**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÀI BÁO CÁO CHỦ ĐỀ NGHIÊN CỨU**

**MÔN: CHUYÊN ĐỀ CÔNG NGHỆ PHẦN MỀM**

**ĐỀ TÀI:**

**TÌM HIỂU FLUTTER VÀ XÂY DỰNG ỨNG DỤNG MINH HỌA**

Sinh viên thực hiện:

**ĐOÀN LÊ MỸ LINH 5951071049**

**TÔ VÕ NHƯ QUỲNH 5951071089**

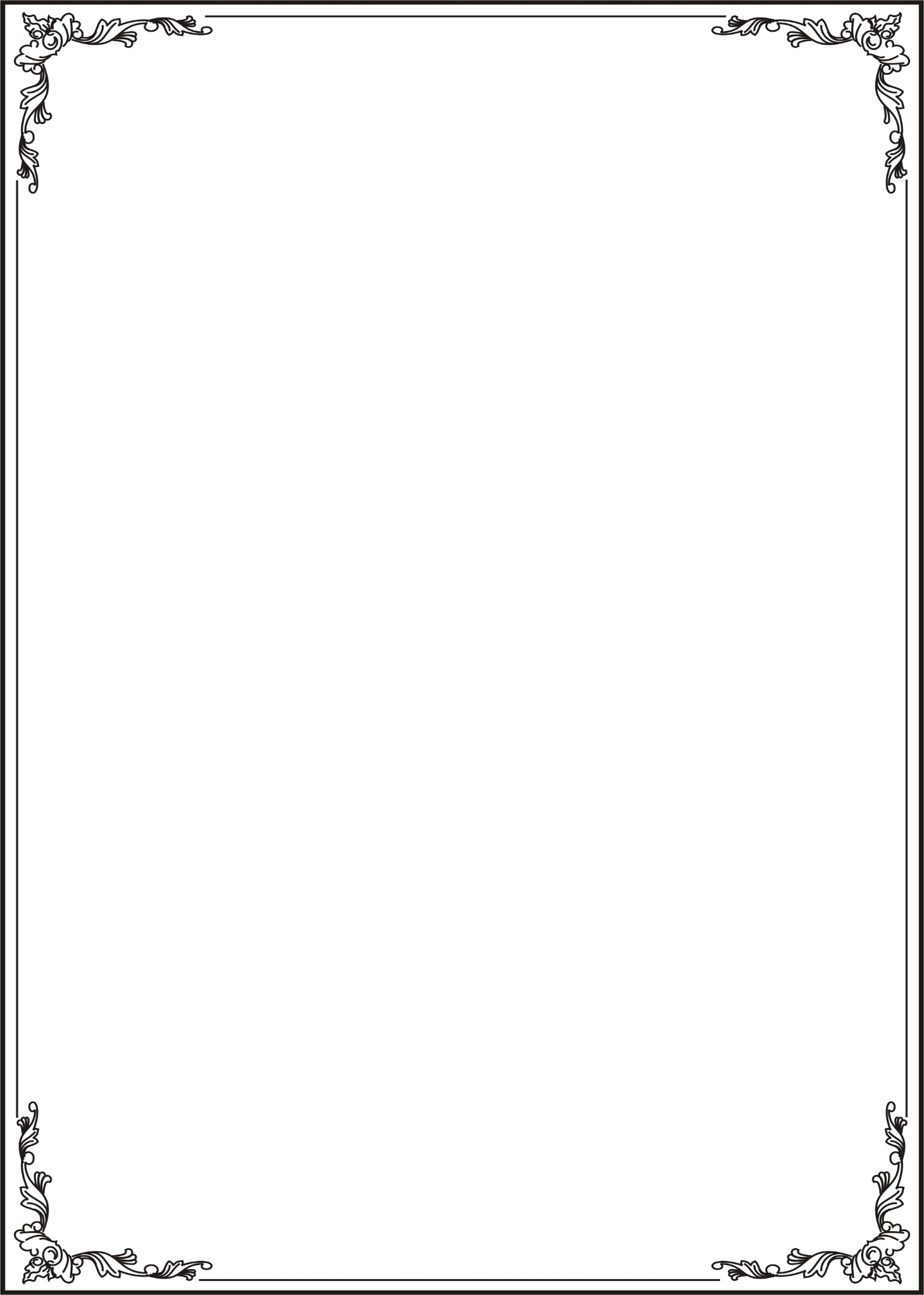
**TRẦN LÊ THANH TÍNH 5951071108**

**TRẦN DUY PHÚC 5951071077**

Lớp: CÔNG NGHỆ THÔNG TIN

Khóa: 59

Thành phố Hồ Chí Minh, ngày 4 tháng 10 năm 2021



**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO ĐỀ TÀI NGHIÊN CỨU**

**MÔN: CHUYÊN ĐỀ CÔNG NGHỆ PHẦN MỀM**

**ĐỀ TÀI:**

**TÌM HIỂU VỀ FLUTTER VÀ XÂY DỰNG ỨNG DỤNG MINH HỌA**

Sinh viên thực hiện:

**ĐOÀN LÊ MỸ LINH 5951071049**

**TÔ VÕ NHƯ QUỲNH 5951071089**

**TRẦN LÊ THANH TÍNH 5951071108**

**TRẦN DUY PHÚC 5951071077**

Lớp: CÔNG NGHỆ THÔNG TIN

Khóa: 59

Thành phố Hồ Chí Minh, ngày 4 tháng 7 năm 2021

# **LỜI MỞ ĐẦU**

Trong tình hình dịch bệnh Covid-19 đang diễn biến phức tạp, để đảm bảo an toàn cho sinh viên học tập năm học 2021 – 2022. Nhà trường đã tổ chức học kỳ online cho sinh viên toàn trường. Để không bị thụ động trong quá trình học online thì thầy cũng đã giao những hoạt động giúp sinh viên tương tác với nhau nhiều hơn.

Các nhóm do sinh viên tự chọn và đề tài là chọn trong các đề cho trước. Vì dịch bệnh nên sinh viên chúng em cũng đã di chuyển về quê để tránh dịch. Trong quá trình nhận đề tài cũng như triển khai làm bài do không thể gặp mặt trực tiếp và thời gian có phần gấp gáp nên sẽ không tránh khỏi những khúc mắc chưa được giải đáp hoàn toàn. Tuy nhiên mỗi thành viên trong nhóm đều nỗ lực hết sức để hoàn thành nhiệm vụ của mình tốt nhất để góp phần hoàn thành bài thi đầy đủ nhất.

**MỤC LỤC**

**[LỜI MỞ ĐẦU](#_Toc84431988)**

**[PHỤ LỤC HÌNH ẢNH](#_Toc84431989)**

**[NHẬN XÉT CỦA GIẢNG VIÊN](#_Toc84431990)**

**[PHẦN I. FLUTTER](#_Toc84431991)** [1](#_Toc84431991)

**[1.](#_Toc84431992)****[Tổng quan về Flutter:](#_Toc84431992)** [1](#_Toc84431992)

**[2.](#_Toc84431993)****[Cấu trúc trong Flutter](#_Toc84431993)** [3](#_Toc84431993)

[-](#_Toc84431994) ***[Cấu trúc của Flutter:](#_Toc84431994)*** [3](#_Toc84431994)

[-](#_Toc84431995) ***[Khám phá cấu trúc của flutter:](#_Toc84431995)*** [4](#_Toc84431995)

**[3.](#_Toc84431996)****[Layout trong Flutter](#_Toc84431996)** [9](#_Toc84431996)

**[4.](#_Toc84431997)****[Gesture trong Flutter:](#_Toc84431997)** [10](#_Toc84431997)

**[5.](#_Toc84431998)****[Quản lý trạng thái state trong Flutter:](#_Toc84431998)** [12](#_Toc84431998)

**[6.](#_Toc84431999)****[So sánh Flutter với Android (java)](#_Toc84431999)** [17](#_Toc84431999)

**[PHẦN II: CHƯƠNG TRÌNH DEMO](#_Toc84432000)** [18](#_Toc84432000)

**[1.](#_Toc84432001)****[Hình ảnh demo](#_Toc84432001)** [18](#_Toc84432001)

# **PHỤ LỤC HÌNH ẢNH**

[Hình 1 Cấu trúc của Flutter 3](#_Toc84431977)

[Hình 2 Cây Widget 4](file:///E:\\Semester%201,%2021-22\\Software_Technology\\Seminar\\BAO_CAO_nhom_so_2.docx" \l "_Toc84431978)

[Hình 3 Rendering 7](file:///E:\\Semester%201,%2021-22\\Software_Technology\\Seminar\\BAO_CAO_nhom_so_2.docx" \l "_Toc84431979)

[Hình 4 3 cây chính trong Flutter 8](file:///E:\\Semester%201,%2021-22\\Software_Technology\\Seminar\\BAO_CAO_nhom_so_2.docx" \l "_Toc84431980)

[Hình 5 InheritedWidget 15](#_Toc84431981)

[Hình 6 Giao diện login 18](file:///E:\\Semester%201,%2021-22\\Software_Technology\\Seminar\\BAO_CAO_nhom_so_2.docx" \l "_Toc84431982)

[Hình 7 Giao diện chính 19](file:///E:\\Semester%201,%2021-22\\Software_Technology\\Seminar\\BAO_CAO_nhom_so_2.docx" \l "_Toc84431983)

[Hình 8 Giao diện detail 19](file:///E:\\Semester%201,%2021-22\\Software_Technology\\Seminar\\BAO_CAO_nhom_so_2.docx" \l "_Toc84431984)

# **NHẬN XÉT CỦA GIẢNG VIÊN**

|  |
| --- |
| ***TP. Hồ Chí Minh, ngày tháng năm 2021***  **Giảng viên** |

# **PHẦN I. FLUTTER**

## **Tổng quan về Flutter:**

* **Flutter** là một **framework** mã nguồn mở cho phép tạo ứng dụng di động với hiệu năng cao, chất lượng tốt hỗ trợ đa nền tảng, phù hợp với phát triển ứng dụng Android và iOS.
* Flutter được phát triển bởi Google và giới thiệu lần đầu tiên vào năm 2018, sử dụng ngôn ngữ lập trình Dart.
* Flutter được gọi là một Fast Development vì có thể lập trình và sử dụng cho nhiều nền tảng khác nhau từ mobile, web, …
* ***Tại sao nên sử dụng Flutter?***
* Nhìn chung phát triển ứng dụng di động là một công việc phức tạp và nhiều khó khăn. Có rất nhiều framework hỗ trợ bạn phát triển một ứng dụng mobile. Android cung cấp một framework cơ bản dựa trên ngôn ngữ lập trình Java còn iOS thì cung cấp framework dựa trên Objective-C / Swift
* Tuy nhiên hầu hết các ứng dụng hiện nay, đều hỗ trợ cả 2 nền tảng Android và iOS, do đó cùng lúc phát triển 2 dự án khác nhau với 2 framework khác nhau là một công việc phức tạp và lãng phí thời gian công sức. Do đó người ta đã phát triển các framework lập trình đa nền tảng để giải quyết vấn đề này. Một framework rất phổ biến hiện nay là React Native được phát triển bởi Facebook đang được sử dụng rất rộng rãi. Tuy nhiên React Native vẫn thông qua các api của các framework gốc như Android hay iOS do đó bị hạn chế và tốc độ kém.
* Như một sự phát triển của tương lai, Flutter được phát triển bới chính Google, đơn vị sở hữu Android như một đối trọng trực tiếp với React Native. Thay vì gọi các api của framework gốc, Flutter tạo ra giao diện trực tiếp từ api của hệ điều hành. Nhờ đó ứng dụng sẽ chạy nhanh hơn, mượt mà hơn và đẹp hơn.
* Flutter cung cấp rất nhiều widgets (UI) là các thành phần đồ hoạ được thiết kế riêng. Những đối tượng đồ hoạ này được tối ưu phù hợp với môi trường mobile và dễ dàng trong việc thiết kế như HTML.
* Cụ thể, ứng dụng Flutter sẽ sử dụng các widget riêng. Flutter widgets cung cấp các animations (hiệu ứng) và gestures (thao tác) riêng. Ứng dụng được phát triển dựa trên logic của reactive programming. Mỗi Widget sẽ có rất nhiều trạng thái. Bằng cách thay đổi trạng thái của widget, Flutter sẽ tự động (reactive programming) so sánh trạng thái của widget (cũ và mới) để tạo ra những thay đổi cần thiết cho UI thay vì khởi tạo lại cả đối tượng.
* **Các tính năng của Flutter**

Flutter framework có những đặc điểm sau:

·   Hiện đã là một react framework.

·   Sử dụng ngôn ngữ lập trình Dart đơn giản và dễ học.

·   Phát triển ứng dụng nhanh chóng.

·   Giao diện người dùng đẹp và linh hoạt.

·   Hỗ trợ nhiều widget khác nhau.

·   Thể hiện cùng một UI trên nhiều nền tảng.

·   Tạo ra các ứng dụng có hiệu năng cao.

* **Điểm mạnh của Flutter**

Flutter đi kèm với nhiều widget đẹp và có độ tuỳ biến cao giúp phát triển ứng dụng hiệu năng cao vượt trội đáp ứng mọi nhu cầu và tuỳ biến. Bên cạnh đó Flutter còn có những điểm mạnh sau:

1.    *Dart* có một kho lớn các gói phần mềm cho phép bạn mở rộng khả năng cho ứng dụng của mình.

2.    Các lập trình viên chỉ cần viết một chương trình duy nhất cho tất cả các ứng dụng (Android và iOS). *Flutter* có thể mở rộng ra các nền tảng khác trong thời gian tới.

3.    Flutter dễ dàng kiểm thử hơn do tiết kiệm thời gian kiểm thử trên từng nền tảng.

4.    Nhờ sự đơn giản của mình, Flutter là lựa chọn hàng đầu cho các ứng dụng mới. Nó còn dễ dàng tuỳ biến và mở rộng lên càng mạnh mẽ hơn

5.    Với Flutter, lập trình viên có toàn quyền để sắp xếp bố trí điều khiển các widget

6.    Flutter có bộ công cụ phát triển (developer tools) rất hoàn thiện và đầy đủ, đặc biệt với tính năng hot reload đẩy nhanh tốc độ build ứng dụng đáng kinh ngạc

* **Tạo project Flutter và khởi chạy**

Sau khi cài đặt Flutter, SDK và android studio để khởi chạy máy ảo thì bạn gõ lệnh sau để tạo mới một flutter project:

          flutter create <TEN\_PROJECT>

Sau khi tạo project flutter thành công, để khởi chạy ứng dụng bạn cần phải mở máy ảo lên, sau đó gõ lệnh:

       flutter run

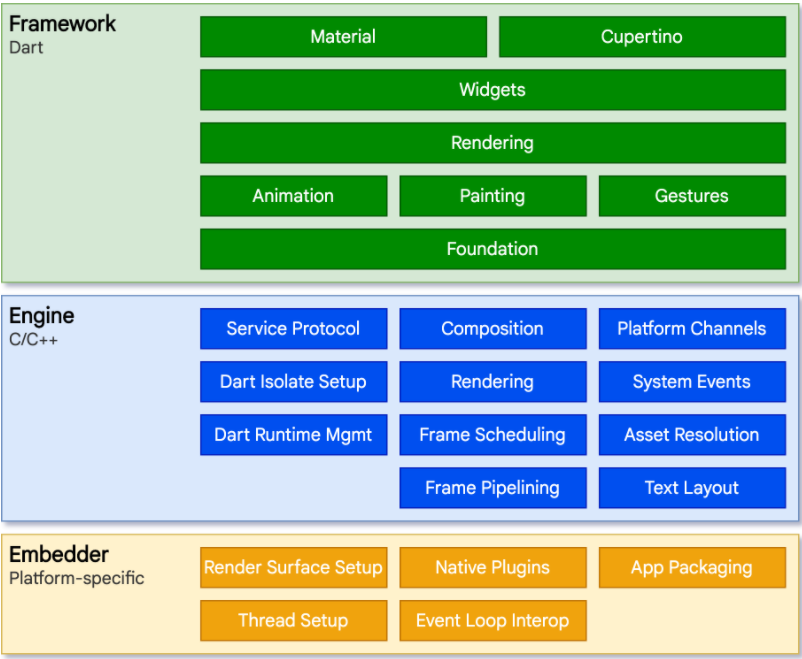
* **Ngôn ngữ Dart**

**Dart** là một ngôn ngữ lập trình mã nguồn mở (open source) đa năng (general purpose). Nó được phát triển bởi Google. **Dart** là một ngôn ngữ lập trình hướng đối tượng sử dụng cú pháp của C (C-style syntax). Nó hỗ trợ các khái niệm như interface, class, ... không giống như các ngôn ngữ lập trình khác, Dart không hỗ trợ mảng (array). Dart collections có thể sử dụng các cấu trúc dữ liệu (data structure) thay thế.

## **Cấu trúc trong Flutter**

### ***Cấu trúc của Flutter:***

Flutter được thiết kế như một hệ thống nhiều lớp, có thể mở rộng. Nó tồn tại dưới dạng một loạt các thư viện độc lập mà mỗi thư viện phụ thuộc vào lớp bên dưới. Không có lớp nào có đặc quyền truy cập vào các lớp bên dưới, mỗi phần framework được thiết kế để có thể tùy chọn và thay thế.

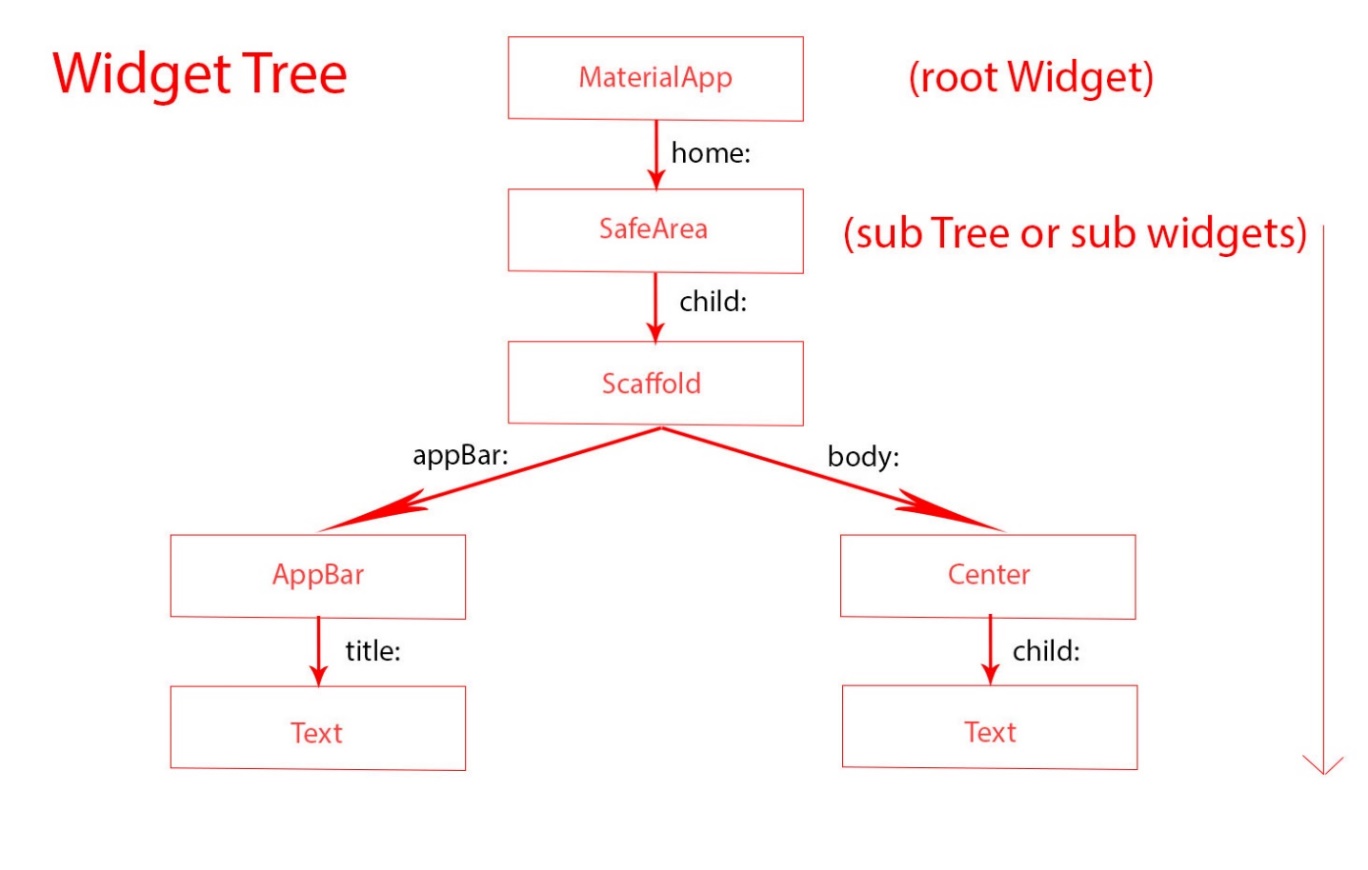


Hình 1 Cấu trúc của Flutter

* Trong flutter gồm có các thư viện và các lớp làm việc từ dưới lên trên như sau:
* Các lớp nền tảng cơ bản và xây dựng các dịch vụ khối như: animation, painting và gestures cung cấp các trừu tượng thường được sử dụng trên nền tảng cơ bản.
* Rendering layer: cung cấp sự trừu tượng để xử lý bố cục. Với lớp này ta có thể xây một dựng một cây render. Giúp thao tác với đối tượng một cách linh hoạt, cây render sẽ tự động update layout khi thay đổi.
* Widgets layer: là một thành phần trừu tượng. Mỗi đối tượng render trong Rendering layer có một layout tương ứng trong Widgets layer. Widgets layer cho phép người dùng xác định sự kết hợp giữa các layout có thể tái sử dụng.
* Các thư viện Material và Cupertino cung cấp các bộ điều khiển toàn diện sử dụng widget layer’composition để thực hiện các ngôn ngữ thiết kế Material hay iOS.

### ***Khám phá cấu trúc của flutter:***

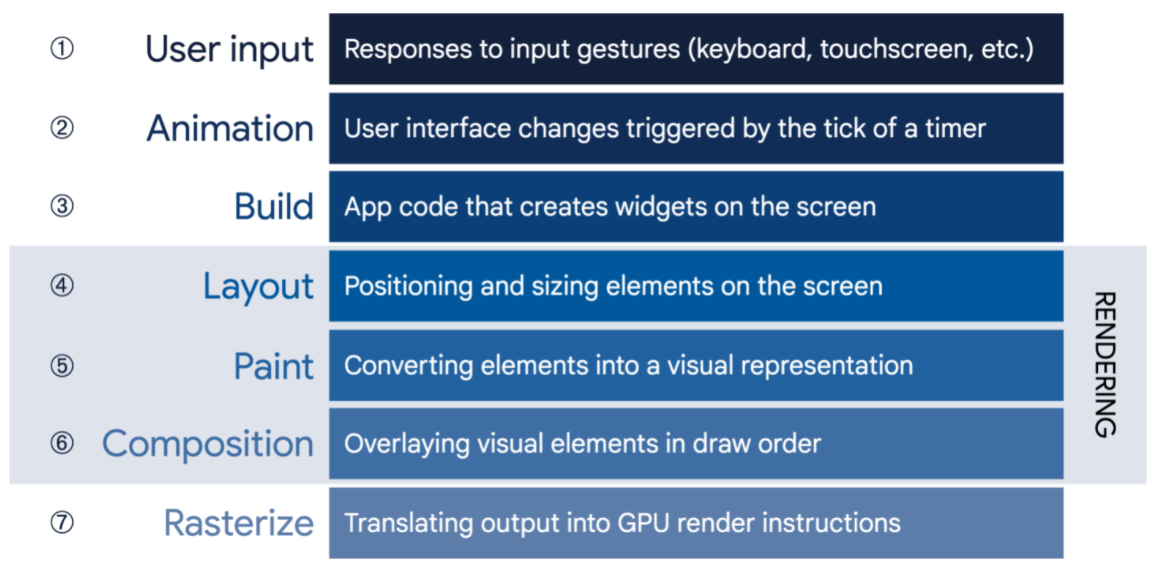
1. **Widgets:**

* Widget là các khối xây dựng giao diện người dùng của ứng dụng Flutter.
* Widget tạo ra một hệ thống phân cấp thành dạng một cây widget tree. Được ghép từ các widgets nhỏ hơn lại với nhau.

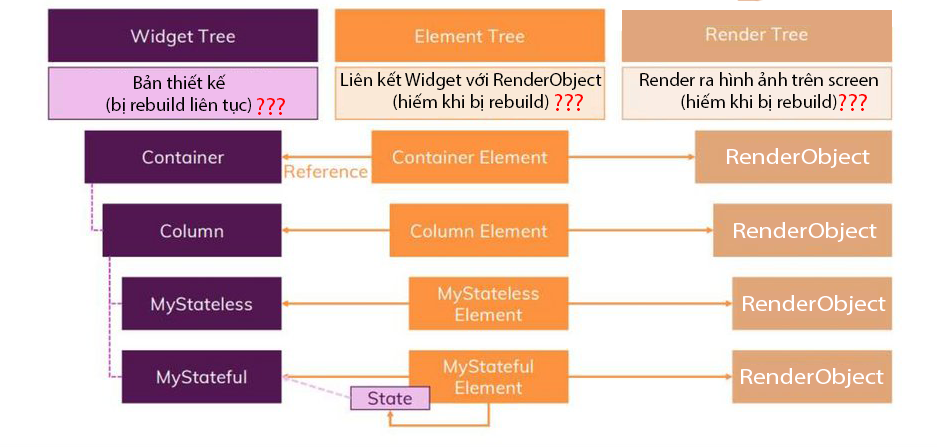
Hình 2 Cây Widget

* Trong mỗi widget có nhiều các thuộc tính như child, color,…. Nhằm tạo nên bản thiết kể UI hiển thị trên màn hình
* Có nhiều loại widget, nhưng được phân loại dựa trên chức năng thành 4 nhóm sau:
* **Platform widgets:** đây là các widget dành riêng cho từng nền tảng Android hay iOS.
* Các widget dành riêng cho Android được thiết kết theo *Material design guideline* cho Android OS nên được gọi là *Material widgets*.
* Các widget dành riêng cho iOS được thiết kế theo *Human Interface Guidelines* bởi Apple và được gọi là *Cupertino* widgets.
* Một số Material widgets phổ biến cho Android:
* Scaffold
* AppBar
* BottomNavigationBar
* TabBar
* TabBarView
* ListTile
* RaisedButton
* Icon button
* …
* Một số Material widgets phổ biến cho iOS:
* CupertinoButton
* CupertinoPicker
* CupertinoDatePicker
* CupertinoTimerPicker
* CupertinoNavigationBar
* CupertinoTabBar
* CupertinoTabScaffold
* CupertinoTabView
* CupertinoTextField
* **Layout widget:**
* Trong Flutter, một widget có thể được tạo thành từ một hoặc nhiều widget khác. Việc kết hợp nhiều widget thành một widget được thực hiện thông qua các layout widget. Ví dụ, các widget con có thể được căn giữa thông *Center* widget.
* Một số layout widgets phổ biến:
* **Container** − Một hình chữ nhật được thiết kế sử dụng
* **Center** − Căn giữa các widget con
* **Row** − Sắp xếp các widget con theo hàng ngang (horizontal direction).
* **Column** − Sắp xếp các widget con theo hàng dọc (vertical direction).
* **Stack** − Sắp xếp các widget con lên trên cùng
* **State maintenance widgets:**
* Trong **Flutter**, tất cả các widget đều kế thừa từ *StatelessWidget* hoặc *StatefulWidget*.
* Widget kế thừa từ *StatelessWidget* sẽ không có bất kì trạng thái nào nhưng nó có thể bao gồm widget được kết thừa từ *StatefulWidget*. Bản chất sự linh hoạt của ứng dụng là thông qua hành vi tương tác của các widget và sự thay đổi trạng thái của chúng.
* Ví dụ khi chạm vào một nút tăng giảm của bộ đếm, nó sẽ làm tăng hoặc giảm trạng thái của bộ đếm trong widget và cơ chế *reactive nature* (tự phản ứng) sẽ tự động thay đổi lại giao diện của widget theo trạng thái mới.
* **Platform independent / basic widgets**
* *Flutter* cung cấp một số lương lớn các widget cơ bản để tạo các giao diện người dùng từ đơn giản đến phứ tạp độc lập với nền tảng hệ điều hành. Chúng ta sẽ tìm hiểu một số widget cơ bản dưới đây:
* Text: *Text* widget được sử dụng để hiển thị một đoạn hay một chuỗi văn bản với một kiểu duy nhất. Chúng ta có thể định dạng văn bản thông qua thuộc tính *style* và*TextStyle* class.
* Image: *Image* widget được sử dụng để hiển thị hình ảnh trong ứng dụng. *Image* widget cung cấp các phương thức khởi tạo khác nhau để load hình ảnh từ các nguồn khác nhau:
* **Image** − Hình ảnh thông thường sử dụng *ImageProvider*
* **Image.asset** − Load hình ảnh từ flutter project’s assets
* **Image.file** − Load hình ảnh từ system folder
* **Image.memory** − Load hình ảnh từ memory
* **Image.Network** − Load hình ảnh từ mạng network
* Icon
* Icon widget hiển thị hình ảnh các icon cơ bản trong *IconData* class.

1. **Rendering**

* Nguyên tắc flutter áp dụng cho render là simple is fast.
* Ta có sơ đồ trình tự sau:

Hình 3 Rendering

* Xây dựng từ widget đến element:
* Element đại diện cho một instance của một widget tại một vị trí cụ thể trên hệ thống cây.
* Khi từng widget trên cây gọi hàm createElement() để tạo Element thì gọi là inflation. Từ đó ta có Element tree được tạo từ Widget tree.
* Có 2 class con của Element cơ bản là:
* ComponentElement: giống như là một tổ hợp (compose) nhiều Element, nó có khả năng tạo ra những RenderObjectElement hoặc những ComponentElement khác.
* RenderObjectElement: mỗi RenderObjectElement khi được gắn lên Element Tree sẽ nhờ widget mà nó đang nắm giữ gọi hàm createRenderObject() để tạo ra object renderObject và nó sẽ nắm giữ tham chiếu của renderObject này luôn.
* Từ đó suy ra một Element tree sẽ tạo ra được một Render Tree.

Hình 4 3 cây chính trong Flutter

* Trong khi các Widget liên tục bị rebuild, tức là bị destroy rồi build lại thì các Element chỉ tạo ra đúng 1 lần và nó chỉ **được update** chứ nó rất hiếm khi phải bị đập đi xây lại Element mới.
* Các thuật toán render hay khởi tạo một RenderObject được Flutter viết rất phức tạp, phức tạp hơn rất nhiều so với những Widget chúng ta code.
* Nên giữ các object RenderObject trong bộ nhớ càng lâu càng tốt thay vì cứ đập đi xây lại liên tục như widget.
* Nói cách khác, RenderObject và Element đắt giá hơn Widget nên hạn chế rebuild mà chỉ nên update.
* Mỗi lần rebuild, Flutter walk down the Element Tree, từng Element sẽ so sánh Widget cũ (cái mà nó đang nắm giữ trong biến widget) và Widget mới. Nếu nó thấy Widget cũ và Widget mới có **cùng Widget Type** nó sẽ **update** biến widget trỏ đến Widget mới đó. Ngược lại, tức là **khác Widget Type**, nó sẽ bị **rebuild**.

## **Layout trong Flutter**

* Trong **Flutter** các **layout** cũng là một loại widget, nhiệm vụ của chúng là bố trí các widget con, tạo nên giao diện người dùng cho ứng dụng. Flutter cung cấp nhiều loại layout khác nhau như *Container, Center, Align…*
* Có hai loại widget layout chính trong Flutter:
* Single Child Widgets - Chỉ có một widget con
* Multiple Child Widgets - Có nhiều widget con
* **Single Child Widgets**
* Các widget layout loại này chỉ có duy nhất một widget con và thường có chức năng bố trí nhất định.
* Ví dụ *Center* widget chỉ căn giữa widget con so với widget cha của nó và *Container* widget cung cấp khả năng linh hoạt trong việc đặt widget con bên trong nó thông qua các tuỳ chọn như padding, đường viền, nền,,,,
* Single child widgets thích hợp cho việc tạo ra các widget có tính ứng dụng cao và chỉ có một chức năng duy nhất như button, label....
* Một số single child layout widgets quan trọng trong *Flutter:*
* **Padding** − Được sử dụng để padding child widget. Ở đây, padding có thể sử dụng *EdgeInsets* class.
* **Align** − Căn lề child widget sử dụng thuộc tính *alignment*. Giá trị của *alignment* có thể được cung cấp bởi  *FractionalOffset* class.  *FractionalOffset* class xác định vị trí của phần tử từ vị trí điểm trên cùng bên trái
* Một số single child widgets khác:
* FittedBox
* AspectRatio
* ConstrainedBox
* Baseline
* FractinallySizedBox
* IntrinsicHeight
* IntrinsicWidth
* LiimitedBox
* OffStage
* OverflowBox
* SizedBox
* SizedOverflowBox
* Transform
* CustomSingleChildLayout
* **Multiple Child Widgets**
* Loại widget layout này sẽ cho phép có nhiều hơn một widget con. Ví dụ *Row* widget cho phép bố trí các widget con theo chiều ngang thành một hàng trong khi *Column* widget cho phép bố trí các widget con theo chiều dọc thành một cột.
* Một số widget layout dạng này được sử dụng phổ biến:
* Row
* Column
* ListView
* GridView
* Expanded
* Table
* Flow
* Stack

## **Gesture trong Flutter:**

* *Gesture* (cử chỉ) là cách mà người dùng tương tác với các thiết bị di động. Có rất nhiều cử chỉ khác nhau như vuốt, chạm, lắc... *Gesture* trong Flutter giúp ta xử lý các cử chỉ của người dùng, tương tự như việc bắt sự kiện trong Android.
* Một số Gesture được sử dụng rộng rãi:
* **Tap** − Chạm vào bề mặt thiết bị bằng đầu ngón tay trong thời gian ngắn sau đỏ thả ngón tay ra ngay
* **Double Tap** − Tap 2 lần trong thời gian ngắn
* **Drag** − Chạm vào bề mặt của thiết bị bằng đầu ngón tay và sau đó di chuyển đầu ngón tay một cách ổn định và cuối cùng thả ngón tay ra.
* **Flick** − Tương tự như drag nhưng thực hiện nhanh hơn.
* **Pinch** − Chụm bề mặt của thiết bị bằng hai ngón tay
* **Spread/Zoom** − Ngược lại với **Pinch**.
* **Panning** − Chạm vào bề mặt của thiết bị bằng đầu ngón tay và di chuyển nó theo bất kỳ hướng nào mà không nhả đầu ngón tay.
* Flutter cung cấp một sự hỗ trợ tuyết vời để xử lý tất cả các loại cử chỉ thông qua một tiện ích duy nhất **GestureDetector**.  Để xác định các cử chỉ tác động lên một widget, ta chỉ cần đặt widget đó bên trong GestureDetector widget. GestureDetector sẽ bắt các cử chỉ và gửi nhiều sự kiện dựa trên cử chỉ đó.
* Một số cử chỉ và các sự kiện tương ứng được đưa ra dưới đây
* Tap
* onTapDown
* onTapUp
* onTap
* onTapCancel
* Double tap
* onDoubleTap
* Long press
* onLongPress
* Vertical drag
* onVerticalDragStart
* onVerticalDragUpdate
* onVerticalDragEnd
* Horizontal drag
* onHorizontalDragStart
* onHorizontalDragUpdate
* onHorizontalDragEnd
* Pan
* onPanStart
* onPanUpdate
* onPanEnd
* **Flutter** cũng cung cấp một cơ chế phát hiện cử chỉ cấp thấp thông qua *Listener widget*. Nó sẽ phát hiện tất cả các tương tác của người dùng và sau đó gửi các sự kiện sau:
* PointerDownEvent
* PointerMoveEvent
* PointerUpEvent
* PointerCancelEvent
* Flutter còn cung cấp một số nhỏ các widget để thực hiện các cử chỉ cụ thể đơn giản cũng như phức tạp:
* **Dismissible** − Hỗ trợ flick gesture để đóng widget.
* **Draggable** − Hỗ trợ drag gesture để di chuyển widget.
* **LongPressDraggable** − Hỗ trợ drag gesture để duy chuyển widget, khi widget cha có thể kéo thả
* **DragTarget** − Chấp nhận *Draggable* widget
* **IgnorePointer** − Ẩn widget
* **AbsorbPointer** − Dừng việc xử lý cử chỉ trên widget
* **Scrollable** − Hỗ trợ cuộn nội dung trong widget

## **Quản lý trạng thái state trong Flutter:**

* **State là gì?**
* State là thông tin thể hiện trên widget mà có thể thay đổi trong suốt thời gian sống sót trên đời của widget.
* **StatefulWidget và StatelessWidget:**
* *StatefulWidget*
* Bản thân nó sẽ tự chủ động update UI
* Trong widget này có state tức là có data để có thể thay đổi được.
* Khi state thay đổi, nó sẽ gọi lại hàm build để rebuild widget dẫn đến thay đổi UI.
* Ví dụ:

class TrafficLightWidget extends StatefulWidget { // 1

final height = 3; // mọi data trong class Widget phải immutable

// data nào mutable xin mời qua class khác, là class State bên dưới :D

// khi StatefulWidget được khởi tạo nó sẽ gọi hàm createState để tạo 1 object State

@override

State<StatefulWidget> createState() {

return TrafficLightState();

}

}

// khi object Widget gọi hàm createState thì object State ra đời

class TrafficLightState extends State<TrafficLightWidget> { // 2

var currentLight = 'đỏ';

// hàm build

@override

Widget build(BuildContext context) {

return Text('Đèn cao ${widget.height} mét và đèn đang có màu $currentLight'); // 3

}

}

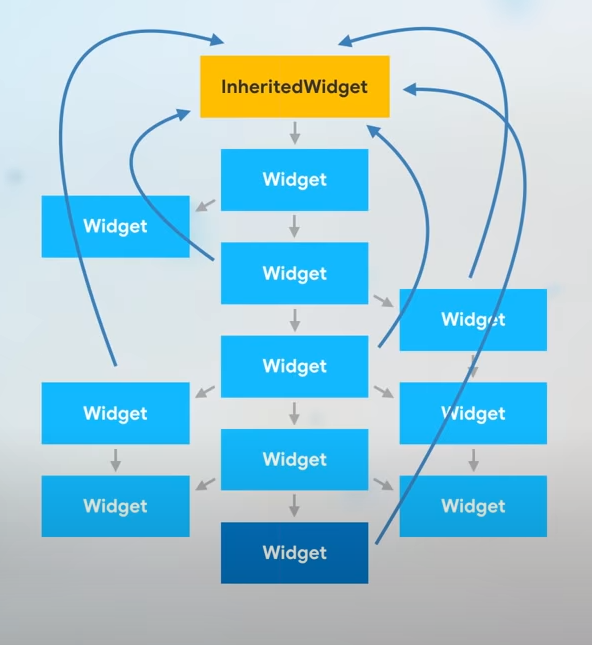
* Bên trong class State có hàm setState. Hàm setState cho phép bạn set lại giá trị state mới (VD set state sang màu vàng, màu xanh).
* Sau đó nó sẽ gọi lại hàm build để rebuild lại StatefulWidget đó bao gồm cả subwidgets luôn.
* *StatelessWidget*
* StatelessWidget là widget không có state, tức là nó không có data để thay đổi.
* Bản thân StatelessWidget không có hàm setState để nói widget chủ động rebuild lại UI.
* Stateless không có state, vậy nên bản thân nó cũng không có hàm createState mà thay vào đó là hàm build(BuildContext).
* Nó chỉ biết nhận data từ Widget ngoài truyền vào constructor của nó rồi nhận lấy data đó để show lên UI.
* Lưu ý mọi data trong cả Stateful và Stateless đều phải khai báo immutable
* Cố gắng sử dụng StatelessWidget hết mức có thể để tránh xảy ra bug.
* Việc quản lý trạng thái có thể được chia làm hai loại dựa vào thời gian tồn tại của trạng thái đó trong ứng dụng:
* **Ephemeral** (ngắn hạn) − Kéo dài trong vài giây như trạng thái của hiệu ứng (animation) hoặc một trang đơn như trang thông tin đánh gía sản phẩm. *Flutter* hỗ trợ quản lý trạng thái loại này thông qua **StatefulWidget**.
* **App state** (trạng thái ứng dụng) − Kèo dài trong toàn bộ ứng dụng như thông tin người dùng, thông tin giỏ hàng... *Flutter* hỗ trợ quản lý trạng thái loại này thông qua *scoped\_model*
* Ví dụ: Chúng ta cùng xem xét một ứng dụng *shopping cart* (giỏ hàng) đơn giản dưới đây:
* Người dùng đăng nhập bằng thông tin của họ vào trong ứng dụng.
* Khi người dùng đăng nhập, ứng dụng sẽ hiển thị thông tin người dùng ở tất cả các màn hình
* Một lần nữa, khi người dùng chọn một sản phẩm và lưu vào trong giỏ hàng (cart). Thông tin giỏ hàng sẽ tồn tại ở tất cả các trang cho đến khi người dùng xem giỏ hàng
* Thông tin người dùng và giỏ hàng tồn tại ở bất kì trường hợp nào gọi là trạng thái của ứng dụng ở thời điểm đó.
* **BuildContext và InheritedWidget?**
* *BuilContext:*
* Build context: như là một tham chiếu (reference) đến cái vị trí của widget (widget's location) trong widget tree.
* Mỗi loại widget đều có hàm build(), mỗi hàm build đều nhận 1 BuildContext làm argument.
* Vậy mỗi Widget đều có 1 BuildContext đại diện cho vị trí của chính Widget đó trên widget tree.
* Ví dụ:

class MyHomePage extends StatelessWidget {

@override

Widget build(BuildContext context)

* Buildcontext giúp ta giải quyết vấn đề truyền thẳng data từ các widgets thế hệ ông, cha xuống các widgets con, cháu mà không nhất thiết cần di truyền qua tất cả các thế hệ ở giữa.
* *InheritedWidget:*
* **Inheritedwidget:** là một nơi lưu trữ data và cung cấp data cho widget con trong widget tree. Tất cả widget con của InheritedWidget đều có thể truy cập vào InheritedWidget để lấy data.



Hình 5 InheritedWidget

* Ví dụ:

class MyHomePageState extends State<MyHomePage> {

  int counter = 0;

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      // khởi tạo MyInheritedWidget tại vị trí cha của cả MyCenterWidget và MyText

      body: MyInheritedWidget(

        child: MyCenterWidget(), // child là sub tree từ MyCenterWidget xuống

        myData: counter, // data cần chia sẻ cho mấy widget child chính là counter

      ),

      floatingActionButton: FloatingActionButton(

        onPressed: () {

          setState(() {

            counter++;

          });

        },

        child: Icon(Icons.add),

      ),

    );

  }

}

// thoải mái xóa hết constructor và data trong MyCenterWidget

class MyCenterWidget extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return Center(

      child: MyText(),

    );

  }

}

// thậm chí xóa luôn constructor và data khai báo trong MyText

class MyText extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    // Nhờ hàm MyInheritedWidget.of ta sẽ get được data

    // Bởi vì hàm of là hàm static nên ta có thể gọi ở bất cứ widget con nào ta muốn

    final counter = MyInheritedWidget.of(context).myData;

    // get được data thì update UI thôi :D

    return Text('Tui là widget Text. Data của tui hiện tại là: $counter');

  }

}

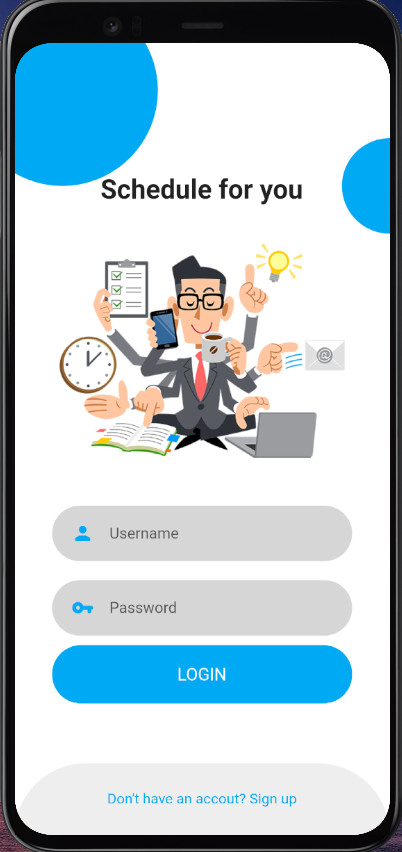
* **InheritedWidget** nên được đặt ở 1 trong 2 vị trí sau:
* App-scoped: Vị trí root của App, cho tất cả widget trong toàn App đều truy cập được, giống như cách Theme đang làm, hoặc nếu bạn biết widget MediaQuery thì nó cũng là App-scoped InheritedWidget.
* Page-scoped: Vị trí root của Page, Page ở đây đại diện cho 1 màn hình ấy, 1 App có thể có nhiều màn hình (nhiều Page).
* Tại sao cần phải biết InheritedWidget và cách nó hoạt động. Vì đây là nền tảng, là gốc gác, là một phương pháp quản lý state low-level. Nhờ đó mà chúng ta có thể học lên các phương pháp quản lý state level cao hơn như provider, bloc.

## **So sánh Flutter với Android (java)**

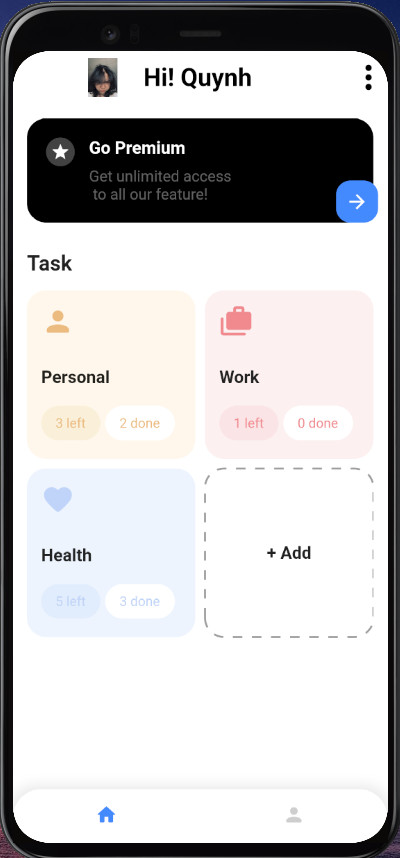
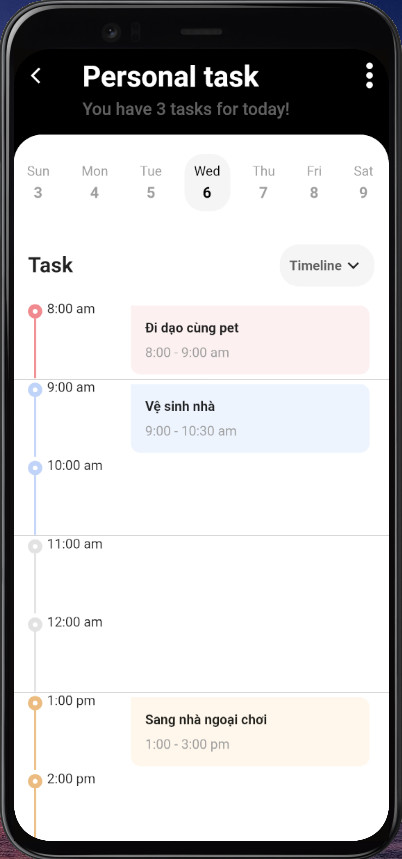
|  |  |
| --- | --- |
| **Flutter** | **Android (Java)** |
| -        - Framework hoạt động được trên cả hai hệ điều hành Android và iOS.  -       - Chạy trên engine render Flutter (C++) và Flutter Framework (viết bằng ngôn ngữ Dart), cả hai bộ này đều được đóng gói cùng mọi ứng dụng sau đó SDK sẽ đóng gói ứng dụng để có thể chạy trên cả hai nền tảng Android và iOS. | -        - Chỉ hoạt động được trên Android.  -        - Chạy trên SDK dành cho di động do Google phát triển, được viết bằng Java. |

# **PHẦN II: CHƯƠNG TRÌNH DEMO**

## **Hình ảnh demo**



Hình 6 Giao diện login



Hình 7 Giao diện chính

Hình 8 Giao diện detail