Họ và tên: Đinh Ngọc Huyền

MSSV: 0850070021

Lớp: 08\_ĐH\_TMĐT

1. **Storage**

Trong Flutter, "storage" thường được dùng để đề cập đến cách lưu trữ và quản lý dữ liệu trên các thiết bị di động, chẳng hạn như điện thoại di động hoặc máy tính bảng. Cụ thể, nó liên quan đến việc lưu trữ và truy xuất các dữ liệu như tệp văn bản, hình ảnh, video, cơ sở dữ liệu hoặc các loại dữ liệu khác mà ứng dụng cần sử dụng.

* **Shared Preferences:** Đây là một cách đơn giản để lưu trữ và truy xuất dữ liệu như các cặp key-value. Shared Preferences hỗ trợ lưu trữ các loại dữ liệu như boolean, số nguyên, số thực và chuỗi. Dữ liệu được lưu trữ trong bộ nhớ của ứng dụng và có thể truy xuất mà không cần sử dụng gói plugin bên ngoài.
* **File System:** Flutter cung cấp giao diện để làm việc với hệ thống tập tin trên thiết bị. Bạn có thể lưu trữ dữ liệu trong tệp văn bản, hình ảnh, video hoặc các loại tệp khác. Bạn cũng có thể đọc và ghi dữ liệu vào tệp theo yêu cầu.
* **SQLite:** SQLite là một hệ thống quản lý cơ sở dữ liệu quan hệ nhỏ gọn và rất phổ biến trong ứng dụng di động. Flutter cung cấp gói plugin SQFlite để tương tác với cơ sở dữ liệu SQLite. Điều này cho phép bạn tạo, đọc, cập nhật và xóa các bản ghi trong cơ sở dữ liệu.
* **Firebase:** Firebase là một nền tảng đám mây của Google cung cấp nhiều dịch vụ, bao gồm cơ sở dữ liệu realtime (Firebase Realtime Database) và lưu trữ dựa trên văn bản (Firebase Cloud Firestore). Firebase giúp bạn dễ dàng lưu trữ và đồng bộ hóa dữ liệu trên các thiết bị và nền tảng khác nhau.
* **Hive:** Hive là một cơ sở dữ liệu NoSQL cực kỳ nhanh và nhẹ cho Flutter. Nó hỗ trợ các loại dữ liệu tùy chỉnh và sử dụng native code để đạt hiệu suất tốt hơn so với một số giải pháp lưu trữ khác.
* **Sử dụng External Storage:** Nếu bạn cần lưu trữ dữ liệu ngoài ứng dụng của mình, ví dụ như lưu trữ hình ảnh, video hoặc tệp dữ liệu lớn khác, bạn có thể lưu trữ chúng trong bộ nhớ ngoài của thiết bị.

1. **3rd Party**

"3rd Party" (Third Party) là thuật ngữ chỉ các công cụ, thư viện, API, gói hoặc phần mềm được phát triển bởi bên thứ ba, tức là không phải do nhà phát triển gốc hoặc người dùng phát triển ứng dụng tạo ra. Trong ngữ cảnh của Flutter, "3rd Party" thường đề cập đến các gói plugin, thư viện và công cụ mà không được tích hợp sẵn trong Flutter SDK, nhưng có thể được sử dụng để mở rộng chức năng của ứng dụng.

Các gói 3rd Party trong Flutter có thể cung cấp các tính năng và chức năng mà Flutter không tích hợp sẵn, giúp chúng ta nhanh chóng và dễ dàng xây dựng các tính năng phức tạp hơn trong ứng dụng của mình. Một số ví dụ về các gói 3rd Party phổ biến trong Flutter bao gồm:

* **http:** Gói này cung cấp các phương thức để thực hiện các yêu cầu HTTP như GET, POST, PUT, DELETE, và cung cấp tích hợp với các API từ các nguồn bên ngoài.
* **provider:** Gói provider cung cấp một giải pháp quản lý trạng thái đơn giản và hiệu quả cho ứng dụng Flutter thông qua mô hình kiến trúc "Provider".
* **firebase\_core và firebase\_auth:** Đây là các gói cung cấp khả năng tích hợp ứng dụng Flutter với dịch vụ Firebase của Google, cho phép xác thực người dùng, lưu trữ dữ liệu trong thời gian thực và thực hiện nhiều tính năng khác.
* **shared\_preferences:** Gói này cung cấp một giao diện dễ sử dụng để lưu trữ và truy xuất các thông tin nhỏ dưới dạng key-value trong ứng dụng.
* **flutter\_svg:** Gói này cho phép bạn hiển thị và tương tác với hình ảnh SVG (Scalable Vector Graphics) trong ứng dụng của mình.

Để sử dụng các gói 3rd Party, chúng ta cần thêm chúng vào tệp **pubspec.yaml** của ứng dụng Flutter và sau đó chạy lệnh **flutter pub get** để cập nhật các phụ thuộc và tải về các gói cần thiết. Sau đó, chúng ta có thể nhập các thư viện và sử dụng chúng trong code của mình.

Việc sử dụng các gói 3rd Party trong Flutter giúp chúng ta tiết kiệm thời gian và công sức phát triển, cho phép chúng ta tập trung vào việc xây dựng các tính năng chính của ứng dụng mà không cần phải xây dựng từ đầu tất cả các chức năng. Tuy nhiên, khi sử dụng các gói 3rd Party, hãy chắc chắn kiểm tra tính tin cậy của chúng, đảm bảo chúng đang được cập nhật và được hỗ trợ bởi cộng đồng đáng tin cậy.

1. **Libraries**

"Libraries" (thư viện) là các tập hợp mã được viết sẵn để cung cấp các chức năng và tính năng cụ thể. Thư viện giúp chúng ta tái sử dụng mã, giảm thiểu thời gian và công sức cần để xây dựng các tính năng phức tạp, và thường được phân phối dưới dạng các gói có thể được thêm vào ứng dụng của chúng ta.

Trong Flutter, có một số thư viện chính thức và các thư viện được cộng đồng phát triển có thể giúp chúng ta nhanh chóng xây dựng ứng dụng đa dạng. Dưới đây là một số ví dụ về thư viện quan trọng và phổ biến trong Flutter:

* **flutter/material:** Đây là thư viện cung cấp các widget và công cụ để xây dựng giao diện người dùng tuân theo chương trình thiết kế tài liệu vật lý (Material Design). Thư viện này chứa các widget như Text, Button, Icon, AppBar, và nhiều widget khác để xây dựng giao diện ứng dụng đẹp mắt và tương thích với nhiều thiết bị.
* **flutter/cupertino:** Thư viện này cung cấp các widget và công cụ để xây dựng giao diện người dùng theo chương trình thiết kế iOS (Cupertino Design). Điều này cho phép chúng ta tạo ra giao diện giống với ứng dụng iOS trên các thiết bị chạy Flutter.
* **dart:async:** Thư viện này cung cấp các lớp và phương thức để làm việc với các tác vụ bất đồng bộ và quản lý luồng trong Dart.
* **http:** Thư viện này cung cấp các phương thức để thực hiện các yêu cầu HTTP như GET, POST, PUT, DELETE để tương tác với các API từ các nguồn bên ngoài.
* **provider:** Thư viện này giúp quản lý trạng thái ứng dụng một cách hiệu quả thông qua mô hình "Provider". Đây là một cách để chia sẻ trạng thái và dữ liệu giữa các widget mà không cần sử dụng quá nhiều InheritedWidget.
* **firebase\_core và firebase\_auth:** Các thư viện này cho phép chúng ta tích hợp ứng dụng Flutter với dịch vụ Firebase của Google, giúp quản lý xác thực người dùng, lưu trữ dữ liệu trong thời gian thực và thực hiện các tính năng khác.
* **flutter\_bloc và rxdart:** Đây là các thư viện hỗ trợ quản lý trạng thái ứng dụng thông qua mô hình quản lý trạng thái (state management) như BLoC (Business Logic Component) và sử dụng các khái niệm liên quan đến Reactive Programming.
* **intl:** Thư viện này giúp chúng ta quản lý và định dạng văn bản đa ngôn ngữ trong ứng dụng của mình.

1. **Behavior**

"Behavior" (hành vi) thường đề cập đến cách mà các thành phần giao diện người dùng phản ứng và tương tác với các sự kiện và hành động từ người dùng. Behavior quyết định cách các widget và phần tử trong ứng dụng hoạt động, phản hồi và thay đổi trạng thái dựa trên tương tác của người dùng.

Dưới đây là một số khái niệm quan trọng liên quan đến Behavior trong Flutter:

* **Interactivity:** Đây là tính năng cho phép người dùng tương tác với các widget và phần tử trong ứng dụng. Interactivity được đạt được thông qua việc gắn kết các hàm xử lý sự kiện (event handlers) như onTap, onPressed, onChanged,... để phản hồi với hành động từ người dùng như nhấp chuột, chạm vào màn hình, nhập liệu, và nhiều hành động khác.
* **Animations:** Behavior có thể được sử dụng để thực hiện các hiệu ứng và hoạt hình khi người dùng tương tác với ứng dụng. Flutter cung cấp nhiều cách để tạo ra các hiệu ứng chuyển động và hoạt hình, từ các animations cơ bản như di chuyển, thu/phóng, đến các animations phức tạp hơn như hoạt hình tùy chỉnh.
* **State Management:** Behavior cũng liên quan đến cách quản lý trạng thái của ứng dụng. Khi người dùng tương tác với ứng dụng, trạng thái của các widget và thành phần có thể thay đổi. Sử dụng các mô hình quản lý trạng thái như BLoC, Provider, Redux,… giúp chúng ta quản lý và cập nhật trạng thái một cách hiệu quả và dễ dàng.
* **Gestures:** Flutter hỗ trợ nhiều loại gestures (cử chỉ) như tap, double tap, long press, swipe, pinch, và nhiều gestures khác. Bằng cách gắn kết các hàm xử lý gestures vào các widget, chúng ta có thể định nghĩa hành vi tương ứng với các cử chỉ từ người dùng.
* **Scrolling Behavior:** Scrolling behavior cho phép chúng ta tạo các danh sách cuộn và phần giao diện khác có thể cuộn. Flutter cung cấp các widget như ListView, GridView, SingleChildScrollView và nhiều widget khác để thực hiện scrolling behavior.
* **Focus Behavior:** Đối với các ứng dụng có khả năng tương tác với bàn phím (ví dụ: nhập liệu), Focus Behavior quy định cách widget có focus (được tập trung) và phản ứng với các sự kiện liên quan đến bàn phím.

Tóm lại, Behavior trong Flutter là cách mà các thành phần của ứng dụng phản hồi và tương tác với người dùng, đồng thời cũng định nghĩa cách chúng thay đổi trạng thái và hoạt động dựa trên các hành động khác nhau.

1. **Components**

"Components" (các thành phần) thường đề cập đến các phần tử cơ bản và đơn giản trong giao diện người dùng của ứng dụng. Các thành phần này thường là những phần tử xây dựng nên giao diện người dùng, và mỗi thành phần thường đảm nhiệm một chức năng hoặc hiển thị một phần cụ thể của thông tin.

Dưới đây là một số ví dụ về các thành phần phổ biến trong Flutter:

* **Text:** Thành phần Text cho phép hiển thị văn bản đơn giản hoặc định dạng trong giao diện người dùng.
* **Button:** Button là thành phần cho phép người dùng tương tác bằng cách nhấn vào nút. Flutter cung cấp nhiều loại nút như RaisedButton, FlatButton, IconButton, và nhiều loại khác.
* **Icon:** Thành phần Icon hiển thị biểu tượng hình ảnh đơn giản hoặc biểu tượng từ các bộ biểu tượng tích hợp.
* **Image:** Thành phần Image cho phép hiển thị hình ảnh trong giao diện người dùng. Chúng ta có thể sử dụng hình ảnh từ mạng hoặc từ tệp được lưu trữ trong ứng dụng.
* **Input Fields:** Các thành phần này cho phép người dùng nhập dữ liệu vào ứng dụng, bao gồm các loại như TextField (dòng văn bản), TextFormField (dòng văn bản với xác nhận), Checkbox (hộp kiểm), Radio (nút radio), và nhiều loại khác.
* **Card:** Thành phần Card cho phép chúng ta tạo ra các hộp chứa thông tin, thường được sử dụng để hiển thị dữ liệu có cấu trúc.
* **AppBar:** AppBar là một thành phần chứa tiêu đề và các tùy chọn điều hướng cho một màn hình trong ứng dụng. Nó thường xuất hiện ở đỉnh của màn hình.
* **ListView:** Thành phần ListView cho phép bạn tạo danh sách cuộn, hiển thị nhiều dòng thông tin hoặc các thành phần khác.
* **GridView:** GridView cho phép bạn xây dựng lưới các thành phần, thường được sử dụng để hiển thị danh sách hình ảnh hoặc thông tin.
* **Container:** Container là một thành phần chứa khác, cho phép chúng ta cung cấp các thuộc tính như màu sắc, kích thước và viền để tạo nên giao diện người dùng phức tạp.

Các thành phần này thường được kết hợp với nhau để tạo thành giao diện người dùng hoàn chỉnh. Flutter cung cấp một loạt các thành phần và widget để bạn có thể tạo ra giao diện ứng dụng đa dạng và hấp dẫn.

1. **Sate Managements**

Trong Flutter, "quản lý trạng thái" (state management) là cách bạn quản lý và cập nhật dữ liệu trong ứng dụng của bạn, đảm bảo rằng các thay đổi về dữ liệu được phản ánh trong giao diện người dùng. Flutter cung cấp các phương pháp khác nhau để xử lý quản lý trạng thái dựa trên sự phức tạp của ứng dụng và sở thích cá nhân. Dưới đây là một số kỹ thuật quản lý trạng thái phổ biến được sử dụng trong Flutter:

* **setState:** Đây là cách đơn giản nhất để quản lý trạng thái trong Flutter. Bạn có thể sử dụng setState trong một widget có trạng thái (stateful widget) để kích hoạt việc xây dựng lại cây widget khi trạng thái thay đổi. Nó hoạt động tốt cho các ứng dụng nhỏ và đơn giản trong đó trạng thái nằm cục bộ của một widget duy nhất.
* **InheritedWidget:** Đây là một widget cho phép bạn chia sẻ dữ liệu xuống cây widget mà không cần chuyển dữ liệu trực tiếp cho mỗi widget. Nó hữu ích để quản lý trạng thái cần được truy cập bởi nhiều widget, chẳng hạn như dữ liệu chủ đề hoặc trạng thái xác thực (authentication).
* **Provider:** Provider là một thư viện quản lý trạng thái phổ biến trong Flutter, sử dụng InheritedWidget bên trong để xử lý trạng thái. Nó cho phép bạn quản lý và chia sẻ trạng thái hiệu quả giữa nhiều widget mà không cần mã khởi tạo dư thừa. Provider được biết đến với sự đơn giản và dễ sử dụng.
* **Bloc (Business Logic Component)**: BLoC là một mô hình kiến trúc phân tách lớp trình chiếu (presentation layer) và logic kinh doanh (business logic). Nó sử dụng streams và sinks để xử lý trạng thái và sự kiện. Gói flutter\_bloc là một cài đặt phổ biến của mô hình BLoC.
* **Redux:** Redux là một kho chứa trạng thái có thể dự đoán được cho ứng dụng Dart/Flutter. Nó duy trì một kho chứa trạng thái toàn cục duy nhất và sử dụng luồng dữ liệu một chiều, giúp làm cho việc thay đổi trạng thái trở nên dễ dàng quản lý và dự đoán hơn.
* **MobX:** MobX là một thư viện quản lý trạng thái sử dụng các đối tượng có thể quan sát (observable) để theo dõi các thay đổi trong trạng thái. Nó nổi tiếng với tính đơn giản và thường được sử dụng trong các kịch bản lập trình phản ứng (reactive programming).
* **GetX:** GetX là một gói Flutter đầy đủ chức năng cung cấp quản lý trạng thái cùng với các tiện ích khác như điều hướng, tiêm phụ thuộc và nhiều hơn nữa. Mục tiêu của GetX là đơn giản, mạnh mẽ và hiệu quả.
* **Riverpod:** Riverpod là một sự lựa chọn thay thế cho Provider, tập trung vào cải thiện API và làm cho việc kiểm thử và quản lý trạng thái trở nên dễ dàng hơn.

Hãy nhớ rằng việc chọn kỹ thuật quản lý trạng thái phụ thuộc chủ yếu vào sự phức tạp của ứng dụng của bạn và sự quen thuộc với một phương pháp cụ thể. Các dự án khác nhau có thể có lợi từ các giải pháp quản lý trạng thái khác nhau, vì vậy hãy đánh giá các yêu cầu cụ thể của ứng dụng của bạn trước khi quyết định sử dụng phương pháp nào. Ngoài ra, cảnh quan quản lý trạng thái trong Flutter luôn luôn tiếp tục phát triển, nên nên theo dõi những cập nhật và phương pháp tốt nhất mới nhất.

1. **Quality Assurance**
   1. **Khái niệm:**

Quality Assurance (QA) trong Flutter là quá trình đảm bảo chất lượng của ứng dụng hoặc dự án Flutter. Nó bao gồm tất cả các hoạt động và quy trình được thực hiện để đảm bảo rằng ứng dụng của bạn hoạt động chính xác, ổn định và đáp ứng tốt với các yêu cầu và kỳ vọng của người dùng.

Dưới đây là một số phương pháp và hoạt động thường được thực hiện trong quá trình QA cho ứng dụng Flutter:

* **Kiểm thử tự động (Automated Testing):** Sử dụng các khung thử nghiệm như ‘flutter\_test’ để viết các bộ thử nghiệm tự động để kiểm tra tính đúng đắn và hiệu suất của mã Flutter.
* **Kiểm thử giao diện người dùng (UI Testing):** Sử dụng các khung thử nghiệm như ‘flutter\_driver’ để kiểm tra giao diện người dùng và tương tác với các phần của ứng dụng như người dùng thực tế.
* **Kiểm thử tích hợp (Integration Testing):** Kiểm tra tích hợp giữa các phần khác nhau của ứng dụng để đảm bảo chúng hoạt động cùng nhau một cách đúng đắn.
* **Kiểm thử đơn vị (Unit Testing**): Kiểm tra từng phần nhỏ của mã để đảm bảo chúng hoạt động như mong đợi.
* **Kiểm tra hiệu suất (Performance Testing):** Đảm bảo rằng ứng dụng mà ta đang thực thi hoạt động mượt mà và hiệu suất của nó có thể đáp ứng được yêu cầu của người dùng, ngay cả trong điều kiện tải cao.
* **Kiểm thử bảo mật (Security Testing):** Đảm bảo rằng ứng dụng mà ta đang phát triển được an toàn khỏi các lỗ hổng bảo mật và tấn công.
* **Kiểm tra tương thích (Compatibility Testing):** Kiểm tra ứng dụng trên các thiết bị và nền tảng khác nhau để đảm bảo tính tương thích.
* **Kiểm thử thực nghiệm (User Acceptance Testing - UAT):** Cho phép người dùng cuối kiểm tra ứng dụng và cung cấp phản hồi về trải nghiệm của họ.
* **Kiểm tra trạng thái ổn định (Stability Testing):** Kiểm tra tính ổn định của ứng dụng trong thời gian dài hoặc trong các tình huống không thường xuyên.
* **Kiểm tra phân tích lỗi và báo cáo (Defect Analysis and Reporting):** Theo dõi và báo cáo các lỗi và vấn đề trong quá trình kiểm thử để chúng có thể được sửa chữa.

Để đạt được chất lượng ứng dụng tốt nhất trong Flutter, quá trình QA cần được tích hợp vào toàn bộ quy trình phát triển.

* 1. **Công cụ và dịch vụ phổ biến của Quality Assurance:**

Trong Quality Assurance (QA) cho ứng dụng Flutter, có một số công cụ và dịch vụ quan trọng mà chúng ta có thể sử dụng để thực hiện kiểm tra và thử nghiệm ứng dụng của mình trước khi phát hành chính thức. Dưới đây là một số ví dụ về các công cụ và dịch vụ phổ biến:

* **Firebase Test Lab:** Firebase Test Lab cho phép chúng ta kiểm tra ứng dụng của mình trên nhiều thiết bị Android và iOS khác nhau, với nhiều phiên bản hệ điều hành và môi trường khác nhau. Ta có thể tự động chạy kiểm thử, bao gồm kiểm tra hiệu năng, sự tương thích, và giao diện người dùng trên các thiết bị ảo và thực tế.
* **Google Play Beta Tests:** Nếu nhà phát triển muốn triển khai phiên bản beta của ứng dụng trên Google Play Store, họ có thể sử dụng Google Play Beta Tests. Điều này cho phép họ mời người dùng tham gia kiểm thử và cung cấp phản hồi trước khi phát hành ứng dụng một cách rộng rãi.
* **TestFlight:** Đối với các thiết bị iOS, TestFlight là một công cụ của Apple cho phép bạn thử nghiệm ứng dụng trước khi phát hành. Bạn có thể mời người dùng tham gia vào các phiên bản beta và thu thập ý kiến ​​phản hồi.
* **App Center:** App Center của Microsoft cung cấp một nền tảng toàn diện cho việc kiểm thử và phân phối ứng dụng. Bạn có thể thực hiện kiểm tra tự động, kiểm thử sự tương thích trên các thiết bị thật và ảo, và triển khai phiên bản beta cho người dùng thử nghiệm.

**5.2.1. Firebase:**

Firebase là một thành phần không thể thiếu trong hầu hết cách mobile application. Flutter đã support hầu hết các tiện ích của firebase với khá nhiều Plugin trên Pub.dev.

* [**Firebase Auth**](https://pub.dev/packages/firebase_auth)**:**

Đây là plugin để sử dụng Firebase Authentication API. Để sử dụng được Firebase Auth cần phải config plugin [Google sign-in](https://pub.dev/packages/google_sign_in#-installing-tab-) Thêm Firebase Auth plugin vào file pubspec.yaml sau đó chạy lện $ flutter pub get để get plugin về.

dependencies:

firebase\_auth: ^0.16.1

Sử dụng Firebase Auth

Add the following imports to your Dart code:

import 'package:firebase\_auth/firebase\_auth.dart';

Initialize GoogleSignIn and FirebaseAuth:

final GoogleSignIn \_googleSignIn = GoogleSignIn();

final FirebaseAuth \_auth = FirebaseAuth.instance;

You can now use the Firebase \_auth to authenticate in your Dart code, e.g.

Future<FirebaseUser> \_handleSignIn() async {

final GoogleSignInAccount googleUser = await \_googleSignIn.signIn();

final GoogleSignInAuthentication googleAuth = await googleUser.authentication;

final AuthCredential credential = GoogleAuthProvider.getCredential(

accessToken: googleAuth.accessToken,

idToken: googleAuth.idToken,

);

final FirebaseUser user = (await \_auth.signInWithCredential(credential)).user;

print("signed in " + user.displayName);

return user;

}

Các phương thức authentication đang được hỗ trợ

* Google
* Email and Password
* Phone
* Anonymously
* GitHub
* Facebook
* Twitter
* [**Firebase Analytics**](https://pub.dev/packages/firebase_analytics)

Tương tự ta cũng thêm Firebase Analytics plugin vào file pubspec.yaml sau đó chạy lện $ flutter pub get để get plugin về.

dependencies:

firebase\_analytics: ^5.0.15

Sử dụng analytics thì cực kỳ đơn giản rồi phải không

Track PageRoute Transitions

FirebaseAnalytics analytics = FirebaseAnalytics();

MaterialApp(

home: MyAppHome(),

navigatorObservers: [

FirebaseAnalyticsObserver(analytics: analytics),

],

);

Cách sử dụng chi tiết

class MyApp extends StatelessWidget {

static FirebaseAnalytics analytics = FirebaseAnalytics();

static FirebaseAnalyticsObserver observer =

FirebaseAnalyticsObserver(analytics: analytics);

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Firebase Analytics Demo',

theme: ThemeData(

primarySwatch: Colors.blue,

),

navigatorObservers: <NavigatorObserver>[observer],

home: MyHomePage(

title: 'Firebase Analytics Demo',

analytics: analytics,

observer: observer,

),

);

}

}

class MyHomePage extends StatefulWidget {

MyHomePage({Key key, this.title, this.analytics, this.observer})

: super(key: key);

final String title;

final FirebaseAnalytics analytics;

final FirebaseAnalyticsObserver observer;

@override

\_MyHomePageState createState() => \_MyHomePageState(analytics, observer);

}

class \_MyHomePageState extends State<MyHomePage> {

\_MyHomePageState(this.analytics, this.observer);

final FirebaseAnalyticsObserver observer;

final FirebaseAnalytics analytics;

String \_message = '';

void setMessage(String message) {

setState(() {

\_message = message;

});

}

Future<void> \_sendAnalyticsEvent() async {

await analytics.logEvent(

name: 'test\_event',

parameters: <String, dynamic>{

'string': 'string',

'int': 42,

'long': 12345678910,

'double': 42.0,

'bool': true,

},

);

setMessage('logEvent succeeded');

}

Future<void> \_testSetUserId() async {

await analytics.setUserId('some-user');

setMessage('setUserId succeeded');

}

Future<void> \_testSetCurrentScreen() async {

await analytics.setCurrentScreen(

screenName: 'Analytics Demo',

screenClassOverride: 'AnalyticsDemo',

);

setMessage('setCurrentScreen succeeded');

}

Future<void> \_testSetAnalyticsCollectionEnabled() async {

await analytics.setAnalyticsCollectionEnabled(false);

await analytics.setAnalyticsCollectionEnabled(true);

setMessage('setAnalyticsCollectionEnabled succeeded');

}

Future<void> \_testSetSessionTimeoutDuration() async {

await analytics.android?.setSessionTimeoutDuration(2000000);

setMessage('setSessionTimeoutDuration succeeded');

}

Future<void> \_testSetUserProperty() async {

await analytics.setUserProperty(name: 'regular', value: 'indeed');

setMessage('setUserProperty succeeded');

}

Future<void> \_testAllEventTypes() async {

await analytics.logAddPaymentInfo();

await analytics.logAddToCart(

currency: 'USD',

value: 123.0,

itemId: 'test item id',

itemName: 'test item name',

itemCategory: 'test item category',

quantity: 5,

price: 24.0,

origin: 'test origin',

itemLocationId: 'test location id',

destination: 'test destination',

startDate: '2015-09-14',

endDate: '2015-09-17',

);

await analytics.logAddToWishlist(

itemId: 'test item id',

itemName: 'test item name',

itemCategory: 'test item category',

quantity: 5,

price: 24.0,

value: 123.0,

currency: 'USD',

itemLocationId: 'test location id',

);

await analytics.logAppOpen();

await analytics.logBeginCheckout(

value: 123.0,

currency: 'USD',

transactionId: 'test tx id',

numberOfNights: 2,

numberOfRooms: 3,

numberOfPassengers: 4,

origin: 'test origin',

destination: 'test destination',

startDate: '2015-09-14',

endDate: '2015-09-17',

travelClass: 'test travel class',

);

await analytics.logCampaignDetails(

source: 'test source',

medium: 'test medium',

campaign: 'test campaign',

term: 'test term',

content: 'test content',

aclid: 'test aclid',

cp1: 'test cp1',

);

await analytics.logEarnVirtualCurrency(

virtualCurrencyName: 'bitcoin',

value: 345.66,

);

await analytics.logEcommercePurchase(

currency: 'USD',

value: 432.45,

transactionId: 'test tx id',

tax: 3.45,

shipping: 5.67,

coupon: 'test coupon',

location: 'test location',

numberOfNights: 3,

numberOfRooms: 4,

numberOfPassengers: 5,

origin: 'test origin',

destination: 'test destination',

startDate: '2015-09-13',

endDate: '2015-09-14',

travelClass: 'test travel class',

);

await analytics.logGenerateLead(

currency: 'USD',

value: 123.45,

);

await analytics.logJoinGroup(

groupId: 'test group id',

);

await analytics.logLevelUp(

level: 5,

character: 'witch doctor',

);

await analytics.logLogin();

await analytics.logPostScore(

score: 1000000,

level: 70,

character: 'tiefling cleric',

);

await analytics.logPresentOffer(

itemId: 'test item id',

itemName: 'test item name',

itemCategory: 'test item category',

quantity: 6,

price: 3.45,

value: 67.8,

currency: 'USD',

itemLocationId: 'test item location id',

);

await analytics.logPurchaseRefund(

currency: 'USD',

value: 45.67,

transactionId: 'test tx id',

);

await analytics.logSearch(

searchTerm: 'hotel',

numberOfNights: 2,

numberOfRooms: 1,

numberOfPassengers: 3,

origin: 'test origin',

destination: 'test destination',

startDate: '2015-09-14',

endDate: '2015-09-16',

travelClass: 'test travel class',

);

await analytics.logSelectContent(

contentType: 'test content type',

itemId: 'test item id',

);

await analytics.logShare(

contentType: 'test content type',

itemId: 'test item id',

method: 'facebook');

await analytics.logSignUp(

signUpMethod: 'test sign up method',

);

await analytics.logSpendVirtualCurrency(

itemName: 'test item name',

virtualCurrencyName: 'bitcoin',

value: 34,

);

await analytics.logTutorialBegin();

await analytics.logTutorialComplete();

await analytics.logUnlockAchievement(id: 'all Firebase API covered');

await analytics.logViewItem(

itemId: 'test item id',

itemName: 'test item name',

itemCategory: 'test item category',

itemLocationId: 'test item location id',

price: 3.45,

quantity: 6,

currency: 'USD',

value: 67.8,

flightNumber: 'test flight number',

numberOfPassengers: 3,

numberOfRooms: 1,

numberOfNights: 2,

origin: 'test origin',

destination: 'test destination',

startDate: '2015-09-14',

endDate: '2015-09-15',

searchTerm: 'test search term',

travelClass: 'test travel class',

);

await analytics.logViewItemList(

itemCategory: 'test item category',

);

await analytics.logViewSearchResults(

searchTerm: 'test search term',

);

setMessage('All standard events logged successfully');

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text(widget.title),

),

body: Column(

children: <Widget>[

MaterialButton(

child: const Text('Test logEvent'),

onPressed: \_sendAnalyticsEvent,

),

MaterialButton(

child: const Text('Test standard event types'),

onPressed: \_testAllEventTypes,

),

MaterialButton(

child: const Text('Test setUserId'),

onPressed: \_testSetUserId,

),

MaterialButton(

child: const Text('Test setCurrentScreen'),

onPressed: \_testSetCurrentScreen,

),

MaterialButton(

child: const Text('Test setAnalyticsCollectionEnabled'),

onPressed: \_testSetAnalyticsCollectionEnabled,

),

MaterialButton(

child: const Text('Test setSessionTimeoutDuration'),

onPressed: \_testSetSessionTimeoutDuration,

),

MaterialButton(

child: const Text('Test setUserProperty'),

onPressed: \_testSetUserProperty,

),

Text(\_message,

style: const TextStyle(color: Color.fromARGB(255, 0, 155, 0))),

],

),

floatingActionButton: FloatingActionButton(

child: const Icon(Icons.tab),

onPressed: () {

Navigator.of(context).push(MaterialPageRoute<TabsPage>(

settings: const RouteSettings(name: TabsPage.routeName),

builder: (BuildContext context) {

return TabsPage(observer);

}));

}),

);

}

}

* [**Cloud Firestore**](https://pub.dev/packages/cloud_firestore)

Tương tự ta cũng thêm Firebase Analytics plugin vào file pubspec.yaml sau đó chạy lện $ flutter pub get để get plugin về.

dependencies:

cloud\_firestore: ^0.13.7

Sử dụng

import 'package:cloud\_firestore/cloud\_firestore.dart';

Adding a new DocumentReference:

Firestore.instance.collection('books').document()

.setData({ 'title': 'title', 'author': 'author' });

Binding a CollectionReference to a ListView:

class BookList extends StatelessWidget {

@override

Widget build(BuildContext context) {

return StreamBuilder<QuerySnapshot>(

stream: Firestore.instance.collection('books').snapshots(),

builder: (BuildContext context, AsyncSnapshot<QuerySnapshot> snapshot) {

if (snapshot.hasError)

return new Text('Error: ${snapshot.error}');

switch (snapshot.connectionState) {

case ConnectionState.waiting: return new Text('Loading...');

default:

return new ListView(

children: snapshot.data.documents.map((DocumentSnapshot document) {

return new ListTile(

title: new Text(document['title']),

subtitle: new Text(document['author']),

);

}).toList(),

);

}

},

);

}

}

Performing a query:

Firestore.instance

.collection('talks')

.where("topic", isEqualTo: "flutter")

.snapshots()

.listen((data) =>

data.documents.forEach((doc) => print(doc["title"])));

Get a specific document:

Firestore.instance

.collection('talks')

.document('document-name')

.get()

.then((DocumentSnapshot ds) {

// use ds as a snapshot

});

Running a transaction:

final DocumentReference postRef = Firestore.instance.document('posts/123');

Firestore.instance.runTransaction((Transaction tx) async {

DocumentSnapshot postSnapshot = await tx.get(postRef);

if (postSnapshot.exists) {

await tx.update(postRef, <String, dynamic>{'likesCount': postSnapshot.data['likesCount'] + 1});

}

});

**5.2.2. App Center:**

App Center là một dịch vụ phát triển ứng dụng do Microsoft cung cấp, được sử dụng để tạo, kiểm tra và triển khai ứng dụng di động trên nhiều nền tảng, bao gồm cả iOS, Android và Windows. Đặc biệt, trong ngữ cảnh của Flutter, App Center là một tùy chọn phổ biến để quản lý quy trình phát triển ứng dụng di động.

* **App Center hoạt động bằng cách cung cấp các tính năng quan trọng sau:**
* **Xây dựng (Build):** App Center cho phép tạo ra các phiên bản ứng dụng với việc tùy chỉnh quy trình xây dựng. Chúng ta có thể tích hợp mã nguồn từ kho lưu trữ Git của mình và thiết lập các quy trình xây dựng tự động cho các phiên bản khác nhau.
* **Kiểm tra (Test):** App Center cung cấp khả năng kiểm tra tự động và kiểm tra thủ công cho ứng dụng trên nhiều thiết bị và nền tảng khác nhau. Điều này giúpđảm bảo rằng ứng dụng hoạt động một cách đáng tin cậy trên nhiều môi trường khác nhau.
* **Triển khai (Deploy):** Sau khi kiểm tra và xây dựng, App Center cho phép triển khai ứng dụng lên các cửa hàng ứng dụng như App Store và Google Play. Ngoài ra, ta có thể quản lý việc phát hành và cập nhật ứng dụng một cách dễ dàng thông qua giao diện quản lý của App Center.
* **Giám sát (Monitor):** App Center cung cấp tính năng giám sát hiệu suất và sự cố của ứng dụng trên thời gian thực. Có thể theo dõi thông tin về lượt tải về, tình trạng hoạt động và các lỗi xảy ra để có cái nhìn tổng quan về hiệu suất của ứng dụng.

**5.2.3. TestFlight (iOS)**

TestFlight là giải pháp của chính Apple để phân phối các bản thử nghiệm của iOS và các ứng dụng trên nền tảng khác của Apple. Apple mua lại TestFlight vào năm 2014 để hỗ trợ nhu cầu phân phối bản thử nghiệm của các ứng dụng trên nền tảng Apple. Khi các ứng dụng trên nền tảng của Apple được tải lên App Store Connect, chúng ta có thể dễ dàng lấy nó các ứng dụng đó bằng cách sử dụng Ứng dụng TestFlight. Và để có được bản app beta trên TestFlight, chúng ta phải có tài khoản Nhà phát triển của Apple và phải là thành viên của nhóm phát triển. Khi bản build được gửi để xem xét, họ có thể thêm người kiểm tra bên ngoài hoặc chia sẻ liên kết công khai cho bất kỳ ai để có bản build mới nhất đó. App Store Connect có giới hạn số lượng người kiểm tra nội bộ nhưng bạn có thể gửi được tới 10.000 người kiểm tra bên ngoài. Tuy nhiên, ứng dụng của chúng ta cần phải được gửi tới quy trình đánh giá của Apple rồi mới có thể gửi cho những người thử nghiệm bên ngoài. TestFlight cho phép chúng ta tạo các nhóm khác nhau và gán các build khác nhau cho các nhóm khác nhau. Nó hỗ trợ thử nghiệm 100 ứng dụng khác nhau cùng một lúc và các bản build có thể được gửi đến người dùng thông qua liên kết công khai.

**5.2.4. Google play beta tests:**

Google Play Beta Tests là một tính năng cung cấp bởi Google Play Console, cho phép nhà phát triển kiểm thử và triển khai các phiên bản beta của ứng dụng Android trước khi chúng được phát hành chính thức trên Google Play Store. Điều này giúp nhà phát triển kiểm tra ứng dụng trong môi trường thực tế và thu thập phản hồi từ người dùng trước khi đưa ứng dụng cho người dùng cuối.

1. **Version control**

* **Git**

Một hệ thống kiểm soát phiên bản phân tán. Mục đích là để giúp các nhóm theo dõi các thay đổi và cải thiện sự hợp tác giữa các nhà phát triển. Mục đích sáng tạo của Git là tạo điều kiện hợp tác và cung cấp các công cụ cho các cộng đồng nguồn mở. Các tệp có xu hướng dày hơn theo thời gian khi các phiên bản được hợp nhất và xác minh.

**Các tính năng của Git cần thiết cho dự án mà bạn cần biết**

* **Lệnh git init**

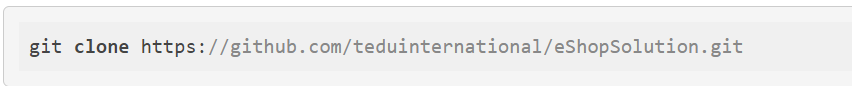
Lệnh này dùng để khởi tạo một Repository trên máy local, tuy nhiên nó chưa được push lên server mà chúng ta vẫn có thể thao tác với tất cả các lệnh như bình thường và nó hoạt động như một repository thực thụ trừ share code cho các thành viên khác.



Để init một repository thì chỉ cần vào một folder trống mở Command Prompt rồi gõ lệnh git init nếu thành công thì thư mục đó sẽ thành một Git repository các bạn có thể show hidden file and folders sẽ thấy thư mục tên là. git ẩn.

* **Lệnh git clone**

Cách 2 để tạo một git repository là lệnh clone, lệnh này sẽ lấy code từ một Git service trên mạng được public và download về máy sau đó tự tạo luôn một Git repository local kết nối với git remote luôn. Ví dụ chúng ta cần download code của repository có tên eShopSolution về máy chúng ta cần chạy:



* **Lệnh git pull**

Lệnh git pull dùng để cập nhật code mới từ trên remote server và nó cũng tự động merge code từ nhánh tương ứng trên server về, ví dụ chúng ta có nhánh develop trên server nó sẽ là tên của origin remote (origin/develop) nếu có ai commit và push code lên nhánh đó khi pull về nó sẽ tự merge code của họ ở nhánh đó vào nhánh develop trên máy chúng ta. Nhánh local trên máy và nhánh origin/develop liên kết với nhau qua cơ chế set upstream. Cú pháp là git pull origin hoặc git pull đều được. Nếu git pull thì mặc định nó sẽ lấy từ nhánh hiện tại còn chỉ ra nhánh sẽ pull từ nhánh đó. Thông thường chỉ cần switch sang nhánh cần pull bằng lệnh git checkout rồi gọi git pull. Trong trường hợp chúng ta có 1 remote repository thì chỉ cần git pull là đủ. Trừ khi có 2 repo thì chỉ ra tên remote repository

A white rectangular object with dots

Description automatically generated

* **Lệnh git add**

Lệnh git add nhằm add các file mới được tạo vào git tracking (stage), một file mới tạo ra sẽ ở chế độ unstage, khi chúng ta add vào stage thì mới commit được vào Git repository.

Để add toàn bộ sự thay đổi chúng ta dùng

A white rectangular object with a black border

Description automatically generated

Nếu không sẽ cần chỉ ra tên của file cần add thay vì dấu \*

* **Lệnh git fetch**

Git fetch cũng cập nhật thay đổi từ server nhưng không tự merge như git pull mà chỉ đóng vai trò get latest xem có commit nào mới từ server hay không nó tương đương F5 refresh vậy.

A white rectangular object with a black border

Description automatically generated

* **Lệnh git remote**

Lệnh này để thao tác với origin remote của chúng ta, ví dụ chúng ta có repository local nhưng khi muốn đưa code lên server Github chúng ta cần tạo một Repository trên Github và tạo origin trên đó sau đó mới đẩy code lên. Để tạo origin thì phải dùng lệnh này.

Lệnh dưới dùng để thêm một remote repository vào local repository với tên và url.

A white rectangle with black text

Description automatically generated

Lấy danh sách remote origin

A white rectangular object with a black border

Description automatically generated

* **Lệnh git reset**

Lệnh này để xóa sự thay đổi của chúng ta về trạng thái ban đầu chưa sửa hoặc về bất cứ commit nào trong quá khứ

Reset về HEAD tức là về commit cuối cùng của nhánh đó:

A screenshot of a computer screen

Description automatically generated

* **Lệnh git checkout**

Lệnh này chuyển nhánh trong Git giữa nhánh nọ nhánh kia

A computer screen shot of a computer screen

Description automatically generated

* **Lệnh git commit**

Lệnh này commit code changes vào Git với một commit message. Ví dụ: chúng ta commit sự thay đổi với message là "First commit":

A white rectangular sign with red text

Description automatically generated

* **GitHub**

GitHub là nền tảng kiểm soát phiên bản dựa trên Git lớn nhất. Giờ đây, nó tổ chức hơn 38 triệu dự án và được sử dụng phổ biến nhất bởi các cộng đồng nguồn mở.

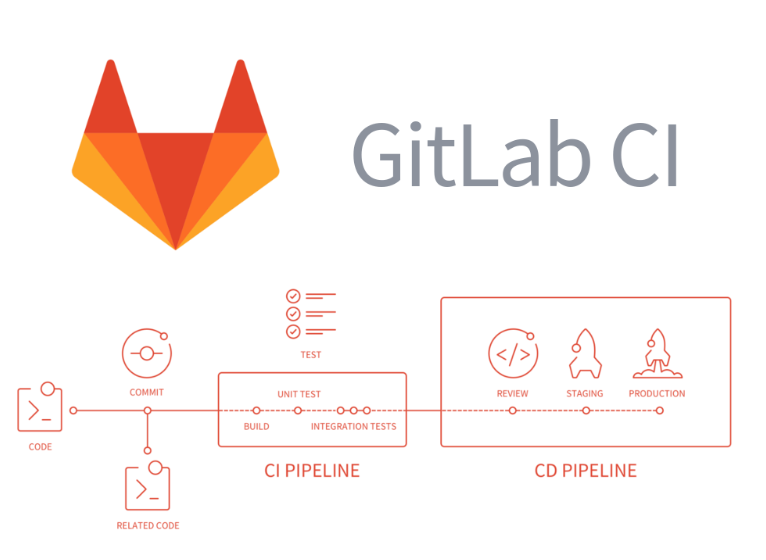
GitHub có Giao diện người dùng đơn giản giúp các nhà phát triển nhanh chóng bắt tay vào làm việc với các thuật toán Git. Một đặc điểm xác định khác là tốc độ của nó - dịch vụ đẩy và kéo các yêu cầu và hợp nhất các phiên bản một cách nhanh chóng.

Phiên bản miễn phí của GitHub cho phép các nhà phát triển làm việc với các kho công khai và riêng tư, đóng góp cho chúng và cộng tác. Chi phí phụ thuộc vào số lượng người dùng.

* **GitLab**

GitLab được thành lập vào năm 2011 để thay thế cho GitHub và BitBucket. Điểm bán hàng chính của nó là chức năng mở rộng, được đóng gói gọn gàng trong một giao diện người dùng tuyệt vời bất kể tính linh hoạt của nó.

Gần đây, các nhóm đã tích cực chuyển sang GitLab từ các dịch vụ khác. Đặc biệt là do sự hỗ trợ liền mạch của GitLab đối với [đường ống CI và Docker](https://jelvix.com/blog/best-ci-cd-tools-comparison) (và việc sử dụng chúng đã phát triển rất nhiều trong thời gian gần đây).



* **Bitbucket**

Bitbucket là một dịch vụ trên web được phát triển bởi Atlassian, giúp lưu trữ mã nguồn và quản lý kho lưu trữ Git. Kho lưu trữ Git được viết bằng ngôn ngữ lập trình Python và được xây dựng bằng web framework Django .

A screenshot of a web page

Description automatically generated

Bitbucket cung cấp sự linh hoạt về mặt hỗ trợ VCS đây được xem là một trong những điểm thu hút chính mà Bitbucket mang lại. Ngoài ra, Bitbucket còn cung cấp các kho lưu trữ private code không giới hạn cho Mercurial và Git.

* **Các tính năng của Bitbucket**
* Tích hợp trực tiếp với Jira, tre, Crucible và Jenkins.
* Khả năng nhập REPOS từ Git, Codeplex, GoogleCode và SVN.
* Hỗ trợ xác thực bên ngoài cho GitHub, Google, Facebook và Twitter
* Tích hợp sâu với Trello
* A Client (Sourcetree) và ứng dụng Android (Bitbeaker)
* **Tùy chọn lưu trữ Bitbucket**
* Bitbucket Cloud

Bitbucket Cloud được lưu trữ trên máy chủ của Atlassian và được truy cập thông qua URL. Bitbucket Cloud có một công cụ CI/CD tích hợp độc quyền, Pipelines, cho phép bạn xây dựng, kiểm tra và triển khai trực tiếp từ bên trong Bitbucket.

* Bitbucket Server

Bitbucket Server được lưu trữ tại chỗ trong môi trường của bạn. Máy chủ Bitbucket có tích hợp mạnh mẽ với Bamboo, công cụ CI/CD cho phép bạn tự động hóa hoàn toàn quy trình của mình.

* Bitbucket Data Center

Bitbucket Data Center như một phiên bản duy nhất của máy chủ Bitbucket cho người dùng, được lưu trữ trên một số server, hoạt động trên môi trường của bạn.

1. **Firebase**

Vào những năm 2011, trước khi Firebase ra đời. Tiền thân của nền tảng Firebase chính là Envolve. Envolve đến với người dùng dưới mô hình startup, họ cung cấp cho người dùng những API để dễ dàng tích hợp tính năng chat vào trang web. Thế nhưng, người dùng lại sử dụng Envolve để truyền dữ liệu ứng dụng chứ không đơn thuần là nhắn tin trò chuyện. Chính điều này đã khiến các nhà phát triển Envolve quyết định tách riêng chat system và the real-time architecture.

Và năm 2012, Firebase ra đời dưới dạng Backend-as-a-Service với chức năng thời gian thực. Sau khi được Google mua lại vào năm 2014 thì Firebase nhanh chóng phát triển thành nền tảng phát triển ứng dụng đa năng của di động và website như ngày nay. Nền tảng này là sự kết hợp giữa cloud với hệ thống máy chủ của Google để tập trung chính cho 2 đối tượng là:

* **Develop & test your app**: phát triển và thử nghiệm các ứng dụng được thiết kế.
* **Grow & engage your audience**: phân tích dữ liệu và tối ưu hóa trải nghiệm với người dùng.

**Những service nổi bật**

Firebase là một nền tảng đa năng, nó cung cấp rất nhiều dịch vụ khác nhau cho người dùng.

* **Real-time Database**

Dịch vụ Realtime database cho phép người dùng lưu trữ và đồng bộ dữ liệu theo thời gian thực. Dịch vụ này được lưu trữ trực tiếp trên iCloud. Trong trường hợp thiết bị của chúng ta ngoại tuyến thì chúng sẽ sử dụng tới bộ nhớ của thiết bị và tự động đồng bộ lên server khi thiết bị online. Do đó ta hoàn toàn có thể yên tâm về độ tương tác.



* **Authentication**

Dịch vụ Authentication cung cấp cho ứng dụng của ta một số phương pháp xác thực thông qua email, mật khẩu, số điện thoải, tài khoản Google, tài khoản Facebook… Với tính năng này, người dùng sẽ dễ dàng xây dựng login mà không cần sử dụng dữ liệu đăng ký riêng vô cùng tiện lợi và nhanh chóng.



* **Firebase cloud messaging**

Firebase được xây dựng với tính năng nguyên thủy là trò chuyện. Bởi thế Firebase cloud messaging (FCM) là tính năng cơ bản nhất của nền tảng này, nó cho phép người dùng xây dựng ứng dụng chat và đẩy thông báo tới nhiều thiết bị khác nhau như web, Android, iOS… Điểm nổi bật của dịch vụ này là hầu như không có bất kỳ mã hóa nào liên quan. FCM được tích hợp hoàn toàn với Firebase Analytics, mang đến cho ta sự tương tác chi tiết (detailed engagement) và theo dõi chuyển đổi (conversion tracking) trong quá trình sử dụng.

Hơn thế, dịch vụ FCM còn giúp ta tùy chọn thời điểm gửi tin là ngay lập tức hoặc vào thời điểm tương lai theo múi giờ địa phương của người dùng.



* **Firebase database query**

Một trong những dịch vụ nổi bật của Firebase chính là nó có thể giúp ta đơn giản hóa quá trình lấy dữ liệu thay vì phải thông qua các câu lệnh SQL phức tạp.

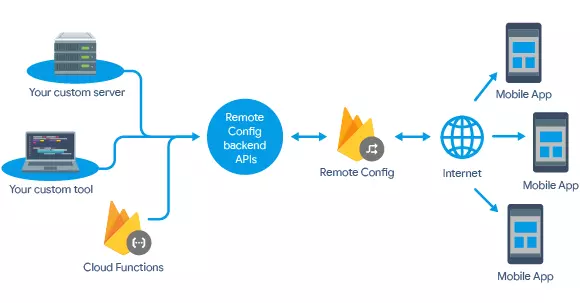


* **Remote Config**

Remote Config giúp ta làm 2 nhiệm vụ cơ bản là:

* Cập nhật các ứng dụng một cách nhanh chóng và dễ dàng mà không cần xuất bản bản dựng mới lên app/play store.
* Ta có thể dễ dàng cài đặt phân đoạn hoạt động trong ứng dụng dựa trên yêu cầu thiết bị hoặc người dùng đang sử dụng nó.

Để làm được điều đó, Firebase sẽ cài đặt các thông số bên máy chủ giúp người dùng cập nhật các dữ liệu ngay lập tức dù là thay đổi bố cục, bảng màu hay một phần cụ thể nào đó trong ứng dụng. Điều này hết sức hữu dụng bởi các ứng dụng thông thường sẽ bị mất hoàn toàn cài đặt nếu người dùng gỡ ứng dụng đi. Nếu muốn cài đặt trên nhiều ứng dụng khác nhau thì họ sẽ phải tự cấu hình thiết lập bằng tay trên từng thiết bị sao cho giống nhau. Quá trình này khá phức tạp và mất nhiều thời gian.



**8.** [**Native integration**](https://cyclr.com/native-integration)

**8.1.** [**Native integration**](https://cyclr.com/native-integration) **là gì?**

[**Native integration**](https://cyclr.com/native-integration)  là khả năng xây dựng quy trình công việc có nhiều ứng dụng SaaS như Salesforce và Quickbooks. Chúng cung cấp phương tiện trực tiếp để tích hợp và giao tiếp với nhau thông qua API. Sau đó, dữ liệu được ánh xạ có thể luân chuyển giữa các ứng dụng SaaS trong thời gian thực và luôn sẵn sàng cho các nhóm thích hợp.

API rất quan trọng đối với Native integration vì chúng là cấu trúc mã đảm bảo sự tương tác giữa các ứng dụng.

Ví dụ: nếu tích hợp [Salesforce](https://cyclr.com/integrate/salesforce) và Quickbooks, có thể chia sẻ thông tin và nhiều loại hồ sơ, chẳng hạn như tài khoản của khách hàng và thông tin đăng ký từ [Quickbooks](https://cyclr.com/integrate/quickbooks) trong Salesforce.

Sự tương tác này xảy ra giữa các máy hoặc hệ thống qua mạng và trực tiếp trong SaaS của người dùng.

**Native integration** có thể được phát triển nội bộ hoặc thông qua [iPaaS nhúng](https://cyclr.com/) cung cấp các công cụ tích hợp để người dùng xây dựng và triển khai tích hợp trong ứng dụng.

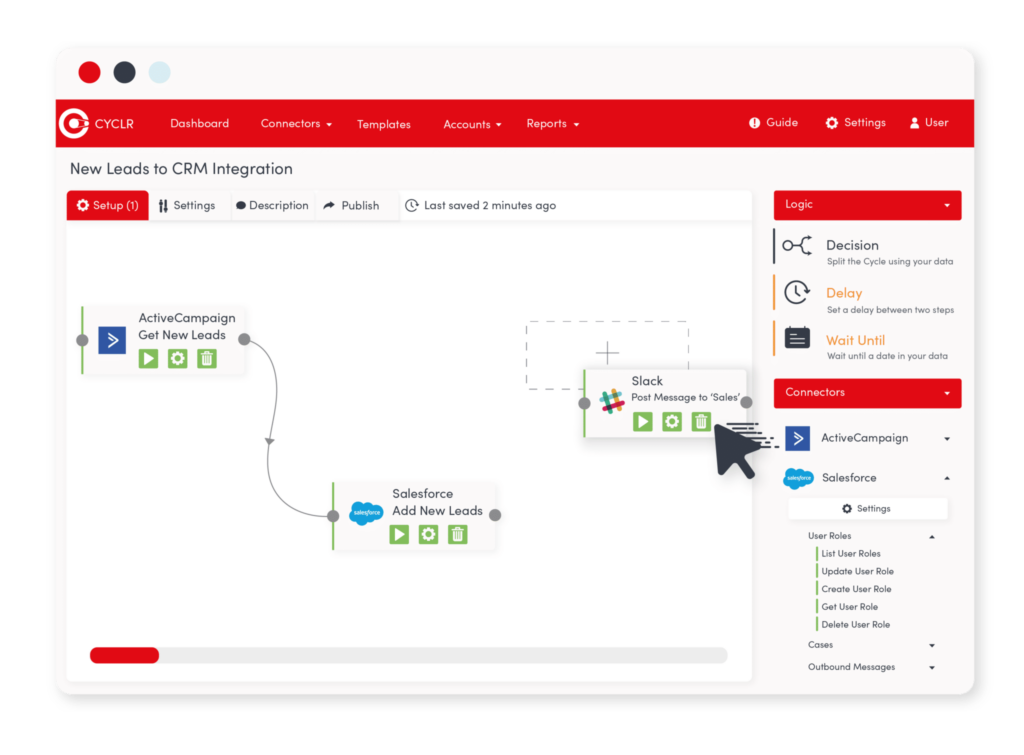
Ví dụ, trong quá trình phát triển ứng dụng di động, tích hợp "native" có thể bao gồm:

Sử Dụng Các Phần Tử Giao Diện Người Dùng Native: Tích hợp các phần tử giao diện người dùng cụ thể của nền tảng, như nút hoặc thanh điều hướng, để tạo ra giao diện người dùng thống nhất và quen thuộc với mẫu thiết kế của hệ điều hành.

Truy Cập Phần Cứng Thiết Bị: Sử dụng các API cụ thể của nền tảng để truy cập các chức năng phần cứng như GPS, máy ảnh, micro, hoặc cảm biến để tăng cường khả năng của ứng dụng .

Tích Hợp Các Dịch Vụ Native: Kết hợp với các dịch vụ cụ thể của nền tảng như thông báo đẩy, mua hàng trong ứng dụng, hoặc chia sẻ qua mạng xã hội, tận dụng các dịch vụ có sẵn của từng nền tảng.

Tối Ưu Hiệu Năng: Sử dụng tối ưu hóa và các phương pháp tốt nhất cụ thể của nền tảng để đảm bảo ứng dụng hoạt động mượt mà trên nền tảng mục tiêu.



**8.2.Ưu,nhược điểm của Native integration**

Ưu điểm:

Hầu hết các phần mềm đều có Native integration cho các dịch vụ phổ biến: Mặc dù cấu trúc công nghệ có thể khác nhau từ doanh nghiệp này sang doanh nghiệp khác, nhưng có một số ứng dụng và dịch vụ được yêu thích như WordPress, Shopify và Facebook. Nền tảng kinh doanh Native integration có thể có tích hợp sẵn để chia sẻ dữ liệu với những dịch vụ này.

Việc kích hoạt tích hợp Native integration thường rất đơn giản: Khi tích hợp đã được tích hợp sẵn trong phần mềm, thì chỉ cần bật cài đặt để thiết lập kết nối.

Thường thì không có chi phí bổ sung cho tích hợp Native integration: Sau khi đã mua hoặc cấp phép phần mềm ban đầu,thì sẽ không phải trả thêm phí cho tích hợp đó.

Nhược điểm:

Hạn chế trong việc tích hợp sẵn trong phần mềm: Nếu một trong các dịch vụ đang sử dụng là mới hoặc không được sử dụng rộng rãi, sẽ không thể tích hợp Native integration với dịch vụ đó.

Tích hợp Native integration tùy chỉnh có thể tốn kém: Nếu đang sử dụng phần mềm độc quyền và cần ứng dụng chia sẻ dữ liệu với một dịch vụ mới, thì phải mời một nhà phát triển tạo code để thực hiện tích hợp đó.