LÝ THUYẾT BUỔI 9

1. **Storage:**

Trong ngữ cảnh của Flutter, " Storage" thường đề cập đến quá trình lưu trữ và truy xuất dữ liệu một cách liên tục. Flutter cung cấp một số tùy chọn để xử lý lưu trữ, tùy thuộc vào loại dữ liệu bạn muốn lưu trữ và mức độ bền bỉ cần thiết. Dưới đây là một số tùy chọn lưu trữ phổ biến trong Flutter:

Shared Preferences:

Shared Preferences là một hệ thống lưu trữ khóa-giá trị đơn giản được cung cấp bởi khung Flutter. Nó cho phép bạn lưu trữ một lượng nhỏ dữ liệu, chẳng hạn như cài đặt người dùng hoặc tùy chọn, một cách liên tục. Dữ liệu này được lưu trữ cục bộ trên thiết bị và có thể được truy cập ngay cả sau khi ứng dụng đã đóng.

File Storage:

Flutter cung cấp API để đọc và ghi tệp trên hệ thống tệp của thiết bị. Ta có thể sử dụng bộ nhớ tệp để lưu trữ lượng dữ liệu lớn hơn hoặc dữ liệu có cấu trúc như tệp JSON hoặc CSV. File Storage phù hợp với các tình huống mà bạn cần duy trì dữ liệu giữa các phiên ứng dụng.

SQLite Database:

Đối với các yêu cầu lưu trữ dữ liệu phức tạp hơn, bạn có thể sử dụng SQLite Database trong Flutter. SQLite là một công cụ cơ sở dữ liệu nhẹ và hiệu quả cho phép bạn tạo và quản lý cơ sở dữ liệu quan hệ trong ứng dụng Flutter của mình. Đó là một lựa chọn tốt để xử lý dữ liệu có cấu trúc yêu cầu truy vấn và lọc.

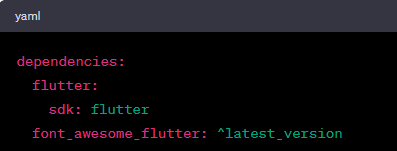
1. **3rd Party Librabies:**

**a. Font\_awesome\_flutter** là gói Flutter của bên thứ ba cho phép ta dễ dàng sử dụng các biểu tượng Font Awesome trong các ứng dụng Flutter của mình. Font Awesome là một thư viện phông chữ biểu tượng phổ biến cung cấp một loạt các biểu tượng có thể được sử dụng trong các ứng dụng web và di động.

Gói font\_awesome\_flutter được phát triển và duy trì bởi cộng đồng Flutter, và nó bao bọc phông chữ biểu tượng Font Awesome trong các widget Flutter. Điều này giúp ta dễ dàng đưa các biểu tượng Font Awesome vào ứng dụng của mình mà không cần thêm nội dung hình ảnh.

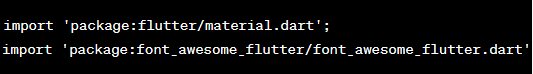
Để sử dụng font\_awesome\_flutter trong dự án Flutter của bạn, hãy làm theo các bước sau:

Thêm gói vào tệp pubspec.yaml của bạn:



Chạy flutter pub get, vào thiết bị đầu cuối để lấy gói.

Import gói trong code Dart:



Sử dụng tiện ích FaIcon để hiển thị các biểu tượng Font Awesome trong giao diện người dùng của bạn. Chẳng hạn:



**b.** Có một số thư viện bên thứ ba liên quan đến **SQFLite** và quản lý cơ sở dữ liệu trong Flutter. Dưới đây là một số ví dụ về những thư viện đó:

Moor: Moor là một thư viện bền, có tính phản ứng và dễ sử dụng cho Flutter, sử dụng SQLite như cơ sở dữ liệu nền. Nó cho phép bạn làm việc với cơ sở dữ liệu bằng cách sử dụng API trôi chảy và an toàn kiểu. Moor tạo mã Dart dựa trên các mô hình dữ liệu đã định nghĩa của bạn, giúp giảm thiểu việc lập trình các hoạt động cơ sở dữ liệu.

ObjectBox: ObjectBox là một cơ sở dữ liệu nhanh, hướng đối tượng hoạt động tốt với Flutter. Nó được thiết kế để đơn giản và hiệu quả, phù hợp cho các ứng dụng di động. ObjectBox hỗ trợ các truy vấn, mối quan hệ và đồng bộ hóa tự động.

Floor: Floor là một thư viện ORM (Object-Relational Mapping) cho Flutter, cung cấp một lớp trừu tượng trên SQLite. Nó cho phép bạn làm việc với cơ sở dữ liệu SQLite bằng cách sử dụng các lớp và chú thích Dart, đơn giản hóa việc thao tác cơ sở dữ liệu.

Hive: Hive là một thư viện cơ sở dữ liệu NoSQL nhẹ dùng cho lưu trữ key-value, hỗ trợ mã hóa tích hợp. Hive cung cấp tốc độ đọc/ghi nhanh và phù hợp cho các trường hợp bạn cần lưu trữ dữ liệu đơn giản một cách hiệu quả.

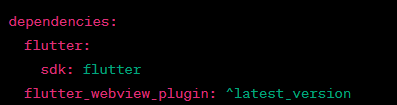
Lưu ý rằng tính phổ biến và liên quan của các thư viện cụ thể có thể thay đổi theo thời gian, và có thể đã xuất hiện những thư viện mới ngoài những cái được liệt kê ở trên. Do đó, bạn nên ghé thăm trang web chính thức của Flutter và khám phá các gói cộng đồng Flutter trên pub.dev để tìm các thư viện bên thứ ba liên quan đến SQFLite và quản lý cơ sở dữ liệu trong Flutter mới nhất và phù hợp nhất.

**c. `flutter\_webview\_plugin`** là một thư viện bên thứ ba cho Flutter, cho phép bạn tích hợp trình duyệt web (webview) vào ứng dụng của mình. Đây là một công cụ hữu ích khi bạn cần hiển thị nội dung web như trang web hoặc trang đăng nhập bên trong ứng dụng của mình.

Dưới đây là một số thông tin về `flutter\_webview\_plugin`:

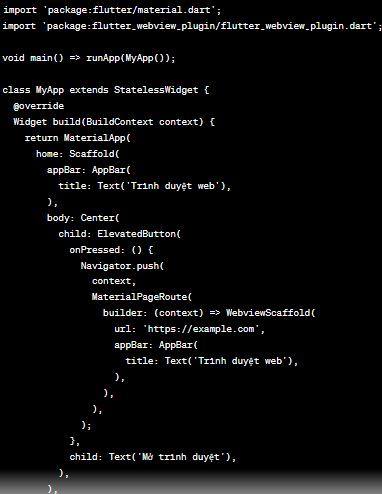
1. Mô tả: `flutter\_webview\_plugin` cho phép bạn dễ dàng tích hợp một trình duyệt web vào ứng dụng Flutter của mình. Thư viện này sử dụng trình duyệt web của hệ điều hành và cho phép bạn hiển thị trang web trong một cửa sổ nhỏ (popup) hoặc trong toàn màn hình.

2. Cách sử dụng: Để bắt đầu sử dụng `flutter\_webview\_plugin`, bạn cần thêm thư viện này vào file `pubspec.yaml` của ứng dụng Flutter và chạy lệnh `flutter pub get` để tải thư viện về.



Sau đó, bạn có thể sử dụng các hàm của `flutter\_webview\_plugin` để tạo và điều khiển trình duyệt web trong ứng dụng của mình.

3. Ví dụ: Dưới đây là một ví dụ đơn giản về cách sử dụng `flutter\_webview\_plugin` để hiển thị một trang web:



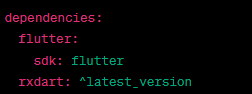
Trên đây là một số thông tin về `flutter\_webview\_plugin` và cách sử dụng nó trong Flutter. Nhưng hãy nhớ rằng tính phổ biến và tính năng của các thư viện bên thứ ba có thể thay đổi theo thời gian, vì vậy hãy luôn kiểm tra trang pub.dev để tìm phiên bản mới nhất và thông tin chi tiết về `flutter\_webview\_plugin`.

**d.** `**rxdart**` là một thư viện bên thứ ba dành cho Flutter, hỗ trợ các tính năng lập trình bất đồng bộ và xử lý dữ liệu dạng luồng (stream) một cách dễ dàng và hiệu quả. "rxdart" được viết tắt từ Rx (Reactive Extensions) Dart, là một bộ thư viện lập trình bất đồng bộ trên Flutter sử dụng Dart language.

Dưới đây là một số thông tin về `rxdart`:

1. Mô tả: `rxdart` cung cấp các lớp và phương thức để làm việc với luồng dữ liệu (stream) trong Flutter một cách dễ dàng và có cấu trúc. Nó sử dụng mô hình lập trình phản ứng (reactive programming) để xử lý sự kiện và sự thay đổi dữ liệu trong ứng dụng của bạn.

2. Cách sử dụng: Để sử dụng `rxdart` trong dự án Flutter, bạn cần thêm thư viện này vào file `pubspec.yaml` của ứng dụng và chạy lệnh `flutter pub get` để tải thư viện về.



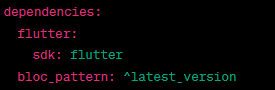
Sau khi đã cài đặt thành công, bạn có thể sử dụng các tính năng của `rxdart` trong mã Dart của ứng dụng.

**e.** `**bloc\_pattern**` là một thư viện bên thứ ba cho Flutter, được sử dụng để triển khai kiến trúc BLoC (Business Logic Component) trong ứng dụng của bạn. BLoC là một mô hình kiến trúc được sử dụng rộng rãi trong Flutter để quản lý trạng thái ứng dụng và xử lý logic kinh doanh. `bloc\_pattern` là một trong các cách tiếp cận để triển khai mô hình BLoC trong Flutter.

Dưới đây là một số thông tin về `bloc\_pattern`:

1. Mô tả: `bloc\_pattern` cung cấp một cách tiện lợi để xây dựng và quản lý BLoCs trong ứng dụng Flutter. Nó giúp bạn tách biệt logic kinh doanh khỏi giao diện người dùng và quản lý trạng thái ứng dụng một cách hiệu quả.

2. Cách sử dụng: Để sử dụng `bloc\_pattern` trong dự án Flutter, bạn cần thêm thư viện này vào file `pubspec.yaml` của ứng dụng và chạy lệnh `flutter pub get` để tải thư viện về.



Sau khi đã cài đặt thành công, bạn có thể sử dụng các tính năng của `bloc\_pattern` trong mã Dart của ứng dụng.

**f.** Trong Flutter, thư viện `**http**` được sử dụng để thực hiện các yêu cầu HTTP như tải tệp tin, gửi dữ liệu và tương tác với các API từ máy chủ. Nó giúp bạn tạo các kết nối mạng trong ứng dụng của mình để truy cập và giao tiếp với các nguồn dữ liệu từ xa.

Dưới đây là một số thông tin về `http` trong Flutter:

1. Mô tả: `http` là một gói chính thức của Flutter, cho phép bạn tạo và quản lý các yêu cầu HTTP như GET, POST, PUT, DELETE, vv. Nó hỗ trợ các chức năng chuyển đổi dữ liệu sang JSON và xử lý các trạng thái yêu cầu như thành công, lỗi hoặc tạm ngừng.

2. Cách sử dụng: Để sử dụng `http` trong dự án Flutter, bạn cần thêm gói này vào file `pubspec.yaml` của ứng dụng và chạy lệnh `flutter pub get` để tải thư viện về.

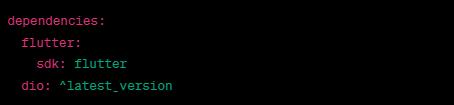
Sau khi đã cài đặt thành công, bạn có thể sử dụng các hàm của `http` trong mã Dart của ứng dụng.

**g.** `**dio**` là một thư viện bên thứ ba cho Flutter, được sử dụng để thực hiện các yêu cầu HTTP và tương tác với các API từ máy chủ. `dio` giúp bạn tạo các kết nối mạng linh hoạt và hiệu quả trong ứng dụng của mình.

Dưới đây là một số thông tin về `dio` trong Flutter:

1. Mô tả: `dio` là một thư viện HTTP mạnh mẽ cho Flutter, được phát triển dựa trên Dart, hỗ trợ các yêu cầu HTTP như GET, POST, PUT, DELETE, vv. Nó cung cấp nhiều tính năng tiện ích như chuyển đổi dữ liệu sang JSON, xử lý các yêu cầu đồng bộ và bất đồng bộ, tải xuống tệp tin, xử lý mã lỗi, và nhiều tính năng mạnh mẽ khác.

2. Cách sử dụng: Để sử dụng `dio` trong dự án Flutter, bạn cần thêm thư viện này vào file `pubspec.yaml` của ứng dụng và chạy lệnh `flutter pub get` để tải thư viện về.



Sau khi đã cài đặt thành công, bạn có thể sử dụng các hàm của `dio` trong mã Dart của ứng dụng.

**h.** `**get\_it**` là một thư viện bên thứ ba cho Flutter, cung cấp một cơ chế dễ sử dụng để quản lý các đối tượng (dependencies) và triển khai Dependency Injection (DI) trong ứng dụng của bạn. DI là một mô hình thiết kế phần mềm giúp giảm sự phụ thuộc giữa các thành phần của ứng dụng và tăng tính linh hoạt trong việc thay đổi, kiểm thử và tái sử dụng code.

Dưới đây là một số thông tin về `get\_it` trong Flutter:

1. Mô tả: `get\_it` là một thư viện dùng để quản lý dependencies (đối tượng) và triển khai DI trong Flutter. Nó giúp bạn dễ dàng tạo, lưu trữ và truy xuất các đối tượng trong ứng dụng của mình một cách hiệu quả.

2. Cách sử dụng: Để sử dụng `get\_it` trong dự án Flutter, bạn cần thêm thư viện này vào file `pubspec.yaml` của ứng dụng và chạy lệnh `flutter pub get` để tải thư viện về.

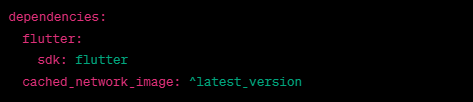


**i.** `**cached\_network\_image**` là một thư viện bên thứ ba cho Flutter, cho phép bạn tải và hiển thị hình ảnh từ mạng với việc tự động lưu trữ (cache) hình ảnh để tăng hiệu suất và giảm lượng dữ liệu tải về từ máy chủ.

Dưới đây là một số thông tin về `cached\_network\_image` trong Flutter:

1. Mô tả: `cached\_network\_image` là một gói cho phép bạn hiển thị hình ảnh từ mạng trong ứng dụng của mình một cách dễ dàng. Gói này tự động lưu trữ (cache) hình ảnh đã tải về, giúp tăng tốc độ hiển thị hình ảnh khi người dùng quay lại màn hình hoặc khởi động ứng dụng lại.

2. Cách sử dụng: Để sử dụng `cached\_network\_image` trong dự án Flutter, bạn cần thêm gói này vào file `pubspec.yaml` của ứng dụng và chạy lệnh `flutter pub get` để tải thư viện về.



Sau khi đã cài đặt thành công, bạn có thể sử dụng các widget và tính năng của `cached\_network\_image` trong mã Dart của ứng dụng.

1. **Behavior Components:**

**a.** Trong Flutter, "**media playback**" (phát lại phương tiện) là quá trình phát các phương tiện như video và âm thanh trong ứng dụng của bạn. Để thực hiện phát lại phương tiện, bạn có thể sử dụng các thư viện bên thứ ba như `video\_player` để phát video và `audioplayers` để phát âm thanh.

Dưới đây là một số thông tin về cách thực hiện "media playback" trong Flutter:

1. Phát lại video với `video\_player`:

Thư viện `video\_player` cho phép bạn phát lại các tệp video trong ứng dụng Flutter. Bạn có thể cung cấp URL của video từ mạng hoặc đường dẫn đến tệp video trong thiết bị và sử dụng widget `VideoPlayer` để hiển thị nội dung video trên giao diện người dùng.

2. Phát lại âm thanh với `audioplayers`:

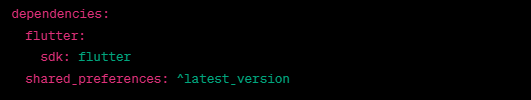
Thư viện `audioplayers` cho phép bạn phát lại các tệp âm thanh trong ứng dụng Flutter. Bạn có thể cung cấp URL của tệp âm thanh từ mạng hoặc đường dẫn đến tệp âm thanh trong thiết bị và sử dụng `AudioPlayer` để điều khiển phát lại âm thanh.

**b.** Trong Flutter, "**preference**" (tùy chọn) thường liên quan đến việc lưu trữ và quản lý các cài đặt hoặc dữ liệu của ứng dụng. Thư viện `shared\_preferences` là một thư viện phổ biến được sử dụng để quản lý tùy chọn trong Flutter.

Dưới đây là một số thông tin về thư viện `shared\_preferences` trong Flutter:

1. Mô tả: `shared\_preferences` là một gói cho phép bạn lưu trữ các tùy chọn và dữ liệu cơ bản của ứng dụng trong bộ nhớ tạm thời (shared preferences) của thiết bị. Thông thường, các tùy chọn này được lưu trữ dưới dạng cặp key-value, giúp bạn dễ dàng lưu trữ và truy xuất thông tin của ứng dụng một cách đơn giản.

2. Cách sử dụng: Để sử dụng `shared\_preferences` trong dự án Flutter, bạn cần thêm gói này vào file `pubspec.yaml` của ứng dụng và chạy lệnh `flutter pub get` để tải thư viện về.



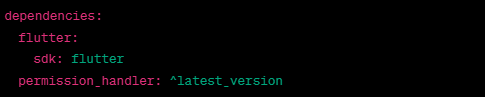
Sau khi đã cài đặt thành công, bạn có thể sử dụng các phương thức của `shared\_preferences` trong mã Dart của ứng dụng.

**c.** Trong Flutter, "**permission**" (quyền truy cập) là quyền mà ứng dụng cần yêu cầu từ người dùng để truy cập các tính năng hoặc tài nguyên của thiết bị, chẳng hạn như máy ảnh, micro, vị trí GPS, vv. Để yêu cầu quyền truy cập, bạn có thể sử dụng thư viện `permission\_handler` để dễ dàng xử lý việc yêu cầu và kiểm tra các quyền trong Flutter.

Dưới đây là một số thông tin về thư viện `permission\_handler` trong Flutter:

1. Mô tả: `permission\_handler` là một gói cho phép bạn yêu cầu và kiểm tra các quyền truy cập trong Flutter. Gói này hỗ trợ yêu cầu quyền đơn giản và kiểm tra trạng thái quyền của các tài nguyên khác nhau trên thiết bị.

2. Cách sử dụng: Để sử dụng `permission\_handler` trong dự án Flutter, bạn cần thêm gói này vào file `pubspec.yaml` của ứng dụng và chạy lệnh `flutter pub get` để tải thư viện về.



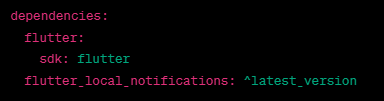
Sau khi đã cài đặt thành công, bạn có thể sử dụng các phương thức của `permission\_handler` trong mã Dart của ứng dụng.

**d.** Thông báo địa phương (**local notification**) trong Flutter là khả năng của ứng dụng để gửi thông báo trực tiếp từ thiết bị của người dùng mà không cần kết nối với máy chủ hoặc mạng. Thư viện `flutter\_local\_notifications` là một trong những thư viện phổ biến được sử dụng để triển khai tính năng thông báo địa phương trong ứng dụng Flutter.

Dưới đây là một số thông tin về thư viện `flutter\_local\_notifications` trong Flutter:

1. Mô tả: `flutter\_local\_notifications` là một thư viện cho phép bạn hiển thị thông báo địa phương trên thiết bị của người dùng. Thư viện này hỗ trợ nhiều tính năng như hiển thị thông báo với âm thanh, hình ảnh, lời nhắn, và các tùy chọn tùy chỉnh khác.

2. Cách sử dụng: Để sử dụng `flutter\_local\_notifications` trong dự án Flutter, bạn cần thêm gói này vào file `pubspec.yaml` của ứng dụng và chạy lệnh `flutter pub get` để tải thư viện về.



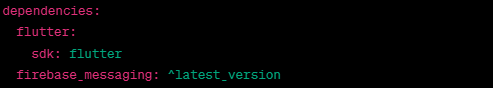
Sau khi đã cài đặt thành công, bạn có thể sử dụng các phương thức của `flutter\_local\_notifications` trong mã Dart của ứng dụng.

**e.** Thông báo đẩy (**push notification**) trong Flutter là khả năng của ứng dụng để gửi thông báo từ máy chủ hoặc dịch vụ đến thiết bị của người dùng mà không cần ứng dụng đang mở hoặc chạy ngay lúc đó. Thư viện `firebase\_messaging` là một trong những thư viện phổ biến được sử dụng để triển khai tính năng thông báo đẩy trong ứng dụng Flutter thông qua Firebase Cloud Messaging (FCM).

Dưới đây là một số thông tin về thư viện `firebase\_messaging` và cách triển khai tính năng push notification trong Flutter:

1. Mô tả: `firebase\_messaging` là một thư viện cho phép bạn tích hợp tính năng push notification vào ứng dụng Flutter. Thư viện này sử dụng Firebase Cloud Messaging (FCM) để gửi và nhận thông báo đẩy từ máy chủ đến thiết bị của người dùng.

2. Cách sử dụng: Để sử dụng `firebase\_messaging` trong dự án Flutter, bạn cần thêm gói này vào file `pubspec.yaml` của ứng dụng và chạy lệnh `flutter pub get` để tải thư viện về.



Sau khi đã cài đặt thành công, bạn cần cấu hình Firebase trong ứng dụng của mình và thiết lập các bộ điều khiển để xử lý việc nhận thông báo đẩy và tương tác với chúng.

**f.** Trong Flutter, "**download manager**" (quản lý tải xuống) là một tính năng cho phép ứng dụng tải xuống các tệp từ Internet và quản lý quá trình tải xuống. Hiện tại, Flutter không cung cấp một download manager tích hợp sẵn, nhưng bạn có thể sử dụng các thư viện bên thứ ba để triển khai tính năng này trong ứng dụng của mình.

Một trong số các thư viện phổ biến trong Flutter để quản lý tải xuống là `flutter\_downloader`. Dưới đây là một số thông tin về `flutter\_downloader`:

1. Mô tả: `flutter\_downloader` là một thư viện cho phép bạn quản lý quá trình tải xuống các tệp từ Internet trong ứng dụng Flutter. Thư viện này cung cấp các phương thức và sự kiện để bắt đầu, tạm dừng, tiếp tục, hủy bỏ và theo dõi tiến trình tải xuống.

2. Cách sử dụng: Để sử dụng `flutter\_downloader` trong dự án Flutter, bạn cần thêm gói này vào file `pubspec.yaml` của ứng dụng và chạy lệnh `flutter pub get` để tải thư viện về.

Sau khi đã cài đặt thành công, bạn có thể sử dụng các phương thức và sự kiện của `flutter\_downloader` để quản lý quá trình tải xuống trong mã Dart của ứng dụng.

1. **State Managements:**

Trong Flutter, quản lý trạng thái (State Management) là quá trình quản lý và theo dõi trạng thái của ứng dụng và các thành phần bên trong nó. Có nhiều phương pháp và thư viện để thực hiện quản lý trạng thái trong Flutter, mỗi phương pháp đều có ưu điểm và hạn chế riêng. Dưới đây là một số thành phần quan trọng trong quản lý trạng thái trong Flutter:

1. `setState`: Đây là phương pháp cơ bản nhất để quản lý trạng thái trong Flutter. Khi bạn muốn cập nhật trạng thái của một widget, bạn có thể sử dụng `setState` để gọi lại hàm `build` và cập nhật giao diện người dùng dựa trên trạng thái mới. Phương pháp này thích hợp cho các ứng dụng nhỏ và đơn giản, nhưng có thể trở nên khó quản lý khi ứng dụng phức tạp hơn.

2. InheritedWidget: `InheritedWidget` là một cơ chế quản lý trạng thái trong Flutter. Nó cho phép bạn chia sẻ trạng thái giữa các widget con một cách hiệu quả. Mỗi widget có thể truy cập trực tiếp đến trạng thái thông qua `InheritedWidget`. Tuy nhiên, sử dụng `InheritedWidget` có thể dẫn đến hiệu suất không tốt khi cần cập nhật nhiều widget cùng một lúc.

3. Provider: `Provider` là một thư viện được cung cấp bởi nhóm Flutter và có thể hỗ trợ bạn trong việc quản lý trạng thái. Thư viện này giúp bạn cung cấp và tiêu thụ các trạng thái trong ứng dụng một cách dễ dàng và linh hoạt. Provider sử dụng cơ chế ChangeNotifier để quản lý và thông báo về các thay đổi trong trạng thái.

4. BLoC (Business Logic Component): BLoC là một kiến trúc quản lý trạng thái phổ biến trong Flutter. Nó tách rời logic kinh doanh và trạng thái khỏi các widget để làm cho mã dễ dàng đọc và bảo trì. BLoC sử dụng Streams và Sink để truyền dữ liệu và cập nhật giữa các lớp. Thư viện rxdart thường được sử dụng cùng với BLoC để xử lý luồng dữ liệu.

5. Redux: Redux là một kiến trúc quản lý trạng thái phổ biến, được mượn từ React và được sử dụng rộng rãi trong các ứng dụng Flutter lớn. Redux tách rời trạng thái và logic kinh doanh vào một kho lưu trữ duy nhất. Các thành phần trong ứng dụng có thể gửi các hành động (actions) để cập nhật trạng thái của kho lưu trữ.

6. MobX: MobX là một thư viện quản lý trạng thái được sử dụng phổ biến trong Flutter. Nó sử dụng kiểm tra và tự động cập nhật các trạng thái khi có thay đổi. MobX giúp giảm thiểu mã boilerplate và cung cấp cú pháp đơn giản cho việc quản lý trạng thái.

Những phương pháp và thư viện quản lý trạng thái được nêu trên đều có ưu điểm và hạn chế riêng. Việc lựa chọn phương pháp và thư viện quản lý trạng thái phụ thuộc vào kích thước và phức tạp của ứng dụng, yêu cầu hiệu suất, và sự quen thuộc của bạn với các mô hình kiến trúc và công nghệ cụ thể.

1. **Quality Assurance:**

Đảm bảo chất lượng (QA) trong Flutter đề cập đến quá trình đảm bảo rằng ứng dụng Flutter của bạn đáp ứng các tiêu chuẩn cao nhất về chất lượng, độ tin cậy và sự hài lòng của người dùng. Nó bao gồm một loạt các hoạt động và thực tiễn nhằm xác định và ngăn ngừa lỗi, đảm bảo hiệu suất nhất quán và mang lại trải nghiệm người dùng bóng bẩy và không có lỗi. Dưới đây là một số khía cạnh chính của đảm bảo chất lượng trong Flutter:

* Kiểm tra đơn vị và Tự động hóa kiểm tra: Viết kiểm tra đơn vị cho các thành phần và tính năng riêng lẻ của ứng dụng Flutter của bạn để đảm bảo rằng chúng hoạt động chính xác. Triển khai tự động hóa thử nghiệm bằng cách sử dụng khung thử nghiệm của Flutter để tự động chạy thử nghiệm bất cứ khi nào thay đổi mã được thực hiện.
* Kiểm tra tích hợp: Tiến hành kiểm tra tích hợp để đảm bảo rằng các phần khác nhau trong ứng dụng của bạn hoạt động liền mạch với nhau. Điều này giúp xác định bất kỳ vấn đề nào có thể phát sinh khi các thành phần khác nhau tương tác với nhau.
* Kiểm tra giao diện người dùng: Thực hiện kiểm tra giao diện người dùng để xác minh rằng giao diện người dùng hoạt động như mong đợi. Các công cụ như `flutter\_driver` của Flutter có thể được sử dụng để tự động kiểm tra giao diện người dùng và đảm bảo rằng các yếu tố trực quan của ứng dụng và tương tác người dùng đang hoạt động bình thường.
* Kiểm tra hiệu suất: Kiểm tra hiệu suất của ứng dụng trong các điều kiện khác nhau, chẳng hạn như các cấu hình thiết bị và điều kiện mạng khác nhau. Xác định và giải quyết bất kỳ tắc nghẽn hiệu suất nào có thể ảnh hưởng đến trải nghiệm người dùng.
* Kiểm tra khả năng truy cập: Đảm bảo rằng ứng dụng của bạn có thể truy cập được đối với người dùng khuyết tật. Flutter cung cấp các công cụ và thư viện để giúp bạn kiểm tra và cải thiện khả năng truy cập ứng dụng của mình.
* Kiểm tra bảo mật: Tiến hành đánh giá bảo mật để xác định các lỗ hổng và rủi ro bảo mật tiềm ẩn trong ứng dụng của bạn. Đảm bảo rằng dữ liệu người dùng được xử lý an toàn và ứng dụng của bạn tuân theo các phương pháp hay nhất để bảo vệ dữ liệu.
* Tích hợp liên tục và Triển khai liên tục (CI/CD): Thiết lập quy trình CI/CD để tự động hóa quy trình xây dựng, thử nghiệm và triển khai ứng dụng của bạn. Điều này giúp phát hiện sớm các sự cố và đảm bảo rằng chỉ mã chất lượng cao và được kiểm tra kỹ càng mới được đưa vào sản xuất.

1. **Version Control**

Quản lý phiên bản Flutter cho phép các loại phiên bản Flutter khác nhau có sẵn trên cơ sở dự án. Điều này có nghĩa là chúng ta có thể xác định các loại phiên bản Flutter cụ thể cho các loại dự án khác nhau, nó cho phép chúng ta phát hành nhiều kênh, lưu vào bộ nhớ cache cục bộ. Vì vậy, khi chuyển đổi các phiên bản thì chúng ta không phải đợi thiết lập.

* **Flutter Version Management (FVM):**

Khi làm việc trên dự án Flutter, chúng tôi cần phát hành phiên bản cập nhật và ứng dụng, xác minh nó và chuyển đổi các loại SDK khác nhau để kiểm tra nó, điều này khiến chúng tôi mất thời gian phát triển. Để tránh điều này, chúng tôi sử dụng Quản lý phiên bản Flutter, cung cấp cho chúng tôi các loại phiên bản Flutter khác nhau cho máy của chúng tôi. Vì vậy, mỗi lần Flutter có thể kiểm tra ứng dụng dựa trên bản phát hành Flutter cập nhật mà không cần chờ cài đặt và sẽ có thể chuyển sang phiên bản Flutter cho phù hợp.

* **Cài đặt FVM:**

Trước hết, chúng ta cần phải đảm bảo rằng flutter đã được cài đặt và flutter có phải là kênh ổn định hay không. nếu không thì Nhập mã dưới đây vào dòng lệnh của bạn.

// set flutter to stable channel

flutter channel stable

// check flutter channel

flutter channel

// output

Flutter channels:

master

dev

beta

\* stable

Sau đó, chúng ta phải xác định xem flutter của chúng ta đã được cài đặt hay chưa, nếu chưa thì trước tiên chúng ta sẽ cài đặt FVM.

$ pub global activate fvm

Bây giờ chúng ta sẽ thấy rằng một số cảnh báo được đưa ra ở cuối quá trình cài đặt, vì vậy chúng ta cần thêm đường dẫn fvm vào tệp cấu hình shell (.bashrc, bash\_profile, v.v.) trước khi thực hiện bước tiếp theo.

export PATH=”$PATH:`pwd`/flutter/bin”$ fvm install

export PATH=”$PATH:`pwd`/bin/cache/dart-sdk/bin”

export PATH=”$PATH:`pwd`/.pub-cache/bin”

**Cài đặt SDK**

FVM cho phép chúng ta cài đặt nhiều loại bản phát hành hoặc nhiều kênh. Để cài đặt kênh sử dụng ổn định và để cài đặt phiên bản phát hành flutter, chúng ta sẽ sử dụng v2.0.5 hoặc 1.17.0-dev.3.1 và ngay sau khi chúng ta chạy - bỏ qua thiết lập sau đó nó sẽ bỏ qua quá trình thiết lập.

$ fvm install stable or fvm install 2.0.5

**Phiên bản SDK cấu hình dự án:**

Sau đó, chúng ta sẽ thấy rằng Dự án có được cấu hình để sử dụng một phiên bản cụ thể hay không, nếu không, chúng ta sẽ cài đặt nó trên phiên bản thích hợp mà không cần đối số.

$ fvm install

**Danh sách các phiên bản đã cài đặt của Flutter:**

Bây giờ bằng cách gõ vào lệnh sau, chúng ta có thể liệt kê phiên bản đã cài đặt trên máy của mình bằng cách sử dụng lệnh dưới đây FVM sẽ lưu trữ phiên bản SDK.

$ fvm list

**Nâng cấp phiên bản SDK:**

Sử dụng lệnh nâng cấp phiên bản SDK khi chúng tôi cần nâng cấp phiên bản SDK hiện tại của mình, do đó bạn phải gọi lệnh của SDK Flutter của mình như trong cài đặt Flutter thông thường.

$ fvm flutter upgrade

**Cài đặt IDE:**

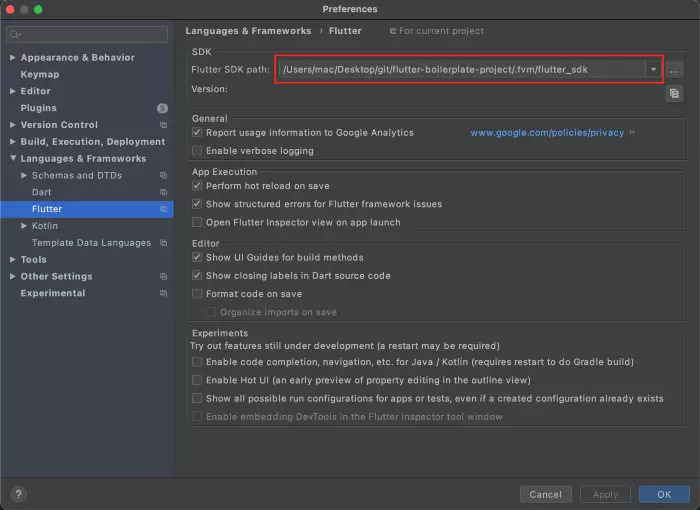
Bây giờ chúng ta sẽ xem cách định cấu hình IDE, dưới đây chúng tôi đã chỉ ra cách định cấu hình trong Android Studio chúng ta hãy xem nó ngay bây giờ.

**Android Studio**

Trong thư mục dự án gốc của bạn, hãy sao chép liên kết tượng trưng sau tuyệt đối.

Example: /absolute/path-to-your-project/.fvm/flutter\_sdk

Sau đó, chúng tôi sẽ mở Languages and Frameworks trong menu của Android Studio -> Bây giờ hãy tìm kiếm Flutter và thay đổi đường dẫn đến Flutter SDK. và thực hiện thay đổi. Bây giờ bạn có thể chạy nó với các phiên bản đã chọn và gỡ lỗi nó. Nếu bạn muốn xem cài đặt mới được thực hiện thì chúng ta có thể sử dụng Android Studio sẽ khởi động lại.



1. **Firebase**

Nếu bạn mới sử dụng Firebase trong Flutter, chúng tôi khuyên bạn nên bắt đầu với video và phòng thí nghiệm mã hóa Tìm hiểu Firebase cho Flutter :

**Điều kiện tiên quyết**

Trước khi bắt đầu, tài liệu giả định rằng bạn có thể tạo (hoặc có sẵn) dự án Flutter và cũng có tài khoản Firebase đang hoạt động.

**Cài đặt**

Trước khi sử dụng bất kỳ dịch vụ nào của Firebase, bạn phải trước tiên cài đặt plugin firebase\_core, nhiệm vụ của nó là kết nối ứng dụng của bạn với Firebase.

Cài đặt plugin bằng cách chạy lệnh sau từ thư mục gốc của dự án:



**Khởi tạo FlutterFire**

FlutterFire hiện đã hỗ trợ khởi tạo trực tiếp từ Dart!

Trước khi có thể sử dụng bất kỳ dịch vụ Firebase nào, FlutterFire cần được khởi chạy (bạn có thể coi quá trình này giống như chính FlutterFire "bootstrapping"). Bước khởi tạo là không đồng bộ, nghĩa là bạn sẽ cần ngăn mọi hoạt động sử dụng liên quan đến FlutterFire cho đến khi quá trình khởi tạo hoàn tất.

**Sử dụng FlutterFire CLI**

Để khởi tạo FlutterFire, hãy gọi ***initializeAppp***hương thức trên lớp ***Firebase***. Phương thức chấp nhận cấu hình ứng dụng dự án Firebase của bạn, cấu hình này có thể được lấy cho tất cả các nền tảng được hỗ trợ bằng cách sử dụng FlutterFire CLI:

****

Sau khi được định cấu hình, một firebase\_options.darttệp sẽ được tạo cho bạn có chứa tất cả các tùy chọn cần thiết để khởi tạo. Ngoài ra, nếu ứng dụng Flutter của bạn hỗ trợ Android ***thì plugin Android Google Services Gradle*** sẽ tự động được áp dụng cho bạn.

**Tìm hiểu thêm về FlutterFire CLI**

Nếu bạn thêm hỗ trợ cho một nền tảng mới trong ứng dụng Flutter của mình (ví dụ: thêm Android khi ứng dụng của bạn trước đây không hỗ trợ Android) hoặc nếu bạn đưa các dịch vụ Firebase mới vào ứng dụng của mình (ví dụ: thêm ) thì bạn nên định cấu hình lại Firebase cho ứng dụng của mình thông firebase\_databasequa CLI ( flutterfire configure).

**Khởi tạo**

Tiếp theo, các tùy chọn đã tạo cần được cung cấp cho initializeAppphương thức. Vì đây là hoạt động không đồng bộ nên mainchức năng có thể được sửa đổi để đảm bảo quá trình khởi tạo hoàn tất trước khi chạy ứng dụng.

Đầu tiên nhập firebase\_coreplugin và firebase\_options.darttệp được tạo:

A blue rectangular object with a white line

Description automatically generated

Tiếp theo, trong mainhàm, ensure WidgetsFlutterBindingđược khởi tạo và sau đó khởi tạo Firebase:

A computer screen shot of a computer screen

Description automatically generated

Được DefaultFirebaseOptions.currentPlatformnhập từ firebase\_options.darttệp được tạo của chúng tôi.

Sau khi khởi chạy, bạn đã sẵn sàng bắt đầu sử dụng FlutterFire!

**Cài đặt thủ công**

Với tính năng khởi tạo chỉ dành cho Dart hiện được hỗ trợ, việc định cấu hình và cài đặt nền tảng theo cách thủ công không còn cần thiết nữa. Nếu bạn muốn xem tài liệu về khởi tạo thủ công, hãy xem tài liệu Cài đặt thủ công .

**Các bước tiếp theo**

Về bản thân, firebase\_core plugin cung cấp chức năng cơ bản để sử dụng với Firebase. FlutterFire được chia thành các plugin riêng lẻ, có thể cài đặt cho phép bạn tích hợp với một dịch vụ Firebase cụ thể.

1. **Native Integration**

Hướng dẫn này mô tả cách viết mã dành riêng cho nền tảng tùy chỉnh. Một số chức năng dành riêng cho nền tảng có sẵn thông qua các gói hiện có; xem bằng cách sử dụng các gói .

Flutter sử dụng một hệ thống linh hoạt cho phép bạn gọi các API dành riêng cho nền tảng bằng ngôn ngữ hoạt động trực tiếp với các API đó:

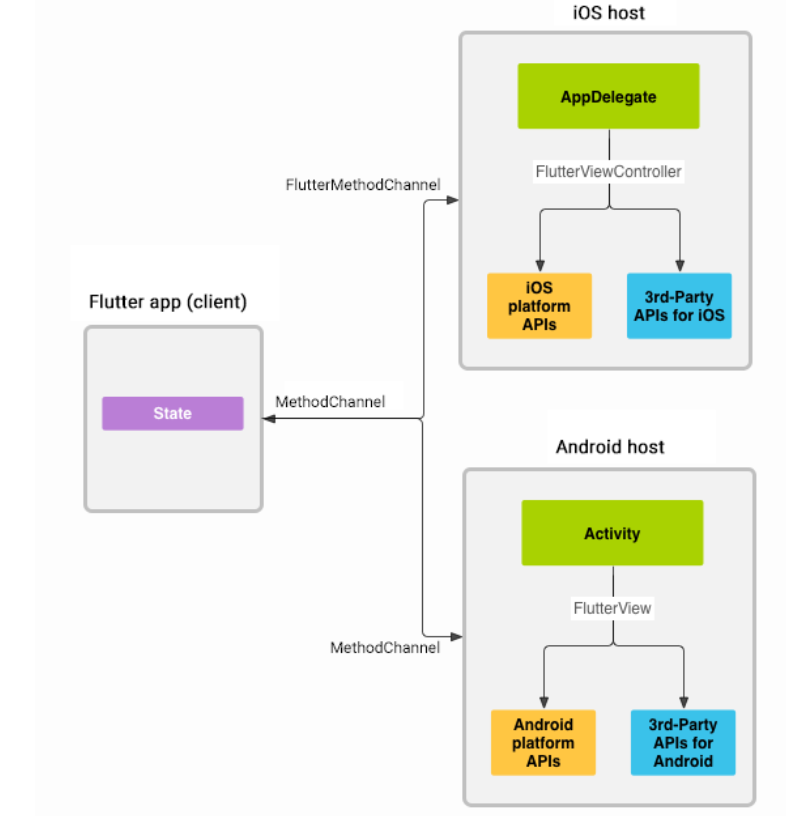
* + - Kotlin hoặc Java trên Android
    - Swift hoặc Objective-C trên iOS
    - C++ trên Windows
    - Mục tiêu-C trên macOS
    - C trên Linux

Hỗ trợ API dành riêng cho nền tảng dựng sẵn của Flutter không dựa vào việc tạo mã, mà dựa vào kiểu truyền thông báo linh hoạt. Ngoài ra, bạn có thể sử dụng gói Pigeon để gửi các tin nhắn an toàn có cấu trúc bằng cách tạo mã:

* Phần Flutter của ứng dụng gửi tin nhắn đến máy chủ của nó , phần không phải Dart của ứng dụng, qua kênh nền tảng.
* Máy chủ lắng nghe trên kênh nền tảng và nhận tin nhắn. Sau đó, nó gọi vào bất kỳ số lượng API dành riêng cho nền tảng nào—sử dụng ngôn ngữ lập trình gốc—và gửi phản hồi lại cho ứng dụng khách , phần Flutter của ứng dụng.

**Tổng quan về kiến trúc: các kênh nền tảng**

Tin nhắn được truyền giữa máy khách (UI) và máy chủ (nền tảng) bằng cách sử dụng các kênh nền tảng như được minh họa trong sơ đồ này:



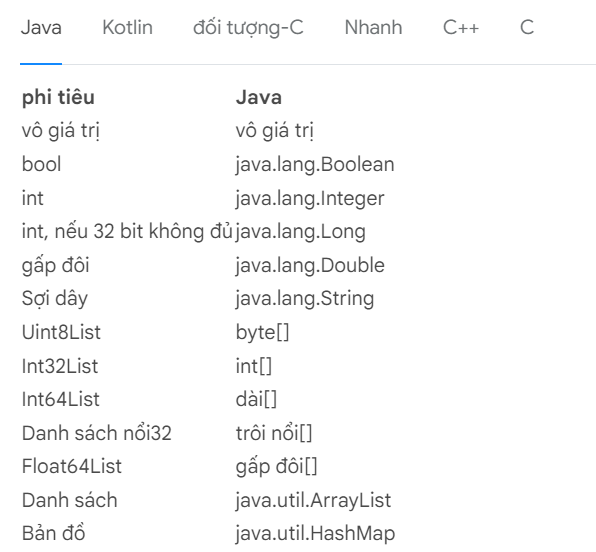
Tin nhắn và phản hồi được truyền không đồng bộ để đảm bảo giao diện người dùng vẫn phản hồi nhanh.

Về phía máy khách, MethodChannelcho phép gửi tin nhắn tương ứng với các cuộc gọi phương thức. Về phía nền tảng, MethodChanneltrên Android ( MethodChannelAndroid) và FlutterMethodChanneltrên iOS ( MethodChanneliOS) cho phép nhận cuộc gọi phương thức và gửi lại kết quả. Các lớp này cho phép bạn phát triển plugin nền tảng với rất ít mã 'bản soạn sẵn'.

**Hỗ trợ các loại dữ liệu kênh nền tảng và codec**

Các kênh nền tảng tiêu chuẩn sử dụng một codec thông báo tiêu chuẩn hỗ trợ tuần tự hóa nhị phân hiệu quả của các giá trị giống như JSON đơn giản, chẳng hạn như booleans, số, Chuỗi, bộ đệm byte cũng như Danh sách và Bản đồ của những giá trị này (xem để biết chi tiết) StandardMessageCodec. Quá trình tuần tự hóa và giải tuần tự hóa các giá trị này đến và từ các thư diễn ra tự động khi bạn gửi và nhận các giá trị.

Bảng sau đây cho biết cách nhận các giá trị Dart ở phía nền tảng và ngược lại:

****

**Ví dụ: Gọi mã dành riêng cho nền tảng bằng các kênh nền tảng**

Đoạn mã sau trình bày cách gọi API dành riêng cho nền tảng để truy xuất và hiển thị mức pin hiện tại. Nó sử dụng API Android BatteryManager, API iOS device.batteryLevel, API Windows GetSystemPowerStatusvà UPowerAPI Linux với một thông báo nền tảng duy nhất, getBatteryLevel().

Ví dụ thêm mã dành riêng cho nền tảng bên trong chính ứng dụng chính. Nếu bạn muốn sử dụng lại mã dành riêng cho nền tảng cho nhiều ứng dụng, thì bước tạo dự án sẽ hơi khác một chút (xem phần phát triển gói ), nhưng mã kênh của nền tảng vẫn được viết theo cùng một cách

**Bước 1: Tạo một dự án ứng dụng mới**

Bắt đầu bằng cách tạo một ứng dụng mới:

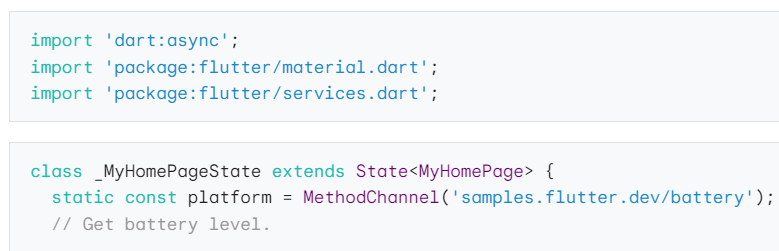
* Trong một thiết bị đầu cuối chạy:flutter create batterylevel
  + - * Theo mặc định, mẫu của chúng tôi hỗ trợ viết mã Android bằng Kotlin hoặc mã iOS bằng Swift. Để sử dụng Java hoặc Objective-C, hãy sử dụng các cờ -I và/hoặc -a:
      * Trong một thiết bị đầu cuối chạy:flutter create -i objc -a java batterylevel

**Bước 2: Tạo ứng dụng khách nền tảng Flutter**

Lớp của ứng dụng Stategiữ trạng thái ứng dụng hiện tại. Mở rộng điều đó để giữ trạng thái pin hiện tại.

Đầu tiên, xây dựng kênh. Sử dụng một MethodChannelphương thức nền tảng duy nhất trả về mức pin.

Các phía máy khách và máy chủ của kênh được kết nối thông qua tên kênh được truyền trong hàm tạo kênh. Tất cả các tên kênh được sử dụng trong một ứng dụng phải là duy nhất; thêm vào đầu tên kênh một 'tiền tố tên miền' duy nhất, ví dụ: samples.flutter.dev/battery.



Tiếp theo, gọi một phương thức trên kênh phương thức, chỉ định phương thức cụ thể để gọi bằng Stringmã định danh getBatteryLevel. Cuộc gọi có thể không thành công—ví dụ: nếu nền tảng không hỗ trợ API nền tảng (chẳng hạn như khi chạy trong trình giả lập), vì vậy hãy kết thúc cuộc invokeMethodgọi trong câu lệnh thử bắt.

Sử dụng kết quả trả về để cập nhật trạng thái giao diện người dùng bên \_batteryLevel trong setState

A screen shot of a computer program

Description automatically generated

Cuối cùng, thay thế buildphương thức từ mẫu để chứa một giao diện người dùng nhỏ hiển thị trạng thái pin trong một chuỗi và một nút để làm mới giá trị.

A screen shot of a computer program

Description automatically generated

**Bước 3: Thêm triển khai dành riêng cho nền tảng Android**

**Kotlin**

Bắt đầu bằng cách mở phần máy chủ Android của ứng dụng Flutter của bạn trong Android Studio:

1. Bắt đầu Android Studio

2. Chọn mục menu Tệp > Mở…

3. Điều hướng đến thư mục chứa ứng dụng Flutter của bạn và chọn thư mục android bên trong nó. Bấm OK .

4. Mở tệp MainActivity.ktnằm trong thư mục kotlin trong chế độ xem Dự án.

Bên trong configureFlutterEngine()phương thức, tạo một MethodChannelvà gọi setMethodCallHandler(). Đảm bảo sử dụng cùng một tên kênh như đã được sử dụng ở phía máy khách Flutter.

A screenshot of a computer program

Description automatically generated

Thêm mã Android Kotlin sử dụng API pin Android để truy xuất mức pin. Mã này hoàn toàn giống với mã bạn viết trong ứng dụng Android gốc.

Đầu tiên, thêm các mục nhập cần thiết ở đầu tệp:

A screenshot of a computer program

Description automatically generated

Tiếp theo, thêm phương thức sau vào MainActivitylớp, bên dưới configureFlutterEngine()phương thức:

A screenshot of a computer program

Description automatically generated

Bây giờ bạn có thể chạy ứng dụng trên Android. Nếu sử dụng Trình giả lập Android, hãy đặt mức pin trong bảng Điều khiển mở rộng có thể truy cập từ nút … trên thanh công cụ.

**Bước 4: Thêm triển khai dành riêng cho nền tảng iOS**

Bắt đầu bằng cách mở phần máy chủ iOS của ứng dụng Flutter của bạn trong Xcode:

1. Bắt đầu Xcode.

2. Chọn mục menu Tệp > Mở… .

3. Điều hướng đến thư mục chứa ứng dụng Flutter của bạn và chọn thư mục ios bên trong nó. Bấm OK .

Thêm hỗ trợ cho Swift trong thiết lập mẫu tiêu chuẩn sử dụng Objective-C:

1. Mở rộng Runner > Runner trong Project navigator.

2. Mở tệp AppDelegate.swiftnằm trong Runner > Runner trong Project navigator.

Ghi đè application:didFinishLaunchingWithOptions:chức năng và tạo một FlutterMethodChannelliên kết với tên kênh samples.flutter.dev/battery:

A screenshot of a computer program

Description automatically generated

Tiếp theo, thêm mã iOS Swift sử dụng API pin iOS để truy xuất mức pin. Mã này hoàn toàn giống với mã bạn viết trong ứng dụng iOS gốc.

Thêm phần sau đây dưới dạng một phương thức mới ở cuối AppDelegate.swift:

A screen shot of a computer code

Description automatically generated

Cuối cùng, hoàn thành setMethodCallHandler()phương pháp đã thêm trước đó. Bạn cần xử lý một phương thức nền tảng duy nhất, getBatteryLevel()vì vậy hãy kiểm tra điều đó trong callđối số. Việc triển khai phương thức nền tảng này gọi mã iOS được viết ở bước trước. Nếu một phương thức không xác định được gọi, thay vào đó hãy báo cáo phương thức đó.

**Bước 5: Thêm triển khai dành riêng cho nền tảng Windows**

Bắt đầu bằng cách mở phần máy chủ Windows của ứng dụng Flutter của bạn trong Visual Studio:

1. Chạy flutter build windowstrong thư mục dự án của bạn một lần để tạo tệp giải pháp Visual Studio.

2. Bắt đầu Visual Studio.

3. Chọn Mở một dự án hoặc giải pháp .

4. Điều hướng đến thư mục chứa ứng dụng Flutter của bạn, sau đó vào thư mục bản dựng , sau đó đến thư mục windows , sau đó chọn batterylevel.slntệp. Nhấp vào Mở .

Thêm triển khai C++ của phương thức kênh nền tảng:

1. Mở rộng batterylevel > Source Files trong Solution Explorer.

2. Mở tệp flutter\_window.cpp.

Đầu tiên, thêm các mục cần thiết vào đầu tệp, ngay sau #include "flutter\_window.h":

A screen shot of a computer code

Description automatically generated

Chỉnh sửa FlutterWindow::OnCreatephương thức và tạo một flutter::MethodChannelliên kết với tên kênh samples.flutter.dev/battery