**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG THƯƠNG TP.HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**

**---o0o---**



**BÁO CÁO THUYẾT TRÌNH:**

**CÔNG NGHỆ PHẦN MỀM**

**ĐỀ TÀI: TÌM HIỂU CÔNG NGHỆ FLUTTER**

**LẬP TRÌNH DI ĐỘNG**

**Nhóm 8**

**Thành phố Hồ Chí Minh, tháng 09 năm 2024**

**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG THƯƠNG TP.HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**

**---o0o---**



**BÁO CÁO THUYẾT TRÌNH:**

**CÔNG NGHỆ PHẦN MỀM**

**ĐỀ TÀI: TÌM HIỂU CÔNG NGHỆ FLUTTER**

**LẬP TRÌNH DI ĐỘNG**

**Nhóm 8:**

Trưởng nhóm: Nguyễn Văn Thành

Thành viên:

1. Lăng Minh Hải
2. Trần Trọng Quí
3. Nguyễn Minh Sang
4. Nguyễn Hoành Thịnh

**Thành phố Hồ Chí Minh, tháng 09 năm 2024**

**BẢNG PHÂN CHIA NHIỆM VỤ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **MSSV** | **HỌ VÀ TÊN** | **NHIỆM VỤ**  **PHÂN CÔNG** | **PHẦN TRĂM ĐÓNG GÓP** |
| 1 | 2001221146 | Lăng Minh Hải | Ngôn ngữ Dart, điều hướng | 20% |
| 2 | 2001223968 | Trần Trọng Quí | Giới thiệu, Widget | 20% |
| 3 | 2001224160 | Nguyễn Minh Sang | Ngôn ngữ Dart | 20% |
| 4 | 2001224715 | Nguyễn Văn Thành | Mở đầu, Quản lý trạng thái, điều hướng, nhược điểm Flutter | 20% |
| 5 | 2001224936 | Nguyễn Hoành Thịnh | Widget | 20% |

**MỤC LỤC**

[MỞ ĐẦU 1](#_Toc178326151)

[Chương 1. Giới thiệu 2](#_Toc178326152)

[1.1. Flutter là gì? 2](#_Toc178326153)

[1.2. Lịch sử hình thành của Flutter 2](#_Toc178326154)

[1.3. Tầm quan trọng của Flutter 3](#_Toc178326155)

[1.4. Những phần mềm thực tế được phát triển bằng Flutter 3](#_Toc178326156)

[Chương 2. Ngôn ngữ Dart 5](#_Toc178326157)

[2.1. Những đặc điểm nổi bật của ngôn ngữ Dart: 5](#_Toc178326158)

[2.2. Ứng dụng của ngôn ngữ Dart: 7](#_Toc178326159)

[2.3. Cú pháp cơ bản của ngôn ngữ Dart 8](#_Toc178326160)

[Chương 3. Kiến trúc của Flutter 10](#_Toc178326161)

[3.1. Widget trong Flutter 10](#_Toc178326162)

[3.1.1. Widget là gì? 10](#_Toc178326163)

[3.1.2. Phân loại widget 10](#_Toc178326164)

[3.1.3. Các widgets cơ bản trong flutter 11](#_Toc178326165)

[3.2. Quản lý trạng thái trong Flutter 16](#_Toc178326166)

[3.2.1. setState 16](#_Toc178326167)

[3.2.2. Provider 17](#_Toc178326168)

[3.3. Điều hướng trong Flutter 19](#_Toc178326169)

[3.3.1. Navigator và MaterialPageRoute 19](#_Toc178326170)

[3.3.2. Sử dụng routes trong MaterialApp 21](#_Toc178326171)

[3.3.3. Một số thư viện hỗ trợ điều hướng 21](#_Toc178326172)

[3.4. Thách thức và hạn chế của Flutter 22](#_Toc178326173)

[KẾT LUẬN 23](#_Toc178326174)

[MỤC LỤC HÌNH ẢNH 24](#_Toc178326175)

[TÀI LIỆU THAM KHẢO 25](#_Toc178326176)

MỞ ĐẦU

Các thiết bị di động hiện nay là một phần không thể thiếu đối với đời sống của mỗi người. Song song với sự phát triển của các thiết bị phần cứng thì phần mềm, công nghệ cũng phải phát triển để tương thích và mang lại trải nghiệm tốt nhất cho người sử dụng. Công nghệ Flutter ra đời mang lại nhiều lợi ích cho nhà phát triển ứng dụng, lập trình viên xây dựng về giao diện người dùng.

Tình hình chung trước khi Google cho ra mắt framework Flutter vào năm 2018, trên thế giới lúc này sử dụng Native Development (Java, Kotlin, Objective-C, Swift), React Native, Xamarin,… Tuy nhiên, những công nghệ này thường gặp phải những hạn chế như phải quản lý nhiều codebase (phải viết code cho từng nền tảng thiết bị chạy ứng dụng), từ đó dẫn đến tốn nhiều chi phí cũng như là thời gian. Thêm vào đó xảy ra những vấn đề hiệu suất kém và sự bất đồng về UI giữa các nền tảng.

Flutter ra đời với mục đích khắc phục những yếu điểm của các công nghệ trước, cải thiện hiệu suất, render trực tiếp giao diện, tạo nên một giao diện nhất quán cho mọi nền tảng, giúp cho lập trình viên chỉ cần viết và quản lý một codebase duy nhất, điều này làm tối ưu chi phí và thời gian phát triển. Flutter sử dụng ngôn ngữ lập trình Dart với cú pháp dễ dàng tiếp cận và tương tự như JavaScript, C#, mang lại một hiệu suất cao và nhiều thư viện phong phú. Ngoài ra, Flutter còn hỗ trợ tính năng hot-reload giúp việc thử nghiệm và phát triển linh hoạt, nhanh chóng hơn. Về sau Flutter còn mở rộng thêm khả năng phát triển ứng dụng trên web và desktop, tạo nên một hệ sinh thái đa nền tảng mạnh mẽ.

1. Giới thiệu
   1. Flutter là gì?

Flutter là một framework phát triển ứng dụng di động được Google phát triển. Nó cho phép lập trình viên tạo ra các ứng dụng di động cho cả Android và iOS từ cùng một codebase (chỉ cần viết mã một lần và có thể chạy trên cả hai hệ điều hành). Flutter sử dụng ngôn ngữ lập trình Dart và cung cấp một bộ công cụ giao diện người dùng phong phú, giúp tạo ra các ứng dụng với giao diện đẹp mắt và hiệu suất cao.

Flutter có thể được xây dựng bằng bất cứ trình soạn thảo hay IDE nào. Được cài phổ biến trên Android Studio, Visual Studio Code hoặc IntelliJ.

* 1. Lịch sử hình thành của Flutter

**2015:** Bắt đầu dự án Sky: Google bắt đầu phát triển dự án "Sky" với mục tiêu cải thiện hiệu suất và trải nghiệm phát triển ứng dụng di động.

**2016:** Công bố Flutter: Google công bố Flutter tại hội nghị Google I/O 2017. Đây là lần đầu tiên Flutter được giới thiệu với cộng đồng.

**2017:** Flutter Alpha: Phiên bản đầu tiên của Flutter được phát hành, cung cấp các công cụ và API cơ bản để phát triển ứng dụng di động.

**2018:** Flutter 1.0: Phiên bản chính thức đầu tiên của Flutter được phát hành tại Flutter Live, đánh dấu sự chuyển mình từ giai đoạn thử nghiệm sang giai đoạn sản xuất.

**2019:** Flutter for Web và Desktop: Google công bố kế hoạch mở rộng Flutter để hỗ trợ phát triển ứng dụng web và desktop.

**2020:** Flutter 1.12: Phiên bản này bao gồm nhiều cải tiến về hiệu suất và các công cụ phát triển, đồng thời cộng đồng Flutter phát triển mạnh mẽ.

**2021:** Flutter 2.0: Phiên bản này đánh dấu sự mở rộng chính thức của Flutter để hỗ trợ phát triển ứng dụng web và desktop, cùng với các cải tiến về hiệu suất và tính năng.

**2022:** Flutter 3.0: Phiên bản tiếp tục mở rộng hỗ trợ các nền tảng như macOS và Linux, đồng thời cải thiện hiệu suất và trải nghiệm phát triển.

**Tương lai:** Flutter tiếp tục phát triển với các bản cập nhật định kỳ, cải thiện khả năng hỗ trợ các nền tảng mới, tăng cường các công cụ phát triển, và mở rộng cộng đồng.

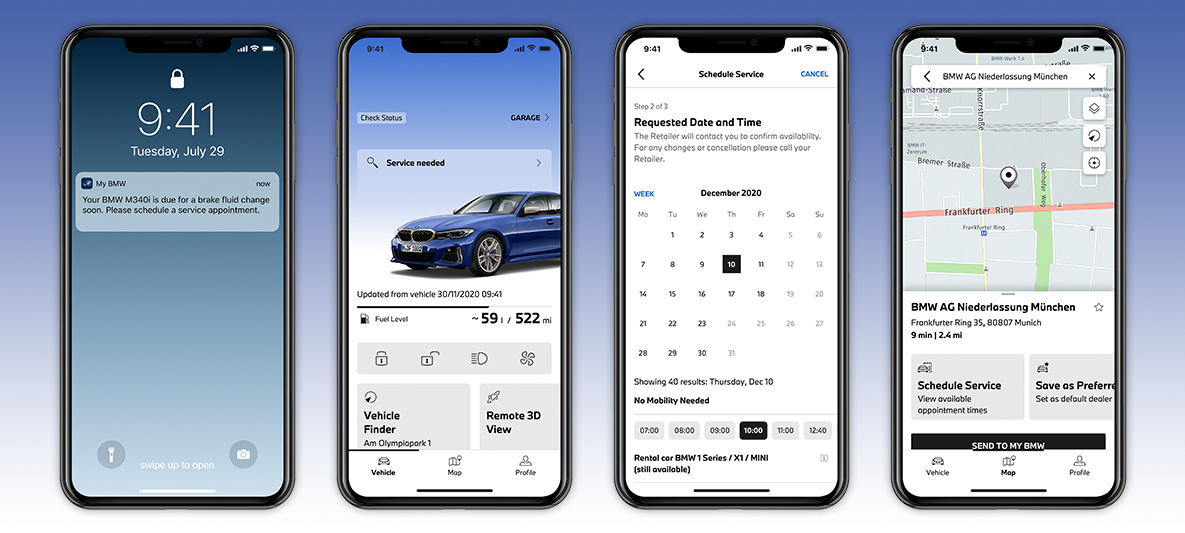
* 1. Tầm quan trọng của Flutter

Flutter giúp lập trình viên viết mã một lần và có thể chạy trên nhiều nền tảng khác nhau ví dụ di động (android và IOS), web, desktop. Điều này giảm thời gian, công sức và đảm bảo tính nhất quán của giao diện.

Flutter sử dụng ngôn ngữ lập trình Dart và cơ chế biên dịch Ahead-of-time(AOT) giúp ứng dụng chạy nhanh, mượt mà và tối ưu hiệu suất tốt trên các thiết bị di động

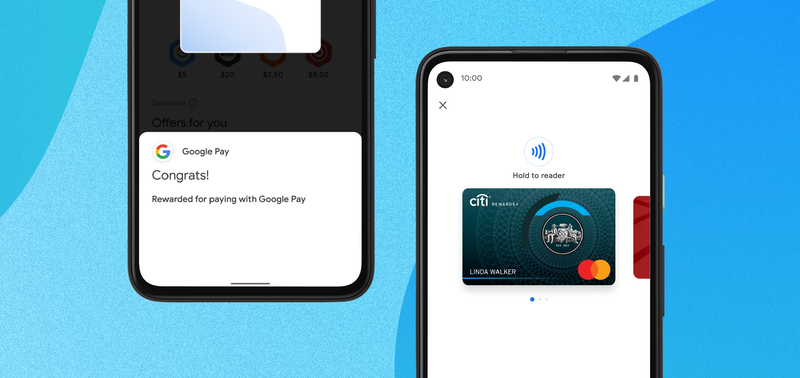
Công cụ giao diện widget phong phú, có khả năng tuỳ biến, giúp cho người lập trình tạo ra giao diện người dừng đẹp mắt hơn, mang lại trải nghiệm người dùng tốt hơn.

* 1. Những phần mềm thực tế được phát triển bằng Flutter



1. Ứng dụng sử dụng Flutter

**MyBMW** là ứng dụng cho phép người dùng điều khiển xe BMW từ xa, xem tình trạng xe, điều khiển điều hòa, khóa/mở cửa từ xa và nhiều tính năng khác.



2. Ứng dụng sử dụng Flutter

**Google Pay** là ứng dụng thanh toán di động chính thức của Google, cho phép người dùng thanh toán trực tuyến và tại cửa hàng bằng điện thoại di động.

1. Ngôn ngữ Dart

Dart, là một ngôn ngữ lập trình đa mục đích mã nguồn mở, đã được đặt nền móng bởi Google. Đây là một khía cạnh của ngôn ngữ lập trình hướng đối tượng, đặc trưng bởi cú pháp kiểu C. Tinh thần lập trình hướng đối tượng thể hiện qua việc hỗ trợ giao diện và lớp, mở ra khả năng sáng tạo không giới hạn khi đặt lên bàn cân với những ngôn ngữ khác. Sự đa dạng của Dart thể hiện thông qua khả năng phát triển ứng dụng web, di động, máy chủ và máy tính để bàn.

* 1. Những đặc điểm nổi bật của ngôn ngữ Dart:
     + 1. Cú pháp đơn giản và dễ học

Dart có cú pháp tương tự như các ngôn ngữ lập trình C-style khác như Java và JavaScript, giúp lập trình viên dễ dàng tiếp cận và bắt đầu sử dụng Dart nhanh chóng. Điều này giúp giảm thời gian học tập và cho phép các lập trình viên nhanh chóng bắt đầu xây dựng ứng dụng.

* + - 1. Hiệu suất cao

Dart được thiết kế để đạt hiệu suất cao khi chạy trên mọi thiết bị và nền tảng. Điều này đạt được nhờ vào khả năng biên dịch Ahead-Of-Time (AOT) của Dart, một phương pháp cho phép biên dịch mã nguồn thành mã máy gốc trước khi ứng dụng được khởi chạy, giúp cải thiện đáng kể tốc độ và hiệu suất của ứng dụng. Điều này đặc biệt quan trọng đối với các ứng dụng di động và ứng dụng yêu cầu hiệu suất cao.

* + - 1. Hỗ trợ lập trình hướng đối tượng (OOP)

Dart là một ngôn ngữ lập trình hướng đối tượng (OOP) với đầy đủ các đặc điểm như class-based, tính kế thừa và đa hình. Điều này giúp các lập trình viên tổ chức mã nguồn một cách rõ ràng và dễ quản lý. Lập trình hướng đối tượng cũng giúp tái sử dụng mã và phát triển các ứng dụng phức tạp hơn một cách hiệu quả.

* + - 1. Null Safety

Một trong những tính năng quan trọng và tiên tiến của Dart là null safety, giúp ngăn chặn các lỗi runtime liên quan đến null bằng cách đảm bảo rằng các biến không thể chứa giá trị null trừ khi được khai báo rõ ràng. Null safety không chỉ cải thiện độ an toàn của mã nguồn mà còn giúp các nhà phát triển tránh được những lỗi phổ biến và khó phát hiện. Điều này đặc biệt hữu ích trong các dự án lớn và phức tạp, nơi việc kiểm soát các giá trị null là rất quan trọng để đảm bảo chất lượng và hiệu suất của ứng dụng

* + - 1. Khả năng biên dịch nhanh

Dart hỗ trợ cả hai phương pháp biên dịch là Ahead-Of-Time (AOT) và Just-In-Time (JIT), giúp tối ưu hóa quy trình phát triển ứng dụng.

* + - 1. Phát triển đa nền tảng

Dart, kết hợp với Flutter, cho phép phát triển ứng dụng đa nền tảng từ một cơ sở mã nguồn duy nhất. Các nhà phát triển có thể xây dựng ứng dụng cho iOS, Android, web và các ứng dụng desktop mà không cần viết mã lập trình riêng biệt cho từng nền tảng.

* + - 1. Tính năng hot reload

Một trong những tính năng nổi bật và được yêu thích nhất của Dart là hot reload. Khi sử dụng Flutter, Dart hỗ trợ tính năng hot reload, cho phép lập trình viên ngay lập tức thấy các thay đổi trong mã nguồn mà không cần phải khởi động lại ứng dụng, tăng tốc quá trình phát triển và gỡ lỗi. Dart cũng tích hợp tốt với các IDE phổ biến như IntelliJ IDEA, Android Studio và Visual Studio Code, cung cấp các tiện ích như tự động hoàn thành mã, gỡ lỗi và phân tích mã nguồn.

* + - 1. Tích hợp tốt với các công cụ phát triển

Dart có khả năng tích hợp tốt với nhiều công cụ phát triển hiện đại ngày này như Visual Studio Code, Android Studio và IntelliJ IDEA. Điều này giúp các lập trình viên dễ dàng viết code, debug và triển khai mã nguồn một cách hiệu quả. Các công cụ hỗ trợ này cũng cung cấp các tính năng như gợi ý câu lệnh code, kiểm tra lỗi và quản lý dự án, giúp tăng cường hiệu suất làm việc

* 1. Ứng dụng của ngôn ngữ Dart:
     + 1. Phát triển ứng dụng di động

Dart trở nên đặc biệt nổi tiếng nhờ Flutter, framework UI do Google phát triển. Flutter cho phép các nhà phát triển xây dựng các ứng dụng di động đa nền tảng (cross-platform) với hiệu suất cao, giao diện người dùng đẹp mắt và trải nghiệm mượt mà. Bằng cách sử dụng một cơ sở mã nguồn duy nhất, các nhà phát triển có thể triển khai ứng dụng của mình trên cả Android và iOS, giúp tiết kiệm thời gian và công sức so với việc phải viết mã riêng cho từng nền tảng.

* + - 1. Phát triển ứng dụng web

Dart ban đầu được thiết kế để thay thế JavaScript trong phát triển web. Mặc dù không đạt được sự phổ biến như mong đợi ban đầu, Dart vẫn là một lựa chọn mạnh mẽ cho phát triển ứng dụng web. Với bộ công cụ như DartPad và các thư viện như AngularDart, các nhà phát triển có thể xây dựng các ứng dụng web tương tác và hiệu quả.

* + - 1. Phát triển ứng dụng desktop

Cùng với sự phát triển của Flutter, Dart không chỉ giới hạn ở các ứng dụng di động và web mà còn mở rộng sang phát triển ứng dụng máy tính để bàn. Flutter for Desktop cho phép các nhà phát triển tạo ra các ứng dụng chạy trên Windows, macOS và Linux từ cùng một cơ sở mã nguồn

* + - 1. Phát triển server-side

Dart không chỉ mạnh mẽ trong phát triển giao diện người dùng mà còn tỏ ra rất hiệu quả trong việc xây dựng các ứng dụng phía máy chủ. Thư viện dart:io và các gói mở rộng như Aqueduct giúp Dart xây dựng các ứng dụng máy chủ mạnh mẽ và hiệu quả. Dart có thể xử lý các yêu cầu HTTP, quản lý kết nối cơ sở dữ liệu và thực hiện các tác vụ phức tạp phía máy chủ, tương tự như Node.js nhưng tận dụng được các ưu điểm vượt trội của Dart.

* + - 1. Phát triển các công cụ

Dart còn được sử dụng để phát triển các công cụ và tiện ích hỗ trợ quá trình phát triển phần mềm. Ví dụ, các công cụ dòng lệnh (CLI) được viết bằng Dart có thể giúp tự động hóa các tác vụ, quản lý dự án và thực hiện các công việc khác một cách hiệu quả.

* 1. Cú pháp cơ bản của ngôn ngữ Dart
     + 1. Khai báo biến

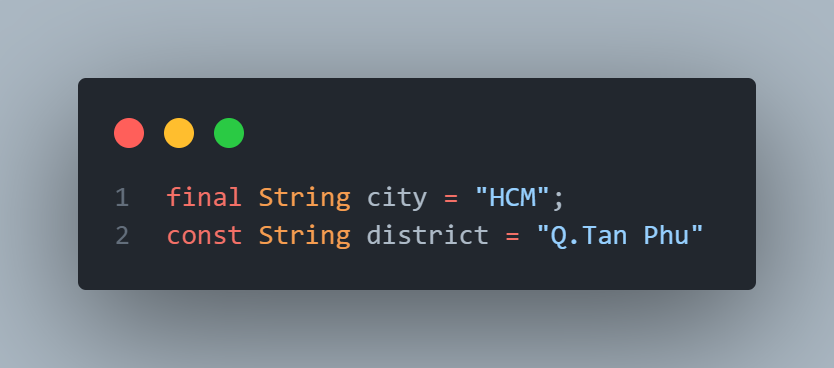
3. khai báo biến

*var* thường dùng khi không rõ kiểu dữ liệu, trình biên dịch sẽ tự động xác định dựa vào biến khởi tạo.

*Dynamic* kiểu dữ liệu này chấp nhận mọi kiểu dữ liệu được truyền vào.

Ngoài ra Dart vẫn có những kiểu dữ liệu như *String, int, double, bool, List, Set, Map, …*

* + - 1. Khai báo hằng số bằng

*final*: giá trị của biến chỉ được khởi tạo một lần.

4. khai báo hằng số

*const*: biến phải được khởi tạo tại thời điểm compile và không thể thay đổi.

* + - 1. Điều kiện và vòng lặp.
* Điều kiện if else và switch case tương tự như những ngôn ngữ khác.

Điều khiển câu lệnh theo điều kiện nhất định.

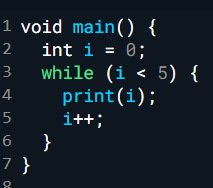
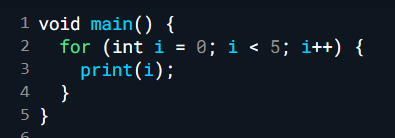
5. ví dụ về câu điều kiện



* Vòng lặp cũng sẽ tương tự như ngôn ngữ lập trình khác.

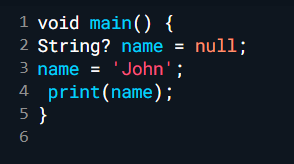
Sử dụng vòng lặp để lặp qua một khối lệnh nhiều lần.

6. ví dụ về vòng lặp



* + - 1. Null Safety

Dart hỗ trợ tính năng an toàn với null. Bạn cần khai báo kiểu dữ liệu nullable với dấu ? nếu biến có thể chứa giá trị null.



7. cách khai báo kiểu nullable

1. Kiến trúc của Flutter
   1. Widget trong Flutter
      1. Widget là gì?

Widget là một khối xây dựng cơ bản của giao diện người dùng trong ứng dụng Flutter. Trong Flutter, mọi thứ bạn thấy trên màn hình đều là widget, bao gồm văn bản, hình ảnh, nút bấm, bố cục và nhiều thứ khác. Các widget trong Flutter chịu trách nhiệm xác định cấu trúc, hành vi và giao diện của UI.

* + 1. Phân loại widget

**Widget Tree** là cấu trúc cây mà bạn xây dựng để mô tả giao diện người dùng của ứng dụng. Mỗi widget trong cây này định nghĩa một phần của giao diện, từ các phần tử đơn giản như Text và Image đến các cấu trúc phức tạp như Row, Column, và Stack.

Widget mô tả cấu hình cho một Element. Một widget là một mô tả bất biến của một phần trong giao diện người dùng.

**StatelessWidget** là một loại widget không yêu cầu trạng thái có thể thay đổi (mutable state).

**StatefulWidget** là một widget có trạng thái có thể thay đổi. Trạng thái (state) là thông tin có thể được đọc đồng bộ khi widget được xây dựng và có thể thay đổi trong suốt vòng đời của widget. Người triển khai widget có trách nhiệm đảm bảo rằng trạng thái được thông báo kịp thời khi có sự thay đổi, thông qua việc sử dụng setState.

**ProxyWidget** là một widget có sẵn một widget con, thay vì xây dựng một widget mới.

**RenderObjectWidget** cung cấp cấu hình cho RenderObjectElement, đối tượng này bao bọc RenderObject, thành phần đảm nhận việc render thực tế của ứng dụng.

**InheritedWidget** là một lớp cơ bản cho các widget giúp truyền thông tin một cách hiệu quả xuống cây widget. Để lấy phiên bản gần nhất của một loại inherited widget cụ thể từ ngữ cảnh xây dựng (build context), ta sử dụng BuildContext.dependOnInheritedWidgetOfExactType. Các widget kế thừa này, khi được tham chiếu theo cách này, sẽ làm cho đối tượng tiêu thụ (consumer) được xây dựng lại khi bản thân inherited widget thay đổi trạng thái.

**ParentDataWidget** là một lớp dành cho các widget cung cấp thông tin ParentData cho con của RenderObjectWidget. Điều này có thể được sử dụng để cung cấp cấu hình riêng cho từng widget con trong RenderObjectWidget có nhiều hơn một con. Ví dụ, Stack sử dụng widget Positioned để định vị từng widget con.

**LeafRenderObjectWidget** là lớp cha cho RenderObjectWidget, cấu hình các RenderObject không có widget con.

**SingleChildRenderObjectWidget** là lớp cha cho RenderObjectWidget, cấu hình các RenderObject chỉ có một slot cho widget con.

**MultiChildRenderObjectWidget** là lớp cha cho RenderObjectWidget, cấu hình các RenderObject có một danh sách các widget con.

* + 1. Các widgets cơ bản trong flutter
       1. *TEXT*

Là một đoạn văn bản với một kiểu duy nhất.

Widget Text hiển thị một chuỗi văn bản với một kiểu duy nhất. Chuỗi văn bản có thể được ngắt dòng thành nhiều dòng hoặc hiển thị trên cùng một dòng, tuỳ thuộc vào các ràng buộc bố cục.

Tham số style là tùy chọn. Nếu không được chỉ định, văn bản sẽ sử dụng kiểu từ DefaultTextStyle bao quanh gần nhất. Nếu thuộc tính inherit của TextStyle được đặt thành true (giá trị mặc định), kiểu được cung cấp sẽ được kết hợp với DefaultTextStyle bao quanh gần nhất. Hành vi kết hợp này rất hữu ích, chẳng hạn, để làm cho văn bản in đậm trong khi vẫn sử dụng phông chữ và kích thước mặc định.

Sử dụng hàm khởi tạo Text.rich, widget Text có thể hiển thị một đoạn văn với các đoạn văn bản (TextSpans) có kiểu khác nhau. Ví dụ sau đây hiển thị chuỗi "Hello beautiful world" với mỗi từ có kiểu dáng khác nhau.

8. ví dụ Text Widget



* + - 1. CONTAINER



9.Container

Container là một widget phổ biến trong Flutter, thường được sử dụng để chứa và sắp xếp các widget con khác. Nó cung cấp khả năng thiết kế và bố trí mạnh mẽ như căn chỉnh, kích thước, khoảng cách (margin), đệm (padding), nền (background), viền (border), và các hiệu ứng khác. Container hoạt động như một hộp bao quanh, nơi có thể kiểm soát không gian và hình dáng của các widget con bên trong nó.

Một số đặc điểm của Container:

• Padding: Khoảng cách bên trong giữa container và widget con.

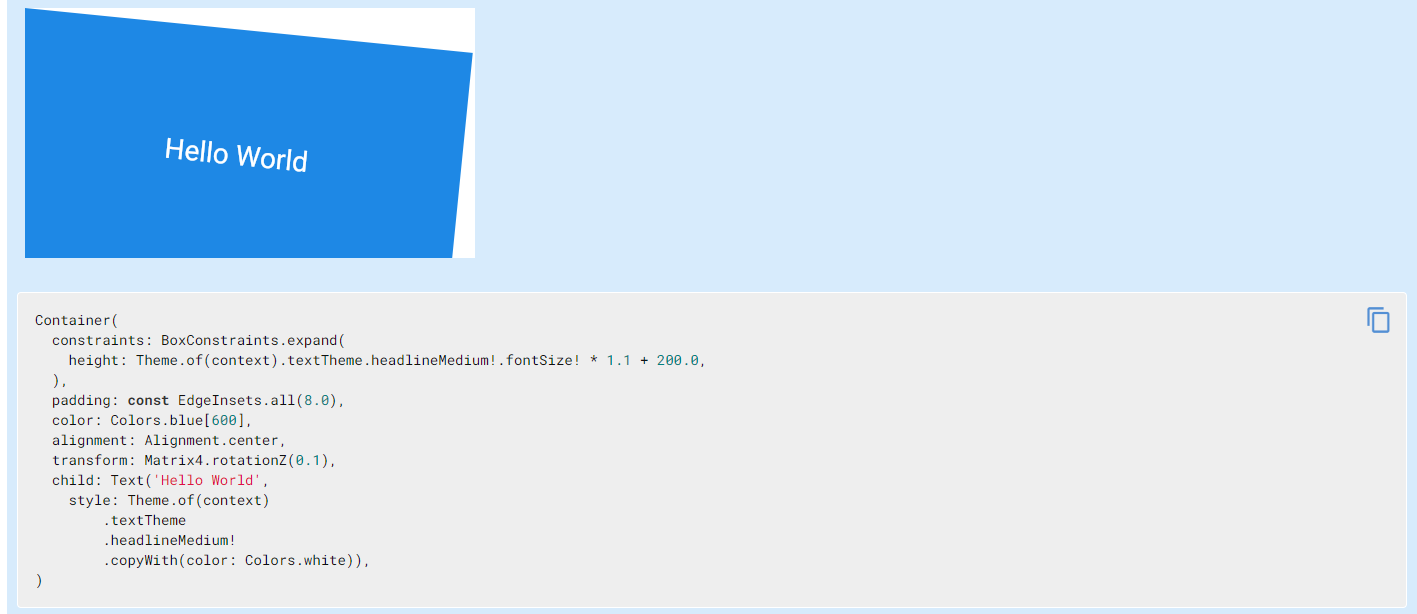
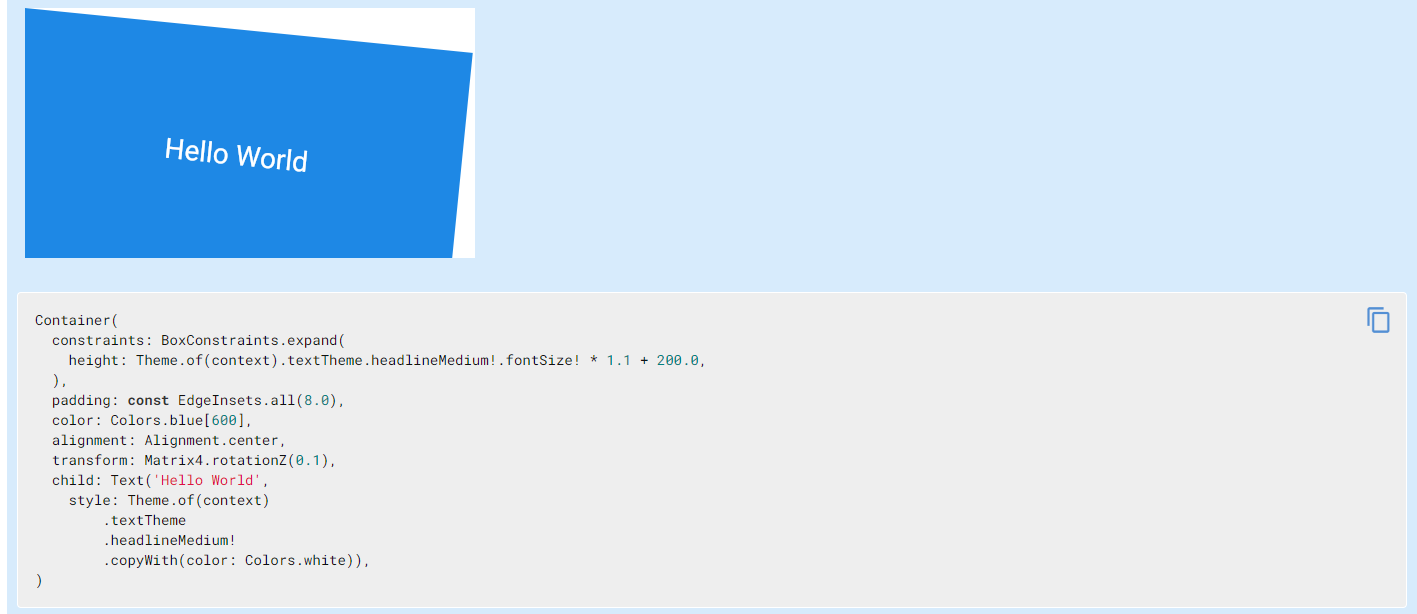
• Margin: Khoảng cách bên ngoài giữa container và các widget khác.

• Decoration: Cho phép đặt nền, màu sắc, viền, hình dạng...

• Alignment: Căn chỉnh widget con bên trong container.

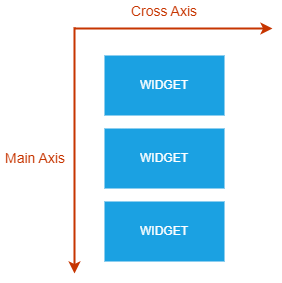
• Constraints: Đặt giới hạn kích thước cho container (chiều rộng và chiều cao tối thiểu và tối đa).

• Ví dụ này cho thấy cách sử dụng nhiều tính năng của Container cùng một lúc. Các ràng buộc (constraints) được đặt để phù hợp với kích thước font chữ cùng với khoảng cách dư thừa theo chiều dọc, trong khi mở rộng theo chiều ngang để phù hợp với widget cha. Padding được sử dụng để đảm bảo có khoảng trống giữa nội dung và văn bản. Color làm cho hộp có màu xanh dương. Alignment căn chỉnh widget con nằm ở giữa hộp. Cuối cùng, transform áp dụng một phép xoay nhẹ cho toàn bộ widget để tạo ra hiệu ứng hoàn chỉnh.



10. ví dụ về Container Widget

* + - 1. COLUMN, ROW



11. Column, Row

Đây là 2 widget giúp định hình 1 tập hợp các widget theo chiều dọc (vertically) hoặc chiều ngang (horizontally). 2 thuộc tính quan trọng nhất của 2 widget này là *mainAxisAlignment* và *crossAxisAlignment*.

Khi bạn sử dụng một Row, các widget con của nó được sắp xếp thành một hàng, theo chiều ngang. Vì vậy, trục chính của Row là nằm ngang. Sử dụng *mainAxisAlignment* trong Row cho phép bạn căn chỉnh các widget con của hàng theo chiều ngang (ví dụ: left, right). Trục chéo đến trục chính của Row là trục dọc. Vì vậy, sử dụng *crossAxisAlignment* trong Row cho phép bạn xác định cách con của nó được căn chỉnh theo chiều dọc. Trong một Cột thì ngược lại. Con của một Column được sắp xếp theo chiều dọc, từ trên xuống dưới (theo mặc định). Vì vậy trục chính của nó là thẳng đứng. Điều này có nghĩa là, việc sử dụng *mainAxisAlignment* trong một Column sẽ căn chỉnh các widget con của nó theo chiều dọc (ví dụ: top, bottom) và *crossAxisAlignment* xác định cách các con được căn chỉnh theo chiều ngang trong Column đó.

* + - 1. STACK

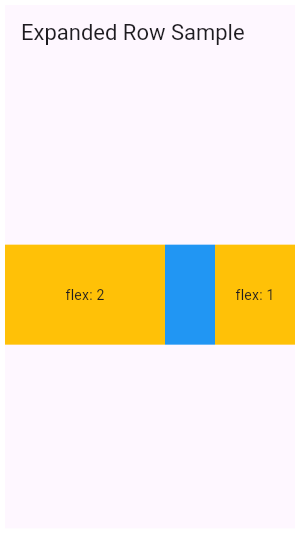


12. Stacck Widget

Cũng giống như Column hay Row, Stack cũng là 1 tập hợp các widget, tuy nhiên điểm khác biệt là các widget con trong stack có thể chồng lên nhau.

Thường đi kèm với widget này là widget *Positioned* để căn chỉnh vị trí của từng widget trong Stack.

* + - 1. EXPAND



13. Expand Widget

Trong Flutter, widget Expanded là một phần quan trọng của hệ thống bố cục (layout) và thường được sử dụng để quản lý không gian trong các cấu trúc bố cục như Column, Row, và Flex. Widget Expanded giúp phân bổ không gian còn lại trong một bố cục theo tỷ lệ hoặc kích thước cụ thể.

Widget Expanded tự động chiếm không gian còn lại trong hướng chính của Column, Row, hoặc Flex. Điều này có nghĩa là nó sẽ mở rộng để lấp đầy không gian còn lại trong widget chứa nó.

Expanded nhận một thuộc tính flex để chỉ định tỷ lệ không gian mà nó chiếm so với các widget Expanded khác. Ví dụ, nếu bạn có hai widget Expanded với flex là 1 và 2, widget thứ hai sẽ chiếm gấp đôi không gian so với widget đầu tiên.

* 1. Quản lý trạng thái trong Flutter

Quản lý trạng thái giúp kiểm soát và cập nhật lại giao diện thi thay đổi dữ liệu. Trong Flutter có những cách quản lý cơ bản là setState, Provider,…

* + 1. setState

Là một phương thức được sử dụng để xây lại giao diện người dùng khi trạng thái của một StatefulWidget thay đổi.

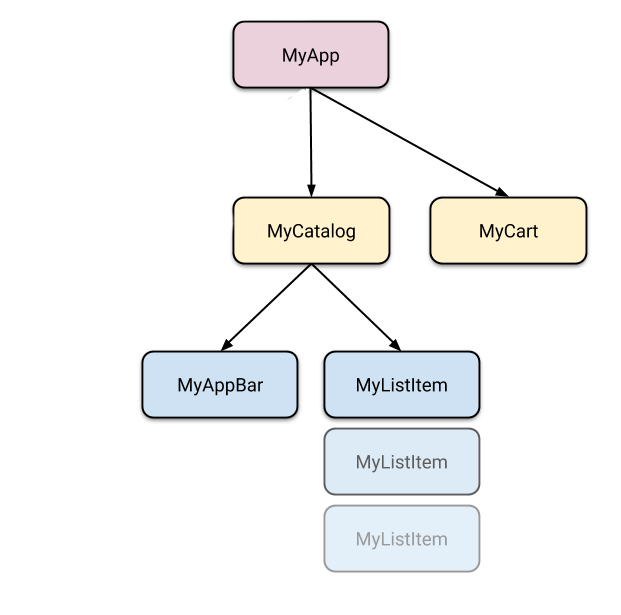
|  |
| --- |
| ElevatedButton(  child: Text("Đổi màu"),  onPressed: (){  // Thay đổi trạng thái UI với các thành phần liên quan đến changedColor  setState(() {  changedColor = ! changedColor;  });  },  ), |

Theo như đoạn code demo, ElevatedButton là một nút bấm với chức năng đổi màu, sau khi lắng nghe được sự kiện onPress hàm setState sẽ được gọi và thực thi cập nhật lại toàn bộ giao diện.

Tránh gọi **setState** quá nhiều trong vòng lặp hoặc liên tục, vì điều này có thể dẫn đến hiệu suất kém và gây nhấp nháy hoặc lag trong giao diện người dùng.

* + 1. Provider

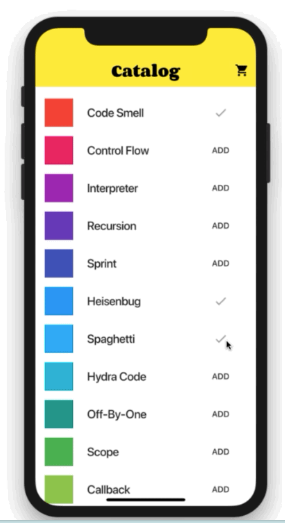
Một trong những vấn đề khi lập trình ứng dụng với Flutter hay ngay cả với React Native đó chính là quản lý state. Mỗi khi state thay đổi thì một số thành phần trên giao diện sẽ thay đổi như vậy ảnh hưởng đến hiệu năng của ứng dụng. Provider sẽ là công cụ giải quyết vấn đề này.

Cho phép các widget chia sẻ và cập nhật trạng thái mà không cần phải truyền dữ liệu qua nhiều cấp độ trong cây widget.

14. Tree provider

Ví dụ như hình bên khi chúng ta muốn cập nhật lại trạng thái của MyCart, các widget con (MyAppBar, MyListItem) sẽ tự động cập nhật lại để phản ánh trạng thái mới.

15. ví dụ ccho provider



Sau khi thêm sản phẩm vào giỏ hàng trạng thái của giỏ hàng được cập nhật lại đồng thời những sản phẩm trong danh sách sản phẩm cũng cập nhật nó để biết rằng nó đã được thêm vào giỏ hàng.

A diagram of a company

Description automatically generated

16. mô hình quản lý trạng thái Provider

Provider tương tự như một chuỗi cung ứng:

Nhà cung cấp sẽ kế thừa lớp ChangeNotifier, mỗi khi có sự thay đổi thì nhà cung cấp sẽ gọi hàm notifyListeners() để chuyển xuống nhà phân phối.

Nhà phân phối sẽ phân phối đến cho người thông qua ChangeNotifierProvider (chuyển 1 kênh phân phối) và MultiProvider (chuyển nhiều kênh phân phối).

Người sử dụng sẽ dùng context.watch<>() và context.read<>()

* **read():** chỉ truy cập trạng thái mà không theo dõi sự thay đổi, giúp tránh việc xây dựng lại không cần thiết.
* **watch():** theo dõi sự thay đổi của trạng thái và xây dựng lại widget khi trạng thái thay đổi.

Tóm lại:

**setState:** Đơn giản, nhanh chóng và thích hợp cho các ứng dụng nhỏ hoặc các trường hợp thay đổi trạng thái cục bộ trong một widget.

**Provider:** Mạnh mẽ hơn, phù hợp cho các ứng dụng phức tạp hoặc khi cần chia sẻ trạng thái giữa nhiều phần của ứng dụng.

Ngoài ra cũng sẽ có những phương pháp cải thiện từ 2 phương pháp cơ bản trên như Riverpod, Bloc,… những phương pháp cũng sẽ có ưu và nhược điểm riêng.

* 1. Điều hướng trong Flutter

Điều hướng là một phần quan trọng để chuyển đổi giữa các trang trong ứng dụng, sử dụng *Navigator* và *MaterialPageRoute* để thực hiện chuyển trang*.*

* + 1. Navigator và MaterialPageRoute

*Navigator* hoạt động như một stack (ngăn xếp) lưu trữ các trang (hoặc màn hình). Khi một trang mới được thêm vào stack, nó được hiển thị lên trên trang hiện tại. Khi một trang bị loại bỏ khỏi stack, ứng dụng quay trở lại trang trước đó.

push(): Đẩy (add) một trang mới lên stack, chuyển sang màn hình mới.

pop(): Lấy (remove) trang hiện tại ra khỏi stack, quay về màn hình trước.

*MaterialPageRoute* là một lớp trong Flutter giúp định nghĩa cách một trang mới sẽ được hiển thị. Nó cung cấp hiệu ứng chuyển cảnh Material Design mặc định, như hiệu ứng trượt từ phải sang trái (khi chuyển trang) hoặc từ dưới lên trên (nếu sử dụng trên thiết bị di động).

Cú pháp: chuyển hướng trang từ FirstPage sang SecondPage.

A computer screen shot of a black rectangle with yellow and orange text

Description automatically generated

17. Điều hướng Navigator Push

Sử dụng phương thức *push* trên đối tượng Navigator để đẩy widget mới của SecondPage vào stack, nó sẽ hiện đè lên trên trang trước đó.

*MaterialPageRoute* định nghĩa cách chuyển cảnh hiển thị trang mới.

* Với **maintainState**: Nếu đặt là true (mặc định), trạng thái của trang trước đó sẽ được giữ lại khi quay về. Nếu đặt là false, trạng thái của trang trước sẽ bị huỷ khi bạn chuyển đi.

Cú pháp: quay lại trang FirstPage

A black rectangular with yellow green and orange text

Description automatically generated

18. Điều hướng Navigator Pop

Sử dụng phương thức *pop* để xoá đi widget của SecondPage => Hiện lại FirstPage.

Ngoài việc sử dụng cách truyền thống, Flutter còn hỗ trợ mạnh mẽ nhiều thư viện có thể chuyển trang một cách dễ dàng như Getx, GoRouter,… Tuỳ thuộc vào quy mô, yêu cầu của sản phẩm mà người lập trình sẽ chọn phương pháp điều hướng hợp lý và tối ưu.

**Navigator.pushReplacement()**

Thay thế trang hiện tại bằng trang mới và không thể quay lại trang cũ.

A black screen with white text

Description automatically generated

19. Navigator PushReplacemnet

**Navigator.pushNamed()**

Điều hướng với các tên đường dẫn định danh (routes), thường dùng cho điều hướng dựa trên tên.



20. Navigator pushName

* + 1. Sử dụng routes trong MaterialApp

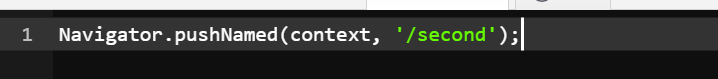
Flutter cho phép bạn định nghĩa các route trong MaterialApp để quản lý điều hướng dễ dàng hơn.

A screenshot of a computer program

Description automatically generated

21. MaterialApp

Khi đó, bạn có thể điều hướng giữa các trang bằng cách sử dụng tên route.



22. navigator pushName

* + 1. Một số thư viện hỗ trợ điều hướng

**go\_router** là một thư viện điều hướng mạnh mẽ và dễ sử dụng, đặc biệt nổi bật với khả năng điều hướng linh hoạt dựa trên URL, phù hợp cho các ứng dụng đa nền tảng như di động và web. Thư viện này giúp quản lý các tuyến (route) và trạng thái trong ứng dụng một cách rõ ràng và dễ hiểu, đồng thời hỗ trợ điều hướng dựa trên URL, rất hữu ích khi phát triển ứng dụng Flutter cho web.

**Navigator 2.0** là hệ thống điều hướng mặc định của Flutter, hỗ trợ cả điều hướng động và phân trang. Linh hoạt, mạnh mẽ, có thể quản lý điều hướng phức tạp bằng cách sử dụng các route với trạng thái rõ ràng. Tuy nhiên cấu hình phức tạp, khó sử dụng cho những ứng dụng đơn giản.

**GetX** thư viện quản lý trạng thái tích hợp điều hướng. Điều hướng trong GetX dễ dàng và không cần cấu hình route phức tạp.

* 1. Thách thức và hạn chế của Flutter
     + 1. Các vấn đề về hiệu suất trên các thiết bị cũ

Mặc dù Flutter mang lại hiệu suất cao, nhưng trên các thiết bị di động cũ, đặc biệt là những thiết bị có cấu hình thấp, hiệu suất của ứng dụng Flutter có thể không tối ưu. Điều này là do việc Flutter sử dụng nhiều tài nguyên bộ nhớ và CPU để xử lý đồ họa và animation mượt mà. Do đó, người dùng trên các thiết bị đời cũ có thể gặp hiện tượng giật, lag khi sử dụng ứng dụng.

* + - 1. Kích thước ứng dụng lớn

Một hạn chế lớn của Flutter là kích thước ứng dụng khởi điểm tương đối lớn, ngay cả với những ứng dụng đơn giản. Điều này xuất phát từ việc Flutter cần tích hợp đầy đủ engine và thư viện để đảm bảo ứng dụng chạy trên nhiều nền tảng khác nhau. Đối với các ứng dụng nhỏ hoặc yêu cầu kích thước nhẹ.

* + - 1. Khả năng mở rộng cho các dự án lớn

Mặc dù Flutter rất phù hợp cho các dự án nhỏ và trung bình, nhưng khi mở rộng lên các dự án lớn, việc quản lý code và bảo trì có thể trở nên phức tạp. Điều này có thể xảy ra nếu kiến trúc của dự án không được thiết kế tốt từ đầu.

KẾT LUẬN

**Flutter** là một framework phát triển ứng dụng di động hiện đại và mạnh mẽ, mang lại nhiều lợi ích vượt trội. Với Flutter, các nhà phát triển có thể xây dựng ứng dụng đa nền tảng với giao diện đẹp mắt, hiệu suất cao mà chỉ cần sử dụng một mã nguồn duy nhất. Điều này không chỉ giúp tiết kiệm thời gian và chi phí phát triển mà còn tạo điều kiện thuận lợi cho việc bảo trì và mở rộng ứng dụng sau này.

Một trong những yếu tố quan trọng khiến Flutter ngày càng trở nên phổ biến chính là sự hỗ trợ từ Google và cộng đồng phát triển rộng lớn, giúp Flutter liên tục được cập nhật và cải tiến. Flutter sử dụng ngôn ngữ Dart, cùng với các tính năng như hotreload và hệ thống widget phong phú, đã biến Flutter trở thành một công cụ linh hoạt, phù hợp cho cả các ứng dụng đơn giản lẫn phức tạp.

Những dự án thực tế được xây dựng bằng Flutter như Google Ads, BMW, eBay Motors, và Alibaba, chúng ta có thể thấy rõ sức mạnh và tiềm năng của Flutter trong việc đáp ứng nhu cầu của các doanh nghiệp lớn. Flutter không chỉ phù hợp cho việc phát triển ứng dụng di động mà còn đang mở rộng sang nhiều lĩnh vực khác như ứng dụng web và máy tính để bàn.

Tóm lại, Flutter là một lựa chọn lý tưởng cho các dự án phát triển ứng dụng di động và đa nền tảng trong tương lai. Hiệu suất, giao diện linh hoạt và khả năng mở rộng đã giúp Flutter trở thành một công nghệ đáng chú ý trong lĩnh vực phát triển phần mềm hiện nay.

MỤC LỤC HÌNH ẢNH

[1. Ứng dụng sử dụng Flutter 3](#_Toc178325853)

[2. Ứng dụng sử dụng Flutter 4](#_Toc178325854)

[3. khai báo biến 8](#_Toc178325855)

[4. khai báo hằng số 8](#_Toc178325856)

[5. ví dụ về câu điều kiện 8](#_Toc178325857)

[6. ví dụ về vòng lặp 9](#_Toc178325858)

[7. cách khai báo kiểu nullable 9](#_Toc178325859)

[8. ví dụ Text Widget 12](#_Toc178325860)

[9.Container 12](#_Toc178325861)

[10. ví dụ về Container Widget 13](#_Toc178325862)

[11. Column, Row 14](#_Toc178325863)

[12. Stacck Widget 15](#_Toc178325864)

[13. Expand Widget 15](#_Toc178325865)

[14. Tree provider 17](#_Toc178325866)

[15. ví dụ ccho provider 17](#_Toc178325867)

[16. mô hình quản lý trạng thái Provider 18](#_Toc178325868)

[17. Điều hướng Navigator Push 19](#_Toc178325869)

[18. Điều hướng Navigator Pop 20](#_Toc178325870)

[19. Navigator PushReplacemnet 20](#_Toc178325871)

[20. Navigator pushName 20](#_Toc178325872)

[21. MaterialApp 21](#_Toc178325873)

[22. navigator pushName 21](#_Toc178325874)

TÀI LIỆU THAM KHẢO

1. <https://viblo.asia/p/gioi-thieu-ve-flutter-bWrZnNxrZxw>
2. <https://docs.flutter.dev/ui>
3. <https://viblo.asia/p/gioi-thieu-ve-widgets-trong-flutter-Az45bzYQ5xY>
4. <https://acdacademy.vn/flutter-dieu-huong-cac-man-hinh-navigation-trong-flutter-phan-2/>