao diện người dùng khác nhau



Hình 2. 32. Một ví dụ về Widget Stack



Hình 2. 33. Một ví dụ khác về Widget Stack khi được ứng dụng vào xây dựng giao diện

2.2.8. Widget Form

Flutter cung cấp widget Form để tạo biểu mẫu, widget form hoạt động như một vùng chứa, cho phép chúng ta nhóm và xác thực nhiều trường biểu mẫu.

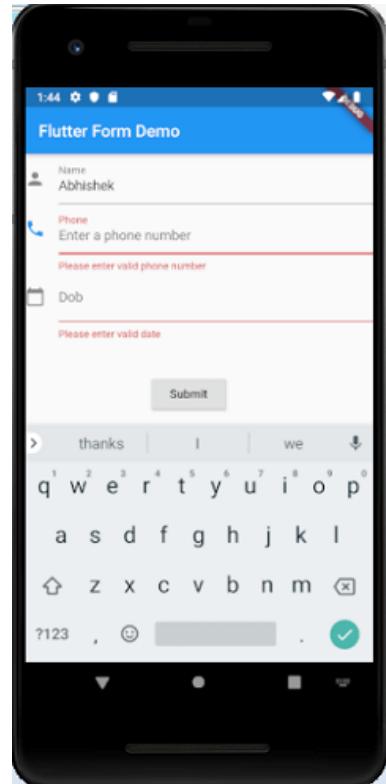
Khi ta tạo một Form, ta cần cung cấp GlobalKey. Khóa này xác định duy nhất biểu mẫu và cho phép ta thực hiện bất kỳ xác thực nào trong các trường biểu mẫu. Xác thực (validation) là một phương pháp, cho phép chúng ta sửa chữa hoặc xác nhận một tiêu chuẩn nhất định. Nó đảm bảo xác thực dữ liệu đã nhập.

- **Đặc điểm chính:**

- Giữ các trường nhập và logic xác thực ở một nơi.
- Cung cấp nhiều tính năng xác thực, bao gồm:
- Kiểm tra kiểu dữ liệu (số, email, v.v.)
- Yêu cầu bắt buộc
- Hiển thị lỗi khi nhập sai
- Quản lý trạng thái của dữ liệu đầu vào.
- Cho phép bạn dễ dàng truy cập và xử lý dữ liệu đầu vào.
- Có thể kết hợp với các widget khác như TextFormField, DropdownButton, Checkbox, v.v.



Hình 2. 34. Một ví dụ về Widget Form được sử dụng để xác thực thông tin nhập vào của các trường



Hình 2. 35. Khi ta nhập sai trong Form thì trường thông báo lỗi

2.2.9. Widget Alert Dialogs

Hộp thoại cảnh báo(Alert Dialogs) là một tính năng hữu ích thông báo cho người dùng thông tin quan trọng để đưa ra quyết định hoặc cung cấp khả năng chọn một hành động cụ thể hoặc danh sách các hành động. Đó là một hộp bật lên xuất hiện ở đầu nội dung ứng dụng và giữa màn hình. Người dùng có thể loại bỏ nó theo cách thủ công trước khi tiếp tục tương tác với ứng dụng.

Cảnh báo có thể được coi là một phương thức nỗi nên được sử dụng để phản hồi nhanh như xác minh mật khẩu, thông báo ứng dụng nhỏ, v.v. Các cảnh báo rất linh hoạt và có thể được tùy chỉnh rất dễ dàng.

Trong Flutter, AlertDialog là một widget, thông báo cho người dùng về các tình huống cần xác nhận. Hộp thoại cảnh báo Flutter chứa tiêu đề tùy chọn hiển thị phía trên nội dung và danh sách các hành động được hiển thị bên dưới nội dung.

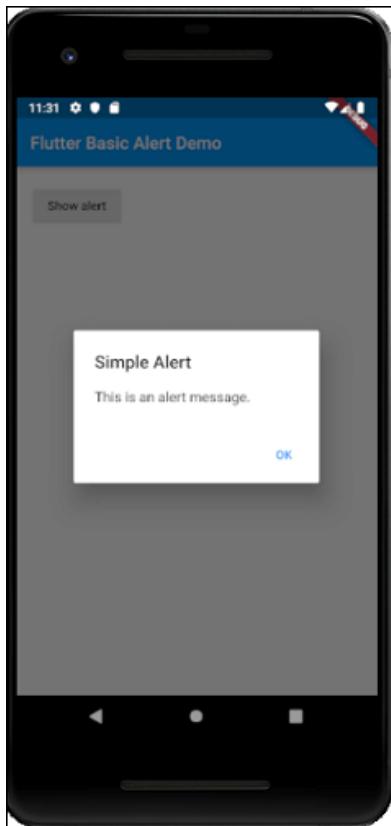
Hàm tạo và thuộc tính của Widget TextField:

```
AlertDialog({  
    super.key,  
    this.icon,  
    this.iconPadding,  
    this.iconColor,  
    this.title,  
    this.titlePadding,  
    this.titleTextStyle,  
    this.content,  
    this.contentPadding,  
    this.contentTextStyle,  
    this.actions,  
    this.actionsPadding,  
    this.actionsAlignment,  
    this.actionsOverflowAlignment,  
    this.actionsOverflowDirection,  
    this.actionsOverflowButtonSpacing,  
    this.buttonPadding,  
    this.backgroundColor,  
    this.elevation,  
    this.shadowColor,  
    this.surfaceTintColor,  
    this.semanticLabel,  
    this.insetPadding = _defaultInsetPadding,  
    this.clipBehavior = Clip.none,  
    this.shape,  
    this.alignment,  
    this.scrollable = false,  
});
```

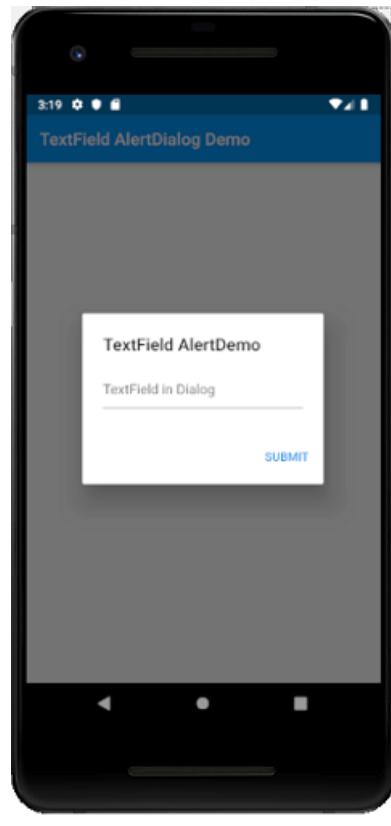
Trong đó, mô tả một số thuộc tính cơ bản như sau:

- **title:** Thuộc tính này cung cấp tiêu đề cho hộp AlertDialog nằm ở đầu AlertDialog. Luôn luôn tốt để giữ tiêu đề càng ngắn càng tốt để người dùng biết về việc sử dụng nó rất dễ dàng
- **action:** Nó hiển thị bên dưới nội dung. Ví dụ: nếu cần tạo một nút để chọn có hoặc không, thì nút đó chỉ được xác định trong thuộc tính hành động
- **content:** Thuộc tính này xác định Content của widget AlertDialog. Nó là một loại văn bản, nhưng nó cũng có thể chứa bất kỳ loại widget bô cục nào.
- **contentPadding:** Nó cung cấp phần đệm cần thiết cho nội dung bên trong tiện ích AlertDialog.
- **backgroundColor:** Màu nền của hộp thoại.
- **icon:** Hiển thị biểu tượng ở đầu hộp thoại.
- **shape:** Thuộc tính này cung cấp hình dạng cho hộp thoại cảnh báo, chẳng hạn như đường cong, hình tròn hoặc bất kỳ hình dạng khác nào khác

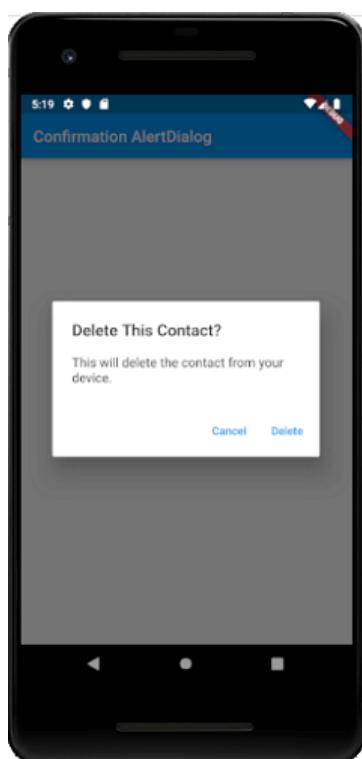
Một số dạng AlertDialog thường thấy như sau:



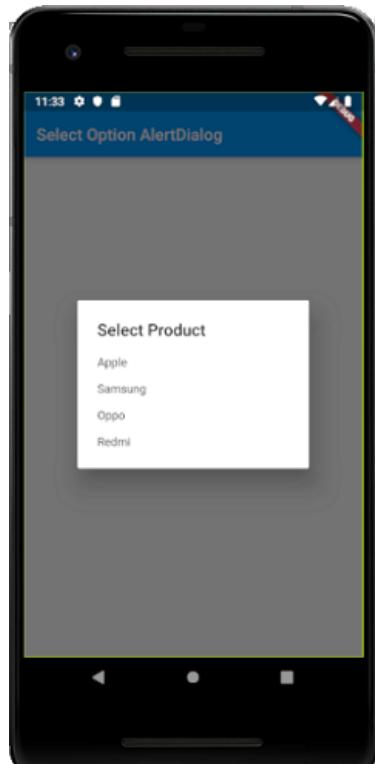
Hình 2. 39. *AlertDialog* cơ bản



Hình 2. 37. *Dạng TextField AlertDialog*



Hình 2. 36. *Dạng Confirmation AlertDialog*



Hình 2. 38. *Dạng Select Dialog*

2.2.10. Widget Icon

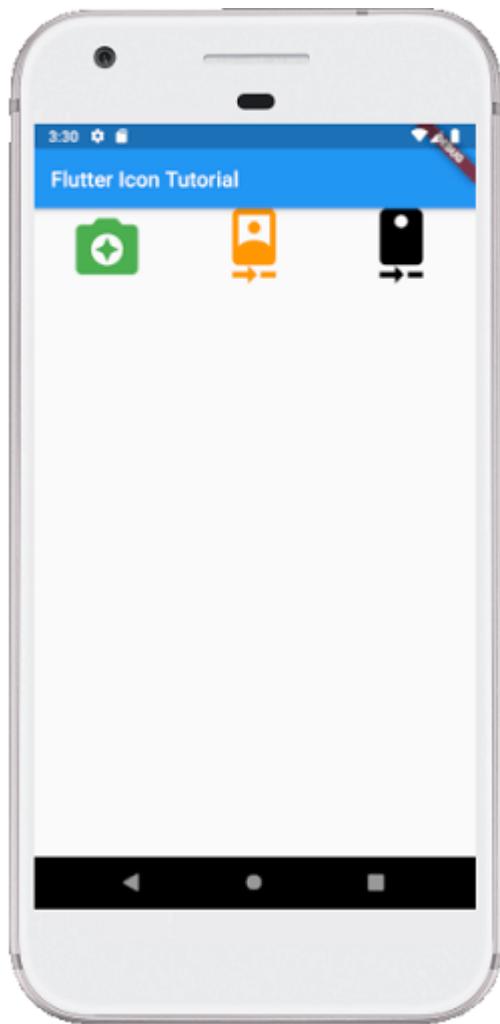
Biểu tượng(Icon) là một hình ảnh đồ họa đại diện cho một ứng dụng hoặc bất kỳ thực thể cụ thể nào có chứa ý nghĩa đối với người dùng. Nó có thể được lựa chọn và không thể lựa chọn. Ví dụ: không thể chọn được biểu tượng của công ty. Đôi khi nó cũng chứa một siêu liên kết để đi đến một trang khác. Nó cũng hoạt động như một dấu hiệu thay cho lời giải thích chi tiết về thực thể thực tế.

Hàm tạo và thuộc tính của Widget Icon như sau:

```
Icon(
    this.icon,
    super.key,
    this.size,
    this.fill,
    this.weight,
    this.grade,
    this.opticalSize,
    this.color,
    this.shadows,
    this.semanticLabel,
    this.textDirection,
    this.applyTextScaling,
)
```

Trong đó, mô tả một số thuộc tính cơ bản như sau:

- **icon:** Nó được sử dụng để chỉ định tên biểu tượng để hiển thị trong ứng dụng. Nói chung, Flutter sử dụng các biểu tượng material design là biểu tượng cho các hành động và mục phổ biến.
- **color:** Nó được sử dụng để chỉ định màu của biểu tượng.
- **size:** Nó được sử dụng để chỉ định kích thước của biểu tượng theo pixel. Thông thường, các biểu tượng có chiều cao và chiều rộng bằng nhau.
- **textDirection:** Nó được sử dụng để chỉ định hướng biểu tượng sẽ được hiển thị.



Hình 2. 40. Một số ví dụ về Widget Icon

2.2.11. Widget Image

Widget Image là một trong những widget cơ bản nhất trong Flutter, được sử dụng để hiển thị hình ảnh. Widget này có thể hiển thị hình ảnh từ nhiều nguồn khác nhau, bao gồm:

1. Tài nguyên assets: Hình ảnh được lưu trữ trong thư mục assets của dự án Flutter.
2. Mạng: Hình ảnh được tải từ internet thông qua URL.
3. Bộ nhớ: Hình ảnh được lưu trữ trong bộ nhớ của thiết bị.

Ưu điểm:

- Dễ sử dụng.
- Hỗ trợ nhiều định dạng hình ảnh phổ biến.
- Hiệu suất cao.
- Có thể tùy chỉnh nhiều tính năng như kích thước, vị trí, màu sắc, v.v.

Nhược điểm:

- Không hỗ trợ hiển thị ảnh động.
- Khó khăn trong việc xử lý các hình ảnh phức tạp.

Widget Image có cấu trúc khá đơn giản, bao gồm các thuộc tính sau:

- **image:** Thuộc tính này bắt buộc, chỉ định nguồn hình ảnh.
- **height:** Chiều cao của hình ảnh.
- **width:** Chiều rộng của hình ảnh.
- **fit:** Cách thức hình ảnh được điều chỉnh cho phù hợp với kích thước của widget.
- **alignment:** Vị trí của hình ảnh trong widget.
- **color:** Màu sắc được sử dụng để tô màu cho hình ảnh (tùy chọn).
- **filterQuality:** Chất lượng của bộ lọc hình ảnh (tùy chọn).

Các bước hiển thị hình ảnh trong Flutter:

- **Bước 1:** Đầu tiên, chúng ta cần tạo một thư mục mới bên trong thư mục gốc của dự án Flutter và đặt tên cho nó là property. Chúng ta cũng có thể đặt cho nó bất kỳ tên nào khác nếu bạn muốn.
- **Bước 2:** Tiếp theo, bên trong thư mục này, thêm một hình ảnh theo cách thủ công.
- **Bước 3:** Cập nhật tệp pubspec.yaml . Giả sử tên hình ảnh là hang.png, thì tệp pubspec.yaml là:

```

}
  uses-material-design: true
)
assets:
  - assets/hang.png
?

```

Hình 2. 41. Cấu hình pubspec.yaml cho Image 1

Nếu thư mục nội dung chứa nhiều hơn một hình ảnh, chúng ta có thể bao gồm nó bằng cách chỉ định tên thư mục với ký tự dấu gạch chéo (/) ở cuối.

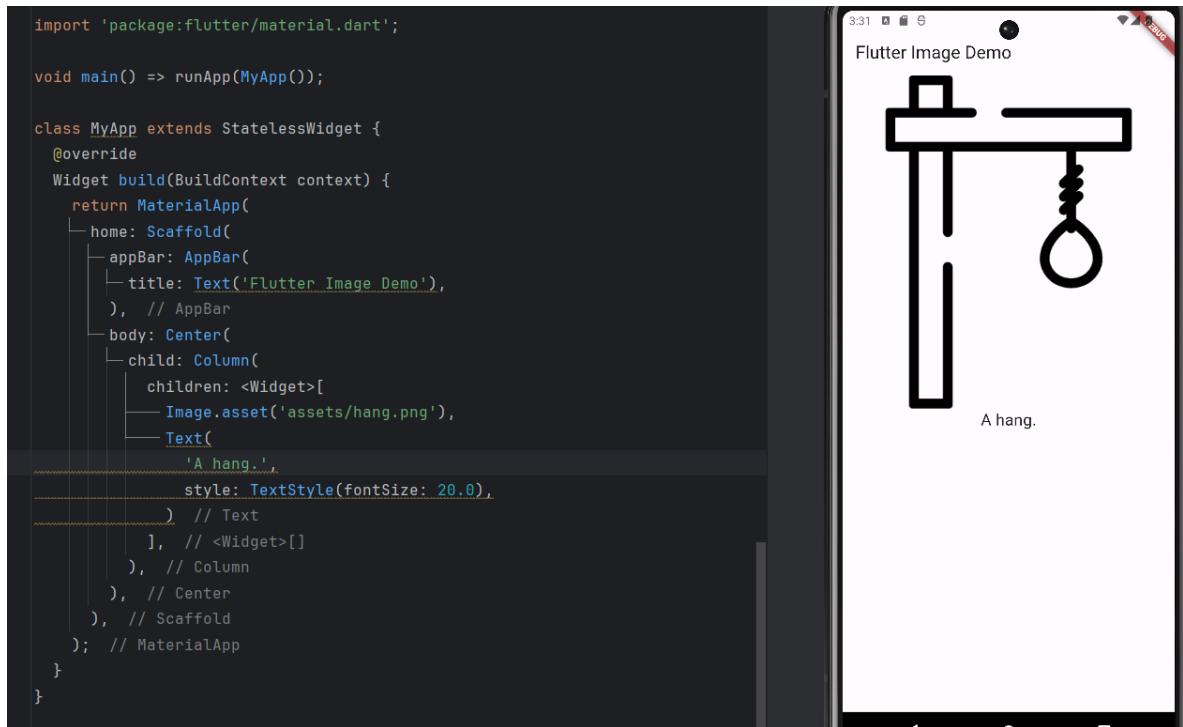
```

59   uses-material-design: true
60
61 assets:
  - assets/
?

```

Hình 2. 42. Cấu hình pubspec.yaml cho Image 2

- **Bước 4:** Cuối cùng, mở tệp chạy main.dart. Thì sẽ nhận được hình ảnh trong ứng dụng demo như dưới đây.



Hình 2. 43. Demo hiển thị hình ảnh trong Flutter

Hiển thị hình ảnh từ internet

Hiển thị hình ảnh từ internet hoặc mạng rất đơn giản. Flutter cung cấp một phương pháp tích hợp Image.network để làm việc với hình ảnh từ một URL. Phương thức Image.network cũng cho phép bạn sử dụng một số thuộc tính tùy chọn, chẳng hạn như chiều cao, chiều rộng, màu sắc, độ vừa vặn và nhiều thuộc tính khác. Chúng ta có thể sử dụng cú pháp sau để hiển thị hình ảnh từ internet.

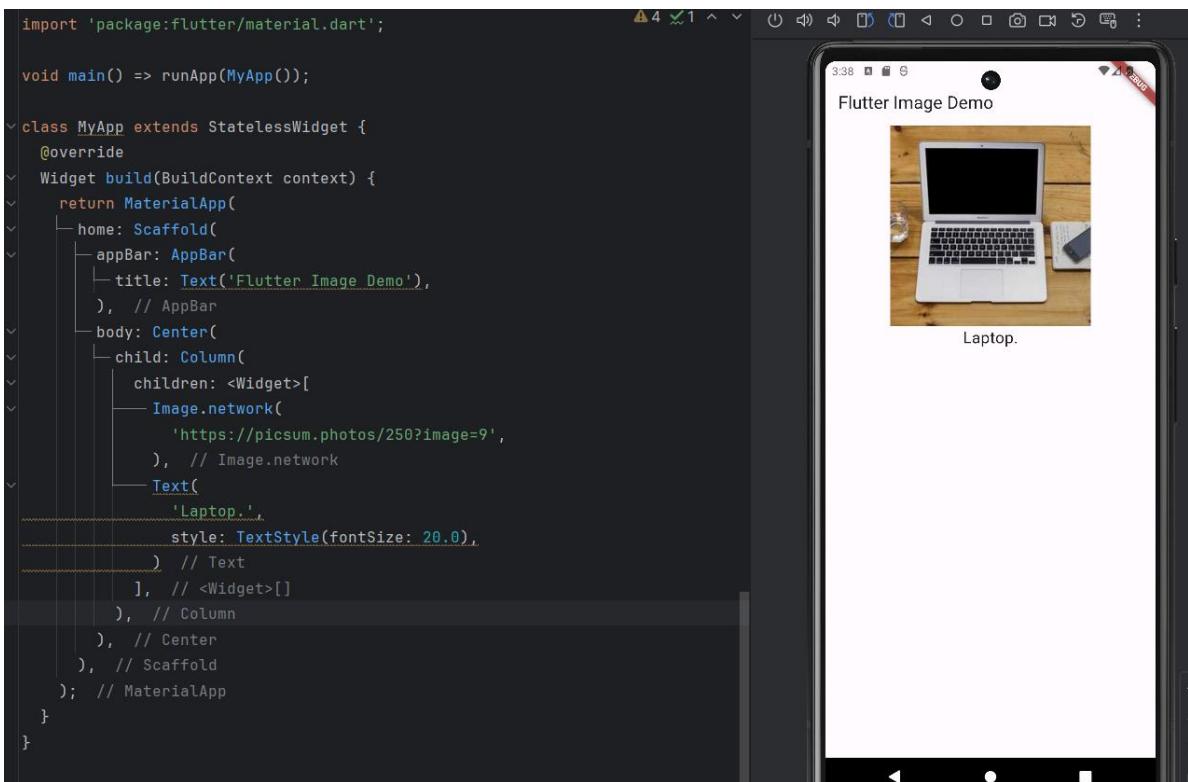
```

- Image.network(
  'https://picsum.photos/250?image=9',
), // Image.network

```

Hình 2. 44. Câu lệnh widget Image hiển thị ảnh từ Internet

Sau khi sử dụng cú pháp vào đoạn code trên ta sẽ nhận được hình ảnh trong ứng như sau.



Hình 2. 45. Demo hiển thị hình ảnh từ Internet

2.2.12. Widget Card

Card là một widget trong Flutter được sử dụng để hiển thị nội dung một cách đơn giản và trực quan. Nó có các góc bo tròn và đố bóng, tạo hiệu ứng 3D giúp nội dung nổi bật hơn.

Ưu điểm:

- Hiển thị nội dung trực quan và thu hút
- Dễ sử dụng và tùy chỉnh
- Có thể sử dụng cho nhiều mục đích khác nhau

Nhược điểm:

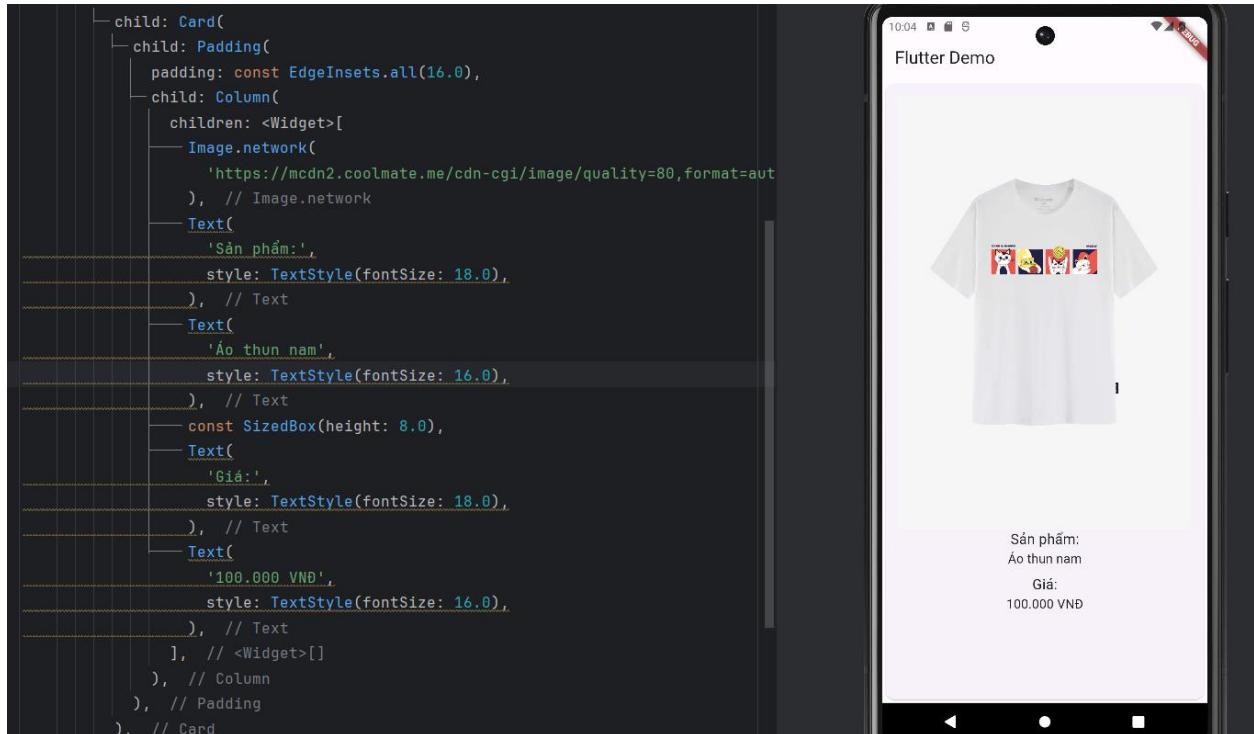
- Có thể chiếm nhiều không gian nếu không được thiết kế hợp lý

Chúng ta có thể tùy chỉnh Card bằng cách sử dụng các thuộc tính. Một số thuộc tính cần thiết được đưa ra dưới đây:

Tên thuộc tính	Mô tả
----------------	-------

borderOnForeground	Nó được sử dụng để vẽ đường viền phía trước của một child. Theo mặc định, nó là true. Nếu nó là false, nó đã vẽ đường viền phía sau đứa trẻ.
color	Nó được sử dụng để tô màu nền của Card.
elevation	Nó kiểm soát kích thước bóng bên dưới Card. Giá trị độ cao lớn hơn làm cho khoảng cách bóng lớn hơn.
margin	Nó được sử dụng để tùy chỉnh không gian bên ngoài của Card.
shape	Nó được sử dụng để chỉ định hình dạng của Card.
shadowColor	Nó được sử dụng để vẽ bóng của Card.
clipBehavior	Nó được sử dụng để kẹp nội dung của Card.

Chúng ta sẽ tạo một widget Card hiển thị thông tin chi tiết sản phẩm áo thun.



Hình 2. 46. Demo hiển thị cho widget Card

2.2.13. Widget Tabbar

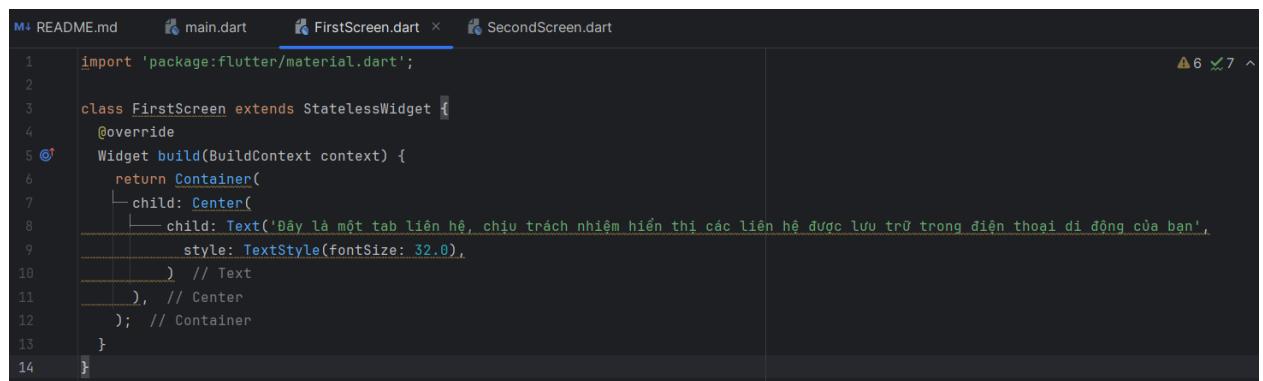
TabBar là một widget quan trọng trong Flutter giúp tạo giao diện, cho phép người dùng chuyển đổi giữa các nội dung khác nhau trong ứng dụng. TabBar thường được đặt ở đầu hoặc cuối màn hình, hiển thị các tab với tiêu đề và biểu tượng. Khi người dùng nhấp vào một tab, nội dung tương ứng sẽ được hiển thị.

TabBar bao gồm các thành phần chính sau:

- **TabBar:** Widget hiển thị các tab.
- **Tab:** Widget đại diện cho một tab riêng lẻ.
- **TabBarView:** Widget hiển thị nội dung tương ứng với tab được chọn.
- **TabController:** Widget quản lý trạng thái của TabBar, bao gồm tab nào đang được chọn.

Các bước tạo Tabbar trong dự án Flutter:

- **Bước 1:** Tạo một dự án Flutter trong IDE của mình
- **Bước 2:** Tạo hai tệp dart và đặt tên là FirstScreen và SecondScreen. Viết code sau trong FirstScreen tương tự với SecondScreen:



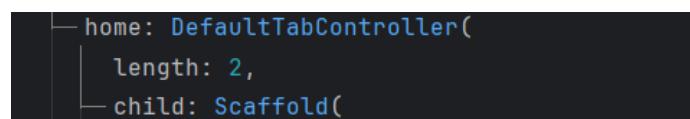
```

1 import 'package:flutter/material.dart';
2
3 class FirstScreen extends StatelessWidget {
4   @override
5   Widget build(BuildContext context) {
6     return Container(
7       child: Center(
8         child: Text('Đây là một tab liên hệ, chịu trách nhiệm hiển thị các liên hệ được lưu trữ trong điện thoại di động của bạn',
9                     style: TextStyle(fontSize: 32.0),
10                    ), // Text
11      ), // Center
12    ); // Container
13  }
14 }

```

Hình 2. 47. Tạo First Screen và Second Screen cho ví dụ Tabbar

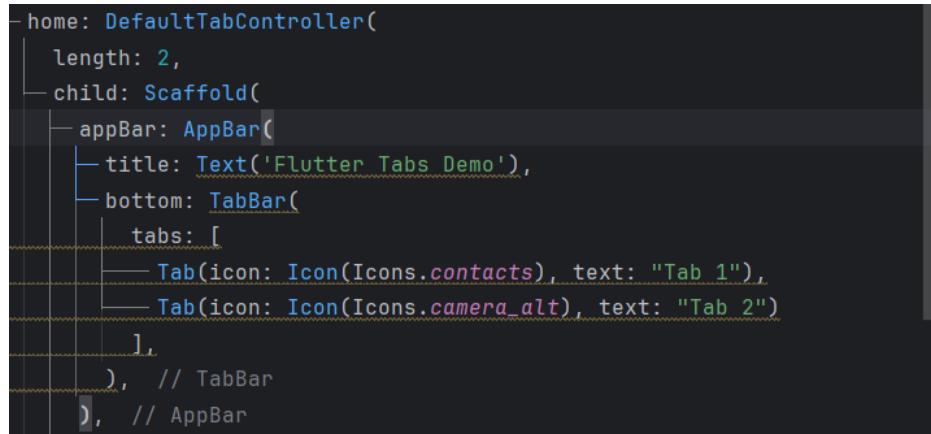
- **Bước 3:** Tiếp theo, chúng ta cần tạo một DefaultTabController. DefaultTabController tạo TabController và cung cấp nó cho tất cả các widget.



Hình 2. 48. Tạo DefaultTabController cho ví dụ Tabbar

Thuộc tính length cho biết số lượng tab được sử dụng trong ứng dụng.

- **Bước 4:** Tạo tab. Chúng ta có thể tạo các tab bằng cách sử dụng widget TabBar như đoạn code dưới đây.



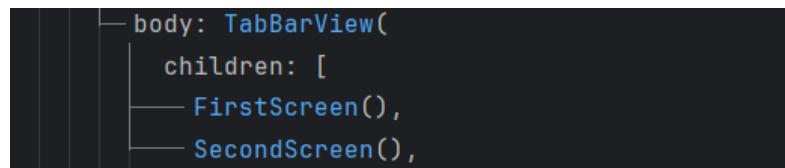
```

- home: DefaultTabController(
  |   length: 2,
  |   child: Scaffold(
  |     appBar: AppBar(
  |       title: Text('Flutter Tabs Demo'),
  |       bottom: TabBar(
  |         tabs: [
  |           Tab(icon: Icon(Icons.contacts), text: "Tab 1"),
  |           Tab(icon: Icon(Icons.camera_alt), text: "Tab 2")
  |         ],
  |       ),
  |     ),
  |   ),
  | )

```

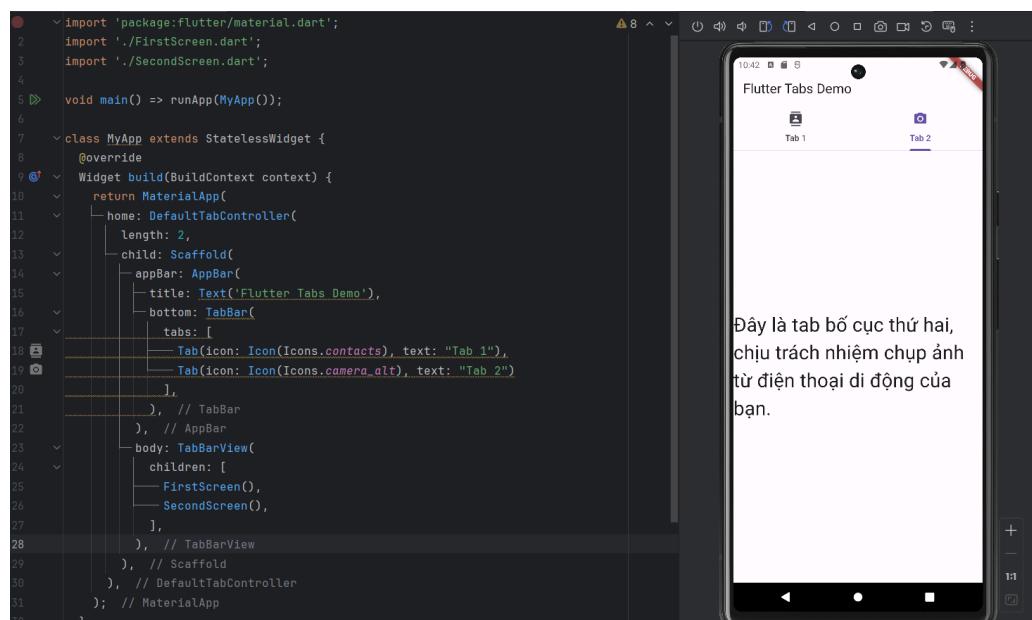
Hình 2. 49. Tạo Tabbar trong màn hình chính

- **Bước 5:** Tạo nội dung cho từng tab để khi chọn tab sẽ hiển thị nội dung. Với mục đích này, chúng ta phải sử dụng widget TabBarView như:



Hình 2. 50. Cấu hình đường dẫn tới First Screen và Second Screen cho ví dụ Tabbar

- **Bước 6:** Cuối cùng, mở tệp main.dart và chèn code sau



```

1 import 'package:flutter/material.dart';
2 import './FirstScreen.dart';
3 import './SecondScreen.dart';
4
5 void main() => runApp(MyApp());
6
7 class MyApp extends StatelessWidget {
8   @override
9   Widget build(BuildContext context) {
10     return MaterialApp(
11       home: DefaultTabController(
12         length: 2,
13         child: Scaffold(
14           appBar: AppBar(
15             title: Text('Flutter Tabs Demo'),
16             bottom: TabBar(
17               tabs: [
18                 Tab(icon: Icon(Icons.contacts), text: "Tab 1"),
19                 Tab(icon: Icon(Icons.camera_alt), text: "Tab 2")
20               ],
21             ),
22           ),
23           body: TabBarView(
24             children: [
25               FirstScreen(),
26               SecondScreen(),
27             ],
28           ),
29         ),
30       ),
31     );
32   }

```

Hình 2. 51. Demo hoàn chỉnh cho ví dụ Tabbar

2.2.14. Widget Drawer

Drawer là một lựa chọn thay thế cho các tab vì đôi khi các ứng dụng dành cho thiết bị di động không có đủ không gian để hỗ trợ các tab. Drawer là một màn hình bên vô hình. Đây là một menu trượt bên trái thường chứa các liên kết quan trọng trong ứng dụng và chiếm một nửa màn hình khi hiển thị.

Drawer bao gồm các thành phần chính sau:

- **Drawer:** Widget hiển thị nội dung menu.
- **DrawerHeader:** Widget hiển thị phần đầu của menu, thường bao gồm logo, tên ứng dụng hoặc thông tin người dùng.
- **DrawerItem:** Widget đại diện cho một mục trong menu.
- **ListView:** Widget hiển thị danh sách các mục menu.

Các bước tạo ra một giao diện Flutter sử dụng Widget Drawer:

- **Bước 1:** Tạo một dự án Flutter trong IDE.
- **Bước 2:** Mở dự án điều hướng đến thư mục lib. Trong thư mục này, hãy mở tệp main.dart.
- **Bước 3:** Trong tệp main.dart, hãy tạo một Drawer trong tiện ích

```
return Scaffold(
  appBar: AppBar(title: Text(title)),
  body: Center(child: Text(...) // Text
), // Center
  drawer: Drawer(
    child: ListView(
```

Hình 2. 52. Tạo một Drawer trong ứng dụng Flutter vừa khởi tạo

- **Bước 4:** Tiếp theo, chúng ta cần thêm nội dung trong Drawer. Trong ví dụ này, chúng ta sẽ sử dụng tiện ích ListView cho phép người dùng cuộn qua ngắn nếu nội dung không vừa với màn hình hỗ trợ.

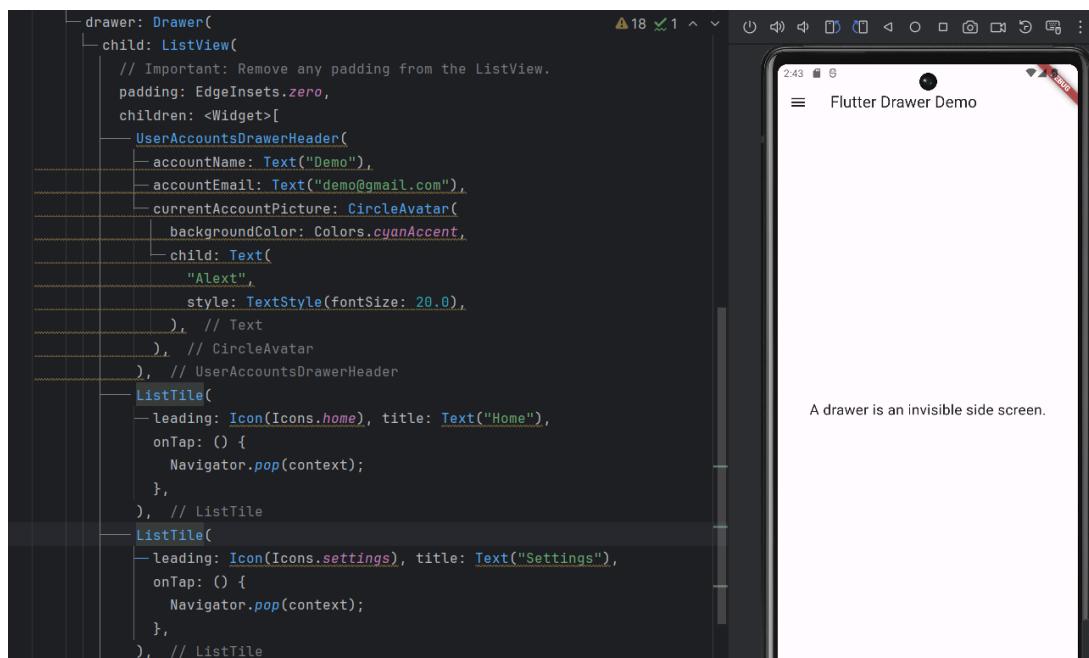
```

    drawer: Drawer(
      child: ListView(
        // Important: Remove any padding from the ListView.
        padding: EdgeInsets.zero,
        children: <Widget>[
          UserAccountsDrawerHeader(
            accountName: Text("Demo"),
            accountEmail: Text("demo@gmail.com"),
            currentAccountPicture: CircleAvatar(
              backgroundColor: Colors.cyanAccent,
              child: Text(
                "Alext",
                style: TextStyle(fontSize: 20.0),
              ),
            ), // CircleAvatar
          ), // UserAccountsDrawerHeader
          ListTile(
            leading: Icon(Icons.home), title: Text("Home"),
            onTap: () {
              Navigator.pop(context);
            },
          ), // ListTile
        ],
      ),
    ),
  ),
)

```

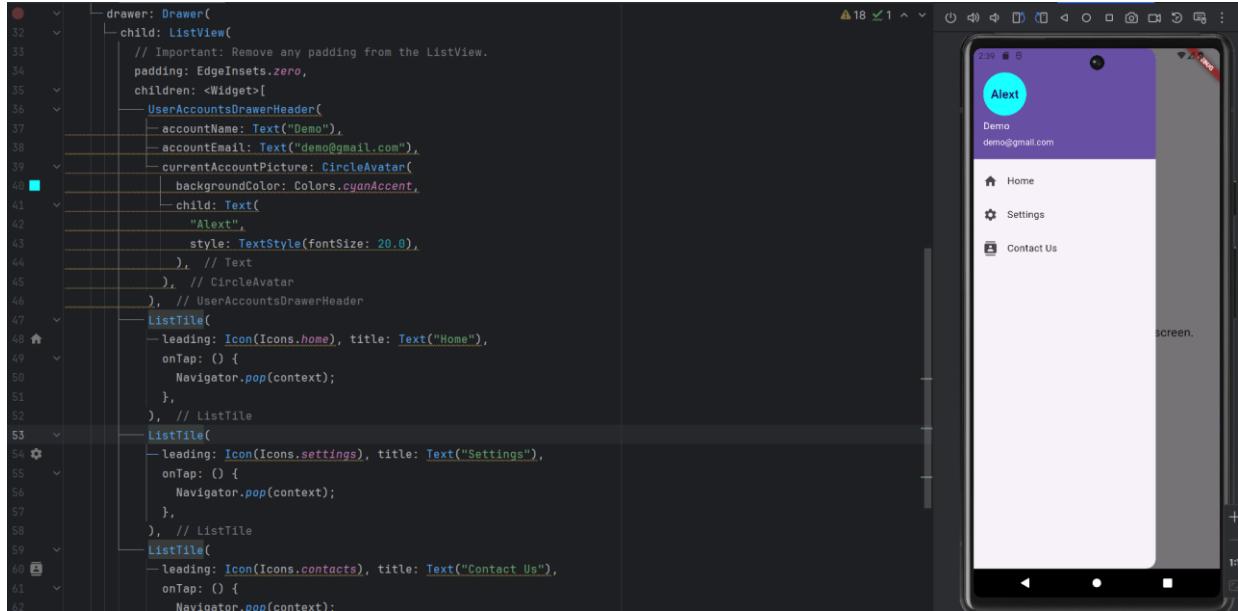
Hình 2. 53. Thêm nội dung vào trong Drawer

- **Bước 5:** Cuối cùng, đóng Drawer lại. Chúng ta có thể làm điều này bằng cách sử dụng công cụ navigator (điều hướng). Nó sẽ đưa ra màn hình sau.



Hình 2. 54. Khởi động máy ảo trong ví dụ Drawer

Khi nhấp vào góc trên cùng bên trái của màn hình trên, bạn có thể thấy Drawer trượt sang trái thường chứa các liên kết quan trọng trong ứng dụng và chiếm một nửa màn hình.



Hình 2. 55. Demo chương trình Flutter sử dụng Widget Drawer

2.2.15. Widget ListView

Widget ListView là một widget cơ bản trong Flutter, được sử dụng để hiển thị danh sách các widget con. Nó có thể hiển thị danh sách các widget theo chiều dọc hoặc chiều ngang, với khả năng cuộn nếu danh sách dài hơn màn hình. Cấu trúc của List như sau:

```
ListView(
  children: <Widget>[
    // Danh sách các widget con
  ],
)
```

Tham số quan trọng:

- **children:** Danh sách các widget con được hiển thị trong danh sách.

Ưu điểm:

- Dễ sử dụng và triển khai.
- Hiệu quả và có thể xử lý danh sách lớn.
- Hỗ trợ nhiều loại widget con khác nhau.
- Có thể tùy chỉnh giao diện của danh sách.

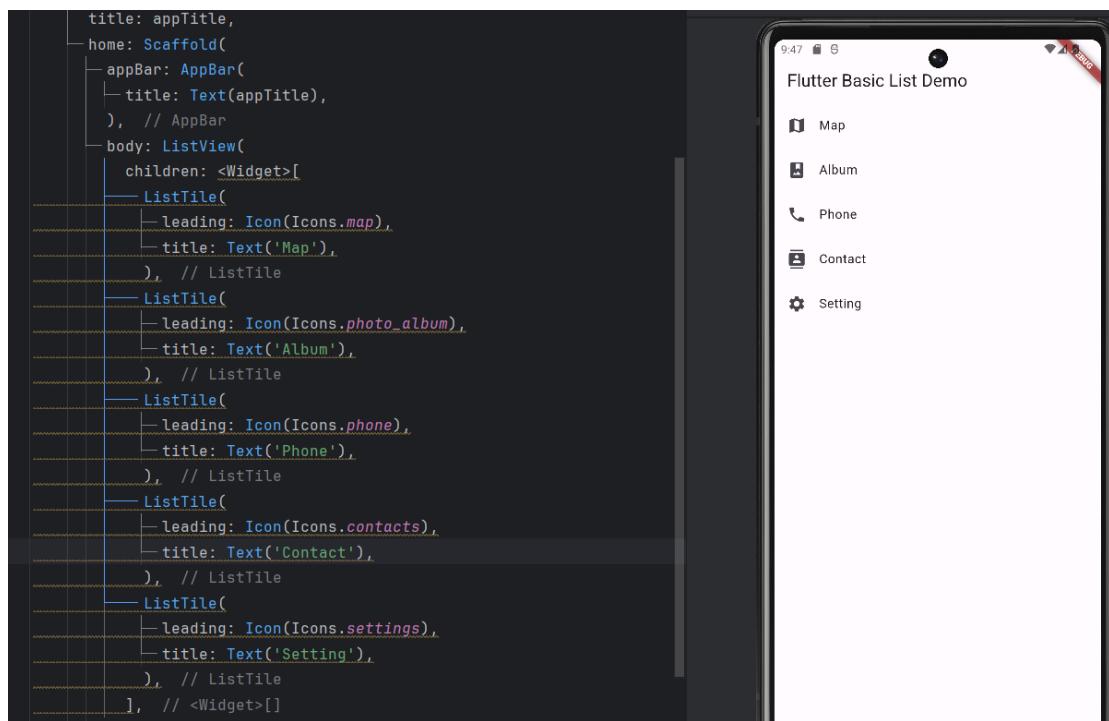
Nhược điểm:

- Có thể gặp vấn đề về hiệu suất nếu danh sách quá lớn.
- Khó khăn trong việc tùy chỉnh giao diện phức tạp.

Một số loại ListView khác:

1. ListView: Hiển thị danh sách các widget theo chiều dọc.
2. GridView: Hiển thị danh sách các widget theo dạng lưới.
3. CustomScrollView: Cho phép tùy chỉnh chi tiết cách thức cuộn của danh sách.

Demo một giao diện Flutter cơ bản có sử dụng ListView:



Hình 2. 56. Một giao diện Flutter Cơ bản có sử dụng ListView

2.2.16. Widget GridView

GridView là một widget trong Flutter giúp hiển thị danh sách các item dưới dạng lưới 2D. Nó tương tự như ListView nhưng hiển thị các item theo hàng và cột thay vì chỉ theo một chiều. Nó có thể được sử dụng để tạo ra các bộ cục như danh sách ảnh, danh sách sản phẩm, v.v.

Ưu điểm:

- Hiển thị danh sách các item một cách trực quan và dễ nhìn.
- Có thể tùy chỉnh bộ cục của lưới, chẳng hạn như số lượng cột, khoảng cách giữa các item, v.v.
- Hỗ trợ cuộn trang để hiển thị nhiều item hơn.

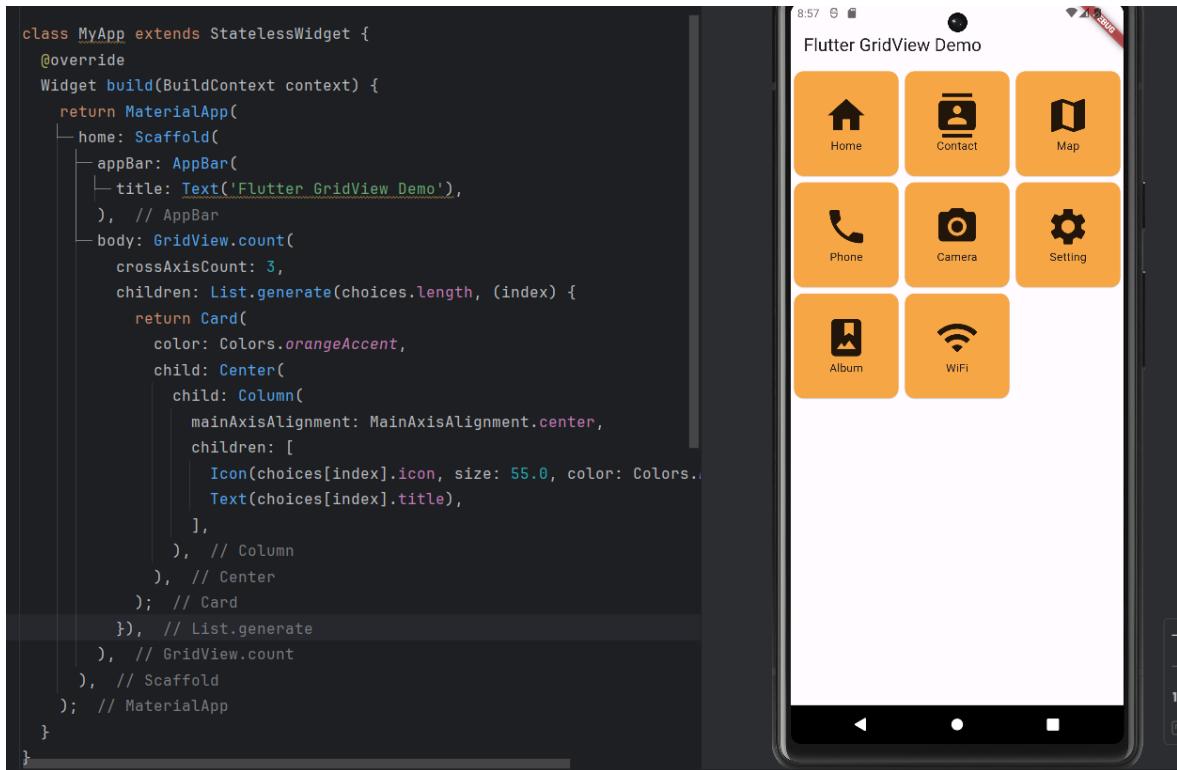
Nhược điểm:

- Có thể phức tạp hơn so với ListView khi tùy chỉnh bộ cục.
- Hiệu suất có thể bị ảnh hưởng nếu có quá nhiều item trong danh sách.

GridView được cấu tạo từ các thành phần sau:

- **children:** Danh sách các widget sẽ được hiển thị trong GridView.
- **gridDelegate:** Một đối tượng xác định cách thức sắp xếp các item trong GridView.
- **scrollDirection:** Hướng cuộn của GridView (theo chiều ngang hoặc chiều dọc).
- **shrinkWrap:** Thu hẹp kích thước của GridView để vừa với nội dung.

Demo một giao diện Flutter cơ bản có sử dụng GridView:



Hình 2. 57. Một giao diện Flutter Cơ bản có sử dụng GridView

2.2.17. Widget CheckBox

Checkbox là một widget Material Design cho phép người dùng chọn một hoặc nhiều tùy chọn từ một danh sách. Nó hiển thị một hộp vuông nhỏ với dấu kiểm bên trong khi được chọn.

Cấu trúc của một Widget CheckBox được mô tả như sau:

Thuộc tính	Mô tả
value	Nó được sử dụng cho dù Checkbox có được chọn hay không.
onChanged	Nó sẽ được gọi khi giá trị được thay đổi.
Tristate	Nó là false, theo mặc định. Giá trị của nó cũng có thể là true, false hoặc null.

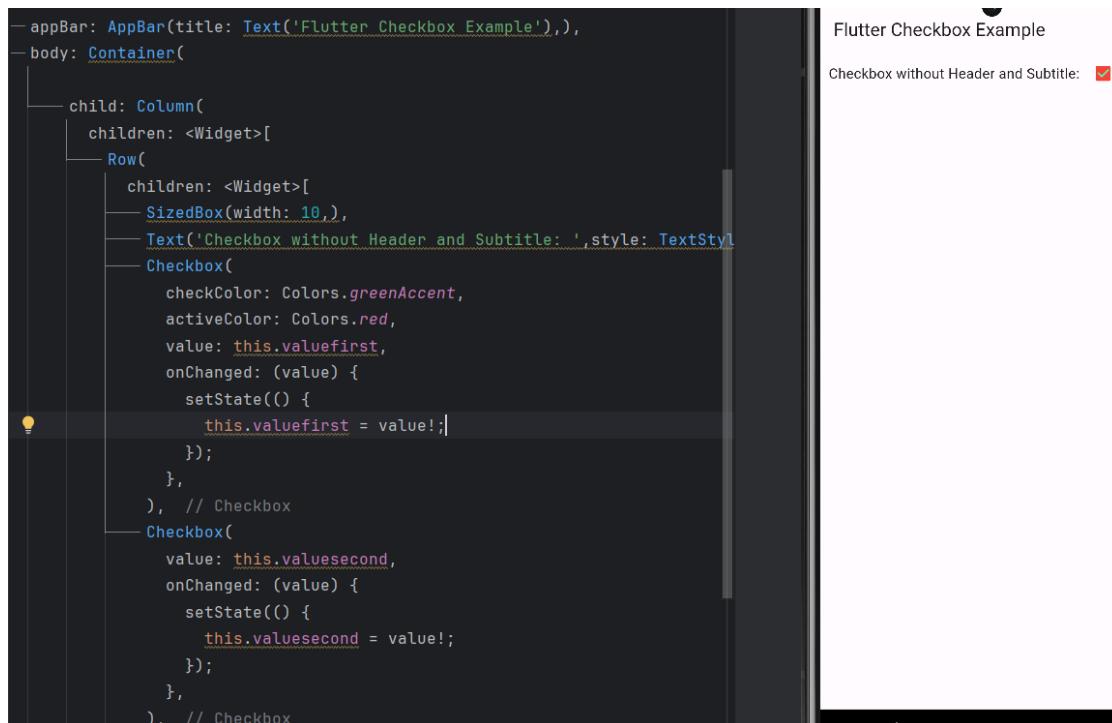
activeColor	Nó chỉ định màu của Checkbox đã chọn.
checkColor	Nó chỉ định màu của biểu tượng kiểm tra khi chúng được chọn.
materialTapTargetSize	Nó được sử dụng để định cấu hình kích thước của mục tiêu chạm.

Demo một giao diện Flutter cơ bản có sử dụng Checkbox:

```
— Checkbox(
    value: this.valuesecond,
    onChanged: (value) {
        setState(() {
            this.valuesecond = value!;
        });
    },
), // Checkbox
```

Hình 2. 58. Đoạn code demo cho checkbox

Cách Checkbox được hiển thị trong Flutter. Từ đoạn code bên dưới thì chúng ta sẽ thấy màn hình bên dưới:



Hình 2. 59. Một giao diện Flutter Cơ bản có sử dụng Checkbox

2.2.18. Widget RadioButton

RadioButton là một widget material design cho phép người dùng chọn một tùy chọn duy nhất từ một danh sách các lựa chọn. Widget này thường được sử dụng trong các biểu mẫu để thu thập thông tin từ người dùng.

RadioButton là một widget stateless, nghĩa là nó không có trạng thái riêng. Widget này nhận một số thuộc tính để tùy chỉnh giao diện và chức năng:

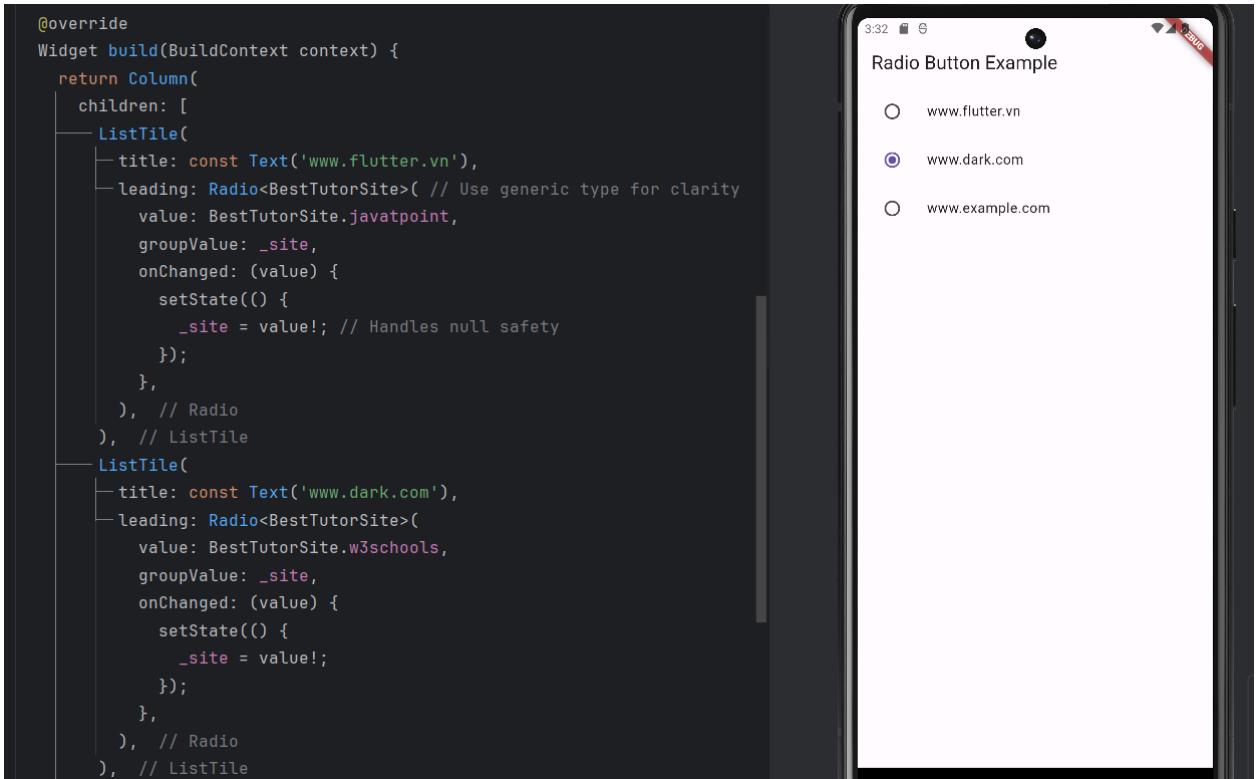
- **groupValue**: Nó được sử dụng để chỉ định mục hiện được chọn cho nhóm nút radio.
- **title**: Nó được sử dụng để chỉ định nhãn nút radio.
- **value**: Nó chỉ định giá trị trái, được hiển thị bằng một nút radio.
- **onChanged**: Nó sẽ được gọi bất cứ khi nào người dùng chọn nút radio.

Demo một giao diện Flutter cơ bản có sử dụng Checkbox:

```
ListTile(
  title: const Text('www.flutter.vn'),
  leading: Radio<BestTutorSite>(
    value: BestTutorSite.javatpoint,
    groupValue: _site,
    onChanged: (value) {
      setState(() {
        _site = value!; // Handles null safety
      });
    },
  ), // Radio
), // ListTile
```

Hình 2. 60. Đoạn code demo cho radioButton

Ví dụ các widget Radio được bao bọc trong ListTile và văn bản hiện đang được chọn được chuyển vào groupValue và được duy trì bởi Trạng thái của ví dụ. Tại đây, nút Radio đầu tiên sẽ bị tắt vì _site được khởi tạo thành BestTutorSite.fluttervn. Nếu nút radio thứ hai được nhấn, Trạng thái của ví dụ được cập nhật với setState, cập nhật _site thành BestTutorSite.dark. Nó xây dựng lại nút với groupValue được cập nhật và do đó nó sẽ chọn nút thứ hai.



Hình 2. 61. Một giao diện Flutter Cơ bản sử dụng radioButton

2.2.19. Widget ProgressBar

Progress bar (Thanh tiến trình) là một phần tử điều khiển đồ họa được sử dụng để hiển thị tiến trình của một tác vụ như tải xuống, tải lên, cài đặt, truyền tệp, v.v. Trong phần này, chúng ta sẽ hiểu cách hiển thị progress bar trong một ứng dụng rung.

Flutter có thể hiển thị progress bar với sự trợ giúp của hai widget con, được đưa ra dưới đây:

- **LinearProgressIndicator**
- **CircularProgressIndicator**

Cấu trúc của widget ProgressBar trong Flutter khá đơn giản. Ta có thể sử dụng widget **LinearProgressIndicator** hoặc **CircularProgressIndicator** để tạo ProgressBar cho giao diện của mình:

```

LinearProgressIndicator(
  value: 0.5, // Giá trị tiến trình (từ 0.0 đến 1.0)
  backgroundColor: Colors.grey, // Màu nền
)

```

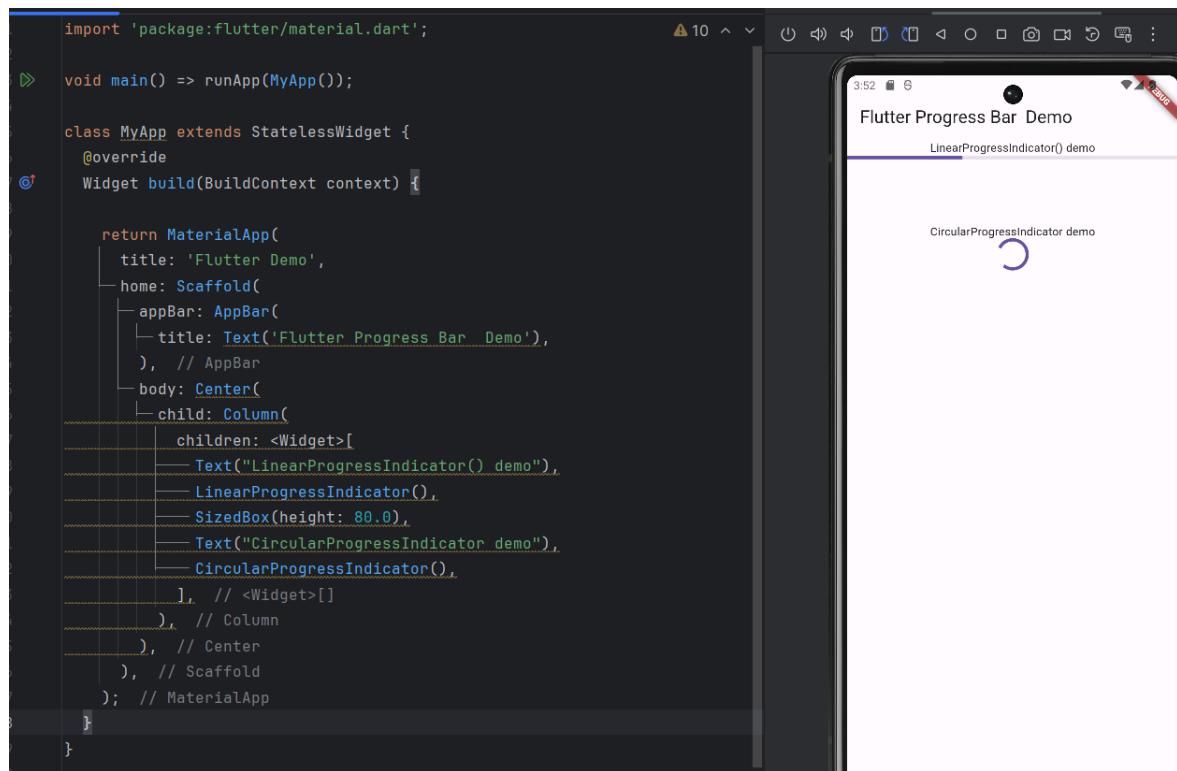
```

    valueColor: Colors.blue, // Màu phần được tô màu
)
CircularProgressIndicator(
    value: 0.5, // Giá trị tiến trình (từ 0.0 đến 1.0)
    backgroundColor: Colors.grey, // Màu nền
    valueColor: Colors.blue, // Màu phần được tô màu
)

```

Demo một giao diện Flutter cơ bản có sử dụng cả 2 dạng ProgressBar:

Ví dụ chúng ta sẽ thấy đầu ra của chỉ báo tiến trình chạy theo chiều ngang và vòng tròn flutter như ảnh chụp màn hình bên dưới.



Hình 2. 62. Một giao diện Flutter Cơ bản sử dụng cả 2 dạng ProgressBar

2.2.20. Widget Snackbar

Snackbar trong Flutter là một widget hiển thị thông báo nhẹ thông báo ngắn gọn cho người dùng khi một số hành động nhất định xảy ra.

Ưu điểm: Widget Snackbar trong Flutter được sử dụng để hiển thị các thông báo ngắn gọn cho người dùng, giúp tạo ra trải nghiệm tương tác trực quan. Snackbar thường được sử dụng để thông báo thành công, lỗi hoặc hành động của người dùng.

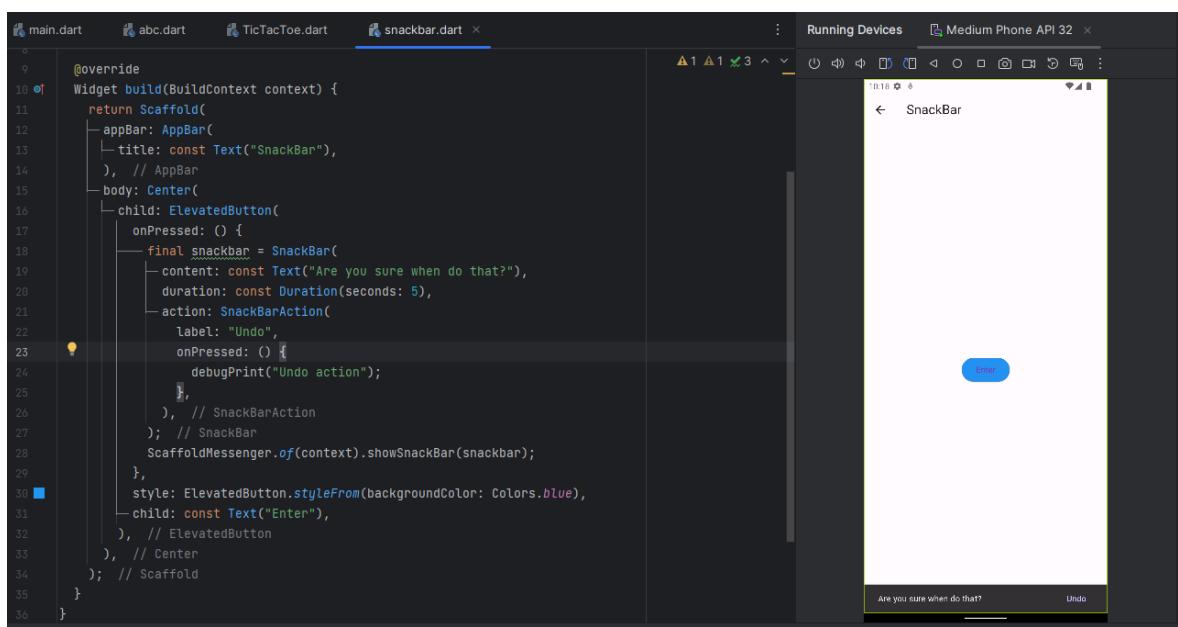
Nhược điểm: Snackbar chỉ hiển thị trong một khoảng thời gian ngắn và biến mất sau một khoảng thời gian cố định, do đó không thích hợp cho các thông báo cần được giữ lại lâu dài hoặc cần sự tương tác của người dùng.

```
const SnackBar({  
    super.key,  
    required this.content,  
    this.backgroundColor,  
    this.elevation,  
    this.margin,  
    this.padding,  
    this.width,  
    this.shape,  
    this.hitTestBehavior,  
    this.behavior,  
    this.action,  
    this.actionOverflowThreshold,  
    this.showCloseIcon,  
    this.closeIconColor,  
    this.duration = _snackBarDisplayDuration,  
    this.animation,  
    this.onVisible,  
    this.dismissDirection,  
    this.clipBehavior = Clip.hardEdge,  
})
```

Các thuộc tính có trong Snackbar

- **content:** Đây là nội dung chính của thanh snack nhanh, thực chất là một widget vsnack bản.

- **duration:** Được sử dụng để chỉ định thời gian hiển thị quầy bar bán đồ snack nhanh.
- **action:** Được sử dụng để thực hiện hành động khi người dùng chạm vào bar snack nhanh. Nó không thể bị loại bỏ hoặc hủy bỏ. Chúng ta chỉ có thể hoàn tác hoặc làm lại trong snack bar nhanh.
- **elevation:** Là tọa độ z nơi đặt quầy bar bán đồ snack bar nhanh. Nó được sử dụng để kiểm soát kích thước bóng bên dưới thanh snack bar nhanh.
- **shape:** Được sử dụng để tùy chỉnh hình dạng của một snack bar nhanh.
- **behavior:** Được sử dụng để đặt vị trí của snack bar nhanh.
- **backgroundcolor:** Chỉ định nền của snack bar nhanh.
- **animation:** Xác định lối ra và lối vào của snack bar nhanh.



Hình 2. 63. Một ví dụ minh họa cho Widget SnackBar

2.2.21. Widget Tooltip

Tooltip là một lớp thiết kế material design trong Flutter **cung cấp các nhãn văn bản để giải thích chức năng** của một nút hoặc hành động trên giao diện người dùng.

Ưu điểm: Widget Tooltip trong Flutter là một cách tiện lợi để cung cấp thông tin bổ sung cho các phần tử trong giao diện người dùng. Nó cho phép người dùng xem thông tin mô tả khi họ di chuyển con trỏ của mình qua một phần tử nhất định. Điều này giúp cải thiện trải nghiệm người dùng bằng cách cung cấp thông tin hữu ích mà không làm phiền quá nhiều không gian màn hình.

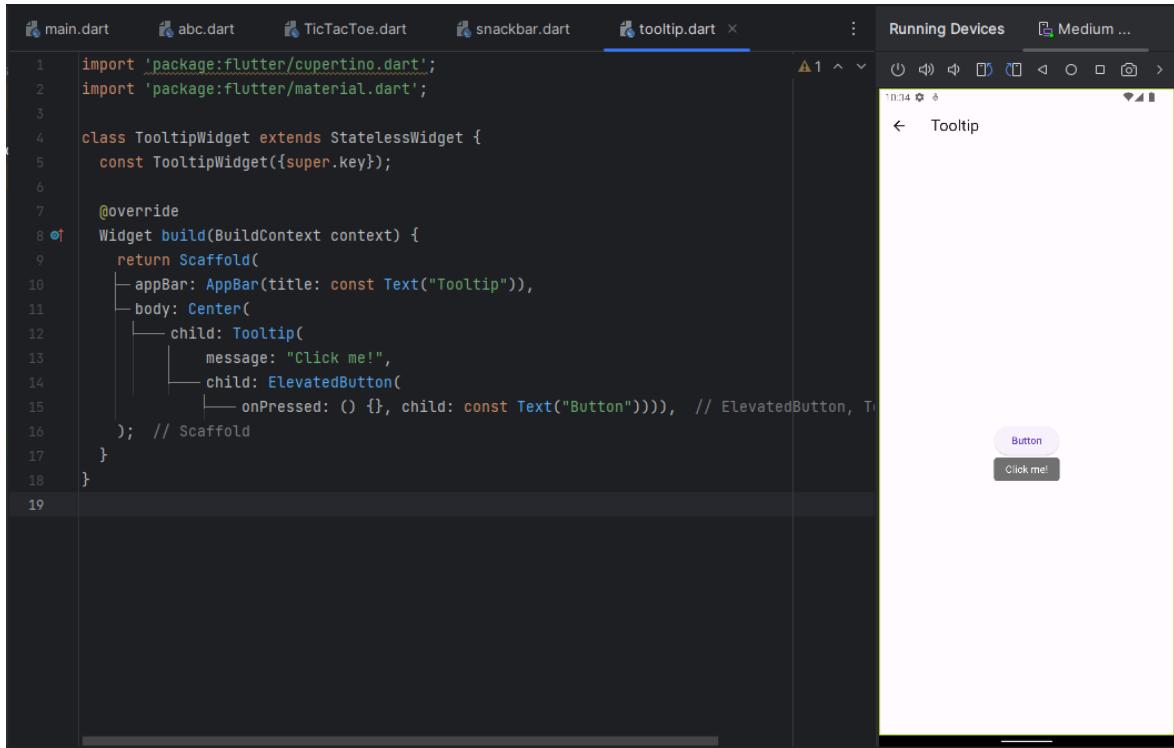
Nhược điểm: Tooltip có thể không phù hợp cho các ứng dụng mà không có sự tương tác trực tiếp với người dùng hoặc không có con trỏ chuột (ví dụ: ứng dụng di động). Ngoài ra, việc sử dụng tooltip quá nhiều có thể làm cho giao diện trở nên rối và gây khó chịu cho người dùng.

```
const Tooltip({  
    super.key,  
    this.message,  
    this.richMessage,  
    this.height,  
    this.padding,  
    this.margin,  
    this.verticalOffset,  
    this.preferBelow,  
    this.excludeFromSemantics,  
    this.decoration,  
    this.textStyle,  
    this.textAlign,  
    this.waitDuration,  
    this.showDuration,  
    this.exitDuration,  
    this.enableTapToDismiss = true,  
    this.triggerMode,  
    this.enableFeedback,  
    this.onTriggered,  
    this.child,  
}
```

})

Các thuộc tính có trong Tooltip

- **message:** Đây là một thông báo chuỗi được sử dụng để hiển thị trong chú giải công cụ.
- **height :** Được sử dụng để chỉ định chiều cao của con của chú giải công cụ.
- **textStyle :** Được sử dụng để xác định kiểu cho **message** của chú giải công cụ.
- **margin :** Được sử dụng để xác định không gian trống bao quanh chú giải công cụ.
- **showDuration :** Được sử dụng để chỉ định khoảng thời gian hiển thị chú giải công cụ sau khi nhấn giữ lâu được thả ra. Theo mặc định, nó là 1,5 giây.
- **decoration:** Được sử dụng để xác định hình dạng và màu nền của chú giải công cụ. Hình dạng chú giải công cụ mặc định là một hình chữ nhật tròn có bán kính đường viền là 4,0 PX.
- **verticalOffset :** Dùng xác định khoảng cách theo chiều dọc giữa chú giải công cụ và widget con.
- **waitDuration :** Được sử dụng để chỉ định thời gian con trỏ di chuột qua widget của chú giải công cụ trước khi hiển thị chú giải công cụ. Khi con trỏ rời khỏi widget, thông báo chú giải công cụ sẽ biến mất.
- **padding :** Dùng xác định không gian để chèn con của chú giải công cụ. Theo mặc định, nó là 16.0 PX ở tất cả các hướng.
- **likesBelow :** Được sử dụng để chỉ định xem chú giải công cụ có được hiển thị bên dưới widget hay không. Theo mặc định, nó là sự thật. Chú giải công cụ sẽ được hiển thị theo hướng ngược lại nếu chúng ta không có đủ không gian để hiển thị chú giải công cụ theo hướng ưu tiên.



Hình 2. 64. Một ví dụ minh họa cho Widget Tooltip

2.2.22. Widget Slider

Slider trong Flutter là một widget thiết kế material design được sử dụng để chọn một loạt các giá trị. Nó là một widget đầu vào, nơi chúng ta có thể **đặt một loạt các giá trị bằng cách kéo hoặc nhấn vào vị trí mong muốn**.

Ưu điểm: Widget Slider trong Flutter cung cấp một cách thuận tiện để người dùng chọn giá trị từ một phạm vi đã cho. Nó linh hoạt và dễ sử dụng, cho phép người dùng điều chỉnh giá trị bằng cách di chuyển thanh trượt. Điều này thích hợp cho các tùy chọn có giá trị liên tục như âm lượng, độ sáng, hoặc giá trị trong một khoảng thời gian.

Nhược điểm: Một số nhược điểm của Widget Slider là khi sử dụng trong các ứng dụng di động, đôi khi việc chính xác khi chọn giá trị có thể gây khó khăn cho người dùng, đặc biệt là trên các thiết bị có màn hình nhỏ.

```

const Slider({
  super.key,

```

```

required this.value,
this.secondaryTrackValue,
required this.onChanged,
this.onChangeStart,
this.onChangeEnd,
this.min = 0.0,
this.max = 1.0,
this.divisions,
this.label,
this.activeColor,
this.inactiveColor,
this.secondaryActiveColor,
this.thumbColor,
this.overlayColor,
this.mouseCursor,
this.semanticFormatterCallback,
this.focusNode,
this.autofocus = false,
this.allowedInteraction,
})

```

Các thuộc tính có trong Slider

- **value:** Là một đối số bắt buộc và được sử dụng để chỉ định giá trị hiện tại của thanh trượt.
- **onChanged:** Là một đối số bắt buộc và được gọi trong quá trình kéo khi người dùng chọn một giá trị mới cho thanh trượt. Nếu nó là null, thanh trượt sẽ bị vô hiệu hóa.
- **onChangeStart:** Là một đối số tùy chọn và được gọi khi chúng ta bắt đầu chọn một giá trị mới.

- **max:** Là một đối số tùy chọn và xác định giá trị lớn nhất có thể được sử dụng bởi người dùng. Theo mặc định, nó là 1.0. Giá trị này phải lớn hơn hoặc bằng min.
- **min:** Là một đối số tùy chọn xác định giá trị tối thiểu có thể được sử dụng bởi người dùng. Theo mặc định, nó là 0,0. Giá trị này phải nhỏ hơn hoặc bằng giá trị tối đa.
- **divisions:** Được xác định số lần phân chia rời rạc. Nếu nó là null, thanh trượt là liên tục.
- **label:** Dùng chỉ định nhãn văn bản sẽ được hiển thị phía trên thanh trượt. Nó hiển thị giá trị của một thanh trượt rời rạc.
- **activeColor:** Dùng xác định màu của phần hoạt động của rãnh trượt.
- **inactiveColor:** Dùng xác định màu của phần không hoạt động của rãnh trượt.
- **SemanticFormatterCallback:** Là một lệnh gọi lại được sử dụng để tạo ra một giá trị context. Theo mặc định, nó là một tỷ lệ phần trăm.

```

main.dart    slider.dart    TicTacToe.dart    snackbar.dart
10
11 class SliderState extends State<SliderWidget> {
12   int _value = 4;
13   @override
14   Widget build(BuildContext context) {
15     return Scaffold(
16       appBar: AppBar(title: const Text("Slider")),
17       body: Center(
18         child: Slider(
19           value: _value.toDouble(),
20           min: 1.0,
21           max: 20.0,
22           divisions: 10,
23           activeColor: Colors.green,
24           inactiveColor: Colors.orange,
25           label: 'Set volume value',
26           onChanged: (double newValue) {
27             setState(() {
28               _value = newValue.round();
29             });
30           semanticFormatterCallback: (double newValue) {
31             return '${newValue.round()} dollars';
32           },
33         ),
34       ),
35     );
36   }
}

```

Hình 2. 65. Một ví dụ minh họa cho Widget Slider

2.2.23. Widget Switch

Switch là một phần tử giao diện người dùng hai trạng thái được sử dụng để chuyển đổi giữa các trạng thái **BẬT (Đã kiểm tra)** hoặc **TẮT (Bỏ chọn)**.

Ưu điểm: Widget Switch trong Flutter là một cách thuận tiện để người dùng chuyển đổi giữa hai trạng thái - Bật hoặc Tắt. Nó cung cấp một cách đơn giản và trực quan để thực hiện các tùy chọn hoặc cài đặt trong ứng dụng.

Nhược điểm: Switch có thể không phù hợp cho các tùy chọn có nhiều trạng thái hơn hai, vì nó chỉ hỗ trợ hai trạng thái. Ngoài ra, trên một số thiết bị hoặc giao diện người dùng, Switch có thể không phù hợp về mặt thẩm mỹ hoặc không thích hợp cho không gian màn hình hạn chế.

Code mẫu để tạo một Switch trên màn hình

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          backgroundColor: Colors.blue,
          title: Text("Custom Switch Example"),
        ),
        body: Center(
          child: SwitchScreen(),
        ),
      ),
    );
  }
}

class SwitchScreen extends StatefulWidget {
```

```

@Override
SwitchClass createState() => new SwitchClass();

}

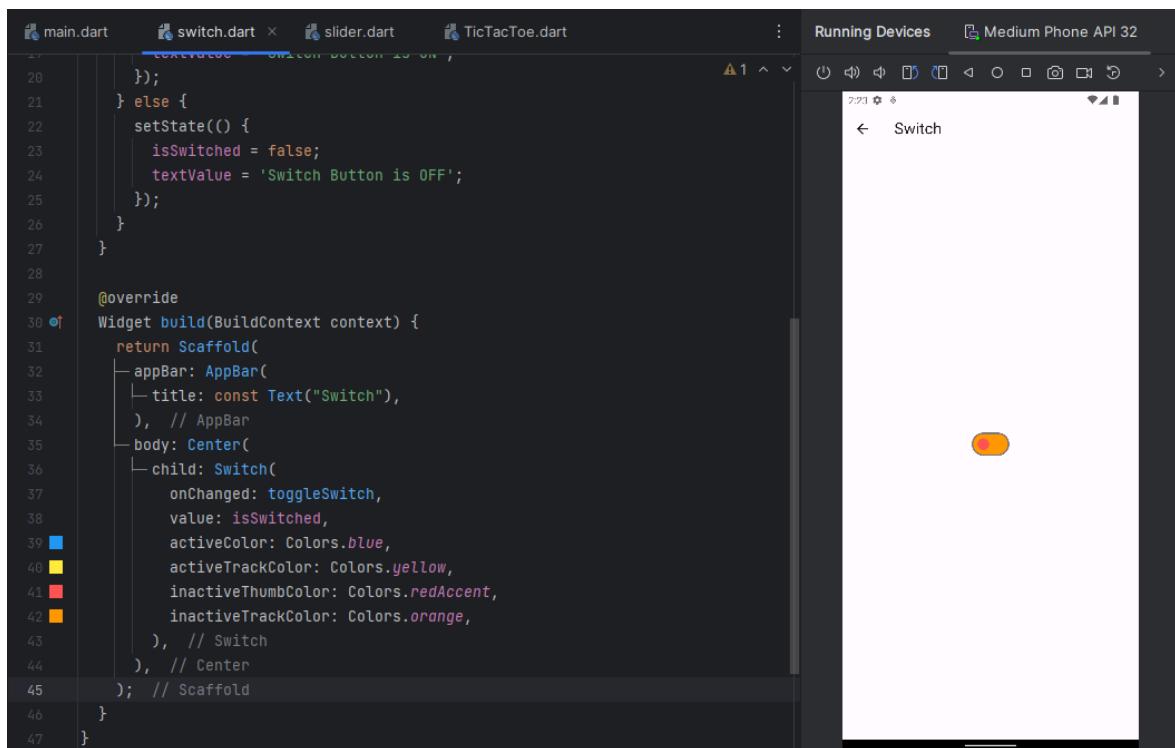
class SwitchClass extends State {
  bool isSwitched = false;
  @override
  Widget build(BuildContext context) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children:<Widget>[
        CustomSwitch(
          value: isSwitched,
          activeColor: Colors.blue,
          onChanged: (value) {
            print("VALUE : $value");
            setState(() {
              isSwitched = value;
            });
          },
        ),
        SizedBox(height: 15.0,),
        Text('Value : $isSwitched', style: TextStyle(color: Colors.red,
          fontSize: 25.0,))
      ]);
  }
}

```

Các thuộc tính có trong Switch

- **onChanged:** Được gọi bất cứ khi nào người dùng chạm vào công tắc.
- **value:** Chứa giá trị Boolean true hoặc false để kiểm soát xem chức năng của công tắc đang BẬT hay TẮT.

- **activeColor:** Được sử dụng để chỉ định màu của công tắc bóng tròn khi nó BẬT.
- **activeTrackColor:** Chỉ định màu thanh chuyển hướng.
- **inactiveThumbColor:** Được sử dụng để chỉ định màu của công tắc bóng tròn khi nó TẮT.
- **inactiveTrackColor:** Chỉ định màu thanh công tắc khi nó TẮT.
- **dragStartBehavior:** Nó đã xử lý hành vi bắt đầu kéo. Nếu chúng ta đặt nó là DragStartBehavior.start, thì thao tác kéo sẽ di chuyển công tắc từ bật sang tắt.



Hình 2. 66. Một ví dụ minh họa cho Widget Switch

2.2.24. Widget Charts

Charts trong Flutter được dùng để **biểu diễn dữ liệu theo cách đồ họa** cho phép người dùng hiểu chúng một cách đơn giản.

Ưu điểm: Widget Charts giúp hiển thị dữ liệu một cách trực quan, dễ hiểu. Nó cung cấp một loạt các loại biểu đồ như đường, cột, tròn, v.v., để phù hợp với nhu cầu

hiển thị dữ liệu khác nhau. Widget này hỗ trợ tùy chỉnh cao, cho phép bạn điều chỉnh các yếu tố như màu sắc, kích thước, và hiệu ứng của biểu đồ.

Nhược điểm: Một số biểu đồ có thể phức tạp và khó hiểu nếu không được trình bày một cách rõ ràng. Điều này có thể làm giảm khả năng tiếp cận của người dùng đối với dữ liệu. Ngoài ra, việc tùy chỉnh cao có thể đòi hỏi kiến thức kỹ thuật và thời gian..

Flutter chủ yếu hỗ trợ ba loại biểu đồ và mỗi biểu đồ đi kèm với một số tùy chọn câu hình. Sau đây là biểu đồ được sử dụng trong ứng dụng Flutter:

LineChart:

- **LineChartData:** Lớp này chứa dữ liệu và cấu hình cho biểu đồ dạng đường.
 - **lineBarsData:** Danh sách các dữ liệu biểu đồ, mỗi dữ liệu biểu đồ là một đường trên biểu đồ.
 - **titlesData:** Thông tin về các tiêu đề của biểu đồ như tiêu đề trực x, trực y, và các chú thích.
 - **borderData:** Dữ liệu liên quan đến đường biên của biểu đồ như viền và màu sắc.
- **LineChartBarData:** Dữ liệu của một đường trên biểu đồ dạng đường.
 - **spots:** Danh sách các điểm trên đường biểu đồ, mỗi điểm có tọa độ x và y.
 - **isCurved:** Xác định xem đường có được vẽ cong hay không.
 - **colors:** Màu sắc của đường biểu đồ.
 - **barWidth:** Độ dày của đường biểu đồ.

BarChart:

- **BarChartData:** Lớp này chứa dữ liệu và cấu hình cho biểu đồ cột.
 - **barGroups:** Danh sách các nhóm cột trên biểu đồ.
 - **titlesData:** Thông tin về các tiêu đề của biểu đồ như tiêu đề trực x, trực y, và các chú thích.

- **borderData:** Dữ liệu liên quan đến đường biên của biểu đồ như viền và màu sắc.
- **BarChartGroupData:** Dữ liệu của một nhóm cột trên biểu đồ cột.
 - **x:** Vị trí trên trục x của nhóm cột.
 - **barRods:** Danh sách các cột trong nhóm.
 - **showingTooltipIndicators:** Danh sách các chỉ số của các cột hiển thị tooltip khi được chạm vào.

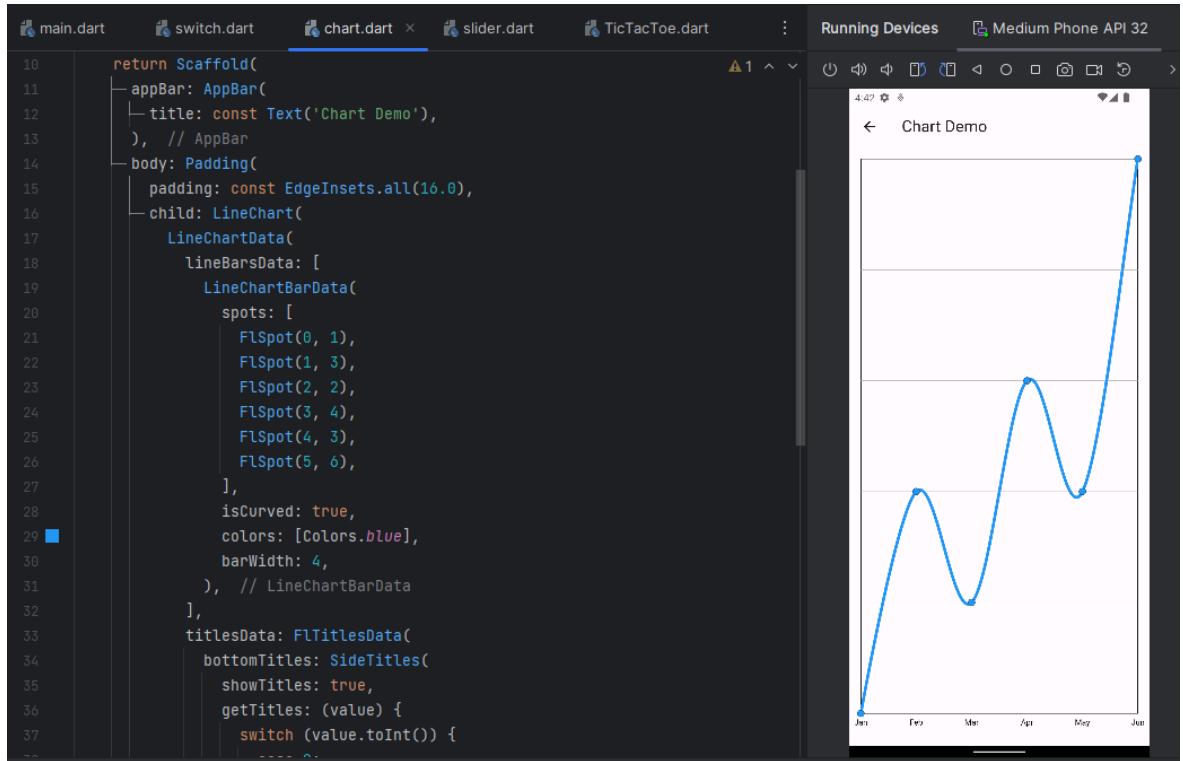
PieChart:

- **PieChartData:** Lớp này chứa dữ liệu và cấu hình cho biểu đồ hình tròn.
 - **sections:** Danh sách các phần của biểu đồ hình tròn.
 - **centerSpaceRadius:** Bán kính của phần trống ở giữa biểu đồ.
 - **borderData:** Dữ liệu liên quan đến đường biên của biểu đồ như viền và màu sắc.
- **PieChartSectionData:** Dữ liệu của một phần trên biểu đồ hình tròn.
 - **value:** Giá trị của phần trên biểu đồ.
 - **color:** Màu sắc của phần trên biểu đồ.
 - **radius:** Bán kính của phần trên biểu đồ.

Demo chương trình yêu cầu cài đặt thư viện **fl_chart**:

```
dependencies:
  font_awesome_flutter: '^>= 4.7.0'
  flutter:
    sdk: flutter
  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.6
  fl_chart: 0.35.0
```

Hình 2. 67. Cài đặt thư viện *fl_chart* để sử dụng Widget Chart



Hình 2. 68. Demo chương trình hiển thị một biểu đồ bằng cách sử dụng Widget Chart

2.2.25. Widget Bottom Navigation Bar

Bottom Navigation Bar trong Flutter có thể chứa nhiều mục như nhãn văn bản, biểu tượng hoặc cả hai. Dùng để điều hướng ứng dụng.

Ưu điểm: Widget Bottom Navigation Bar cung cấp một cách tiện lợi để điều hướng giữa các màn hình chính trong ứng dụng. Nó cho phép người dùng truy cập nhanh chóng đến các phần chính của ứng dụng mà không cần phải thực hiện nhiều thao tác. Bottom Navigation Bar thường được sử dụng trong các ứng dụng di động để tạo ra một trải nghiệm điều hướng dễ sử dụng và trực quan.

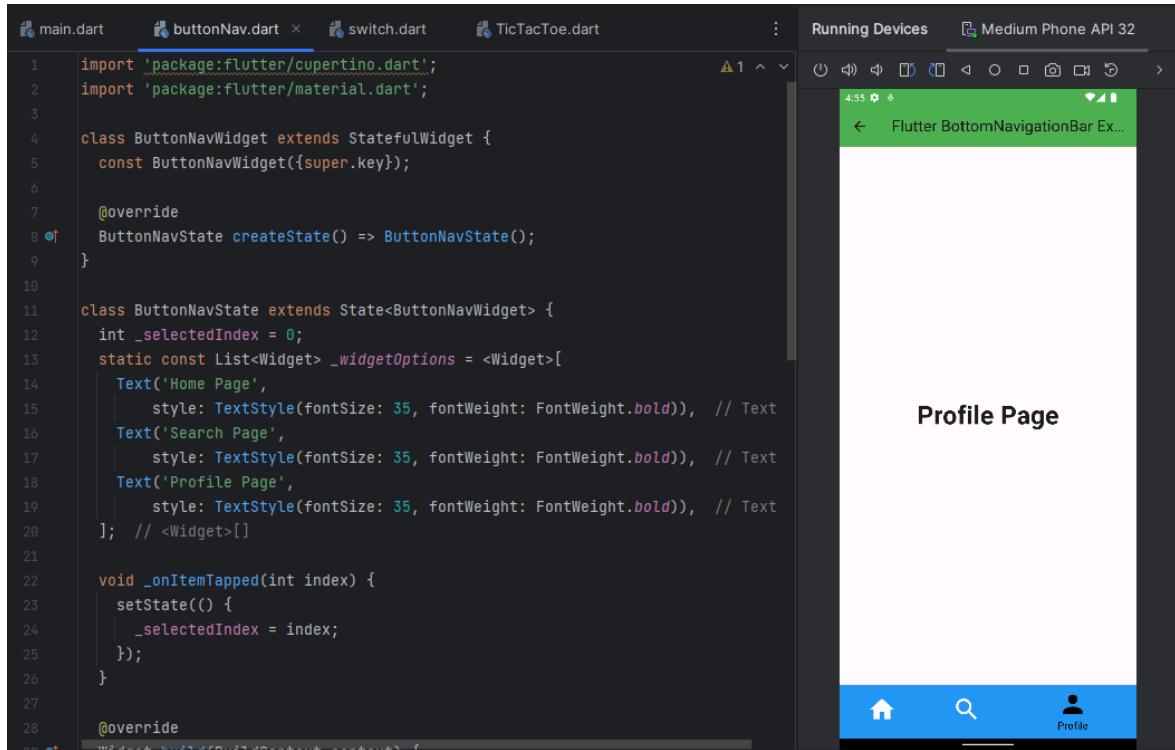
Nhược điểm: Bottom Navigation Bar không phù hợp cho các ứng dụng có số lượng màn hình chính lớn hoặc cần tích hợp nhiều tính năng phụ. Nếu không được thiết kế cẩn thận, nó có thể gây ra sự cồng kềnh hoặc khó sử dụng cho người dùng.

Các thuộc tính có trong Bottom Navigation Bar

- **items:** Nó xác định danh sách để hiển thị trong thanh điều hướng dưới cùng. Nó sử dụng đối số BottomNavigationBarItem có chứa các thuộc tính sup được đưa ra bên dưới:

```
const BottomNavigationBarItem({
  @required this.icon,
  this.title,
  Widget activeIcon,
  this.backgroundColor,
})
```

- **currentIndex:** Nó xác định mục thanh điều hướng dưới cùng đang hoạt động hiện tại trên màn hình.
- **onTap:** Nó được gọi khi chúng ta chạm vào một trong các mục trên màn hình.
- **iconSize:** Nó được sử dụng để chỉ định kích thước của tất cả các biểu tượng mục điều hướng phía dưới.
- **fixedColor:** Nó được sử dụng để đặt màu của mục đã chọn. Nếu chúng ta chưa đặt màu cho biểu tượng hoặc tiêu đề, nó sẽ được hiển thị.
- **type:** Nó xác định bố cục và hành vi của thanh điều hướng dưới cùng. Nó hoạt động theo hai cách khác nhau, đó là: **fixed** và shifting. Nếu nó là null, nó sẽ sử dụng fixed. Nếu không, nó sẽ sử dụng tính năng chuyển đổi nơi chúng ta có thể xem hoạt ảnh khi chúng ta nhấp vào một nút.



Hình 2. 69. Một ví dụ minh họa cho Widget BottomNavigationBar

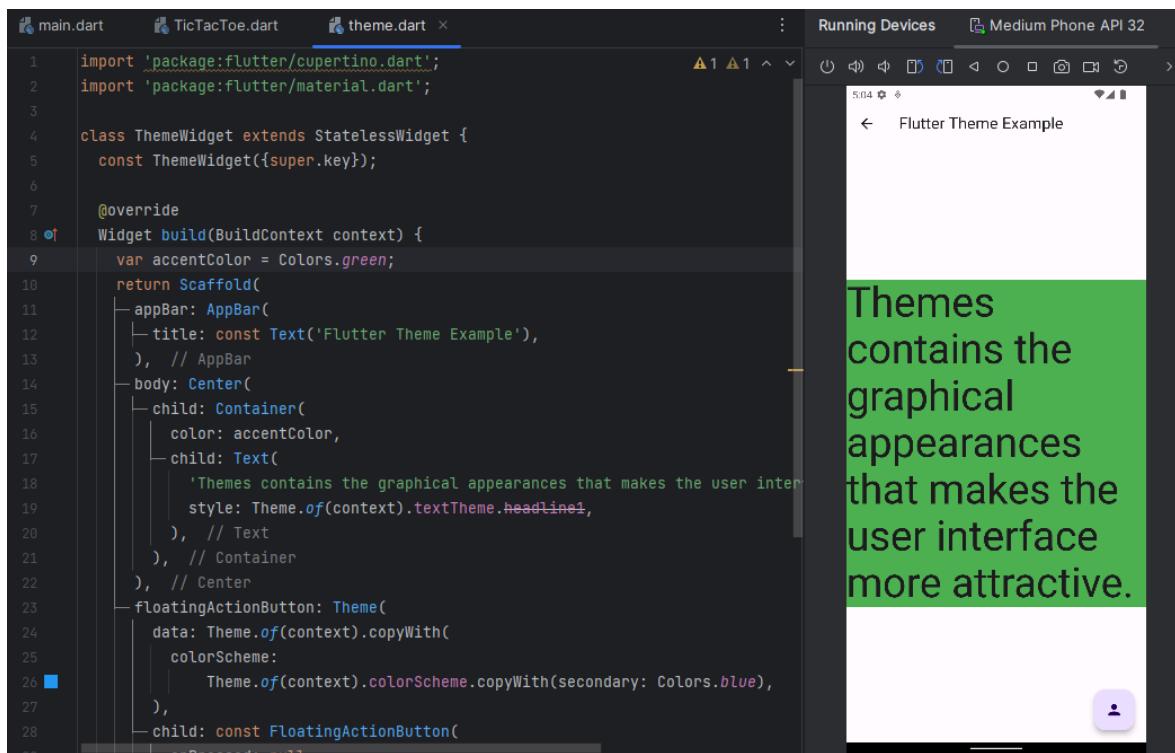
2.2.26. Widget Theme

Theme là các gói đặt trước chứa các **giao diện đồ họa trên** trang web hoặc màn hình ứng dụng dành cho thiết bị di động của chúng ta.

Ưu điểm: Widget Theme cho phép bạn tùy chỉnh giao diện của ứng dụng một cách dễ dàng và linh hoạt. Bạn có thể định nghĩa các thuộc tính giao diện chung một lần và áp dụng chúng cho nhiều widget trong toàn bộ ứng dụng. Điều này giúp giảm thiểu việc lặp lại code và giữ cho code của bạn dễ bảo trì.

Nhược điểm: Mặc dù Widget Theme là một công cụ mạnh mẽ, nhưng đôi khi việc tùy chỉnh giao diện có thể phức tạp và đòi hỏi hiểu biết sâu rộng về Flutter và cách làm việc với chủ đề.

Cấu trúc: Widget Theme trong Flutter được sử dụng bên trong widget tree để áp dụng các thuộc tính giao diện nhất định cho một phần hoặc toàn bộ ứng dụng. Một Widget Theme bao gồm một hoặc nhiều thuộc tính giao diện được định nghĩa bởi ThemeData.



Hình 2. 70. Một ví dụ minh họa cho Widget Theme

2.2.27. Widget Table

Widget Table trong Flutter là một cách để hiển thị dữ liệu dưới dạng bảng, với hàng và cột.

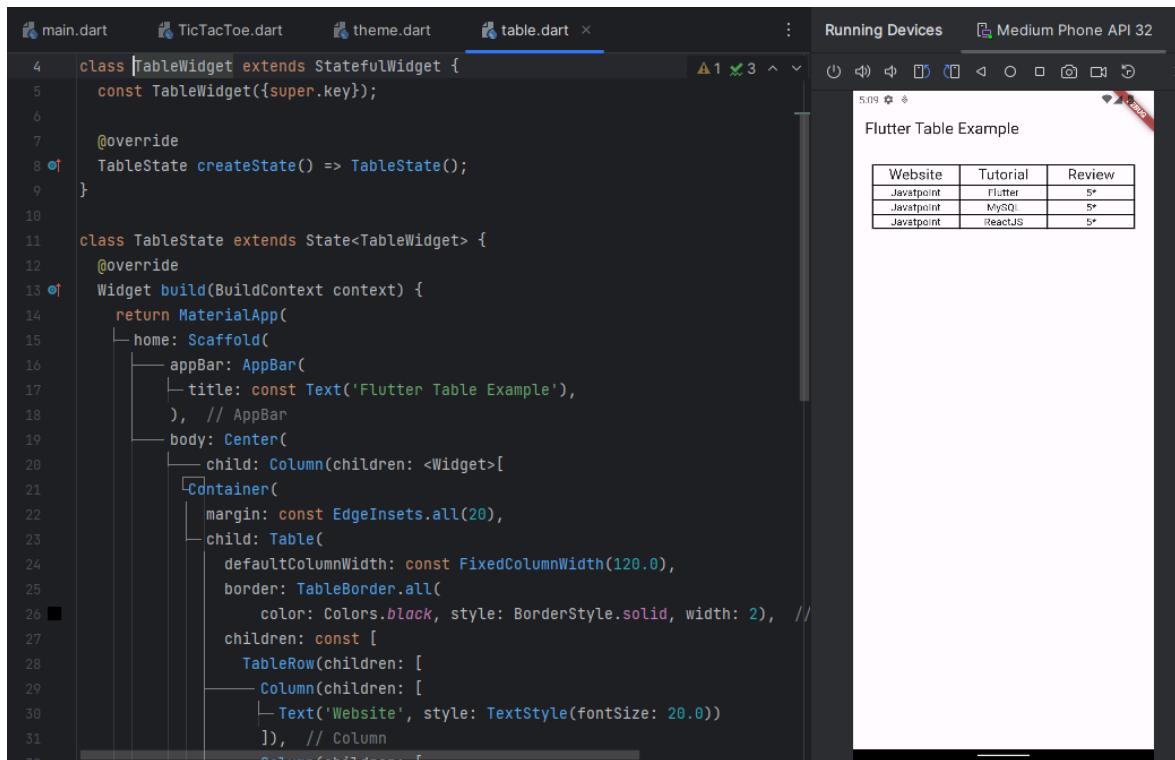
Ưu điểm: Widget Table cung cấp một cách tiện lợi để hiển thị dữ liệu dưới dạng bảng trong ứng dụng Flutter. Nó cho phép bạn sắp xếp và hiển thị dữ liệu một cách cấu trúc, giúp người dùng dễ dàng đọc và tìm kiếm thông tin.

Nhược điểm: Widget Table có thể không phù hợp cho việc hiển thị dữ liệu lớn hoặc phức tạp, vì nó có thể gây ra sự cồng kềnh và tốn nhiều không gian trên màn hình.

Widget Table trong Flutter có cấu trúc đơn giản, bao gồm một danh sách các TableRow, mỗi TableRow chứa một danh sách các TableCell. Mỗi TableCell có thể chứa một widget, thường là Text hoặc một widget khác để hiển thị dữ liệu. Ví dụ đoạn code như sau:

TableRow(children: [

```
TableCell(child: Text('javatpoint')),
TableCell(
    child: Text('Flutter'),
),
TableCell(child: Text('Android')),
TableCell(child: Text('MySQL')),
],
```



Hình 2. 71. Một ví dụ minh họa cho Widget Table

2.2.28. Widget Animation

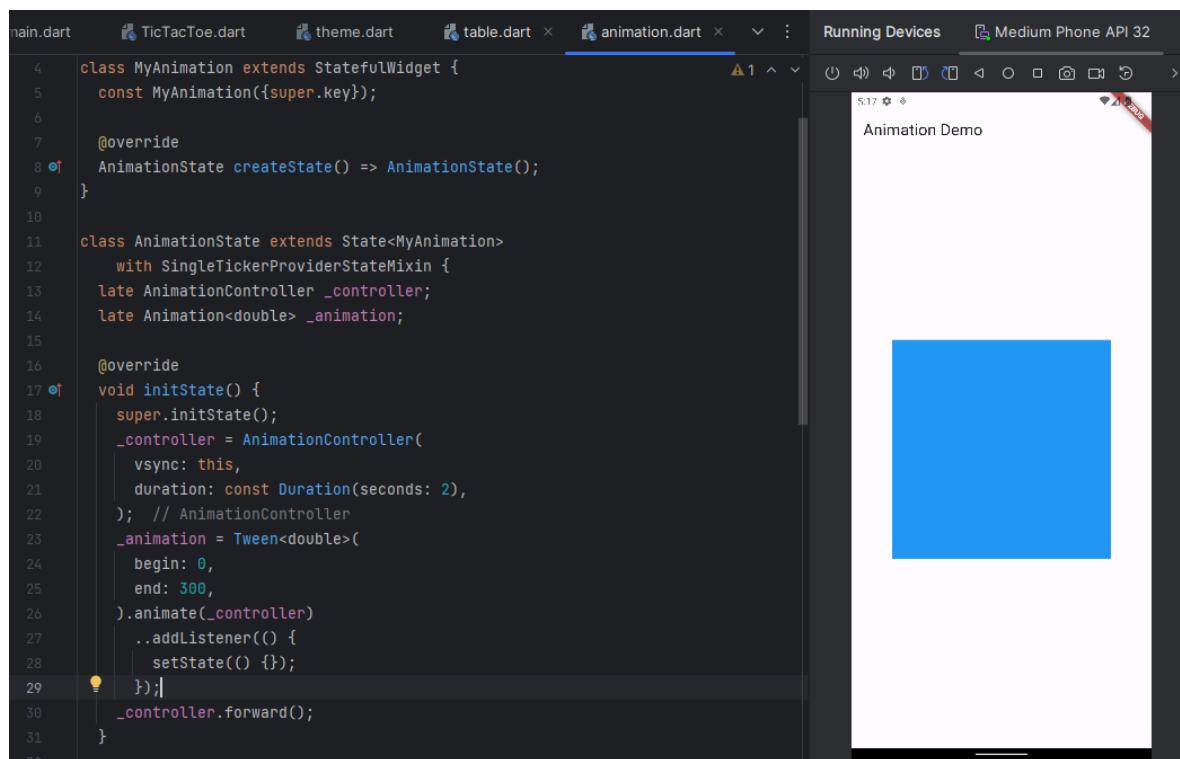
Animation trong Flutter là một cách mạnh mẽ để tạo ra các hiệu ứng chuyển động và thay đổi trong giao diện người dùng của ứng dụng.

Ưu điểm: Animation trong Flutter cho phép bạn tạo ra các hiệu ứng chuyển động mượt mà và hấp dẫn trong ứng dụng của bạn. Nó giúp cải thiện trải nghiệm người dùng và tạo ra giao diện động hơn, từ việc di chuyển đối tượng đến việc thay đổi màu sắc và kích thước của chúng.

Nhược điểm: Việc sử dụng Animation trong Flutter có thể phức tạp và đòi hỏi hiểu biết về cách làm việc với các loại AnimationController, Tween, và AnimatedBuilder. Nếu không được xử lý cẩn thận, việc sử dụng animation có thể dẫn đến code khó hiểu và khó bảo trì.

Cấu trúc: Animation trong Flutter thường được xây dựng từ ba phần chính:

- AnimationController: Quản lý quá trình chạy của animation, bao gồm việc điều khiển thời gian và tốc độ của animation.
- Tween: Xác định các giá trị đầu vào và đầu ra của animation, cho phép bạn xác định các thay đổi trong animation.
- AnimatedBuilder hoặc các widget tương tự: Sử dụng để tạo ra các hiệu ứng dựa trên animation, bằng cách cập nhật giao diện của widget theo giá trị animation được cung cấp.



```

main.dart      TicTacToe.dart    theme.dart    table.dart    animation.dart    Running Devices    Medium Phone API 32
4   class MyAnimation extends StatefulWidget {
5     const MyAnimation({super.key});
6
7     @override
8     AnimationState createState() => AnimationState();
9   }
10
11   class AnimationState extends State<MyAnimation>
12     with SingleTickerProviderStateMixin {
13       late AnimationController _controller;
14       late Animation<double> _animation;
15
16     @override
17     void initState() {
18       super.initState();
19       _controller = AnimationController(
20         vsync: this,
21         duration: const Duration(seconds: 2),
22       ); // AnimationController
23       _animation = Tween<double>(
24         begin: 0,
25         end: 300,
26       ).animate(_controller)
27       ..addListener(() {
28         setState(() {});
29       });
30     }
31   }

```

Hình 2. 72. Một ví dụ minh họa cho Widget Animation

CHƯƠNG 3. XÂY DỰNG MỘT SỐ CHƯƠNG TRÌNH FLUTTER CƠ BẢN

Với mục tiêu là áp dụng lý thuyết vào thực hành, nhóm chúng em xin trình bày một số chương trình cơ bản được xây dựng bằng framework Flutter với ngôn ngữ lập trình Dart, một số chương trình này là các ứng dụng cơ bản như **Máy tính (Calculator)**, **Ghi chú (Todo App)**, **Trò chơi TicTacToe (Game)**, **Tính toán BMI (BMI Calculator)**.

Lưu ý: Các chương trình trên chỉ có phần Front-End là chủ đạo cho các kiến thức đã trình bày ở các chương trước. Mục đích là áp dụng các kiến thức về các Widget đã được học nhằm xây dựng các giao diện người dùng cơ bản.

3.1. Ứng dụng Máy Tính (Calculator App)

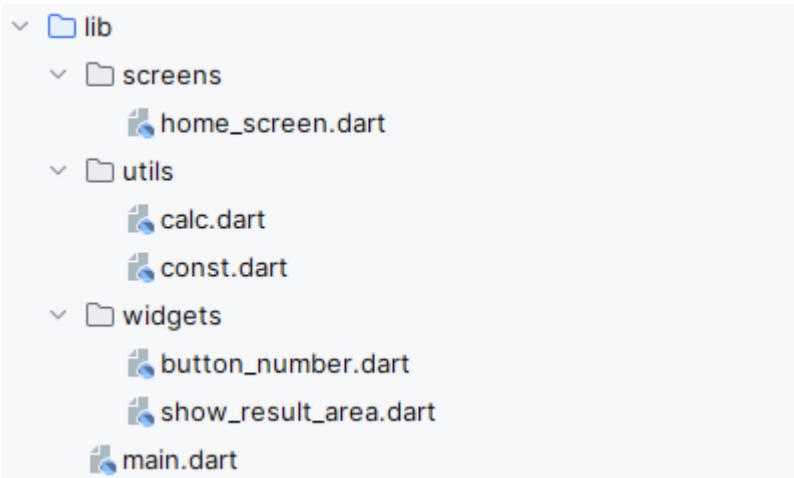
3.1.1. Giới thiệu

Máy tính (Calculator) - Một trong những ứng dụng cơ bản giúp cho những lập trình viên có thể làm quen với cấu trúc giao diện khi mới bắt đầu xây dựng những dự án mới, flutter cũng không phải ngoại lệ. Sau khi xây dựng xong chương trình này những gì mà ta đạt được là cách cấu trúc thư mục, tổ chức code, sử dụng chức năng, thư viện, framework một cách hợp lý.

3.1.2. Cấu trúc chương trình

3.1.2.1. Phân loại thư mục

Do là một chương trình đơn giản và chỉ thao tác trên một màn hình duy nhất. Cho nên cấu trúc thư mục của chương trình cũng đơn giản và dễ hiểu, cây thư mục của chương trình được biểu thị như sau:



Hình 3. 1. Cây thư mục chương trình Máy tính (Calculator App)

Trong đây, ta có thể thấy cây thư mục được chia ra làm 3 folder chính được mô tả như sau:

- **screens:** Là thư mục chứa các màn hình của chương trình. Ở đây ta chỉ thực hiện trên một màn hình duy nhất nên chỉ có một màn hình *home_screen.dart*.
 - **home_screen.dart:** File này chứa code của màn hình chính của app.
- **utils:** Là thư mục chứa các hàm và lớp tiện ích của ứng dụng. Có nghĩa là mỗi lần cần sử dụng các hàm build-in sẵn ta chỉ cần gọi chúng lại thay vì phải code lại như từ đầu gây mất thời gian và quá nhiều code trong chương trình.
 - **calc.dart:** File này chứa các hàm tính toán.
 - **const.dart:** File này chứa các hằng số của ứng dụng và một số hàm getter nhanh
- **widgets:** Thư mục này chứa các widget của ứng dụng. Mỗi widget được biểu hiện thành 1 File.
 - **button_number.dart:** File này chứa code hiển thị các nút bấm số của máy tính.
 - **show_result_area.dart:** File này chứa code hiển thị khu vực kết quả sau quá trình nhập và tính toán

Việc phân chia và tách code ra thành các thư mục sẽ giúp cho chương trình dễ quản lý code cũng như góp phần tái sử dụng lại code cho những phần code sau.

3.1.2.2. Các hàm mặc định

❖ Hàm hỗ trợ tính toán

Trong một chương trình tính toán, việc không thể thiếu đó chính là các hàm chức năng (function) giúp chúng ta hỗ trợ tính toán. Trong chương trình của mình, nhóm chúng em đã sử dụng thư viện ***math_expression*** để hỗ trợ cho việc tính toán, theo mô tả trên tài liệu thư viện có input như sau:

Input	Một chuỗi biểu thức gồm số học và kí hiệu toán học VD: (1+2)-3, 3*122+23
Output	Kết quả thực hiện

Sau đây là đoạn code để thực hiện công việc trên:

```
static String calculate(String input) {
    try {
        var exp = Parser().parse(input);
        var evaluation = exp.evaluate(EvaluationType.REAL, ContextModel());
        return evaluation.toString();
    } catch (e) {
        return "Error";
    }
}
```

Hàm được đặt sau *static* để có thể được gọi ở bất kì class nào.

Hàm được đặt trong class *MyHelper* và nằm trong File **calc.dart**.

❖ Các hàm tiện ích và hằng số

Việc chuẩn bị trước các biểu thức và những thứ cần thiết cho chương trình giúp cho chúng ta có thể tái sử dụng lại code nhiều hơn và cũng như có thể code thoái mái hơn mà không phải khai báo lại phần code đã làm quá nhiều lần

Một số đoạn code của chương trình mà nhóm đã chuẩn bị như sau:

1. Danh sách các nút bấm

Đoạn code chứa danh sách các nút bấm bàn phím:

```
/// Các nút bấm của máy tính
static List<String> buttonList =
[ "AC", "(", ")", "/", "7", "8", "9", "*", "4", "5", "6", "+", "1", "2", "3", "-", "C", "0",
".", "=" ];
```

2. Hàm getter lấy màu sắc

Đoạn code giúp ta lấy được màu sắc được chỉ định:

```
/// Lấy màu sắc
static Color getColor(String text) {
    if (text == "/" ||
        text == "*" ||
        text == "+" ||
        text == "-" ||
        text == "C" ||
        text == "(" ||
        text == ")") {
        return Colors.redAccent;
    }
    if (text == "=" || text == "AC") {
        return Colors.white;
    }
    return Colors.indigo;
}

/// Lấy màu sắc của Button được chỉ định
static Color? getButtonColor(String text) {
    if (text == "AC") {
        return Colors.redAccent;
    }
    if (text == "=") {
```

```

    return const Color.fromARGB(255, 104, 204, 159);
}

return null;
}

```

3. Hàm lấy chiều dài và chiều rộng của màn hình hiện tại

Đoạn code giúp lấy chiều dài và chiều rộng của màn hình:

```

/// Lấy chiều dài màn hình
static double screenHeight(BuildContext context) {
    return MediaQuery.of(context).size.height;
}

/// Lấy chiều ngang màn hình
static double screenWidth(BuildContext context) {
    return MediaQuery.of(context).size.width;
}

```

3.1.2.3. Màn hình hiển thị

Trong chương trình, chỉ có duy nhất một màn hình chính gồm:

1. Khu vực hiển thị nhập biểu thức tính toán và hiển thị kết quả.
2. Khu vực hiển thị bàn phím số và dấu tính toán

Toàn bộ đoạn code cho màn hình chính như sau, đoạn code này nằm trong File **home_screen.dart**:

```

import 'package:flutter/material.dart';

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

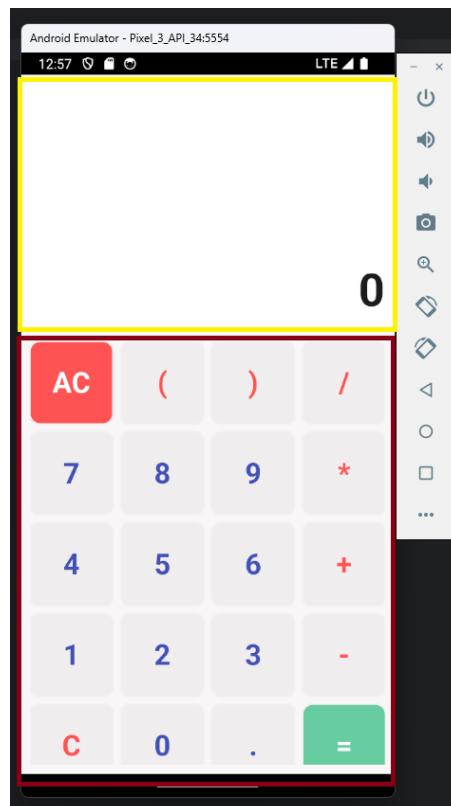
    @override
    State<HomeScreen> createState() => _HomeScreenState();
}

```

```
class _HomeScreenState extends State<HomeScreen> {  
    String userInput = "";  
    String result = "0";  
  
    @override  
    Widget build(BuildContext context) {  
        return SafeArea(  
            child: Scaffold(  
                body: Column(  
                    children: [  
                        /// Khu vực hiển thị kết quả tính toán  
                        ShowResultArea(  
                            userInput: userInput,  
                            result: result,  
                        ),  
  
                        /// Khu vực hiển thị các nút bấm  
                        Expanded(  
                            child: Container(  
                                padding: const EdgeInsets.all(10),  
                                color: const Color.fromARGB(66, 233, 232, 232),  
                                child: GridView.builder(  
                                    itemCount: MyConst.buttonList.length,  
                                    gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(  
                                        crossAxisCount: 4,  
                                        crossAxisSpacing: 10,  
                                        mainAxisSpacing: 10,  
                                    ),  
                                    itemBuilder: (context, index) {  
                                        return ButtonNumber(  
                                            value: buttonList[index],  
                                        );  
                                    },  
                                ),  
                            ),  
                        ),  
                    ],  
                ),  
            ),  
        );  
    }  
}
```

```
text: MyConst.buttonList[index],  
        handleBtnPress: handleBtnPress,  
    );  
},  
(  
(  
),  
],  
(  
),  
);  
}  
  
// Xử lý sự kiện khi ấn vào nút bấm  
void handleBtnPress(String text) {  
    setState(() {  
        if (text == "AC") {  
            userInput = "";  
            result = "0";  
            return;  
        }  
        if (text == "C") {  
            if (userInput.isNotEmpty) {  
                userInput = userInput.substring(0, userInput.length - 1);  
                return;  
            }  
            return;  
        }  
        if (text == "=") {  
            if (userInput.isNotEmpty) {  
                result = MyHelper.calculate(userInput);  
            }  
        }  
    });  
}
```

```
userInput = result;  
if (userInput.endsWith(".0")) {  
    userInput = userInput.replaceAll(".0", "");  
    return;  
}  
if (result.endsWith(".0")) {  
    result = result.replaceAll(".0", "");  
    return;  
}  
return;  
}  
userInput = userInput + text;  
});  
}  
}
```



Hình 3. 2. Giao diện chương trình Máy tính được chia làm 2 phần khu vực hiển thị

Trong đó, phần **khung màu vàng (ô vuông bên trên)** là phần hiển thị nhập biểu thức tính toán và hiển thị kết quả, phần **khung màu đỏ (ô vuông bên dưới)** là phần hiển thị hiển thị bàn phím số và dấu tính toán.

Trong đoạn code màn hình chính của mình nhằm thuận tiện hơn và tối ưu hơn nhóm chúng em đã tách các khu vực đã kể trên thành các widget riêng cho để xử lý như sau:

❖ Khu vực nhập biểu thức và hiển thị kết quả (Khung 1. Màu vàng)

Trong chương trình của nhóm thì, khu vực đó được đặt tên là **ShowResultArea** và nó nhận vào 2 giá trị gồm **userInput** (biểu thức) và **result** (kết quả). Mỗi lần ấn nút thì các giá trị trên sẽ tự động cập nhật thông qua *setState()* và hiển thị lên màn hình.

```
import 'package:calcula_tor/utils/const.dart';
import 'package:flutter/material.dart';

class ShowResultArea extends StatelessWidget {
  const ShowResultArea({
    super.key,
    required this.userInput,
    required this.result,
  });

  final String userInput;
  final String result;

  @override
  Widget build(BuildContext context) {
    return SizedBox(
      height: MyConst.screenHeight(context) / 2.80,
      child: Container(
        color: Colors.white,
```

```
child: Column(  
    mainAxisAlignment: MainAxisAlignment.end,  
    children: [  
        Container(  
            padding: const EdgeInsets.all(20),  
            alignment: Alignment.centerRight,  
            child: Text(  
                userInput,  
                style: const TextStyle(  
                    fontSize: 32,  
                ),  
            ),  
        ),  
        Container(  
            padding: const EdgeInsets.all(10),  
            alignment: Alignment.centerRight,  
            child: Text(  
                result,  
                style: const TextStyle(  
                    fontSize: 48,  
                    fontWeight: FontWeight.bold,  
                ),  
            ),  
        ),  
    ],  
,  
    );  
}  
}
```

❖ Widget nút bấm (Khung 2. Màu đỏ)

Là widget tượng trưng cho nút bấm trong chương trình:



Hình 3. 3. Lấy ví dụ đây là một nút bấm trong khu vực bàn phím

Đoạn code để tạo ra nút bấm trên như sau:

```
import 'package:calcula_tor/utils/const.dart';
import 'package:flutter/material.dart';

class ButtonNumber extends StatelessWidget {
  const ButtonNumber({
    super.key,
    required this.text,
    required this.handleBtnPress,
  });

  final String text;
  final Function handleBtnPress;

  @override
  Widget build(BuildContext context) {
    return InkWell(
      onTap: () => handleBtnPress(text),
      child: Container(
        decoration: BoxDecoration(
          color: MyConst.getButtonColor(text),
          borderRadius: BorderRadius.circular(10),
          boxShadow: [

```

```

BoxShadow(
    color: Colors.grey.withOpacity(0.1),
    blurRadius: 1,
    spreadRadius: 1,
)
],
),
child: Center(
    child: Text(
        text,
        style: TextStyle(
            color: MyConst.getColor(text),
            fontSize: 30,
            fontWeight: FontWeight.bold,
        ),
    ),
),
),
),
),
),
);
}
}

```

Và để tạo ra một giao diện bàn phím hoàn chỉnh như trên **hình 3.2** ta cần kết hợp nó với một Widget chuyên tạo giao diện lưới như **GridView**:

```

GridView.builder(
    itemCount: MyConst.buttonList.length,
    gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
        crossAxisCount: 4,
        crossAxisSpacing: 10,
        mainAxisSpacing: 10,
    ),
)

```

```

itemBuilder: (context, index) {
    return ButtonNumber(
        text: MyConst.buttonList[index],
        handleBtnPress: handleBtnPress,
    );
},
)

```

Chỉ định cách bố trí các item trong GridView:

- **crossAxisCount:** 4 chia GridView thành 4 cột.
- **crossAxisSpacing:** 10 tạo khoảng cách 10 pixel giữa các cột.
- **mainAxisSpacing:** 10 tạo khoảng cách 10 pixel giữa các hàng.

Đoạn code “*itemBuilder: (context, index) { ... }*” này gọi để xây dựng từng item trong GridView:

- **context:** Cung cấp context hiện tại của widget.
- **index:** Vị trí của item đang được xây dựng.

Return trả về một widget *ButtonNumber* cho **mỗi item**

handleBtnPress là một callback function gọi tới hàm **onTap()** trong mỗi đối tượng nút bấm và hàm này có nhiệm vụ bắt sự kiện khi người dùng ấn vào nút bấm và ghi nhận sự kiện đó lại và hiển thị lên màn hình

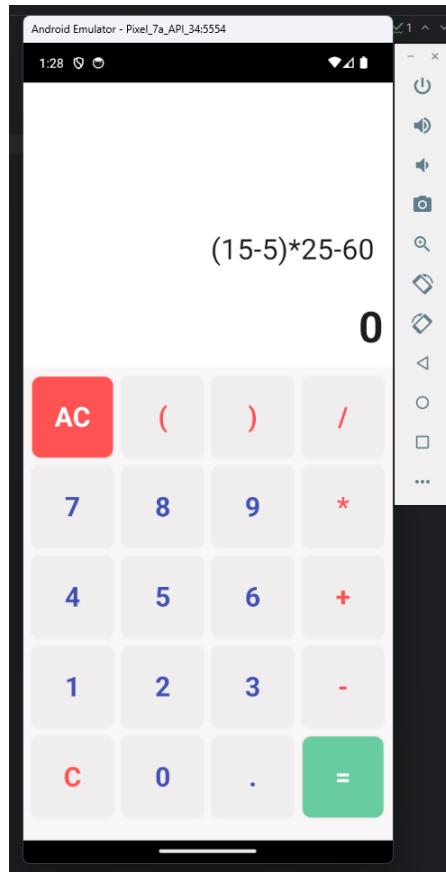
Tóm lại:

1. Đoạn code tạo ra giao diện ở khung 2 dùng một **GridView** gồm các nút số, được sắp xếp thành 4 cột với khoảng cách 10 pixel giữa các cột và hàng.
2. Nội dung **text** của các nút được lấy từ thuộc tính **buttonList** ở lớp **MyConst**.

3.1.3. Demo chương trình

1. Nhập biểu thức tính toán
2. Án = để thực hiện việc tính toán.

3. Ngoài ra ta cũng có thể ấn **AC** để xoá toàn bộ biểu thức hiện tại hoặc **C** để xoá đi một kí tự vừa nhập vào biểu thức.



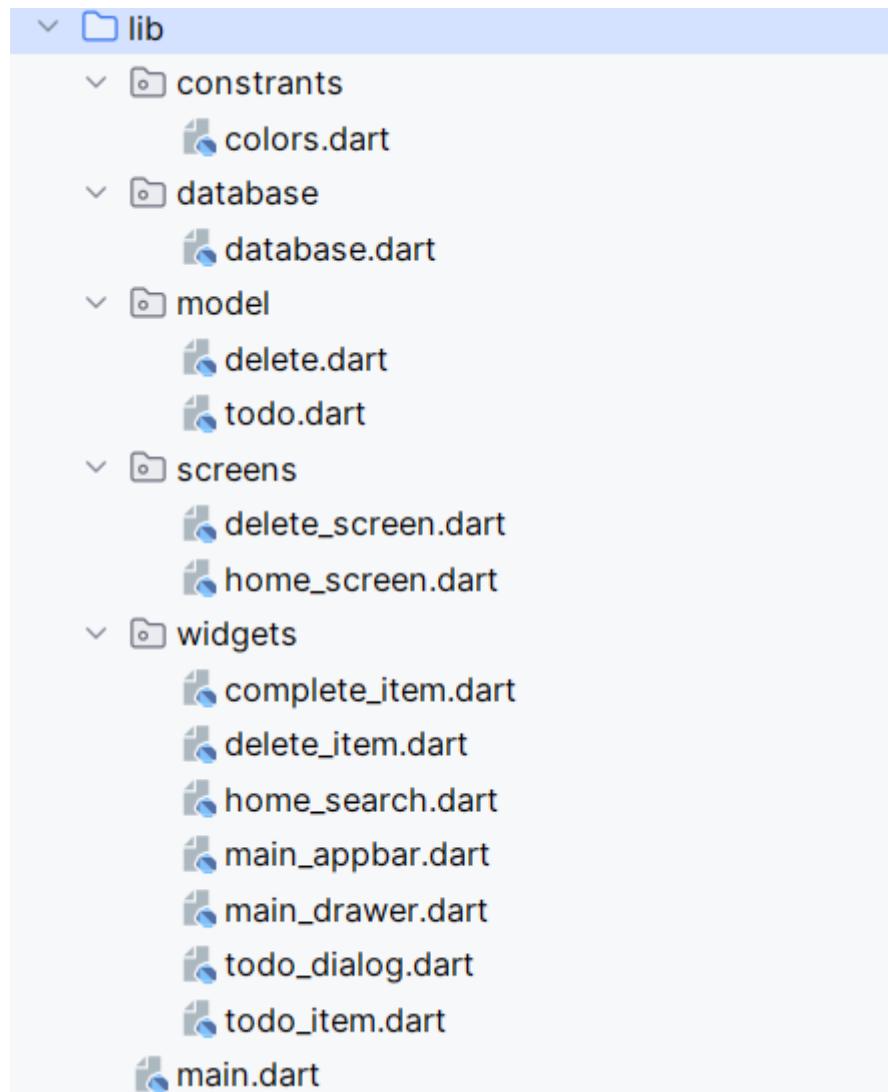
Hình 3. 4. Demo chương trình máy tính

3.2. Ứng dụng Ghi Chú (Todo App)

3.2.1. Giới thiệu

Todo là một ứng dụng ghi chú cở bản dành cho mục đích ghi lại các công việc thường làm trong một khoảng thời gian hoặc một tiến trình cụ thể nhưng lại có nhiều công việc cần làm. Có một và tính năng độc đáo như là chỉnh sửa nội dung công việc cần làm hay khôi phục lạ việc mà mình đã lỡ tay xoá nhầm

3.2.2. Cấu trúc chương trình



Hình 3. 5. Cấu trúc thư mục chương trình todo

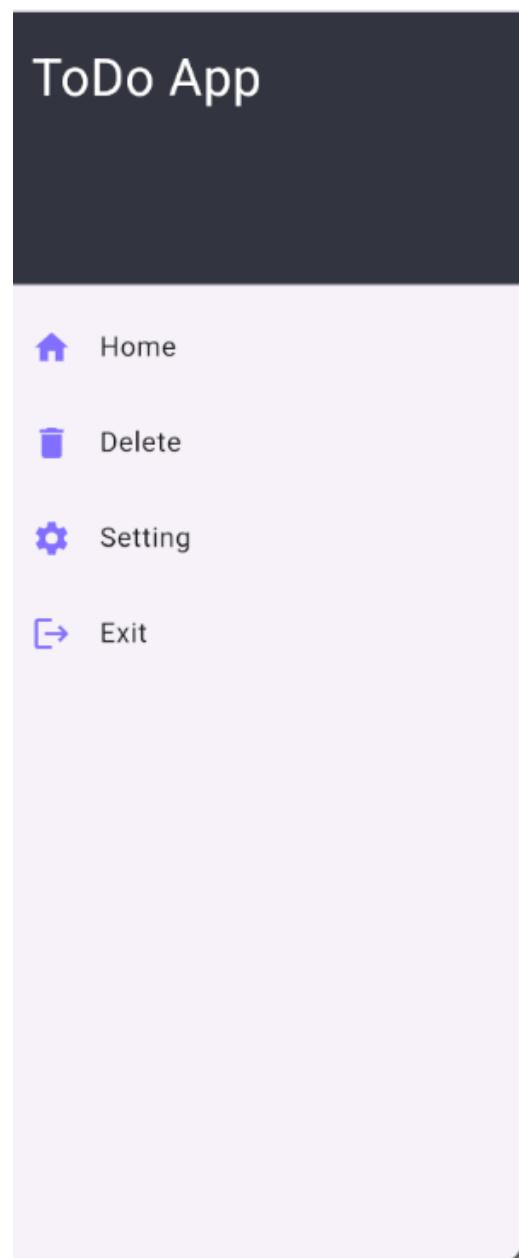
Chi tiết thư mục:

- Constraints:
 - Khai báo các hằng số được sử dụng trong chương trình.
- Database:
 - Giả lập một database để lưu trữ các todo item hay delete item. Có thể gọi nó là một local database.
- Model:
 - Khai báo các đối tượng todo, delete, để làm đối số cho các item để hiện thị lên màn hình.

- Screens:
 - Các màn hình hiển thị chính cho ứng dụng.
- Widgets:
 - Các widget sử dụng trong ứng dụng như: thanh tìm kiếm, các nút bấm hay thanh bên cạnh trái màn hình.

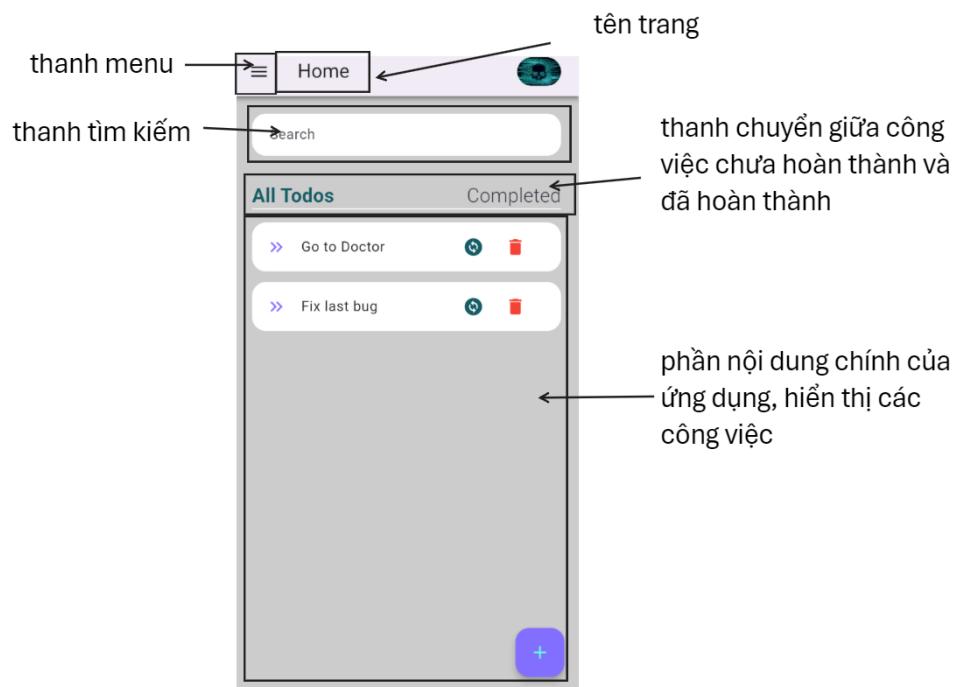
3.2.3. Demo chương trình

Giao diện thanh menu:



Hình 3. 6. Menu của ứng dụng

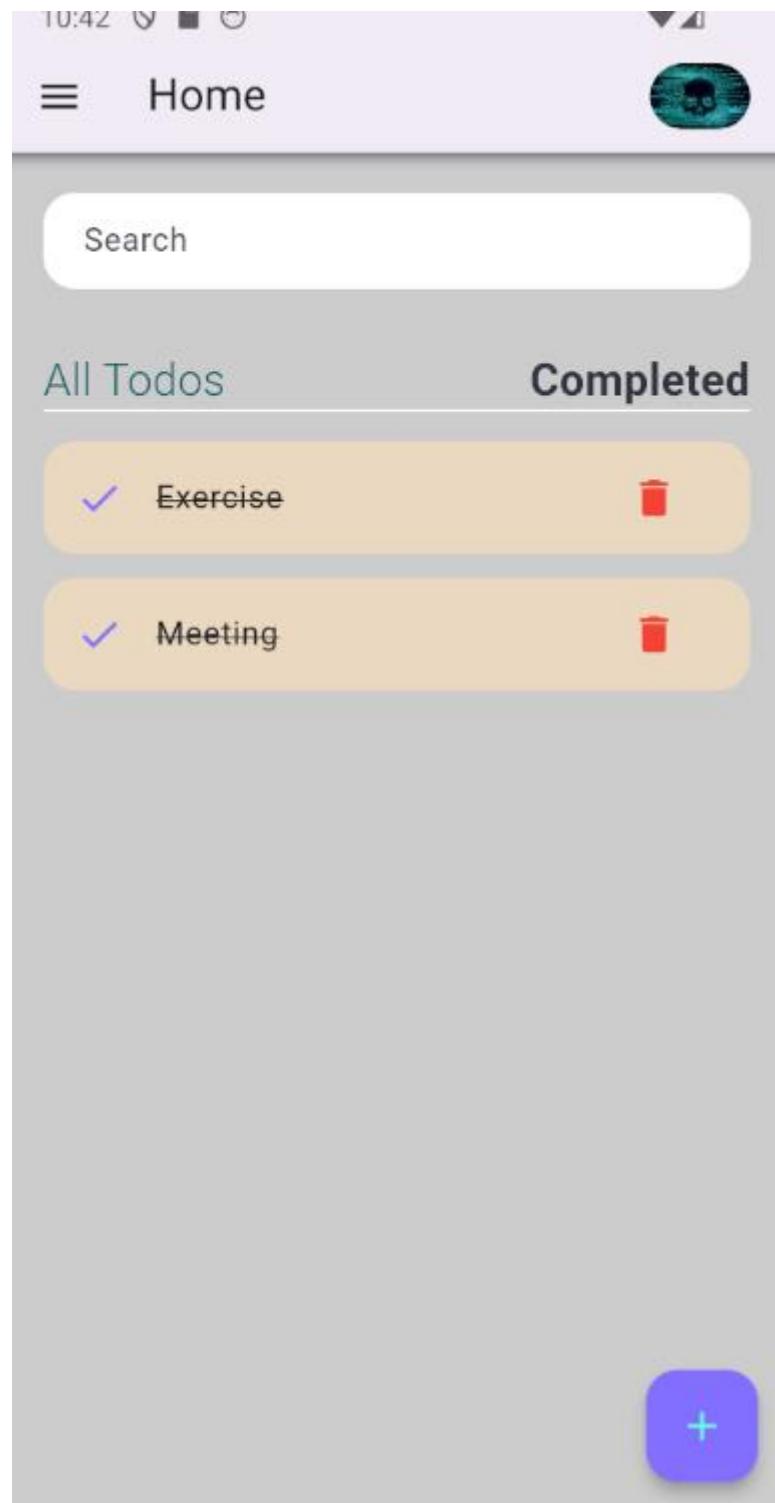
Giao diện khi mở ứng dụng: Sẽ hiển thị mặc định các việc cần làm ở ‘all todos’ và các công việc sẽ hiện thị với nền trắng.



Hình 3. 7. Màn hình chính khi mở ứng dụng

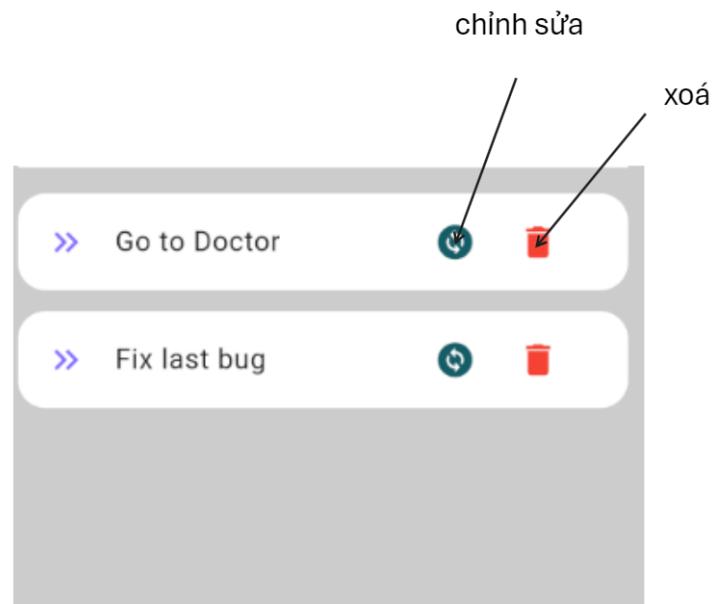
Để hoàn thành công việc chỉ cần nhấn vào công việc đó sau đó công việc đó sẽ hiển thị bên tab ‘completed’.

Khi chuyển sang tab ‘completed’: sẽ hiển thị các công việc đã làm xong với nền cam nhạt .



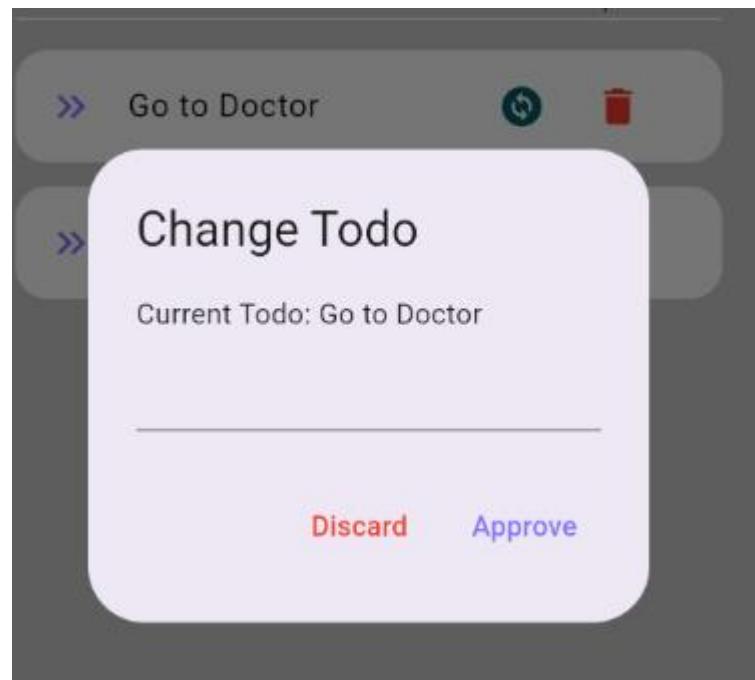
Hình 3. 8. Màn hình các công việc đã xong

Mỗi công việc sẽ có 2 nút là: chỉnh sửa và xoá.



Hình 3. 9. Các thành phần của một item

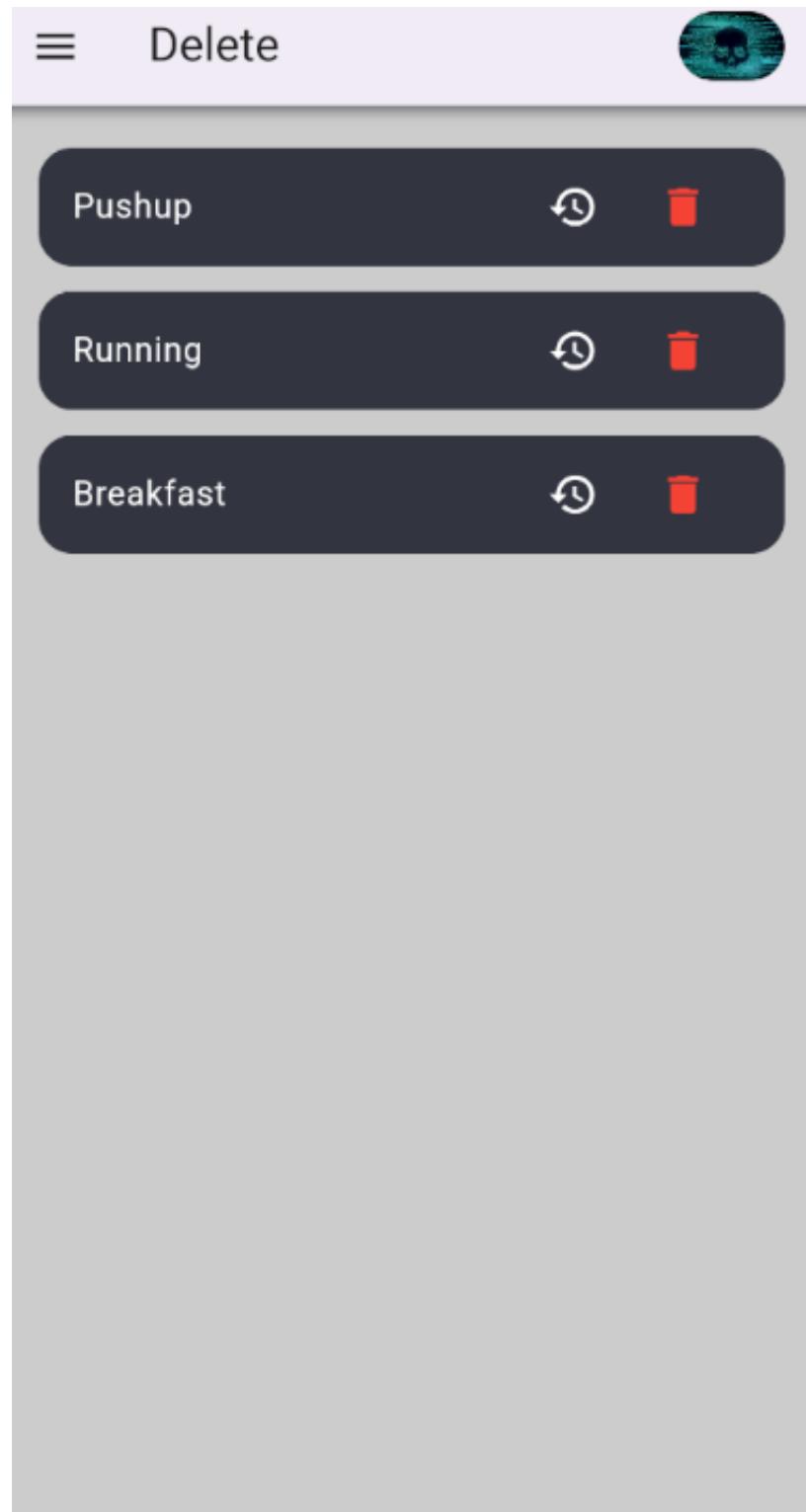
Khi nhấn nút chỉnh sửa sẽ hiện thị lên một popup để cho người dùng thay đổi công việc của mình.



Hình 3. 10. Giao diện sửa chữa công việc

Khi nhấn nút xoá thì công việc được xoá ở giao diện hiện tại và được chuyển tới thư mục xoá.

Giao diện hiển thị các công việc được xoá:



Hình 3. 11. Giao diện các item đã xoá

Cũng giống với các công việc ở trang home mỗi item sẽ có 2 nút là: hoàn tác và xoá vĩnh viễn.



Hình 3. 12. Thành phần của delete item

Khi nhấn vào xoá sẽ xoá vĩnh viễn item này.

Khi nhấn vào hoàn tác thì sẽ thêm vào lại danh sách các todo.



Hình 3. 13. Sau khi hoàn tác xoá công việc

3.3. Trò chơi TicTacToe (TicTacToe)

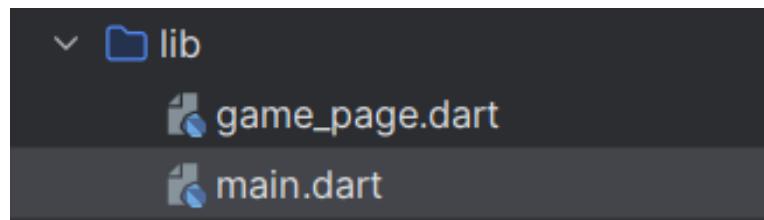
3.3.1. Giới thiệu

TicTacToe (còn gọi là XOX) là một trò chơi kinh điển, đơn giản nhưng đầy thú vị. Với Flutter, bạn có thể dễ dàng tạo ra một phiên bản trò chơi này với giao diện đẹp mắt và logic chơi hoàn chỉnh. Trò chơi dành cho hai người chơi, mỗi người chơi lần lượt đánh dấu X hoặc O vào các ô trên bảng 3x3. Người chơi đầu tiên tạo ra một hàng, cột hoặc đường chéo gồm 3 dấu hiệu của mình sẽ chiến thắng. Nếu tất cả các ô được đánh dấu mà không có người chiến thắng, trò chơi kết thúc hòa.

3.3.2. Cấu trúc chương trình

3.3.2.1. Phân loại thư mục

Do là một chương trình đơn giản và chỉ thao tác trên một màn hình duy nhất. Cho nên cấu trúc thư mục của chương trình cũng đơn giản và dễ hiểu, thư mục của chương trình được biểu thị như sau:



Hình 3. 14. Thư mục chương trình Tic Tac Toe

Thư mục:

- **lib:** Chứa mã nguồn cho ứng dụng.
 - **main.dart** khởi tạo ứng dụng Flutter và thiết lập thông tin cơ bản, sau đó khởi tạo widget GamePage để hiển thị giao diện và xử lý logic trò chơi Tic Tac Toe.
 - **game_page.dart** định nghĩa lớp GamePage là widget trạng thái, chịu trách nhiệm quản lý trạng thái, hiển thị giao diện và xử lý logic trò chơi Tic Tac Toe.

3.3.2.2. Các hàm mặc định

❖ **Hàm khởi tạo và quản lý trạng thái:** Hàm này đảm nhiệm việc khởi tạo và quản lý trạng thái ban đầu của trò chơi. Nó thiết lập các biến như người chơi hiện tại, trạng thái kết thúc trò chơi và danh sách các ô trên bàn cờ để bắt đầu trò chơi từ một trạng thái ban đầu và chuẩn bị sẵn sàng cho việc chơi.

- **initState():** Hàm này được gọi khi widget được tạo ra. Trong trường hợp này, nó được sử dụng để khởi tạo trạng thái ban đầu của trò chơi bằng cách gọi hàm **initializeGame()**
- **initializeGame():** Hàm này khởi tạo trạng thái ban đầu của trò chơi bao gồm người chơi hiện tại, trạng thái kết thúc trò chơi và danh sách các ô trên bàn cờ.

```
void initState() { // Phương thức khởi tạo State.
    initializeGame(); // Khởi tạo trạng thái ban đầu của trò chơi.
    super.initState();
}

void initializeGame() { // Phương thức khởi tạo trò chơi.
    currentPlayer = PLAYER_X; // Người chơi hiện tại là X.
    gameEnd = false; // Trò chơi chưa kết thúc.
    occupied = ["", "", "", "", "", "", "", "", ""];
    // Khởi tạo các ô trống trên bàn cờ.
}
```

Hình 3. 15. Khởi tạo trò chơi TicTacToe

- **createState():** Được sử dụng để tạo một State mới cho widget. Trong trường hợp này, nó tạo một _GameState mới để quản lý trạng thái và tương tác của trang chơi game.
- ❖ **Hàm xây dựng giao diện:** Hàm này chịu trách nhiệm xây dựng giao diện người dùng cho trò chơi. Nó tạo ra các thành phần như tiêu đề, bàn cờ và nút khởi động lại trò chơi, giúp người chơi tương tác và tham gia vào trò chơi một cách thuận tiện.
 - **build(BuildContext context):** Hàm này xây dựng giao diện chính của trò chơi, bao gồm _headerText() (tiêu đề), _gameContainer() (bàn cờ) và _restartButton() (nút khởi động lại trò chơi).

- **_headerText()**: Tạo tiêu đề cho trò chơi, bao gồm tiêu đề "Tic Tac Toe" và thông tin về lượt chơi hiện tại.
- **_gameContainer()**: Tạo bàn cờ trò chơi bằng cách sử dụng GridView để hiển thị danh sách các ô trên bàn cờ.

```
- child: GridView.builder( // Sử dụng GridView để hiển thị bàn cờ.
    gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount( // Xác định cách sắp xếp ô trong GridView.
        crossAxisCount: 3), // Số ô trên mỗi hàng của GridView. // SliverGridDelegateWithFixedCrossAxisCount
    itemCount: 9, // Số lượng ô trên bàn cờ.
    itemBuilder: (context, int index) { // Hàm xây dựng ô trên bàn cờ.
        return _box(index); // Trả về widget của ô trên bàn cờ.
    },
), // GridView.builder
```

Hình 3. 16. Sử dụng GridView để hiển thị các ô trong TicTacToe

- **box(int index)**: Tạo một ô trên bàn cờ. Mỗi ô sẽ là một *InkWell* để xử lý sự kiện khi người dùng nhấp vào ô.

```
return InkWell( // InkWell để xử lý sự kiện khi người dùng nhấp vào ô.
    onTap: () { // Hành động khi người dùng nhấp vào ô.
        if (gameEnd || occupied[index].isNotEmpty) { // Kiểm tra xem trò chơi đã kết thúc hoặc ô đã được chọn.
            return; // Không làm gì nếu trò chơi đã kết thúc hoặc ô đã được chọn.
        }

        setState(() { // Cập nhật trạng thái của trò chơi.
            occupied[index] = currentPlayer; // Đánh dấu ô đã được chọn bởi người chơi hiện tại.
            changeTurn(); // Chuyển lượt chơi cho người chơi tiếp theo.
            checkForWinner(); // Kiểm tra xem có người chiến thắng không.
            checkForDraw(); // Kiểm tra xem có hòa không.
        });
    },
},
```

Hình 3. 17. InkWell để xử lý sự kiện khi người dùng nhấp vào ô.

- **restartButton()**: Tạo nút khởi động lại trò chơi, cho phép người dùng bắt đầu lại trò chơi từ đầu.
- ❖ **Hàm xử lý luồng trò chơi:** Hàm này điều chỉnh luồng của trò chơi bằng cách xác định lượt chơi, kiểm tra người chiến thắng, và kiểm tra trạng thái hòa. Nó đảm bảo rằng trò chơi diễn ra một cách hợp lý và tự động kết thúc khi có người chiến thắng hoặc hòa.

- **changeTurn()**: Chuyển lượt chơi từ người chơi hiện tại sang người chơi tiếp theo (chuyển giữa X và O).
- **checkForWinner()**: Kiểm tra xem có người chiến thắng không bằng cách so sánh các vị trí trên bàn cờ để xác định xem có ai thắng hay không.

```

List<List<int>> winningList = [
    // Danh sách các vị trí có thể chiến thắng.
    [0, 1, 2],
    [3, 4, 5],
    [6, 7, 8],
    [0, 3, 6],
    [1, 4, 7],
    [2, 5, 8],
    [0, 4, 8],
    [2, 4, 6],
];

```

Hình 3. 18. Danh sách các vị trí có thể chiến thắng.

- **checkForDraw()**: Kiểm tra trò chơi hòa bằng cách kiểm tra xem tất cả các ô trên bàn cờ đã được chọn hay chưa.
- ❖ **Hàm hiển thị thông báo: showGameOverMessage(String message)**: Hiển thị thông báo kết thúc trò chơi thông qua một Snackbar, bao gồm thông báo về người chiến thắng hoặc trạng thái hòa của trò chơi.

3.3.2.3. Màn hình hiển thị

Trong chương trình, chỉ có duy nhất một màn hình chính gồm:

1. Khu vực hiển thị giao diện, các ô trong trò chơi Tic Tac Toe và nút bắt đầu lại trò chơi.
2. Khu vực hiển thị thông báo người chơi thắng hay hòa.

Toàn bộ đoạn code cho màn hình chính nằm trong File **game_page.dart**.

```

import 'dart:ui';

import 'package:flutter/material.dart';

class GamePage extends StatefulWidget {
  @override
  State<GamePage> createState() => _GamePageState();
}

class _GamePageState extends State<GamePage> {
  static const String PLAYER_X = "X";
  static const String PLAYER_Y = "O";
}

```

```

late String currentPlayer;
late bool gameEnd;
late List<String> occupied;

@Override
void initState() {
    initializeGame();
    super.initState();
}

void initializeGame() {
    currentPlayer = PLAYER_X;
    gameEnd = false;
    occupied = ["", "", "", "", "", "", "", "", ""];
    //9 empty places
}

@Override
Widget build(BuildContext context) {
    return Scaffold(
        body: Center(
            child: Column(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                    _headerText(),
                    _gameContainer(),
                    _restartButton(),
                ],
            ),
        ),
    );
}

Widget _headerText() {
    return Column(
        children: [
            const Text(
                "Tic Tac Toe",
                style: TextStyle(
                    color: Colors.green,
                    fontSize: 32,
                    fontWeight: FontWeight.bold,
                ),
            ),
            Text(
                "$currentPlayer turn",
                style: const TextStyle(
                    color: Colors.black87,
                ),
            ),
        ],
    );
}

```

```

        fontSize: 32,
        fontWeight: FontWeight.bold,
      ),
    ),
  ],
);
}

Widget _gameContainer() {
  return Container(
    height: MediaQuery.of(context).size.height / 2,
    width: MediaQuery.of(context).size.height / 2,
    margin: const EdgeInsets.all(8),
    child: GridView.builder(
      gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
        crossAxisCount: 3),
      itemCount: 9,
      itemBuilder: (context, int index) {
        return _box(index);
      },
    );
}

Widget _box(int index) {
  return InkWell(
    onTap: () {
      //on click of box
      if (gameEnd || occupied[index].isNotEmpty) {
        //Return if game already ended or box already clicked
        return;
      }

      setState(() {
        occupied[index] = currentPlayer;
        changeTurn();
        checkForWinner();
        checkForDraw();
      });
    },
    child: Container(
      color: occupied[index].isEmpty
        ? Colors.black26
        : occupied[index] == PLAYER_X
        ? Colors.blue
        : Colors.orange,
      margin: const EdgeInsets.all(8),
      child: Center(

```

```

        child: Text(
            occupied[index],
            style: const TextStyle(fontSize: 50),
        ),
    ),
),
);
}

_restartButton() {
    return ElevatedButton(
        onPressed: () {
            setState(() {
                initializeGame();
            });
        },
        child: const Text("Restart Game"));
}

changeTurn() {
    if (currentPlayer == PLAYER_X) {
        currentPlayer = PLAYER_Y;
    } else {
        currentPlayer = PLAYER_X;
    }
}

checkForWinner() {
    //Define winning positions
    List<List<int>> winningList = [
        [0, 1, 2],
        [3, 4, 5],
        [6, 7, 8],
        [0, 3, 6],
        [1, 4, 7],
        [2, 5, 8],
        [0, 4, 8],
        [2, 4, 6],
    ];

    for (var winningPos in winningList) {
        String playerPosition0 = occupied[winningPos[0]];
        String playerPosition1 = occupied[winningPos[1]];
        String playerPosition2 = occupied[winningPos[2]];

        if (playerPosition0.isNotEmpty) {
            if (playerPosition0 == playerPosition1 &&

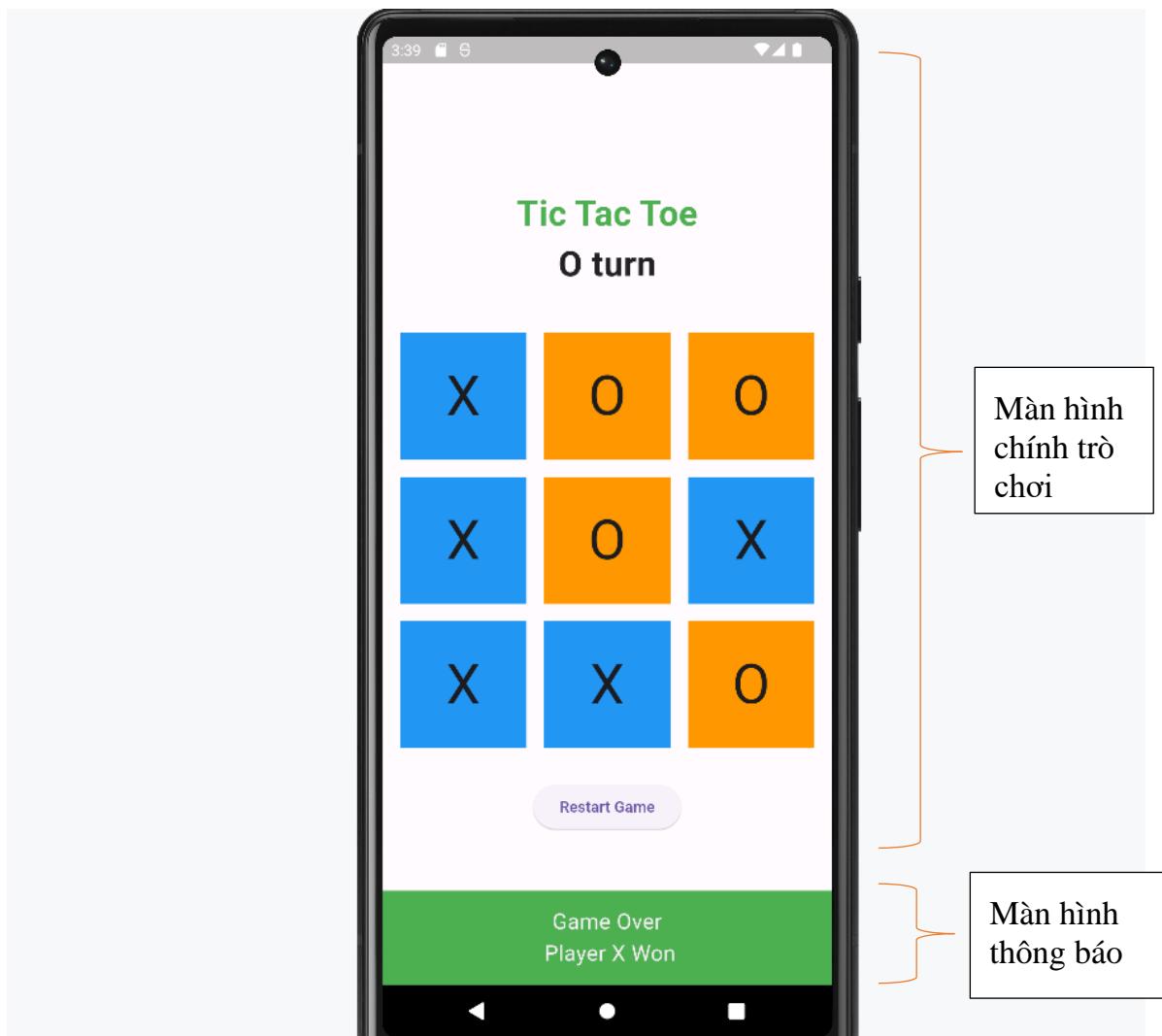
```

```
        playerPosition0 == playerPosition2) {
            //all equal means player won
            showGameOverMessage("Player $playerPosition0 Won");
            gameEnd = true;
            return;
        }
    }
}
}

checkForDraw() {
    if (gameEnd) {
        return;
    }
    bool draw = true;
    for (var occupiedPlayer in occupied) {
        if (occupiedPlayer.isEmpty) {
            //at least one is empty not all are filled
            draw = false;
        }
    }

    if (draw) {
        showGameOverMessage("Draw");
        gameEnd = true;
    }
}

showGameOverMessage(String message) {
    ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
            backgroundColor: Colors.green,
            content: Text(
                "Game Over \n$message",
                textAlign: TextAlign.center,
                style: const TextStyle(
                    fontSize: 20,
                ),
            ),
        ),
    );
}
```



Hình 6 Màn hình được chia làm 2 màn hình phụ

❖ Màn hình chính trò chơi

Trong màn hình trò chơi thì sẽ có:

- Tiêu đề: Hiển thị tiêu đề trò chơi "Tic Tac Toe" và xử lý hiện thị lượt chơi hiện tại ("X" hoặc "O") lên màn hình.

```
Widget _headerText() { // Widget hiển thị tiêu đề trò chơi.
  return Column( // Sắp xếp các thành phần theo chiều dọc.
    children: [
      const Text( // Tiêu đề "Tic Tac Toe".
        "Tic Tac Toe",
        style: TextStyle(
          color: Colors.green,
          fontSize: 32,
          fontWeight: FontWeight.bold,
        ), // TextStyle
      ), // Text
      Text( // Hiển thị lượt chơi hiện tại.
        "$currentPlayer turn",
        style: const TextStyle(
          color: Colors.black87,
          fontSize: 32,
          fontWeight: FontWeight.bold,
        ), // TextStyle
      ), // Text
    ],
  ); // Column
}
```

Hình 3.19. Hiển thị tiêu đề và lượt chơi người hiện tại

Vùng chơi

- Mỗi ô được biểu diễn bởi widget **_box**.
- Nhấp vào ô sẽ kích hoạt hàm onTap của widget **_box**.
- Ô vuông thay đổi màu sắc và hiển thị "X" hoặc "O" dựa trên người chơi hiện tại và trạng thái đã chọn.

```

Widget _box(int index) { // Widget hiển thị một ô trên bàn cờ.
    return InkWell( // InkWell để xử lý sự kiện khi người dùng nhấn vào ô.
        onTap: () { // Hành động khi người dùng nhấn vào ô.
            if (gameEnd || occupied[index].isNotEmpty) { // Kiểm tra xem trò chơi đã kết thúc hoặc
                return; // Không làm gì nếu trò chơi đã kết thúc hoặc ô đã được chọn.
            }

            setState(() { // Cập nhật trạng thái của trò chơi.
                occupied[index] = currentPlayer; // Đánh dấu ô đã được chọn bởi người chơi hiện tại.
                changeTurn(); // Chuyển lượt chơi cho người chơi tiếp theo.
                checkForWinner(); // Kiểm tra xem có người chiến thắng không.
                checkForDraw(); // Kiểm tra xem có hòa không.
            });
        },
        child: Container(
            color: occupied[index].isEmpty // Màu nền của ô trên bàn cờ.
            ? Colors.black26 // Nếu ô trống.
            : occupied[index] == PLAYER_X // Nếu ô được chọn bởi người chơi X.
            ? Colors.blue // Màu của người chơi X.
            : Colors.orange, // Màu của người chơi O.
            margin: const EdgeInsets.all(8), // Khoảng cách giữa các ô.
            child: Center(
                child: Text(
                    occupied[index], // Hiển thị ký tự của người chơi hiện tại trên ô.
                    style: const TextStyle(fontSize: 50), // Kích thước và kiểu chữ.
                ),
            ),
        ),
    ); // InkWell
}

```

Hình 3. 20. Hiển thị vùng chơi Tic Tac Toe và xử lý hành động người chơi

Nút khởi động lại trò chơi khi người dùng ấn vào sẽ khởi động lại trò chơi.

```
_restartButton() { // Widget nút khởi động lại trò chơi.
  return ElevatedButton(
    onPressed: () { // Hành động khi người dùng nhấn vào nút.
      setState(() { // Cập nhật trạng thái của trò chơi.
        initializeGame(); // Khởi động lại trò chơi.
      });
    },
    child: const Text("Restart Game")); // Văn bản trên nút. // ElevatedButton
}
```

Hình 3. 21. Hiển thị vùng chơi Tic Tac Toe và xử lý hành động người chơi

❖ Màn hình thông báo

Màn hình này không hiển thị trực tiếp mà hiện ra tạm thời thông qua widget Snackbar (Là một widget Material Design được sử dụng để hiển thị thông báo ngắn gọn, tạm thời cho người dùng tại phần dưới màn hình).

Xuất hiện trong các trường hợp sau:

- Khi trò chơi kết thúc với người chiến thắng ("Người chơi X chiến thắng" hoặc "Người chơi O chiến thắng").
- Khi trò chơi kết thúc hòa ("Hòa").

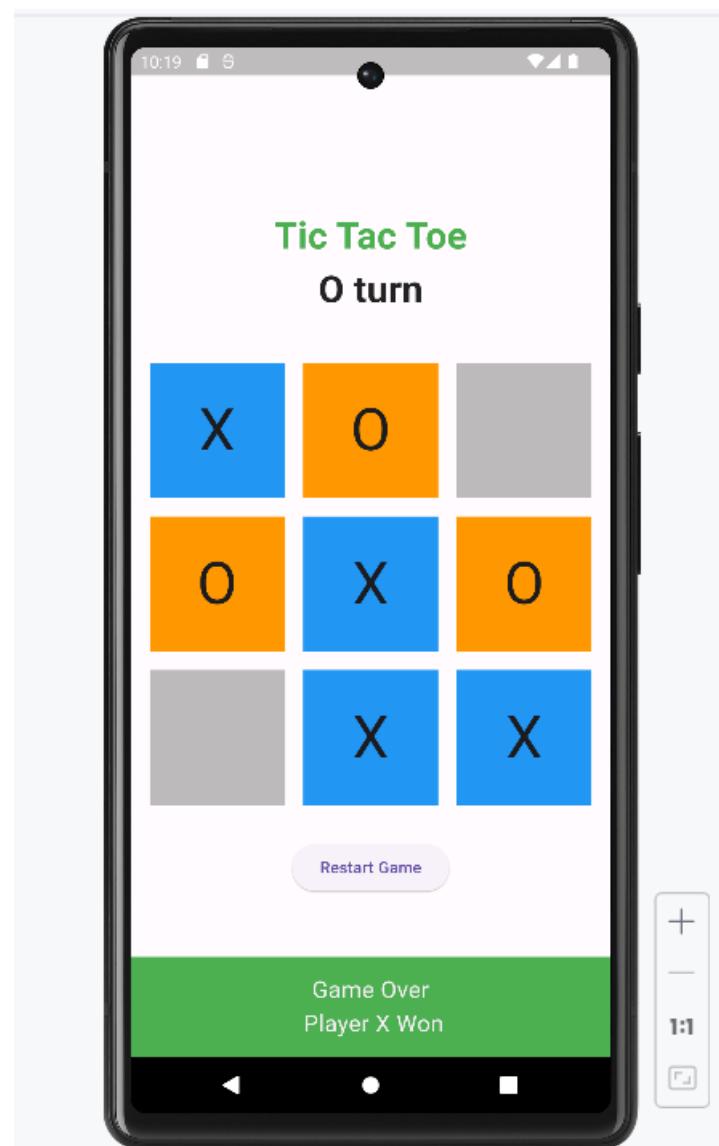
Thông báo hiển thị tin nhắn ("Kết thúc trò chơi \n Người chiến thắng hoặc Hòa") và có nền màu xanh lá cây.

```
showGameOverMessage(String message) { // Hiển thị thông báo kết thúc trò chơi.
  ScaffoldMessenger.of(context).showSnackBar( // Hiển thị Snackbar.
    SnackBar(
      backgroundColor: Colors.green, // Màu nền của Snackbar.
      content: Text(
        "Game Over \n $message", // Nội dung của Snackbar.
        textAlign: TextAlign.center, // Căn giữa nội dung.
        style: const TextStyle(
          fontSize: 20, // Kích thước chữ.
        ), // TextStyle
      )), // Text, SnackBar
  );
}
```

Hình 3. 22. Hiển thị thông báo kết thúc trò chơi.

3.3.3. Demo chương trình

1. Mở ứng dụng Tic Tac Toe
2. Người chơi 1 nhấp vào ô trống trên bàn cờ Tic Tac Toe
3. Người chơi 2 nhấp vào ô trống khác
4. Lặp lại đến khi tìm được người chiến thắng hoặc hòa
5. Hiện thông báo người chiến thắng hoặc hòa



Hình 3. 23. Hình ảnh demo trò chơi TicTacToe

3.4. Ứng dụng Tính toán chỉ số BMI (BMI Calculator)

3.4.1. Giới thiệu

BMI Calculator là ứng dụng di động giúp ta tính toán và theo dõi Chỉ số Khối cơ thể (BMI) một cách nhanh chóng và chính xác. Chỉ với vài thao tác đơn giản, ta có thể: Nhập chiều cao và cân nặng, xem kết quả BMI và cuối cùng là biết được ý nghĩa của kết quả BMI (thiếu cân, bình thường, thừa cân, béo phì).

Trong phần này, chương trình sẽ sử dụng các kỹ thuật xây dựng ứng dụng như:

- Thiết lập thay đổi Theme Mode:** Ví dụ: Light Mode và Dark Mode tùy theo chế độ theme mode mà hệ thống điện thoại đang sử dụng.
- Sử dụng GetX để thao tác chuyển màn hình:** Thay vì dùng Navigation để di chuyển trang như bình thường thì trong ứng dụng này sẽ sử dụng GetX để di chuyển trang. **GetX là một thư viện bên thứ 3.**

GetX là gì ? GetX là một thư viện Flutter phổ biến cung cấp nhiều tính năng mạnh mẽ để đơn giản hóa quá trình phát triển ứng dụng. Nó được biết đến với khả năng quản lý state dễ dàng, điều hướng linh hoạt và tiêm phụ thuộc hiệu quả.

Ưu điểm của GetX:

- Dễ sử dụng:** GetX cung cấp API đơn giản và dễ hiểu, giúp bạn dễ dàng bắt đầu sử dụng.
- Linh hoạt:** GetX có thể được sử dụng cho nhiều trường hợp sử dụng khác nhau, từ các ứng dụng đơn giản đến phức tạp.
- Hiệu quả:** GetX được tối ưu hóa cho hiệu suất, giúp cải thiện tốc độ và khả năng phản hồi của ứng dụng của bạn.
- Cộng đồng lớn:** GetX có một cộng đồng người dùng lớn và tích cực, cung cấp hỗ trợ và tài nguyên phong phú.

Nhược điểm của GetX:

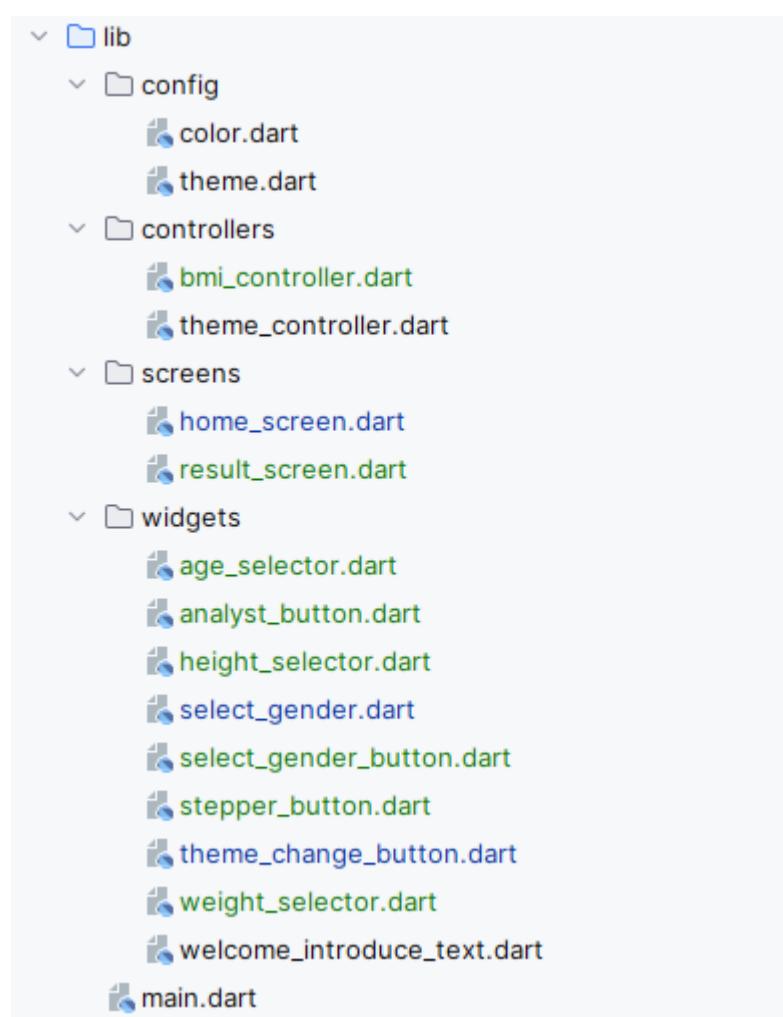
- Mới hơn so với các giải pháp khác:** GetX là một thư viện tương đối mới, vì vậy nó có thể ít được kiểm tra hơn các giải pháp khác.

2. Có thể dẫn đến spaghetti code: Nếu không sử dụng cẩn thận, GetX có thể dẫn đến spaghetti code, khó bảo trì và gỡ lỗi.
3. Học thêm: Việc học GetX có thể tốn thêm thời gian so với các giải pháp quản lý state truyền thống.

3.4.2. Cấu trúc chương trình

3.4.2.1. Phân loại thư mục

Cấu trúc thư mục của chương trình được xây dựng như sau:



Hình 3. 24. Cấu trúc thư mục chương trình tính toán BMI

- **config:** Là Folder chứa các File cấu hình chương trình, hệ thống
 - **color.dart:** Gồm các dòng code định nghĩa màu sắc chính, chủ đạo của toàn bộ chương trình.

- **theme.dart:** Chứa các thuộc tính, phương thức định nghĩa **Chủ đề (Theme)** của toàn bộ hệ thống, chương trình. Ở đây gồm **Chủ đề Sáng** và **Chủ đề Tối**.
- **controllers:** Là Folder chuyên chứa các File cốt lõi của chương trình, hệ thống. Chịu trách nhiệm vận hành và lưu trữ các biến, xử lý và hiển thị các thông tin lên màn hình.
 - **bmi_controller.dart:** Chịu trách nhiệm xử lý và chứa các thuộc tính liên quan đến chỉ số BMI như **cân nặng, chiều cao, độ tuổi....** Và bên cạnh đó, nó đóng vai trò cốt lõi cho cả chương trình.
 - **theme_controller.dart:** Chịu trách nhiệm xử lý và thay đổi **chủ đề** hệ thống khi cần thiết.
- **screens:** Là Folder chứa các File liên quan đến **các màn hình hiển thị** của chương trình hệ thống
 - **home_screen.dart:** Là màn hình chính của cả chương trình. Người dùng sẽ thực hiện nhập thông tin của mình để tính toán BMI.
 - **result_screen.dart:** Là màn hình hiển thị kết quả tính toán BMI cho người dùng, bao gồm chỉ số và một số lời khuyên.
- **widgets:** Là Folder chứa các Widget đã được **cắt ra** từ màn hình chính như **home_screen** và **result_screen** nhằm mục đích tối giản code và tăng tính tái sử dụng code.
 - **welcome_introduce_text.dart:** Widget chứa phân vùng chữ hiển thị **lời chào** khi vào chương trình.
 - **age_selector.dart:** Widget chứa phân vùng nhập độ tuổi.
 - **height_selector.dart:** Widget chứa phân vùng nhập chiều cao.
 - **weight_selector.dart:** Widget chứa phân vùng nhập cân nặng.
 - **select_gender.dart:** Widget chứa phân vùng lựa chọn giới tính.
 - **select_gender_button.dart:** Widget nút bấm cho phân vùng chọn giới tính.
 - **theme_change_button.dart:** Widget chứa phân vùng nút bấm giúp thay đổi **chủ đề hiện tại** của chương trình.

- **stepper_button.dart:** Widget nút bấm được sử dụng cho phân vùng như **tăng và giảm**.
- **analyst_button.dart:** Widget này chứa nút bấm thực hiện xử lý và chuyển sang trang **kết quả**.

3.4.2.2. Thiết lập màu sắc và chủ đề

Để chương trình có thể thay đổi giao diện tuỳ ý dựa trên Theme Mode mà hệ thống Android/IOS của người dùng hiện tại đang sử dụng thì ta cần một Controller và các thuộc tính đã được thiết lập sẵn để sẵn sàng thay đổi khi cần thiết. Các thuộc tính được thiết lập ảnh hưởng đến các thuộc tính mặc định, nói cách khác là ghi đè nó theo ý thích và tuỳ chỉnh của mình, những thuộc tính đó là **background**, **primary**, **onBackground**, **primaryContainer**, **onPrimaryContainer**.... Hoặc cũng có thể dùng các màu sắc đã được thay đổi đó làm màu chữ, màu nền nếu thích.

Các bước thiết lập màu sắc và Controller được mô tả chi tiết như sau:

❖ Bước 1: Cấu hình thuộc tính

Trong đó, màu sắc của các thành phần được định nghĩa như sau:

1. Màu sắc (nằm trong File **color.dart** ở thư mục **config**):

```
// Light Mode
const lightBackgroundColor = Color(0xffD1D9E6);
const lightPrimaryColor = Color(0xff264AFE);
const lightLabelColor = Color(0xff8C8C8C);
const lightDivColor = Color(0xffffffff);
const lightFontColor = Color(0xff000000);

// Dark Mode
const darkBackgroundColor = Color(0xff242424);
const darkPrimaryColor = Color(0xff264AFE);
const darkLabelColor = Color(0xff8C8C8C);
const darkDivColor = Color(0xff373737);
const darkFontColor = Color(0xffffffff);
```

2. Theme (nằm trong File **theme.dart** ở thư mục **config**):

```
// Light Mode
var lightTheme = ThemeData(
  useMaterial3: true,
  colorScheme: const ColorScheme.light(
    background: lightBackgroundColor,
    primary: lightPrimaryColor,
    onBackground: lightFontColor,
    primaryContainer: lightDivColor,
    onPrimaryContainer: lightFontColor,
    onSecondaryContainer: lightLabelColor,
  ),
);

// Dark Mode
var darkTheme = ThemeData(
  useMaterial3: true,
  colorScheme: const ColorScheme.dark(
    background: darkBackgroundColor,
    primary: darkPrimaryColor,
    onBackground: darkFontColor,
    primaryContainer: darkDivColor,
    onPrimaryContainer: darkFontColor,
    onSecondaryContainer: darkLabelColor,
  ),
);
```

Và cả 2 theme này được cấu hình ^đang vào trong **main.dart**:

```
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
```

```

return GetMaterialApp(
    title: 'BMI Calculator',
    theme: lightTheme,
    darkTheme: darkTheme,
    debugShowCheckedModeBanner: false,
    home: const HomeScreen(),
);
}
}

```

Việc cấu hình trước **lightTheme** và **darkTheme** trong chương trình cho thấy việc chủ động thiết kế ứng dụng để hỗ trợ cả chế độ sáng và chế độ tối cho người dùng. Mục đích hướng tới những điểm sau:

1. Tăng tính linh hoạt của ứng dụng
2. Tăng trải nghiệm của người dùng
3. Thể hiện sự chuyên nghiệp và chu đáo

❖ **Bước 2: Cấu hình thuộc tính**

Sau khi cấu hình xong thuộc tính cần thiết để thay đổi chế độ sáng/tối. Ta đến bước tiếp theo là thiết lập hàm để có thể gọi đến hệ thống và yêu cầu thay đổi khi cần thiết. Đoạn code thiết lập điều khiển (controller) nằm trong File **theme_controller.dart** ở thư mục **controllers** :

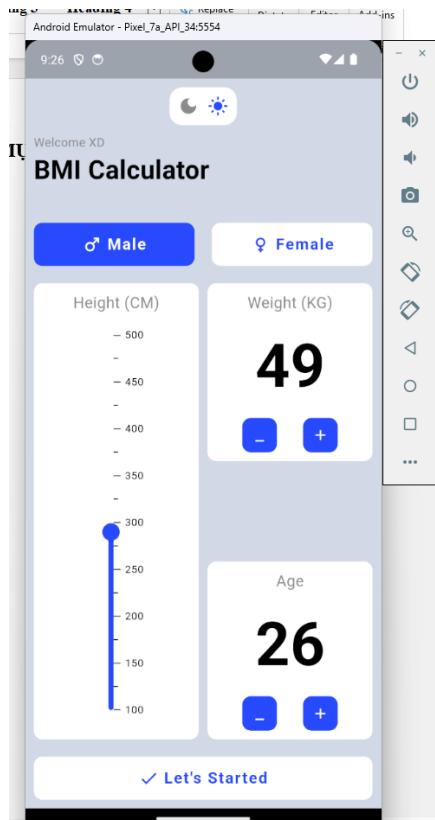
```

class ThemeController extends GetxController {
    RxBool isDark = false.obs;
    void changeTheme() async {
        isDark.value = !isDark.value;
        Get.changeThemeMode(isDark.value ? ThemeMode.dark : ThemeMode.light);
    }
}

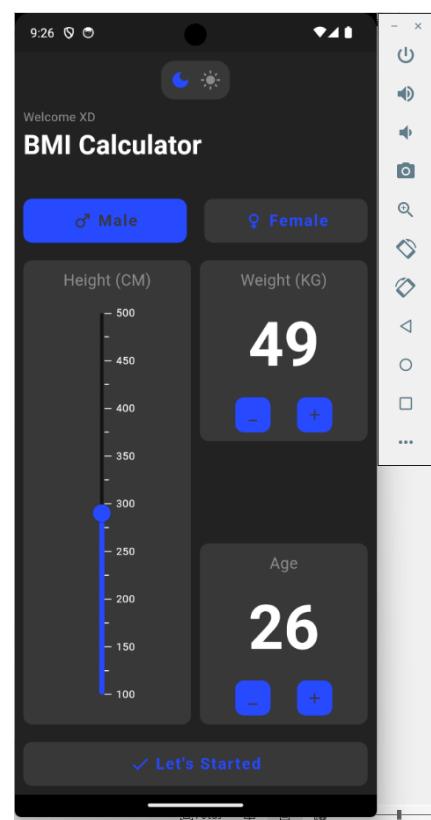
```

Cách hoạt động: Mỗi lần ta gọi hàm **changeTheme()** thì hệ thống sẽ tự động thay đổi chủ đề của ứng dụng thông qua thư viện **GetX** bằng lớp **GetxController** (Lớp này đóng vai trò quản lý trạng thái và chủ đề của ứng dụng).

Áp dụng lý thuyết đó vào trong chương trình hiện tại, ta có giao diện như sau:



Hình 3. 26. Chương trình BMI Calculator chủ đề sáng



Hình 3. 25. Chương trình BMI Calculator chủ đề tối

3.4.2.3. Thiết lập GetX Controller

Vì trong chương trình này sử dụng thêm thư viện **GetX** để quản lý trạng thái. Do đó, thư viện này đã được sử dụng để quản lý các thông tin input và output từ giao diện vào trong chương trình và lưu trữ chúng vào trong các biến đã được định sẵn. Sử dụng **GetX** để quản lý trạng thái có những ích lợi sau:

- Tự động cập nhật:** Khi ta thay đổi giá trị của một biến Rx, bất kỳ widget nào đang quan sát biến đó sẽ tự động cập nhật để phản ánh giá trị mới. Điều này giúp ta dễ dàng viết code UI phản ứng với thay đổi trạng thái.
- Gỡ lỗi dễ dàng hơn:** Các biến Rx giúp dễ dàng gỡ lỗi code vì chúng cung cấp cho ta thông tin chi tiết về cách giá trị của chúng thay đổi theo thời gian.
- Code gọn gàng hơn:** Việc sử dụng các biến Rx có thể giúp code gọn gàng hơn vì ta không cần viết code thủ công để cập nhật UI khi trạng thái thay đổi.

Trong chương trình, nhóm đã sử dụng **GetX** với các biến **Rx** để quản lý trạng thái ứng dụng (hay còn được gọi là **controller**) như sau:

```
class BMIController extends GetxController {
    RxString Gender = "Male".obs;
    RxInt Weight = 50.obs;
    RxDouble Height = 100.0.obs;
    RxInt Age = 18.obs;
    RxString BMI = "".obs;
    RxDouble tempBMI = 0.0.obs;
    RxString BMIstatus = "".obs;
    RxString BMIAdvice = "".obs;
    Rx<Color> colorStatus = const Color(0xff246AFE).obs;

    /// Hàm thay đổi giá trị Gender
    void genderHandle(String gender) {.....}

    /// Hàm tính toán BMI
    void calcBMI() {.....}

    /// Hàm set trạng thái
    void findStatus() {....}
```

Đây là Controller quản lý các thông tin của App (cắt đoạn), trong chương trình thì nó là các thông tin như **Gender (giới tính)**, **Weight (Cân nặng)**, **Height (Chiều cao)** và **Age (Độ tuổi)** tương ứng với những thuộc tính cần phải có trong một ứng dụng tính toán BMI

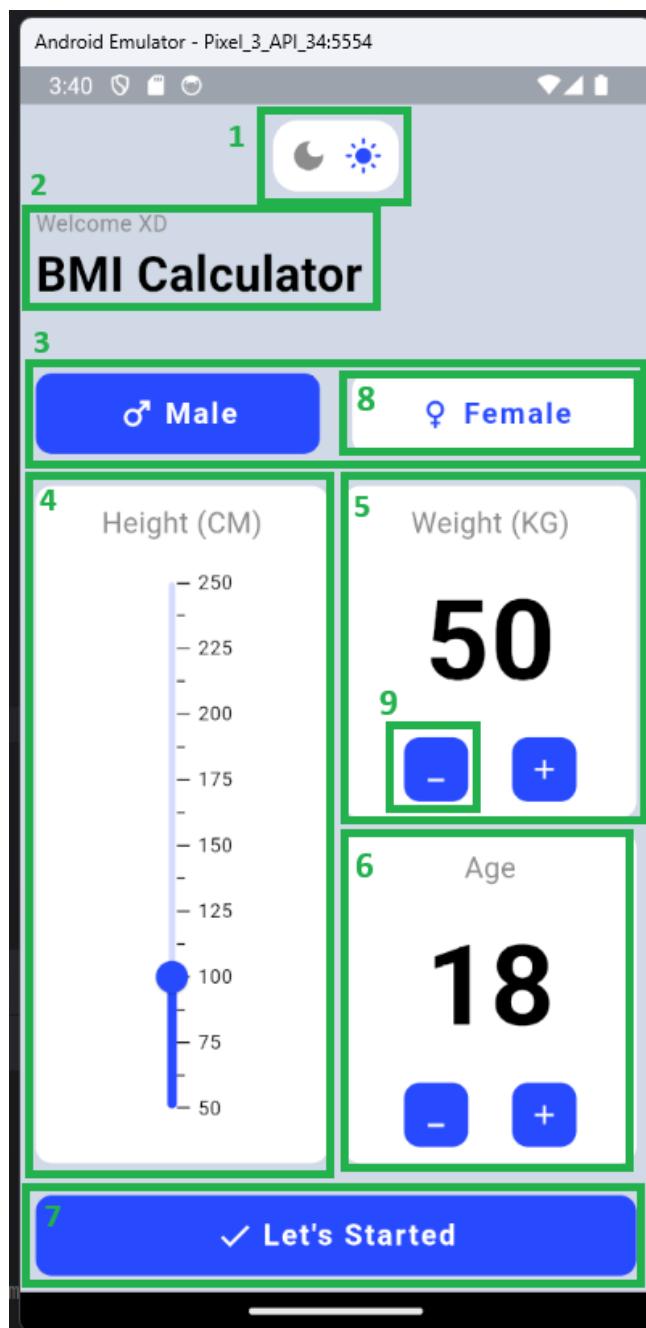
Trong chương trình này sử dụng **GetX** làm quản lý trạng thái là chính cho nên hầu hết các màn hình hay widget cần thiết đều sử dụng nó để thực hiện. Ngoài ra, **theme_controller** cũng sử dụng **GetX** để thay đổi chủ đề,

3.4.2.4. Phân tích giao diện người dùng

Trong phần này, ta sẽ đến với phân tích giao diện chương trình. Giao diện chương trình gồm 2 màn hình chính:

1. **home_srceen.dart**: Là màn hình chính của cả chương trình.
2. **result_screen.dart**: Là màn hình hiển thị kết quả tính toán BMI.

❖ home_screen



Hình 3. 27. Từng thành phần được đánh số trong màn hình chính home_screen của chương trình BMI Calculator

Trong hình trên đã được đánh dấu từng thành phần được tách rời để đảm bảo tính tái tạo code và code tối giản. Lấy ảnh chụp từ chương trình làm minh chứng:

The screenshot shows a file browser on the left and a code editor on the right. The file browser displays the project structure of 'bmi_calculator' with various files and folders like .dart_tool, .idea, android, build, lib, config, controllers, screens, and widgets. The code editor shows a Dart file with code for a home screen. The code uses comments to explain the purpose of different parts of the UI, such as 'Nút chuyển đổi chủ đề giao diện', 'Dòng giới thiệu', 'Khu vực hiển thị', 'Khu vực chọn chiều cao', 'Khu vực chọn cân nặng và độ tuổi', and 'Khu vực chọn độ tuổi'. The code uses Scaffold, SafeArea, Padding, Column, Row, Expanded, and various Selector widgets.

```

return Scaffold(
  body: SafeArea(
    child: Padding(
      padding: const EdgeInsets.all(10),
    ),
    child: Column(
      children: [
        /// Nút chuyển đổi chủ đề giao diện
        const ThemeChangeButton(),
        const SizedBox(height: 10),

        /// Dòng giới thiệu
        const WelcomeAndIntroduceText(),
        const SizedBox(height: 40),

        const SelectGenderWidget(),
        const SizedBox(height: 20),

        /// Khu vực hiển thị
        const Expanded(
          child: Row(
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: [
              /// Khu vực chọn chiều cao
              HeightSelector(),
              SizedBox(width: 10),
            ],
          ),
        ),
        const Expanded(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: [
              /// 1. Khu vực chọn cân nặng
              WeightSelector(),
              const SizedBox(height: 10),
              /// 2. Khu vực chọn độ tuổi
              AgeSelector(),
            ],
          ),
        ),
      ],
    ),
  ),
);

```

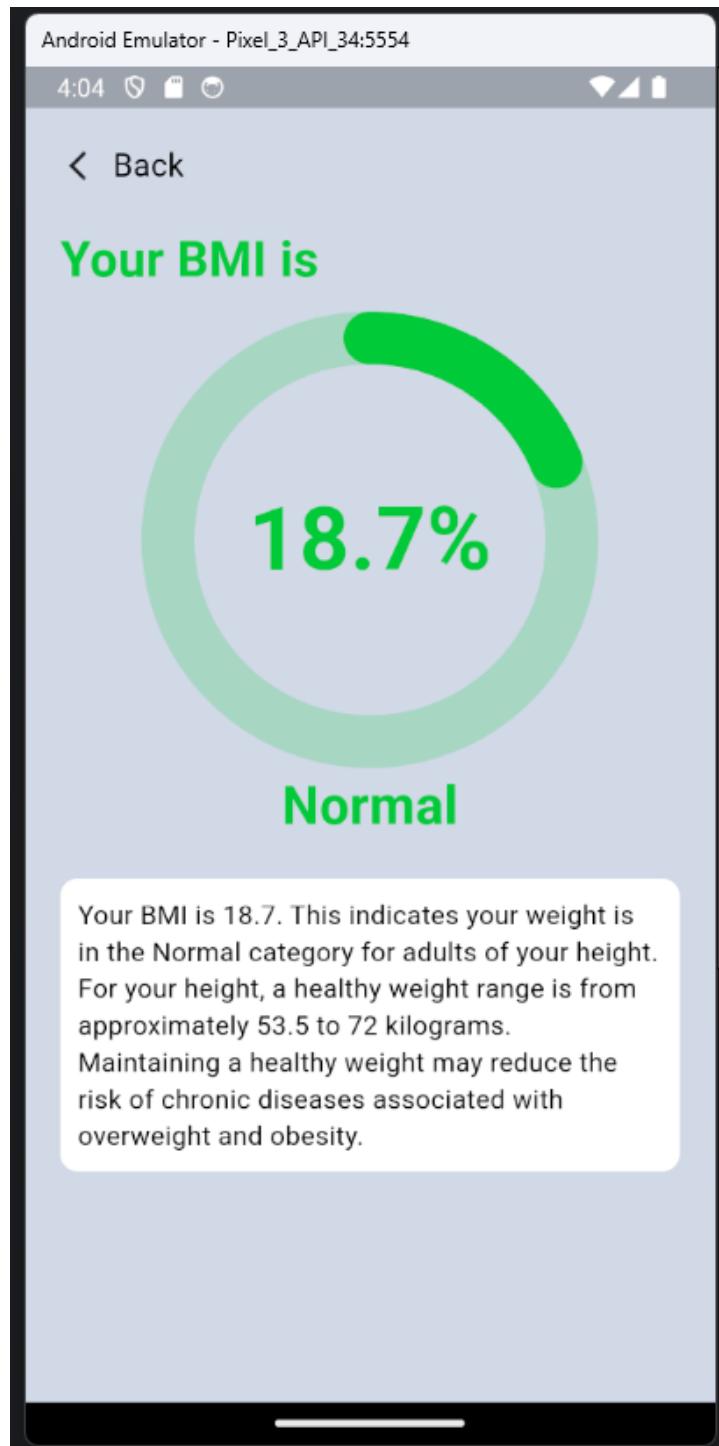
Hình 3. 28. Xây dựng giao diện đảm bảo code tối giản và dễ hiểu

Các thành phần Widget được đánh số là các thành phần như sau:

- 1. theme_change_button**
- 2. welcome_introduce_text**
- 3. select_gender**
- 4. height_selector**
- 5. weight_selector**
- 6. age_selector**
- 7. analyst_button**
- 8. select_gender_button**
- 9. stepper_button**

❖ result_screen

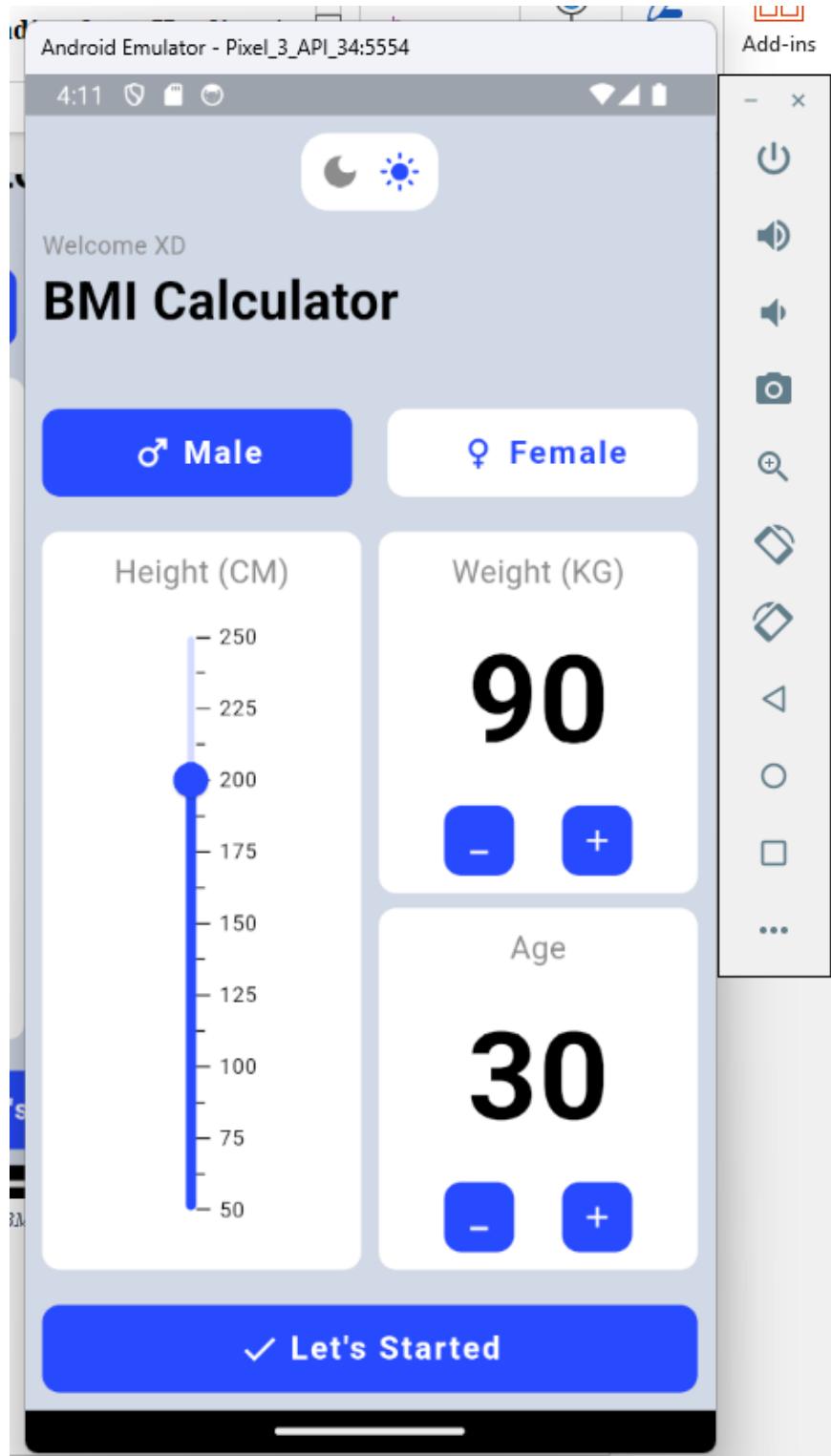
Màn hình này là duy nhất và đã tối giản nên không cần phải tối giản code nữa:



Hình 3. 29. Màn hình hiển thị kết quả của chương trình tính toán BMI

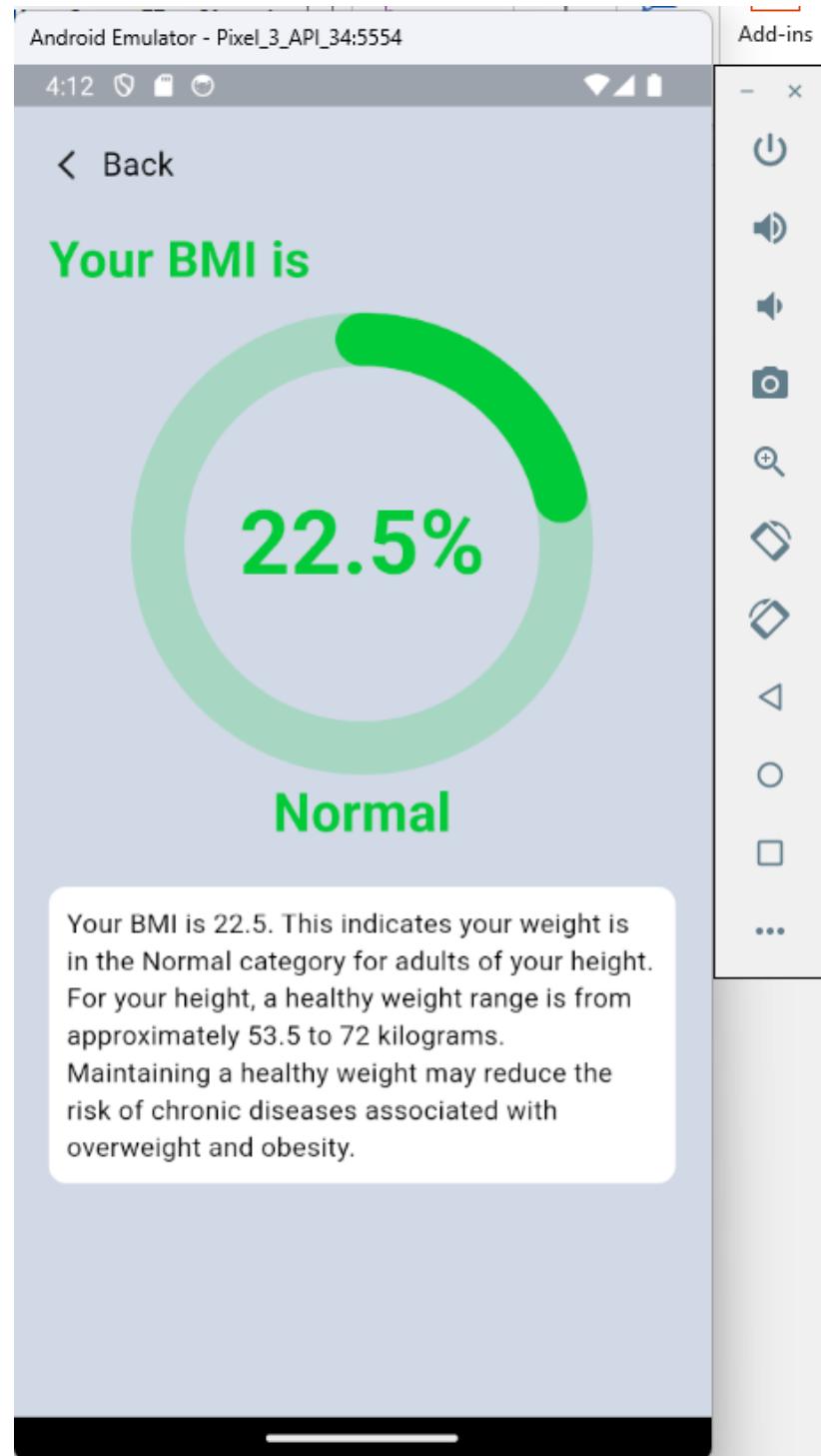
3.4.3. Demo chương trình

Trường hợp: Cân nặng 80kg, Chiều cao 2m, 30 tuổi và giới tính Nam:



Hình 3. 30. Demo chương trình BMI Calculator với Input nhu trên hình

Kết quả thu được là:



Hình 3. 31. Demo chương trình BMI Calculator với Output như trên hình

Kết luận: Khi so sánh với kết quả thực tế thì hoàn toàn chính xác

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Kết luận

Flutter đã và đang trở thành một trong những Framework phát triển ứng dụng di động ngày càng phổ biến với những ưu điểm nổi bật như sau:

- **Cấu trúc mã đa nền tảng:** Viết một lần, chạy trên nhiều nền tảng (Android, iOS, Web, Desktop) tiết kiệm thời gian và chi phí phát triển. Ví dụ như việc phát triển một ứng dụng Hybird đa nền tảng.
- **Giao diện người dùng mượt mà:** Cung cấp hiệu suất cao, khả năng tùy chỉnh linh hoạt, và trải nghiệm người dùng mượt mà.
- **Cộng đồng phát triển lớn:** Lượng người dùng và nhà phát triển Flutter đông đảo, nguồn tài liệu phong phú, hỗ trợ tốt. Một trong những trang web lớn hỗ trợ nhiều tiện ích và giao diện phong phú như **pub.dev**.
- **Công nghệ hiện đại:** Sử dụng Dart, ngôn ngữ lập trình hiện đại, dễ học, dễ sử dụng.

Như ta có thể thấy, Flutter đang là một công cụ phát triển app vô cùng tiềm năng, phù hợp với các doanh nghiệp đang muốn phát triển sản phẩm trên cả nền tảng Android và iOS. Tuy không phải lời giải cho mọi bài toán, nhưng trong tương lai Flutter hứa hẹn rất nhiều cơ hội trong lĩnh vực lập trình số, nhất là khi xét đến độ phủ trong cộng đồng và tốc độ cập nhật liên tục. Đây chắc chắn là một lựa chọn khó có thể bỏ qua với những ai muốn tối ưu app với trọng tâm nằm ở giao diện UI và hiệu năng cao.

Hướng phát triển

Hướng phát triển tiếp theo trong quá trình học Flutter của nhóm có thể là xây dựng những ứng dụng hoàn chỉnh và chỉnh chu hơn như việc thêm các bộ xử lý, tổ chức cấu trúc code chuyên nghiệp hơn, ứng dụng các Backend khác hoặc kết hợp Flutter với các ngôn ngữ lập trình khác để tăng tính tương tác và trải nghiệm người dùng. Bên cạnh đó là luôn luôn cập nhật và theo dõi các thay đổi mới nhất về Flutter.

DANH MỤC THAM KHẢO

- [1] Trang chủ Flutter - Flutter Docs
<https://docs.flutter.dev/>
- [2] Trang Web Học Lập Trình - CafeDev
<https://cafedev.vn/series-tu-hoc-flutter-tu-co-ban-toi-nang-cao/>
- [3] Youtube học lập trình của Flutter
<https://www.youtube.com/@flutterdev>
- [4] Youtube tự học Flutter – TinCoder
<https://www.youtube.com/@tincoder>
- [5] Youtube tự học Flutter – DearProgrammer
<https://www.youtube.com/@DearProgrammer>