1. Storage

thường đề cập đến việc quản lý tính bền vững của dữ liệu trong các ứng dụng web và di động. Flutter cung cấp một số cơ chế lưu trữ và truy xuất dữ liệu để duy trì trạng thái ứng dụng, tùy chọn người dùng và thông tin được lưu trong bộ nhớ cache. Dưới đây là một số phương pháp lưu trữ phổ biến trong Flutter:

* SharedPreferences:

SharedPreferences là giải pháp lưu trữ khóa-giá trị để duy trì lượng dữ liệu nhỏ, chẳng hạn như tùy chọn và cài đặt của người dùng. Nó đơn giản để sử dụng và không yêu cầu phụ thuộc bên ngoài. SharedPreferences lưu trữ dữ liệu cục bộ trên thiết bị.

* Cơ sở dữ liệu cục bộ:

Flutter hỗ trợ cơ sở dữ liệu cục bộ thông qua các gói như sqflitevà moor. Các gói này cung cấp tích hợp cơ sở dữ liệu SQLite, cho phép bạn lưu trữ dữ liệu có cấu trúc, thực hiện các truy vấn và quản lý các mối quan hệ dữ liệu trong ứng dụng Flutter của mình.

* Lưu trữ tệp:

Flutter cung cấp các phương thức để đọc và ghi vào tệp. Điều này có thể hữu ích để lưu trữ dữ liệu như nhật ký, tệp cấu hình và nội dung được lưu trong bộ nhớ cache.

* Hive:

Hive là một cơ sở dữ liệu NoSQL nhẹ, dành cho Flutter, cung cấp tính ổn định dữ liệu nhanh với sự hỗ trợ cho cả cặp khóa-giá trị và cấu trúc dữ liệu phức tạp hơn. Nó được biết đến với hiệu suất và sự đơn giản của nó.

* Cơ sở dữ liệu thời gian thực Firebase và Firestore:

Firebase là một giải pháp phụ trợ phổ biến do Google cung cấp. Cơ sở dữ liệu thời gian thực Firebase và Firestore cung cấp các tùy chọn lưu trữ dựa trên đám mây, cho phép bạn lưu trữ và đồng bộ hóa dữ liệu trong thời gian thực trên các thiết bị.

* Lưu hình ảnh và tệp vào bộ đệm:

Các ứng dụng rung thường cần lưu hình ảnh và các nội dung khác vào bộ đệm để tải nhanh hơn. Các gói như flutter\_cache\_managergiúp quản lý bộ nhớ đệm của tài nguyên mạng.

Lưu trữ an toàn:

Khi xử lý thông tin nhạy cảm như mã thông báo xác thực hoặc mật khẩu, bạn có thể sử dụng các gói như flutter\_secure\_storageđể lưu trữ dữ liệu theo cách được mã hóa và bảo mật.

Quản lý trạng thái:

Mặc dù không chính xác là lưu trữ, các giải pháp quản lý trạng thái như Nhà cung cấp, Riverpod, Redux hoặc MobX giúp quản lý và chia sẻ trạng thái ứng dụng trên các phần khác nhau của ứng dụng Flutter của bạn.

Khi chọn phương pháp lưu trữ cho ứng dụng Flutter của bạn, hãy xem xét các yếu tố như loại dữ liệu bạn cần lưu trữ, kích thước của dữ liệu, yêu cầu về hiệu suất và mối quan tâm về bảo mật. Tùy thuộc vào nhu cầu của ứng dụng, bạn có thể sử dụng kết hợp các phương pháp lưu trữ này để quản lý và duy trì dữ liệu một cách hiệu quả.

1. 3rd party

Trong bối cảnh phát triển Flutter, "3rd party " thường đề cập đến gói hoặc thư viện của bên thứ ba không phải là một phần của khung Flutter cốt lõi nhưng có thể được thêm vào dự án Flutter của bạn để mở rộng chức năng hoặc hợp lý hóa quá trình phát triển. Flutter có một hệ sinh thái phong phú gồm các gói của bên thứ ba mà các nhà phát triển có thể sử dụng để tiết kiệm thời gian và công sức trong việc xây dựng các tính năng khác nhau trong ứng dụng của họ.

Đây là cách bạn có thể tích hợp và sử dụng các gói của bên thứ ba trong dự án Flutter của mình:

* Tìm gói:

Bạn có thể tìm các gói của bên thứ ba trên trang web Gói Dart , đây là kho lưu trữ gói chính thức cho Dart và Flutter. Trang web này lưu trữ nhiều loại gói cho các mục đích khác nhau, chẳng hạn như các thành phần giao diện người dùng, kết nối mạng, quản lý trạng thái, v.v.

* Thêm gói:

Để thêm gói của bên thứ ba vào dự án Flutter của bạn, bạn cần đưa gói đó làm phần phụ thuộc vào pubspec.yamltệp của mình. Mở tệp này và thêm tên gói và phiên bản trong dependencies.

* Nhập và sử dụng các gói:

Khi bạn đã thêm gói vào dự án của mình, bạn có thể nhập các lớp của nó và sử dụng các chức năng của nó trong mã của mình. Thông thường, các gói cung cấp tài liệu và ví dụ về cách sử dụng các tính năng của chúng. Nhập gói vào tệp Dart của bạn bằng cách sử dụng import 'package:some\_package/some\_package.dart';rồi sử dụng các lớp và hàm của gói.

* Xử lý các phần phụ thuộc:

Đôi khi, các gói của bên thứ ba có thể có các phần phụ thuộc của riêng chúng. Các phụ thuộc này được liệt kê trong tài liệu của gói. Đảm bảo kiểm tra tài liệu để biết bất kỳ thiết lập đặc biệt hoặc cấu hình bổ sung nào được yêu cầu khi sử dụng gói.

* Lập phiên bản:

Chỉ định phạm vi phiên bản của gói trong pubspec.yamltệp của bạn bằng cách sử dụng các ràng buộc phiên bản như ^, >=v.v. Điều này giúp đảm bảo rằng ứng dụng của bạn sử dụng phiên bản tương thích của gói và không bị lỗi với các bản cập nhật trong tương lai.

* Cập nhật gói:

Khi các phiên bản mới của gói được phát hành, bạn có thể muốn cập nhật chúng để truy cập các tính năng mới hoặc sửa lỗi. Chạy flutter pub upgradeđể cập nhật các gói trong dự án của bạn lên phiên bản tương thích mới nhất của chúng.

1. Behavior

" Behavior" đề cập đến cách các phần tử giao diện người dùng (widget) phản ứng với các tương tác, sự kiện và thay đổi trạng thái của người dùng. Hành vi trong Flutter liên quan đến việc xác định cách các widget phản ứng với các đầu vào và điều kiện khác nhau, tạo ra trải nghiệm người dùng tương tác và hấp dẫn. Hãy khám phá một số khía cạnh phổ biến của hành vi trong Flutter:

* Tương tác người dùng:

Hành vi tương tác của người dùng liên quan đến việc xác định cách tiện ích con phản hồi với các hành động của người dùng như chạm, vuốt, kéo, v.v. Ví dụ: bạn có thể sử dụng GestureDetectortiện ích để phát hiện các cử chỉ khác nhau và kích hoạt các hành động cụ thể khi những cử chỉ đó xảy ra.

* Xử lý sự kiện:

Các tiện ích rung có thể xử lý các sự kiện bằng cách chỉ định các chức năng gọi lại được kích hoạt khi các sự kiện xảy ra. Chẳng hạn, onPressedthuộc tính của a FlatButtonhoặc ElevatedButtoncó thể được sử dụng để xác định hành vi khi nhấn nút.

* Quản lý trạng thái:

Hành vi thường xoay quanh việc quản lý và cập nhật trạng thái của các vật dụng. Quản lý trạng thái xác định cách các widget phản ứng với những thay đổi về dữ liệu hoặc điều kiện. Bạn có thể sử dụng phương pháp tích hợp sẵn của Flutter setStateđể thay đổi trạng thái đơn giản hoặc chọn các giải pháp quản lý trạng thái nâng cao hơn như mẫu Nhà cung cấp, Riverpod hoặc BLoC cho các ứng dụng phức tạp hơn.

* Hoạt ảnh và chuyển tiếp:

Flutter cung cấp một khung hoạt ảnh mạnh mẽ cho phép bạn tạo hành vi năng động và hấp dẫn. Bạn có thể xác định hoạt ảnh để thay đổi các thuộc tính của tiện ích con theo thời gian, tạo các hiệu ứng và hiệu ứng chuyển tiếp mượt mà. Lớp Animationvà các tiện ích như AnimatedContainer, Herovà SlideTransitionđược sử dụng cho mục đích này.

* Tiêu điểm và hành vi bàn phím:

Các tiện ích liên quan đến đầu vào của người dùng, như trường văn bản, có hành vi cụ thể liên quan đến tiêu điểm, không tiêu điểm và tương tác với bàn phím. Bạn có thể sử dụng các tiện ích Focusvà FocusScopeđể quản lý hành vi tiêu điểm và tiện ích này TextFieldcó các thuộc tính để kiểm soát tương tác bàn phím.

* Điều hướng và định tuyến:

Hành vi điều hướng rất quan trọng để di chuyển giữa các màn hình khác nhau trong ứng dụng của bạn. Bạn có thể sử dụng Navigatorlớp để quản lý điều hướng và xác định các tuyến xác định cách chuyển đổi giữa các màn hình. Các tiện ích như Navigator.pushvà Navigator.popđược sử dụng để điều hướng giữa các tuyến đường.

* Hành vi cuộn:

Các tiện ích chứa nội dung có thể cuộn, chẳng hạn như danh sách hoặc lưới, có hành vi liên quan đến việc cuộn. Bạn có thể sử dụng các tiện ích như ListView, GridViewhoặc CustomScrollViewđể tạo các dạng xem có thể cuộn và kiểm soát cách các mục được hiển thị và cuộn.

* Hộp thoại và Phương thức:

Hành vi liên quan đến hộp thoại và phương thức liên quan đến việc hiển thị các cửa sổ bật lên yêu cầu tương tác của người dùng. Flutter cung cấp các tiện ích hộp thoại khác nhau như AlertDialog, SimpleDialogvà BottomSheetđể quản lý các tình huống này.

Hãy nhớ rằng kiến ​​trúc dựa trên widget của Flutter cho phép bạn tạo hành vi bằng cách kết hợp các widget khác nhau và thuộc tính của chúng. Cách tiếp cận "khai báo" của Flutter có nghĩa là bạn mô tả hành vi và giao diện mong muốn của giao diện người dùng của mình bằng cách sử dụng các tiện ích con và Flutter sẽ đảm nhận việc cập nhật giao diện người dùng để phản ánh các thay đổi về hành vi hoặc trạng thái.

1. Sate Managements

Trong Flutter, Sate Managementslà một khái niệm quan trọng liên quan đến cách dữ liệu (hoặc trạng thái) của ứng dụng được chia sẻ, cập nhật và đồng bộ hóa trên các phần khác nhau của giao diện người dùng. Flutter cung cấp nhiều cách tiếp cận khác nhau để quản lý trạng thái, mỗi cách phục vụ cho các yêu cầu và độ phức tạp khác nhau của ứng dụng của bạn. Dưới đây là một số kỹ thuật quản lý trạng thái phổ biến trong Flutter:

* StatefulWidget : Đây là hình thức quản lý trạng thái đơn giản nhất. Các tiện ích mở rộng StatefulWidgetcó trạng thái có thể thay đổi có thể thay đổi theo thời gian. Khi trạng thái thay đổi, tiện ích được xây dựng lại, cập nhật giao diện người dùng. Cách tiếp cận này phù hợp để quản lý trạng thái cục bộ trong một widget.
* Nhà cung cấp : Nhà cung cấp là gói phổ biến để quản lý trạng thái tuân theo mẫu "InheritedWidget". Nó cho phép bạn chia sẻ trạng thái trên cây widget và tự động xây dựng lại các widget khi trạng thái chia sẻ thay đổi. Nhà cung cấp cung cấp các loại nhà cung cấp khác nhau, chẳng hạn như ChangeNotifierProvider, StreamProvider, v.v.
* Khối (Thành phần logic nghiệp vụ) : Mẫu BLoC tách logic nghiệp vụ khỏi giao diện người dùng, giúp kiểm tra và quản lý các trạng thái phức tạp dễ dàng hơn. Nó liên quan đến việc tạo các luồng sự kiện và trạng thái, cho phép các tiện ích con phản hồi các sự kiện và cập nhật giao diện người dùng của chúng cho phù hợp. Các gói thích flutter\_blocvà blocgiúp thực hiện mẫu này.
* Redux : Redux là một mẫu quản lý trạng thái tập trung trạng thái ứng dụng trong một cửa hàng duy nhất. Các hành động kích hoạt thay đổi trạng thái và bộ giảm tốc xác định cách sửa đổi trạng thái để đáp ứng với các hành động đó. Gói flutter\_reduxcó thể được sử dụng để tích hợp Redux vào Flutter.
* GetX : GetX là gói cung cấp một loạt các tiện ích, bao gồm cả quản lý trạng thái. Nó cung cấp khả năng quản lý trạng thái phản ứng, quản lý tuyến đường và tiêm phụ thuộc, tất cả trong một gói.
* Riverpod : Riverpod là giải pháp thay thế cho Nhà cung cấp giúp đơn giản hóa việc quản lý trạng thái, phần phụ thuộc và dữ liệu không đồng bộ. Nó sử dụng một cách tiếp cận linh hoạt hơn và khuyến khích một thiết kế mô đun hơn.
* MobX : MobX là một thư viện quản lý trạng thái tập trung vào việc làm cho việc quản lý trạng thái trở nên đơn giản và trực quan. Nó sử dụng các quan sát và phản ứng để tự động theo dõi các thay đổi và cập nhật giao diện người dùng cho phù hợp.
* ScopedModel : ScopedModel là một cách đơn giản để quản lý trạng thái bằng cách xác định phạm vi trạng thái cho các phần cụ thể của cây widget. Nó tốt cho các ứng dụng đơn giản hơn nhưng có thể trở nên khó bảo trì hơn đối với các ứng dụng phức tạp hơn.
* GetX : GetX là gói cung cấp giải pháp quản lý trạng thái nhẹ và linh hoạt. Nó cung cấp quản lý trạng thái phản ứng, tiêm phụ thuộc và quản lý điều hướng.
* Cơ sở dữ liệu thời gian thực Firebase / Firestore : Đối với các ứng dụng dựa trên đám mây, bạn có thể sử dụng cơ sở dữ liệu thời gian thực của Firebase hoặc Firestore để xử lý đồng bộ hóa dữ liệu theo thời gian thực giữa các thiết bị và đám mây.

Việc chọn kỹ thuật quản lý trạng thái phù hợp tùy thuộc vào các yếu tố như độ phức tạp của ứng dụng, mức độ quen thuộc của bạn với các mẫu và sở thích cá nhân của bạn. Mỗi cách tiếp cận đều có điểm mạnh và điểm yếu, vì vậy điều cần thiết là phải hiểu các sắc thái của từng cách và chọn cách phù hợp nhất với nhu cầu của dự án của bạn.

1. Quality Assurance

Quality Assurance (QA) trong Flutter liên quan đến việc áp dụng các nguyên tắc và thực tiễn của QA để đảm bảo chất lượng và độ tin cậy của các ứng dụng Flutter. Cũng giống như bất kỳ hoạt động phát triển phần mềm nào, QA trong Flutter nhằm mục đích xác định và ngăn ngừa lỗi, đảm bảo ứng dụng đáp ứng yêu cầu của người dùng và mang lại trải nghiệm người dùng chất lượng cao. Đây là cách các thực hành QA có thể được áp dụng để phát triển Flutter:

* Các loại thử nghiệm trong Flutter :

Kiểm tra tiện ích : Điều này liên quan đến việc kiểm tra các tiện ích riêng lẻ và các tương tác của chúng một cách riêng biệt. Gói này flutter\_testcung cấp các công cụ để viết các bài kiểm tra tiện ích nhằm đảm bảo các tiện ích hiển thị chính xác và phản hồi các tương tác của người dùng như mong đợi.

Kiểm tra tích hợp : Các kiểm tra này tập trung vào việc xác minh sự tương tác giữa các thành phần khác nhau của ứng dụng. Chúng đảm bảo rằng các widget, quản lý trạng thái và các thành phần khác hoạt động hài hòa với nhau.

Kiểm tra đơn vị : Kiểm tra đơn vị nhắm mục tiêu các chức năng, phương thức hoặc lớp riêng lẻ để đảm bảo rằng chúng hoạt động như mong đợi. Các thư viện thích testvà flutter\_testthường được sử dụng để viết bài kiểm tra đơn vị.

Thử nghiệm từ đầu đến cuối : Các thử nghiệm này mô phỏng các tương tác của người dùng trên toàn bộ ứng dụng, đảm bảo rằng hành trình của người dùng hoạt động như dự kiến. Gói của Flutter flutter\_drivercho phép thử nghiệm từ đầu đến cuối.

* Kiểm thử tự động :

Flutter cung cấp các công cụ và gói để viết các bài kiểm tra tự động, bao gồm kiểm tra đơn vị, kiểm tra tiện ích và kiểm tra tích hợp. Các thử nghiệm này có thể được tự động hóa để chạy như một phần của quá trình xây dựng và triển khai, phát hiện sớm các vấn đề trong chu kỳ phát triển.

* Tích hợp liên tục (CI) và Triển khai liên tục (CD) :

Việc tích hợp các ứng dụng Flutter vào đường dẫn CI/CD đảm bảo rằng các thử nghiệm tự động được thực hiện bất cứ khi nào thay đổi mã được thực hiện. Điều này giúp nắm bắt các hồi quy và lỗi trước khi chúng được đưa vào sản xuất.

* Đánh giá mã :

Đánh giá mã ngang hàng là điều cần thiết để duy trì chất lượng mã. Người đánh giá có thể giúp xác định các vấn đề tiềm ẩn, đề xuất cải tiến và đảm bảo rằng mã tuân thủ các phương pháp hay nhất.

* Kiểm tra hiệu suất :

Sử dụng các công cụ như kiểm tra tích flutter\_driverhợp và kiểm tra tích hợp để đánh giá các chỉ số hiệu suất, chẳng hạn như tốc độ phản hồi và tốc độ kết xuất của ứng dụng. Bạn có thể mô phỏng các tình huống khác nhau để đảm bảo ứng dụng hoạt động tốt trong các điều kiện khác nhau

* Kiểm tra khả năng sử dụng :

Mặc dù kiểm tra khả năng sử dụng thường liên quan đến người dùng thực, các nhà phát triển và kỹ sư QA cũng có thể tiến hành kiểm tra khả năng sử dụng nội bộ để đảm bảo rằng giao diện người dùng của ứng dụng trực quan và thân thiện với người dùng.

* Kiểm tra thiết bị và nền tảng :

Flutter hỗ trợ xây dựng ứng dụng cho nhiều nền tảng khác nhau, bao gồm iOS, Android và web. Các kỹ sư QA nên thử nghiệm ứng dụng trên các thiết bị, kích thước màn hình và phiên bản hệ điều hành khác nhau để đảm bảo hiệu suất và giao diện nhất quán.

* Thử nghiệm bản địa hóa và quốc tế hóa :

Xác minh rằng các thành phần giao diện người dùng và văn bản của ứng dụng hiển thị chính xác bằng các ngôn ngữ và nền văn hóa khác nhau.

* Kiểm tra khả năng tiếp cận

Đảm bảo người dùng khuyết tật có thể truy cập ứng dụng bằng cách sử dụng các công cụ như trình đọc màn hình và công cụ kiểm tra khả năng truy cập tự động.

* Kiểm tra bảo mật :

Áp dụng các phương pháp kiểm tra bảo mật để xác định các lỗ hổng và đảm bảo rằng dữ liệu nhạy cảm được xử lý an toàn.

* Xử lý lỗi và kiểm tra các trường hợp biên :

Kiểm tra cách ứng dụng hoạt động trong các trường hợp khó khăn, chẳng hạn như kết nối mạng kém, đầu vào không mong muốn và các tình huống cực đoan.

Bằng cách triển khai các thực hành QA này trong quá trình phát triển Flutter, bạn có thể nâng cao chất lượng, độ ổn định và độ tin cậy của các ứng dụng Flutter của mình, mang lại trải nghiệm người dùng tốt hơn và ít sự cố sau phát hành hơn.

1. Version Control

Version Control là một khía cạnh thiết yếu của quá trình phát triển phần mềm, cho phép nhiều nhà phát triển cộng tác trong một dự án đồng thời theo dõi các thay đổi, quản lý các phiên bản khác nhau và đảm bảo quy trình làm việc trôi chảy. Flutter, một bộ công cụ giao diện người dùng của Google, có thể được tích hợp với các hệ thống kiểm soát phiên bản như Git để quản lý các dự án Flutter của bạn một cách hiệu quả.

Đây là cách bạn có thể thiết lập kiểm soát phiên bản cho dự án Flutter bằng Git:

* Cài đặt Git :

Nếu bạn chưa có, hãy cài đặt Git trên máy của bạn. Bạn có thể tải xuống từ trang web chính thức của Git ( https://git-scm.com/ ).

* Tạo Kho lưu trữ Git :

Mở một thiết bị đầu cuối hoặc dấu nhắc lệnh.

Điều hướng đến thư mục gốc của dự án Flutter của bạn bằng cdlệnh.

Chạy git initđể khởi tạo kho lưu trữ Git mới trong thư mục dự án của bạn.

* Tạo một .gitignoretập tin :

Tạo một tệp có tên .gitignoretrong thư mục gốc của dự án của bạn.

Bên trong tệp này, liệt kê các tệp và thư mục bạn muốn Git bỏ qua (ví dụ: tệp xây dựng, tệp tạm thời, tệp dành riêng cho IDE). Điều này ngăn các tệp không cần thiết bị Git theo dõi và thêm vào kho lưu trữ.

* Thêm và cam kết thay đổi :

Sau khi thực hiện các thay đổi đối với dự án của bạn, hãy sử dụng git addlệnh để phân chia các thay đổi mà bạn muốn thực hiện.

Chạy git commit -m "Your commit message here"để thực hiện các thay đổi theo giai đoạn đối với kho lưu trữ.

* Tạo Kho lưu trữ từ xa (Tùy chọn) :

Bạn có thể tạo kho lưu trữ từ xa trên các nền tảng như GitHub, GitLab hoặc Bitbucket để lưu trữ dự án của mình trực tuyến và cho phép cộng tác với những người khác.

* Liên kết kho lưu trữ từ xa (Tùy chọn) :

Thực hiện theo các hướng dẫn được cung cấp bởi nền tảng kho lưu trữ từ xa để liên kết kho lưu trữ cục bộ của bạn với kho lưu trữ từ xa. Điều này thường liên quan đến việc thêm một URL từ xa bằng git remote add origin <remote\_repository\_url>lệnh.

* Thay đổi đẩy :

Sau khi được liên kết với kho lưu trữ từ xa, bạn có thể sử dụng git pushlệnh để tải các cam kết cục bộ của mình lên kho lưu trữ từ xa. Điều này làm cho mã của bạn có thể truy cập được đối với những người khác và cho phép phát triển hợp tác.

* Thay đổi kéo :

Khi các cộng tác viên khác thực hiện thay đổi và đẩy chúng vào kho lưu trữ từ xa, bạn có thể sử dụng git pulllệnh để tìm nạp và hợp nhất những thay đổi đó vào kho lưu trữ cục bộ của mình.

1. Firebase

Firebase là một nền tảng toàn diện do Google cung cấp, cung cấp nhiều công cụ và dịch vụ khác nhau để xây dựng và mở rộng các ứng dụng web và thiết bị di động. Nó bao gồm một loạt các chức năng, bao gồm dịch vụ phụ trợ, quản lý cơ sở dữ liệu, xác thực, lưu trữ, phân tích, v.v. Firebase thường được sử dụng kết hợp với Flutter để tạo các ứng dụng mạnh mẽ và giàu tính năng.

Dưới đây là một số tính năng và dịch vụ chính do Firebase cung cấp thường được tích hợp với các ứng dụng Flutter:

* Xác thực Firebase : Cung cấp các giải pháp xác thực dễ triển khai, bao gồm email/mật khẩu, thông tin đăng nhập mạng xã hội (Google, Facebook, Twitter, v.v.), v.v.
* Cloud Firestore : Cơ sở dữ liệu NoSQL để lưu trữ và đồng bộ hóa dữ liệu trên các thiết bị. Nó cung cấp tính năng đồng bộ hóa theo thời gian thực và rất phù hợp cho các ứng dụng yêu cầu cập nhật theo thời gian thực.
* Cơ sở dữ liệu thời gian thực Firebase : Một tùy chọn cơ sở dữ liệu thời gian thực khác cho phép bạn lưu trữ và đồng bộ hóa dữ liệu ở định dạng JSON. Nó phù hợp cho các ứng dụng yêu cầu cộng tác và đồng bộ hóa thời gian thực.
* Firebase Cloud Function : Cho phép bạn chạy mã phía máy chủ để đáp ứng các sự kiện như thay đổi dữ liệu, yêu cầu HTTP và trình kích hoạt xác thực.
* Lưu trữ Firebase : Cung cấp dịch vụ lưu trữ an toàn và nhanh chóng cho các ứng dụng web, bao gồm các ứng dụng một trang được xây dựng bằng Flutter.
* Lưu trữ Firebase : Cung cấp lưu trữ dựa trên đám mây cho nội dung do người dùng tạo như hình ảnh, video và các tệp khác.
* Firebase Cloud Messaging (FCM) : Cho phép bạn gửi thông báo và tin nhắn cho người dùng trên các nền tảng khác nhau.
* Firebase Analytics : Cung cấp thông tin chi tiết về hành vi, mức độ tương tác và hiệu suất của ứng dụng của người dùng.
* Firebase Remote Config : Cho phép bạn tùy chỉnh và cá nhân hóa hành vi của ứng dụng mà không cần xuất bản bản cập nhật ứng dụng.
* Giám sát hiệu suất Firebase : Giúp bạn theo dõi và tối ưu hóa hiệu suất của ứng dụng.
* Phòng thí nghiệm thử nghiệm Firebase : Cung cấp cơ sở hạ tầng thử nghiệm dựa trên đám mây để thử nghiệm ứng dụng của bạn trên nhiều thiết bị và cấu hình thực.

1. Native Integration

Khi làm việc với Flutter, có thể có những trường hợp bạn cần tích hợp mã gốc vào ứng dụng Flutter của mình. Điều này có thể liên quan đến việc sử dụng các tính năng, thư viện hoặc API nền tảng gốc không được khung đa nền tảng của Flutter hỗ trợ trực tiếp. Flutter cung cấp các cơ chế để thu hẹp khoảng cách giữa mã Dart của Flutter và mã gốc được viết bằng Java/Kotlin cho Android và Objective-C/Swift cho iOS.

Dưới đây là tổng quan chung về cách tích hợp gốc hoạt động trong Flutter:

* Kênh nền tảng : Flutter cung cấp một tính năng gọi là "kênh nền tảng" cho phép giao tiếp giữa mã Dart và mã gốc. Các kênh nền tảng cho phép bạn gửi và nhận tin nhắn cũng như dữ liệu giữa lớp Flutter và lớp gốc.
* Kênh phương thức : Kênh phương thức là một loại kênh nền tảng cho phép bạn gọi các phương thức trong mã gốc từ mã Flutter Dart của bạn. Bạn xác định các phương thức trong cả mã Flutter và mã gốc, đồng thời bạn có thể truyền đối số và nhận kết quả.
* Kênh sự kiện : Kênh sự kiện cho phép giao tiếp từ mã gốc sang mã Flutter Dart cho các tình huống mà bạn cần lắng nghe các sự kiện trong mã gốc và xử lý chúng trong Flutter.

Đây là quy trình từng bước để tích hợp mã gốc vào ứng dụng Flutter của bạn:

* Tạo mã gốc :

Viết mã gốc bằng Java/Kotlin cho Android hoặc Objective-C/Swift cho iOS.

Triển khai các phương thức bạn muốn gọi từ Flutter và, nếu cần, hãy thiết lập trình xử lý sự kiện sẽ giao tiếp với Flutter.

* Thiết lập kênh phương thức và sự kiện :

Trong mã Flutter Dart của bạn, hãy thiết lập các kênh phương thức và sự kiện bằng cách sử dụng các lớp MethodChannelvà EventChanneltừ flutter/servicesgói.

Xác định các phương thức bạn sẽ gọi từ mã gốc và chỉ định cách xử lý các phản hồi.

* Gọi các phương thức bản địa :

Gọi các phương thức gốc bằng kênh phương thức bạn đã tạo. Vượt qua bất kỳ đối số cần thiết.

Trong mã gốc, hãy triển khai logic để xử lý các lệnh gọi phương thức này và trả về kết quả nếu cần.

* Xử lý Giao tiếp từ Native-to-Flutter :

Nếu bạn đang sử dụng các kênh sự kiện, hãy thiết lập trình nghe trong mã Dart để nhận các sự kiện từ lớp gốc.

* Kiểm tra và gỡ lỗi :

Kiểm tra kỹ lưỡng tích hợp gốc của bạn trên cả thiết bị Android và iOS hoặc trình mô phỏng.

Gỡ lỗi mọi vấn đề có thể phát sinh trong quá trình tích hợp.

Hãy nhớ rằng tích hợp gốc nên được xem xét khi không có plugin Flutter phù hợp hoặc khi bạn cần tận dụng các tính năng gốc cụ thể không thể truy cập trực tiếp thông qua hệ sinh thái plugin của Flutter. Tuy nhiên, việc sử dụng tích hợp gốc cần được quan tâm và chú ý nhiều hơn để duy trì trải nghiệm người dùng nhất quán trên cả hai nền tảng và đảm bảo xử lý lỗi thích hợp.

Trước khi triển khai tích hợp gốc, bạn nên khám phá các plugin và gói Flutter hiện có để xem liệu chúng có thể đáp ứng các yêu cầu của bạn mà không yêu cầu tích hợp mã gốc trực tiếp hay không.